C++ Library for Flicker Intensity Removal in Old Films

Final Report
of Diploma Internship
(01.10.2009 - 31.08.2010)

Marek Jagielski

September 17, 2010

Supervisors:
Koehler Ralf - Technicolor
Cazuguel Guy - Telecom Bretagne

Telecom Bretagne

Technopôle Brest-Iroise CS 83818

29238 Brest Cedex 3 - France

Technicolor

Deutsche Thomson OHG

Karl-Wiechert-Allee 74

30625 Hannover - Germany

Preface

This document is a summary of the almost annual internship in the company Deutsche Thomson OHG, one of the research center of Technicolor SA. Working in a commercial environment is a part of the curriculum at the University Telecom Bretagne. The time spent in the company can be divided into two periods. The first part of the internship from 1 October to 31 January 2010 took place mostly at Villingen branch in Germany. During this period I was involved in their previous work for restoration of old digitalized movies. Having no prior professional experience with image processing, I received a very good preparation for an independent work on algorithms and its software implementations. The period from 1 February 2010 to 31 August 2010 was devoted to a self-reliant project that overlaped a comprehensive treatment of intensity flicker in old films. The work performed ranged from information research to a finished product that is a C++ library for flicker removal.

The Final Report in its content is a technical documentation. However, there are several recipients of the document. The expectations of each forced containing certain parts or adoption of presentation form. They are:

Deutsche Thomson OHG - The employer. Practical dimension is the most important feature. Basing on the document, a reader should understand the scope of work done. The project itself was carried out on several levels - most generally there was a design and implementation part. If the work was developed or its components were used in other projects, the document should easily introduce the reader into all subworks. The document should delve into the details just to make able a further independent study.

Telecom Bretagne - The requirements for the document are formulated in two internal documents of the University: "GUIDE DES STAGES DE FIN D'ETUDES ou validation du SFE dans le cadre de la JI" and "Guide de rédaction de rapport de projet". Important additional demand is to demonstrate my integration with work environment and my ability to project management.

Gdansk University of Technology - As a student of double diploma agreement, my work should also be a basis for obtaining a master degree in Gdansk University of Technology. Briefly it can be said that the Master Thesis in Poland should have a scientific nature. Above all, it should be divided into theoretical and practical part.

The report is accompanied by a DVD. Description of its contents can be found in the end of the report (Appendix C).

Abstract

This report summarizes a project whose goal was to provide a C++ library to remove the flicker intensity from old films that are already in digital format. The document was split into five parts that reflect the performed work:

1. Project
2. Understanding the problem
3. Design
4. Implementation
5. Recapitulation

In the first part the project context is presented as well as its objectives and expected results. There is also a methodology of work that was developed during its lifetime. The methodology was adapted to the nature of the project and had a high agility.

The second part presents a gathered knowledge of the issues. In this paper, there are short summaries of 15 articles and one PhD Thesis. In view of the fact that the flicker occurs together with other elements of the movie, a few "use cases" were defined. They are typical situations that the future algorithms will have to cope with.

The third part includes the algorithm design that base on one articles from the second part. Drawing conclusions was made by theoretical considerations and performed tests. The tests were performed using own implementations (part four). There are also several improvements suggested what makes the work innovative.

The fourth part describes the implementation of C++ library. A software design is based on the best practices of software engineering. The methodology is discussed separately. This is purely technical part. But it became crucial for the success of the project. It's no possible to make conclusions on the quality of algorithms without having a reliable implementation. Also the implementation was the biggest time consumer.

Ce rapport est un résumé du projet dont le but était de fournir une bibliotheque C++ pour supprimer le pompage de l'intensité dans des vieux films qui sont déja transférés au format numérique. Le document a été divisé en cinq parties qui reflètent le travail effectué :

1. Projet (Project)
2. État de l'art (Understanding the problem)
3. Conception (Design)
4. Mise en ouvre (Implementation)
5. Conclusion (Recapitulation)

Dans la première partie le contexte du projet est présenté ainsi que ses objectifs et les résultats attendus. La méthodologie de travail décrite était développé au cours du projet. Elle a été adaptée a la nature du projet et en conséquence est devenue très agile.

La deuxième partie présente les connaissances acquises sur le problème. Dans le document il y a un bref résumé des 15 articles et une PhD thèse. Compte tenu du fait que le pompage se montre parmi d'autres éléments du film, quelques « cas d'utilisation » sont définis. Ce sont des situations typiques auxquelles les algorithmes créés devront faire face.

La troisième partie contient la conception de l'algorithme qui est sur la base de l'un des articles de la deuxième partie. Les conclusions sont basées sur des considérations théoriques et des tests pratiques. Les tests ont été effectués en utilisant des implémentations propres (quatrèeme partie). Aussi plusieurs d'améliorations ont été proposées. Cela fait le projet novateur.

La quatrième partie décrit la mise en ouvre de la bibliotheque C++. La conception du logiciel est basée sur les meilleures pratiques du génie logiciel. La méthodologie est présentée séparément. Bien que la partie est purement technique, elle est devenue décisive pour la réussite du projet. Ce n'est pas possible de tirer des conclusions sur la qualité des algorithmes sans avoir une implémentation fiable. Aussi la mise en ouvre a pris plus de temps.

Contents

Introduction

There are many elements of the world cultural heritage. Throughout the history the man built, painted, wrote or composed. Most of the works have already seen a lot human generations. The culture is the memory of mankind. It makes us move forward. That's why it is important to protect it for future generations. Unfortunately, not always were we success. We can mention for example the Library of Alexandria or Buddha statues at Bamyan. However there a lot of smaller examples - like paintings that are aging and disappearing.

More than a hundred years ago the man began making movies. Very soon they became an important part of our culture. They are characterized by a completely new language than have for example theater or book.
Film stock became an information carrier. First movies were put on an unstable, highly flammable cellulose nitrate film base. They are liable to continual and inevitable aging process. Even later cellulose-triacetate film stocks are not eternal. They all require careful storage. There are also many protection techniques as there are techniques to protect paintings.

Information-film is independent of its carrier like the literature. The best way of movie protection is to digitalize them. The name of process is *telecine*. We don't lose any information as we don't lose nothing prescribing a book to a computer. The main advantages that we should mention are:

- film becomes eternal,
- film is not changing with time or during copying,
- it is possible to boundlessly duplicate,
- in the age of Internet - easy and fast transfer;

Many of content is now free from licensing restrictions. And there are still new releases. Let's try to imagine the Internet of future. We type into (or say) a web browser that we want to find information, for example, about actress Ingrid Bergman. Of course we would find the biography and

infromation about movies where she played. But let's say now to our browser: *Ingrid Bergman Humphrey Bogart scene*. As a result we would expect that we could watch all scenes where Ingrid Bergman and Humphrey Bogart are together. Queries are limited only by our imagination. Number of movies made and its increment force the automation of creation process of such a knowledge base. The current state of science makes it possible. There already exist algorithms for face recognition in digital movies. However, they require a certain quality of the material. Old films are characterized by the presence of many distortions (like noise, scratches, blotches, ...). For the future Web is important to fix them.

Also, owners of content are very interested in film restoration. There is a niche of viewers who would be able to pay for old films that are once again returned to life (for example in HD).

One of the significant film distortion is the flicker - random or regular short period variations of screen luminous intensity. The main cause is irregular exposure of frames. The flicker is undesirable for two reasons. It significantly reduces the viewing pleasure. Watching movies with the flicker can be tiring for a viewer. The second reason is reduction of quality of image processing algorithms. Most of them rely heavily on the absolute value of pixels. The easiest way to see it is to compare the size of compressed movies (for example with MPEG) with and without flicker. When compressing video with the high flicker there's no motion estimation made. Thus the file can be several times bigger. This makes an important difference in network transmission like the Internet.

In conclusion we can say that provided work - C++ library with the algorithm design - in the wider context can have a big significance.

Glossary

f	brightness distortion function.
I	intensity of movie frame.
I^O	intensity of movie frame without flicker.
I^R	intensity of restored movie frame.
t	time.
$\mathbf{x}$	pixel location (x, y).
X	horizontal dimension of frame.
Y	vertical dimension of frame.

Acronyms

LUT	Lookup table.
MAP	Maximum a posteriori.
ML	Maximum likelihood.
pdf	Probability density function.

Part I

Project

Chapter 1

Project Environment

The project was made for the company Technicolor SA during a diploma internship. A short presentation of the enterprise can be found in the appendix A. Starting work in the Hanover branch (1 January), I became a part of the team of Image Processing Lab. I was introduced to the project "THESEUS/-CONTENTUS". "THESEUS is a research program initiated by the Federal Ministry of Economy and Technology (BMWi) in Germany, with the goal of developing a new Internet-based infrastructure in order to better use and utilize the knowledge available on the Internet." [The]. One of the types of information are digitalized movies. In order to enable the extraction the content from them they should be of certain quality. However, many old films have artifacts that reduce the quality of image processing algorithms. Information on this can be found on the joanneum project site [Joa]. Developed tools should also be used to create products for independent customers such as owners of movie content especially in Hollywood. Subject of the project fits perfectly into a new business strategy described in appendix A.

The goal of team that I became a part of is to create tools for automatic film restoration. The challenges include, inter alia, the removal of noise, scratches, dirts and so on [Joa]. The movie restoration project is carried out in cooperation with the Techicolor branch in Rennes. This corresponds to elements of matrix management in natural divisional structure of the corporation.

The division of tasks in our team occurred based on the theme. For each of us generally is assigned one movie distortion to cope with. Methods of work has not been imposed. The final result of the work of a single person should be:

- C++ library which solves the problem,
- the documentation of the code,

- the documentation of used algorithms with general study of the problem;

This document responses to the last two points. The shape of the library hasn't been imposed as far as the user's specification or an example of use will be delivered. Depending on the complexity of the solutions it can be a single function up to even a huge object-oriented system.

At the end of 2009 when I was working in the Villingen branch I was informed that my main task will be to deal with the flicker. In January of this year I began to work in the branch in Hanover. In connection with the reorganization of the company the start of my project moved to the beginning of February. I spent the first month to familiarize myself with the organization of the company. I learned about used methodologies of project management like SCRUM. I had learned tools which later I was using every day in my engineering work. And above all I was integrating with employees of the company in order to create the best work atmosphere.

Chapter 2

Project Objectives

My work is a part of team project which aims to create tools for old-film restoration. Primary task entrusted to me sounded in general:

- **Provide a C/C++ library for removing flicker distortion and a small test executable to demonstrate the used approach.**

The start of work was adopted to **February 1, 2010**. The work should be completed before the end of my internship in the company that is **August 31, 2010**. The methodology of work remained to be my choice. I was left with a large autonomy. These three factors (aim, limit dates and autonomy) let me to treat my work as a separate project. Only I was responsable to carry it out. My manager has become my client. From the perspective of the project other colleagues also became "actors" - generally advisors.

From my own observations, I conclude that the work of students in a company is often forgotten and unused. Higher productivity of the whole team is often achieved through a greater autonomy of its participants. Doing so, however, there is a risk of loosing the knowledge together with leaving of its holder. Natural continuity of know-how is obtained by overlapping competences and knowledge of workers. In big structures where decisions are mostly organizational the way of technical communication is a documentation. Documentation does not always mean the same thing, and its quality depends on the culture of work.

My additional goal is:

- **My work should have an economic value and should be used in the development of the final product.**

I take as determinants of that:

- Integration time of my work with an external system.

- Time of changes in various parts of my work if needed.
- My work shouldn't impose on the user any additional recurrent or non-recurrent charges (for example for licenses). And above all it must respect the copyrights of any other institutions.

Additional duties are imposed by the other "actor" of the project - Telecom Bretagne. Main of them are to provide:

- **Work Plan,**
- **Progress Report,**
- **Final Report (this document),**
- **Short Summary Report,**
- **Oral presentation about the internship;**

The details can be found in internal documents of Telecom Bretagne: "livrables SFE", "Evaluation Stage de fin d'etudes" and "GUIDE DE STAGE DE FIN D'ETUDES".

Chapter 3

Methodology

It was decided to treat my work as a separate project. The advantages of this approach are:

- I know exactly when the work is completed,
- I feel full responsibility for my work. It is more efficient;

During the project realizaton I also noticed a few drawbacks of such an organization:

- Projects carried out by team members are related to the various topics (scratches, dirts, flicker, ...), but all are moving in one area which is Movie/Image Processing. It is very likely that our component tasks would simply be duplicated. Example could be a reading film from a file during tests,
- Lack of a common interface to our products will hinder the system integrator;

The proposed solution would be to specify the joint and cross-cutting tasks that should be solved by a separate person or group. Not opting for a common interface we are forcing systems integrator to build adapters.

We can easily guess that the work on C++ library for image processing will consist of two parts: mathemathical and computer science. I need to provide two products: design of the algorithm and its implementation. My experience says that these different parts are inextricably linked. Implementation shouldn't be done even by the best programmer but with no mathematical understanding of the problem. The programmer must be a mathematician. Programmer tool has its limitations. For example, a mathematician can manipulate equations over real numbers what is not possible

for a programmer. Perhaps because of the performance the programmer may wish to use a fix-point representation. The programmer must be aware of the consequences in the world of mathematics of any such a restriction or made compromis. Generally speaking, if someome wants to manipulate with the code (for example in optimization), then he should know and understand the algorithm.

What in that case means *more economic value*? It is difficult to expect that the system integrator would be a great software developer and also a good mathematician. If such experts exist, they are very expensive. My keys to success are:

- Treat a system integrator as the final client-user of my project,
- Implement a product, not a demo version,
- The most simple and intuitive interface to my library;

A simple interface allows to treat my library as a black box. The system itegrator has frames of movie with distortion. He puts these frames into library or he use the library on these frames and he gets new frames without distortion. The advantages are:

- Final system can being created before work on libraries will be completed,
- The library can be changed or optimized even when the final product-system is finished;

Despite the existance of many techniques for code managing still one of the worst thing for the programmer is reading the code written by someone else. I can't expect that anyone will read my entire code. Basing on my observations I conclude that "demo versions" are often wasted. Most of the demo code, especially that without documentation, is discarded. Even obtaining positive results with the demo version most of the work would have to be rewritten. It would require from a new programmer to explore the mathematical details. This would increase the likelihood of implementing errors. He may not also take into account the same compromises between the tool and mathematical theory as it was done with demo version. Thus, the project should be completed by the finished product. Basing on a simple demo application the system integrator should already understand how to use the library.

For those who would like to develop the library this document contain:

- A complete description of the problem and implemented algorithm (part II and III),
- The organization of the library and all its modules (part IV);

3.1 Project Management

An important part of any project is a management. There are many techniques and models. My project has a dual nature. On the one hand it is a mathematical research. But on the other it is a software engineering. The classical approach would be to make first a mathematical research with an appropriate mathodology and after make implementation in the best way. Mathematical tests probably could be done in Matlab environment. As part of such an approach would be a plan in the form of the Gantt chart. Initially I also went that way.

Not having a great experience in a project management I didn't want to keep rigidly to once adopted rules. For the first month and a half I was trying elements of several methodologies with continuous development of my work. Paradigms presented below stem from the experiences with working on this project. The final outcome of the project is the result of applying them in my work.

Understanding the problem

This paradigm comes from the book *How to solve it* [Pol57] written by G.Polya a Hungarian mathematician. The book itself has had a big inspiration for me during this project.

It is impossible to create credible plans wihout knowing and understanding the issue. One of the first task should be a reading about stated problem. Any plan or Gantt chart, which contains the point "State of the Art" shouldn't be considered as credible. Only after good orientation a plan can be devised.

Design and implementation together

The most important is to achieve a goal. Duration of the project is limited. Testing the mathematical algorithms in Matlab and after transfer them to C++ actually comes down to do the same work twice. Grammar of these two languages are different. Each low-level part of the algorithm would have to be written from the beginning.

So why to use MATLAB?

Easy syntax: It's true that the Matlab language is simpler. However, for advanced C++ programmer it isn't a point. C++ is more difficult to learn, but after it offers to programmer a lot more.

Many built-in libraries: Solved problem turned out to be not custom. Most of the implemented functions have no equivalent in the Matlab libraries.

Rich set of 2D/3D visualization: This is probably the fundamental reason why the Mathworks product become so successful. There is no open C/C++ library that would offer a replacement. The visualization of results is essential during designing algorithms, but no longer during their target implementation. Interestingly Matlab still doesn't support very well Video Processing. Only in Simulink there is the ability to buy the Image and Video Processing BlockSet. However, my previous experience tells me that this tool on a typical computer PC is only suitable for testing single and simple algorithms, and not to simulate the hole video-processing systems. The limit arises from performance.

Nevertheless Matlab was still a primarily tool used for functional testing of implementation (described in section 14.2).

Small steps

Each task can be split into smaller ones. Reading and understanding an article can be considered as reading and understanding individual paragraphs. Implementation of the system becomes an implementations of functions. Every day from one to several tasks should be done. The most important is to **finish a day with a fully completed task**. A product developed in this way is always doing something. And above all, it always works. The tasks are not defined a priori. They appear during the progress of the project. Priorities for the known tasks should to be adjusted so that as soon as possible we would achieve the product desired by the customer. So, for example, video-processing library to remove a flicker distortion is primarily a library. The library is developed in the direction to allow the video-processing. Video-processing functionality is being developed as far as it is required by the flicker-removal algorithm. After reaching the final shape of the product the new improved version should being released. This approach is inspired by *sprint* from the SCRUM methodology. It has many advantages:

Very agile: It is appropriate to research where step results can change the global conception.

Very motivating: Even if the target is still far away, every day we feel that we draw to it. Easily we can summarize the work so far done.

Few low-level errors: Each small step can and should be tested. If after a step the project stops working correctly, we know exactly where lies the reason or at least from where start to search for an error.

Work automation

The only resource that may run out is time. The project time should be spent in the most efficient way. Each time when we do something often that doesn't require us to think we should try to do it automated. This is the reason why the report is written using LaTeX. It relieves us from responsibility for the appearance of the document. It is also why it was decided to use a code generator. During implementation we shouldn't think about programming-language structure but only about a functionality.

To increase security and simplify the management of versions (both documents and code) the repository SVN was used.

Part II

Understanding the problem

Chapter 4

Introduction to the flicker

The flicker is generally described as "random or regular short period variations of screen luminous intensity" [Joa]. But this is not a formal definition and it is not possible to find one. A counter-example that satisfies the description and isn't an unwanted part of the film could be always found. The best is to look at examples. Many can be found on the attached DVD. Two of them are presented in Figures 4.1 and 4.2 as sequence of frames (from left to right).

In the first example we see a global change of luminance. The frames from the second row are brighter than those from the first or third row. It should be noted that the presented frames are just half a second of the movie. The second example comes from the site [Joa]. This is a very damaged film in which we can try to find even local variations in luminance.

Figure 4.1: Frames from the movie *The Vagabond* of Charlie Chaplin - 1916

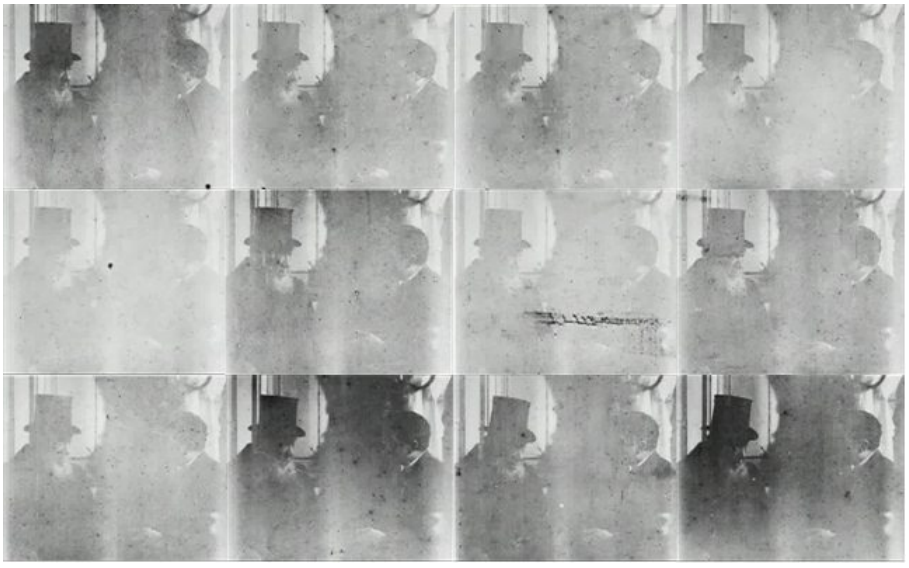


Figure 4.2: Frames from the movie *Paul Kruger in Gesprek* - 1900

Chapter 5

State of the Art

The digital treatment of the flicker is described in the literature by many researchers relatively wide. Nevertheless, we can distinguish two general approaches to the arifact. Initially, the problem was seen as a global distortion. It means that an entire frame should be corrected in uniform manner. Subsequent works draw attention to the spatial variability.

5.1 Global compensation

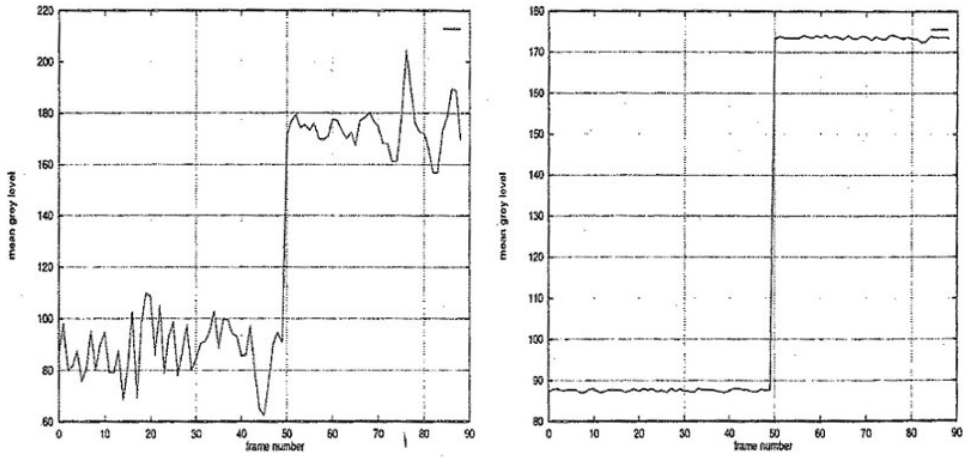
1. For [WD95] luminance fluctuations are only question of histogram shifting. If the average grey level in the current frame is different from average grey level in the scene, correction is applied. What is important in this work that the flicker is considered as a problem of scenes (shots). Conception has been described by equation 5.1.

$$I^R{}_i = I_i + \Delta I_i \ , \ if \ |\Delta I_i| > \beta \tag{5.1}$$

   Where i is a index frame and ΔI_i is a difference between i'th frame grey level and a shot mean grey level $\Delta I_i = I_i - I_{seq}$. β is a treshold determining a difference between frames mean grey level. In Figure 5.1 there are tests carried out by the authors.

   A good part of this approach is the simplicity and relatively few mathematical operations. Weak point is that the fluctuations in a mean grey level are not only caused by flicker (also camera pan, changes of content, ...). As a result, we could expect a situation that corrected film could get an additional flicker in restoration process because of some changes in the content.

   Such a solution is not necessarily bad. For example, if the input sequence has already characteristics as the second chart in Figure 5.1



(e) Mean grey level of uncorrected "ned1.90" (f) Mean grey level of corrected "ned1.90"

Figure 5.1: Comparison of brightness uncorrected and corrected frames ($\beta = 0$) selected from a sequence from *"Story of the Kelly Gang"* avaible on DVD - *source:* [WD95]

shows, it would be clear information that there is no flicker effect.

From a practical point of view, if the flicker-restoration subsystem had to be independent, it would require to operate twice on the same sequence - for the first time to determine the average grey level, then for artifact detection. The algorithm could be redesigned to operate on appropriate moving average grey level.

2. In [NA00] firstly also time-model methodes were tested. The authors suggest the existence of three components in time representation of mean grey level and of grey-level variance:

   - Random variations
   - Periodic variations
   - Linear tendency

   This is presented in Figure 5.2. The output characteristic is considered by the authors as a superposition. The idea is to leave the linear component and reduce the rest. A simple histogram shifting was used. The authors confirmed an improvement in image contrast but they didn't reduce a flicker.

   The second proposed method was a histogram equlization into a target histogram. The target histogram for certain frame is computed by

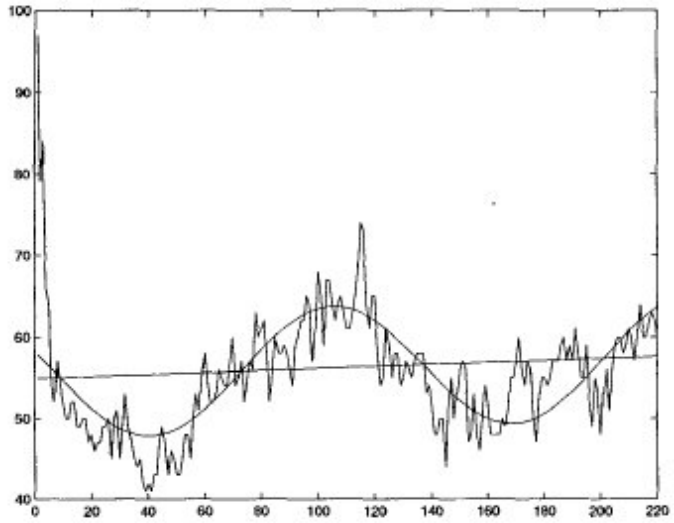


Figure 5.2: Linear tendency and periodic component of mean grey level in time - *source:* [NA00]

averaging the histograms of neighbouring frames. The global transformation from original histogram to target can be represented by the formula 5.2.

$$H[r] = T_2^{-1}[T_1[r]] \tag{5.2}$$

Where $T_1[r]$ is the intesity transformation used to change the original histogram into a uniform one. T_2 is intensity transformation used to change the target histogram into a uniform one. Implementation could be made by one Lookup table (LUT).

The main advantage of this method is its relative simplicity and small number of mathematical operations.

3. We can find a very comprehensive study in [SPH99]. For the authors the brightness variations differentiate into two categories:

   - Flicker - global (homogenous inside a frame) brightness variations from one frame to the next. The main cause is irregular exposure time in early movie camera.
   - Mould (mold) - results from an improper storage environment for the film material. It causes local (inhomogeneous inside a frame) brightness over a few frames of an image sequence.

   According to the article an amount of flicker variation can go up to 10% of full grey-value range.

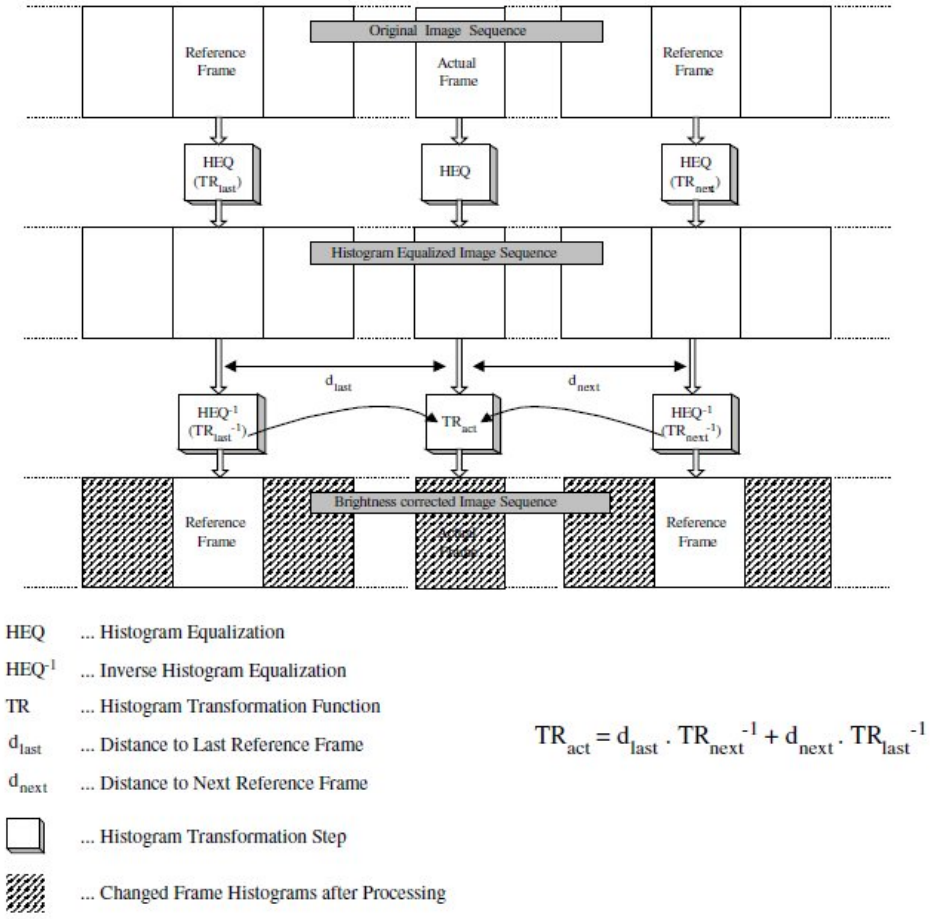


Figure 5.3: Scheme of temporal brightness correction. - *source:* [SPH99]

Removal method was also proposed. The concept is shown in the Figure 5.3. A two-step histogram transformation procedure is used. First, all frames in the image sequence are histogram equalized. Then a nonreference frame is corrected by the linear combination of inverse histogram-equalization functions of the nearest reference frames. The idea is similar to that discussed in paragraph 2. In the place of the target histogram, histogram of the reference frames is taken into account. Automatic selection of the reference frame isn't proposed. An interesting idea is to adjust the distance to the reference frames, depending on the amount of change in the content of the scene.

4. In [Vla04] the main contributing cause of flicker is inconsistent film ex-

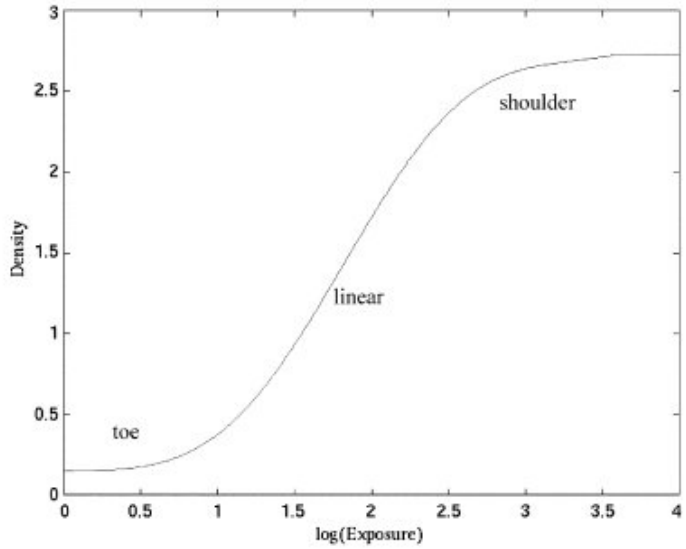


Figure 5.4: Example of a Hurter-Driffield characteristic. - *source:* [Vla04]

posure at the image acquisition stage. The author introduces a nonlinear model of flicker which is based on a *Hurter-Driffield characteristics*[1] from the theory of photography[2]. This family of characteristics present a relationship between density and log of exposure for a photographic film. And the density is logarithmically related to the opacity of film. Example of such a curve is in Figure 5.4. We can deduce from the curve that at the wrong time of exposure the absolute value of density error (therefore brightness) will be different for various values of brightness levels of frame not affected by artifact.[3]. An example of the error characteristics is in Figure 5.5 The proposed removal algorithm requires an indication of a reference frame. The relationship between the grey levels of co-sited pixels in the reference and degraded images is:

$$I = \Delta I_t(I_{ref}) + I_{ref} \tag{5.3}$$

Where I_{ref} grey level of reference frame F_{ref} and $\Delta I_t(I_{ref})$ represents a suitable greyscale error profile. The reference frame is selected among

[1]Also known as: characteristic curve, H-D curve, HD curve, D-logE curve or D-logH curve

[2]Sensitometry and densitometry

[3]In simple words we can say that a pixel, which should be almost white in nornal exposition, will be not whiter than white in overexposed frame. So in that case error is smaller

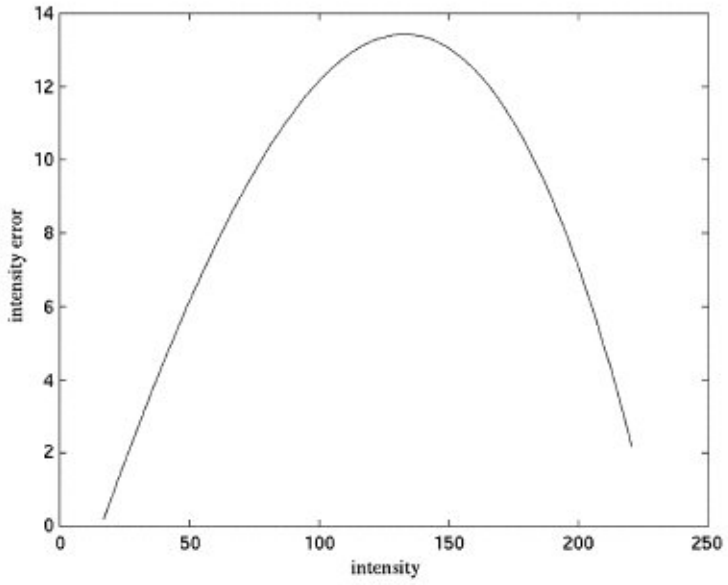


Figure 5.5: Intensity error profile as a function of intensity (all units are grey levels). - *source:* [Vla04]

available frames (for example the first frame in shot). Unknown is $\Delta I_t(I_{ref})$. In practice, the HD characteristic of film is not available. Instead, author proposes the following algorithm to estimate the error characteristic:

(a) For each grey level I_{ref} (e.g. from 0 to 255): Count the differences between the corresponding pixels in current frame and F_{ref} where the value of pixel in F_{ref} is I_{ref}.

(b) Compute a histogram from these differences. $\Delta I_t(I_{ref})$ is an argument of a maximum value of the histogram.

$$\Delta I_t(I_{ref}) = \arg\max H_t(I_{ref}) \tag{5.4}$$

(c) Repeat the process for the next grey level.

(d) At the end apply a cubic least-squares approximation for correction of obteined profile.

In the paper is also said that longer scenes would require reference frame updating. The author stresses that his solution has a double protection against influence of motion in content. Firstly, only the maximum value of the histogram is taken into account. Secondly, the characteristics of error is approximated. It's the difference to the algorithms

discussed earlier where the intensity shift was fixed using all values of the histogram. Author indicates that the algorithm can also cope with large uniform-intensity area moving against a uniform-intensity background which usual are the problem for flicker removal and detection algorithms.

The author says that he obtained better results than [SPH99] and [vRJBRL00] - paragraph 3 of 5.1 and 1 of 5.2.

5. In [Del06] the author develops the conception of Naranjo [NA00]. New interesting observation is that the evolution of the current frame's histogram during time is not always smooth and can present severe jumps. But the author presents completly different point of view on flicker model from the others. Here, the flicker is mainly a problem of loosing the contrast in images.

5.2 Local compensation

1. One of the frequently cited items in the literature is Roosmalen's algorithm [vRJBRL00] [vRRLLJB99]. It has been applied in the AURORA project [4]. As a starting point the author defines a general model of distortion - equation (5.5).

$$I(\mathbf{x},t) = \alpha(\mathbf{x},t)I^O(\mathbf{x},t) + \beta(\mathbf{x},t) + \eta(\mathbf{x},t) \tag{5.5}$$

I^O may have already been distorted by other artifacts. The authors distinguishes multiplicative α and additive β parameters. They assume that these parameters are spatially smooth functions. The flicker independent noise is represented by η. It is zero-mean signal uncorrelated with the original image intensities but with known variance (e.g. quantization noise).

α and β are unknows. In practice, their calculation is as follows. The parameters are assumed to be locally constant. First we should divide the image into square regions. From (5.5) we can obtain (5.6) and (5.7).

$$\beta_{m,n}(t) = E[I(\mathbf{x},t)] - \alpha_{m,n}(t)E[I^O(\mathbf{x},t)] \tag{5.6}$$

[4] "A project named AURORA was initiated in 1995, stimulated by the European Union ACTS program. The acronym AURORA stands for AUtomated Restoration of ORiginal film and video Archives. The objective of this 3-year project was to create state-of-the-art algorithms in real-time hardware for the restoration of old video and film sequences." - *source* [ROO99]

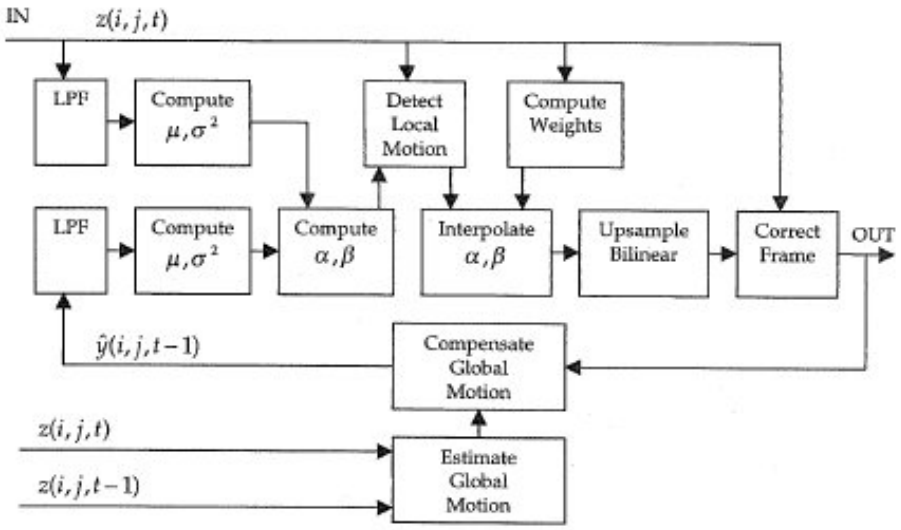


Figure 5.6: Global structure of the intensity-flicker correction system - *source:* [vRJBRL00]

$$\alpha_{m,n}(t) = \sqrt{\frac{var[I(\mathbf{x},t)] - var[\eta(\mathbf{x},t)]}{var[I^O(\mathbf{x},t)]}} \tag{5.7}$$

Where $\alpha_{m,n}(t)$ and $\beta_{m,n}(t)$ are for each block and $\mathbf{x}$ belongs to this block $\mathbf{x} \in \Omega_{m,n}$.

In the above, apart from noise, $E[I^O(\mathbf{x},t)]$ and $var[I^O(\mathbf{x},t)]$ remain unknowns. The authors propose to take these values from the frame corrected previously (or reference frame for the first time) with the requirement of temporal stationarity. This approach is more attractive in terms of computational load and memory requirements.

The problem may be adding and propagation of error in the sequence. However, the author proposes one solution ad hoc.

The real problem is also the lack of stationarity in the blocks due to movement in content. The authors propose relatively simple methods for compensation of global and local motions. For global motion a phase correlator is proposed. Local motions are treated as lack of information. In such blocks parameters are missing. The authors propose three methods of its calculation by interpolation. After, all values are smoothed. The complete system is presented in Figure 5.6.

2. In [TSTK00] and [OSKS00] the authors adopt the same model of flicker as (5.5) but wtihout the noise. What is new that they present a hierachy

of flicker-correction models:

$$
\begin{aligned}
II : I^R(\mathbf{x},t) &= \gamma(t)I(\mathbf{x},t) + \delta(t) && (5.8)\\
VI : I^R(\mathbf{x},t) &= \gamma(\mathbf{x},t)I(\mathbf{x},t) + \delta(t) && (5.9)\\
IV : I^R(\mathbf{x},t) &= \gamma(t)I(\mathbf{x},t) + \delta(\mathbf{x},t) && (5.10)\\
VV : I^R(\mathbf{x},t) &= \gamma(\mathbf{x},t)I(\mathbf{x},t) + \delta(\mathbf{x},t) && (5.11)
\end{aligned}
$$

Where γ and δ are restoration parameters defined as two-dimensional polynomial functions. In (5.8) we treat $I(\mathbf{x},t))$ that its flicker distortion parameters are only spatially constant and in (5.11) each parameter is spatially variant.

The idea is to progressively apply more and more complicated model. The authors consider as the best combination (5.8), (5.9) and (5.11) ($II \rightarrow VI \rightarrow VV$). In the first step we obtain:

$$\hat{I^R}_{II}(\mathbf{x}) = \hat{\gamma} I(\mathbf{x}) + \hat{\delta} \tag{5.12}$$

Where $\hat{\gamma}$ and $\hat{\delta}$ are estimators. Due to the fact that the formulas are not derived in the article and consequently one of them is printed with an error, the deduction is presented below:

From (5.12) and (5.5) we get:

$$
\begin{aligned}
\hat{I^R}_{II}(\mathbf{x}) &= \hat{\gamma}\left(\, \alpha I^O(\mathbf{x}) + \beta \,\right) + \hat{\delta} && (5.13)\\
&= \hat{\gamma}\alpha I^O(\mathbf{x}) + \hat{\gamma}\beta + \hat{\delta} && (5.14)
\end{aligned}
$$

That estimator corespond to the original image, we must ensure:

$$
\begin{aligned}
\hat{\gamma}\alpha &= 1 && (5.15)\\
\hat{\gamma}\beta + \hat{\delta} &= 0 && (5.16)
\end{aligned}
$$

Consequently:

$$
\begin{aligned}
\hat{\gamma} &= 1/\alpha && (5.17)\\
\hat{\delta} &= -\hat{\gamma}\beta && (5.18)
\end{aligned}
$$

From (5.5) we get:

$$
\begin{aligned}
E[I(\mathbf{x})] &= \alpha E[I^O(\mathbf{x})] + \beta && (5.19)\\
var[I(\mathbf{x})] &= \alpha^2 var[I^O(\mathbf{x})] && (5.20)
\end{aligned}
$$

$$\alpha = \sqrt{\frac{var[I(\mathbf{x})]}{var[I^O(\mathbf{x})]}} \tag{5.21}$$

$$\beta = E[I(\mathbf{x})] - \alpha E[I^O(\mathbf{x})] \tag{5.22}$$

From (5.22), (5.21), (5.18) and (5.17) we get;

$$\hat{\gamma} = \sqrt{\frac{var[I^O(\mathbf{x})]}{var[I(\mathbf{x})]}} \tag{5.23}$$

$$\hat{\delta} = E[I^O(\mathbf{x})] - \hat{\gamma} E[I(\mathbf{x})] \tag{5.24}$$

We don't have access to I^O so the authors indicate the need of a reference frame.

As a next step is model (5.9). We shouldn't forget that the parameters are modeled as two-dimensional polynomials.

$$I^R{}_{VI}(\mathbf{x}) = \gamma(\mathbf{x}) I(\mathbf{x}) + \delta \tag{5.25}$$

$$= \left(\sum_{i,j} \gamma_{ij} x^i y^j \right) I(\mathbf{x}) + \delta \tag{5.26}$$

But the authors propose to use (5.27).

$$\hat{I^R}{}_{II \to VI}(\mathbf{x}) = \hat{I^R}{}_{II}(\mathbf{x}) + \left(\sum_{i,j} \gamma'_{ij} x^i y^j \right) I(\mathbf{x}) + \delta' \tag{5.27}$$

In the above equation, a flicker-corrected image is given by adding perturbations to the flicker-corrected image produced in previous step. The new estimators γ'_{ij} and δ' are calculated iteratively by minimizing error:

$$\epsilon = I^O(\mathbf{x}) - \hat{g}_{II \to VI}(\mathbf{x}) \tag{5.28}$$

As a estimation method Tukey's biweight function was adopted from the robust statistics[5]. Details can be found in the articles. Iteration consists in finding the arguments (γ'_{ij} and δ') for which the robust function has minimum and then use these values to re-calculate the error. For the first iteration in place of $\hat{I^R}{}_{II \to VI}(\mathbf{x})$ is used $\hat{I^R}{}_{II}(\mathbf{x})$. The authors don't provide conditions for the end of the iteration process, or whether if this process is convergent.

The idea of using the robust estiamtion for the parameters may be attractive in terms of reducing the bad influence of image motion and others distortion as the outliners of image stationarity.

[5]Good introduction to robust statistics can be found in [Hub09]

3. Further works were delivered by the group which was working on the project BRAVA[6]. The first article is [H.D03]. The authors consider that an image affected by the flicker shows fluctuations in intensty between frames that are spatially varying of low frequency and temporally varying at a high frequency. Firstly, the authors take the linear robust estimation scheme as in paragraph 2 but of two consecutive frames:

$$I_{n+1}(\mathbf{x}) = a(\mathbf{x})I_n(\mathbf{x}) + b(\mathbf{x}) + \epsilon(\mathbf{x}) \tag{5.29}$$

For the basis for spatially representing the parameters they propose cosine function instead of polynomial.

$$\Phi_{pq}(m,n) = \alpha_p\alpha_q \cos\left(\frac{\pi p(2n+1)}{2X}\right)\cos\left(\frac{\pi q(2n+1)}{2Y}\right) \tag{5.30}$$

$$where\ \alpha_q = \begin{cases} n/\sqrt{X} & q = 0 \\ \sqrt{2/X} & 1 \leq q \leq X-1 \end{cases}$$

$$and\ \alpha_p = \begin{cases} n/\sqrt{Y} & p = 0 \\ \sqrt{2/Y} & 1 \leq p \leq Y-1 \end{cases}$$

Here $\Phi_{pq}(m,n)$ is the $p-q$'th basis over an image of size XxY. Thus the spatial model for gain parameter is given by:

$$a(\mathbf{x}) = \sum_{p,q} \alpha_{pq}(\mathbf{x})\Phi_{pq}(\mathbf{x}) \tag{5.31}$$

Where α_{pq} are coefficients to be estimated. A similar function exists for $b(\mathbf{x})$. Since the functions are orthogonal, this should provide a faster convergence.

To perform the estimation of flicker parameters the influence of local motions should be reduce. The principe is to minimizing the following robust function:

$$\jmath(\epsilon) = \sum_{\mathbf{x}} \rho\left(\frac{\epsilon(\mathbf{x})}{\sigma_\rho}\right) \tag{5.32}$$

Where $\rho(x) = \frac{1}{2}\log(1+x^2)$ is a Cauchy's function use as weighting functional.

[6]"The Brava Project (2/2000 - 11/2002) was supported by the European Commission within the Framework of the first Call of the IST (Information Society Technologies) Programme. The project was coordinated by Institut National de L'Audiovisuel , in partnership with Snell-Wilcox, leader in the domain of high-end video equipment, other broadcast archive owners, (RTP), Universities (TU Delft, TCD Dublin), SGT. The project aimed at developping tools for digital restoration of large amounts of broadcast archive documents (video and film), and for re-exploitation purposes." - *source* [BRA]

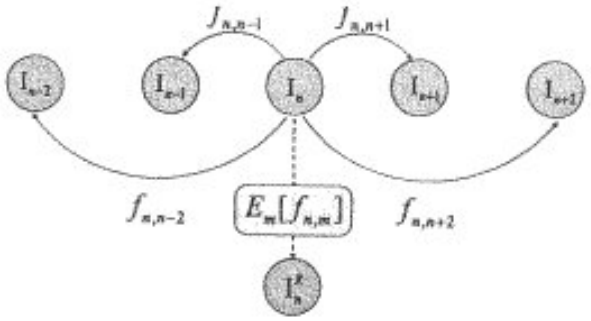


Figure 5.7: Compensation of frame: I_n original pictures, I_n^R restoration of I_n, $f_{n,m}$ brightness variation characteristic between frames I_n and I_m - *source:* [Pit04]

The authors indicate a strong relation between their flicker removal algorithm and a global motion (also shake). The practical solution should be to use a global motion estimator as a pre-process of deflicker agorithm.

4. The main keys to modeling in [Pit04] are spatial dependence and non-linearity. The authors focus on describing the changes in brightness between observed frames - in opposite to the others who try to restore distored frames. A new model was proposed:

$$I_m(\mathbf{x}) = \sum_i^N s(\mathbf{x} - \mathbf{x_i}) f_{n,m}^i(I_n(\mathbf{x})) \tag{5.33}$$

It describes the changes in brightness between two frames n and m. Where $\mathbf{x_i}$ are the positions of regularly spaced control-point pixels, $f_{n,m}^i$ is the non-linear transformation in these points and $s(\mathbf{x})$ is a interpolating 2D mask. The number of control points on one axis is defined as the *flicker order*. In practice, this value should vary from 3 to even 14 for old heavily demaged movies.

In Figure 5.7 all the estimations $f_{n,m}^i$ between I_n and its neighbouring frames I_m are considered.

To seperate the impulsive flicker in compensation a teporal robust expectation is computed.

$$\hat{f}_n(k) = E[f_{n,m}(k)] = \tag{5.34}$$

$$= \arg\min_{f_n(k)} \sum_{m=n-M}^{n+M} e^{-\frac{(m-n)^2}{\sigma_w}} \rho(f_{n,m}(k) - f_n(k)) \tag{5.35}$$

Where $f_{n,m}(k)$ is the k'th component of a lookup table $f_{n,m}$, ρ is a robust functional, an σ_w a temporal scale factor. The authors propose empirical value $M = 7$ (15 frames). To calculate $f_{n,m}(I)$ the authors firstly consider Naranjo method [NA00] (paragraph 2 section 5.1). After they propose to work on the joint histogram of two images. To determine the profile $f_{n,m}(I)$ from the histogram they use Maximum a posteriori (MAP) method from the Bayesian statistics.

Presented algorithm has many points that could make the restoration adaptive. Nevertheless it is very computationally exigent. The authors propose to make the calculation with a support from GPU (solution is cheap and very effective). They provide very representative results of performance tests. To expand the algorithm ability also for the color movies it is suggested to work only on their luminance.

5. In [FP06] the authors provide the most general model of flicker between two images m and n as a mapping s on the greyscale component that depends on the pixel location:

$$I_m(\mathbf{x}) = s(I_n(\mathbf{x}), \mathbf{x}) + \epsilon(\mathbf{x}) \tag{5.36}$$

The outlier term $\epsilon(\mathbf{x})$ accounts for the image disparities due to e.g. motions or missing data. The authors point out, following the other works, that from a practical point of view parameters estimation can't be conducted for each pixel. Moreover, flicker parameters seems to be spatially smooth functions. But in this article, the authors try to cope with brightness fluctuation at a pixel resolution. The principe is to find a model with only one parameter to decline computational requirements. The authors says that particular pixel is only affected by a percentage of the original flicker source:

$$I_m(\mathbf{x}) = \alpha(\mathbf{x})(s_0(I_n(\mathbf{x})) - I_n(\mathbf{x})) + I_n(\mathbf{x}) \tag{5.37}$$

Where $s_0(u(\mathbf{x}))$ is a global impact of flicker and α is a part of this changes affecting a particular pixel. It is presented in Figure 5.8.

It should be noted that the proposed method offers a generic framework, which would allow other flicker models to be incorporated.

6. The author of [SPH99] propose also a method for removal local brightness variations. The two-step histogram transformation procedure (as in paragraph 3 of section 5.1) is first applied to small regions of the actual processed frame. The resulting grey-value corrected regions are then merged. Overlaping of tiles is presented in Figure 5.9. The sizes of tiles are adapted localy by the amount of motion. 5.9

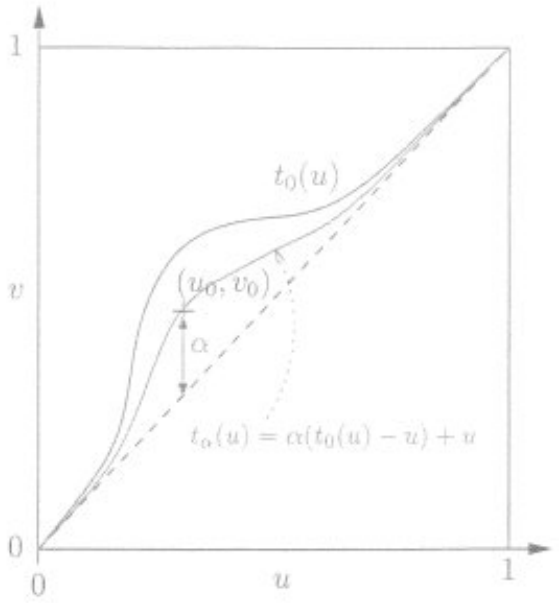


Figure 5.8: At a pixel $\mathbf{x}$, the flicker mapping s_α is linear combination of the identity (i.e no flicker) and the flicker model s_0 - *source:* [FP06]

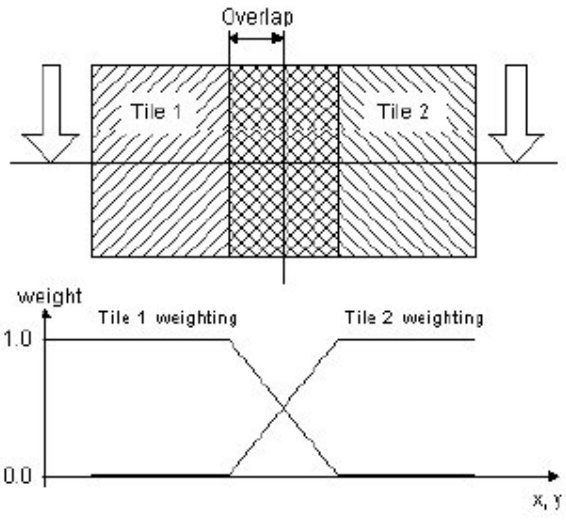


Figure 5.9: Weighting function for overlapping tiles - *source:* [SPH99]

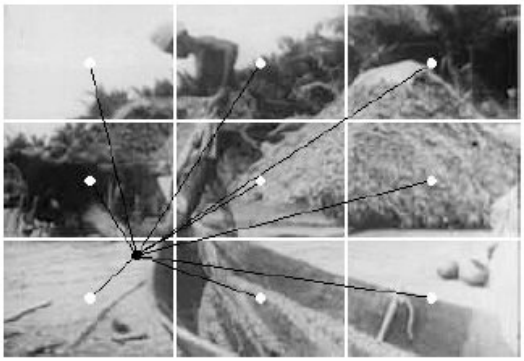


Figure 5.10: Split of the frame using 3 x 3 grid. The consider pixel and the centre of each blocks are represented by black and white dots respectively. The black lines represent the Euclidean distances. - *source:* [Tre05]

7. In [Tre05] authors improve their previous idea ([Vla04] and paragraph 4 in 5.1) with spatially adaptive compensation using block-based approach. The degraded frame is split into rectangular blocks and correction is applied individually to each block (Figure 5.10). Boundary effects are avoided using weighted bilinear interpolation based on Euclidean distance. The authors also consider to use a motion compensated weighting.

8. In [Tre06] only one improvement is introduced. Instead of motion compensated weighting, greylevel tracing is used (details in article).

9. In [Vla07] to frame division into blocks the additional segmentation into regions is added.

10. The whole work done by a group led by Dr. Theodore Vlachos was once again summed up in [For08] and [Vla08]. Each their new proposal is becoming more complicated. In Figure 5.11 the flow chart illustrate the complexity of their solution.

11. The Crowning of the Forbin's work is the PhD Thesis [For09]. It can be find on the website [For]. The content is a reflection of previously published articles. The interesting from the practical point of view is the comparison of existing and leading algorithms in the issue. On the website we can also find sample movies presenting the algorithms in work.

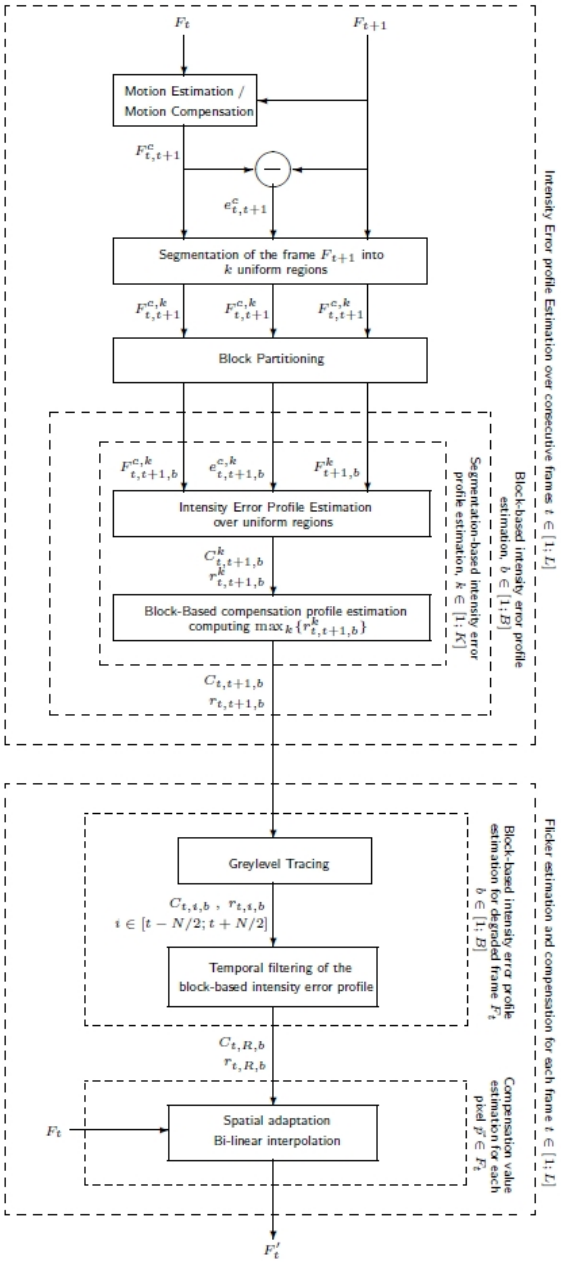


Figure 5.11: Flow chart of the proposed in [For08] compensation algorithm. The algorithms operates in two stages: intensity error profile over consecutive frames are first computed on a block-based basis. Afterwards these profiles are employed to calculate block-based compensation profiles related to a specific degraded frame, which are finally bi-linearly interpolated to obtained pixels compensation values - *source:* [For08]

5.3 Summary

As we can see that there is a huge number of various ideas. Most of them are the result of empirical tests. They often require the use of non-trivial mathematical operations. So it is hard to predict the results. And each of system or algoritm is a implementation challenge. Many of the proposed solutions require also an additional local or global motion estimator that themselves can be a great subject for a separate project.

Only two authors provide the results of its work in the form of restored films - G. Forbin on his webside [For] and F. Pitié on his reasearch group site [Sig].

Chapter 6

Use Cases

The flicker is one of distortions which overlaps the entire useful information (like noise). Useful information - or content of the film - is transfered by spatial and temporal variations in luminance. The same as flicker. Therefore, its detection is not a trivial process and may confuse with the normal content of movie. For this reason it was decided to provide test footages which represent typical situations in movies. They are listed in Table 6. Movies can be found on the attached DVD. The list can be extended.

Real movies are always a superposition of different content factors. The design of the algorithm and the library was first of all based on artificial footages. This allowed a more relevant analysis. A special application was written to create these footages (FlickerGeneratorDemoCons.exe). It is avaible also on DVD. The program is based on the engine described in chapter 15. If we add more effects at once, they will be placed in a specific order of the natural occurrence in the film. Structure with added all effects is shown in Figure 6.1.

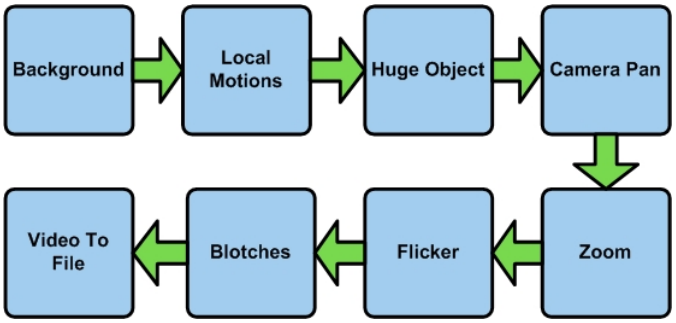
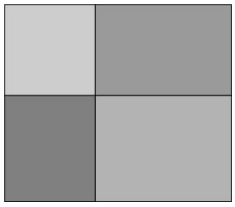
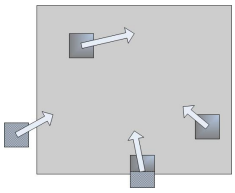
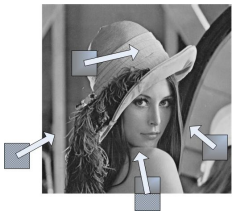
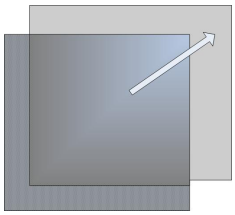
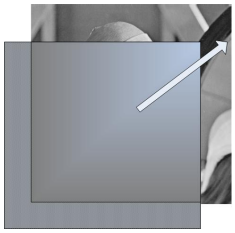
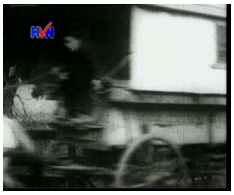


Figure 6.1: The order of diffrent effects in footage generator - FlickerGeneratorDemoCons.exe.

	Artificial		**Real**	
	Pattern	Image	Without flicker	With flicker
static background	 MovPatCl(FL)	 MovImgCl(FL)	 MovRealCl	 MovRealClFL
moving objects	 MovPatObj(FL)	 MovImgObj(FL)	 MovRealObj	 MovRealObjFL
huge object	 MovPatHuOb(FL)	 MovImgHuOb(FL)	 MovRealHuOb	 MovRealHuObFL

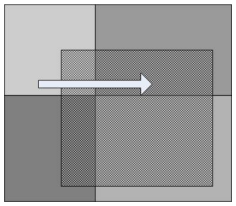
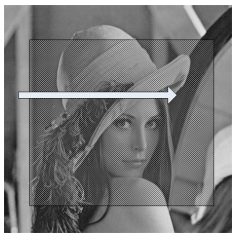
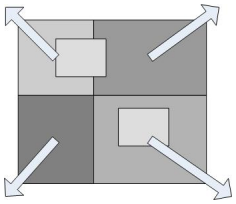
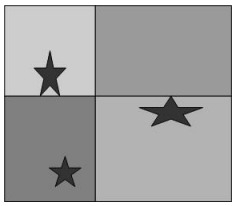
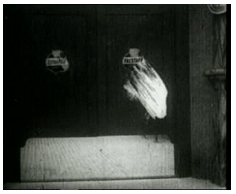
	Artificial		**Real**	
	Pattern	Image	With flicker	Without flicker
camera pan	 MovPatPan(FL)	 MovImgPan(FL)	 MovRealPan	 MovRealPanFL
zoom	 MovPatZoom(FL)	 MovImgZoom(FL)	 MovRealZoom	 MovRealZoomFL
blotches	 MovPatBlo(FL)	 MovImgBlo(FL)	 MovRealBlo	 MovRealBloFL

Table 6.1: List of footages used in the design and tests of algorithms.

Part III

Design

Chapter 7

Algorithm-base selection

Presented in the chapter 5 algorithms to remove the effect of flicker mostly are complex systems. In terms of implementation the difficulty is increased by the fact that we should operate at the same time on many frames of the film. It can be expected that the biggest challenge will be the data organization. The project time is limited. Therefore, the initial chosen way is so important. It's likely that there wouldn't be enough time to change diametrically the base of the algorithm.

The aim is to have the working library which removes the flicker. The final results can not be determined by equations or graphs. The quality of the final results will be judged by the viewer - human. It would be ideal to see a priori the effects of described algorithms. Unfortunately, only for three of them we can find the restored footages:

Roosmalen's algorithm described in the paragraph 1 of chapter 5.2.
The footages are avaible on the website [For]. The implementation is made by G. Forbin. There is also an AviSynth-plugin implementation in the Internet [Bal]. But it is not completed version.

Pitié's algorithm described in the paragraph 4 of chapter 5.2.
The footages are avaible either on the website of the authors [Sig] or on the Forbin's site [For]. The results obtained by Forbin are different from these presented by the authors - the output footages are brightened. The implementation made by Forbin seems to have errors.

Forbin's algorithm described in the paragraphs from 8 to 11 of chapter 5.2. The footages are avaible on the Forbin's website [For].

Incorrect implementation of the Pitié's algorithm by Forbin can put into question the quality of implementation of the Roosmalen's algorithm. Nevertheless we can positively assess the results of Forbin's and Pitié's algorithms.

Among these two algorithms it was decided to use Pitié's algorithm. In favour of that choice are:

- The algorithm is not covered by patents. It has been established on the basis of own searches and of an e-mail correspondence with the author.
- The algorithm was developed on the need for serious and huge European project - BRAVA[1].
- The algorithm is "gentle'. How it will be presented - most of the action can be justified. This is not a purely empirical algorithm.
- The algorithm is very configurable. Decisions between performance and quality can be parameterized and left to be determined by the final user.

[1] page 35

Chapter 8

Joint histogram

The brightness in one frame can be treated as a two-dimensional random variable. The value range usually corresponds to the grey levels. In practice they are often natural numbers from 0 (black) to 255 (white). A relationship between more than one random variables it is good to express by the joint Probability density function (pdf). The joint histogram is its estimator and can be counted with:

$$p_{n,m}[u,v] = \frac{\sum_{\mathbf{x}|(I_n(\mathbf{x})=I_m(\mathbf{x}))} 1}{X \cdot Y} \tag{8.1}$$

Graphical representation of the operation is in Figure 8.1. Each value in the histogram corresponds to the frequency of occurrence of certain pairs of pixel values in two images but in the same location. From the viewpoint of global brightness changes between frames we still possess full information. The information lost are the positions of pixels. One of the profits is the reduction of volume of the information. From the initial two matrices of unknown size, we have one of dimensions 256x256.

Let's now look for the usefulness of this information to detect the flicker. We can observe in the following figures the histograms for different use cases described in chapter 6. "Jet" colormap was used[1]. The colormap is every time normalized to a maximal value in matrix.

In the Figure 8.2 we can see that if the frames are identical, their joint histogram should be on a straight line. Maxima correspond to the biggest areas of the same grey level. The used footage is quite specific. One of the feature of picture "Lena"[2] is that it has values from almost whole grey range.

[1]Color map from blue to red - details on http:// www.mathworks.com/access/helpdesk/help/techdoc/ref/colormap.html

[2]The story of this picture can be found on *http://www.cs.cmu.edu/ chuck/lennapg/*

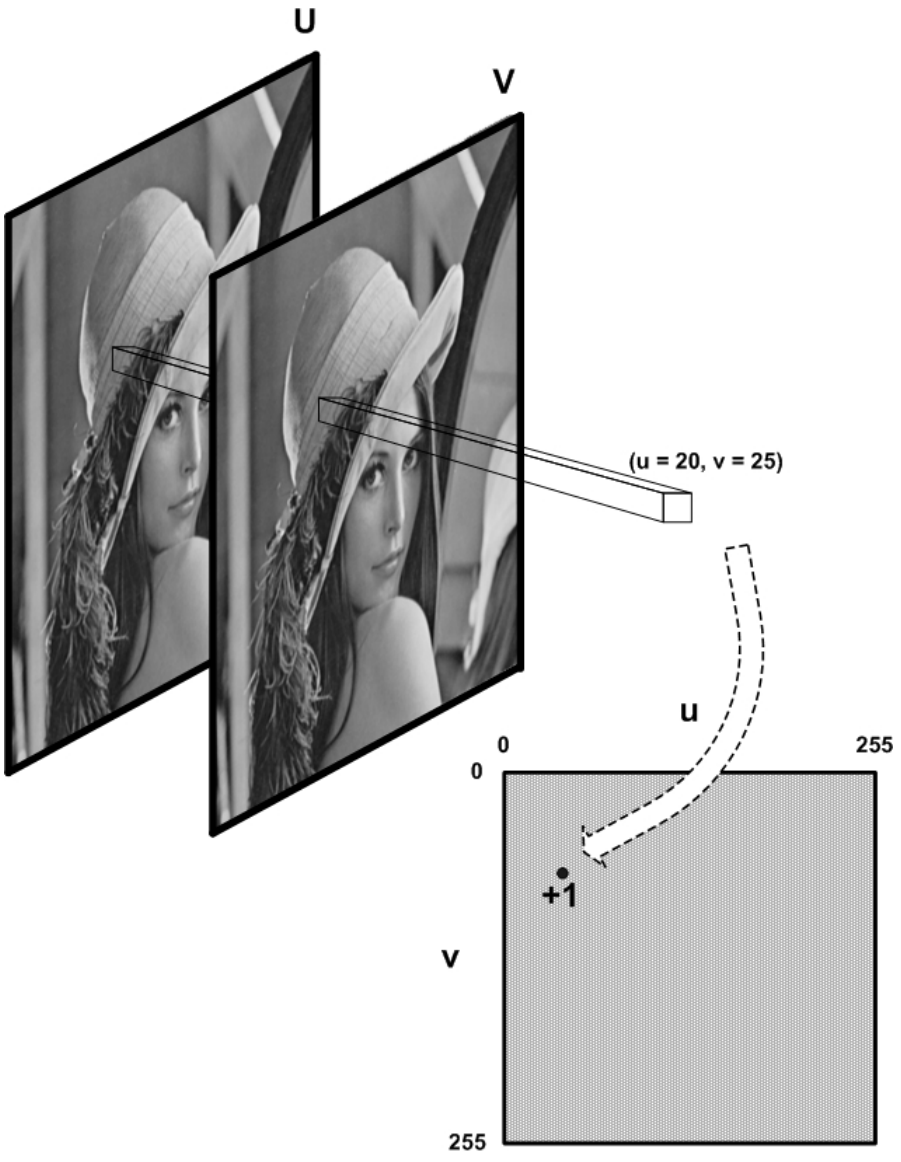


Figure 8.1: A way for calculating the joint histogram of two images

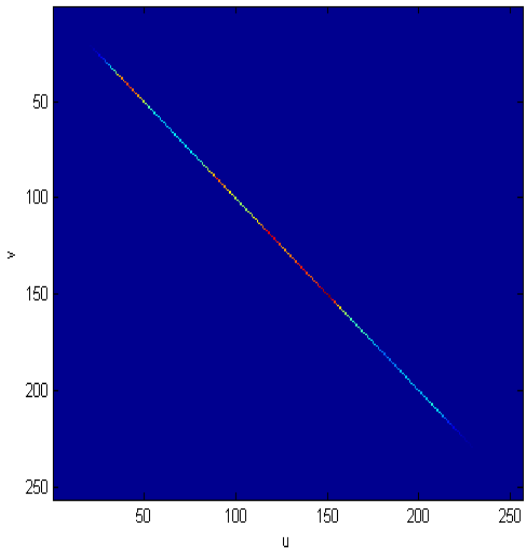


Figure 8.2: Joint histogram of two frames from MovImgCl

Adding the linear flicker only shifts the characteristics - Figure 8.3. Nonlinear flicker may also bend it geometrically - Figure 8.4.

Blur of characteristics with flicker results from using compressed footages. As a result of changes in luminance, used MPEG-2 codec can't find corresponding macro-blocks in neighboring frames. A lossy compression is made for each frame individually. This causes some floating of values.

In Figure 8.5 there is an example obtained from a real movie. On the basis of many such tests it can be concluded that the flicker is characterized by a nonlinearity.

Unfortunately not only the flicker affect the shape of joint histogram. We assume that the flicker is independent of movie conten. We can say that the resulting histogram is a composition of more factors (the order will affect the appearance of histogram). To better understand the problem a few examples have been presented.

Local movements cause the dispersion of values - Figure 8.6. It is interesting that the histogram values generally remain within a certain range. Local movements cause the displacement of pixels and not creation of new ones. From the standpoint of flicker detection is worth noting that the local motions don't change the nature of the main part of joint pdf - even despite of their high presence.

A large moving object generates a horizontal or vertical lines - Figure 8.7. Maxima of that new line are on the same ordinates (abscissas) as maxima of

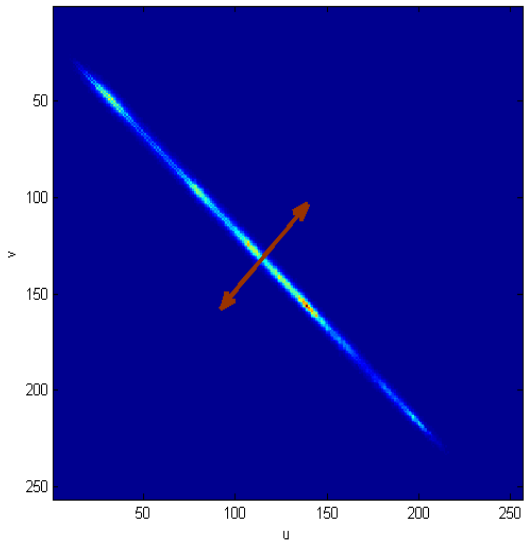


Figure 8.3: Joint histogram of two frames from MovImgClFL with linear flicker

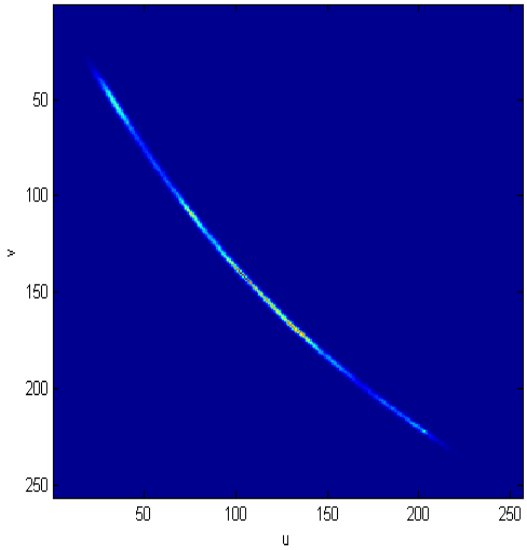


Figure 8.4: Joint histogram of two frames from MovImgClFL with nonlinear flicker

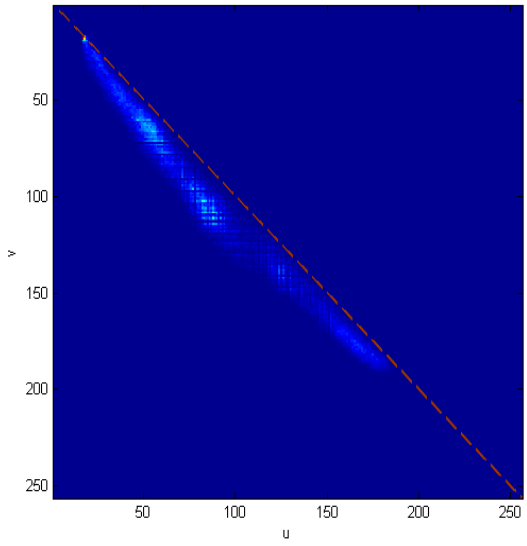


Figure 8.5: Joint histogram of two frames from MovRealClFL

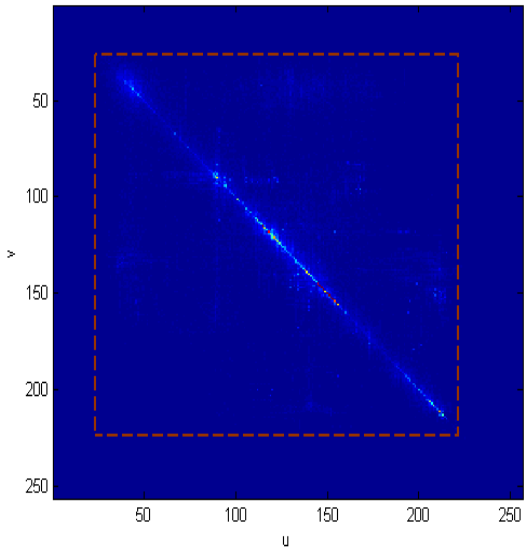


Figure 8.6: Joint histogram of two frames from MovImgObj. Local movements covered more than half of the frame. The speed of movements was 25 pixels per frame. The distance of the compared frames was 7

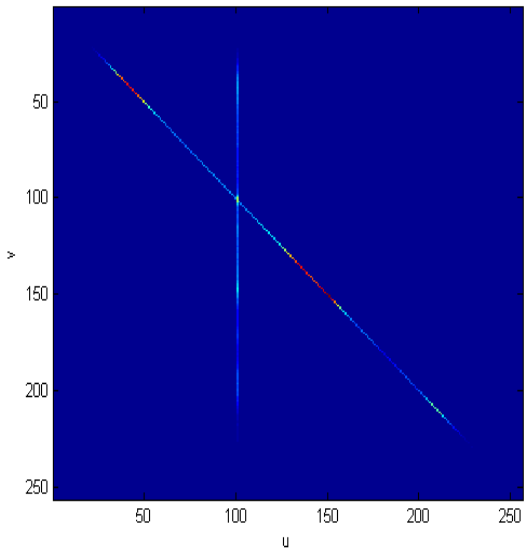


Figure 8.7: Joint histogram of two frames from MovImgHuOb. The speed of movement was 20 pixels per frame. The distance of the compared frames was 4

characteristic without the large moving object.
Global movemets have big impact on joint pdf. In Figure 8.8 we can see the formation of blurred lines - horizontal and vertical. This is due to large areas of the movie content that during a global motion either take a place of the other pixels, or their location is replaced. It may also occur that large areas will replace other large areas. This may result in changing the position of maxima - as in Figure 8.9.
Joint pdf during zoom has tendency to asymmetry relatively to the diagonal - Figure 8.10. Central segments occupy positions of others not being pushed out from own.
Blotches generate vertical and horizontal thin lines - Figure 8.11.

The information that we ought to read from these histograms is brightnes distortion function f between two frames. In case of presence of only flicker it should overlaps joint pdf - for example in Figures 8.3, 8.4 and 8.5. In the case of presence of other factors but without the flicker the function should show a diagonal in joint pdf, that is $v = f(u) = u$. In the case of composition of flicker with the other factors the f should show only the characteristic responsible for flicker. The example in Figure 8.12 is to show that it won't be always an easy task. It is hard to say if there is or not the flicker.

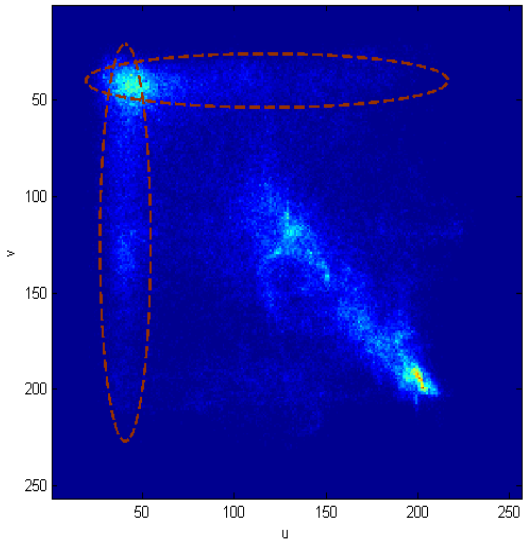


Figure 8.8: Joint histogram of frames 19 and 17 from MovImgPan. The speed of camera pan was 15 pixels per frame

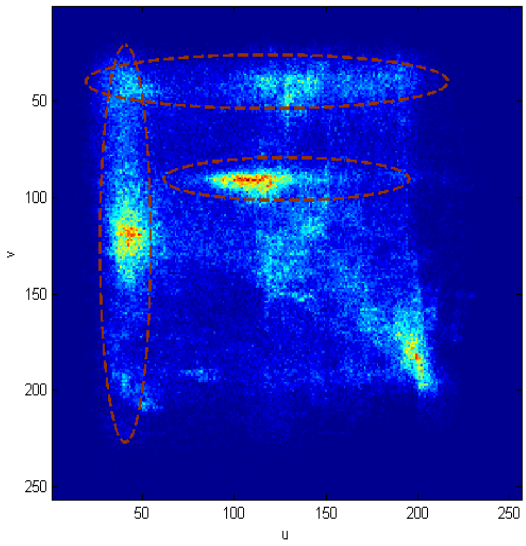


Figure 8.9: Joint histogram of frames 19 and 12 from MovImgPan. The speed of camera pan was 15 pixels per frame

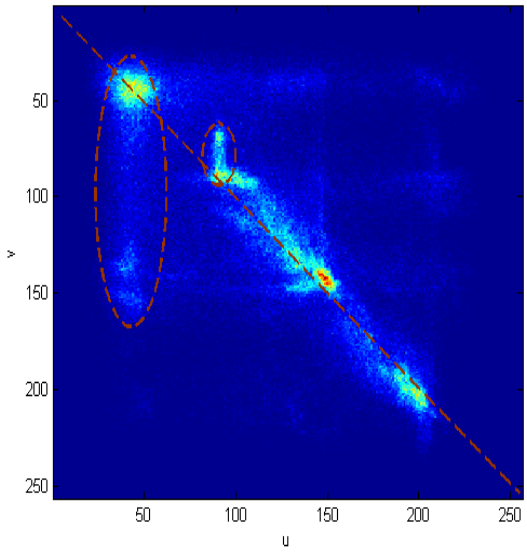


Figure 8.10: Joint histogram of two frames from MovImgZoom. Zoo0.03

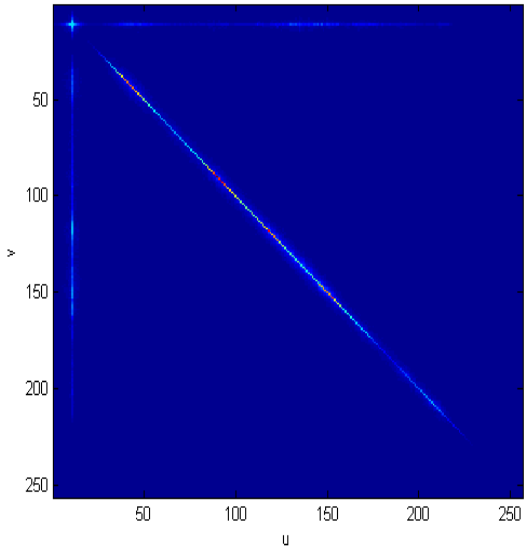


Figure 8.11: Joint histogram of two frames from MovImgBlo. On average, 5% of the area of each frame was covered with distortion

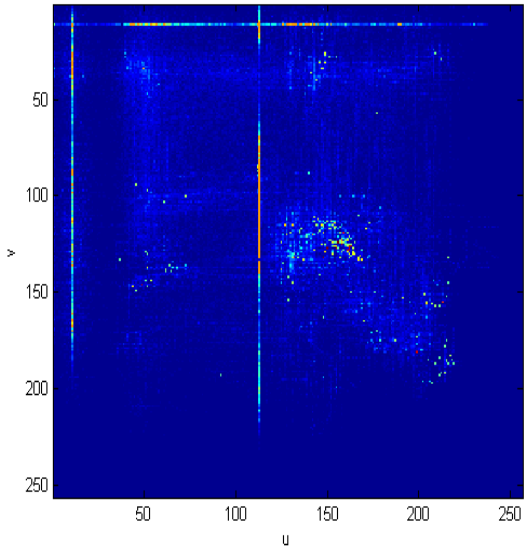


Figure 8.12: Joint histogram of two frames from MovImgAll

Chapter 9

Maximum A Posteriori

In the article [Pit04] the authors propose to use MAP to determine function f from joint pdf - described by 9.1.

$$\hat{f} = \arg\max_{f} \prod_{u=1}^{255} \mathcal{P}(u, f(u)|f) \cdot \mathcal{P}(f(u)|f(u-1)) \tag{9.1}$$

As it was written in the article the goal is to estimate "f that maximises of the number of couples (u, v) that follow the flicker model". The authors gives some more conditions that represent the nature of f in reality:

$$f(0) = 0 \tag{9.2}$$
$$f(255) = 255 \tag{9.3}$$
$$f(u+1) \geq f(u) \tag{9.4}$$

The equation 9.1 can be solved with a force method. However, for performance reasons it would not be a practical approach. The authors suggests to use Viterbi algorithm.

9.1 Viterbi

Presented discussion corresponds to a guidance provided in book [Pol57].

9.1.1 UNDERSTANDING THE PROBLEM

The input data is a joint histogram of two luminances. In practice, this is a matrix 256x256 indexed with u and v from 0 to 255. Each coordinate corresponds to the grey levels for each of two input frames. The unknow are two discrete functions: $v = f[u]$ and $u = g[v]$ in domains $[0, 255]$ where

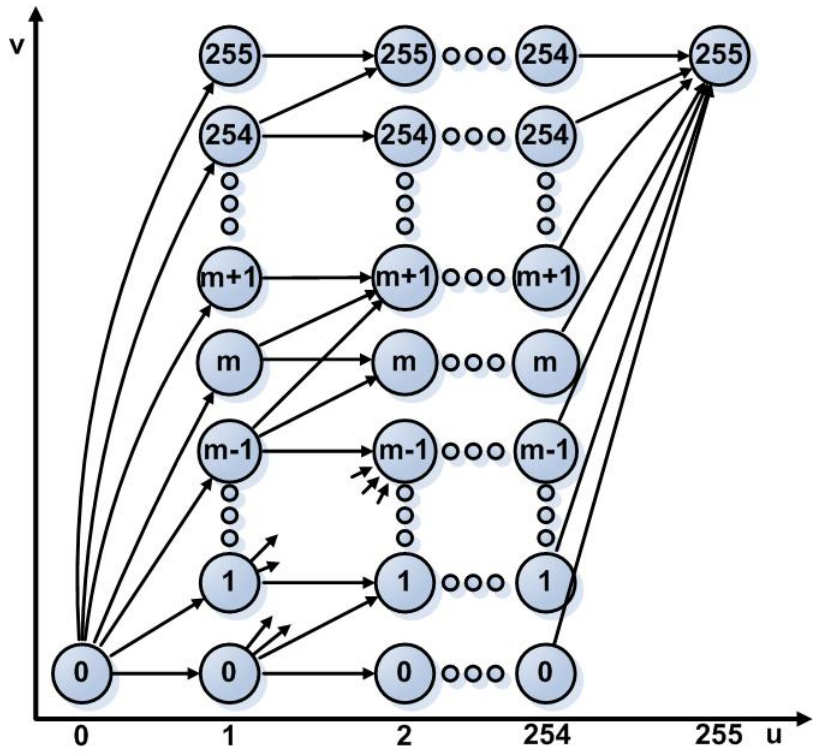


Figure 9.1: MAP - The problem stated graphically

f is an inverse function to g. So we look for one line passing through the matrix. While functions are defined in the whole domain the results can be presented as two vectors of 256 elements. The equation 9.1 includes only one function. But we might as well change the coordinates and the result present the same line. The condition is written in the equation. We look for the function (line) for which the following expression is maximal. The probabilities $\mathcal{P}(u, v)$ are the values in the matrix - values of joint historgram. The probabilities $\mathcal{P}(u, v = f(u)|f)$ are only this values in the matrix through which the function (line) f pass. The probabilities $\mathcal{P}(f(u)|f(u-1))$ try to answer to the question: What is the chance that function will pass through point (u, v) if we know that it has been already passing by point $(u-1, f(u-1))$. So the whole expression is really a product of the values in matrix through which the function goes and multiplied by local likelihoods. Still we don't have enough information about$(\mathcal{P}(f(u)|f(u-1)))$. Also there are some more condition added. The function f shouldn't be decreasing and the boundary conditions are 9.2 and 9.3.

The problem is presented in the graphical way in Figure 9.1. The aim is to draw a line, that is, give a value v for each u. In restored picture we

should have a possibility to obtain each value, so black and white shouldn't be changed. For $u = 0$ the value of v is 0. For $u = 1$ the value of v could be any. We should remember that the function can't be decreasing. So if for $v = f(u = 1) = 5$ for $u = 2$ the v could be only $5, 6, 7, ...255$. The final value is arbitrarily 255. Now we can ask about transitions. What is the probality $\mathcal{P}(f(u)|f(u-1))$? Let's look at point $(u = 0, v = 0)$. What is the likelihood that for $u = 1$ we get $v = 0$ or 1 or 2 and so on. The sum of probabilities should be 1. But we have really no more requirements. Therefore, we can write that

$$\mathcal{P}(f(u)|f(u-1) = \mathcal{P}(f'(u)|f'(u-1)) \quad if \quad f(u-1) = f'(u-1) \tag{9.5}$$

It means that the probabilities of each transition is equal. So for example from any point where $v = 64$ the probabilities are $\frac{1}{256-64} = \frac{1}{192}$. Only for $u = 254$ the probabilities of transitions to point $(255, 255)$ are equal 1.

But what about the likelihood $\mathcal{P}(u, f(u)|f)$? In equation 9.1 it is a weight of normal transition. We can put this value together. The transtion would be $\mathcal{P}(u, f(u)|f) \cdot \mathcal{P}(f(u)|f(u-1))$.

In that way we have designed a discrete Markov process. We can add just a few practical observations. Taking into account the condition 9.5 and also that the process is of memory of only one state, we can skip the expression $\mathcal{P}(f(u)|f(u-1))$. In this case the MAP algorithm is simplified to simple Maximum likelihood (ML). Also, to resolve the equation 9.1 we don't need to do any normalization. All this simplifies the design to the chain as in Figure 9.1 but with the transition equal to corresponding values in the joint histogram (input matrix). The task is to find such a crossing, for which a product of these values will be the biggest.

At this point there is one problem. Not everywhere the joint histogram has values. Often there are zeros! What does this problem mean we will see in an example. In Figure 9.2 there is an sample of typic joint histogram. The values are bigger if the fulfill color is darker. White points mean that the values are 0. If we left the assumption like there are, the line yellow will be chosen even if the values on the path are very small. Worse case would be if there was a vertical line without any value. Everything because of multiplication. We must introduce one more assumption. When we multiply we do not take into account zero values. In that case is likely that the path chosen in the example would be blue. Missing points between the selected points should be chosen by interpolation.

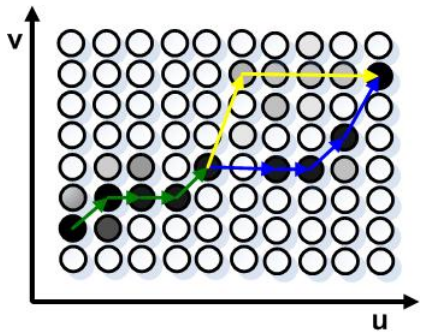


Figure 9.2: Sample fragment of a joint histogram

9.1.2 DEVISING A PLAN

The problem was presented in the form of a Markov process. The problem belongs to the optimization category (looking for the optimal path). Typical solutions that come to mind are genetic algorithms or dynamic algorithms as for example Viterbi. The mentioned algorithm is widely used and above all it gives the exact answer. In appendix B there is an example of "Bridge traversal" problem. The solution is also found thanks to Viterbi algorithm. This example should be well understood. In Figure B.3 we see a similar design to ours. By analogy our problem can be solved in the same way. In the example they look for the least sum and we look for the maximal product. The intermediate values in the example are the earliest arrivals. We would have "tha maximal product for now". We can call it here "weights". We will need to indroduce in implementation one temporal structure of data. Looking at the example as well as Figure 9.1 we can deduce that it will be one matrix of size 255x256. Responsibility for the issue of "zeros" is left to the implementation part.

9.2 Results

The algorithm detects flicker very well - Figure 9.3. It is also extremely resistant to the local motions and blotches - Figure 9.4. Unfortunately, it fails in the cases where in joint histogram the values create an additional clear line - Figure 9.5 and 9.6. The reason of that is the assumption that f should pass through maximum number of values.

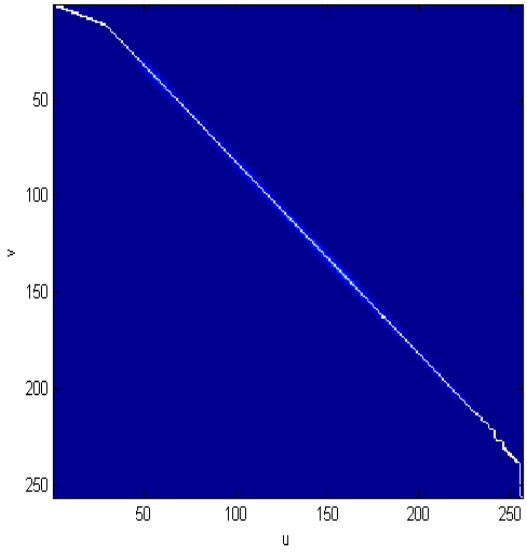


Figure 9.3: Joint histogram of two frames from MovImgClFL and its corresponding estimated distortion function (white)

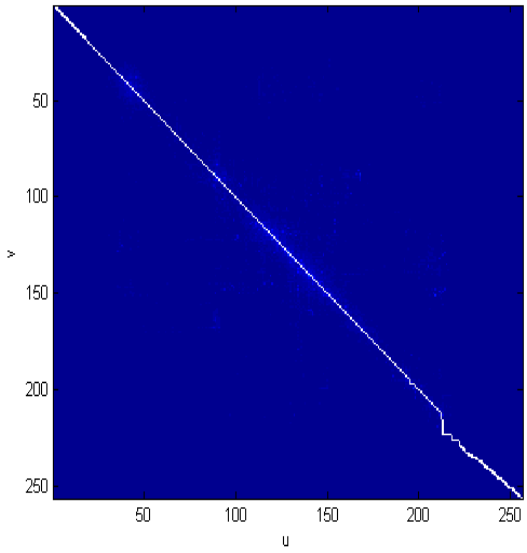


Figure 9.4: Joint histogram of two frames from MovImgObj and its corresponding estimated distortion function (white)

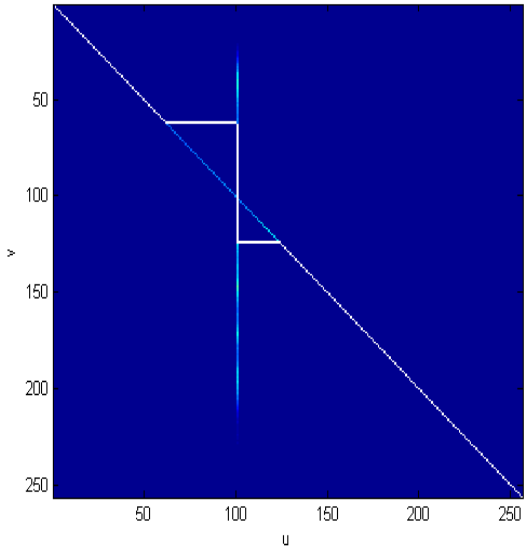


Figure 9.5: Joint histogram of two frames from MovImgHuOb and its corresponding estimated distortion function (white)

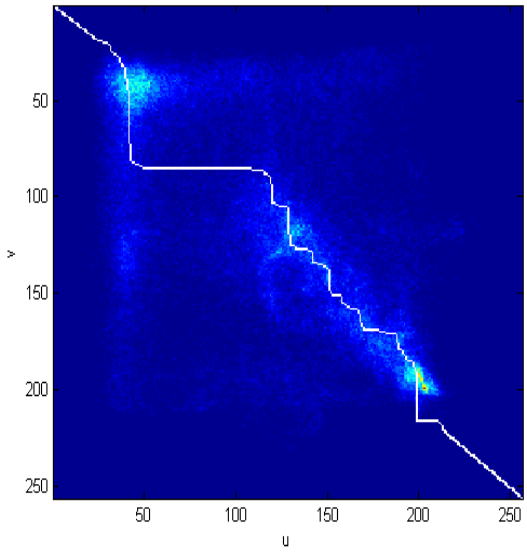


Figure 9.6: Joint histogram of two frames from MovImgPan and its corresponding estimated distortion function (white)

Chapter 10

Inliers Enhancement

Pitié propose to process pdf before using MAP to increase the robustness of f determination. He introduces a weight:

$$\mathcal{R}(u,v) = \mathcal{P}(u|v) = \mathcal{P}(v|u) = \frac{\mathcal{P}(u,v)^2}{\mathcal{P}(u)\mathcal{P}(v)} \tag{10.1}$$

Weight is iteratively applied to pdf as follow:

$$\mathcal{P}(u,v)^{(n+1)} = K^{(n+1)}\mathcal{P}(u,v)^{(n)}\mathcal{R}(u,v)^{(n)} \tag{10.2}$$

where $K^{(n+1)}$ is a normalising constant. Grounds of this mathematical operation are described in detail in the article [Pit04]. Here, let's look at issue from a practical point of view. In the equation 10.1 the bigger are marginal distributions, the smaller is the weight. The big marginal distribution graphically means that there are a lot of values in vertical or horizontal line in joint pdf. Proposed inliers enhancement should reduce the lines in our histograms. Tests confirm this. In Figure 10.1 there is a joint histogram of two frames from footage with camera pan. The resulting function f is characterized by vertical and horizontal line segmants. In resotored film this takes effect in undesirable image segmentation. Using inliers enhancement improve the results (Figure 10.3).

Unfortunately, the method is not without drawbacks. The most annihilated are values on those locations for which two marginal likelihoods are huge - that is in crossings of vertical and horizontal lines. However, it seems that these values can carry the most information about possible flicker. This is illustrated in Figure 10.3 where the maximum values of the original joint histogram have been omitted by f.

The next problem is illustrated in Figure 10.4. The inliers enhancement could give also the greatest importance for the insignificant data. As shown

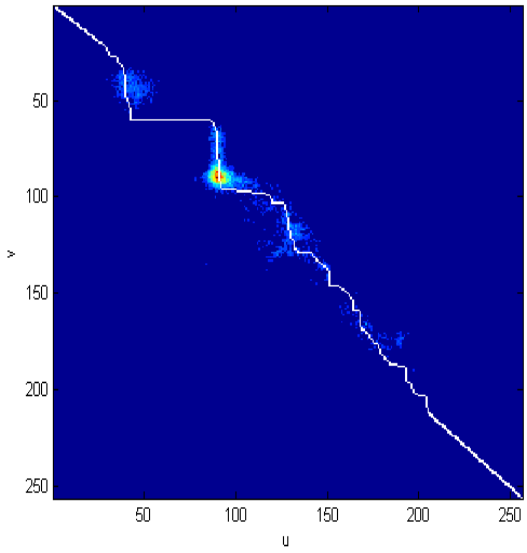


Figure 10.1: Joint histogram with plotted f for two frames from MovImgPan footage

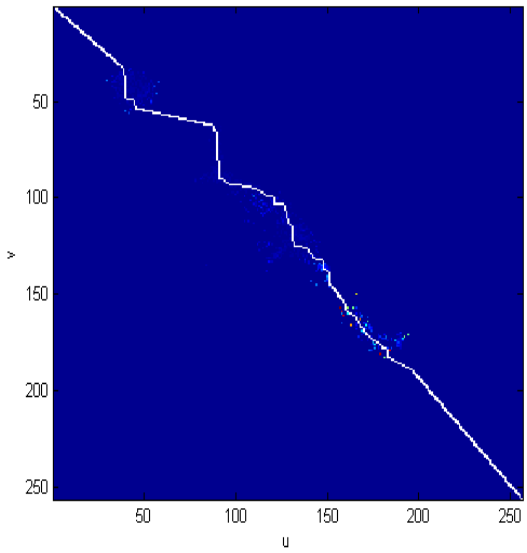


Figure 10.2: Joint histogram with plotted f for two frames from MovImgPan footage after one iteration of inliers enhancement

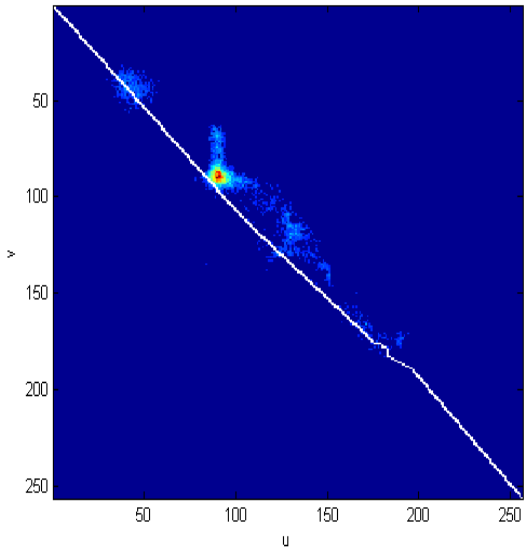


Figure 10.3: Joint histogram for two frames from MovImgPan footage. Plotted f was counted for the joint histogram after two iterations of inliers enhancement

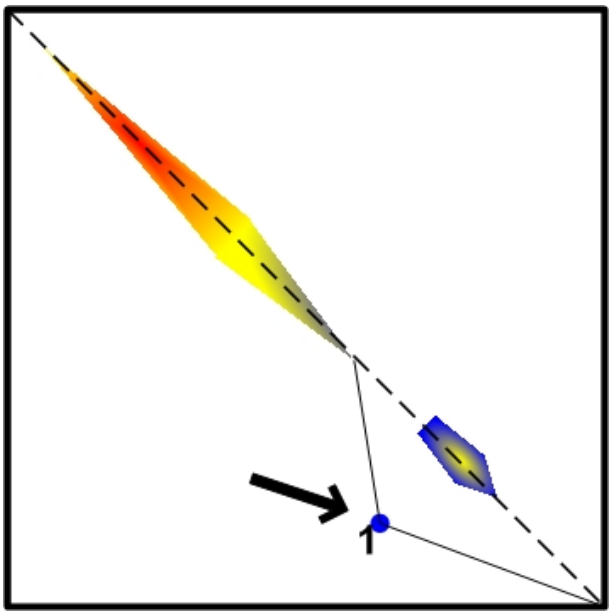


Figure 10.4: Joint histogram with plotted function f without (dotted line) and after using inliers enhancement (solid line)

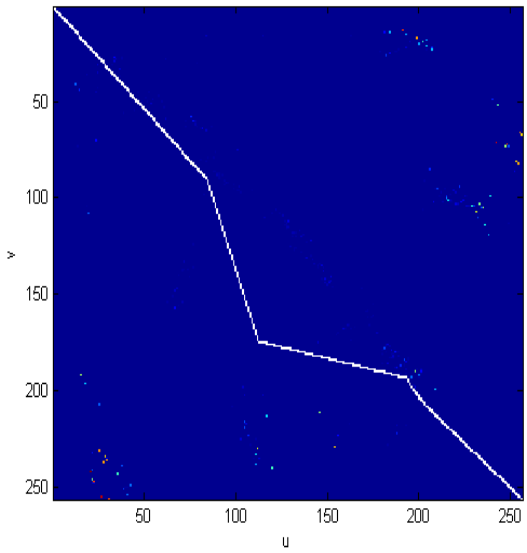


Figure 10.5: Joint histogram with plotted f for two frames from MovImgPan footage after three iteration of inliers enhancement without prior diagnosis

in the Figure, the point of value 1 becomes the most important. This causes sometimes gross errors. A good solution would be to introduce elements of prior diagnosis. Points, which carry negligible information should be cleared. The results of the simulation of the phenomenon is in Figure 10.5.

Chapter 11

Expectation of f-s and Frame Restoration

We are able to extract the information about brightness changes between frames of the movie which are caused by the flicker. As it was in the paragraph 4 of chapter 5.2, we want to reduce these fluctuations. Let's look once again at the Figure 8.1. Function $v = f(u)$ describes how the frame V is changed in relation to the frame U. Let's assume that we want to fix the frame V. Pitié propose to make a restored frame similar to its neighbors. To make V similar to U we should reverse the effects of flicker. If U is followed by V in time, we should to map all values of pixels in V with an inverse function of f which we will note as $g = f^{-1}$. For example, a lot of pixels from U with value 20 only because of flicker changed their values to 25 in frame V. What we should do to restore V is to change the values of all pixels 25 to value 20. However, if V is followed by U, the function f will be a recipe.

Frame V hasn't only one neighbor. They are many - both preceding and following. There is a need of averaging functions f in order to make the frame V in some part similar to many of its neighbors. It has been already presented in Figure 5.7. The averaging method has been also presented in equation 5.35. The robustness hasn't been tested yet. A simplified version has been checked:

$$\hat{f}_n(k) = \frac{1}{K} \sum_{m=n-M}^{n+M} e^{-\frac{(m-n)^2}{\sigma_w}} f_{n,m} \tag{11.1}$$

It is a weighted average (sum of weights normalized to 1). Where σ_w is a temporal scale factor. The example of weights is in Figure 11.1. The shape was proposed by Pitié. We should make the restored frame more similar to its further neighbors - the author argues. It should be noted that the f characterisctcs are not signals. They are different for each frame. Thus the

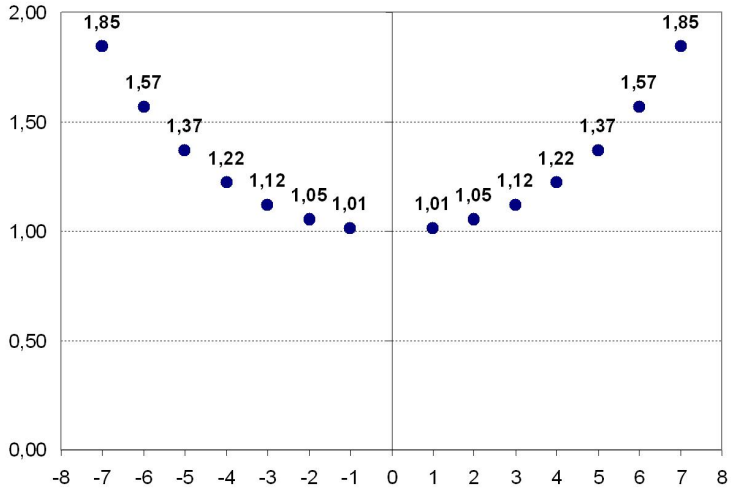


Figure 11.1: Weighted average example - not normalized. $\sigma = 80$

.

weighted average can't be treated as a filter.

The statement from the article [Pit04] that the best results can be achived when the frame is compared with 14 neighboring frames was confirmed. But it is still an empirical finding.

11.1 Symmetrical Averaging

One of the open question is: Shouldn't we change the coordinate system in the averaging? This issue applies only to non-linear flicker. It was presented in Figure 11.2. Many experiences show that f is rather symmetrical in relation to the diagonal. We can come also to such conclusions theoretically starting for example from the H-D curve described on page 28. The issue hasn't been settled yet. However, its impact on the results is incomparably smaller than the factors discussed earlier. It does not get a high priority.

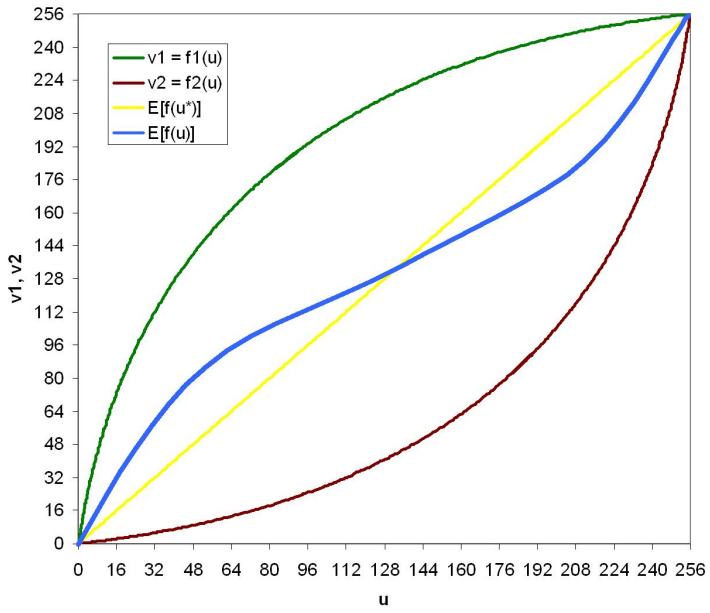


Figure 11.2: Two f between frame u and its neighbors. Blue line is an arithmetical average in relation to u coordinate; yellow in relation to v = u

.

Chapter 12

Local Flicker

In the article the authors convince that the flicker can have a local character. They take the equation 5.33 to describe the relation between frames. The issue has been already generally presented in paragraph 4 of the chapter 5.2. Appropriate tests havn't been carried out yet. Nevertheless, the idea is the same as for global flicker. The calculations relate to smaller areas - parts of frame. Therefore, we can expect that the use cases presented in the chapter 6 will have greater impacts on results. But still they will be of the same nature. All considerations from the previous chapters are still applicable.

In the case of removing a local flicker the local joint pdf is given by:

$$p^i_{n,m}[u,v] = \frac{\sum\limits_{\mathbf{x}|(I_n(\mathbf{x})=I_m(\mathbf{x}))} w(\mathbf{x}-\mathbf{x_i})}{\sum\limits_{\mathbf{x}} w(\mathbf{x}-\mathbf{x_i})} \tag{12.1}$$

Where $w(\mathbf{x})$ is the spatial weighting function. As it was said in the State of the Art, $\mathbf{x_i}$ are the positions of regularly spaced control-point pixels. The operation is shown graphically in Figure 12.1. All next operations on the histograms are carried out in an identical manner to those described for global flicker. The last activity performed should be the restoration combined with interpolation of $f$on the surface of frame:

$$I^R{}_n(\mathbf{x}) = \sum_i^N s(\mathbf{x}-\mathbf{x_i})\hat{f}^i_n(I_n(\mathbf{x})) \tag{12.2}$$

The splines of order 3 were proposed as an interpolation method.

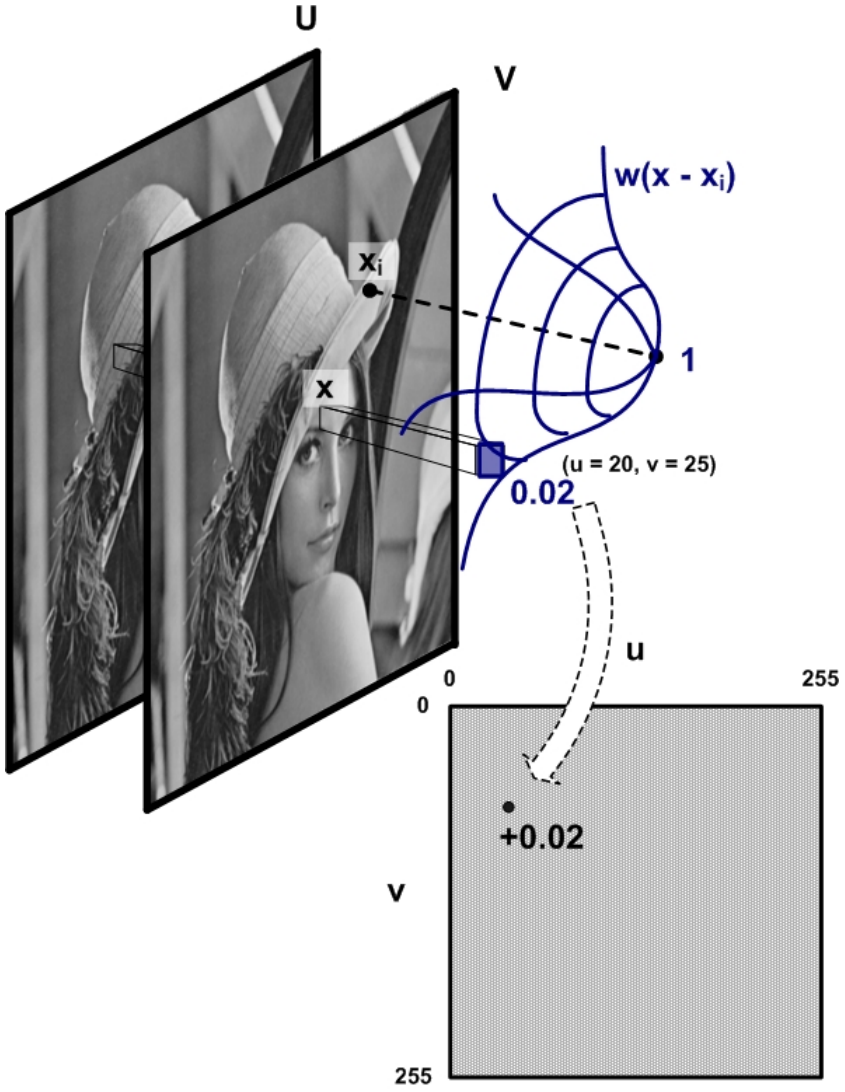


Figure 12.1: A way for calculating the local joint histogram of two images

Chapter 13

Improvements proposal

The algorithm presented by Pitié in a very coherent way solve the flicker problem. However the ideal wasn't achieved. Moreover, its full implementation requires many calculations and restoration time per frame can be too long. It empowers us to seek some improvements also here in design part.

13.1 f - sum instead of multiplication

In the chapter 9.2 we have seen some pitfalls in which MAP can be trapped. They result from the assumptions. We should ask, where is exactly the util information. When the movie content is not very complicated - the scenes are static and there are not a lot of other distortion - the information about the brightness ditortion can be found on a line with the most values.

But the truth information about the flicker is not in the multiplicity of values. The use case MovImgClFL is the best example. In that case, we should have more or less 256 values. If the number of values is significantly higher it means that we have additional distorting information. Laying on the straight lines is a common phenomenon (chapter 8). However, the maximum values of joint pdf are still talking a lot about the original brightess distortion.

In place of MAP or its simplified version ML (implemented) a new method is proposed to estimate f:

$$\hat{f} = \arg\max_{f} \sum_{u=0}^{255} \mathcal{P}(u, f(u)|f) \tag{13.1}$$

This should ensure a better balance between the number and size of values through which f pass. An example is shown in the Figure 13.1. In the case of global frame treatment, each pair of pixels during the determination of joint histogram gives a information which is "1" in a particular place. There

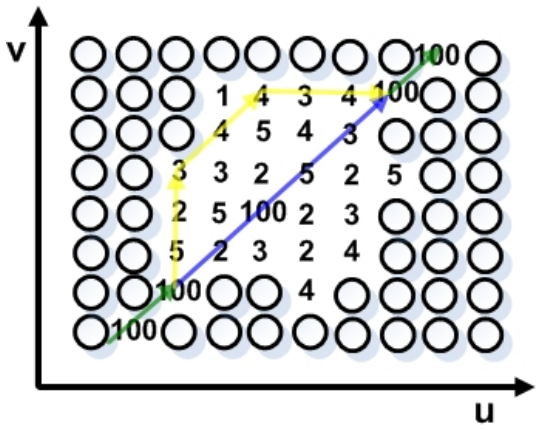


Figure 13.1: Comparison of behavior of two algorithms in a typical situation: MAP (yellow) and proposed (blue)

is no reason to so significantly increase the importance of this information when it is between the others of large dispersion - how it is done by MAP. The assumption described by 13.1 says that f should pass through a lot of **big** values. After applying this method probably there won't be a need to use Inliers Enhancement (chapter 10).

The task can be also solved by Viterbi algorithm. The implementation should be simpler than that made for MAP.

13.2 f - more assumptions

When we compare the characteristics from Figures 9.6 and 9.3, we can say that the first one can't really represent the flicker. We could give more assumptions.

Only ascending function

With expression 9.4 we said that f should be a non-decreasing function. The result is that there is the possibility that the characteristic would have long vertical or horizontal line segments - as in Figure 9.6. Rather, it is not a typical for flicker. Using such a characteristic in the restoration process causes the loss of local contrast - it is an image segmentation. Originally in the article there is the following expression instead of 9.4:

$$f(u+1) > f(u) \tag{13.2}$$

However, it has been recognized as an error. Since f was a discret function which is definite in its entire domain, $v = f(u)$ would be the

only possible characteristic.

One possibility to realize 13.2 would be an upsampling of joint histogram. Another method (approximate) could be for example adoption of the following assumptions:

$$f(u+1) = f(u) + i \tag{13.3}$$

Where:

$$i \in \{1,2\} \quad if\ f(u) = f(u-1) \tag{13.4}$$
$$i \in \{0,1,2\} \quad if\ f(u) \neq f(u-1) \tag{13.5}$$

The Markov process used to solve the issue would be of order 2.

Not crossing the line u = v

The characteristics of flicker shouldn't cross the line $u = v$. The statement is based on many observations made on real footages with flicker.

13.3 Global Motion Estimator

The presented algorithm of Inliers Enhancement significantly improves the quality of the restoration even of the footages with huge global motion. However, it must be emphasized that the quality depends mainly on the cross-correlation between neighboring frames. The more details are in the content, the worse is restoration. Worse restoration above all means a generation of new artificial flicker.

The obvious solution would be to use a global motion estimator. Thanks to that we could have an information in the vector form about the shifts between frames. This information should be taken into account during a joint-histogram calculation. From a practical point of view, we don't need to calculate directly the vectors for each pair of frames. The vectors may be calculated relative to each other. Depending on the quality of the estimator we could also estimate some of vectors - all to make entire system more efficient. One of the possible estimators is the Phase Correlator.

Part IV

Implementation

Chapter 14

Introduction

14.1 Review of existing tools

There exist already a few tools for digital video processing. Some are suitable for prototyping but some may also facilitate the creation of targeted solutions. Below, there is a brief overview of some of them in the context of their usefulness for the project.

NMM [1] Network-Integrated Multimedia Middleware. This solution was chosen by Deutsche Thomson OHG for prototyping some of their previous works. It provides a generic multimedia architecture for all kinds of distributed multimedia applications. However NMM is completly not suitable for standalone products like the designed library.

OpenCV [2] The most popular C/C++ computer vision library. This library has over 500 optimized algorithms. It is released under a BSD license. The library could be a support for the flicker removal library.

Intel IPP [3] Integrated Performance Primitives. This solution has been chosen by Deutsche Thomson OHG as a support for whole project of movie restoration. However, it solves only some image-processing problems .

There are also some tools and free code under license GNU GPL. However, they can't apply to a commercial project.

[1] http://www.networkmultimedia.org/
[2] http://opencv.willowgarage.com/wiki/
[3] http://software.intel.com/en-us/intel-ipp/

14.2 Adopted methodology

The implementation methodology is very important. Without correct and done on time implementation it is not possible to do any analysis. Like it was stated in chapter 3.1, the implementation should be also a final product. The project is specific. At the beginning and also during development what we know is only what the library generally should do at the end. There is no trustworthy algorithm so the structure of the library can change diametrically. We must ensure a high flexibility of the developed product.
Algorithms and various types of systems are best presented by a workflow. This is the basis of such methodologies as MBD (Model Based Design). Model updating is simply an exchange of objects and a reconfiguration of connections. To make the implementation equally agile we should bring it closer to the modeling language. It is sure that the programming should be modular and at the possible highest level of abstraction. The language has been imposed - C or C++. The C++ language allows for object-oriented programming. It allows also to create a software faster and safer.

However, object-oriented programming itself does not guarantee the success. The system can easily become stiff and difficult to maintain. Due to the nature of the project we need an agile methodology. Own work style has been created - Figure 14.1. Accordingly to the paradigm from the chapter 3.1 we should develop a product by small steps. The global task should be splitted into smaller one. There is no need to know them all a priori. As a starting point we should adopt the user interface and user requirements. Firstly, we should create basic use cases. Generalization of the user's requirements protect against frequent changes of system. For example if the user says that he wants to operate on data of a specific type we should develop a possibility that he could operate on any type of data. If the system is intended to work on Windows platform we should think about multi-platform system. Experience tells that generalized programming is not at all a big surcharge of work and can save a project. Generalized programming is rather a working philosophy. Each development cycle should start with one demand and should finish with a working improved version of the product.

Application modules should be repesented by a model that is close to the problem domain. Simulink was used as a tool. Programming should be purely object-oriented. Round-trip engineering significantly increases the productivity. UML is now the standard in software development. Automatic code generation saves time of code and file management. Coding is only adding a functionality. Bouml[4] has been selected as a tool. It is a free

[4]http://bouml.free.fr/

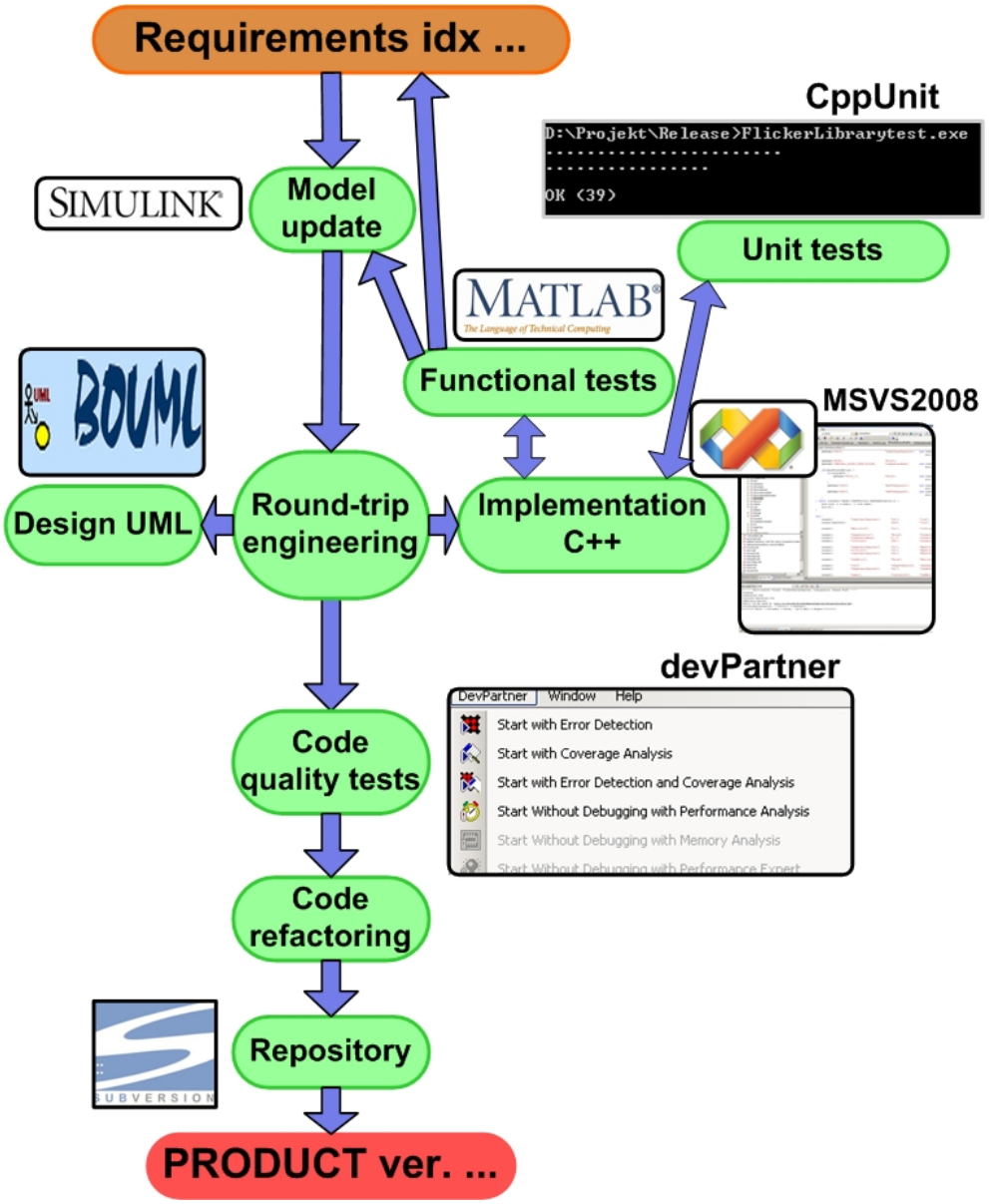


Figure 14.1: Implementation methodology

software that with its functionality surpasses most of commercial applications (e.g. a support for templates in C++). Microsoft Visual Studio 2008 was used for editing, compilation and debugging. Unit tests (here made with CPPUnit) can prevent many software errors. A good practice is to write tests before coding. Mathematical algorithms were tested with Matlab. Relevant data can be generated to the hard drive. Then it can be imported to the Matlab workspace. If something is not physically possible to implement we can change a model or requirements. Code quality tests were performed only once and after whole implementation. However, they should be done after each step. A trial version of devPartner plugin was used. On of the keys to a bug-free implementation is continuous refactoring. Experience shows that the time devoted to it is compensated several times during further development of the product. When new requirements are met we should keep a work done in a repository. SVN was for that adopted.

Chapter 15

Library Package

15.1 Purpose

This package is the result of a continuous code refactoring. The package Library contains a universal engine written in C++ for developing any kind of libraries by providing an universal interface. The Library helps in design of applications that process large amounts of data. Specific libraries can be created as data flow or work flow. This helps to quickly transfer the idea to implementation. The possibility of nesting the architecture helps to organize even very complicated design. It is possible to create a universal collection of basic tasks that increases the chance of re-using a once written code.

15.2 Main design

The principal block is the library class - LibraryMain. Specific libraries should inherit from it.

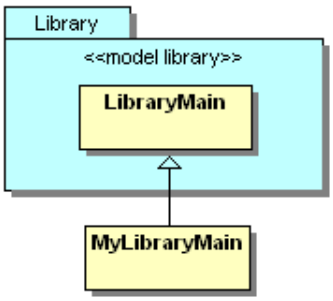


Figure 15.1: LibraryMain

This is a quite trivial approach. However, let's define now requirements. The client's code should be able to send following messages:

- RUN : after which the library starts to carry out its task;
- STOP : after which the library stops working and any its internal state is restored to the default;
- RESET : after which internal state is restored to the default;
- PAUSE : after which internal state is kept but the library stops working, the control of application is completely on the client side;
- RESUME : after which the library resume its work after PAUSE message;
- ... : messages collection should be extensible;

At this stage we don't define a duration of a task performed by the library. It may be also infinite. How then we could reconcile the possible infinite working time with a possibility of control by the client's code? The library, when is working, will be calling from time to time the client's code, from where the client can send an appropriate message. How to make that our library knows how to call the client's code? Let's introduce an interface, of which the client must inherit his class to establish a communication between him and the library (Figure 15.2). Among the methods of the interface *LibraryUI* we can distinguish runUI(), stopUI(), pauseUI(), resumeUI(). They all answer to the aforementioned requirements. Calling them should evoke a desired behavior in the concrete LibraryMain. Also one abstract method *idle()* could be find. The client must define it, and this is the method that will be called by the LibararyMain from time to time. From this method the messages STOP, PAUSE and RESET can be send. This solution is known as *Template Method* [RJ94]. The concept is shown in a sequence diagram in Figure 15.3.

The method idleUI() will be furhter named a SLEEP message. The creator of the particular library should send it reasonably frequent. As a good programming technique, the client shouldn't implement a long idleUI() function. It should examine the relevant conditions and if necessary send a message. If there is a need to execute other operations, the client can simply send a PAUSE message. An example could be seen.

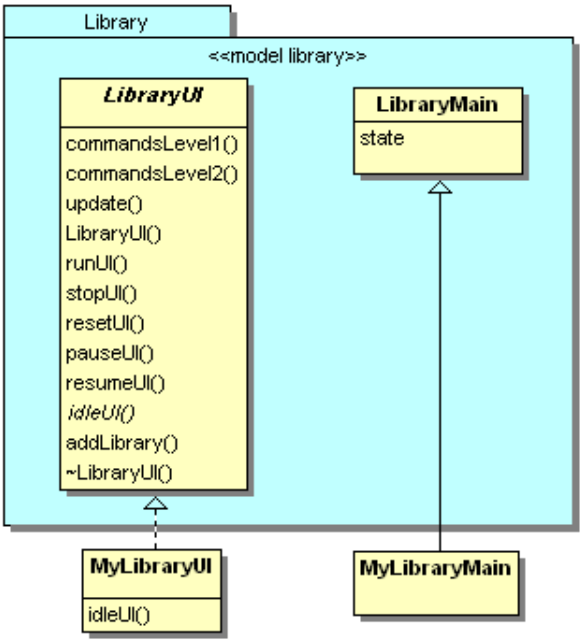


Figure 15.2: LibraryMain with LibraryUI

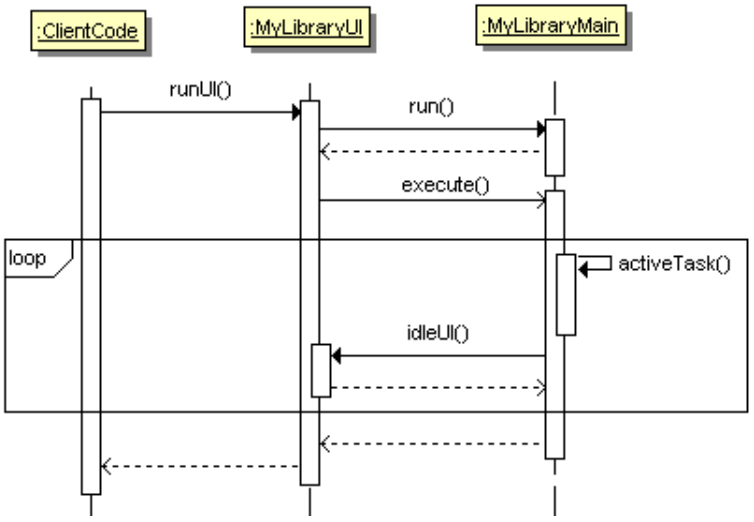


Figure 15.3: Template Method - basic idea of the library

```
#include <conio.h>

void MyLibraryUI::idleUI() {

        char a;

        if(_kbhit()) {
                a = _getch();
                if( a == 's') stopUI();
                else if( a == 'p') pauseUI();
                else if( a == 'r') resetUI();
        }
}
```

In the example the library will be controlled using a keyboard. RUN and RESUME messages in a typical solution will be called from outside on the instantiated interface.

One thing that we shouldn't forget is to establish a connection between the library and the interface. Let's define two methods: addLibrary() for the LibraryUI and addUserInterface() for LibraryMain. We can now demonstrate how to use any library based on the Library Package. Objects can also be created on the heap.

```
#include "MyLibraryMain.h"
#include "MyLibraryUI.h"
.
.
.
MyLibraryUI myInfc;

MyLibraryMain myLib;

myLib.addUserInterface(myInfc);
// or   :
// myInfc.addLibrary(myLib);

myInfc.runUI();
```

In connection with transmitted messages, the library may pass through states showed in Figure 15.4.

We can ask also whether more interfaces can be attached to one library, or a single interface to several libraries. To these two questions the answer is "yes". Messages are sent to all of the attached items. As an example, the

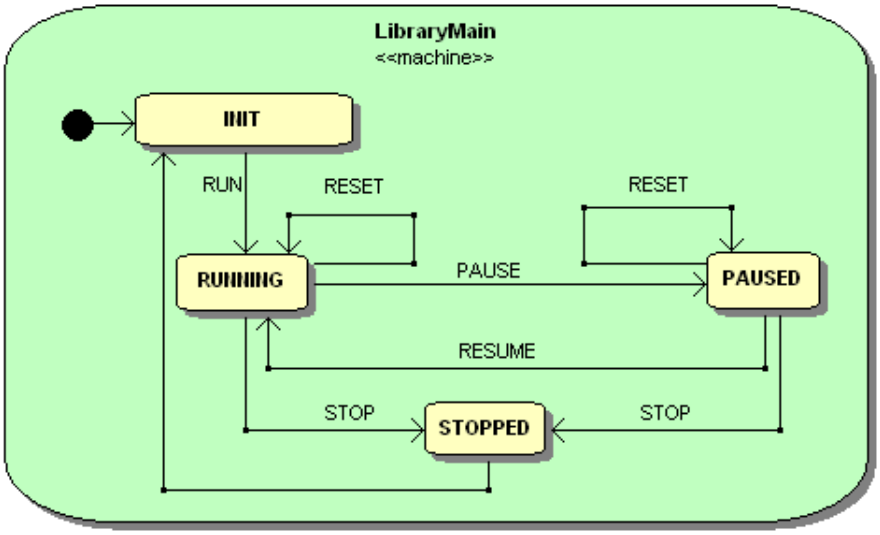


Figure 15.4: The state diagram for LibraryMain

message SLEEP from sent to two connected liraries is presented in Figure 15.5. To realize the advantages of such a solution, let's look at an example of client's software application in Figure 15.6. Each library has its own interface supporting its specific messages. However, all libraries have a common interface through which the client can run or stop all the libraries at once. Operations associated with threads, for example, can be operated in functions idle() of each library. Of course, the client can define any application architecture.

15.3 Use cases

We can now bring together the use cases (Figures 15.7, 15.8, 15.9). For example, sending STOP message should terminate the library's work. Also the library may do it itself if the library will fulfill its task. The same concern any message.

15.4 Interface-Library communication

In the full conviction it may be said that the communication between the library and its interfaces is a separate functionality. For this reason it was decided to separate a special block *LibraryInterface* (Figure 15.10). Question that we have to ask early is: *How can we multiplicate the number of messages*

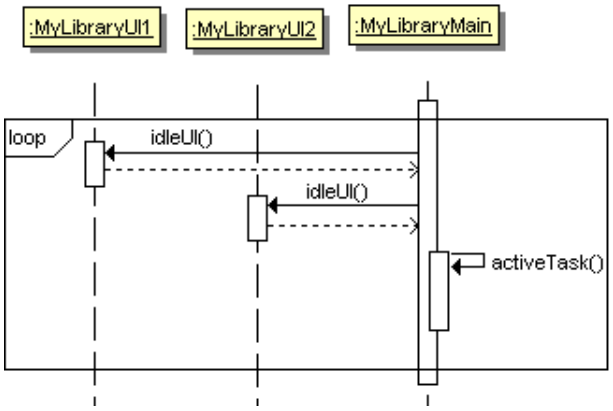


Figure 15.5: SLEEP message for the library with two interfaces

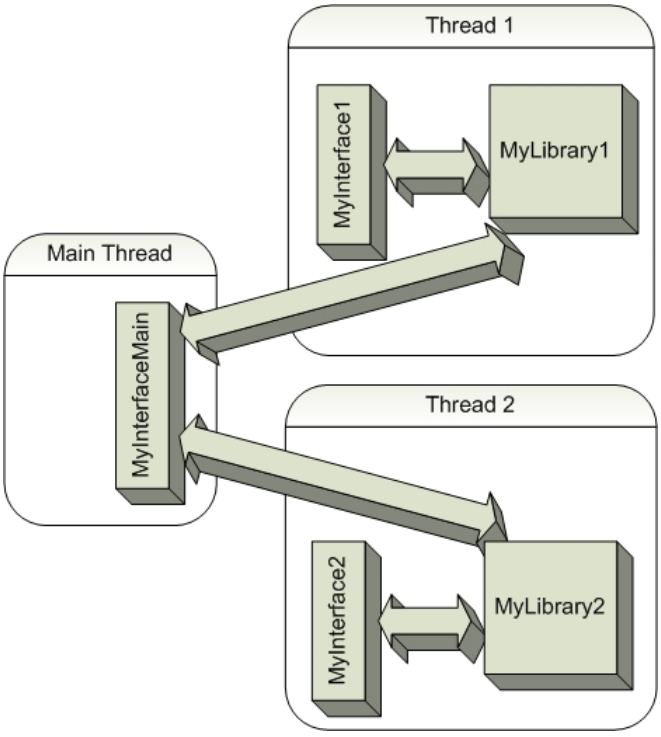


Figure 15.6: Example of client's software application using libraries from Library Package

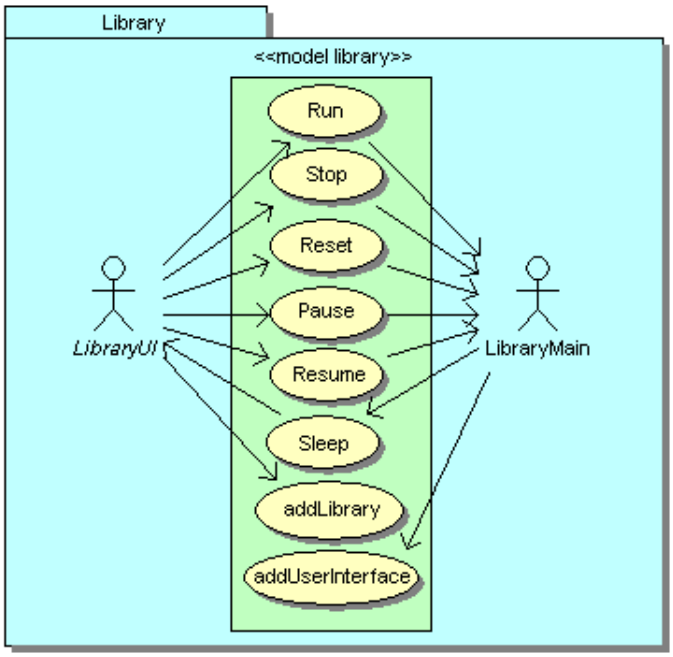


Figure 15.7: Use cases using the model library from Library Package

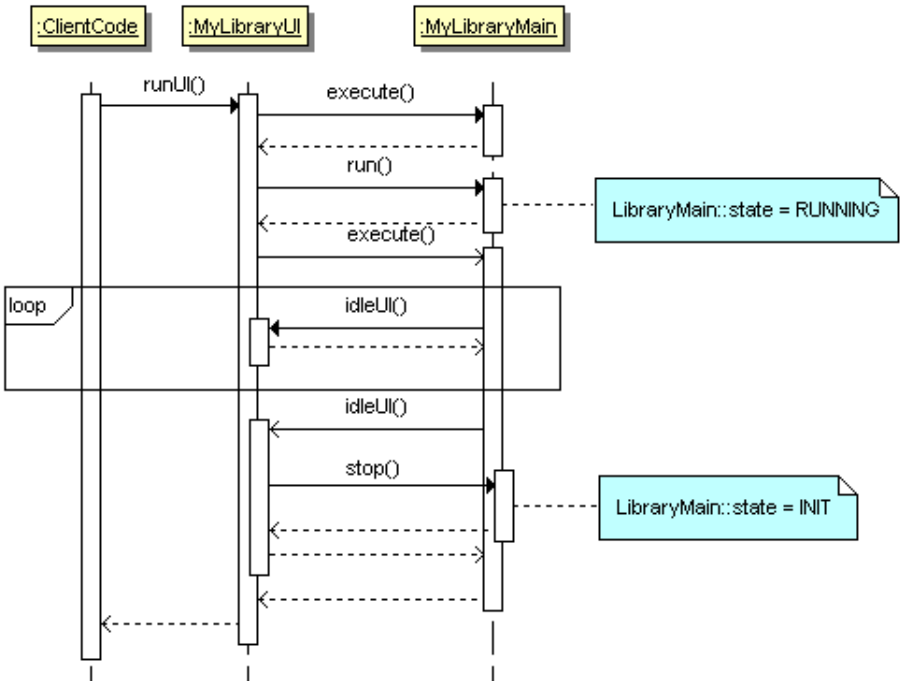


Figure 15.8: Use case - Stop

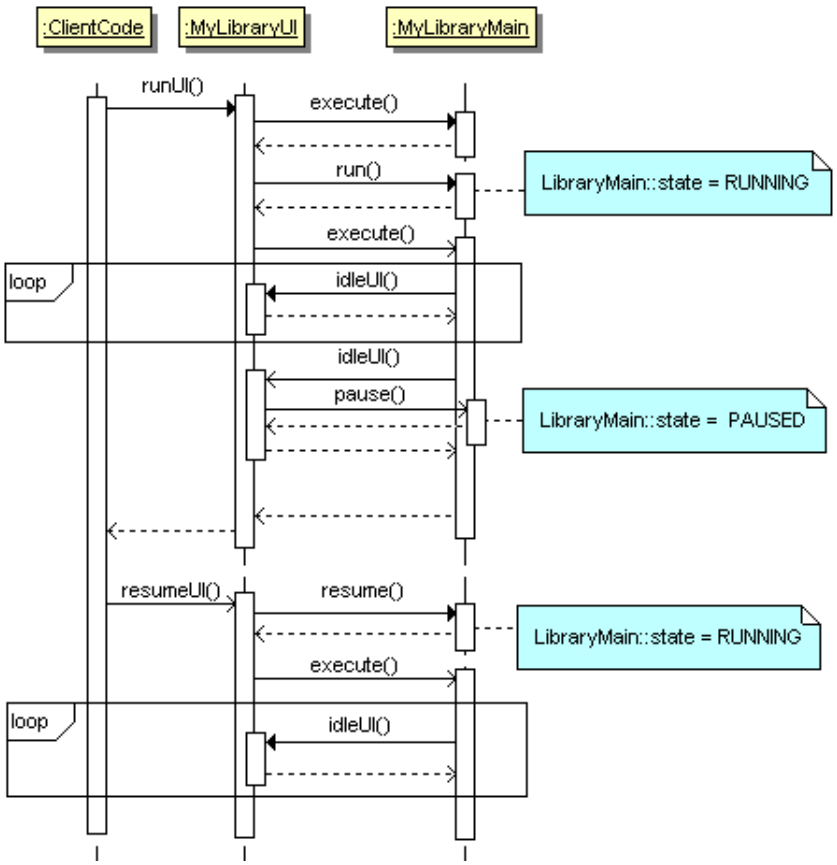


Figure 15.9: Use cases - Pause and Resume

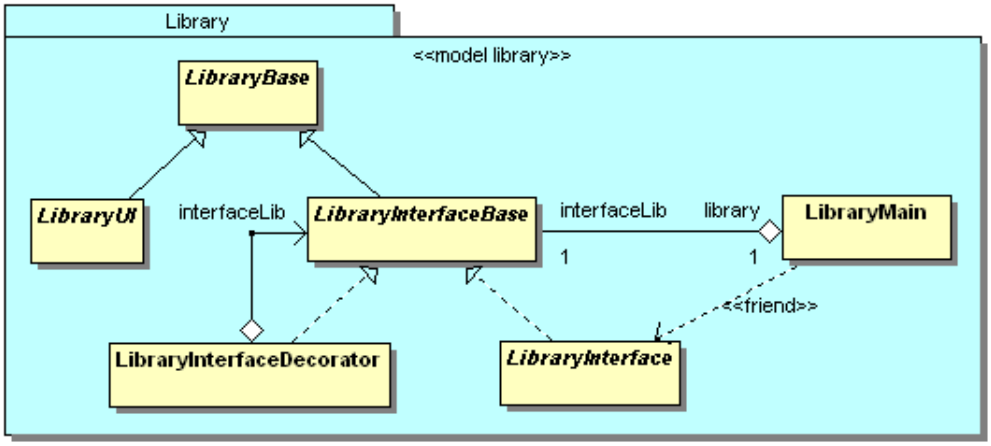


Figure 15.10: Interface-Library communication I

and thus expand the functionality of the library?. On the part of *LibraryUI* it should be done by an ordinary inheritance. Main reason is that the final client of the library must define for example function idle(). However, we have more freedom on the library side. Adopting the principle that the composition is better than the inheritance, *Decorator Pattern* was used [RJ94] (Figure 15.10). Both *LibraryUI* and *LibraryInterface* are derived from the same interface *LibraryBase*.

The relation between the interface and the library shouldn't be too close. We don't know the number of connections. In addition, the message is more important than information about its sender. If we would like to know the sender, appropriate information could be sent as a message attachment. This reasoning leads to the use of the *Observer Pattern* [RJ94] (Figure 15.11). The actual implementation of the pattern is taken from the book *Thinking in C++* of B. Eckel [BE03]. The pattern was covered by the Listener and Notifier classes. This ensures an insensitivity of the system to changes in pattern implementation. *LibraryBase* may have many Listener-s and Notifier-s. All they are mapped. They can be treated as channels through which messages flow. To do is to properly connect these channels. Class Mediator can facilitate this. Evoking addLibrary() or addUserInterface(), the Mediator is linking in pairs Notifier-s and Listener-s that are mapped to the same key. In the current implementation the Mediator is implemented as a *Singleton* [RJ94]. If this would cause problems in multi-threaded systems, Mediator can be reduced to the object which is owned by the library.

To understand the communication let's trace a sending of one message (PAUSE) in Figure 15.12. The client wants to pause the library (1). He

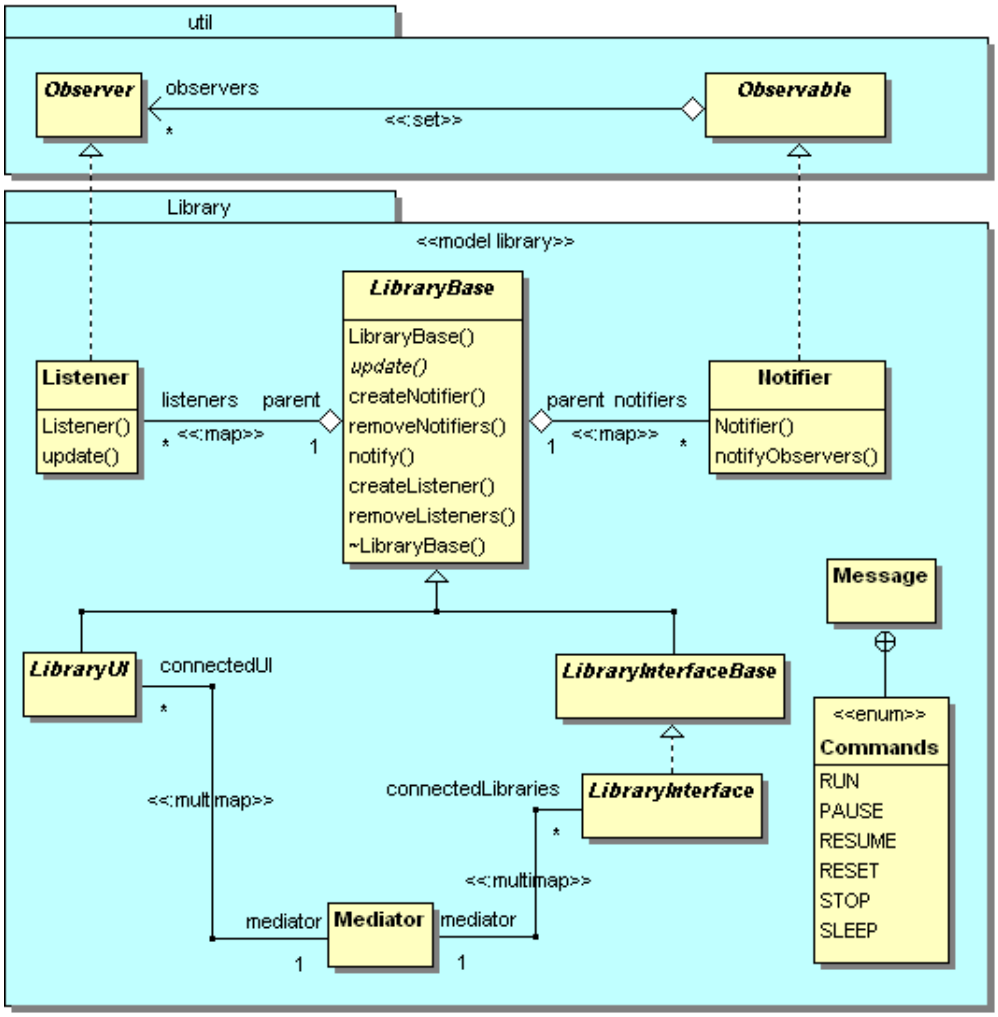


Figure 15.11: Interface-Library communication II

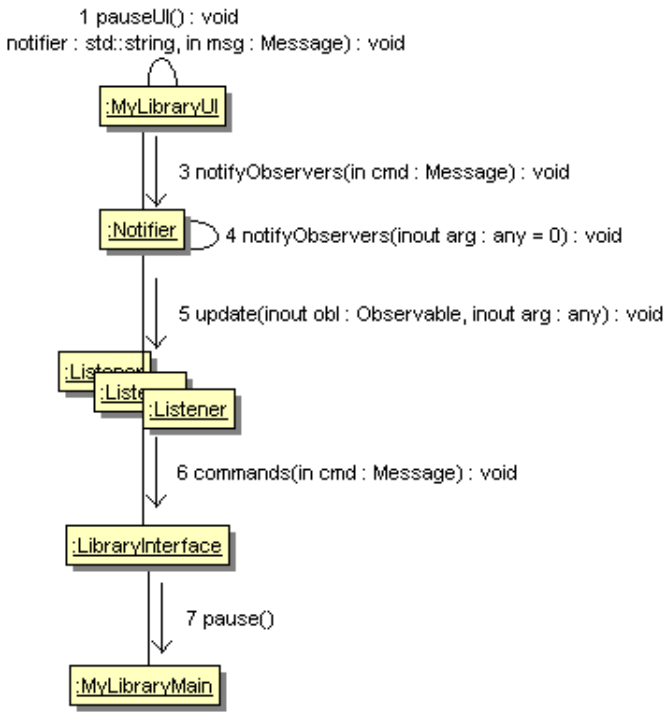


Figure 15.12: One message dispatch - PAUSE

creates the Message (2) and sends it through the appropriate channel (3). This Message is converted into binary form in an Observable's method (4) and sent in order to each registered Listener (5). Each of them restore the Message and pass it to the LibraryInterface that the Listener belongs to (6). LibraryInterface interprets the Message and causes the changing of the library state (7). Possible interface decorators weren't taken into account in this example.

15.5 Task

It is obvious that the library should do something. As shown in the Figure 15.13, the library has one main task. We have also new requirement from the previous paragraph. The library should from time to time send the message SLEEP. If the task is large and complicated, how can we ensure that the application doesn't block? Let's look at the creator of a particular library, as he was also our client. His work is to define the mainTask. It would be dangerous to leave him the responsibility for puting to sleep the library.

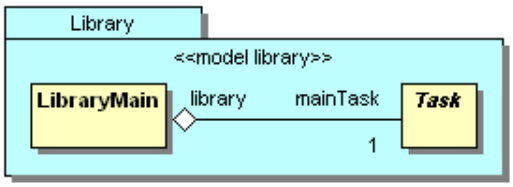


Figure 15.13: Library with its Task

In addition, the client-creator don't need to be necessarily familiar with the software engineering. He may specialize only in algorithms designing. The only thing that we require from him is to split his task into smaller if the whole one would be relatively large. What does it mean - "smaller task"? During execution of a particular task it won't be possible to send messages to the library. So if the reaction time to a message should be maximal $5\mu s$ in any moment, duration of any small task can't be longer. Fastness depends also on the hardware. Thinking about that is left to the library creator. Required approach seems to be natural and induce the creator to make a modular design. How does the library work? Between sending SLEEP messages one small task is performed. It is showed in Figures 15.3 and 15.5.

Task may be simple or complex (Figure 15.14). The SimpleTask is a block which execution is not interrupted. The ComplexTask is a block, which consists of other tasks. The ComplexTask doesn't perform any operations of the client-creator. The best implementation would be be here to use the *Composit Pattern* [RJ94]. Tasks know the ComplexTask to which they belong. The mainTask is Only one Task without a "parent".

Client-creator should inherit from these two classes his own specific tasks. For his SimpleTask-s the client-creator should define a function run(). This code will be executed at runtime of the library. There are no restrictions on the code behind the previously mentioned (reaction time). For his ComplexTask-s more important will be method configure(), where the client-creator defines a Task-s structure. The ComplexTask has an access to the TaskFactory which will be creating desired blocks (Figure 15.15). There is only one factory for the entire library and the client-creator instantiate it when he instantiate the mainTask. The TaskFactory is a design pattern named *Factory Method* [RJ94]. Made implementation is based on the idea from the book *Thinking in C++ vol.2* [BE03]. It was combined also with the *Decorator Pattern* to enable to expand it. TaskFactory is continuously extended with new universal blocks. If the client-creator tries to create a block, which is not a part of the TaskFactory or any attached decorators, he will be informed by the exception BadTaskCreation.

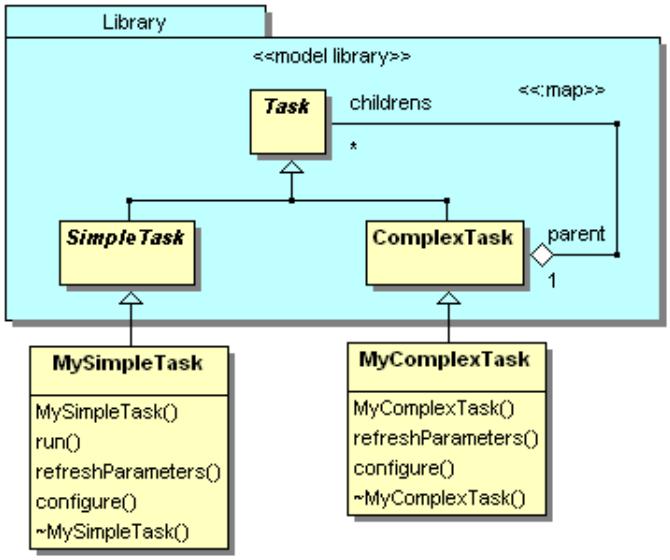


Figure 15.14: Task

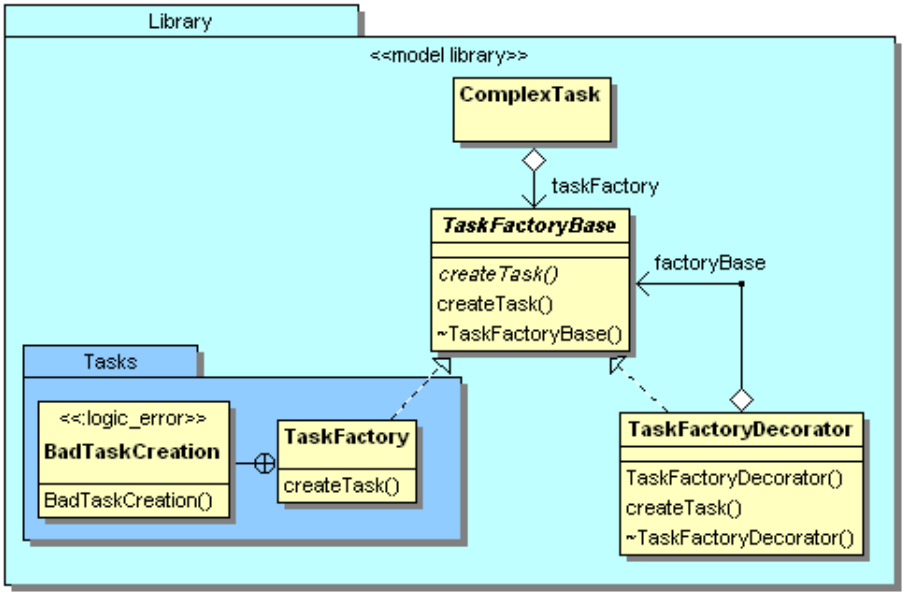


Figure 15.15: TaskFactory

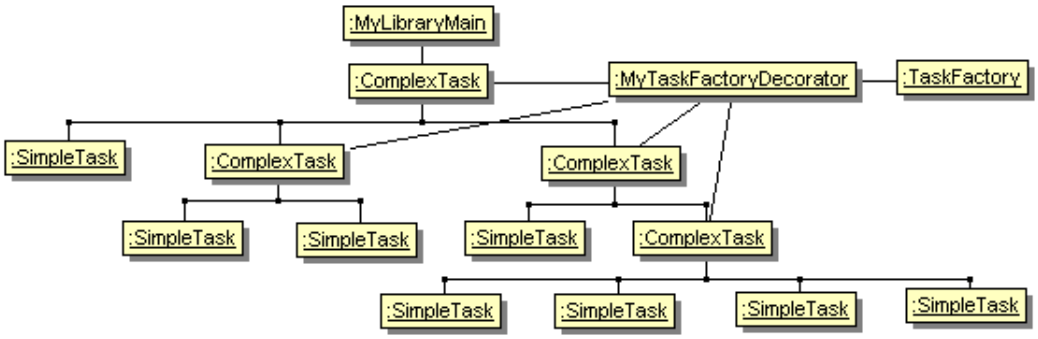


Figure 15.16: Example of library Task that contains many other Tasks

Let's now look at the example of how the client-creator should create task in configure() method of ComplexTask:

```
try {
 addTask("ONE_WEIGHTED_LOCAL_JOINT_PDF","owljp");
 addTask("REGISTER_DATA_SELECTOR","Matrix1Select");
 addTask("REGISTER_DATA_SELECTOR","Matrix2Select");
}catch(Library::Tasks::TaskFactory::BadTaskCreation e){
        std::cout << e.what() << std::endl;
        exit(0);
}
```

Creating a Task requires two arguments. The first is an identifier of the type of Task that we want to create. The second will be a name of the created object and a key in an associative array "childrens". From the client-creator standpoint, there is no difference here between a SimpleTask and a ComplexTask. We can now present an example of instantiated Tasks (Figure 15.16). To be precise we must say that the figure is not fully correct. In fact, we don't instantiate ComplexTask-s or SimpleTask-s but classes that inherit from them. The structure has also one decorator which extends the resource of TaskFactory. The mere presence of the decorator is for tasks imperceptible (*Decorator Pattern* [RJ94]).

It is often need to easily configure Task-s. Each Task can have a set of parameters named ParametersContainter (Figure 15.17). The container can have many parameters. Parameters can be set either when the Task is being created, as well as dynamically during the runtime of the library. This gives the possibility to create a self-configure library. Any change of parameters calls the method refreshParameters() of the Task. This method is virtual, so the client-creator can define it for any task (can be seen in Figure 15.14). The

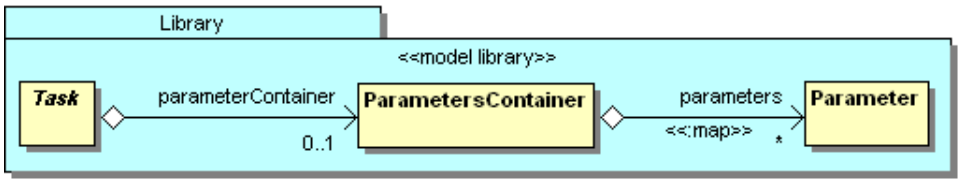


Figure 15.17: ParametersContainer

client-creator's Task should upgrade its attributes depending on parameter values. From this method also the client-creator can, for example, force a reconfiguration in ComplexTask-s. Automatic configuration is ensured after the creation of Task. Let's show now an example how to create a parameterized Task.

```
addTask("MAP_WITH_INLIERS","MapWithInliers",
              new Library::ParametersContainer(5,
  "SIZE",           new Library::Parameter(int(size)),
  "TEST_MAP",       new Library::Parameter(bool(false)),
  "TEST_VITERBI",   new Library::Parameter(bool(false)),
  "RANGE_UP",       new Library::Parameter(int(40)),
  "RANGE_DOWN",     new Library::Parameter(int(35))  ));
```

The constructor of ParametersContainer accepts a variable number of parameters. Parameters can be of any basic C++ type, as well as the universal type Data which is discussed later. Nevertheless, we can say that the Parameter may be, for example, a matrix of elements of the type defined by the client-creator.

It should be noted that the setting of parameters in the hierarchy of Task-s can be recursive. In one place we can configure our entire library, as well as it can be done by the final client from his code by evoking methods defined by the client-creator.

The above ideas are an implementation of **reflection** in computer program.

15.6 Interaction between Task-s

All created Task-s will not work at the same time. One should perform before the others. Let's introduce a term "task ready", or equivalently "active task". It is such a Task that is ready to be executed. The owner of a group of Task-s is always ComplexTask and it must decide which task will be executed at a

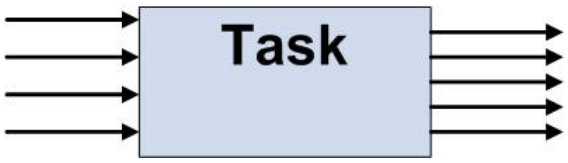


Figure 15.18: General Task

time. Therefore, the ComplexTask has a method execute() where is evoked execution of one active Task. The SimpleTask has also this method, but here will be evoked the method run() which was defined by the client-creator. Let's present the fragment of execute method from ComplexTask:

```
// Do one SimpleTask
std::map<std::string, Task *>::iterator it;
it = childrens.begin();
while(it != childrens.end()) {
        if(it->second->isReady() ) {
                it->second->execute();
                return;
        }
        ++it;
}
```

ComplexTask is asking its subtasks whether they are ready to work. The first of them that answer "yes" (true) will be executed and the polling is finished. In case of SimpleTask run() method will executed. In the case of ComplexTask further polling will be made but in the level more nested. Anyway, maximaly only one SimpleTask will be carried out. This solution is a combined version of the pattern *Chain of Responsibility* with the pattern *Composite* [RJ94]. Now we know that the SLEEP message will be sent between each global polling starting from the mainTask.

What does it mean that the task is ready?. In procedural programming functions are performed one after another. We can call them when we have all arguments ready. Also there are functions that do not require arguments. However they must often wait for the end of other functions execution. Looking at these functions as Task-s, we can say that the Task-s are not ready to start up if they don't have input data.

Task can have zero or many input ports (Figure 15.18). The same it is with output ports. Ports will serve us to connect Task-s together. There are no bi-directional ports in the current implementation. Implemented inputPorts and outputPorts are of the same type (Figure 15.19). Task collects

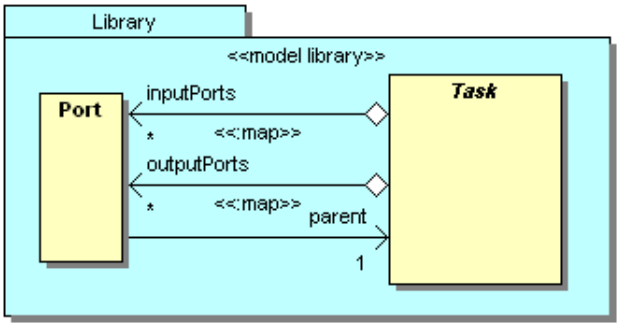


Figure 15.19: Task with its Port-s

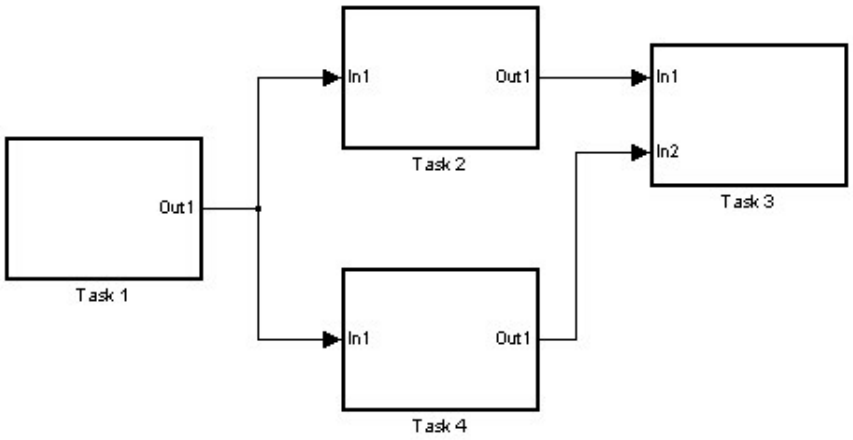


Figure 15.20: Example of four related tasks.

them in separate containers. Each Port belongs to only one Task. To better understand the role of Port-s let's look at example in Figure 15.20. We see that the relation between output ports and input ports is like one to many. Each input port means information that Task is interested in. In information produced by Task and sent through an output Port may be interested more Task-s. The problem is a typical case where it is appropriate to use the *Observer Pattern* [RJ94]. What is the input Port and what is the output Port - Observer or Observable? Figure 15.20 may suggest the answer. But before answering definitely let's take a look at Figure 15.21. Input Port-s must listen for external information, as well as inform the internal blocks. Output ports listen, whether inside information is ready and transmit it outside. As we can see ports are generally both Observable and Observer-s (Figure 15.22).

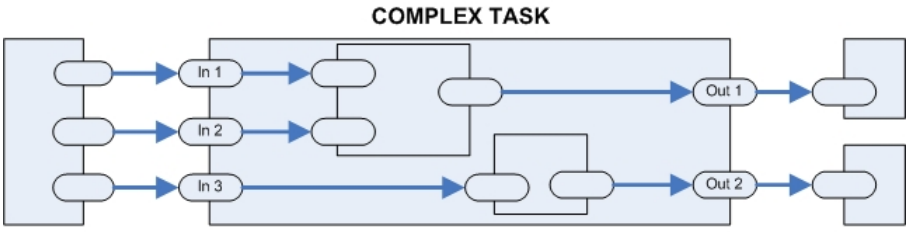


Figure 15.21: Example of ComplexTask with its ports.

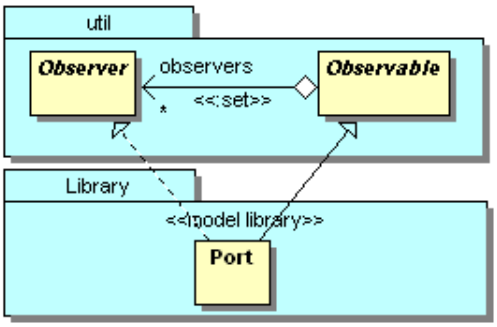


Figure 15.22: Port

To add a Port to a Task we should write in configure() method for example:

```
addInputPort("In");
// ...
addOutputPort("Out");
// ...
```

Parameter will be a name of the Port. Now we can connect Port-s. In a typical solution the client-creator will do it just after creating tasks. The client-creator can connect in ComplexTask:

- directly input and output Port of ComplexTask (by giving the name of input Port and the name of output Port);
- output Port of one Task with input Port of second Task (by giving the name of first Task with the name of its output Port and the name of second Task with the name of its input Port) - it is also possible to make a feedback!;
- input Port of ComplexTask with an input Port of one of internal Task (by giving the name of input Port and the name of a Task with the

name of its input Port);

- an output Port of one of internal Task with an output Port of ComplexTask (by giving the name of Task with the name of its input Port and the name of output Port);

```
try {
 connect("In1","Out1");
// ...
 connect("wLocPdfs","Out","InliersEnhancement","In");
// ...
 connectInputPort("ImgRegister","wLocPdfs","ImgReg");
// ...
 connectOuputPort("fs_i","Out","Out");
// ...
} catch (Library::ComplexTask::BadConnection e) {
        std::cout << e.what() << std::endl;
        exit(0);
}
```

Information about attempt to do an illegal connection is sent by the exception BadConnection.

15.7 Data

Connecting Port-s allows notifications between Task-s. Practice requires something more. Task-s perform certain operations and the results of them would like to be sent to other Task-s that they can operate on them and so on. The results of operations we can generally treat as Data (Figure 15.23). We present two approaches.

15.7.1 Data as a string of bits

For Task-s it can be anything that can be stored in a binary form. And it is the Task who should interpret it. Let us discuss the information carrying by Data. In the current implementation they are:

- tag - name of Data;
- ptr - this is a place where data are stored physically, of type "any" so for C++ *void **;

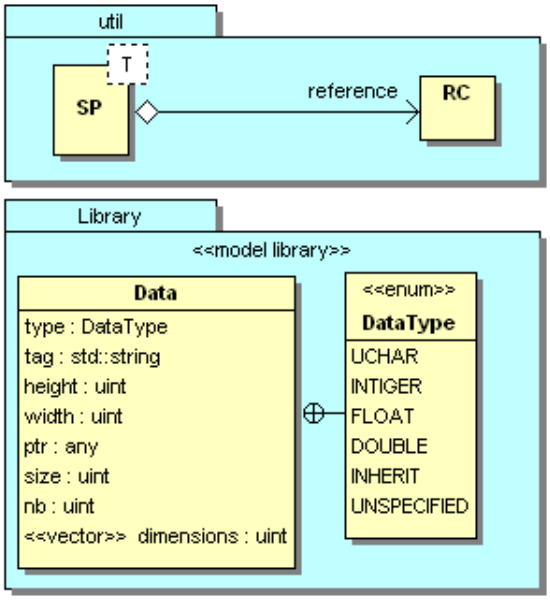


Figure 15.23: Data

- nb - number of collected items;
- size - memory size in bytes of one item;
- dimensions - multi-dimensional data organization;
- type - by default is UNSPECIFIED, if it is specified it allows to check the compatibility of types or it allows that blocks with DERIVATED data type can dynamically determine data type;

Data is created in SimpleTask and it is held in its ouput Port. But Data can be shared with more Port-s (inputs of other SimpleTask-s, inputs and outputs of ComplexTask-s). Who should own it? For programming languages such as C++ it is an essential question. Someone has to destroy unused Data. Creator of Data could theoretically know how many listeners there are, but certainly doesn't if these Data are still used by them. Giving someone the property of the Data would create a large number of use cases. Therefore a different solution is proposed. Ports will held a smart pointer to Data (util::SP<Data>). The concept of smart pointers is widely discussed on the Internet. Used implementation comes from the site of The Code Project [Pro]. Now SP-s that are independent of the system will be responsible for cleaning up.

Below are some examples of how to create Data. We can create Data statically and give it to Port (here of the name "Out").

```
unsigned int table[2];
.
.
.
std::vector<unsigned int> dim;
dim[0] = 5;
dim[1] = 10;
dim[2] = 7;
putData("Out", table, sizeof(unsigned int),dim);
```

We need to specify a name of Port, an address of elements, a size of a single element and dimensions of a data matrix. After that new Data will be automatically created and assigned to Port as also connected Observers will be notified. The entire array of elements will be copied to Data. In the case of very frequent evoking a Task that produce a large data structure, copying could be time consuming. Thus a possibility of dynamic creation of data was left also to the client-creator. For example:

```
int * out =
reinterpret_cast<int *>(operator new(h*w*sizeof(int)));
.
.
.
std::vector<unsigned int> dim;
dim[0] = h;
dim[1] = w;

util::SP<Library::Data> outDTA(new Library::Data(
                                "LUTout",
                                out,
                                sizeof(int),
                                dim,
                                false) );

putData("Out", outDTA);
```

The current implementation may be incomprehensible for someone not fully friendly with C++. It is recommended to use a global operator new in place of a operator new of particular type. In anyway binary data is stored as *void ** and for its destroying a global operator delete is used. Next, we can

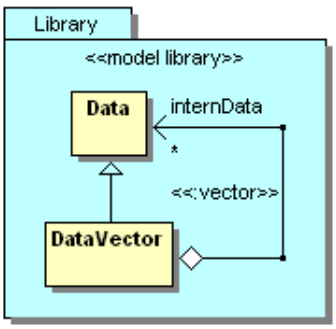


Figure 15.24: DataVector

process on the allocated memory. At the end we create statically a smart pointer SP to the dynamically created Data. We need to specify a name of data, an address of its elements, a size of a single element and dimensions. The last argument means that we don't want to copy elements. Then we send the Data to the output Port.

As we can see, binary data is sent between Task-s - not objects. The client-creator should be careful if he tries to send objects. Leaving the entire responsibility, for example, for std::vector<int> to the SP<Data> causes small memory leakage. For this purpose a new type was introduced - DataVector, which has the same interface as the class from the standard library (Figure 15.23). DataVector is Data which is able to store other Data. As long as DataVector object exist, there will be at least one reference (in SP) to each stored internal object Data. Let's see an example:

```
Library::DataVector * dv = new Library::DataVector();
unsigned int x = 256;

int * p1 =
  reinterpret_cast<int *>(operator new(x*sizeof(int)));

int * p2 =
  reinterpret_cast<int *>(operator new(x*sizeof(int)));
.
.
.
std::vector<unisgned int> dim;
dim[0] = x;
```

```
Library::Data * d1 =
 new Library::Data("data",p1,sizeof(int),dim,false);

dv->push_back(d1);

Library::Data * d2 =
 new Library::Data("data",p2,sizeof(int),dim,false);

dv->push_back(d2);
.
.
.
util::SP<Library::Data> dsp(dv);

putData("Out",dsp);
```

In the example DataVector dv will store two Data objects. The method push_back works well when we are creating a new Data object by giving a pointer to Data (push_back(Data *)) or with already existing Data that could be created anywhere, by giving a smart pointer (push_back(SP <Data>)). In the first case a new SP will be created anyway to the particular Data.

Up to this chapter the whole design was independent of programming language. We could easily implement this project in any other object-orinted language - Java, C#, ... The solution presented here is unfortunately only typical for C++. If anyone still uses C++, it is certainly because of its performance. Presented solution is very efficient and actually it comes straight from C. We can operate directly on memory cells. And thanks to C++ we don't need to bother with cleaning up (smart pointers). If we need more abstraction, data be easily reinterpreted to the chosen structure in a specific Task.

15.7.2 Data as an object

What if the client-creator needs to send objects of classes that are owners of other objects. Their destructors wouldn't be called. Class DataVector may suggest the answer. Client-creator's class should inherit Data which destructor is virtual.

In languages such as Java or C# project would be much simpler. There would be no need of any kind of smart pointers.

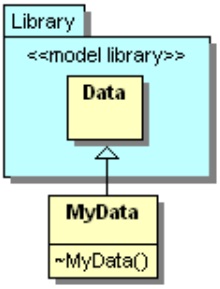


Figure 15.25: Client-creator's Data

15.7.3 Reading Data

We already know how to create and send Data. Thanks to the *Observer* design pattern Data arrives to others Port-s. An example shows how to read it:

```
util::SP<Library::Data> dta(getData("inMatrix"));

unsigned int height = dta->dimensions.at(0);
unsigned int width = dta->dimensions.at(1);

const int * const input_beg =
 reinterpret_cast<const int * const>(dta->ptr);
```

In the current implementation we need to take a smart pointer from an input Port (here of the name "inMatrix"). We can check the dimensions. And a mere interpretation of data depends on the client-creator. A dynamic_cast should be used for objects that inherits from Data.

15.7.4 Data transmission

Let's look at an example of Data transmition in Figure 15.26. The process is very short. We sends Data to an output Port. Port shall notify its observers. Each observer knows now that Data has arrived. Port-observer copies smart pointer into its resource (dta). At the end the port-observer calls a portManager of its parent.

At the end we should add that it is also possible to send only a message without Data through an output Port.

```
putData("SET");
```

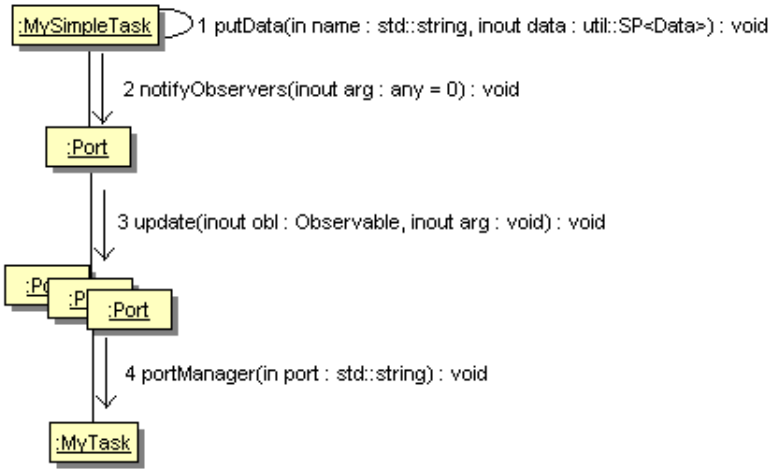


Figure 15.26: Data transmission

15.8 Task activation

We know how to create a structure of any Task. But how do we know that at any given time a Task can be done? - when the task is active? When the SimpleTask could perform a client-creator's code and ComplexTask could look for internal actvie Task. To visualize how many cases we can have let's look again at Figure 15.21 on page 97. What should we do if the information in the port In 3 has arrived and there is still no information in the ports In 1 and 2. Should we wait for the others and then do all internal tasks? Or maybe we can activate this ComplexTask and let to one internal Task be done. There is no answer to this question. It all depends on what the client-creator will need in a particular case. To simplify Task-s management two general types of Task-s were specified:

- atomic - activate, if all input information is available;
- non-atomic - activate with any input information;

We conclude that it doesn't fulfill all use cases. Still we need a block that will behave like atomic, but will be also sensitive to some Port-s as a non-atomic block. An example can be an atomic block with a reset input. Let's introduce an attribute of the Port: asynchronous. Then the atomic block should be activated if all non-asynchronous input Port-s have been notified or at least one of asynchronous. The addition of asynchronous Port looks as follows:

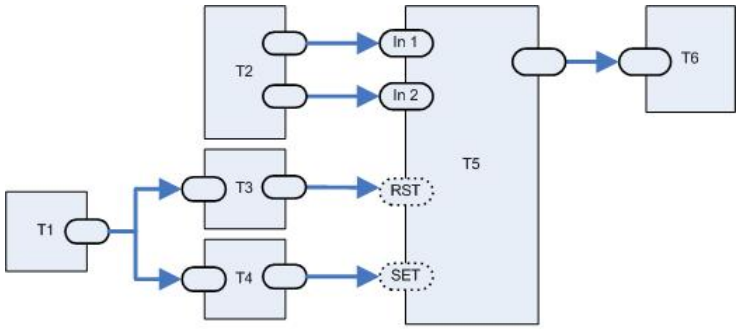


Figure 15.27: Example of ambiguity of Task executions' order

```
addInputPort("RST",true);
```

SimpleTask should know why it was evoked. For example, if it happens due to reset. This will be indicated in the parameter of run(std::string & port) method as the name of input Port or it will be indicated "ALL" as a information that all non-asynchronous input Port-s have been notified.

Let's consider now an example in Figure 15.27. Asynchronous Ports are marked with dotted line. Let's imagine a scenario: Task T1 notified its observers by its Port. Task T3 and T4 are active. Let's suppose that Task T2 hasn't still produced any information on its output Ports. We can now do the Task T3 just as well T4. Which one should be the first? From the logical point of view, it doesn't matter. Each of them can wait, and the results of execution will be the same. But thanks to T3 RST Port of Task T5 will make T5 active. Thanks T4 Task T5 will be activated by Port SET. More, with *Chain of Responsibility* proposed in chapter 15.6 we could have five different execution orders:

- T1 -> T3 -> T5 -> T4 -> T5
- T1 -> T4 -> T5 -> T3 -> T5
- T1 -> T4 -> T5 -> T3 -> T5
- T1 -> T4 -> T3 -> T5
- T1 -> T3 -> T4 -> T5

We could consider also that in two last cases T5 is done twice. This is not an easy decision. One could say that the design of client-creator is incorrect. What does he really want to do? This issue hasn't been resolved yet.

The implementation of for SimpleTask and ComplexTask is slighty different. Let's consider an atomic ComplexTask in Figure 15.28. The Task

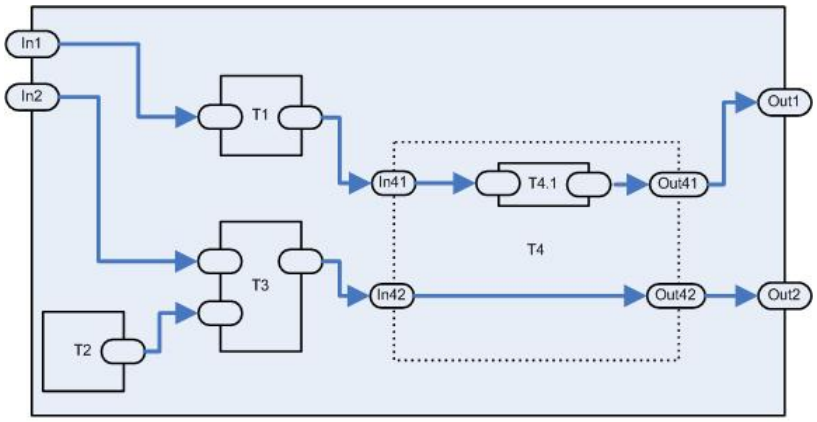


Figure 15.28: Example of atomic ComplexTask

is atomic because the client-creator doesn't want to start processing until there isn't all input data prepared for him. It can do anything that takes two arguments: In1 and In2. We shouldn't forget that Task could have also a memory.
Suddenly a new Data comes to Port In1. Data is written into that Port. But nothing special beyond that happens. Next - once again new Data comes to Port In1. That new Data is saved and the old one is forgoten. For the Task it means that someone from outside hasn't still a Data for Port In2 and has already changed his mind what to give to the Port In1. Finally new Data comes to the port In2. The Task is still not active. But input Port-s In1 and In2 can now notify their observers. We don't know what happends inside T1 but it starts to be active and it wants to have an opportunity to execute! It is making our ComplexTask active as a parent Task and he ask us to do the same with our parent and so on. Like this the Task which is on the top of hierarchy will know that there is for sure at least one Task that wants to be executed. And what about T2? It has no inputs to make it active. What it really is? Probably it is a constant. It wants to be executed once after parent Task is activated. So T2 has been already active - even when our ComplexTask wasn't. So now T1 and T2 can be executed. Task T3 is activated after the execution of T2.

Now we have on our way T4. T4 is a non-atomic ComplexTask. The non-atomic ComplexTask will often help to organize internal Tasks. If the ComplexTask becomes too complex, we can group internal Tasks into functional blocks. We should expect the same functionality with and without non-atomic ComplexTask-s. So what is happening when Data cames to an input Port of non-atomic ComplexTask? At once this Port notify its ob-

servers. The activation of non-atomic ComplexTask also depends on internal Task-s. The input Port In42 is notified after execution of T3. This Port at once notify output Port Out42 and Out42 at once notify port Out2. The input Port In41 is notified after execution of T1. At once In41 notify the input Port of T4.1. T4.1 becomes active and it makes T4 with its parents active as well. Now T4.1 could be found with polling and can be executed. After execution Port Out41 and Out1 are notified.

The SimpleTask generally stops to be active after one execution. ComplexTask stops to be active when during a polling there isn't any active Task among its childrens. After that it will make active all internal Task without inputs.

A method of Task - portManager is involved in management of Task activation. It is evoked by every Port when it gets a new notification (Figure 15.26). Figures 15.29 and 15.30 present the current implementations of portManager method for SimpleTask and ComplexTask.

Let's first look at SimpleTask. The behavior will be different for atomic Task and non-atomic. For non-atomic we need to remember the ID of Port. Then we can activate the Task previously restoring the Port to a default state. If it is an atomic SimpleTask, we need to consider if the Port was asynchronous or non-asynchronous. For asynchronous the case is the same as for non-atomic SimpleTask. In the another case we need to check if all asynchronous input Port-s were notified. If not - nothing happens. If yes, we will remeber that the activation of Task was because of all inputs. We resotore them to default and finaly we can activate the Task.

The ComplexTask's method is a little bit different. Also an output Port could be notified. In this case we should immediately send this message forward. The rest is similar to SimpleTask method. But we don't need to remember the ID of Port. What we need is to send forward the Data from Port-s. The mysterious action is here setReadyWithParents. The ComplexTask will be activated anyway if one of internal Task is activated. It matters in the case of "empty" atomic Task where there are only connections between the input and output Port-s.

15.9 Advanced use of Library

In this section we will find a few additional functionalities. They visualize how the further development can be carried out.

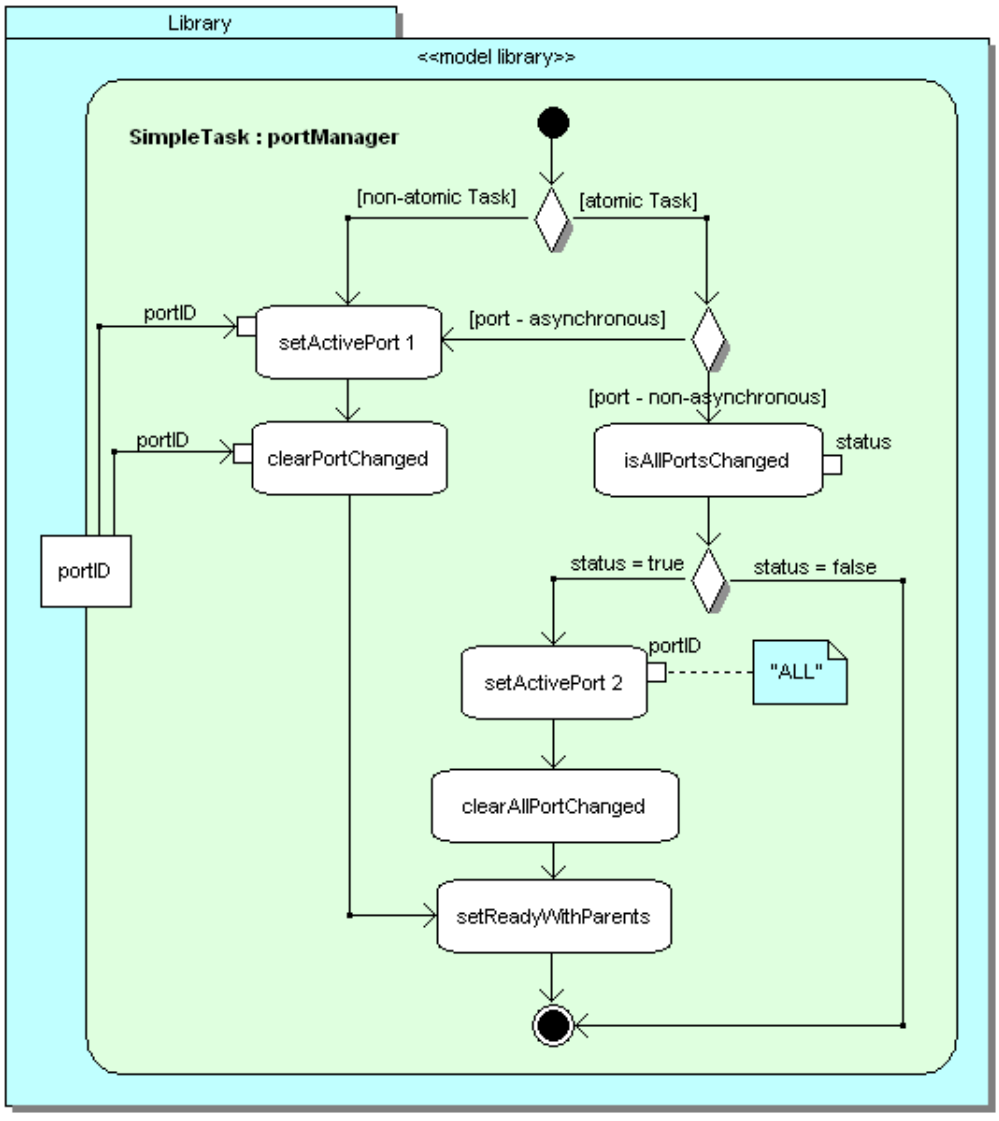


Figure 15.29: Method portManager of SimpleTask

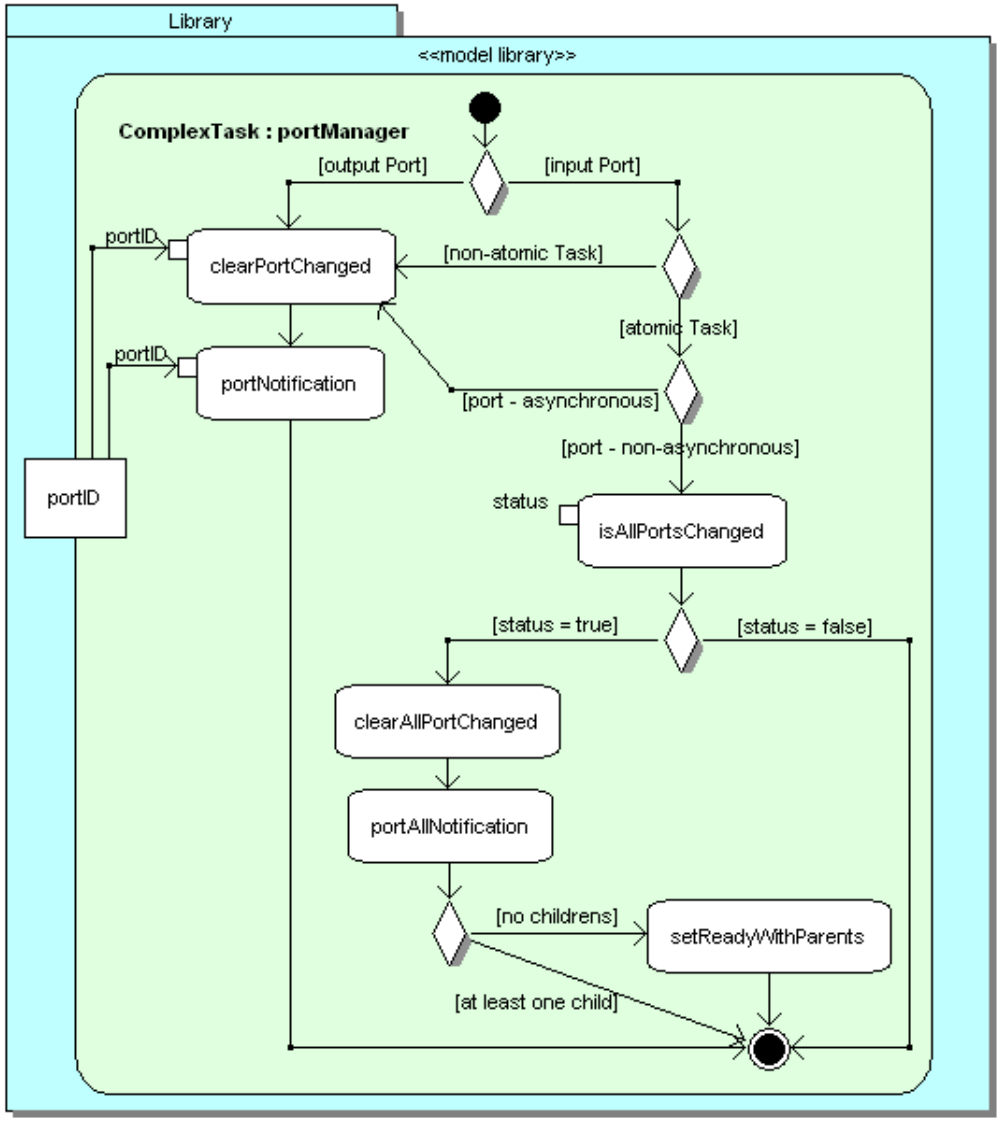


Figure 15.30: Method portManager of ComplexTask

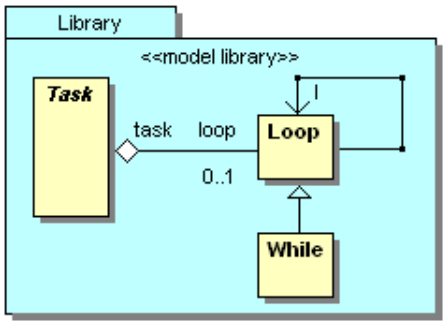


Figure 15.31: Loop

15.9.1 Loops

Sometimes there is a need to perform Task many times after only one activation. Such Task-s have Loop (Figure 15.31). After deactivation of an atomic ComplexTask we will give a possibility that any child (or child of child which is a non-atomic ComplexTask and so on) which has Loop can be performed once again. So far only "while" for SimpleTask is implemented. We add the option during a configuration:

```
addLoop("while");
```

In the current implementation, if we want to control the Loop "while" during the execution we need to use methods:

```
loop->setActive();
// or
loop->setNoActive();
```

15.9.2 Dynamic structure

In chapters 15.5 and 15.6 we learned how to create a structure of ComplexTask-s by. In examples the methods were invoked by the ComplexTask on itself. However, nothing precludes that these methods could be called by Task-child. We can create structures that reconfigures themselves during the runtime in dependence on some intermediate results.

15.9.3 Dynamic types

One of the main feature of C++ is a strong type control. Task-s expect a particular type of data. The client-creator must keep this in mind! However, often we would like to make more universal Tasks. We shall give an example

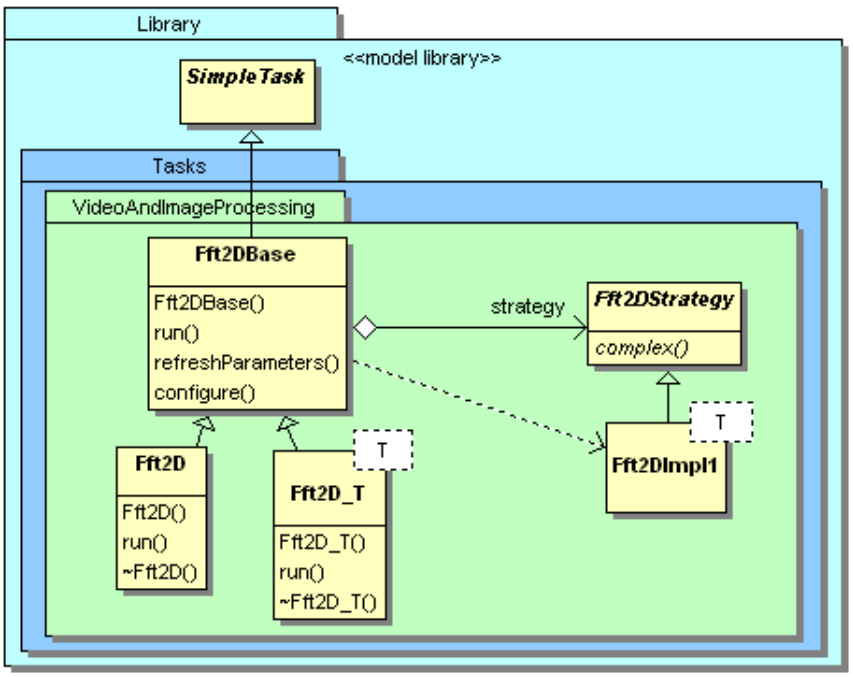


Figure 15.32: Example of universal Task

of Task which calculates a complex FFT2D. The basis of the solution is a *Strategy* design pattern [RJ94] - Figure 15.32. The client-creator has a choice. He can ask a TaskFactory to create an universal Task Fft2D or he can specify Fft2D_T<T> in his TaskFactory decorator. The first proposition is more interesting. Let's recall that comming Data can have an information about its type (chapter 15.7). The run() method of Ffft2DBase is virtual, so in fact run method of Fft2D will be evoked. There a strategz will be created and after the run() method of Fft2DBase will be evoked. Fft2DBase and Fft2DStrategy don't need to know types as they don't do any calculation. They use "any"-type, so for C++ *void **.

We have two kinds of SimpleTask. One with a specified type. The type of comming Data will be checked before creating a strategy. In the case of dataType = INHERIT the proper strategy will be chosen depending on the type of comming Data during the runtime.

The cost of this solution is that the code for all kind of strategies will be generated anyway. So the size of program will be bigger even if we don't use all of types.

15.10 Task Package

This package is designed for a collection of Task-s that can be easily reused. Task-s are grouped according to its function. The factory has references to all

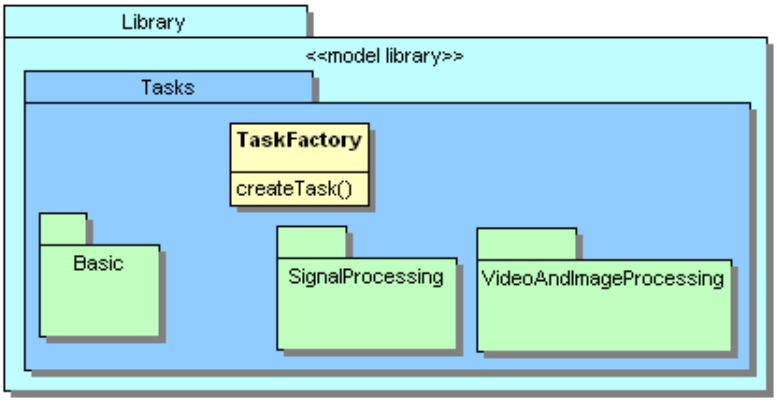


Figure 15.33: Task Package

Task-s. Task-s should be distributed among TaskFactoryDecorator-s if the library will grow significantly. Then the client generates only those groups of Tasks that are used. In addition, it would be a very good solution compiling the Task-s separately into dynamic libraries. If we use many libraries based on the Library Package we will obsevre a significant reduction in sizes of applications.

15.11 Future work

The development of the Library should be followed as far as are discovered needs. We will give here some examples:

- generator - it is kind of Task which is activated when the Library ends its mainTask (ex. Library could control an external electronic device all the time);
- universal Task-s - set of universal Task-s should being constantly extended. New libraries will be created faster and faster.

Chapter 16

DataProcessingLibrary Package

16.1 Purpose

The Library Package allows to create Task-s that take data from a hard-drive or some external devices, Task-s that process data and finally Task-s that are able to write data to hard-drive or send it to external devices. However we are often dealing with a situation that data already exist in the client application. He would like to send it in the simplest way to the library and also be able to directly receive results. This is the reason of extension for the Library Package.

16.2 Use cases

The package DataProcessingLibrary is only an extension of the Library Package (Figure 16.1). We can guess that if the library has to process external data, the mainTask should have Ports. Data should show up at its input Port-s to make it activated. Data will have to be sent from a client code to the library. Let's distinguish two types of libraries:

- passive : if the mainTask is not active, the library will be in PAUSED state, sending data can resume the library by activating the mainTask;
- active : if the mainTask is not active, the library will ask the client for necessary data;

The Figure 16.2 presents the use cases of DataProcessingLibrary. They are:

- DataRequest : In the case of active library. A function dataRequestUI will be evoked with ID (std::string) of needed data as an argument;

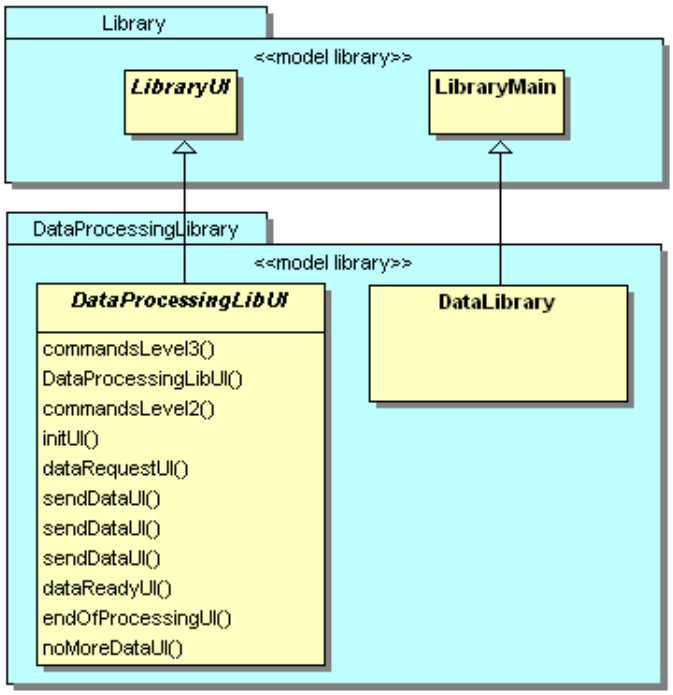


Figure 16.1: DataProcessingLibrary - Main classes

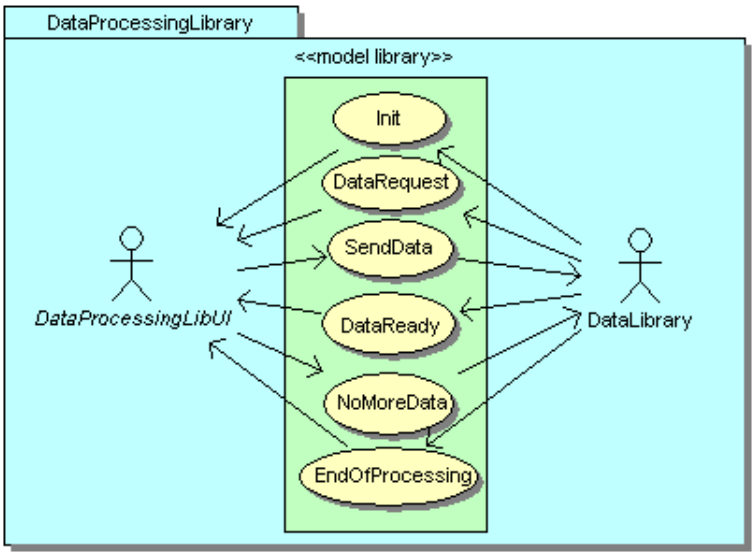


Figure 16.2: DataProcessingLibrary - Use cases

- SendData : The client can use this function to transfer his data;
- DataReady : Here data from output Port-s of mainTask will come (with ID);
- NoMoreData : This is information for the library that there'll be no more data sent. Probably the library will generate the final results of the processing and finish its work;
- Init : The client can define a function initUI to initialize transmission of data on his side. This function will be evoked after RUN message. This solution can help the client to keep all code connected with a particular library in one place.
- EndOfProcessing : The concept of this functionality is the same as Init but it will be evoked after STOP message.

The examples of use cases for passive and active library are presented in sequence diagrams in Figures 16.3, 16.4 and 16.5. In the last example there is a library that needs one type of input information and give back also one information. But to produce the first results it needs to store two data from input. When the client says that he won't send more data it doesn't mean that the library can't work any longer. The library gives to the client more one output data.

16.3 Main design

In Figure 16.6 we can see the class hierarchy of DataProcessingLib Package. LibraryMain is represented as DataLibrary. DataMessage adds new messages. But above all, the DataMessage extends Message class with the possibility of data transmission. LibraryUI was enhanced by inheriting DataProcessingLibUI to read new messages. On the library side a new type of LibraryInterfaceDecorator was created.

In the current implementation sending data looks like below:

```
void sendDataUI(const std::string & tag,
                const void * const data,
                unsigned int size,
                unsigned int dim, ...);
```

Where *tag* should be the same as an input Port of mainTask to which we want to send data. The adress of data is here *ptr*. Because the data is casted to type "any" we need to know how much of memory take one element of

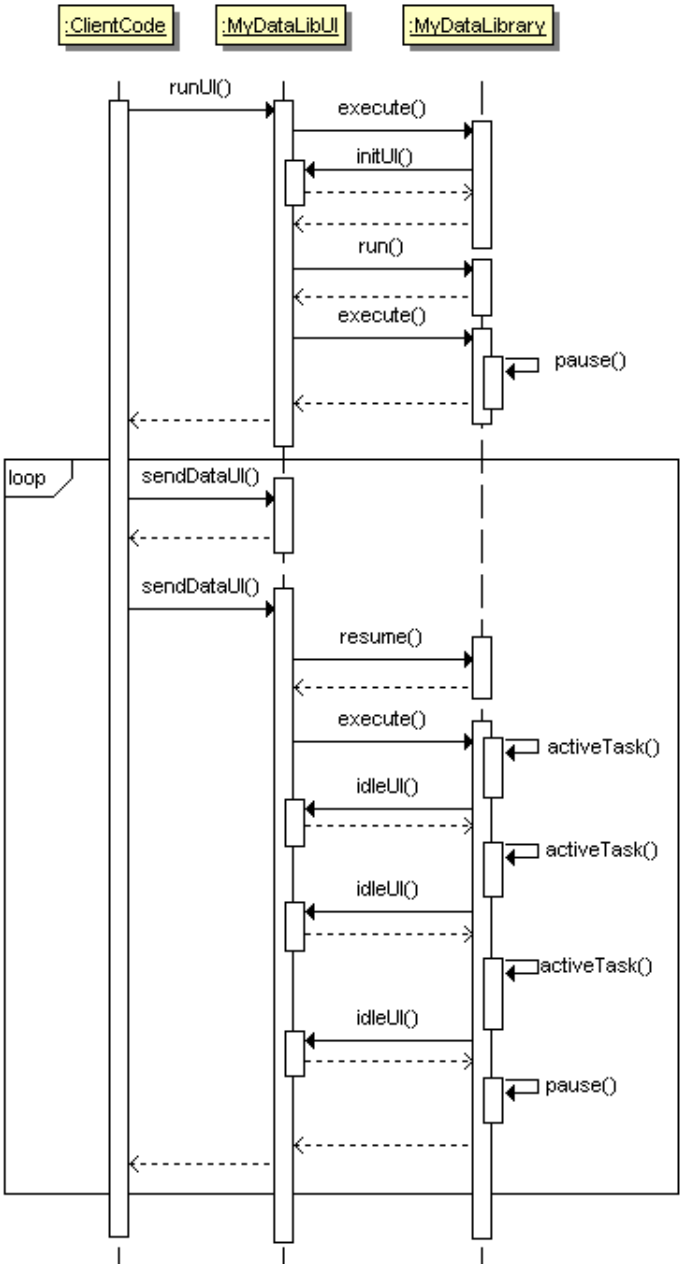


Figure 16.3: Example of data transmission to a passive library

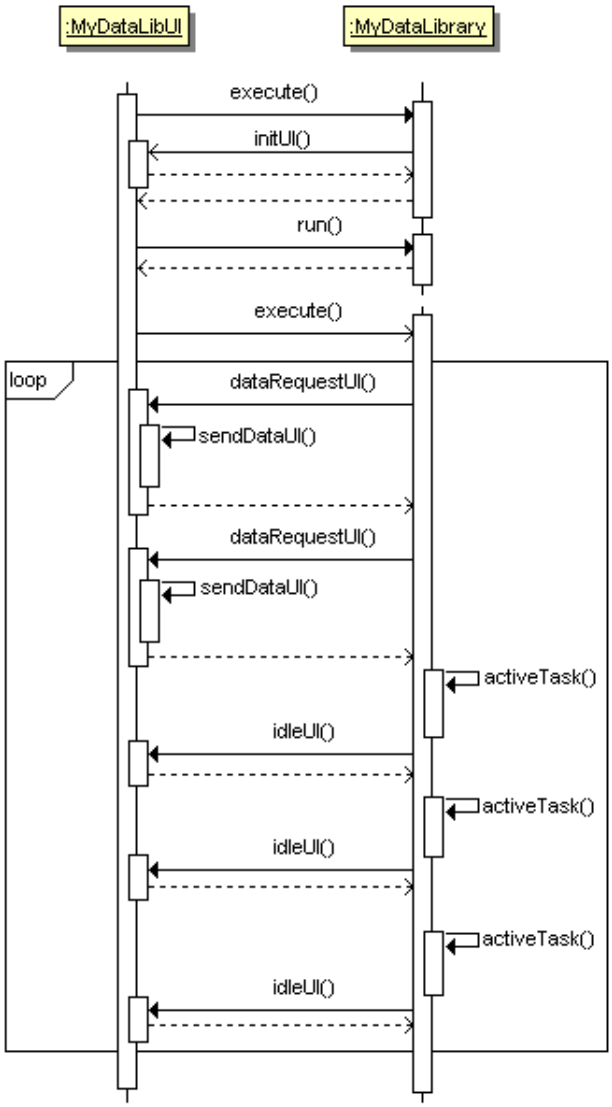


Figure 16.4: Example of data transmission to an active library

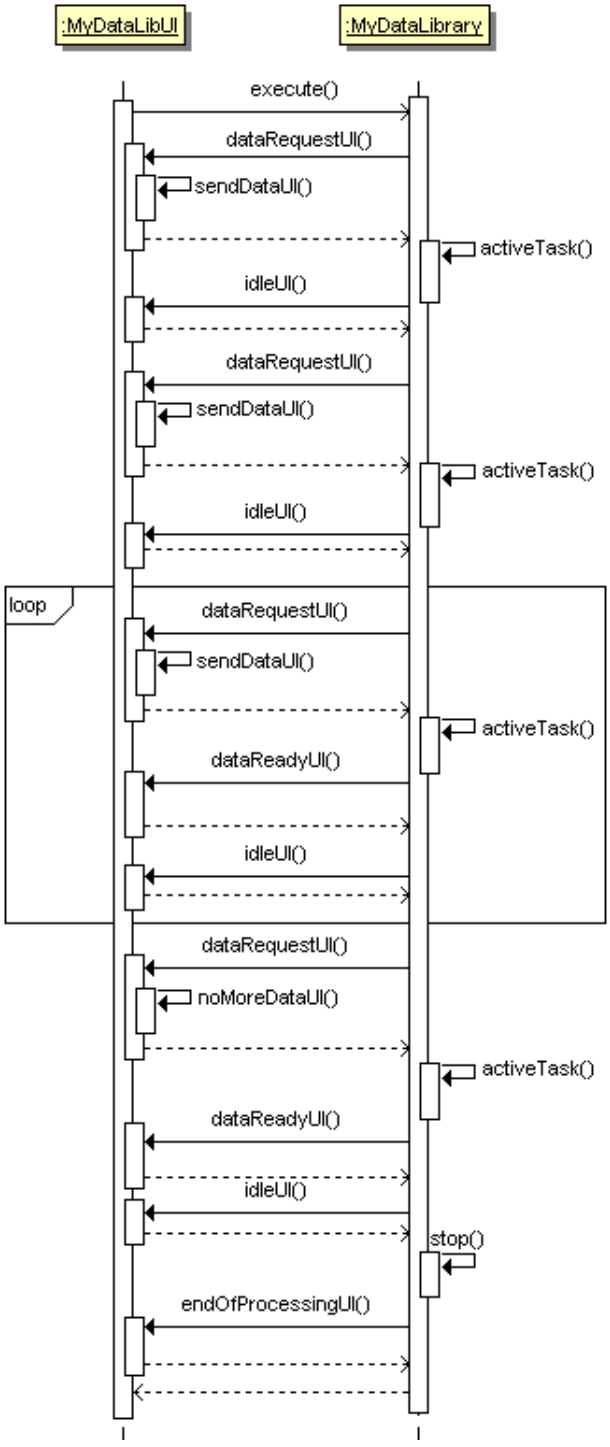


Figure 16.5: Example of data reception from an active library

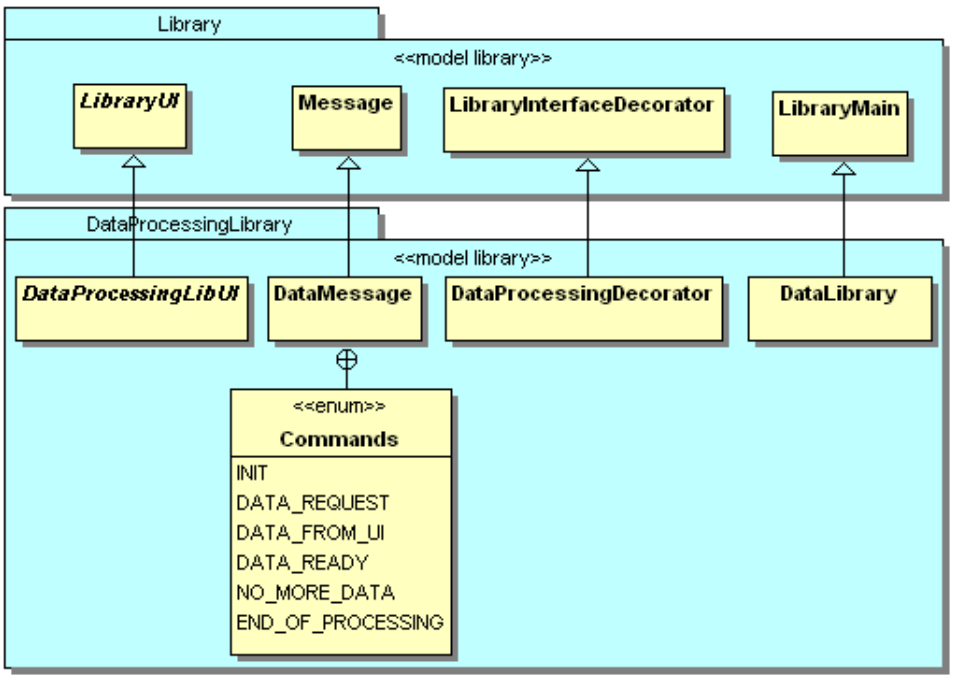


Figure 16.6: DataProcessingLib Package - architecture

data. Next we need to inform about organization of data. First we give a number of dimensions. After, function takes a variable list of arguments that each of them represents one size. An alternative calling is:

```
void sendDataUI(const std::string & tag,
                const void * const data,
                unsigned int size,
                const std::vector<unsigned int> & dms);
```

Where the dimensions are stored in vector. For active libraries sending of data will probably take place in the method dataRequestUI() when the library asks for data to its input non-asynchronous Port-s. Declaration of this method looks like this:

```
void dataRequestUI(const std::string & tag);
```

The argument is the name of Port. Data for asynchrounous Port-s can be sent from anywhere - for example from the method idleUI().

Receiving data is also intuitive. When the library generates some output data, the function dataReady() from interface side will be called:

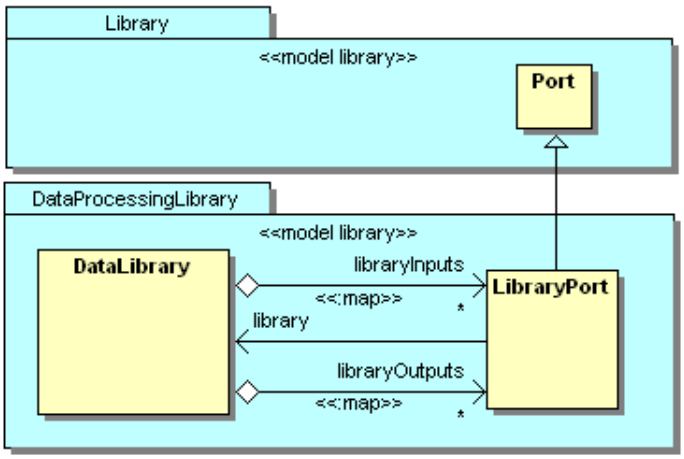


Figure 16.7: LibraryPort - adapter between DataLibrary and mainTask

```
void dataReadyUI(const std::string & tag,
                 const void * const ptr,
                 unsigned int nb,
                 const std::vector<unsigned int> & dim)
```

We get information about output Port of mainTask that produced data, its physical address, number of elements and data organization. The definition of function is up to client. He is supposed to know of what a data type (for example from a library specification).

The only indicator of the mainTask is having no parent. Any Task could be the mainTask. So the mainTask should still think that Data-s comes from other Task-s. This is a typical problem where it is appropriate to use *Adapter* design pattern [RJ94]. The applied implementation is a kind of *Adapter of objects*. Library has also Port-s named LibraryPort-s (Figure 16.7). They could be connected with Port-s of the mainTask. The mainTask will not notice a difference. But from the DataLibrary's side the LibraryPort accepts data format - DataMessage.

One of the last issue to discuss is the use case noMoreData. Sending this message will normally end the library's Task. However, the client-creator may connect this message with one of the asynchronous input Ports in the library's constructor. For example:

```
connectNoMoreDataTo("End");
```

Now, after sending noMoreData message the library will continue its work having the information that no more data will come.

Chapter 17

FlickerRemovalLib Package

17.1 Purpose

The purpose of this package is to provide a library that removes the flicker of luminosity in old movies. We don't specify with which system it will work or on what platform. The client application could be one or multi-threaded working on Linux, Windows, MacOs or even in embedded systems. Implemented algorithm is based on design from the part III.

17.2 Main design

FlickerRemoval Package is based on Library Package with extension of DataProcessingLibrary Package (Figure 17.1). The FlickerRemovalUI is really only an alias for the DataProcessingLibUI. FlickerRemoval that is a generalization of DataLibrary will create a mainTask - FlickerRemovalTask that is a ComplexTask. By FlickerRemoval interface we can choose a demo version. In demo, the output movie is divied vertically into two parts. One is processed and second is original from the input. We can change also the algorithms parameters. By method setSize() we can set with how many neighbouring frames each frame should be compared in a restoration process. A default number is 14 (7 backward + 7 forward). By the method setLocality() we can set what locality of flicker the library should consider. A default value is 0. It means that the flicker is a global artifact. Value 1 would mean that there are 2 sources of flicker positioned horizontally. Value 2 would mean that there are 4 sources spaced regularly throughout the space of movie frame.

There is also FlickerRemovalTasksFactory. This TaskFactoryDecorator is a collection of Task-s that are specific to flicker removal algorithm. All new classes are parameterized with a data type of pixels. For now only fixed-point

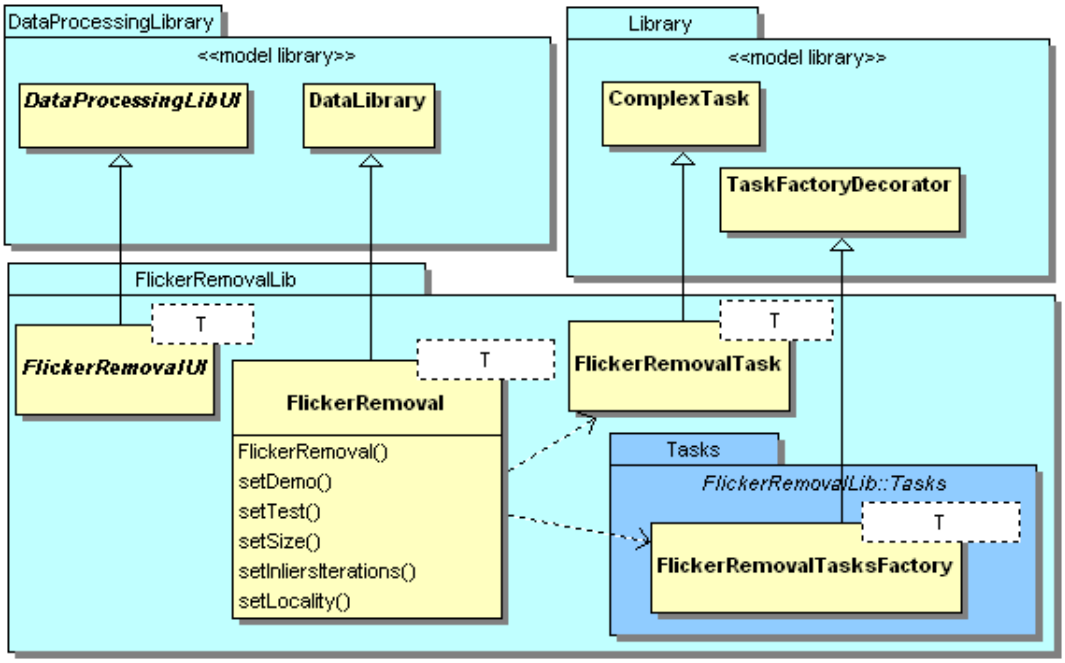


Figure 17.1: FlickerRemvovalLib - architecture

reprezentations are supported (as unsigned char, int, ...).

A user who wants to use this library doesn't need to make more than is required by DataLibrary. He should just instantiate the class FlickerRemoval and define his own FlickerRemovalUI with methods: idleUI(), dataReadyUI() and dataRequestUI() if he wants to use the library as active. Aslo he can define function init() and endOfProcessing() if he wants that the library will call his initialization before processing and his cleaning after the end of processing.

17.3 FlickerRemovalTask

First, let's define an interface of the mainTask (Figure 17.2). We regard the Task as a black box. As the flicker is a distortion of luminance, the library needs only this component. The library will ask for the luminance of only one frame at once (Yin). The client has to send data with two dimensions of image - height and width. As a results on the output side a de-flickered image should show up (Yout). However, to restore one frame we need to have also its neighbors. This necessitates storing images. Processed images will be appearing with an indefinite delay. To facilitate code organization of

Figure 17.2: Interface of FlickerRemovalTask

the user he is also enforced to send a frame index (unsigned int). With this number a corresponding image will come out.

When the client sends the last image and notifies the library with noMoreData message, the library will still have images inside. It must give everything back. So the library has one asynchronous Port which will be notified after noMoreData message. The library finishes working when all data are given to the client.

Let's now look inside the mainTask (Figure 17.3). Images as well as indexes are stored in two SIPO registers (universal Task) named ImageInputRegister and IndexInputRegister. The sizes of registers are half of that how many neighbors we want to consider in restoration of one frame. Task named Core is a manager of the library. It choose an image from the ImageInputRegister that should be restored. It informs the client which image is going outside through the IdxOut Port. It listenes also for noMoreData message as well as finishes works of whole library by Stop (universal Task). One picture from the ImageInputRegister is chosen by a RegisterDataSelector (universal Task) named here ImageSelector.

Control points are selected basing on the size of input images (Figure 17.4). For that one RegisterDataSelector - OneMatrix and one DataDimensions (universal Task) - MatrixSize were used. A Constant Task (universal Task) - Index is a Selector for OneMatrix.

The main chain of processing is presented in Figure 17.5.
MAPwithInliersEnhancement produce models of flicker f for each pair of images from the ImageInputRegister. If the locality parameter is different than 0, f funcitons are also calculated for each control point. Output functions are marked with the corresponding indexes from the IndexInputRegister.

TemporalRobustMean calculates from f functions a one for the chosen image. Calculations are made for each control point. The last Task is DynamicLUT2D. It takes an image chosen by Core Task and restore it according to the function f. Spatial interpolation of f hasn't been implemented yet. For now, the Task is removing only a global flicker.

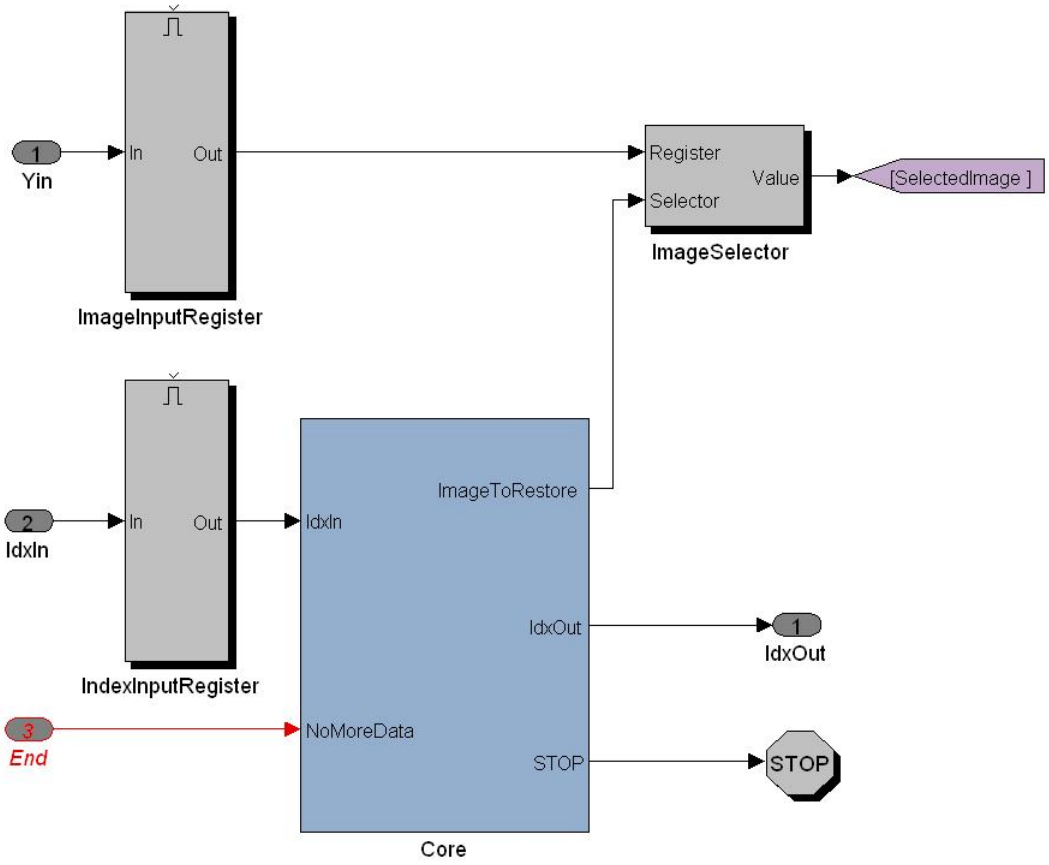


Figure 17.3: Mangement of FlickerRemoval

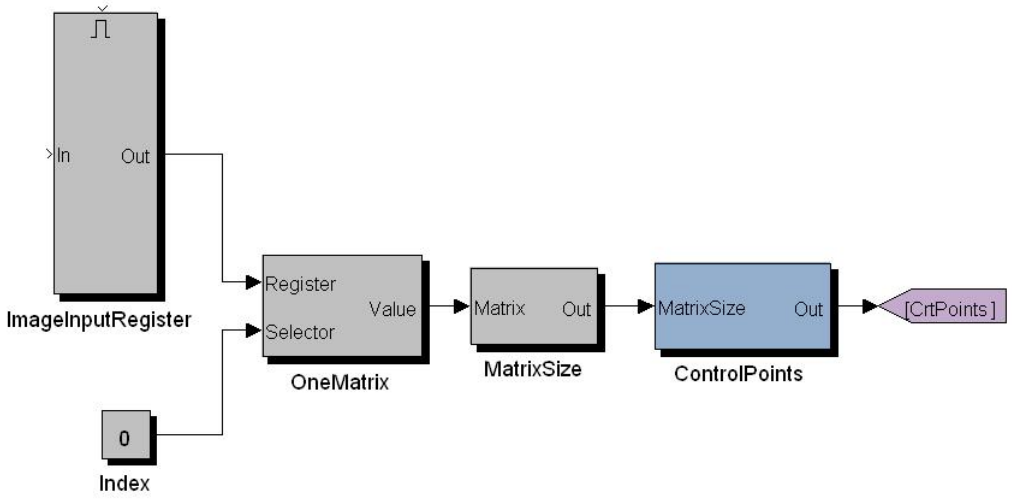


Figure 17.4: Choice of the control points

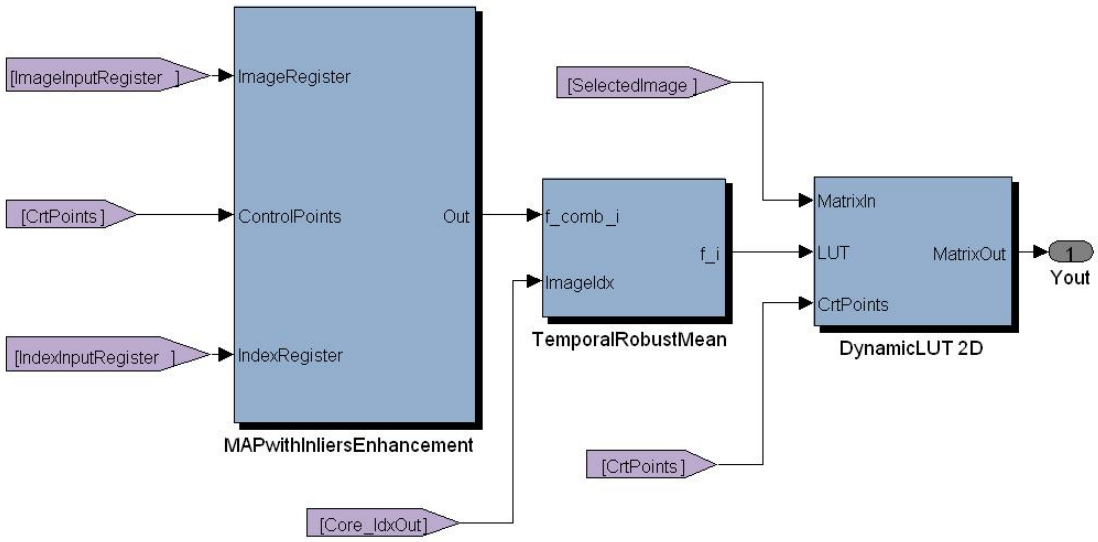


Figure 17.5: Main chain of processing

17.4 MapWithInliersEnhancement Task

MapWithInliersEnhacement is a ComplexTask (Figure 17.6). WeightedLocalJointPdfs is responsable for calculating joint histograms for each combination of input images. This Task will be done every time when new image comes. In order to avoid performing repeatedly the same calculations, only pairs with the first picture are taken into account. After all the joint histogram for the matrix A with matrix B is the same as for B with A. So if the size of ImageInputRegister is for example 8, only 7 calculations will be performed. Calculations are made for each control point.

Every time when new image comes, the WeightedLocalJointPdfs sets (size(ImageInputRegister) - 1) Data in its Out Port. In parallel a pair of indexes are given for which the calculations were made. After, the collection of histograms (for each control point) is processed in block InliersEnhancement. Task Viterbi is creating from each histogram a profile function of flicker f. It should be noted that a normalization of histogram is not needed. Next we combine the results with the indexes in a multiplexer Mux (universal Task) and we put them together in SIPO register fs_i. The size of fs_i should be equal to size(ImageInputRegister)^2. Optional configuration also includes two test blocks. They can directly write appropriate data into a hard-drive.

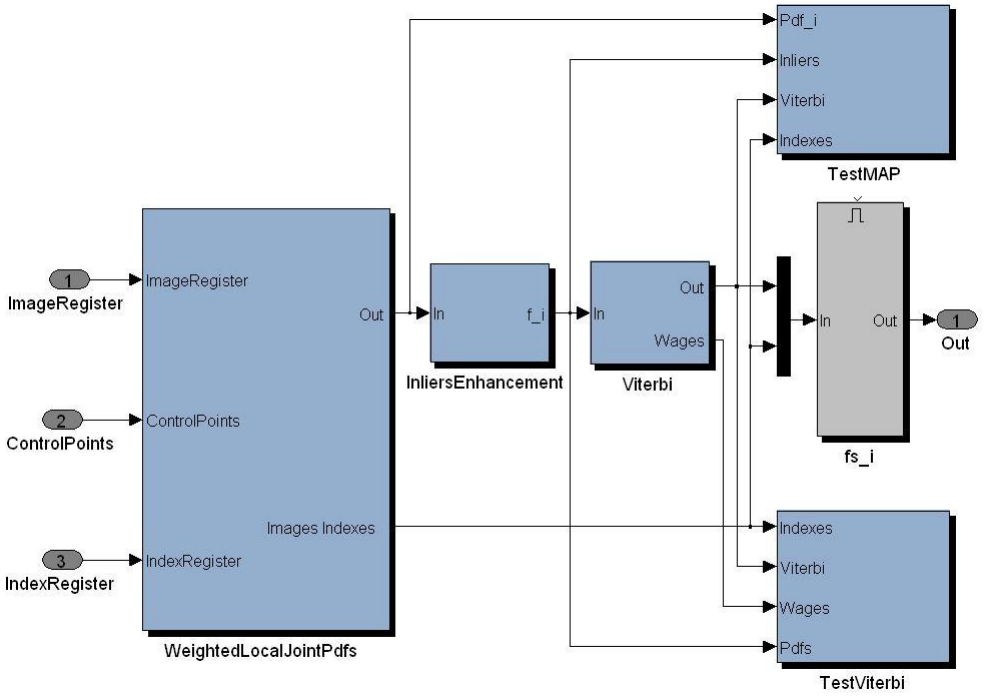


Figure 17.6: MapWithInliersEnhacement Task

17.5 WeightedLocalJointPdfs Task

OneWeightedLocalJointPdf counts joint histogram of two input matrices (Figure 17.7). In the case of localised flicker it creats mask matrix for each control point. In the current implementation it is a simple cone:

$$w_p = \sqrt{height(I)^2 + width(I)^2}/locality \tag{17.1}$$

$$w(\mathbf{x}) = \begin{cases} -|\mathbf{x} - \mathbf{x_p}| + w_p & \text{for } |\mathbf{x} - \mathbf{x_p}| < w_p \\ 0 & \text{others} \end{cases}$$

It should be noted that this block consumes almost 10% of the restoration time when removing global flicker (according to DevPartner tests).

The WLJPmanager is an atomic SimpleTask with a loop (chapter 15.9.1). It means that it could be activated more times after it gets only one complet of input Data-s. Indeed, WLJPmanager is sending all appropriate combinations of images throught two DataRegiterSelector-s.

In an optional configuration, it is possible to connect the testing unit - TESToneWeightedLocalJointPdf.

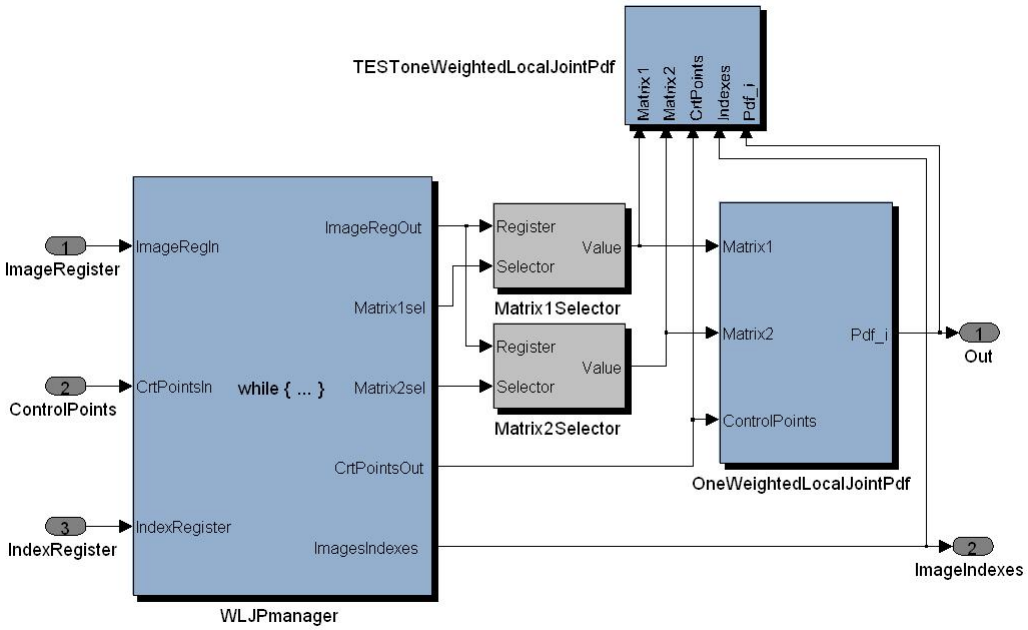


Figure 17.7: WeightedLocalJointPdfs Task

17.6 Viterbi Task

It is worthwhile to discuss this Task. On the one hand it was one of the major challenges to implement. On the other this is a block that takes almost 90% of computation time (according to the results of DevPartner's tests). Any further optimizations should start from here.

Data structure is presented in Figure 17.8. As input we have a joint histogram H - matrix 256x256. As output we need to produce two vectors of size 256 that represents functions $u = g[v]$ and $v = f[u]$. As it was devised in chapter 9.1.2, we will need a temporary structure to hold the "maximal products for now" - matrix W. It should be emphasized that rows of this matrix correspond to the abscissa u in Figure 9.1 (page 58) and columnes to ordinate v. The first state (0,0) occupies the entire first row. The last state (255,255) hasn't got its representation.

Like in appendix B we can distinguish two etaps. First we must fill the matrix W with weights (function fillWeightsTable()). After we can create output vectors (function makeVectors()). The first action is a real bottleneck. Only this function is decided to be discussed. Its activity diagram is in Figure 17.9. Proper implementation is strongly based on pointer arithmetic to fixed-point types. Input arguments are the addresses of beginnings of data structures. Filling the first row with ones is to provide for the following

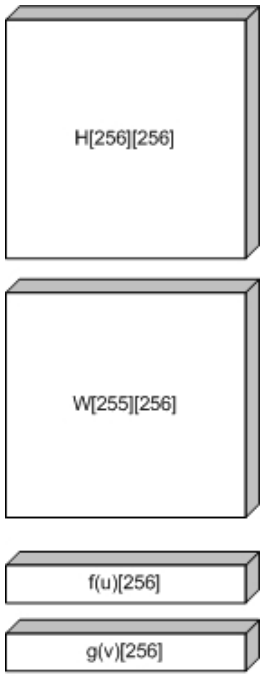


Figure 17.8: Data structure in Viterbi Task

loop neutral conditions at the first iteration. The poiter src_p is initially set on second row of input H. The main loop goes through all the rows of table W. In internal loop W table is filled. First position in every row is set to 0 (Figure 17.10). This helps the second function makeVectors() in its operations and it doesn't affect the results. Selecting a rule to fill a W-table field w depends on whether the corresponding field in H has a value or not.

If "yes" we will look for the maximal value in the previous row of W. But we will look only up to position $w - 256$. In the second example (Figure 17.11), if we want to fill a position $(14, 19)$ we shall look for the maximal values from the row 13 and colums from 0 to 19. Probably this value is 28 on the position $(13, 17)$. It should be noted that this searching procedure is a place where the program spends most time. Next, the found value is multiply by new transition that is corresponding value from the table H (*src_p).

If "no", the answer won't be so obvious. The practice showed that the best solution is when we put the biggest accesible value in graph from Figure 9.1.

The last remaining is "Limit values to X bits". There is a physical limit of multiplication - overflow. We should keep the values from table W in limited number of bits. If the maximal value in row was higher, all values should be

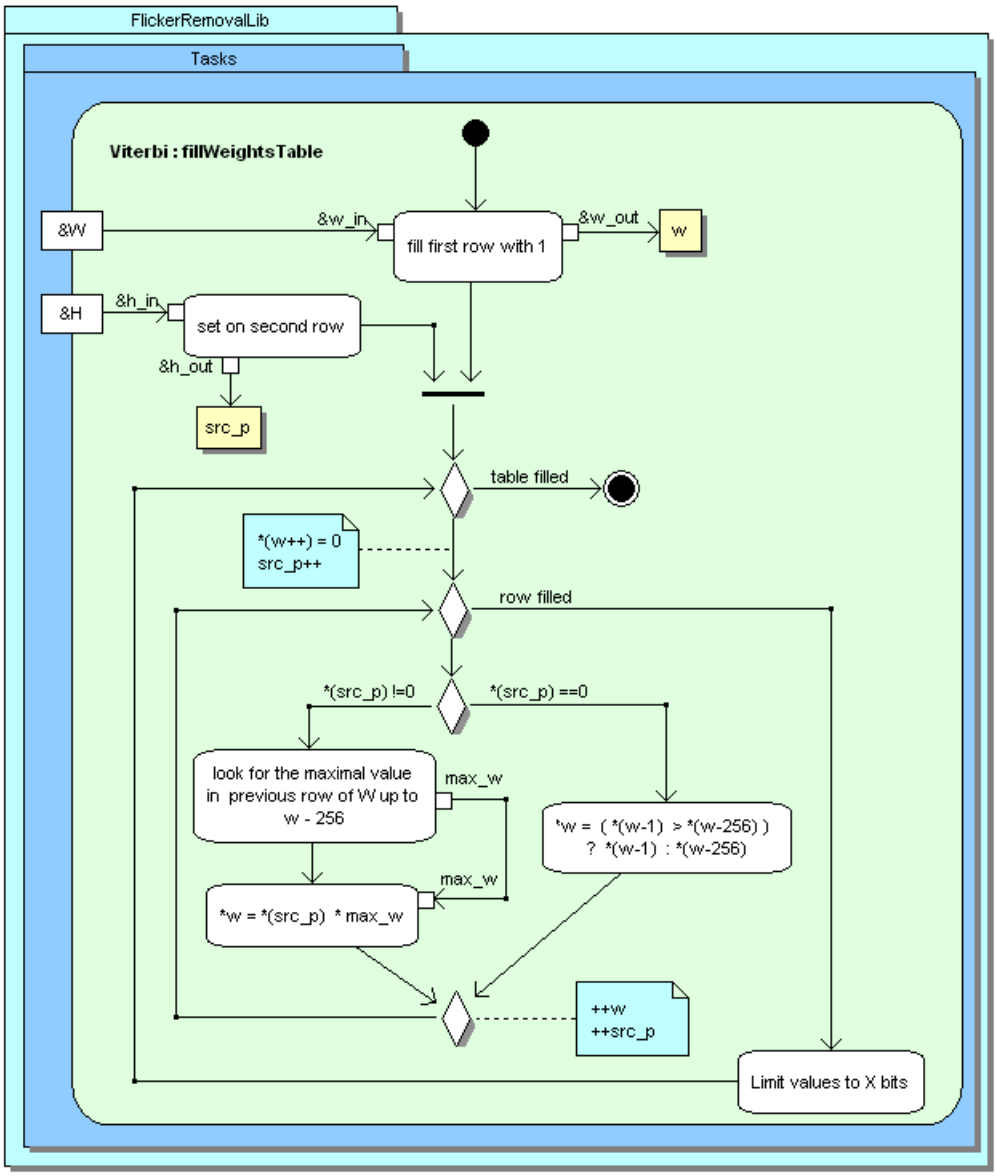
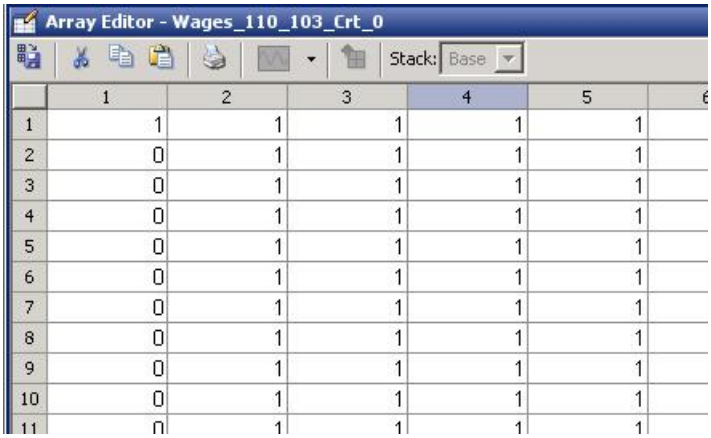
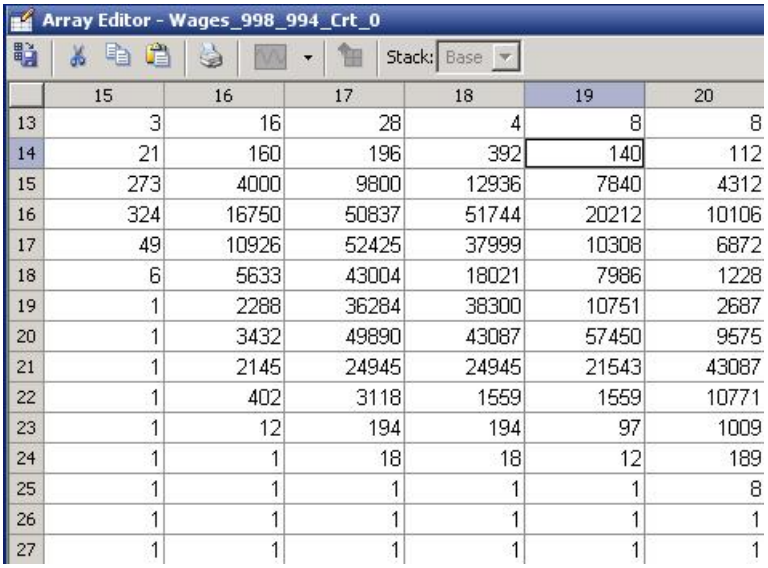


Figure 17.9: Function fillWeigthsTable



	1	2	3	4	5
1	1	1	1	1	1
2	0	1	1	1	1
3	0	1	1	1	1
4	0	1	1	1	1
5	0	1	1	1	1
6	0	1	1	1	1
7	0	1	1	1	1
8	0	1	1	1	1
9	0	1	1	1	1
10	0	1	1	1	1
11	0	1	1	1	1

Figure 17.10: Part of W table - example 1



	15	16	17	18	19	20
13	3	16	28	4	8	8
14	21	160	196	392	140	112
15	273	4000	9800	12936	7840	4312
16	324	16750	50837	51744	20212	10106
17	49	10926	52425	37999	10308	6872
18	6	5633	43004	18021	7986	1228
19	1	2288	36284	38300	10751	2687
20	1	3432	49890	43087	57450	9575
21	1	2145	24945	24945	21543	43087
22	1	402	3118	1559	1559	10771
23	1	12	194	194	97	1009
24	1	1	18	18	12	189
25	1	1	1	1	1	8
26	1	1	1	1	1	1
27	1	1	1	1	1	1

Figure 17.11: Part of W table - example 2

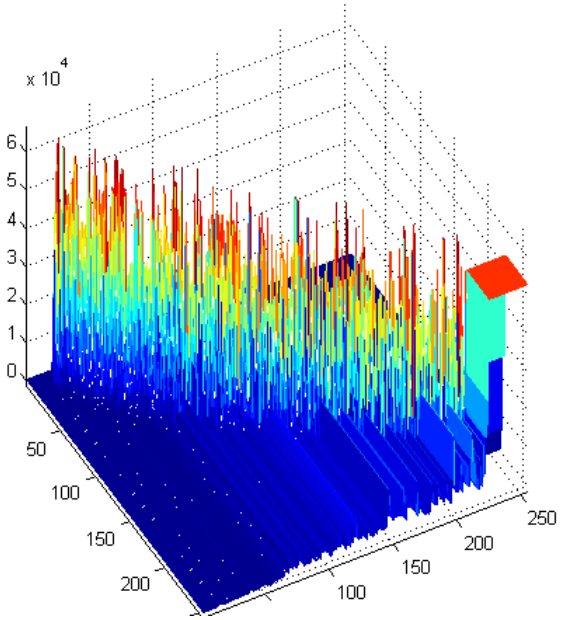


Figure 17.12: W table - example

bit-shifted to the right. Bit operators are extremely faster than division. In Figure 17.11 we can see how the values are within a certain range. The case when it doesn't work is when during shifting we reach 1. It happends for example when there is a lot of small values in one column in H table (effects can be seen also in Figure 17.11). Therefore we loose information about proportion. To prevent this, a mechanism was created which remembers the number of shifts that possibly we would need to adjust the results in the next rows.

We can see an example of graphic representation of W table in Figure 17.12.

17.7 Demo application

A simple console demo application was created that use the developed library. Version avaible on DVD is compiled for Windows OS (FlickerRemovalDemo-Cons.exe). By its interface we can set:

-i filename an input file : it can be any movie file that is supported by FFmpeg,

-o filename an output file : for now it is only a movie weakly compressed with MPEG2 codec,

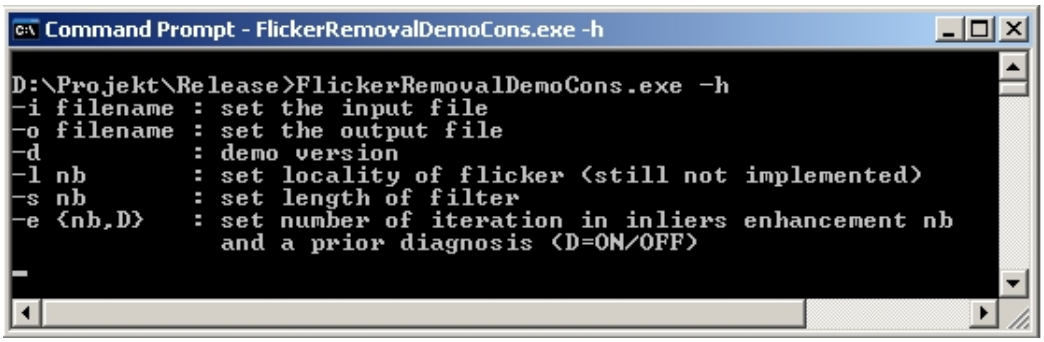


Figure 17.13: Help command of demo application

-d demo version : with this option a movie will be divided verticaly into two parts - one restored and one original,

-s nb number of compared frames during restoration,

-e {nb,D} set number of iteration in inliers enhancement nb and a prior diagnosis (D=ON/OFF);

Support of input and output movie files was made thanks to the open source library - FFmpeg[1].

[1] http://ffmpeg.org/

Part V

Recapitulation

Chapter 18

Summary

18.1 Project

The project was ended with a success. The C++ library exists and the Final Report should serve as a documentation. The whole work took no more than seven months. The additional set goal that my library should be use in the final product can't be checked. The integration process hasn't begun yet. The project shows that I am able to conducte a task from the very beginning (client's need) to the finnish product.

Additionally I gained an extensive experience - especially technical. Clearly visible it could be by the implementation part. Before the internship I had no knowledge of object-oriented systems.

18.2 Problem Understanding

I found and examined 15 articles and one PhD Thesis. They practically cover everything that has been written on the flicker of digitalized old films. To improve the quality of analysis an application was developed that can generate artificial footages. The footages represent typical scenes in movies.

18.3 Design

One from the existing algorithms was chosen to develop that had been promissing the best results. It has been profoundly examined. Most of its components has been tested. In this report I conducted a discussion on the identified weaknesses. I have also proposed some improvements that should increase the quality and performance of the restoration.

The design has covered the mathematical topics such as: Stochastic Processes, MAP and Viterbi algorithm.

18.4 Implementation

During the implementation three packages were developed. First one - Library Package is an universal C++ engine for developing any kind of dataflow libraries. A reflection mechanism has been created that moves programming into modeling level of abstraction. It allows to significantly reduce a "time to market". Second package - DataProcessingLib is an extention to the previous package that allows to easily integrate a created library with already existing application. The last package - FlickerRemovalLib takes advantages of previous packages. The package is an implementation of the designed algorithm for flicker removal. The implementation is characterized high modularity and configurability.

Noteworthy is also the methodology used. It is based on the best practices of software engineering.

Appendices

Appendix A

Presentation of Technicolor

Technicolor SA is the successor of the corporation Thomson SA. The rebranding occurred on 1st February 2010. This is one of the effects of the strategic refocus which has occurred for the last year. The main mission is for Technicolor to be the worldwide leader in services and solutions enabling creation, management and delivery of premium content. The company moves in areas such as film production, postproduction, and distribution. The main customers are content creators, network service providers and broadcasters.

Thomson was a huge and renowned French company in electronic field. Its history is longer than 100 years. It consists of many mergers, partitions and even one nationalization by the French state. The corporation had never made a norrow positioning of the brand. It dealt with many subjects. However, for several years, this part of the company which is today derived by Technicolor had been approaching to the multimedia sector. The crowning of that is taking the company name from one of the acquired businesses. Technicolor is a brand very well known in the Hollywood where the present customers mostly are. For a fan of cinema this name should be associated with the early film coloring technique.

Current Technicolor is a global corporation with over 20 000 employees. The main branches are located in Europe and in the United States. The headquarters is located in Issy-les-Moulineaux near Paris. The buisness revenues for 2009 were €3.529 bn.

A.1 Hannover site

Deutsche Thomson OHG is one of Technicolor's research center. It's placed in Hanover. The main domains are Signal Acquisition and Processing - with huge experince in audio treatment and digital video storage. Hannover is

started to be also one of the development centers for 3D audio/video.

The organizational structure is relatively flat. In addition to functional departments of each typical organization, employees are assigned to research thematic teams.

Appendix B

Viterbi example - Bridge traversal

This example comes from the lecture of Dr. Hubert Kaeslin - *A Gentle Introduction to Dynamic Programming and the Viterbi Algorithm*. It is avaible at site of the University of Cambridge [Cam].

Problem: Four persons want to traverse a suspension bridge at night.

- The bridge can carry no more than two persons at a time.
- The four persons take 5, 10, 20 and 25min respectively to traverse.
- Each party must carry a torch while traversing the bridge.
- The torch lasts no longer than 60 minutes.

Can you help these people? How did you arrive at your plan? Can you formulate a general policy (i.e. an algorithm)?

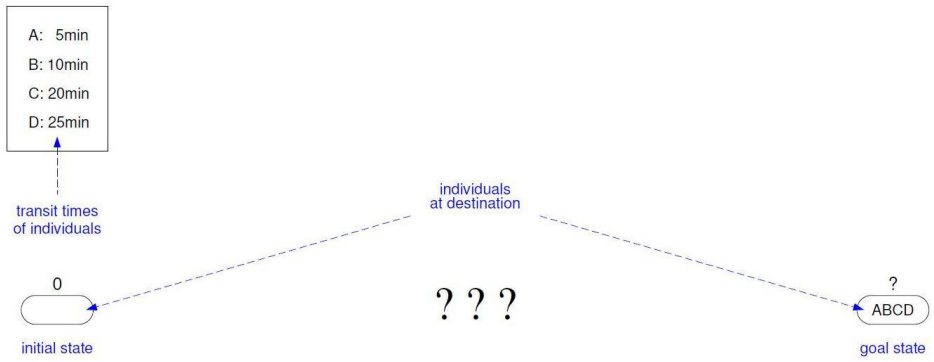


Figure B.1: The problem stated graphically

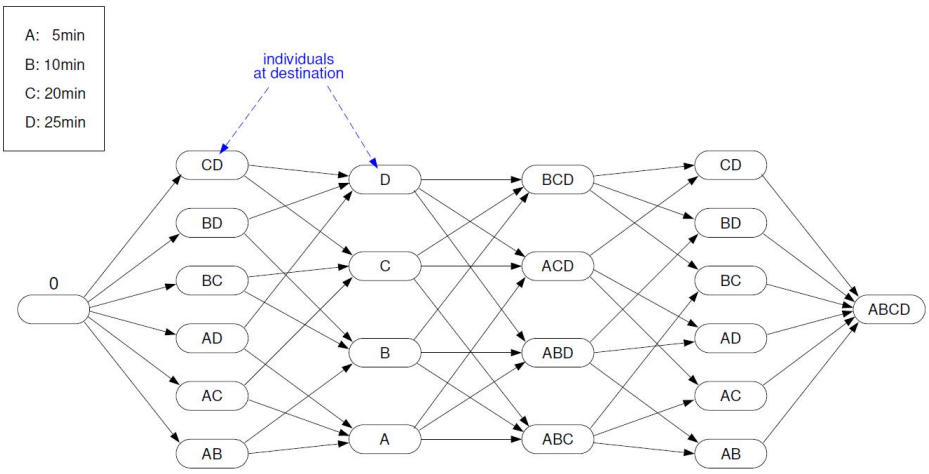


Figure B.2: The solution space drawn as a trellis graph

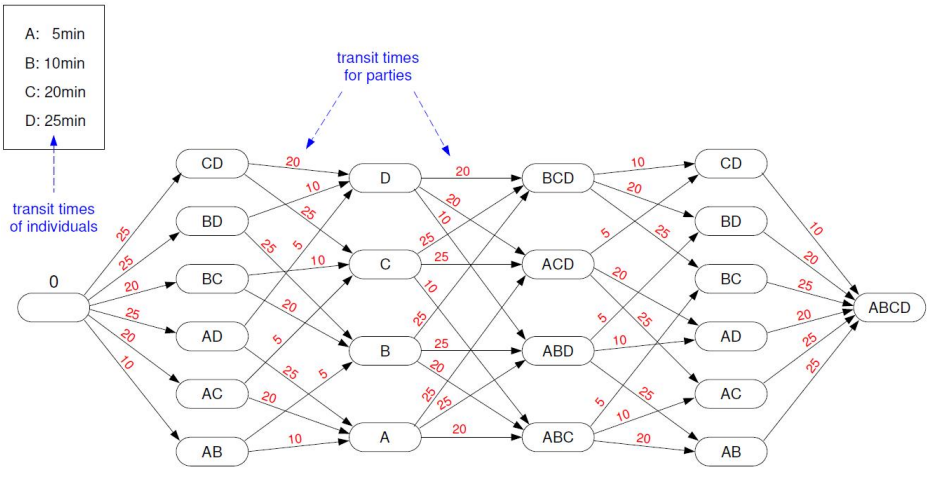


Figure B.3: ... with branch metrics assigned to all state transitions (edges)

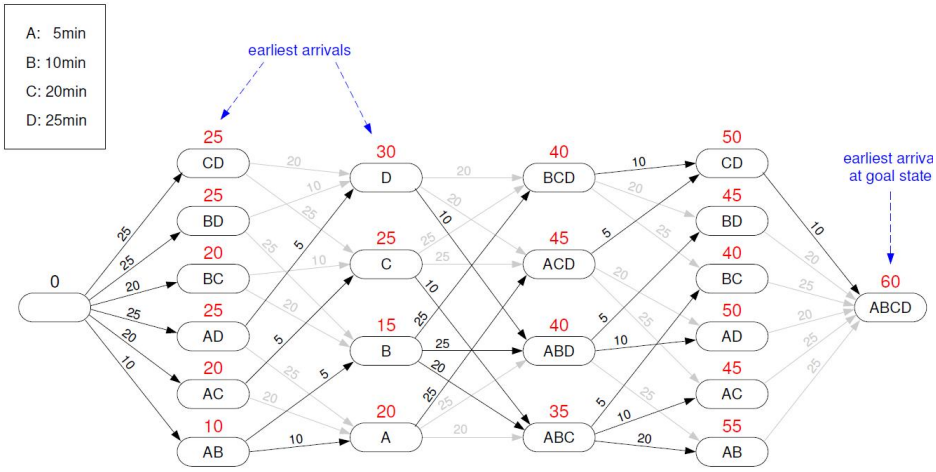


Figure B.4: ... with minimum path lengths updated for all states (nodes)

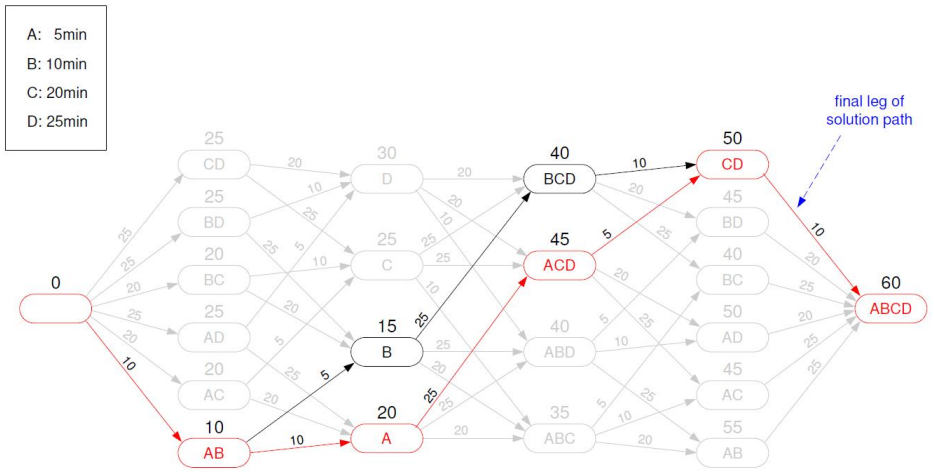


Figure B.5: The surviving trace yields the solution (shortest path through trellis)

Appendix C

Content of DVD

On the accompanying DVD can be found:

flicker examples examples footages from several movies representing real flicker,

use case footages described in chapter 6,

bibliography some articles on flicker - also the implemented [Pit04],

reports documents related to the project,

results some results of performed tests,

programs FlickerGeneratorDemoCons.exe and FlickerRemovalDemoCons.exe with all dependencies,

Bibliography

[Bal] Alexander G. Balakhnin, *http://avisynth.org.ru/deflicker/deflicker.html.*

[BE03] Chuck Allison Bruce Eckel, *Thinking in c++*, vol. 2, 2003, available at http://www.mindview.net.

[Bou] Bouml, *http://bouml.free.fr/.*

[BRA] BRAVA, *D12 - final project report*, avaible on the website http://brava.ina.fr/.

[Cam] Cambridge, *http://www.cambridge.org/resources/0521882672/7934_kaeslin_dynpro_new.pdf.*

[Del06] J. Delon, *Movie and video scale-time equalization application to flicker reduction*, IEEE Transactions on Image Processing **15** (2006), 241–248.

[For] G. Forbin, *http://personal.ee.surrey.ac.uk/ personal/g.forbin/thesis.html.*

[For08] T. Forbin, G. Vlachos, *Flicker compensation for archived film sequences using a segmentation-based nonlinear model*, EURASIP Journal on Advances in Signal Processing (2008).

[For09] G Forbin, *Flicker and unsteadiness compensation for archived film sequences*, Ph.D. thesis, University of Surrey, May 2009.

[FP06] B. Collins A.C.Kokaram F. Pitié, B. Kent, *Localised deflicker of moving images*, Proceedings of the 3rd IEE European Conference on Visual Maedia Production (CVMP'06) (2006), 134–143.

[H.D03] A.C.Kokaram R.Dahyot F.Pitié H.Denman, *Simultaneous luminance and position stabilization for film and video*, Image

and Video Communications and Processing **5022** (2003), 688–699.

[Hub09] Peter J. Huber, *Robust statistics*, 2nd ed., John Wiley & Sons, Inc, Hoboken, New Jersey, 2009.

[IPP] Intel IPP, *http://software.intel.com/en-us/intel-ipp/*.

[Joa] Joanneum, *http://diamant.joanneum.at/film_ restoration/*.

[NA00] Valery Naranjo and Antonio Albiol, *Flicker reduction in old films*, Proceedings of IEEE International Conference on Image Processing ICIP'00 **2** (2000), 657–659.

[NMM] Motama NMM, *http://www.motama.com/nmm.html*.

[Ope] OpenCV, *http://opencv.willowgarage.com/wiki/*.

[OSKS00] T. Ohuchi, T. Seto, T. Komatsu, and T. Saito, *A robust method of image flicker correction for heavily-corrupted old film sequences*, Proceedings of IEEE International Conference on Image Processing (ICIP'00) **2** (2000), 672–675.

[Pit04] R. Dahyit F. Kelly A.C. Kokaram E. Pitié, *A new robust technique for stabilizing brightness fluctuation in image sequences*, Proc. ECCV Workshop SMVP (2004), 153–164.

[Pol57] George Polya, *How to solve it - a new aspect of mathematical method*, 2 ed., Princton University Press, New Jersey, 1957, avaible at http://www.scribd.com/doc/3158079/Polya-How-to-solve-it.

[Pro] The Code Project, *www.codeproject.com/ kb/cpp/smartpointers.aspx*.

[RJ94] ErichGamma RichardHelm RalphJohnson and JohnVlissides, *Design patterns: Elements of reusable object-oriented software*, AddisonWesley, 1994.

[ROO99] Peter Michael Bruce VAN ROOSMALEN, *Restoration of archived film and video*.

[Sig] Sigmedia, *http://www.mee.tcd.ie/ sigmedia/*.

[SPH99] P. Schallauer, A. Pinz, and W. Haas, *Automatic restoration algorithms for 35mm film*, Videre **1** (1999), no. 3, 60–85.

[The] Theseus, *http://www.theseus-programm.de.*

[Tre05] G. Forbin T. Vlachos S. Tredwell, *Spatially adaptive flicker compensation for archived film sequences using a nonlinear model*, Proceedings of the 2nd IEE European Conference on Visual Media Production (CVMP'05) (2005), 241–250.

[Tre06] G Forbin T Vlachos S Tredwell, *Flicker compensation for archived film using a spatially-adaptive non-linear model*, ICASSP 2006 (2006).

[TSTK00] T. Ohuchi T. Seto T. Saito T. Komatsu, *Image processing for restoration of heavily-corrupted old film sequences*, 15th International Conference on Pattern Recognition (ICPR'00) **3** (2000), 3017.

[Vla04] T. Vlachos, *Flicker correction for archived film sequences using nonlinear model*, IEEE Transactions on Circuits and Systems for Video Technology **14** (2004), 508–516.

[Vla07] T. Tredwell G. Forbin T. Vlachos, *Flicker compensation for archived film using a mixed segmentation/block-based nonlinear model*, 15th European Signal Processing Conference (EUSIPCO) Poznan, Poland, 2007, pp. 135 – 139.

[Vla08] G. Vlachos, T. Forbin, *Nonlienear flicker compensation for archived film sequences using motion-compensated greylevel tracing*, IEEE Transactions on Circuits and Systems for Video Technology **18** (2008), no. 06, 803 – 815.

[vRJBRL00] A. Hanjalic G.C. Langelaar P.M.B. van Roosmalen J. Biemond R.L. Lagendijk, *Image and video databases: Restoration, watermarking and retrieval*, Advances in Image Communication, vol. 8, ELSEVIER SCIENCE B.V., 2000.

[vRRLLJB99] P.M.B. van Roosmalen R. L. Lagendijk J. Biemond, *Correction of intensity flicker in old film sequences*, IEEE Transactions on Circuits and Systems for Video Technology **9** (1999), 1013–1019.

[WD95] Y. Wu and D.Suter, *Historical film processing*, Applications of digital Image Processing XVIII, vol 2564 of Proceedings if SPIE (1995), 289–300.