Design and Implementation of Efficient Diminished Reality Mechanisms

Jon Arrieta Etxeberria

June 3, 2016

Contents

List of Tables

List of Figures

Acronyms

VR: Virtual Reality.

AR: Augmented Reality.

DR: Diminished Reality.

HD: High Definition.

ROI: Region Of Interest.

SSD: Sum of Squared Differences.

DOF: Degrees of freedom.

PDF: Probability Distribution Function.

OBB: Oriented Bounding Box.

WP: Work Package.

FPS: Frames Per Second.

Trilingual summary

Virtual Reality (VR) as well as Augmented Reality (AR) have awaken the interest in the society in the last few years. Big companies like Google (with Google Glass) or Microsoft (with HoloLens) are already betting for these technologies by developing their own hardware and putting them in the market. Diminished Reality (DR) can be considered as one of the branches of AR, that instead of adding objects to the user view (like AR), it removes them from it. DR, combined with AR, can create an interactive virtual environment that alters the reality. After analysing and evaluating the existing DR techniques, a need of a system capable of propagating structures for video has been found. To the best of our knowledge this is the first feasible approach to implement DR technology capable of propagating structures as well as textures efficiently in video applications. Apart from that, a performance analysis has been done to proof that the optimizations introduced were efficient to adapt the solution for video applications.

La Realidad Virtual (VR en inglés) tanto la Realidad Aumentada (AR en inglés) han despertado el interés en la sociedad en los últimos años. Grandes compañías como Google (con Google Glass) o Microsoft (con HoloLens) están apostando por estas tecnologías desarrollando su propio hardware e introduciéndolo en el mercado. La Realidad Disminuida (DR en inglés) puede ser considerada como una de las ramas de la AR, donde en vez de añadir objetos a la vista del usuario (como la AR), se eliminan de ella. DR, junto con la AR, puede crear entornos virtuales interactivos capaces de alterar la realidad. Después de analizar y evaluar las tecnologías DR existentes, se identificó la necesidad de un sistema capaz de propagar estructuras en aplicaciones de video. Hasta donde se conoce, este proyecto presenta la única tecnología de DR capaz de propagar texturas tanto estructuras de una manera eficiente en aplicaciones de video. Además, se ha realizado un análisis de rendimiento del sistema que comprueba que las optimizaciones aplicadas para video son eficientes.

Errealitate Birtualak (VR ingelesez) eta baita Errealitate Areagotuak (AR ingelesez), gizartean interesa piztu dute azken urte hauetan. Google (Google Glassekin) edo Microsoft (HoloLensekin) bezalako konpainia handiek, jada beren apustua egin dute beren hardware propioa garatuz eta merkatuan sartuz. Errealitate Txikiagotua (DR ingelesez), AR-ren adarretako bat bezala kontsideratu daiteke, erabiltzailearen bistara objektuak gehitu beharrean (AR), kendu egiten dituena. AR DR-rekin konbinatuz gero, errealitatea aldatzeko gai den ingurune birtual interaktibo bat sor daiteke. DR-aren inguruan gaur egun existitzen diren teknologiak ikertu eta ebaluatu ondoren, bideoan egiturak hedatzeko gai den sistema baten beharra ikusi zen. Ezagutzen den arte, proiektu honetan aurkezten den DR teknologia, egiturak hedatzeko baita texturak efizienteki hedatzeko gai den soluzio bakarra da. Honez gain, proposatutako sistemaren errendimendu analisi bat burutu da biderorako egokitzeko egindako optimizazioak efizienteak direla frogatzen dituena.

Part I

REPORT

1 | Introduction

The last improvements that have been done in technology, and specifically in virtual reality, have made the concepts like Augmented Reality (AR) sound realistic and not something futuristic. Even if the first applications of Virtual Reality (VR) that come to our minds are mainly for gaming, there are already some implementations for e-learning, medicine or architectural design. Actually, VR makes the user feel immerse in a virtual world, and this virtual world can be a school, a hospital or whatever it is needed in each application. Therefore, there are applications to train possible surgeons [5], that create virtual cadavers instead of using real ones. This way, costs are reduced because instead of using a cadaver for each practicum a VR machine is used.

Although almost all the people have heard about AR just few of them know what it is exactly. AR is a mechanism that enriches the real world by adding virtual elements to it. A typical example could be the Google Glass (figure 1.1), where the user is able to see everything normally (real world), but it is also able to see virtual content (messages, videos or images) at the same time.

Figure 1.1: Google Glass.

On the contrary, Diminished Reality (DR) basically does the opposite effect of AR. The objective of DR is to remove undesired objects from the image, video or user view. For example, there are some cases where there is a person in a video who should not appear in it. Nowadays, they blur the face of that person, but this technique is not 100% efficient because this person can be recognised by its clothes, movements, etc. Despite DR can be applied in images, the concept is used more for video applications. Hence, it is not enough to adapt a frame, it also requires object detection and tracking steps that are explained in the following page. Although existing research in DR techniques is not vast, these techniques can be classified in two groups:

1. **Multi-view based methods:** They use widely separated images taken from different viewpoints (using multiple cameras) in order to see what is in the background of the object and reconstruct the image with certainty. These methods are not common in real world scenarios because there are few cases where the user has more than one camera. The technique presented by Zokai *et al.*[6] is one of the examples of this kind of inpainting techniques.

2. **Frame based methods:** Consists in using information from a single camera to reconstruct the selected area. Some methods like the proposed by Criminisi *et al.*[3] only process the actual frame (they are more oriented to be used with image editing programs). Others like Wexler *et al.* [7] use previous and next frames to recover the final image, so they are only valid for video post processing. Finally, solutions like Herling *et al.* [8] focus on implementing DR in real-time, so they just employ actual and previous frames.

According to the processes involved in DR for video applications, they can be divided in three steps:

1. **Object identification:** It is basically the part where the user selects an area of the image to be reconstructed. This area is translated to a mask that wraps the object of interest and which has to be tracked in the following frames.

2. **Object tracking:** Consists in following the object of interest in the subsequent frames. The tracking can be based on different visual cues, such as color matching or the movement of the keypoints that lie on the surface of the object.

3. **Image inpainting:** Consists in reconstructing the user selected area, based on the information given by the frame(s).

Even though optimizations can be done in all of these three phases, this project has mainly focused on the improvement of the image inpainting phase. Object identification and tracking modules have already been optimized for another AR applications with the objective of being robust and fast for real-time applications. The problem of actual inpainting techniques is that they are too slow. Existing techniques have made efforts to improve the visual quality of the results without taking care of the processing time. Consequently, it is impossible to implement these techniques directly to process a live video because they would take too much time to process each of the frames.

The objective of this project is to start with a simple prototype of DR, adding more complexity until a prototype with a sophisticated inpainting method is implemented. Special emphasis will be placed to guarantee that the processing is light, trying to get a solution that is close to real-time. After this final prototype, some tests will be done to measure and quantify the effects of each optimization.

2 | Objectives and Scope

The main objective of this project is to develop and implement a diminished reality engine that is capable of working close to real-time. As it is mentioned in the introduction, diminished reality in video applications is divided in three major processes: object detection, object tracking and image inpainting. There are several methods for each one of the phases that fit better or worse depending on the scenario. In this document a study and comparison between some of these techniques is included to present a complete view of the problem.

A philosophy based on incremental prototypes has been followed to ensure the correct achievement of the project. More precisely, the following development steps have been considered:

1. **Controlled scenario with simple background:** Consists in implementing diminished reality in a controlled environment with a simple background in order to create a simple prototype. For example, occluding an object that is in a table with simple color and no structures. For this first approximation, "Camshift" and "inpaint" techniques can be used for tracking and image registration respectively. Both of this methods are included in OpenCV library [9].

2. **Improvement in the reconstruction technique:** Consists in introducing more complexity to the approach trying different combinations between object detection, tracking and inpainting. The aim is to create a modular code where the user can change the input parameters to choose different configurations for each scenario. Adaptation of the algorithm to get closer to real-time will be applied here as well.

3. **Study of different optimization algorithms:** Consists in performing tests with different configurations that will show the performance of the implemented inpainting algorithm. Processing time as well as visual quality will be the parameters to be evaluated.

It is clear that the inpainting process will be crucial for this project, because it needs to be fast, efficient and robust for any kind of environment. The problem is that high-quality image inpainting techniques are not fast enough for real-time applications, because these methods have been designed for images and not for videos. There are methods that require less computational time like pixel interpolation, for example. However, the result is not appropriate for textured or structural image reconstruction. Barnes *et al.* [2], Criminisi *et al.* [3] and Wexler *et al.* [7] describe some interesting designs for the image inpainting process that can be applied in this project, but they need some optimizations in timing in order to fulfill the objective of a real-time approach.

Even though the final design is open, the final prototype should implement at least an inpainting method that is able to propagate texture close to real-time. In this prototype, the user selects an object, and it is tracked and inpainted during the whole process automatically and close to real-time.

3 | Benefits

The benefits of this project can be classified in three parts: technical benefits, economical benefits and social benefits.

3.1 Technical benefits

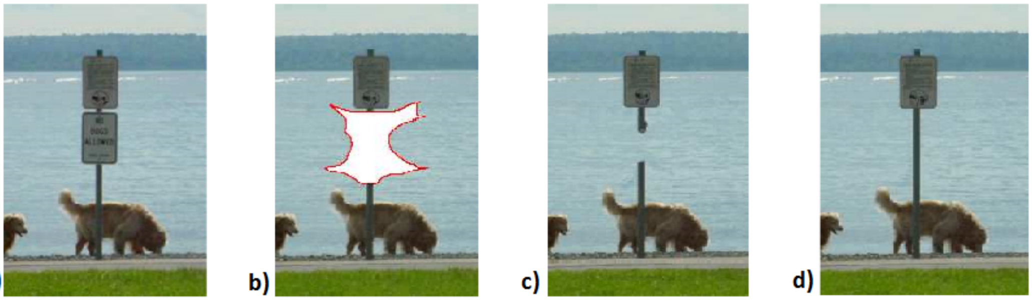
The proposed prototype is one of the few DR designs for video editing. Moreover, it is close to real-time and implements a similar technique presented by Criminisi *et al.* [3], which makes this project unique. Using this approach, it is possible to reconstruct an image preserving structures with no more user interaction than selecting the object of interest.

However, as the aim of the project is to implement this technique for video editing, the process needs to be modified in order to get closer to real-time applications. This project contains not only the design of the solution, but also a study between different optimization techniques that can be useful for future applications. It includes also quantitative results for each optimization with different configurations and scenarios. Thus, strength and weaknesses of our approach are identified, and new research areas are opened.

AR is a technology that has several years of investment and DR can be considered as a new branch of AR. As it is a new concept, there is still not much research in the area. There are some image editing tools like Photoshop CS [10] or GIMP [11] that are well suited for any kind of operations (inpainting included). Nonetheless, taking a look at existing video editing applications, there is nothing about inpainting or object occlusion. It is possible to extract the frames, modify as images and introduce them again to the video, but if the object is in a considerable number of frames this procedure is not viable (too time-consuming). Using the solution that is presented here, a reconstruction of a video can be done with minimal user interaction and much less processing time.

Therefore, the evolution of these techniques in combination with AR offers a huge opportunity to develop new applications. For example, if a reconstruction of a room is needed, the designer makes a 3D prototype which is shown to the customer. This 3D modelling takes quite a lot of time, and it can take even longer if the customer disagrees with the prototype. Using a combination between DR and AR it is possible to change the colour of the walls, move the position of the window and so on. All this could be done at an interactive time with user interaction, so the customer would decide everything at the moment.

An attempt to achieve such an objective has been conducted by Herling *et al.* [8], where the inpainting technique developed by Barnes *et al.* [2] is used. Even if the method designed by Barnes is faster than the one that is used in this project (Criminisi *et al.*[3]), it doesn't take into account the structural information. This leads to a bad reconstruction of borders when an object is between two textures. Figure 3.1 shows the importance of the filling order when a structural part needs to be recovered (Images are taken from [3]):



a) b) c) d)

Figure 3.1: **Importance of the reconstruction order.** a)Original image. b) The target region has been selected and marked with a red boundary. c)Result of filling by concentric layers. d)Results with Criminisi's algorithm

As far as is known, the implementation of the proposed mechanism for DR is the first working solution that is able to propagate structures as well as textures to real-time. Tests were performed using High Definition (HD) quality videos.

3.2 Economical benefits

As this project is not targeted for a certain application, its economical benefits can be different from one case to another. In this part, benefits that are shared in most of the cases are defined, and some specific examples are proposed.

Taking into account that the cost of the project is also based in the amount of people that works on it, a company would reduce its costs noticeably comparing to the existing techniques because less user interaction is needed. This is due to the automatic detection, tracking and inpainting systems implemented in this project.

However, it is easy to imagine several areas where this technique can be implemented, and it will be wider when more mature inpainting processes are developed. For example, it can be applied in a video editing environment, where the customer has a video with an undesirable object and wants to remove it from all the frames. There is no application on the Internet to solve it nowadays, and the only way to do it is by reconstructing frame by frame. Then, the costs of the project raise considerably and it is normally not viable to do it.

Another possible application that it is mentioned before is the one for reconstructions. The designer needs to create 3D models for customers, and every time a customer wants to change an object the designer needs to modify the 3D model again. On the other hand, using the proposed approach combined with AR the customer and the designer can make the design with little interaction. This reduces the costs of a designer and fastens the process at the same time, becoming a win-win situation.

Additionally, a technology that is becoming more and more popular nowadays is the 360° videos. When a user is creating panoramas or 360° images, it is common to have some gaps that need to be filled. It is quite easy to fill this gaps in an image, but what happens if the same concept is used to process a video? There are some existing software products like Kolor Autopano Video [12] that stitches the images in order to create the ° videos. This is another possible market that this project could work in.

Finally, it is known that AR is growing really fast these days, together with the release of different devices like Oculus Rift or Google Glass it is easy to guess that big companies are betting for this technologies. Therefore, this project can be a great add-on for the existing tools, creating a more complete system.

3.3 Social Benefits

The impact of this technique in the society is still uncertain. As mentioned before, it is a project that doesn't target a specific market, and then, it is hard to guess how the society is going to integrate it in different applications.

DR could be used to detect and remove offensive or +18 content from the user view. For example, children below 18 could have an adapted version of the movie that is automatically generated. This application could be interesting specially for tv channels or cinemas.

DR can also be used for surgeons, for example. In this case, the surgeon would just see the tools and would be able to see what is behind the hands. It is not possible to implement this with frame based inpainting techniques, as there is no certainty of the reconstructed background, but is affordable using video inpainting methods.

4 | State of the Art

The following section provides a brief introduction to the existing DR techniques. There are various ways to implement DR with different processing times, inpainting qualities and amount of cameras.

First of all, techniques will be classified in order to have a general perspective, and afterwards, there is a description and comparison of each one of the techniques. Classification is done based on three characteristics: number of cameras or views, reconstruction methods and the media.

There are basically two options based on the number of cameras or views:

1. **Multiview-based methods:** Zokai *et al.* [6] and Lepetit *et al.* [13] approaches are examples of this kind of method. They use cameras or frames from different points of view in order to build the background of the object that needs to be occluded. The result is almost perfect because these techniques have the information of the background, so the only thing they need to do is pair the frames and reconstruct the information. The weakness of these methods is that there must exist several cameras recording from different points of view.

2. **Frame-based methods:** Most of the existing methods like Barnes *et al.* [2], Criminisi *et al.* [3], Wexler *et al.* [7], Herling et al. ([8], [14]) and Kawai et al. ([1], [15]) are approaches that are based on the information of a single camera. Generally, they process the current frame, although sometimes they include previous frames as well ([8], [14]), to inpaint the selected object. Even if this technique is not as accurate as the previous one, it is applicable in much more scenarios.

According to the reconstruction methods they can be classified in three categories:

1. **Pixel based inpainting:** This technique reconstructs the area pixel by pixel. Approaches like the ones presented by Herling *et al.* ([14]) and Kawai *et al.* are based on image information to recover the area pixel by pixel. On the other hand, it is possible to implement a pixel interpolation based on the pixels surrounding the mask. The results

of the pixel interpolation are acceptable for small areas, but degrade considerably with the size of the area.

2. **Patch based inpainting:** Barnes *et al.*,Criminisi *et al.*, Herling *et al.* ([8]) and Wexler *et al.* find patch correspondences in the whole image that match with the patch that needs to be inpainted.

3. **Image matching:** Lepetit *et al.* and Zokai *et al.* find the matching between two or more frames with different points of view, and this way, they are able to guess what is behind the object that needs to be occluded.

The last classification corresponds to the media that needs to be processed:

1. **Inpainting for video:** Herling *et al.* ([8], [14]), Kawai *et al.* ([15], [1]) and Wexler *et al.* explain different methods to implement DR for video applications. Herling *et al.* focus on applying DR for real-time applications, whereas Kawai *et al.* explain some techniques to get better results in the final video. Finally, the method proposed by Wexler *et al.* is more oriented to video post-processing, focusing mainly on giving both spatial and temporal coherence to the frames.

2. **Inpainting for images:** Barnes *et al.*, Criminisi *et al.*, Lepetit *et al.* and Zokai *et al.* propose some of the techniques that are oriented to image inpainting. Lepetit *et al.* and Zokai *et al.* are designed for multiple view scenarios and the result is more accurate than in Barnes *et al.* and Criminisi *et al*. But, as previously explained, a multiview scenario is not common in a real life scenario.

All this articles can be grouped using different criterias. In this state of the art, the classification of camera quantities or views is used in order to differentiate the most and least similar articles.

4.1 Multiview-based methods

In the following section, some of the well known multiview-based articles are explained. At the end of each article, there is a review remarking its advantages and disadvantages.

Lepetit *et al.* [13]

The article presented by Lepetit *et al.*, was one of the first articles in the area of DR. They argue that the lack of accuracy of the existing tracking methods for 2D processing results in a bad reconstruction of the image, especially in cluttered environments. On the other hand, the advancements done in computer vision make it possible to use 3D stereo-based reconstructions to segment the image and determine the occluding object. In this case, the camera viewpoint is computed based on the frame sequences. A matching error between two or more frames would result in a bad computation of the camera viewpoint, and therefore, a bad detection of the occluding object's position.

Figure 4.1 is a scheme used by Lepetit *et al.* to explain the basics of their algorithm. The user needs to select the keyframes manually from the video sequence. The keyframes are selected each time the aspect of the occluding object is changed. Then, the occluding object is selected manually in each one of the key-frames. Once those steps are done, a 3D occluding object approximation is calculated between each pair of keypoints in the image. And finally, this 3D approximation is refined to determine the 2D boundary of the occluding object.

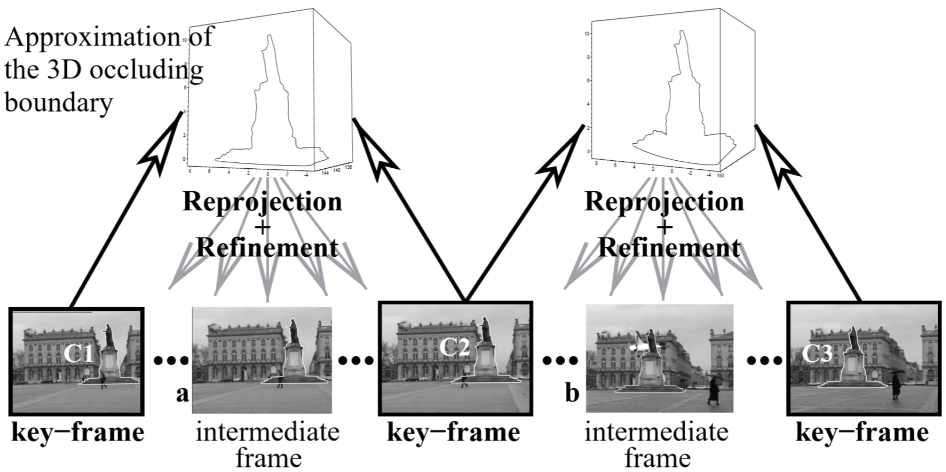


Figure 4.1: Scheme proposed by Lepetit *et al.*

Regarding the reconstruction of the occluding objects area, they employ two methods depending on the area. If the area covered by the object is thin, interpolation is used to fill the region. But if a big area needs to be reconstructed, interpolation results are not acceptable, and consequently another method needs to be used. In order to overcome this problem, Lepetit *et al.* use Delaunay algorithm [16] to recover the background image. The main problem of using triangulation (Delaunay algorithm) comes when there is a triangle that is occluded in all the frames. In this case, there is no way of mapping a texture, and therefore interpolation is used.

Summarizing, the technique presented in this paper is one of the referents according to multiview techniques. Even though almost perfect reconstructions can be done for specific scenarios, this algorithm presents some problems when the object is close to the background. In this case, the triangulation would fail, and interpolation would be used in the whole object obtaining a non-acceptable result. Apart from that, this algorithm requires excessive user iteration making it impossible to implement it for large videos.

Zokai *et al.* [6]

Zokai *et al.* proposed a new system for DR in where they used widely separated images from different viewpoints in order to regenerate the background. They assume that images need to be calibrated first, no matter if they use marker or markerless techniques. Once setup is prepared, the scenario should look like the one in figure 4.2.

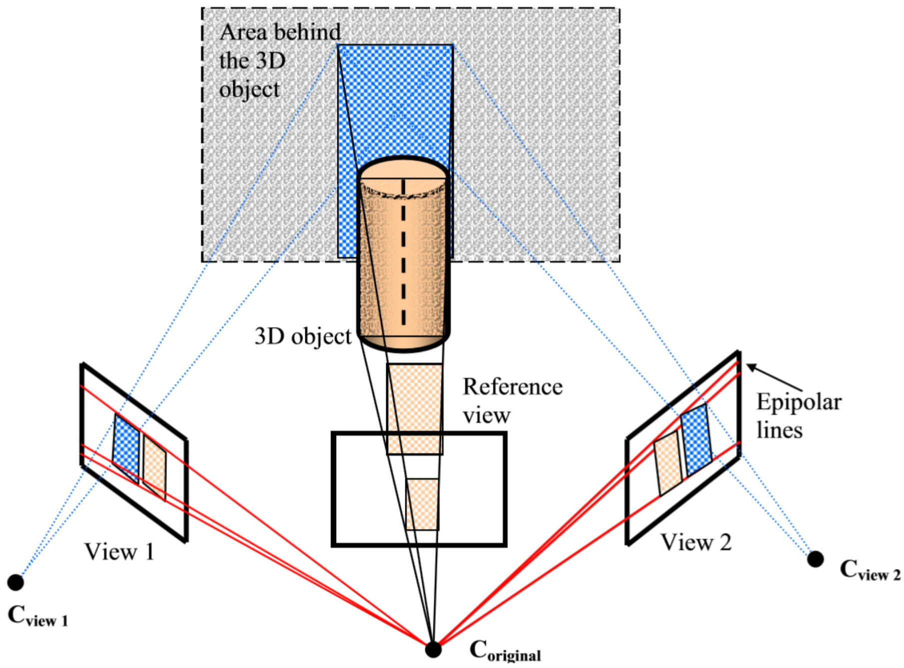
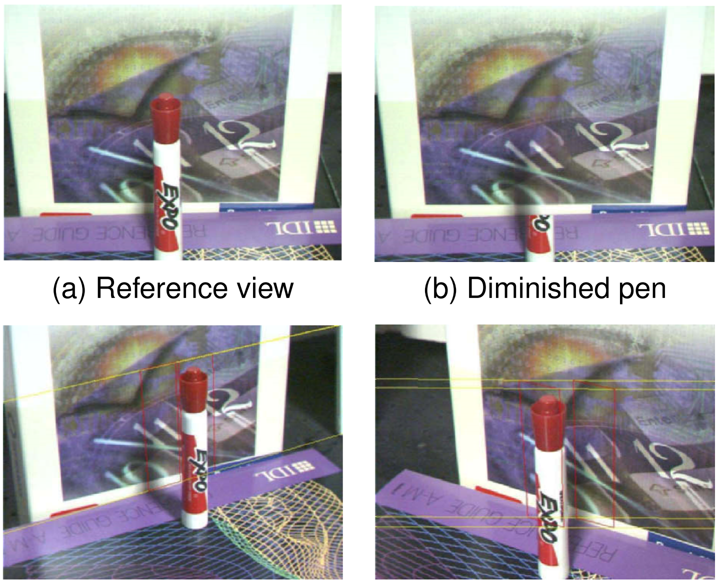


Figure 4.2: Reference scenario presented by Zokai *et al.*

In figure 4.2, the "C"s represents the centers of the cameras. The "View"s are the 2D projections of the scenario. The yellow cylinder represents the object that needs to be occluded and the yellow rectangles are the virtual planes generated from the user selection. And finally, the blue rectangle is the background image that needs to be recovered.

Compared to the approach presented by Lepetit *et al.*, in this case the user just needs to select the object in the reference view (in which the object is occluding the desired area). From this selection, the algorithm automatically generates virtual planes parallel to the image plane. Each plane defines a set of homographic transformations between its images in all the calibrated views (View 1 and View 2 from figure 4.2 e.g.). This homographic transformation is used to detect the image of the virtual plane in the calibrated views, and replacing them in the reference view for the background reconstruction.

In their approach they define solutions for three kinds of scenarios: planar background parallel to reference image plane, planar background with arbitrary orientation and non-planar background. The first case is the perfect scenario, where the resulting image would be ideal. In the second case, images from the calibrated views are rectified and then passed through a registration step to recover the distortion introduced by the orientation. Finally, in the third case, as each part of the reconstruction area will have a different depth, it can't be recovered as is. To overcome this problem, Zokai *et al.* propose a method where the Region Of Interest (ROI) selected by the user is subdivided in different tiles which are defined with different depths.



(a) Reference view (b) Diminished pen

(c) View 1 (d) View 2

Figure 4.3: Zokai *et al.* inpainting example.

Figure 4.3 shows how this algorithm reconstructs the background image in an efficient and visually accurate way. The major problem of this technique is that they use more than one camera view to get the final result, and this is not a common scenario in a real world. Apart from that, this approach as well as the one designed by Lepetit *et al.* need the object to be apart from the background. This is an important weakness of these techniques as it would be impossible to remove a picture, scratch or similar from the wall.

4.2 Frame-based methods

Sophisticated frame-based methods are usually exemplar-based techniques. These kinds of techniques make use of patches that are copied from one area of the image to another one. As this subsection includes some technical concepts, each of these concepts needs to be defined for the correct understanding of the document:

- **Target region:** It is also called mask or masked region. It is the area where the object that needs to be removed is located.
- **Source region:** It is the area that contains useful information for the reconstruction of the image. Source patches always come from the source region.
- **Patch:** It is a variable size pixel matrix that is used to compare different regions in the image.
- **Target patch:** It is the area that is going to be replaced with a patch from another region of the image.
- **Source patch:** It is normally the patch that matches better with the target patch. The source patch is copied to the target region.

This subsection includes the most important proposals in the area, and as it is done in the previous part, there will be an explanation and reflexion for each one of them.

Criminisi *et al.* [3]

In this article, Criminisi et al. presented a novel inpainting algorithm that was able to propagate textures and structures at the same time. Criminisi states that the filling order has a significant impact in the result and he proposes an algorithm that is capable of determining which patch is going to be filled first. Figure 4.4 is a scheme used by Criminisi et al. to define different regions of the image.

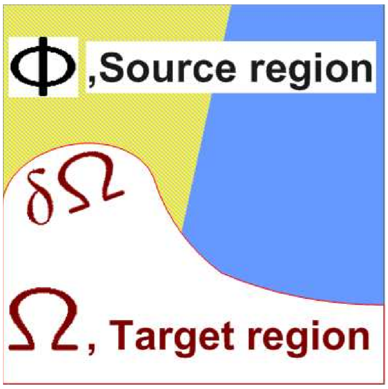


Figure 4.4: Regions defined by Criminisi *et al.*

The algorithm is based in two terms: confidence and data. Confidence is basically a relationship between the amount of pixels of the patch placed in the source region and the total amount of pixels of the patch. On the other hand, data is a function of the strength of isophotes hitting the front $\delta\Omega$ at each iteration. Both of these terms are calculated and multiplied in each iteration in order to define priorities to determine the best patch to fill first. Using this algorithm, normally structural lines are inpainted first, and the texture is propagated later, resulting in a complete technique for complex environments.

So basically, this algorithm has three steps that execute iteratively until the whole mask is covered with patches from the source region:

1. **Compute the priorities for each one of the points of the front:** In each iteration, confidence and data terms are calculated in order to determine the best patch to fill first. This task is quite heavy computationally, but the advantage is that there is no need of user interaction to propagate the structure.

2. **Find a best match for the highest priority patch:** Once the patch to be filled is determined, SSD (Sum of Squared Differences) technique is used to compare and find the best patch. This technique is widely used in image processing, as the results are good for the computational time required. It is a subtraction between the possible source patch and the target patch, and then this difference is squared. The minimum value of all the possible source patches is elected to be pasted in the target patch.

3. **Update pixel values, confidence and data terms:** With the best patch defined, the algorithm copies the values from the source patch to the target patch, but it also updates the confidence and the data terms in order to compute the new front.

As a result, it can be said that this algorithm offers interesting features for video applications. For example, it is robust against any kind of environment, because it is capable of propagating both structures and textures with a minimal user interaction (just selecting the object that needs to be removed). In contrast to the multiview methods presented in the previous section, this algorithm is also able to reconstruct objects that are placed together with the background. The last advantage of Criminisi's method is the ability to use only one frame to reconstruct all the image.

Besides, there are some disadvantages to be considered. The main problem comes when processing time is considered. This algorithm consumes a considerable amount of time computing patch priorities. Comparing to the multiview techniques, accuracy is lower because the whole reconstruction is a deduction of possible patches. Apart from that, patches are searched in the whole image pixel by pixel resulting in a heavy solution.

Wexler *et al.* [7]

An innovative method called bidirectional similarity is presented in their article in order to summarize data. This summarization can be applied in images as well as videos, and it can be used for several applications, such as automatic cropping, photo reshuffling, image collage, object removal and more. As this project's aim is to design and implement DR techniques, this summary is mainly focus on video applications and object removal techniques.

Wexler et al. define the bidirectional similarity in two terms called completeness and coherence. As it is shown in figure xx, completeness means that all the patches contained in the input image should be in some part of the output image. For example, the red and green patches are copied with the same size, but different positions. On the other hand, the blue patches are merged into a unique blue patch in the output image summarizing the information. But at the same, the output image maintains all the information from the input image as both blue patches are considered equal. Additionally, the coherence term means that all the patches contained in the output image should come from the input image. In other words, no new undesired visual artifacts should appear in the output image.

As it is mentioned before, object removal technique is the most interesting part of this article for this project. The technique is actually the same as the bidirectional similarity. In this case, a mask is applied in the object that needs to be removed, and afterwards, the mask is filled with patches from the other part of the image. The notation used by Wexler

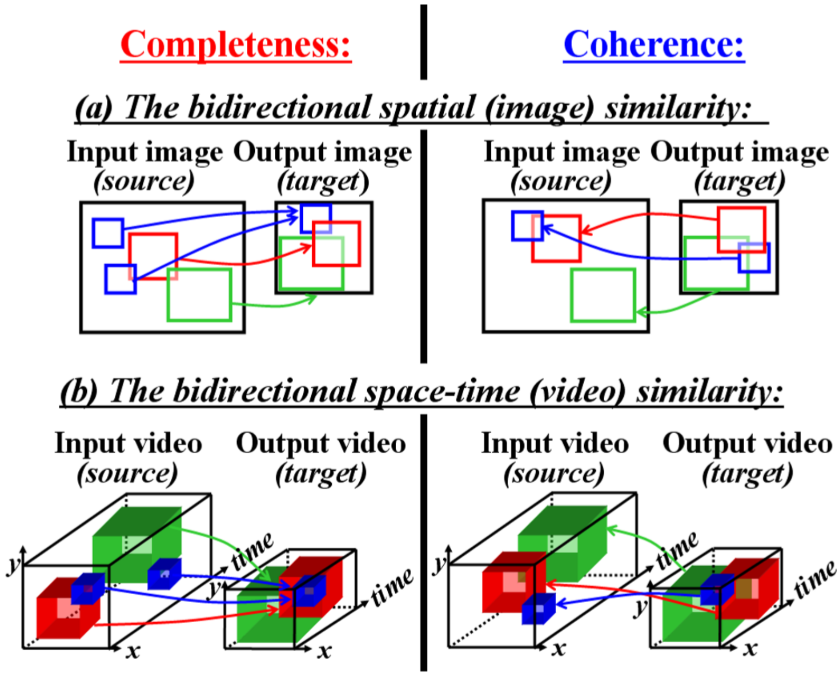


Figure 4.5: Wexler *et al.* bidirectional similarity.

et al. is similar to the one used in the article presented by Criminisi. T is used to define the target region (also called mask), and S is the source region where the information can be taken from. The technique used for comparing the patches is also SSD.

An important feature from this article is the coherence term in video applications. The coherence, assures that the patch in the previous frame and the actual frame will come from the same position. This technique, gives better visual results and computing time is reduced as the source patch needs to be calculated less times.

This article doesn't present any innovative technique for image inpainting. This algorithm includes patch importance weighting functions, but in the case of object removal this weighting is translated to S and T regions. Therefore, patches are searched around the whole image with no priorities. But according to video processing, the coherence term is an interesting feature that needs to be taken into account. This term makes the video look stable during the process reducing the computing time considerably.

Barnes *et al.* [2]

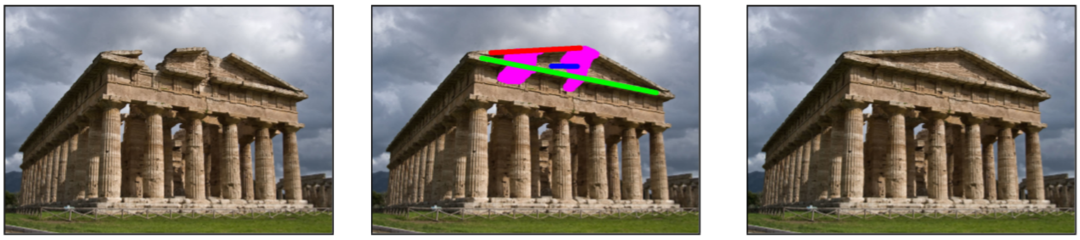
The following article presented by Barnes et al. includes the design of an optimized inpainting technique that is able to provide interactive performance. The main improvement is done in the patch search algorithm, where instead of searching patches in the whole image they search in random positions. As the previous approach presented by Wexler et al., this algorithm can be used for several applications like image retargeting, completion or reshuffling for example. The following summary will focus on image completion or inpainting techniques to find interesting features for this project.

The algorithm is called randomized nearest neighbour algorithm and it is divided in three steps, which are iterated to refine the result:

1. **Initialization:** Each target patch searches some random patches and keeps the position of the best result obtained. The comparison of the patches is also done using SSD technique.
2. **Propagation:** Each target patch compares its values with the neighbour's values to determine which one is better. If the neighbour's patch is better the third step is done with the values of the neighbour's position.
3. **Search:** The algorithm performs a search around the source patch position looking for a better result.

As it has been said before, this is an iterative process and the amount of iterations is undetermined. The result is improved with every iteration, but the computing time is also higher. Therefore, it is interesting to maintain a balance between both of them. Apart from that, it is proved in the article that the algorithm has the ability to propagate structures too. In this case, the user needs to define the areas where the structure is occluded, and the region where the information needs to be taken. Thus, the algorithm has a good performance even in complex scenarios, with the information given by the user. Figure 4.6 is an example of it.

Although this algorithm obtains good results in reconstructing structural image, the need of the user interaction is undesirable and not viable for video applications. Therefore, the only interesting algorithm that is useful for the DR application is the patch search algorithm, which is capable of working in an interactive time. Previous methods like Criminisi *et al.* processed the images in the order of minutes, whereas this algorithm is able to do it in



(a) Original image. (b) User constraints. (c) Result.

Figure 4.6: Barnes *et al.* algorithm with user constraints.

few seconds. It is also true that Criminisi's searching algorithm can be optimized to fasten the process, as searching in the whole image pixel by pixel is not an efficient method in terms of time.

Herling *et al.* [8] [14]

The first self-contained real-time capable DR in video application was presented by Herling et al. in [8]. Their approach is a complete system that includes an object detection module, tracking and inpainting methods. Apart from integrating existing techniques into a single system, they also propose optimizations that can be interesting for this project.

The reconstruction of the image is done from the information obtained in each new frame and the previous ones. They divide the process in three steps:

1. **Object detection:** The user needs to select the object that needs to be removed from the video. As active snake algorithm [17] is implemented, this phase detects the contours inside the selected area.
2. **Object tracking:** The system tracks the object selected in order to locate the area that needs to be inpainted. In this case, the contours detected in the first step are tracked using active snake algorithm.
3. **Inpainting:** The last step consists in reconstructing the area based on the information from actual and previous frames. Herling *et al.* made some changes to Barnes *et al.* inpainting approach in order to achieve the real-time capable objective.

As the main challenge for this technique was making the algorithm real-time capable but without loosing much quality in the image. Hence, they applied three optimizations to achieve that goal:

1. **Data reduction:** Consists of reducing the information to be processed in order to get better timing constraints. Herling et al. made use of image pyramids together with grayscale images to speedup the patch search process.
2. **Frame to frame propagation:** Patch correspondences calculated for the actual frame are used as initialization for the the next frame. Thus, instead of applying a random search in each iteration, the patches are searched around a specific point.
3. **Multicore support:** The last optimization included in their work is the optional parallelization of almost all tasks in the image completion pipeline. This means that the processing time decreases linearly with the number of CPU cores.

The solution presented in [8] was improved and published in [14]. They added a fingerprint selection feature to select the area and they used segmentation technique to select the object inside the area that needs to be inpainted. They also changed the object tracking mechanism from an active snake approach to a two phase contour tracking approach. They used a homography based contour tracking in the first phase, while in the second phase, the new contour is refined and adjusted according to the undesired object area. This improvement leads to better contour points correspondences between successive video frames.

Results of this algorithm can be seen in figure 4.7. In a) the glasses are the object to be occluded, hence, the user would make a selection around the glasses. Using contour detection, the silhouette of the object is determined and the mask is created as it can be seen in b). Finally, c) shows the result of applying the modified version of the inpainting technique proposed by Barnes et al.

(a) Original image. (b) Mask. (c) Result.

Figure 4.7: Example of Herling *et al.* process.

As it is mentioned before, this system can be an interesting reference for the development of this project, as there are optimizations that can be applicable in any kind of inpainting processes. The inpainting technique that is used in this project is the one presented by Barnes *et al.*, which means that is not able to propagate structures without user interaction. As this system asks the user just to select the object, it can be said that it is not able to work with structural backgrounds. This is an important weakness of this algorithm, as it is not robust for any kind of environment.

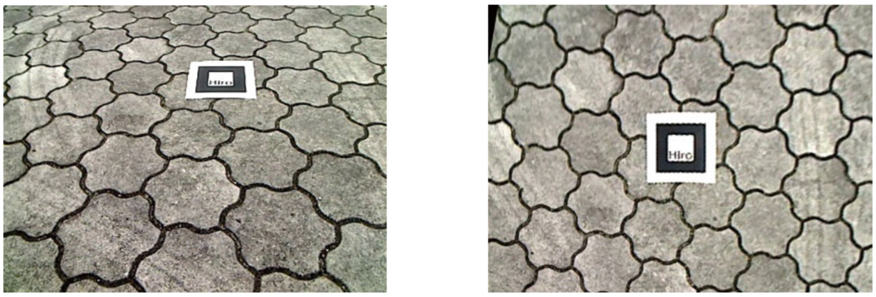
Kawai *et al.* [1] [15]

Kawai *et al.* made a great contribution to DR by presenting the articles [15] and [1]. Even though both articles can be combined in the same system, they are independent solutions.The first one, is a solution for recovering background structures. Whereas the second one, is a solution for inpainting with reflection of luminance changes.

For some applications, it is interesting for the AR to work together with DR, as explained in [15]. Usually, AR is implemented using markers because this method assures robustness and easiness. The main problem is that the virtual object is superposed in the image over the marker, but sometimes doesn't cover all the marker and it produces an undesirable effect. In order to solve the problem, they propose a process combining correction of perspective distortion and exemplar-based inpainting based on Wexler et. al [7] as well as detection of luminance changes. Their main contributions are the process of rectifying the image and detecting reflection of luminance changes in order to improve the inpainting process.

Exemplar-based image inpainting techniques use patches to copy information from one area to another. If the background has a perspective distortion, exemplar patches will not correspond exactly to the target patch. The correction of perspective distortion proposed in this article unifies the size and shape of texture patterns used as exemplar. Figure 4.8 is an example presented in their article where the correction of perspective distortion is performed. It is important to note that this technique needs a marker in order to determine the transformation matrix to obtain the rectified image.

In [1] Norihiko Kawai et al. propose a design for DR considering background structures. They focus mainly in the inpainting more than the detecting or tracking process. The aim of this work is to overcome the problem of perspective distortion of regular patterns that appears in exemplar-based inpainting. This is done by rectifying the input image and applying changes based on similar patterns from the image. Their previous scheme [15]



(a) Input image. (b) Rectified image.

Figure 4.8: Correction of perspective distortion (Kawai *et al.*).

as well as Herling *et al.* [8], used homography to ensure temporal consistency determining searching areas in the next frame. The problem is that homography works well when the background is almost planar, but the results in non-planar backgrounds are not accurate. In their method, the scene around the target object is divided into multiple planes whose number is automatically determined, and inpainted textures are successfully overlaid on the target object under comparatively unrestricted camera motion using the estimated multiple planes and camera pose by Visual-SLAM.

(a) Input image. (b) Mask. (c) Segmented planes.

Figure 4.9: Visual SLAM implemented in [1] (Kawai *et al.*).

Figure 4.9 represents the Visual SLAM technique that is used by Kawai *et al.* In step b), the mask is selected and different planes are detected by the Visual SLAM. But before, all the planes need to be separated and numbered to identify the regions as it can be seen in c). In this case, the segmentation is pretty simple, as the walls have several textures, different from one to the other.

It is obvious that these articles contribute positively to the area of DR. First of all, the rectification process is an interesting feature to be taken into account, as exemplar-based inpainting systems like Barnes' or Criminisi's algorithm would improve their results with this optimization. But this technique introduces extra processing time to the inpainting process and it is necessary to use markers with it. Then, several markerless applications would be discarded directly. On the other hand, the second article introduces Visual SLAM tracking system which demonstrates good results in textured backgrounds, but it wouldn't accomplish our objective of having a robust DR system for any type of environment.

4.3 Conclusion

To sum up, although all the presented works have features that can be applied in this project, it is impossible to combine all of them to form a system. Hence, some of the most interesting features have been considered for the design of the prototype that is going to be developed in this project.

Criminisi's algorithm is the most robust technique that is able to propagate structures as well as textures with a minimal user interaction. As it has been mentioned before, this algorithm has a high computational cost because of its patch searching algorithm. The concepts of patch search optimization explained in Barnes et al. article, can be taken as reference to improve Criminisi's algorithm patch search technique, reducing considerably the processing time. Apart from that, the coherence term explained by Wexler et al. can be applied in the video application to obtain a better visual effect reducing even more computing time (like Herling et al. applied in their prototype). Finally, Kawai et al. contributions can be interesting for future works. As inpainting algorithms consume most of the time, it is hard to get a real-time or even close to real-time application. If these techniques proposed by Kawai et al. are implemented, it is even harder to achieve that objective. Nevertheless, the improvement introduced by these techniques is an interesting feature that more powerful computers could handle in the future.

5 | Analysis of Alternatives

In this section, the alternatives considered during the progress of the project are described. Some indicators are fixed for each one of the alternatives, and the last decision is taken based on these values.

5.1 Election of inpainting method

As mentioned in the introduction, the biggest part of this project consists of determining a good inpainting algorithm that is fast and robust for different scenarios. Knowing this, the hardest part of this project was to determine which algorithm fits better to the project requirements. After reviewing the literature, it can be said that Barnes *et al.* [2] and Criminisi *et al.*[3] are the reference works in the area of inpainting techniques. Additionally, there is also an interesting function called "inpaint" based on Telea *et al.* [4] that is implemented in OpenCV [9].

5.1.1 A model: A randomized correspondence algorithm

In his article [2], Barnes describes an innovative technique for inpainting able to get good results in a fast way. It is an iterative algorithm that searches random patches around the image, and then, refines the result by searching around the best patch along the random patches. Thus, instead of searching the patches through all the image, it searches just in some random positions, reducing the computational cost and getting acceptable results. The following patch, that is next to the one that has already been defined, is searched around the previous one, but also in a random fashion throughout the image, to check if there is a better result.

Figure 5.1 represents the scheme used by Barnes *et al.* to explain the functionality of their approach. In step (a), random patches are searched for each target patch. Afterwards, in step (b), each patch is compared with the neighbour patches to check if the neighbours have a better result. If so, the algorithm searches the new patch around the neighbours patch to refine the solution like it is done in step (c). For example, the blue patch in figure 5.1 realises that the red patch obtained a better matching. So in the next iteration, the blue patch will search around the red patch to improve the resulting image.

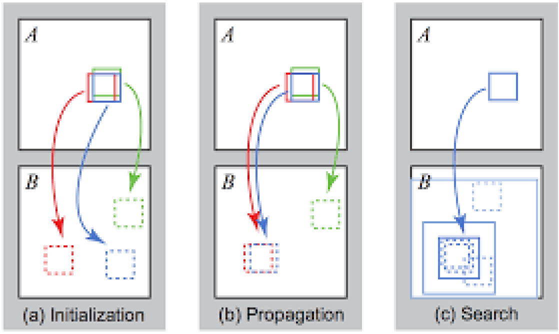
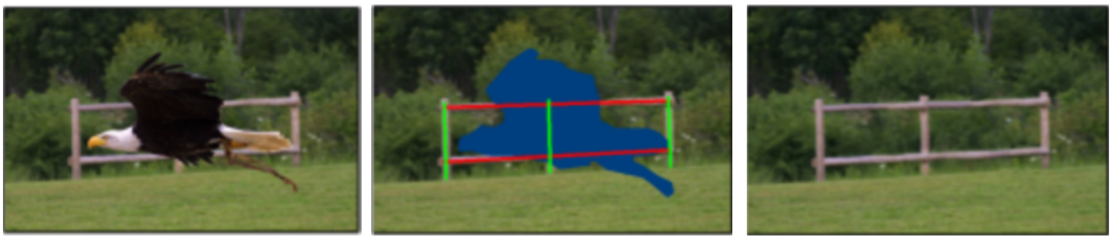


Figure 5.1: Scheme used by Barnes [2] to explain his random search algorithm.

Barnes also describes a technique that make it possible to maintain structural objects in the result. In this case, the user needs to define some constraints manually in order to fill that information with a different texture. Figure 5.2 (b) shows how user constraints are defined in the image, in order to maintain structural information. Finally, as it can be seen in (c), the user defined constraints help in the reconstruction process obtaining a good visual result.



(a) Input image. (b) Hole and guides. (c) Completion result.

Figure 5.2: Example included in [2] to show how the algorithm can propagate the structures with user interaction.

The advantages and disadvantages of the algorithm proposed by Barnes *et al.* with respect to Criminisi *et al.* and Telea *et al.* are exposed in the following classification:

Advantages:

- Speedup comparing to the previous techniques.
- There is already an implementation by Herling *et al.*[8] that shows that this algorithm is real-time capable with some optimizations.

Disadvantages:

- It is not robust for all scenarios.
- Even though Barnes implements a method to preserve structural information, this needs to be done with the help of the user. Therefore, it is not viable to apply this user interaction frame by frame if this feature is intended to be applied in video applications.

5.1.2 B model: Structure and texture propagation

The main improvement of Criminisi *et al.* [3] comparing to previous techniques is that it defines the order in which the mask is filled. This algorithm is based in two indicators: the first one is called confidence, which is the percentage of the pixels inside the source region, and the second one is called data, which determines the amount of structural information in that patch.

According to the method used for comparing the patches, it is similar to the one used by Barnes *et al.*. SSD (Sum of Squared Differences) is a common technique that is used for correlation based similarity measures. Criminisi as well as Barnes use SSD, but the difference is that Criminisi apply SSD in the whole image pixel-by-pixel, whereas Barnes optimizes the algorithm by a random search.

Figure 5.3 shows how the filling order has importance even in a simple scenario. In this image, the onion peel filling order is compared with the algorithm designed by Criminisi. The onion peel filling order is simply filling the patches clockwise or counter-clockwise from the outside area to the inside. According to Criminisi's algorithm, it follows a criteria based on the two terms mentioned in the previous paragraph. The (a) image shows the original image with a masked region (the white region that is similar to a half-moon). Images (b,c,d) are the progress of the result applying the onion peel filling order, whereas (b',c',d') represent the progress using the algorithm proposed by Criminisi. As it can be seen in (d'), the image is recovered perfectly, because the structural component (the line dividing both colors) is recovered first. On the other hand, (d) has a significant error because the green patches are propagated before the blue ones.

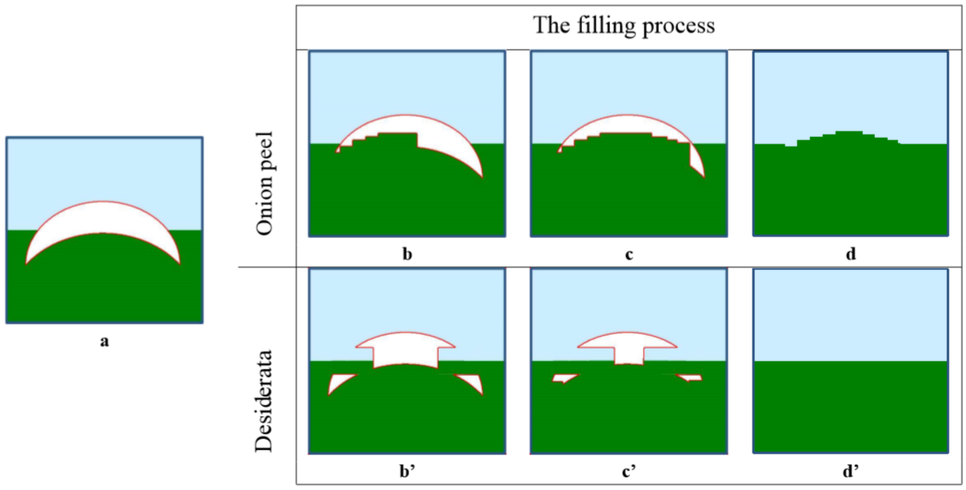


Figure 5.3: Filling order inportance in the resulting image (Criminisi *et al.* [3])

Advantages:

- The ability to preserve the structural information together with the texture pattern. This makes it robust for most of the scenarios, because it doesn't matter if the object is occluding an structure or a plain background.
- The searching in the whole image is not efficient, so it is possible to implement some optimizations to speed-up the process.

Disadvantages:

- The processing time. This algorithm calculates which patch is filled first at every iteration, and apart from that, it compares the possible patch with the whole image. This leads to a high processing time, and therefore, it needs some optimizations to be viable in video editing.

5.1.3 C model: Pixel weighting function

This method is based on Telea's article [4], where it applies a weighted average of the surrounding pixels to each pixel in the mask. This weighted value is based on the pixel's neighbourhood, so the final result is like a pixel interpolation around the mask. Computational effort is really low, calculating every pixel from the outside area to the center of the mask individually without prior information or any information about other regions of the image.

Figure 5.4 shows the scheme designed by Telea in his article [4].In this image, *p* is the notation for the pixel that is going to be reconstructed. B(ε) is the known neighborhood of the point *p*, which is the information that is going to process to get the value of *p*. The point p is defined as a function of all points *q* in B(ε) by summing the estimates of all points *q*, weighted by a normalized weighting function. For further details about the weighting function, please refer to the section 2.3 in [4].

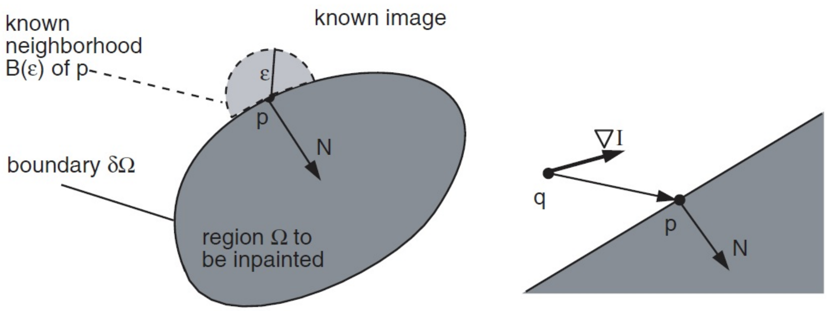


Figure 5.4: Telea *et al.* [4] inpainting scheme.

Although this algorithm looks pretty simple comparing to other solutions, the results obtained in small inpainting masks are acceptable, and considering its little processing time, it becomes interesting for some applications. It is normally used for old image reconstruction with small cuts or malformations like it is shown in figure 5.5.

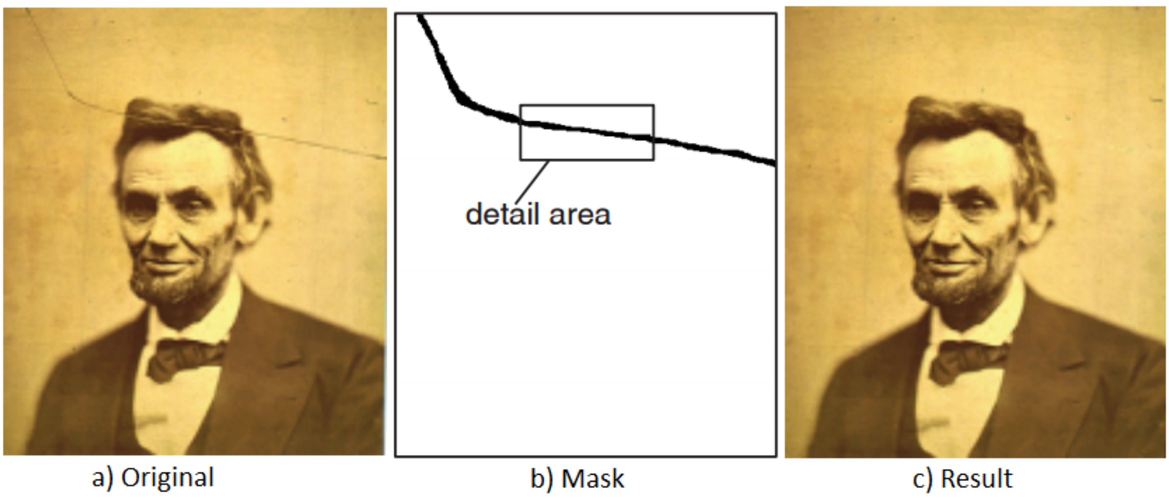


Figure 5.5: Example of an image reconstruction taken from [4]

Advantages:

- It is very fast.
- It is a good option for inpainting objects that occupy a small area in the image.

Disadvantages:

- If this method is applied in large objects the result becomes visually non-acceptable.
- The object is filled with a blurred colour based on the surroundings, looking artificial and untextured. This means that this option doesn't work in all scenarios, and then, it is not robust.

5.1.4 Criteria of choice

The following indicators are defined for the election of the inpainting method that fits best the project requirements:

	WEIGHT	EVALUATION		
		A model	B model	C model
Ability to propagate structures	25%	5	10	0
Ability to propagate textures	25%	10	10	5
Computational effort	25%	7	3	10
Scalability	25%	5	8	2
TOTAL		**6,75**	**7,75**	**4,25**

Table 5.1: Criteria of choice: Election of inpainting method

Taking into account all these parameters, Criminisi's method is chosen by a significant difference with Telea and closely followed by Barnes. The main difference is that Criminisi's algorithm is the only method that is not tested in video applications. Even though Criminisi's algorithm is heavy computationally, it is scalable by applying some optimizations, as the original algorithm searches along the whole image.

5.2 Election of object detection and tracking system

Another important part of this project is the tracking system, which is directly related with the object detection. It is necessary to search some interesting visual cues that will help us in the process of determining the new position of the object. In this case, it is also important to have a robust tracking system that consumes the minimum possible time in order to have more margin in the inpainting process.

During this project, two tracking systems that are available in OpenCV have been implemented and tested: "Camshift" [18] and Optical Flow based in Lukas-Kanade [19].

5.2.1 Camshift

OpenCV is a huge library for Computer Vision that implements a lot of functions for different purposes. One of them is this one called *Camshift* that identifies and tracks an object based on the colour information. As its name suggests, Camshift (Continuously Adaptive Meanshift) is actually an adapted version of *Meanshift*.

A probability distribution function (PDF) is determined to associate a pixel value with a probability that the given pixel belongs to the target. Histogram back-projection is the technique used by meanshift and camshift to implement a PDF. With the help of it, Meanshift moves the selected area to the area of maximum pixel density. Thus, it detects the location where it has more colour information (normally the center of the object). This is an advantage and disadvantage at the same time, as the algorithm can rectify or add an error to the user selection. As this algorithm tracks only based on the colour information, there is a need of pre-configuration to determine which colour needs to be tracked. And apart from that, it is also important to know that only one colour can be tracked at the same time. For example, if there is an object divided in two colours and this colours are different from each other, this algorithm would be capable of tracking just one of them.

In figure 5.6 the mass centre is re-targeted using the information provided by the histogram back-projection, where *C1* is the initial selection of the user and *C2* is the algorithms approximation of the mass centre. This is done because this tracking system is designed to track a coloured object based on the configuration fixed by the user. Then, the optimal position is the area where this colour distribution is higher.

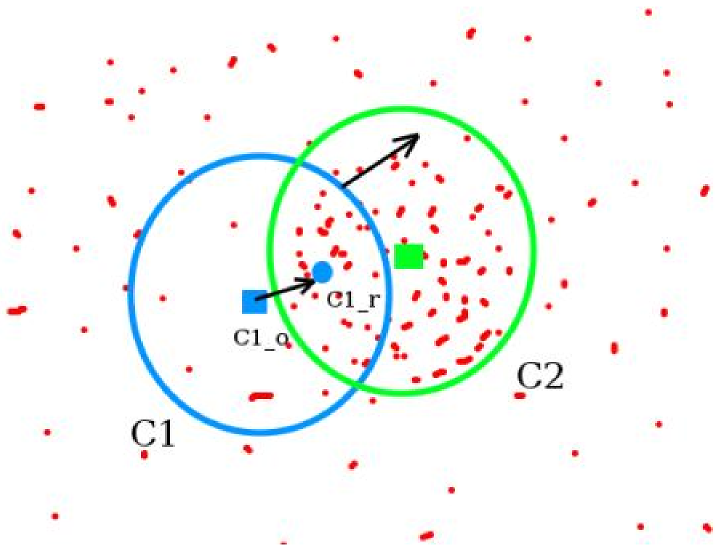


Figure 5.6: Scheme of meanshift functionality.

By default, Meanshift just calculates and detects the object at once and tracks till the end, so the problem is that if the object suffers transformations like scaling, the region maintains the same size. That happens because Meanshift is based on static distributions, which are not updated unless the target experiences significant changes in shape. Camshift is an evolution that has been designed to overcome this problem by continuously adapting the probability distributions of the the object (distributions are computed in each frame).

Advantages:

- It is simple and easy to implement in OpenCV.
- It works well when the object has a different color comparing with the background.
- Computational effort is quite low being possible to apply it in real-time applications.
- It works well when the object is homogeneous, without texture or characteristic points.

Disadvantages:

- The object is detected properly, but the corner points that define the OBB (Oriented Bounding Box) of the object don't rotate in the same way that the image does. The problem comes when the homography matrix is calculated, because the pixel correspondences have a significant error in the result.
- It is necessary to pre-configure some parameters to track a specific colour.

5.2.2 Lucas-Kanade optical flow

Another option available in OpenCV is the optical flow tracking based on Lucas-Kanade algorithm [19].

A feature detection algorithm is a technique to detect characteristic points or keypoints in a given image. For example, some feature detectors are designed to detect the edges or shapes of the objects. Others just look for high textured points that might be unique in the subsequent frames. Then, this points are normally used to determine the new position of the object in the next frame.

Lucas-Kanade algorithm uses keypoints detected by the feature detection algorithm in order to get the new location of the input points. With this new points and the previous ones it is possible to calculate a matrix to determine the translation, rotation, scaling and other transformations. Depending on the transformation that is used, it will have more or less degrees of freedom (DOF). For example, similarity transformation has less DOF than the perspective transformation (represented by a homography), so it is less accurate. To achieve the objective of being close to real-time, this method downsamples the image and then propagates the changes to the finest pyramid layer. As it can be seen in the Figure 5.7, computation is done in each pyramid layer, but the computation in the original image is based on the information from the previous image scale and so on. Thus, computational time is reduced obtaining good results.

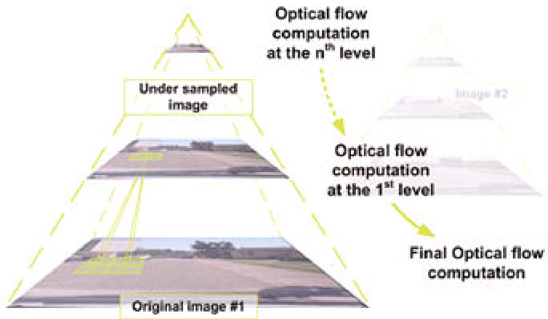


Figure 5.7: Scheme of the pyramidal design in Lucas-Kanade optical flow.

Advantages:

- The resulting perspective transformation (homography) is fairly precise.
- It is fast enough to be implemented in real-time applications.
- There is no need of any pre-configuration.

Disadvantages:

- This tracking system is not applicable when the object that needs to be tracked is homogeneous, with no texture (in this case no keypoints would be detected).
- If a keypoint is lost during the process of tracking it is lost forever, so there is a need of recalculating the points after a while.

5.2.3 Criteria of choice

There are lots of features that differ from one tracking system to another, since one is based on color distribution and the other one focuses on keypoints. In table 5.2 the features that have greater impact in the project are highlighted:

	WEIGHT	EVALUATION	
		Camshift	**Lucas-Kanade optical flow**
DOF	70%	6	9
Dependence of pre-configuration	10%	3	10
Precission	10%	7	9
Easy to implement	10%	8	6
TOTAL		**6**	**8,8**

Table 5.2: Criteria of choice: Election of object detection and tracking system

Altough from the results of the evaluation performed in table 5.2 Lucas-Kanade optical flow mechanisms seems more appropiate for this project, in fact, there are some cases where Camshift performs better (textureless objects). For this reason, in order to achieve the best possible results in all the cases, both alternatives have been implemented. With this, versatility and robustness is assured in the prototype.

5.3 Election of the patch searching area

Image inpainting algorithms are mainly designed for getting the best possible quality in the result. It doesn't matter the resources that are used for it or the amount of time used. Because of this reason, some inpainting algorithms can spend several minutes to reconstruct a single image.

As this project's objective is to implement this techniques for video applications, there is a need of optimizing them losing the minimum quality possible. For example, [2] introduced a significant optimization even for image editing using random searches for patches instead of a search in the whole image. During this project three different searching areas are evaluated comparing the processing time and visual results.

5.3.1 A model: Search in the whole image

It is the way that the Criminisi's algorithm is implemented by default. The algorithm searches for source patches along the whole image pixel by pixel resulting a heavy task. Considering the high computational effort to calculate the priority of the patches together with this patch searching technique, it leads to an even higher processing time. In the following image shows the scheme of the implementation:

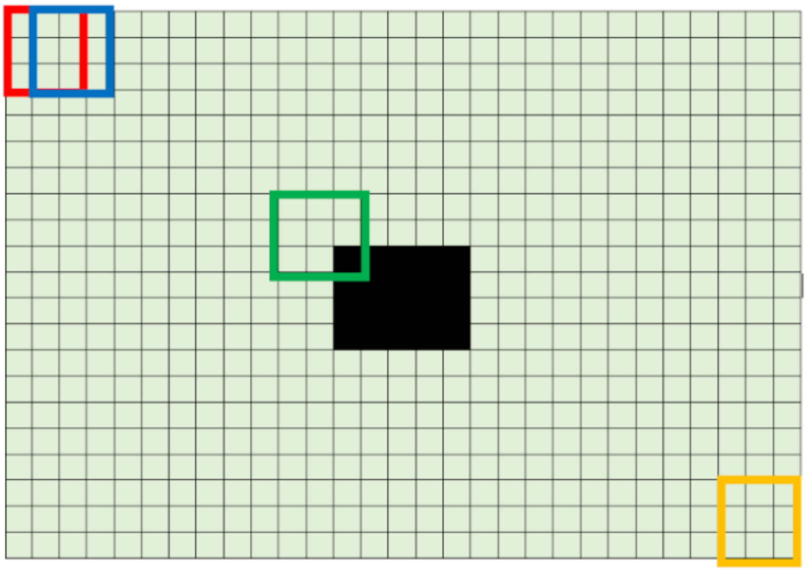


Figure 5.8: Criminisi's patch searching scheme.

The black rectangle is the mask or target region, the area where the object that needs to be occluded is situated. The green rectangle is the patch that needs to be inpainted. The red one is the first patch that is compared with the green one, the blue one is the next one and this sequence goes until the yellow patch is reached. This comparison is done for each patch in the target region, becoming a heavy task in the inpainting process.

Even if this algorithm is not viable for video editing, it has to be considered as the most complete option, because it takes all the information from the image, no matter the cost or distance from the target patch.

Advantages:

- This option takes into account all the information contained in the image.

Disadvantages:

- Long processing time, because the algorithm goes pixel-pixel in the whole image.
- Not feasible for real-time applications.

5.3.2 B model: Crop a part of the image

One of the possibilities to reduce computing time is to reduce the searching area. In this case, instead of searching around the whole image, the target patch is just searched in an area proportional to the mask size. In order to fulfil this objective that area is cropped from the original image and processed by the original inpainting algorithm reducing the amount of patches to be compared. This new size must be proportional to the mask size to assure enough information to reconstruct the area.

The difference with the previous approach can be seen in figure 5.9. Comparing to the previous image, it is easy to identify that the search area is reduced considerably, but it is also obvious that there is a loss of information that can affect to the result. Normally, information for the inpainting process is taken from the surroundings of the mask, because the result is a propagation of texture or structure. Anyway, there might be some cases where this reduction will present some visual errors.

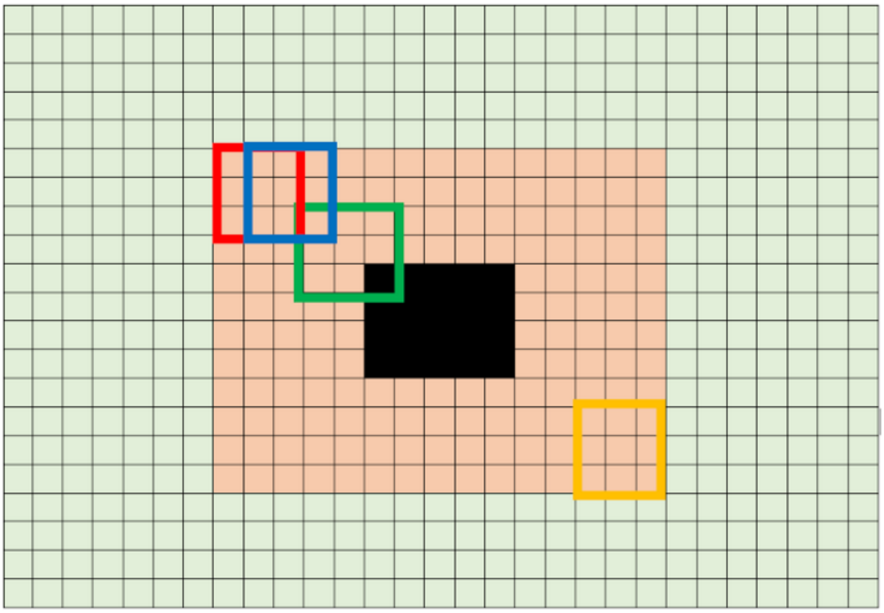


Figure 5.9: Cropped image patch searching scheme.

Advantages:

- Considerable reduction of processing area.
- Processing time is reduced.

Disadvantages:

- Some possible patches are discarded directly.
- Final result can suffer degradation comparing to the original method.

5.3.3 C model: Search around the target patch

The last option that is proposed in this alternative analysis is similar to [4]. In [4] each pixel is calculated based on its neighbour pixels. But instead of calculating pixels this algorithm will search patches around the target patch. This way the algorithm is able to propagate textures that come from the surroundings, reducing even more processing time than the previous approach and getting better results.

Figure 5.10 represents the proposed patch searching scheme. In comparison with the previous approach, this algorithm searches patches in an even more localized area. Instead of searching patches along the croped image, this algorithm focuses just in the neighbour-

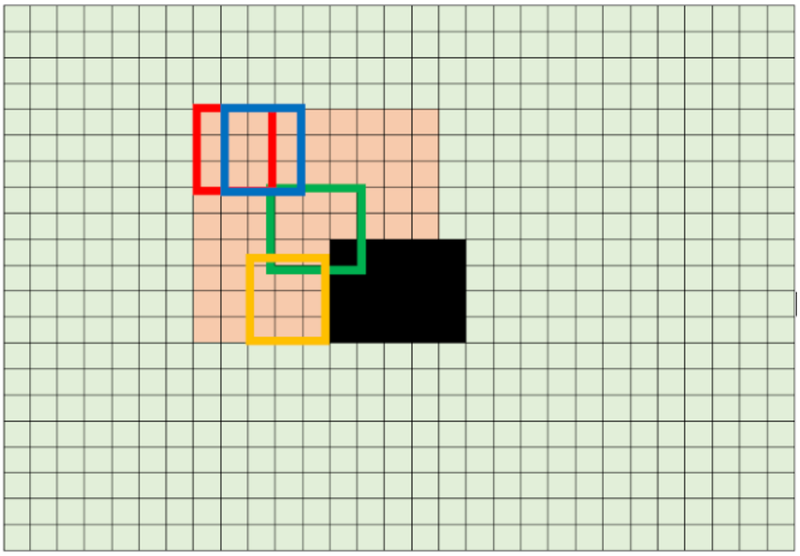


Figure 5.10: Scheme showing the search around the target patch.

hood of the target patch. Like in the previous design, there will be a lose of information, but considering that most of the information is taken from the surroundings and the low computational time makes this algorithm interesting.

Advantages:

- Processing area is even smaller than the previous case.
- Processing time is reduced with respect to the other approaches.

Disadvantages:

- Reduction of the search area implies a loss of information.
- The final result might degrade in favour of the computational time.

5.3.4 Criteria of choice

The patch election mechanism has a direct impact in the resulting image being a crucial part of the algorithm. In order to fix an objective scenario to measure the speed and quality, a 720p input video with a 32x32 patch size was processed equally by all the options presented in this section. However, the output video's quality is a subjective term that needs to be measured. Hence, forms were created for random users to qualify the approaches. As the objective is to implement this algorithm for video applications, the main objective is to obtain a balance between speed and quality as it can be seen in the following table:

	WEIGHT	EVALUATION		
		A model	B model	C model
Speed	50%	1	7	8
Quality	50%	9	7	8
TOTAL		**4,5**	**7**	**8**

Table 5.3: Criteria of choice: Election of the patch searching area

Even though it is not possible to measure this table being fully objective, most of the users determined that A model had the best quality of reconstruction, followed closely by B and C.

Apart from that, speed measure is based in patch size and and mask size so it can be relative from one scenario to another. But for common applications mask size is bigger than the patch size, and consequently model C is faster than B. Determining that C model was 17,6 times faster than model A and 1,46 times faster than B. It is not correct to say that the speed of an algorithm is 0 and that is the reason why model A has 1, but it is also not fair to put a 10 to C because there might be better solutions for it.

Considering both parameters, C model was determined as the best option for the patch search mechanism.

6 | Risk Analysis

In this part of the project risks need to be identified, defined and a plan needs to be done in order to have a solution whenever one of this risk comes true.

6.1 Risk identification

Risks are divided in two groups depending on the source of each risk. If the risk comes from the project itself it will be classified as internal, otherwise it is considered as an external factor.

Internal risks

1. The project needs more time.
2. Projects costs increased.
3. Inpainting algorithm is not efficient enough for video applications.
4. Tracking system is not precise enough.

External risks

5. An innovative inpainting process is invented.
6. The project is damaged or deleted.
7. Competence has designed a better solution.

6.2 Risk probability and impact

Each one of the risks that are identified are defined in this part, deducing its probability and impact in the project.

1. **The project needs more time**

   There arise some issues during the project execution that might make it last longer that initially expected. Probability for this risk to come true is quite **high** because this algorithm is designed for images, and its performance in video applications is still not tested.

The impact that this risk has in the project is **high** because the probability for external risk like 5 and 7 will increase considerably.

2. **Project's costs increased**

   It is possible that some factors can change the duration of the project and consequently changing the costs of it. However, it is rare that the costs of this project will change during the process, because they are calculated in detail in the project budget. Therefore, it is defined as a **low** probability risk. Because of the same reason, the impact in the final result is also **low**.

3. **Inpainting algorithm is not efficient for video applications**

   One of the main part of this project is the inpainting process. This inpainting process is modified and adapted for video editing applications. The whole project is centered in this inpainting technique as it is the algorithm that creates the virtual layer superposing the object. It is **highly** probable that the algorithm doesn't perform as fast as it is supposed to.

   That hypothetical situation would have a **big** impact in the project, as this is detected normally in the last part of the project while the testing is done.

4. **Tracking system is not precise**

   Another significant part of the project is the tracking system. There are a lot of tracking systems available in the network that work better or worse depending on the scenario. Even though this project implements two tracking systems able to cover most of the scenarios, there might be some special cases where it doesn't track properly. For example, there are techniques that employ proximity sensors to help in the detection process. Then, the probability of not getting the desired tracking is **medium** as there might be some scenarios where the tracking fails.

   But if taking look at the impact that this tracking system has in the project it is quite **low**, because it can be replaced easily in the future to a more complex and complete tracking system.

5. **An innovative inpainting process is invented**

   Even though the research in inpainting techniques is quite low, it is possible that some researchers come up with a revolutionary algorithm. The probability of this case is **medium** because there is space for optimizations in existing algorithms and the impact in the project is also **medium** because that new algorithm is designed for image and the effects for video can be different.

6. **The project is damaged or deleted**

   It sometimes happen that the hard drive where the project is stored suffers some problems and all the information inside is damaged. Or it is possible that the computer gets some virus that deletes everything that you have. Even if its probability is **low** the impact of this event is extremely **high**.

7. **Competence has designed a better solution**

   Finally, it is also imaginable that other developers can come up with a better solution for diminished reality. As it is mentioned before, there is not much research in the area making the probability **low** but the impact in the project would be **high** because it makes it useless and nobody would pay for it.

6.3 Probability-Impact matrix

		Impact		
		Low	Medium	High
Probability	Low	2		6
	Medium	4	5	7
	High			1, 3

Table 6.1: Probability-Impact matrix

6.4 Contingency plan

A contingency plan is done to avoid the risks that are identified. The solutions proposed in this section will lead to an extra work in order to assure the correct development of the project. As some of the detected risk probabilities or impacts are low, the contingency plan becomes undesirable. Therefore, just the risks 1,3,5,6 and 7 will be considered for the contingency plan.

1. **The project needs more time**

   A change on the duration of the project means a change in the costs. If the project ends faster than it was supposed to there will be more benefits, but if the project lasts longer than it was supposed there will be a loss of money. In order to avoid this problem, a margin of 10% is fixed in the duration of the critical tasks.

3. **Inpainting algorithm is not efficient for video applications**

   The election of a proper inpainting algorithm will lead this project to a successful end or to a dead end. Instead of choosing the algorithm randomly or without any previous study, some time will be fixed to make an extensive state of the art to select the algorithm that fits best for this. Optimizations will be implemented for the algorithm in order to make it feasible for video applications.

5. **An innovative inpainting process is invented**

   It is impossible to guess if there will be a new algorithm in the next days, months or years. But it is important to be aware of the projects that different companies or research centers are working in. During the process of the state of the art study some of these companies will be identified and might be possible to learn about their objectives and lines of work. Depending on the progress of the project it is possible to adapt this inpainting technique to our project becoming beneficial to our project.

6. **The project is damaged or deleted**

   Although it is improbable to have a problem of this type, its impact makes it important for the process of the project. Nowadays there are several solutions to backup projects in the cloud, maintaining different versions like Git. Apart from that it is also interesting to save the project in another cloud platform that assures projects availability at any time.

7. **Competence has designed a better solution**

   As it is said in the third point it is almost impossible to know if another solution will be released soon. Regardless, a good state of the art will help to know about the situation of different DR techniques as well as companies or research centres that are working in the area of DR. With this, the risk is not completely avoided, but the probability for this risk to come true will become lower.

Part II

METODOLOGY

7 | Task description

In this section all the tasks that were completed during the project are described. Some of these tasks are subdivided to improve the organization of the methodology.

WP1:Project definition

The first work package (WP) consists in defining the project's objectives or requirements. Then, an exhaustive study of the available alternatives is done in order to achieve the previously defined objectives in the best possible way. This WP was divided in two tasks called "Project specification definition" and "Study of the available alternatives".

T1.1 Project specification definition

The first step of the project consists of defining the specifications of it. It is basically defining the scope and requirements for it.

T1.2 Study of available alternatives

As it is mentioned in the report, the election of an appropriate inpainting technique is the most important part of the project. If a selection of an algorithm is done randomly, without previous research, it is possible to have problems with the optimization step. There are some algorithms that are already optimized but are not fast enough to apply them in video processing applications. If one of these algorithms is chosen, the project would have an unsuccessful ending. On the other hand, if a suitable algorithm is chosen, the next steps will be easier and success will be more probable.

WP2: Software design and implementation

This WP is the main part of the project where the prototypes are designed and implemented. Due to its complexity and in order to maintain an organization, this WP is divided in several tasks and subtasks. Every task in this WP is sequential, as it can be seen in the Gantt diagram.

T2.1 Design and planning of the prototypes

As it is mentioned in the objectives (chapter 2), two prototypes have been developed during this project. In this task, a design and development of a detailed version of the planning was done for both prototypes. According to the design, code needed to be modular in order to accept different inpainting methods, object detection algorithms or tracking systems.

T2.2 Realization of the initial prototype

During the development of this task an initial prototype is created. This initial prototype must be a simple implementation of a DR system that will serve as an structure for the final prototype. It is basically combining different OpenCV functions to track and inpaint the object in each frame.

T2.3 Realization of the final prototype

This final version of the prototype must implement a high-quality image inpainting technique that is at least able to propagate textures. Because of its complexity, this task is subdivided at the same time in other tasks: the first objective was to speed up the inpainting process, and afterwards, implement a more precise tracking system.

T2.3.1 Optimizations in the inpainting algorithm

During the realization of the project some optimizations were implemented to accelerate the execution. This task is also subdivided to structure better all the planning process.

T2.3.1.1 Patch search area

The first optimization that made a big difference was the change on the patch searching area. The original algorithm searches patches around the whole image and this process is computationally heavy. There are different options to delimit the search area with its advantages and disadvantages. The election of the solution is discussed in the analysis of alternatives (chapter 5).

T2.3.1.2 Pyramidal design

A common technique to speed up the processing in an image is to use a pyramidal design. In this case, the input image is downsampled "x" times and it is processed in a lower resolution. Then, changes from the coarse level of the pyramid are propagated to the finest layer. These tasks consisted of implementing this design to the prototype that is presented in this project.

T2.3.1.3 Temporal coherence

Previous optimizations were oriented to optimize the reconstruction process of a frame. However, as the objective is to implement DR in video applications, it is essential to apply optimizations that help in the process of recalculating the patches from frame to frame. The idea was to reduce computational time by applying temporal coherence to the prototype.

T2.3.2 Optimization of the tracking system

There are several tracking systems that could be applicable to this project. Even though an improval of the tracking system was not the main objective, the tracking system of the initial prototype was not robust in some scenarios. In order to obtain more robustness in the DR solution, this task consists of implementing a new tracking system complementary to the one implemented in the initial prototype.

T2.4 Performance and quality tests

The objective of this task was to generate measurable results that helped in the quality and performance analysis. The original algorithm was compared with the final result, but also with each one of the optimizations mentioned in the previous task. In order to have a consistent measurement, different scenarios with different objects of interest were chosen.

WP3: Project management

The third and last WP is focused on helping future researches to improve the existing prototype in order to get a even more robust and efficient solution. This WP is divided in two tasks, that are: "Project documentation" and "Project follow-up".

T3.1 Project follow-up

The last task is the follow-up of the project, in order to guarantee that the project objectives are achieved. It is basically revising that each one of the objectives mentioned are fulfilled successfully within the envisioned time frame.

T3.2 Project documentation

It consists in generating a documentation that will help future studies in the area. It is also useful to understand several concepts and ideas about inpainting algorithms, tracking or object detection systems as well as the optimizations that are included in the project. Last but not least, this document will also include the conclusions of this project, and a comparison between different optimizations arguing the advantages and disadvantages of each one of them. Apart from that, a presentation is developed summarizing all the information included in the documentation and highlighting the most interesting features of it. This presentation also includes weaknesses as well as the next steps in order to facilitate the future works in the area of DR.

8 | Gantt diagram

Design and Implementation of Efficient Diminished Reality Mechanisms

Jun 1, 2016

Vicomtech-IK4

http://www.vicomtech.org/

Project manager	Jasone Astorga Burgo, Hugo Álvarez Ponga
Project dates	Mar 1, 2016 - May 31, 2016
Completion	100%
Tasks	16
Resources	3

Tasks

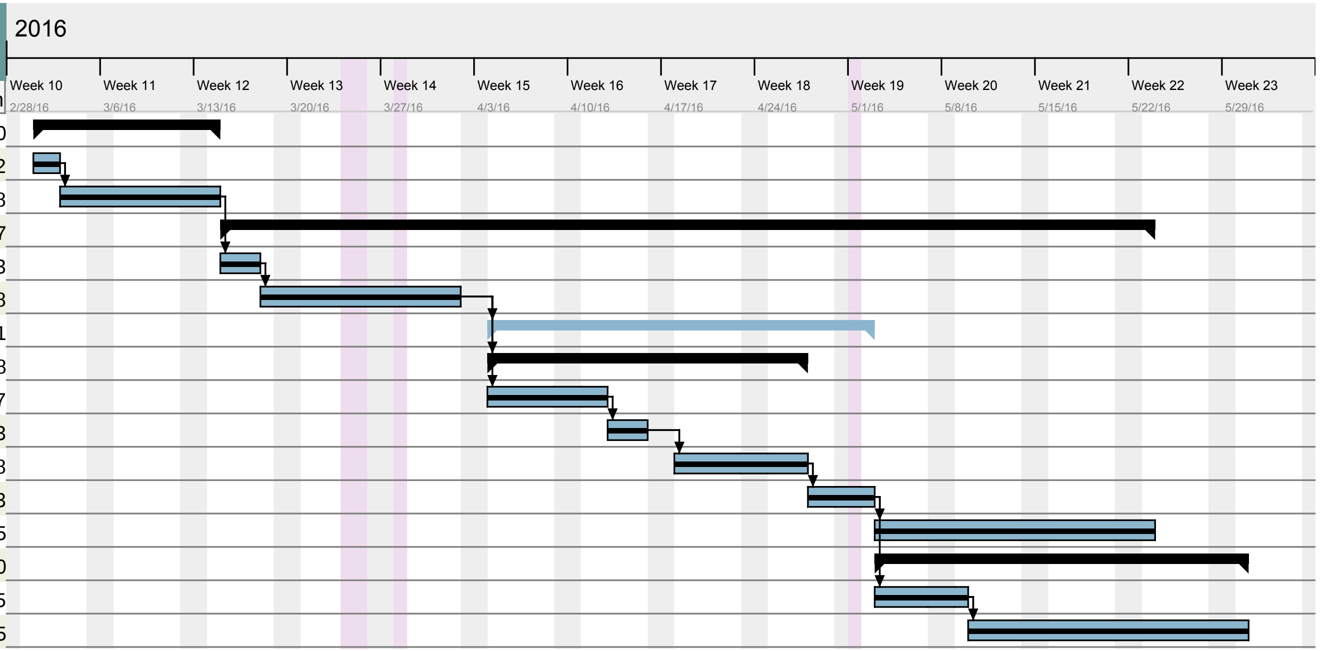
Name	Begin date	End date	Duration
WP1: Project definition	3/1/16	3/14/16	10
T1.1 Project specification definition	3/1/16	3/2/16	2
T1.2 Study of available alternatives	3/3/16	3/14/16	8
WP2: Software design and implementation	3/15/16	5/23/16	47
T2.1 Design and planning of the prototype	3/15/16	3/17/16	3
T2.2 Realization of the initial prototype	3/18/16	4/1/16	8
T2.3 Realization of the final prtototype	4/4/16	5/2/16	21
T2.3.1 Optimizations in the inpainting algorithm	4/4/16	4/27/16	18
T2.3.1.1 Patch search area	4/4/16	4/12/16	7
T2.3.1.2 Pyramidal design	4/13/16	4/15/16	3
T2.3.1.3 Temporal coherence	4/18/16	4/27/16	8
T2.3.2 Optimizations in the tracking system	4/28/16	5/2/16	3
T2.4 Performance and quality tests	5/3/16	5/23/16	15
WP3: Project management	5/3/16	5/30/16	20
T3.1 Project follow-up	5/3/16	5/9/16	5
T3.2 Project documentation	5/10/16	5/30/16	15

Resources

Name	Default role
Jasone Astorga Burgo	project manager
Hugo Álvarez Ponga	project manager
Jon Arrieta Etxeberria	developer

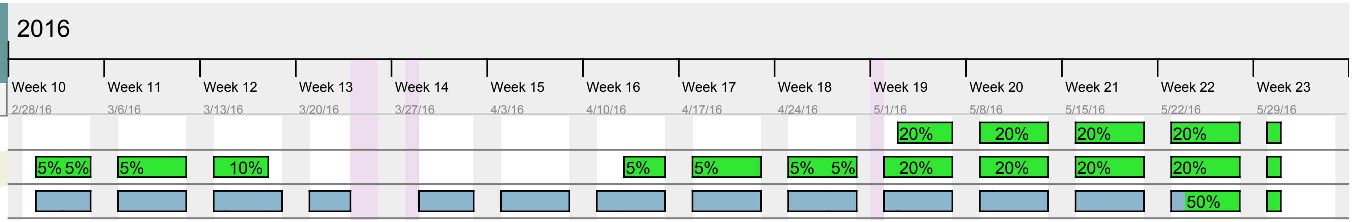
Gantt Chart

Name	Begin date	End date	Duration
WP1: Project definition	3/1/16	3/14/16	10
T1.1 Project specification definition	3/1/16	3/2/16	2
T1.2 Study of available alternatives	3/3/16	3/14/16	8
WP2: Software design and implementation	3/15/16	5/23/16	47
T2.1 Design and planning of the prototype	3/15/16	3/17/16	3
T2.2 Realization of the initial prototype	3/18/16	4/1/16	8
T2.3 Realization of the final prtototype	4/4/16	5/2/16	21
T2.3.1 Optimizations in the inpainting algorithm	4/4/16	4/27/16	18
T2.3.1.1 Patch search area	4/4/16	4/12/16	7
T2.3.1.2 Pyramidal design	4/13/16	4/15/16	3
T2.3.1.3 Temporal coherence	4/18/16	4/27/16	8
T2.3.2 Optimizations in the tracking system	4/28/16	5/2/16	3
T2.4 Performance and quality tests	5/3/16	5/23/16	15
WP3: Project management	5/3/16	5/30/16	20
T3.1 Project follow-up	5/3/16	5/9/16	5
T3.2 Project documentation	5/10/16	5/30/16	15



Resources Chart

Name	Default role
Jasone Astorga Burgo	project manager
Hugo Álvarez Ponga	project manager
Jon Arrieta Etxeberria	developer



Part III

ECONOMICAL ASPECTS

9 | Cost Analysis

This chapter includes the breakdown of all the costs that were generated during the progress of the project. The following classification is done based on the type of cost in the project.

9.1 Personnel cost

Table 9.1 details the role and cost per hour of all the staff involved in the project development:

NAME	CONCEPT	COST
Jasone Astorga Burgo	Project Director	50 €/h
Hugo Álvarez Ponga	Project Manager	50 €/h
Jon Arrieta Etxeberria	Developer	10 €/h

Table 9.1: Worker's cost per hour.

The total cost of the human resources is divided in tasks, that are defined in the methodology section (chapter 7). Table 9.2 shows the cost of each one of the tasks of this project as well as the total cost of the human resources:

	PROJECT DIRECTOR		PROJECT MANAGER		DEVELOPER	
	HOURS	COST	HOURS	COST	HOURS	COST
T1.1	0 h	0 €	3 h	150 €	64 h	640 €
T1.2	0 h	0 €	1 h	50 €	16 h	160 €
T2.1	0 h	0 €	2 h	100 €	24 h	240 €
T2.2	0 h	0 €	0 h	0 €	64 h	640 €
T2.3	0 h	0 €	4 h	200 €	168 h	1.680 €
T2.4	0 h	0 €	0 h	0 €	60 h	600 €
T3.1	8 h	400 €	8 h	400 €	20 h	200 €
T3.2	24 h	1 200 €	24 h	1 200 €	60 h	600 €
TOTAL	**8460 €**					

Table 9.2: Human resource costs.

9.2 Redeemable stock/expenses

The resources like software or hardware that can be used in future projects are summarized here. The total cost shown in table 9.3 is based on the hours that this resource has been used during this project.

CONCEPT	COST	USEFUL LIFE (years)	UNITARY COST (€/month)	QUANTITY (months)	TOTAL COST
Visual Studio	475 €	1	39,58	3	118,75 €
Computer	800 €	4	16,67	3	50 €

Table 9.3: Amortisations.

9.3 Not redeemable expenses

The following table 9.4, includes the costs that took part in this project, but can't be used in another one:

EXPENSES	
CONCEPT	COST
Transport	300 €
Office material	100 €
Office rental	500 €
Telephone and Internet	60 €

Table 9.4: Expenses

9.4 Total project costs

Table 9.5 includes a summary of all the costs that took part during the progress of this project:

TOTAL COST	
Human resource costs	8460 €
Expenses	960 €
Amortisations	168,75 €
TOTAL	**9588,75 €**

Table 9.5: Total cost of the project.

Part IV

Calculations

10 | Description of the Solution

The following part includes the description of the DR solution as well as the results of the tests that show the difference in time and quality.

10.1 System description

As it is mentioned in the introduction (chapter VI), there are three main parts in DR applications. Each one of these layers can use different techniques improving or deteriorating the result. This general structure is divided in object detection, tracking and inpainting modules as it is shown in figure 10.1.

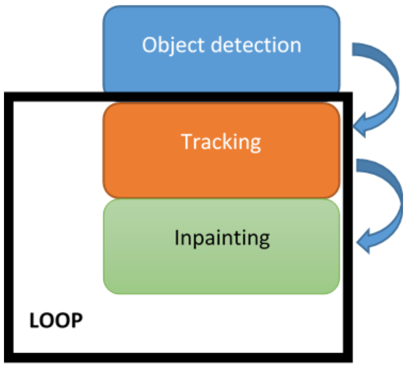
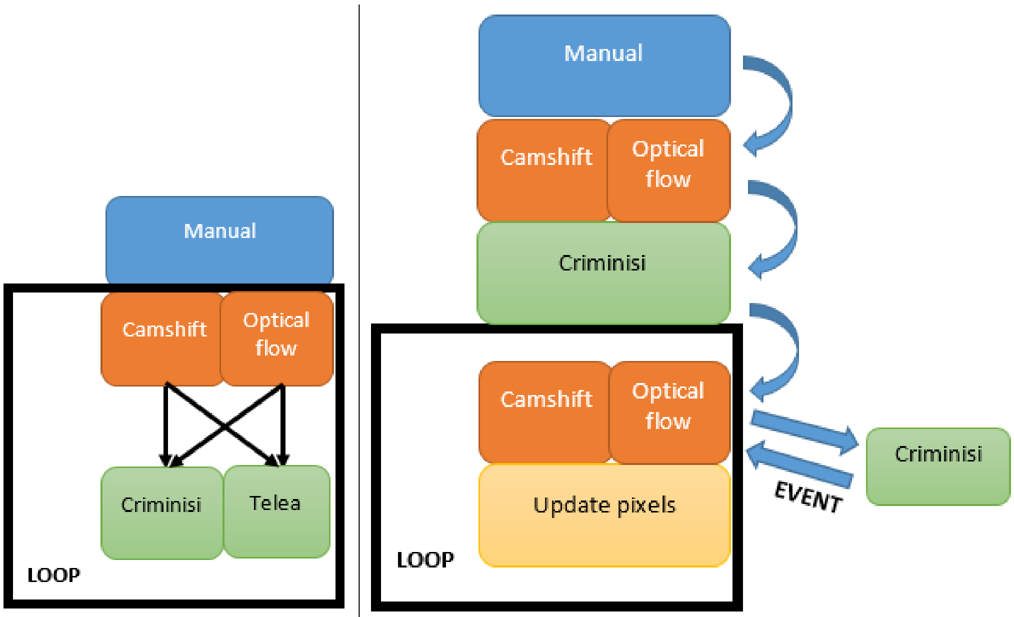


Figure 10.1: General scheme for DR.

This project is designed to be modular in order to ease the testing process for the further study. Thus, it is simple to change between different object detection, tracking and inpainting algorithms, and it is also possible to change optimizations configurations like patch searching area size, pyramid scaling size or activate or deactivate the update of the pixels with homography. All these mechanisms were explained briefly in previous chapters (5, 7) and will be explained with more detail in this section.

In this project, two modes can be defined depending on the execution type. The first one executes the inpainting process in each new frame, whereas the second one applies homography to recalculate the points and it just calls the inpainting function when there is a considerable movement. It is important to know that the object detection is done manually, as the user needs to select the object that needs to be removed. In the case of the Camshift algorithm, it recalculates the shape and position based on a colour histogram. According to the optical flow, the FAST [21] algorithm searches keypoints inside the ROI that will be tracked in subsequent frames. However, it is also possible to implement an automatic object detection algorithm to remove the objects with no user interaction. As this project is mainly focused on the inpainting phase, this automatic detection module is not included in the prototype. Nevertheless, the modular design would help the process of introducing this feature into the prototype for future applications.



a) Inpainting function is executed in every frame.

b) Points are recalculated in every frame using homography and inpainting is executed when there is a considerable movement.

Figure 10.2: Prototype execution modes.

In figure 10.2 both execution modes are represented by a diagram. In the case of figure 10.2 a), there are 4 possible combinations: manual object identification and Camshift tracking combined with Criminisi's or Telea's inpainting algorithm and manual object identification together with optical flow combined with the same inpainting algorithms. In the analysis of alternatives (chapter 5) it is included a comparison between both tracking and inpainting methods.

Figure 10.2 b) consist in a more complex structure, where inpainting algorithm is just called whenever a considerable movement has happened. In the rest of the frames, a pixel updating function will use the homography matrix to renew pixel correspondences. Thus, the processing time is reduced noticeably, although a proper tracking system for each environment is required as well as a good event handler for the movement to maintain as much quality as possible.

10.2 System design

The final prototype is developed under C++ language together with the OpenCV library. As C++ is an object oriented language, the concepts "class" and "method" will be used several times in this section. A class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods).

This project is divided in three clases: Application, DiminishedAPI and Inpainter. Figure 10.3 represents the class diagram of this project. Each one of these classes is simplified in the diagram for the correct understanding of it. For example, there are methods that have been developed and are not used in the final design, so they are not included in this diagram. Apart from that, there are a lot of variables that in each one of them, but give no information about the design. There is only one global structure that is important in this process, because it is shared between all the classes. This global structure is called dataTS and saves information about the target and source pixel matching, for the pixel updating function. As it it shown in the class diagram (figure 10.3), the relationship between the application and DiminishedAPI is 1-n, because the design allows multiple object removal. But the relationship between DiminishedAPI and Inpainter is 1-1, because each DiminishedAPI can handle a single object removal. These classes are explained in more detail in the following paragraphs describing the functionality of each method briefly.

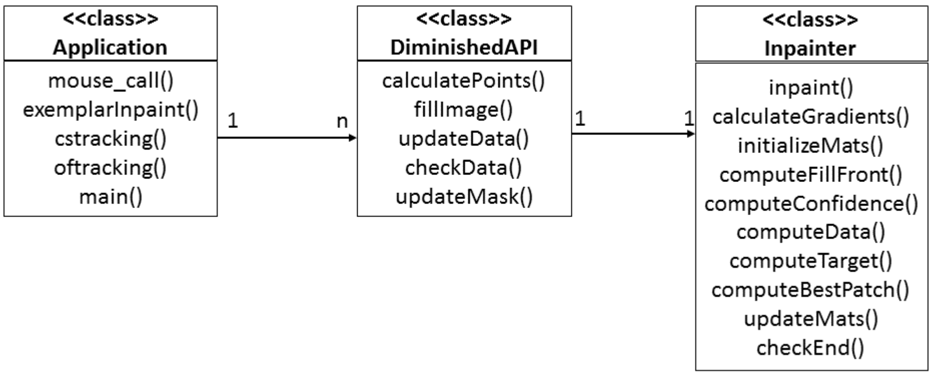


Figure 10.3: Prototype's class diagram.

Application is the main class that controls the whole system. Both tracking systems are included in this class, together with the mouse event handler and the call to the Inpainter. This class loads the video and stops it in the first frame, so that the user can select the object to be removed. Once this object is selected, the user presses "ESC" in the keyboard and the algorithm enters in a loop. This loop is executed until there is no more frames to process. It first passes through the selected tracking method (cstracking() or oftracking())

and then calls to the inpainting method (exemplarInpaint()). This inpainting method controls the DiminishedAPI class to reconstruct the image. Depending on the execution mode, the inpainting method will search the patches or it will update them based on the tracking information.

- **mouse_call():** Detects the actions performed by the user with the mouse and creates a mask that covers the selected area. The selection is done with the right button, the left button is clicked when the selection is finished and the middle (scroll) button clears the selection.
- **exemplarInpaint():** Controls the DiminishedAPI class in order to reconstruct the selected area. The original image and the mask are passed to the DiminishedAPI, as well as the parameters for the reconstruction process.
- **cstracking():** It is the method where the Camshift tracking is performed. It detects the object based on the user selected area's colour information and tracks it during the whole process. The output of this method is a mask that is updated in each new frame.
- **oftracking():** It is the method where the Optical Flow tracking is performed. Keypoints are detected in the Region Of Interest (ROI) and then tracked during the whole video. Keypoints from the previous frame are compared with the ones in the actual frame to obtain the homography matrix. As this method doesn't generate an output mask, the original mask need to be updated by employing the updateMask() method of DiminishedAPI class.
- **main():** It is the method that controls which configuration will be used in the execution. It makes calls to all the methods explained in the previous bullets. There is a main loop that reads the frames and process them one by one until the end of the video.

DiminishedAPI is a class that manages the inpainting process. As the code is modular, this class does not depend on any code or algorithm about object detection or tracking systems. The idea of this class is to introduce an abstract layer that simplifies the use of this DR algorithm. It consist of 5 methods that are explained in the following list:

- **calculatePoints():** This method is called from the Application class in order to calculate the patches of the ROI. First of all, it crops a part of the image that is proportional to the mask size. Then, the cropped part together with the resized mask, is sent to the Inpainter class to reconstruct the image.
- **fillImage():** It uses the information from the global structure dataTS to fill the target region with information from the source region.

- **updateData():** This is the "update pixels" module that it is explained in the system description (Figure 10.2). It uses the information from dataTS structure and updates the position of them using the homography matrix.
- **checkData():** Checks the if the updated information obtained from updateData() passes the range of the image. In other words, it checks if some part of the object is out of the image. If so, the correspondences that are out of the image are removed from dataTS.
- **updateMask():** It is used by the Optical Flow tracking to update the shape and position of the mask, based on the homography matrix. It is applicable to any tracking system that does not generate a mask. It is useless in the case of Camshift because the new mask is automatically generated.

Inpainter is the last class of this DR system. It implements the algorithm proposed by Criminisi *et al.* and includes the optimizations that are proposed in this project. These optimizations are the reduction of the patch search area and the pyramidal design. This class, as well as DiminishedAPI class, has no user interaction.

- **inpaint():** It is the method that coordinates the inpainting process. It enters a loop finding a patch in each iteration and ends when the masked area is covered with patches from the source region.
- **calculateGradients():** It computes the gradients of the image in order to detect the isophotes (structural information) from the image. This information will be used to compute the data term, in order to determine the target patch.
- **initializeMats():** It basically initializes the matrixes that are used in this class. It down-samples the image as well as the mask to fasten the inpainting process (pyramidal design).
- **computeFillFront():** Using a laplacian kernel, the boundary of the mask is calculated. This boundary is also called fill front, and represents the points where the priorities of the filling order are calculated.
- **computeConfidence():** It calculates the confidence along the whole fill front by dividing the amount of pixels contained in the source region, with the pixels from the target region of each possible patch. If there is a high confidence, it means that there are few pixels to reconstruct in that particular patch, which means that the probability of success in the reconstruction is high.
- **computeData():** Data is the other term that is used to calculate the priorities of the patches. In this case, structural information is used to compute the data value. It uses information from the gradients to determine if an isophote is hitting the fill front.
- **computeTarget():** It calculates the priorities based on the confidence and data terms,

and gets the position of the highest priority patch.

- **computeBestPatch():** The target patch that was determined in computeTarget() is compared with patches along the source region. Instead of searching in the whole image, the optimized search of this project searches in the neighbourhood of the target patch. For more information please refer to "Optimizations" section of this chapter.
- **updateMats():** Once best patch has been matched, the gradient and colour values from the source patch are mapped to the target patch. At the same time, the information about the pixel correspondences is mapped to the original scale and saved in the dataTS structure.
- **checkEnd():** It checks if the masked region has been fully covered.

All the description about the classes and methods functionalities is summarized in the sequence diagram included in figure 10.4. It is explained step by step how the algorithm works, the relationships between them and the main methods that take part in each action.

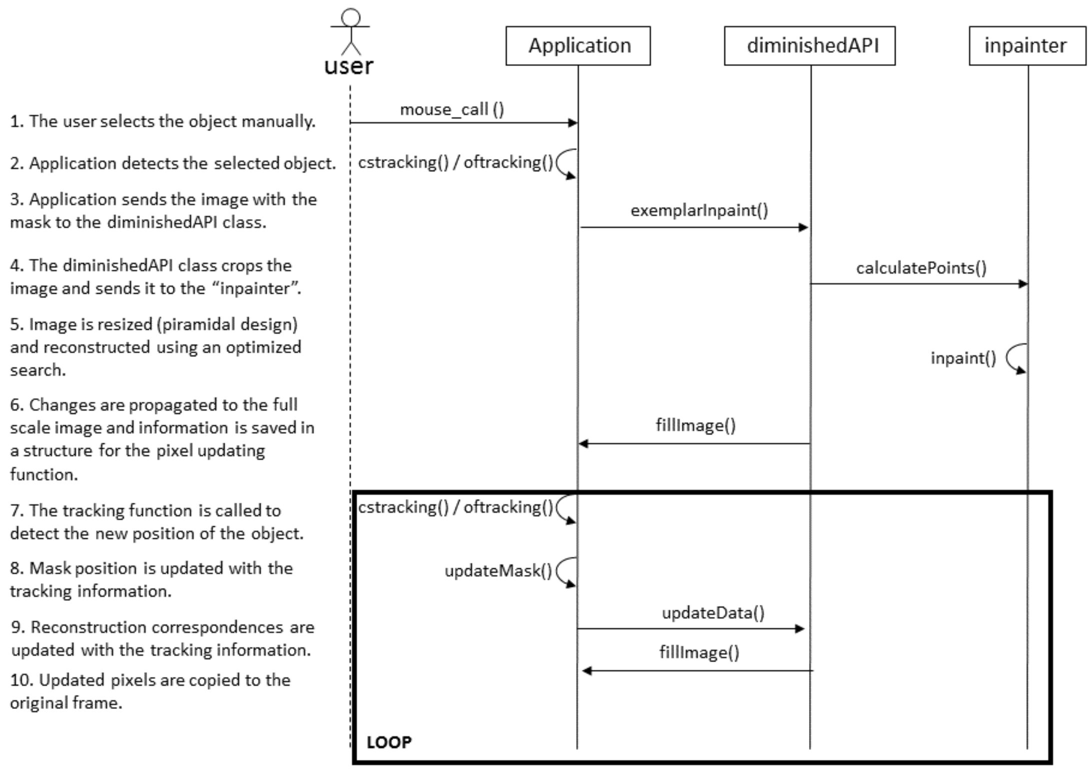


Figure 10.4: Prototype's sequence diagram.

10.3 Optimizations

This project is based on Criminisi's algorithm, but it has not been implemented in its original version because the algorithm is too heavy to implement it in video applications. Therefore, optimizations have been implemented to speedup the process of inpainting. But first of all, it is necessary to know the basics of Criminisi's algorithm to understand the optimizations that were introduced in this project.

10.3.1 Background: Criminisi *et al.* inpainting algorithm

Criminisi et al. made a great contribution to inpainting algorithms by introducing a self-contained structure and texture propagation technique. Figure 10.5 helps to understand the concepts of Criminisi's filling order.

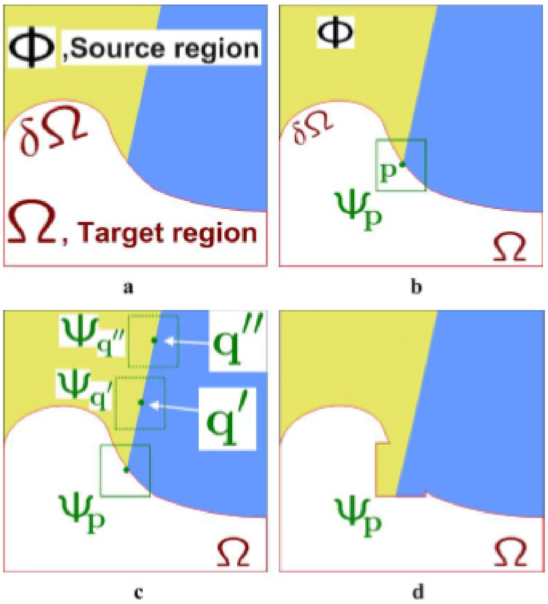


Figure 10.5: **Fundamentals of Criminisi's inpainting algorithm.** a)Definition of relevant image regions. b) Selection of the target patch. c) Possible source patches. d) Result.

The *target region* (Ω) is the masked area that needs to be filled with patches from the *source region* (Φ), and $\delta\Omega$ is the boundary of the target region which is going to be computed to define priorities. According to figure 10.5 b), Ψ_p patch has been defined as the highest priority patch to be filled. This priority is defined by two terms: confidence and data. *Confidence* is basically a relationship between the amount of pixels of the patch placed in the source region and the total amount of pixels of the patch. On the other hand, *data* is a function of the strength of isophotes hitting the front $\delta\Omega$ at each iteration. Both of these terms are calculated and multiplied in each iteration in order to define priorities to determine the best patch to fill first. Figure 10.5 c) shows possible source patches that fit with the selected target patch. Even though just two patches are marked, SSD (Sum of Squared Differences) is used in the whole image to compare the target patch with any possible patch in the image. Once the selection of the best patch is done, the source patch is pasted in the target patch as it is shown in figure 10.5 d).

10.3.2 Reduction of the patch search area

One of the most important changes from the Criminisi's original algorithm is the change of the patch searching area. As it is stated in the analysis of alternatives (chapter 5), the adapted algorithm searches patches only in the surroundings of the target patch instead of searching in the whole image. Even though there is a loss of information, in most of the cases this information is useless for the reconstruction, and therefore, a pointless processing. During the progress of this project, it has been observed that in many cases, the closest patches from the mask were the best matches for the target patch. This is coherent because the lighting conditions share more similarities when a patch is searched around an approximate area. The search area defined in this project is proportional to the mask size and can be changed for user preferences.

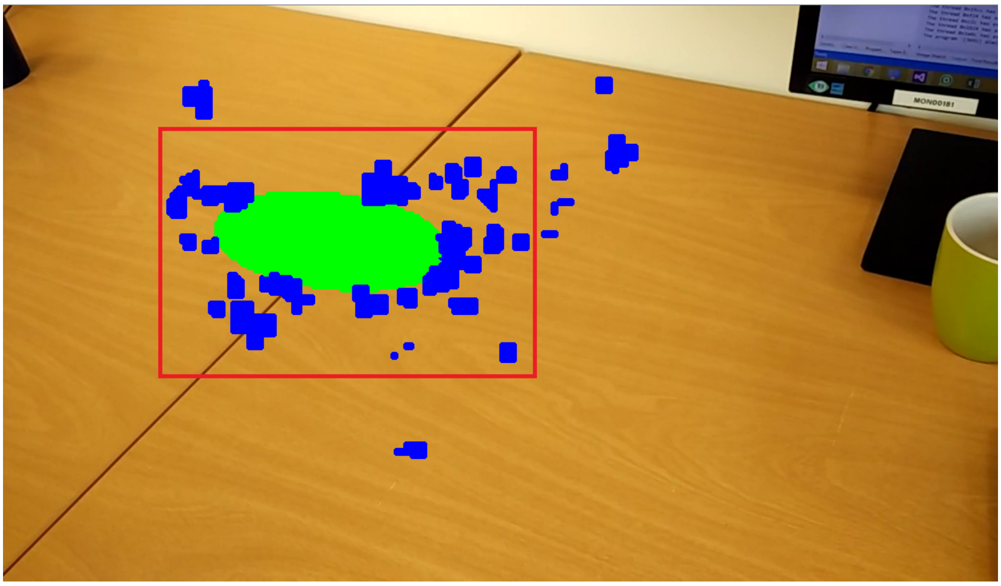


Figure 10.6: Example of patch correspondences.

Figure 10.6 show the target region marked in green, and the source patches marked in blue. As it can be seen in the image, most of the source patches come from the surroundings of the masked region, and therefore, a reduction of the search area would improve the processing time with a small impact on the result. The red area is an example of a search area. This search area size can be defined by the user before executing the code and it is fixed during the whole video processing.

10.3.3 Pyramidal design

A common optimization technique used in image processing is the use of image pyramids. In this technique, the image is downsampled, and therefore, less pixels are processed, reducing the processing time substantially. It is important to note that if there is a reduction in processing time, there is also a reduction in the reconstruction quality. Even though downsampling rate can be configured for each execution, a fixed rate of 4 has been used for the testing process. Thus, a balance between speed and quality has been assured.

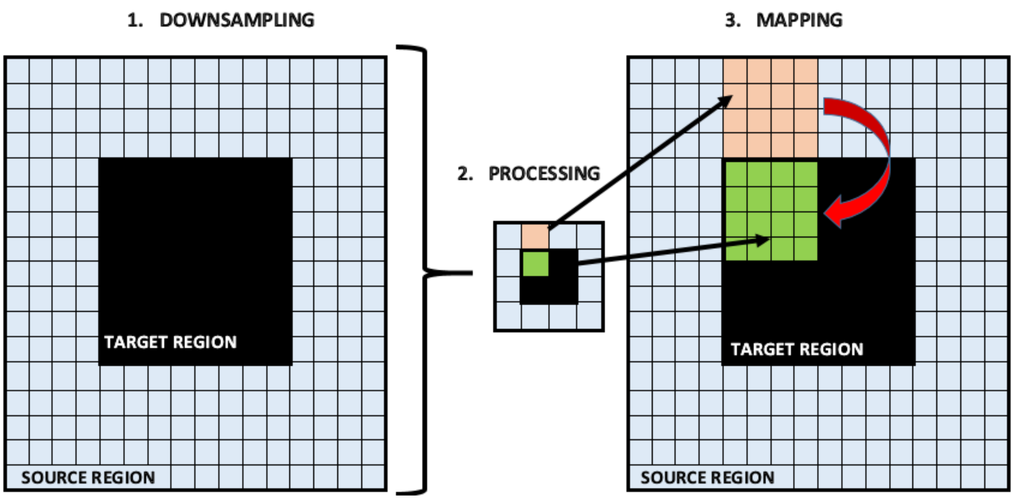


Figure 10.7: Multi-scale pixel mapping.

Figure 10.7 shows the scheme that has been implemented in this project for mapping pixels from the undersampled image to the full scaled image. The orange patch in the undersampled image is mapped to a corresponding position of the patch in the original size. The same thing happens with the green patch. The black area is the mask or target region, whereas the blue area is the source region. Finally, the red arrow shows how the source patch (orange) is copied into the green patch.

Pyramidal design has been applied in the inpainting process to reduce the time of both, the computation of the patch priority as well as the search of the patch correspondence. After completing the whole inpainting process in the coarsest pyramid layer, patch correspondences are directly mapped to the finest pyramid layer.

10.3.4 Homography

The last optimization included in this project is the homography technique. Homography matrix is calculated by finding a perspective transformation between two planes. In this approach, homography between two consecutive frames has been calculated using the tracking information. Thus, a matrix to transform the points is obtained, and inpainting patch correspondences can be updated to the new values instead of calculating them in each frame.

The efficiency of this method relies on having an accurate tracking system. For example, camshift tracking system is based on colour characteristics and detects and tracks the object with an accurate precision. Nevertheless, this tracking system is not robust against rotation, translation or other kind of transformation, as it just marks four points around the object. On the other hand, Lucas-Kanade optical flow [19] tracks feature points in the ROI, which are fixed over the time. Therefore, the obtained precision in the homography is higher and the algorithm is more robust against the movements of the camera. However, it is important to know that objects need to have some peculiarities to detect these feature points as well as for tracking. For example, it is hard to track an homogeneous object with the optical flow. Figures 10.8 and 10.9 show the difference between both tracking systems.

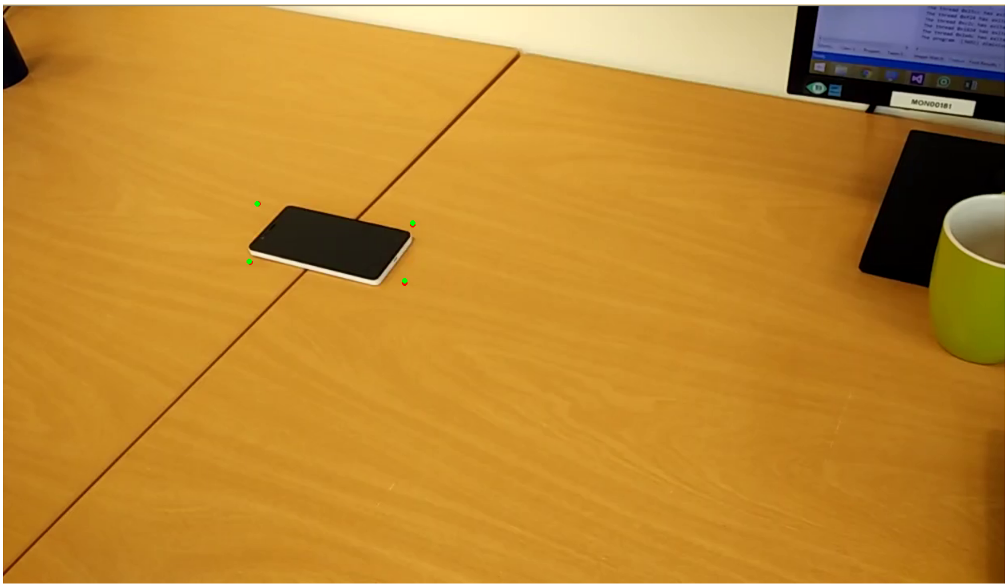


Figure 10.8: Camshift tracking example.

The green spots represent the new position of the object, whereas the red ones represent the position of the object in the previous frame. In figure 10.8 (Camshift) there are just eight points (four from the previous frame and four from the actual) to calculate the homography matrix, whereas in figure 10.9 (optical flow) there are dozens of points. Apart from that, the four points obtained by camshift are not stable over the time because the tracking is based on the object's colour, whereas the optical flow maintains the points in the same place of the object. Hence, the homography matrix is more precise in the case of the optical flow rather than calculated with the Camshift function.

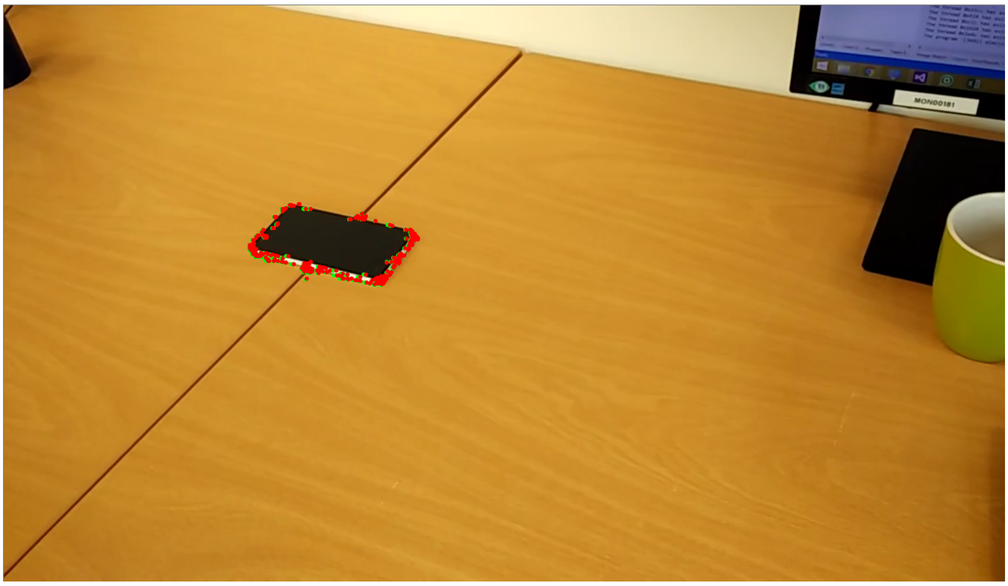


Figure 10.9: Lucas-Kanade optical flow example.

In figure 10.2 (execution modes) b) there is a function called update pixels. This function employs homography matrix to recalculate pixel correspondences, reducing processing time noticeably and maintaining visual quality. Whenever there is a considerable movement, the inpainting function is called to recalculate the patches again, increasing the processing time of the frame. Considering the results of one of the tests that has done, a frame calculated by the inpainting function would last around 2 seconds, whereas a pixel updating function lasts around 0,05 seconds. Implementing an efficient tracking algorithm improves the homography matrix, and hence, movement margin can be increased reducing processing time.

10.3.5 Movement control

As it is mentioned in the previous section, this project implements an event handler to detect the movement of the camera relative to the object. It is calculated by detecting the center of the object from the tracking points, and comparing it to the original position. When this distance exceeds a threshold, the inpainting function is called to recalculate the patches and the "original" position is updated. The threshold can be configured manually by the user in each execution.

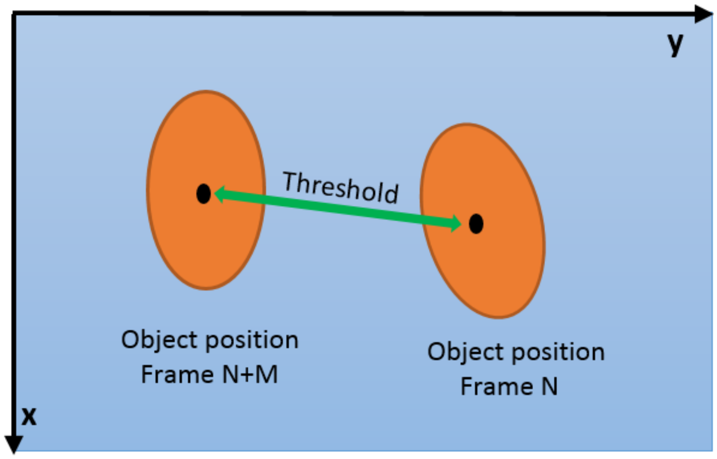


Figure 10.10: Scheme of the movement control.

Figure 10.10 is a scheme of the movement control implemented in this project. The orange region is the object's shape in the image, and the calculated center point is the black dot. The green arrow is the distance from the original center to the center in the subsequent frame. This distance is compared with a threshold in every iteration.

With the help of this event handler, the inpainting function is called just whenever a considerable movement has happened. Thus, computational time is affected comparing with a fully homography patch updating application, but the output quality is improved because the error introduced by the camera movement is corrected.

11 | Tests and Results

As it is stated in the objectives (chapter 2), this project includes also a performance and quality study of the prototype. This section is divided in three subsections: testing design, testing results and conclusion.

11.1 Testing design

In this project, there are two indicators that will be crucial for the success or failure of it. These indicators are quality and speed.

The main module of this project is the inpainting process and, as it is mentioned in the analysis of alternatives (chapter 5), the algorithm should maintain a balance between quality and speed. Speed is a parameter that can easily be measured objectively, and therefore, the results that are obtained from this kind of measurement can directly be interpreted. However, the quality is a subjective indicator that needs an extra step to proof its validity. In order to quantify the quality, a form with videos that were processed with different configurations was prepared, and the results of it are exposed in the present document. Figure 11.1 is a screen-shot of the form that was used in this project. As it is shown, users evaluate each video from 1 to 5, being 1 the worse quality and 5 the best one.

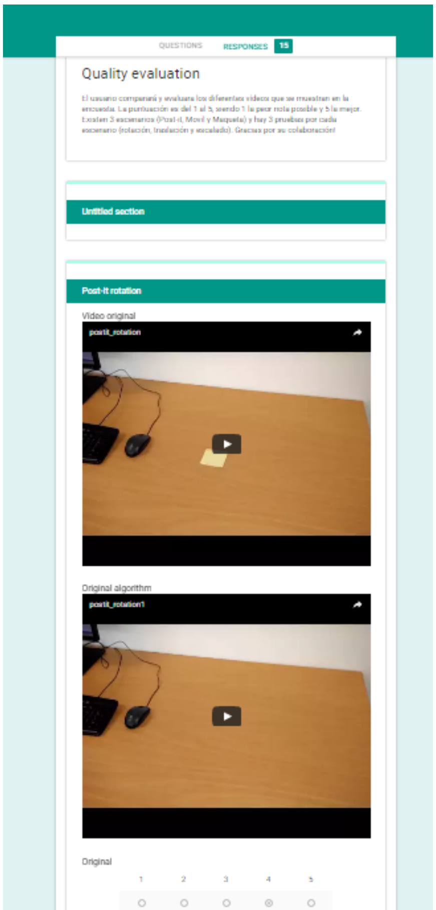


Figure 11.1: Form used for the quality test.

The modular design allowed to change from one configuration to another easily in the process of testing. Hence, the intention was to create a progressive testing process, starting from the original algorithm and introducing each optimization until the final prototype. It was also executed in several scenarios, starting from the most simple one until the most complex one.

A classification can be done according to the scenarios:

- **Simple scenario:** Consists in removing an object in a non-structural background. For example, an object on the top of a table.
- **Medium scenario:** Consists in removing an object in a simple structural background. For example, an object that partially occludes a line.
- **Complex scenario:** Consists in removing an object in a complex structural background. For example, an object in front of a building.
- **Outdoor scenario:** Consists in removing an object in an outdoor scenario. For example, removing a signal or a car matricule number.
- **Moving object with a simple background:** Consists in tracking and removing a moving object with a monocolor background. For example, a small ball in a table.
- **Moving object with a complex background:** Consists in tracking and removing a moving object with a structural background. For example, a small ball in a structured floor.

Each scenario was recorded with three different transformations. To understand the axes kindly refer to figure 11.2:

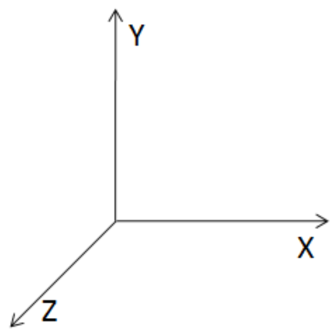


Figure 11.2: Reference axis.

- **Rotation:** The camera rotates 180° around the "y" axis. One side of the object is shown in the first frame and the other side in the last one.
- **Translation:** It is a camera moving along the "x" axis. The object starts in the left side of the video frame and ends in the right side.
- **Scaling:** The camera moves towards the "z" axis. The object's size is increasing as the video progresses.

On the other hand, there is a classification of configurations:

- **Original:** Consists in executing the original algorithm proposed by Criminisi [3] frame by frame.
- **Original + Search window:** Consists in executing the original algorithm plus the first optimization which consists in reducing the patch search area.
- **Original + Search window + Image pyramids:** Consists in executing the algorithm with the optimized search and using image pyramids to accelerate the process.
- **Final version:** Consists in executing the algorithm with all the optimizations mentioned in this project. Patches are calculated just when there is a big movement, and the rest of the video processing consists in finding the correspondence using the homography matrix.

Because of the long execution time of the original algorithm (around 25 hours for a video of 7 seconds), some of these tests weren't performed for this project. However, the testing process will continue even after the end of the project in order to have a consistent measurement of the solution. Therefore, this project includes the tests of three scenarios (simple scenario, medium scenario and complex scenario) in all the configurations mentioned before with Camshift tracking, and the final version with the optical flow tracking. Actually, this is the way that this project progressed, starting from the original algorithm, introducing optimizations and changing the tracking system to a more precise one. Moreover, these tests already provide enough information to show clear results.

All these cases generated timing information as well as the output video to be able to compare and get some conclusions. After the completion of this task, all the information was saved in an excel file to be able to compare and calculate differences between optimizations. The computer where the tests were performed integrates an Intel i5-3470 CPU @ 3.20 GHz, 8 GB RAM and a NVIDIA GeForce GTX 560 graphic card running under windows 8.1.

Nine videos were recorded in three scenarios in order to have a consistent result. Each scenario had three videos, recording the object with different transformations: rotation, translation and scaling. These videos were recorded in HD quality (720p and 30 FPS (Frames Per Second)).

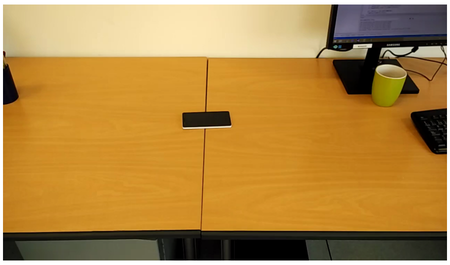
A post-it on the top of the table was the scenario used to simulate a simple scenario. This table had an homogeneous texture that was propagated to cover the object of interest (the post-it). Figure 11.3 shows a frame of the video that was used for the testing process.

(a) Input image.

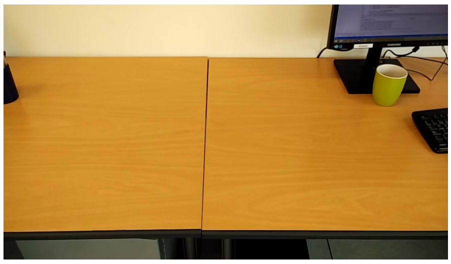
(b) Processed image.

Figure 11.3: Simple scenario (Post-it).

The second scenario presented a difficulty in the inpainting process, because in this case, apart from propagating textures there was a need to propagate structures too. The medium scenario consisted of a mobile phone in a line that was generated because of the junction of two tables. Figure 11.4 represents a frame that was used to simulate the medium scenario.



(a) Input image.



(b) Processed image.

Figure 11.4: Medium scenario (Mobile phone).

The third and last scenario was a mock-up of a building, that represented even more complexity than the previous one. In this case, the building had a unique colour and several structural components that made it difficult for the reconstruction process. Figure 11.5 represents a frame of this scenario.

(a) Input image.

(b) Processed image.

Figure 11.5: Complex scenario (Mock-up of a building).

11.2 Testing results and discussion

Results are divided in two groups based on the indicators mentioned in the previous subsection. These subsections include processed information in form of charts for the correct understanding, but if a further information is needed, Appendix II includes rough information in form of tables or diagrams describing whole timing information as well as the results of the form for the quality measurement.

11.2.1 Timing results and discussion

Table 11.1 is the reference that will be used to compare the results in different scenarios. This table has three columns that represent the transformations of each one of the scenario, and each one of the cells include a diagram, the total number of frames and the processing time of the algorithm for each case.

According to the diagram, the "x" axis represent the frame number and the "y" axis represent the time in a logarithmic scale. The logarithmic scale of the "y" axis is used to differentiate better between optimizations, because there is a big gap between the original algorithm and the rest of the solutions. The color code is shown in the lowest part of the table and it is applicable for all the charts included in it.

All the charts share similar patterns that show the consistency of the tests that were performed to obtain the information. The difference in time between the optimizations is constant in all charts, which means that all the optimizations that are implemented in this project have a linear effect in the result. As it can be observed in the "Original", "Original_Window" and "Original_Window_Pyramid" lines, rotation and translation charts share almost the same values for the whole video. But in the case of the scaling charts, there is an increase of the processing time in the latter frames. This happens because the object that needs to be inpainted becomes bigger, and as a result, more patches are needed to cover the area, augmenting the processing time.

With respect to the final versions with Camshift and Lucas-Kanade optical flow tracking, it can be seen that they have a really low processing time in most of the frames. The pixel updating function based in the homography matrix is the responsible of this improvement. In the case of the optical flow, there is a peak in the beginning where the patches are calculated using the inpainting function and the rest of the video maintains low, because the precission of this tracking algorithm allows more movement without calculating the patches again. On the other hand, the Camshift line (purple) shows peaks all over the video because it needs to recalculate the patches to maintain a good quality of reconstruction. As it is expected, these peaks coincide with the processing time of "Original_Window_Pyramid" as they share the same configuration for the image reconstruction process.

Another interesting effect is the small gap between the optical flow and Camshift approaches. This might have happened because the optical flow needs more time for processing when there are several points to track than the Camshift that is only based in colour detection. Therefore, in the last scenario, the gap between these tracking systems is higher because the card has more keypoints than a plain post-it.

The only undesired effect, is the drop in the processing time of the rotation, in the mobile phone scenario. This is due to a Camshift tracking failure, where the resulting mask is smaller and not even able to cover the object. This failure might have happened because the lighting conditions have changed considerably as a result of the rotation. Hence, Camshift detects just a part of the object and the reconstruction becomes erroneous in some frames. Apart from that, the movement event handler is affected because the center of the object is not detected properly, forcing the algorithm to calculate the patches again increasing the overall processing time in the final version with Camshift tracking.

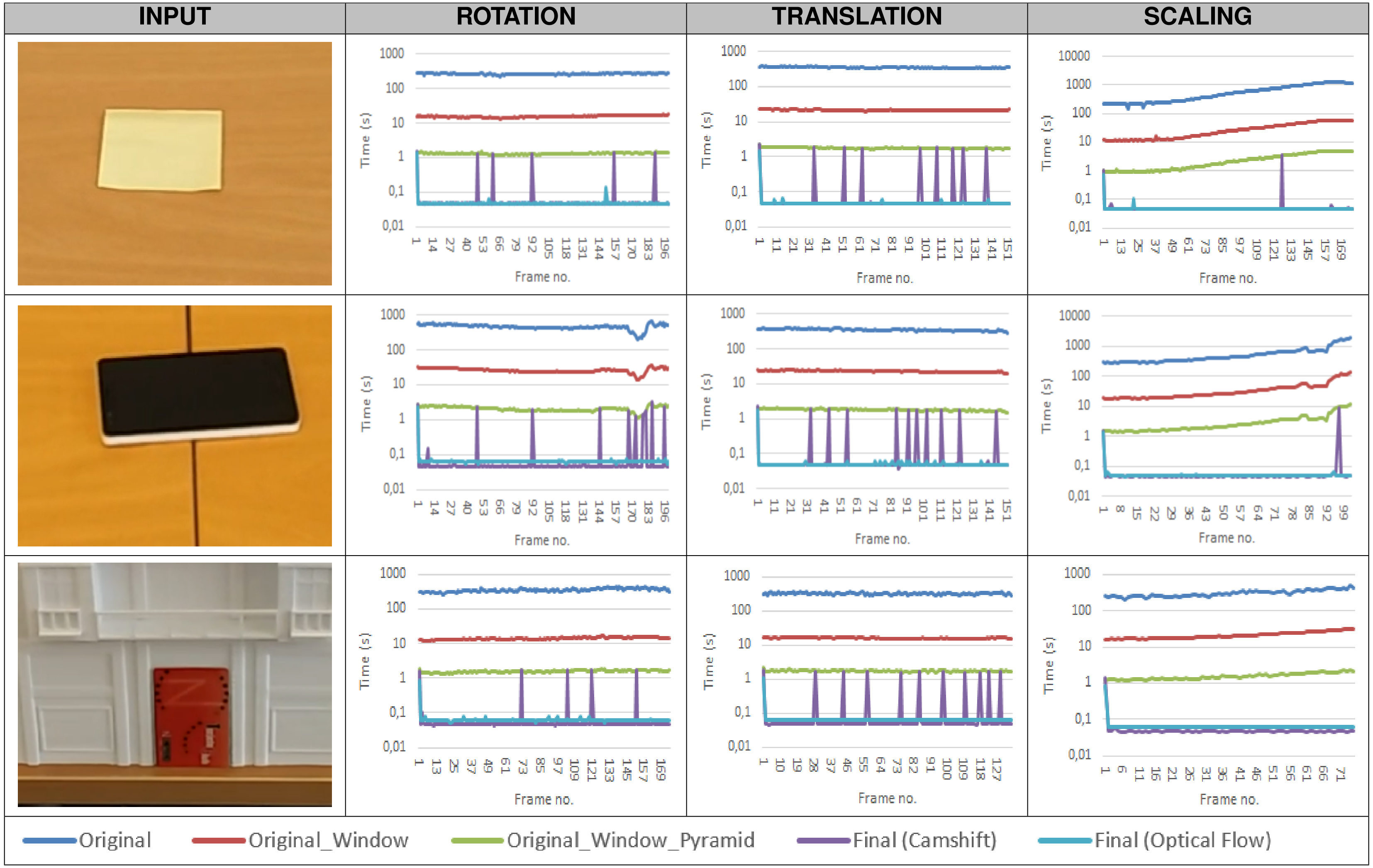


Table 11.1: Timing comparison table.

Tables 11.2, 11.3 and 11.4 include the total processing times of each optimization in all scenarios mentioned above. It is easy to identify that there is a huge improvement between the original algorithm executed frame by frame and the final version of the prototype with any of the tracking systems. However, considering the video quality that was used in this project (720p), it is not capable of processing all the frames with the same input FPS (in some scenarios they reach 19,49 FPS against 25 of the input video). Apart from that, the peaks introduced by the event handler and the inpainting function, affect considerably in the final result. Nevertheless, this problem can be solved by implementing multi-core support using threads. In this case, the patches would be calculated in parallel to the tracking and pixel updating function, and would not introduce the peaks that are shown in the purple lines of the table 11.1.

CONFIGURATION	ROTATION (199 frame)	TRANSLATION (152 frame)	SCALING (177 frame)
Original	52862,84 s	53526,35 s	105292,73 s
Original_Window	3097,53 s	3226,45 s	5098,17 s
Original_Window_Pyramid	259,69 s	266,97 s	425,49 s
Final (Camshift)	17,04 s	23,12 s	12,85 s
Final (Optical Flow)	10,77 s	8,62 s	9,08 s

Table 11.2: Simple scenario total processing times.

CONFIGURATION	ROTATION (199 frame)	TRANSLATION (152 frame)	SCALING (102 frame)
Original	92435,89 s	53548,90 s	57551,18 s
Original_Window	5209,06 s	3478,99 s	3712,20 s
Original_Window_Pyramid	409,29 s	283,65 s	309,46 s
Final (Camshift)	31,00 s	37,49 s	15,47 s
Final (Optical Flow)	14,80 s	22,33 s	6,18 s

Table 11.3: Medium scenario total processing times.

CONFIGURATION	ROTATION (175 frame)	TRANSLATION (135 frame)	SCALING (75 frame)
Original	61440,18 s	43449,94 s	22814,54 s
Original_Window	2467,36 s	2119,33 s	1574,21 s
Original_Window_Pyramid	278,24 s	232,88 s	117,40 s
Final (Camshift)	16,69 s	23,61 s	4,91 s
Final (Optical Flow)	11,83 s	9,40 s	5,46 s

Table 11.4: Complex scenario total processing times.

11.2.2 Quality results and discussion

Quality measurement results show similar tendency in all the scenarios that were tested. In most of the cases, the original algorithm obtained the lowest punctuation by the users. Followed by the pyramidal design that obtained a slightly lower punctuation in comparison with the optimized patch search area. Then, even if the the final version with Camshift has a good punctuation, the clear winner is the final version with optical flow tracking system. The main difference between the last two approaches and the rest, is the temporal coherence that reduces fluctuations in the reconstructed image improving the visual perception considerably. Table 11.5 shows the average punctuation of the users for each configuration.

CONFIGURATION	Original	Original_Window	Original_Window_Pyramid	Final (Camshift)	Final (Optical Flow)
PUNCTUATION	1,73	2,61	2,37	3,30	4,54

Table 11.5: Average punctuations of each configuration.

Another representation of the data obtained from the form is shown in figure 11.6, where the tendency of the algorithm in each scenario can be observed individually. Some scenarios have higher punctuation than others, and that is obvious, because a reconstruction can be better or worse depending on the scenario and transformation. But a good point is that these tests show similar trends, proving that it is not a coincidence that the users chose the last approach as the best one.

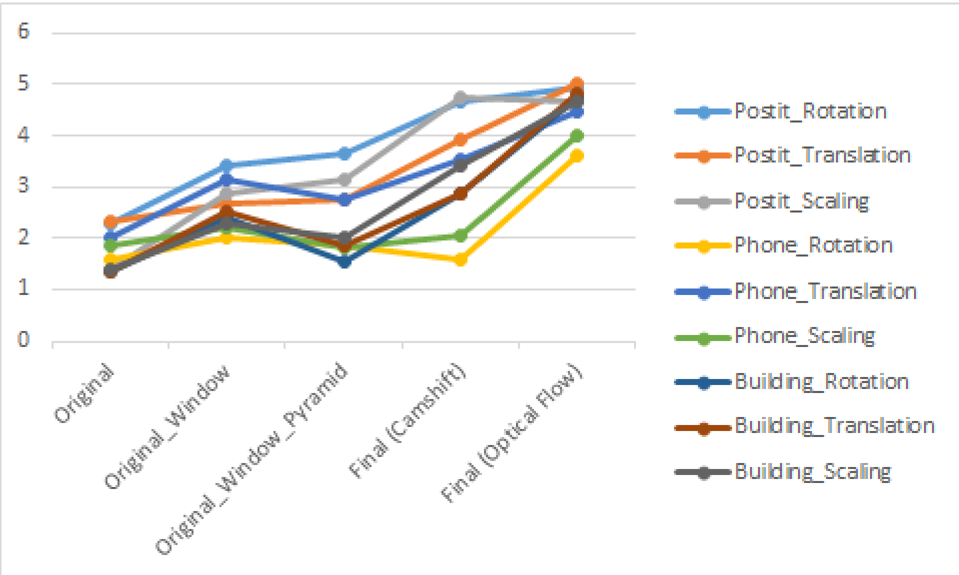


Figure 11.6: Quality measurement chart.

11.3 Summary

Summarizing, it has been demonstrated that the final version of this prototype is faster than the original algorithm executed frame by frame. Even though it is not possible to quantify the improvement (because it depends on the scenario, object size and movement), the results in all tests show that it is thousands times faster (around 4000 to 5000 thousands generally). In addition, the users that completed the form determined that the final version of the prototype (with the optical flow tracking) obtains the best output video qualities. Hence, it can be said that the optimizations included in this project not only improved the processing time but also the output video quality.

Part V

CONCLUSIONS

12 | Conclusions

First of all, it is important to remark that the objectives that were set in the beginning of the project have been fulfilled successfully. A DR system that implements Criminsi's algorithm has been developed in this project. The algorithm has been optimized by limiting the search area of the patches, introducing a pyramidal processing and adding temporal coherence restrictions. Apart from that, two tracking systems (Camshift and Optical Flow) were implemented to avoid having to call the inpainting process all the time, assuring robustness in random scenarios. Finally, a performance study was accomplished, which proved that the solution proposed in this project improves considerably the original algorithm in terms of processing time and quality.

Existing inpainting techniques like the ones proposed by Barnes et al., Criminisi et al. or Wexler et al. have been considered for the image reconstruction process as they are the reference in the area. After an exhaustive research in the literature, Criminisi's approach was elected for the inpainting process because of its ability of propagating structures without user interaction. Most of the inpainting techniques share the weakness of the processing time, as they consume minutes to inpaint a single frame. This problem has been solved in this project by optimizing the algorithm presented by Criminisi et al. and introducing temporal coherence for video sequences.

Maintaining a balance between quality and speed was crucial for the success of this project. It has been proved that the final version is not only faster, but has also a better quality than the original algorithm executed frame by frame. In conclusion, the final prototype has achieved efficiency and robustness in different scenarios showing that it is capable of propagating structures and textures, with a minimal user interaction, almost in real-time.

This prototype can be extended by implementing new modules and spreading out the testing process in order to identify possible improvements in the algorithm. For example, there are peaks in the processing time that are introduced by the inpainting function. These peaks can be removed by implementing multi-core support, calculating the patches in parallel to the pixel updating function.

Bibliography

[1] Norihiko Kawai, Takao Sato, and Naoto Yokoya. Diminished reality considering background structures. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 259–260. IEEE, 2013.

[2] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.

[3] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *Image Processing, IEEE Transactions on*, 13(9):1200–1212, 2004.

[4] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

[5] Uwe Kühnapfel, Hüseyin Kemâl Cakmak, and Heiko Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & graphics*, 24(5):671–682, 2000.

[6] Siavash Zokai, Julien Esteve, Yakup Genc, and Nassir Navab. Multiview paraperspective projection model for diminished reality. In *Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on*, pages 217–226. IEEE, 2003.

[7] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):463–476, 2007.

[8] Jan Herling and Wolfgang Broll. Advanced self-contained object removal for realizing real-time diminished reality in unconstrained environments. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 207–212. IEEE, 2010.

[9] OpenCV. `http://opencv.org/documentation.html`.

[10] Photoshop CS. `http://www.adobe.com/es/products/photoshop.html`.

[11] GIMP. `https://www.gimp.org/`.

[12] Kolor Autopano Video. `http://www.kolor.com/`.

[13] Vincent Lepetit, Marie-Odile Berger, and LORIA-INRIA Lorraine. An intuitive tool for outlining objects in video sequences: Applications to augmented and diminished reality. *tC*, 2:t3, 2001.

[14] Jan Herling and Wolfgang Broll. Pixmix: A real-time approach to high-quality diminished reality. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 141–150. IEEE, 2012.

[15] Norihiko Kawai, Masayoshi Yamasaki, Tomokazu Sato, and Naokazu Yokoya. [paper] diminished reality for ar marker hiding based on image inpainting with reflection of luminance changes. *ITE Transactions on Media Technology and Applications*, 1(4):343–353, 2013.

[16] Boris Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[17] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[18] Gary R Bradski. Real time face and object tracking as a component of a perceptual user interface. In *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on*, pages 214–219. IEEE, 1998.

[19] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[20] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[21] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

Part VI

APPENDIX I: Bid specifications

Introduction

The following document includes the conditions that the contractor must accept in order to assure the realization of this project.

These conditions cover all the information about the implementation of the software, as well as the warranty and maintenance that will assure the validity of the project once it is finished.

To validate this project, it needs to undergo the tests of verification that assure the agreement that was established in the beginning of it, as well as what it is exposed in the report attached in this writing, in accordance with every specification and functionality mentioned.

Technical specifications

The following section describes the requirements that need to be accomplished for the correct performance of this project, no matter if they are material resources or human resources. Because of the complexity of this project, material resources are divided in two: hardware resources and software resources.

Material resources

Hardware resources

The necessary hardware requirements for the correct development of this project are shown in the list below:

- 3 Personal computers with an Intel Core 2 Quad at 2,4 GHz and a minimum RAM of 4GB.
- Nvidia GeForce GT 730 graphic card or better.
- Lead and cables.

Software resources

The necessary software resources that assure the correct development of this project are listed below:

- Windows 7 or higher (Windows 8 / Windows 8.1 / Windows 10).
- Visual Studio for C++.
- OpenCV library.

Human resources

The required profiles for the development, installation and configuration of this project are listed below:

- **Project director:** It is needs to be a senior engineer with experience, but not necessarily an expert in the area. The main tasks are coordinating the project and structuring the documentation process.
- **Project manager:** It needs to be a senior engineer with experience in computer vision. The main tasks are supervising the project and the documentation process.
- **Developer:** It is a junior engineer with some knowledge in computer vision techniques.

Documentation of the project

The documents that are part of the project are described in this section. The documentation establishes the functionalities and characteristics of the project, and therefore, they are taken as reference to proof that it accomplishes with what it is agreed.

Document No.1 - REPORT

This document includes all the relevant information about why this project should be developed and the benefits that this project can generate.

The objectives and different design alternatives are described in it, evaluating and selecting the characteristics that fit better to this project. The state of the art is also included in this document.

Apart from that, it incorporates a summary of the human resources, as well as the material resources, fixing a working plan with the tasks that need to be fulfilled and its terms.

Document No.2 - METHODOLOGY

This document includes the working plan that was followed for the correct realizations of this project, presenting the human resources that took part in it, defining the tasks that were performed and the terms. A Gantt diagram is included summarizing all the information described.

Document No.3 - CALCULATIONS

This document offers a detailed description of the system as well as a study of the performance of it. It includes the high level design describing the connections and functionalities of the modules, but it also includes a low level design for the correct understanding of each one of them.

Document No.4 - ECONOMICAL ASPECTS

This document includes a cost analysis of the project divided in four sections. The first one describes the personnel costs task by task; the second one describes the redeemable stock/expenses; followed by a description of not redeemable expenses; and finishing with a summary of the whole costs.

Document No.5 - BID SPECIFICATIONS

It refers to the actual document. In this document the conditions and responsibilities for the client are described.

Document No.6 - DRAWINGS AND DIAGRAMS

This document includes all the necessary drawings and diagrams for the correct understanding of the solution presented in this project.

Reception conditions

The conditions that need to be verified and accomplished in the moment of the delivery of the project are described in this document.

Delivery terms and execution

Based on the terms fixed in the methodology described in the attached document (Document No.2 - METHODOLOGY), the delivery of this project must be done by the 30th of May.

Any other proposal or changes of the dates established in the methodology after the subscription of both parts, must be approved and discussed by both of them, establishing respective modifications in the appendix(es) of the project, in the contract and in the estimate evaluation of the total cost of the project (if this is affected).

Product exploitation rights

The resulting prototype as well as the study of the optimizations is going to a shared property between the client and the projector, and the results of it will also be shared between both of them, being possible for both parts to use them freely and with no need of consent from the other part.

Financial terms

This section includes the economical conditions that need to be fulfilled for the delivery of the project.

Project cost

The total amount of the budget for the project *"Design and implementation of efficient diminished reality mechanisms"* is fixed at nine thousand five hundred and eighty eight coma seventy five euros 9 588,75 €, taxes included.

Derivations of future updates and/or applications as well as corrections and future consults are not included in it. The formation needed for this project is also excluded from the costs, because the only cost that is contemplated is related to the design and development of it.

Method of payment

The payment will be done as it is stated here:

The first payment from the part of the contracting company to the firm of the contract of this project will be of an amount of 1 000 € as an advance and in conformity with the same.

A second payment of 2 000 € will be done once the problematic has been analysed and considering the viability study of the project. In this part the firm of this contract should start with the analysis of the project. If for any reason that has nothing to do with the company, there is no possibility for the company to develop this analysis, the company must give back the 1 000 € that were received as an advance and in conformity with the client.

Finally, the last payment of 6 588,75 € will be done when the reception of the work happens.

Legal and contractual conditions

Handover certificate

The provisional reception of the services will take place once these services are provided. Both parts must sign the Provisional Handover Certificate when this happens.

The claim period will start after the creation of this certificate and will have a duration of two weeks.

After this period, if the contractor has not notified the defects that changed the terms, the Definite Handover Certificate will be subscribed.

The warranty period, whose conditions are established in the section "Warranty terms and conditions" of the present document, will start when the mentioned certificate is created.

Acquisition agreement and maintenance contract

The acquisition agreement will be sealed from both parts once the data consistency of the analysis associated with the project *"Design and implementation of efficient diminished reality mechanisms"* is verified. Maintenance/Update conditions offered in this contract are developed in the section *"Maintenance and update conditions"* of this document.

A maintenance and update contract must be done from both parts if further support or an extension of the specified condition is needed.

Client responsibilities

The client is responsible for ensuring compliance with current legislation concerning the contracted services, including permissions and licenses of utilization, as well as the regulations of intellectual property and confidentiality of the development done in this project.

The client doesn't obtain any ownership of the analysis done in this project, but it has the rights to make use of it as well as for the storage in the digital format that is presented.

The client has no authorization to distribute copies or adaptations to third parties of the data given by the firm without a consent of it.

Project developer responsibilities

The developer of the project compromises to accomplish the deadlines established during the execution of the project.

The developer will also be responsible to offer a consistent analysis with a detailed technological background based on results that are coherent and close to reality.

Contract expiry

The contract will expire when the project is concluded or finished, or because one of the parts that is involved has decided it. The following list includes the reasons that will be considered for the resolution:

- Breach of the clause(s) included in the *"Bid specifications"*.
- Mutual agreement between both parts.
- One of the part has declared bankruptcy or suspension of payments.

Maintenance and update conditions

A maintenance and update contract can be done with a specific duration that is established in it.

Two steps of action are defined to encompass all necessary operations to assure the functioning and prolong the duration of it.

Update for improvements

The maintenance and update plan based on improvements of the elements involved includes any update on the analysis resulting from the updating or improvement over some of the technologies involved in the analysis.

A new study will be done including the changes that encompass the improvement and a comparative between the previous and actual analysis.

Extended study

The maintenance and update plan offers the possibility to make a certain amount of changes (the amount that is agreed in the contract) over the proposed initial solution, resulting in a new analysis that will be delivered to the contractor together with the previous one in order to compare both solutions.

Warranty terms and conditions

The designer is committed to perform a preliminary study that will check the degree of depth that can be obtained with the approach the customer has proposed and the viability of development of it.

In the case where the results offered do not comply with the feasibility and depth shown in the previous study , the designer guarantees a full refund the amount paid.

Legal aspects

Legal aspects related to the recruitment and development of the project are explained in detail in this section.

Force majeure

The contractor is not responsible for the breach of responsibilities if the execution of the tasks is delayed or not done because of force majeure.

They are considered force majeure all those events or circumstances, out of control of the contractor or the buyer and any other circumstances that were unforeseeable, or being foreseeable were inevitable, according to the case law and legal doctrine sitting on this concept in the Civil Code.

The causes of force majeure mentioned in the previous paragraph will be taken into consideration only when they affect directly to the supply.

Tribunals and arbitrations

The buyer as well as the contractor compromise to fulfil the conditions that are established in this "Bid specifications" as well as in the complementary documentation of the contract, resolving through agreements and negotiations any differences that may arise between them regarding the application, development, implementation, execution or interpretation of it.

If there is any discrepancy or controversy between both of them that can't be solved, the contractor compromises to submit such disputes to arbitration, formalized according to the regulatory rules contained in the current Private Arbitration Law.

The arbitration will take part in Bilbao and the writing must include the commitment within 30 days as a term in which the arbitrators have to pronounce the corresponding decision.

The arbitration procedure must be done in accordance with the Spanish legislation and will be resolved in law by three arbitrators, one chosen by each party and the third, as designated.

If the parties fail to reach an agreement to designate the third arbitrator, it will be selected by sortition from a list that has been provided by the Bar Association of Bilbao.

The referees will proceed to liquidate the expenses of the arbitration proceedings, including their fees, and to establish which party will be charged the costs of prosecution.

The parties declare from now that they accept the decisions of the Arbitral Tribunal.

Part VII

APPENDIX II: Drawings and diagrams

Speed tests

Simple scenario - Rotation

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	372,258	FrameToFrame	22,8847	FrameToFrame	2,18895	FrameToFrame	2,24968	FrameToFrame	1,47074
FrameToFrame	386,076	FrameToFrame	22,7336	FrameToFrame	1,92185	FrameToFrame	0,0548956	FrameToFrame	0,04588
FrameToFrame	386,472	FrameToFrame	22,9839	FrameToFrame	1,90622	FrameToFrame	0,0479589	FrameToFrame	0,0479147
FrameToFrame	364,659	FrameToFrame	22,5937	FrameToFrame	1,92177	FrameToFrame	0,0459598	FrameToFrame	0,0459236
FrameToFrame	382,067	FrameToFrame	22,1246	FrameToFrame	1,82827	FrameToFrame	0,047966	FrameToFrame	0,0459156
FrameToFrame	371,646	FrameToFrame	22,7498	FrameToFrame	1,90624	FrameToFrame	0,0459813	FrameToFrame	0,0479159
FrameToFrame	365,579	FrameToFrame	22,0639	FrameToFrame	1,8751	FrameToFrame	0,0469598	FrameToFrame	0,0469066
FrameToFrame	357,596	FrameToFrame	22,2813	FrameToFrame	1,89062	FrameToFrame	0,0459611	FrameToFrame	0,0469329
FrameToFrame	372,868	FrameToFrame	21,4052	FrameToFrame	1,84378	FrameToFrame	0,0479798	FrameToFrame	0,0469477
FrameToFrame	372,33	FrameToFrame	22,3123	FrameToFrame	1,84362	FrameToFrame	0,046235	FrameToFrame	0,0619311
FrameToFrame	366,006	FrameToFrame	22,1407	FrameToFrame	1,79706	FrameToFrame	0,0479717	FrameToFrame	0,0459396
FrameToFrame	372,485	FrameToFrame	22,4058	FrameToFrame	1,89042	FrameToFrame	0,0449835	FrameToFrame	0,0479377
FrameToFrame	373,378	FrameToFrame	21,8903	FrameToFrame	1,92376	FrameToFrame	0,0469768	FrameToFrame	0,045964
FrameToFrame	380,02	FrameToFrame	21,8158	FrameToFrame	1,84621	FrameToFrame	0,0479842	FrameToFrame	0,0480022
FrameToFrame	358,567	FrameToFrame	21,2908	FrameToFrame	1,84571	FrameToFrame	0,0459977	FrameToFrame	0,0622486
FrameToFrame	379,192	FrameToFrame	22,1103	FrameToFrame	1,84397	FrameToFrame	0,0479403	FrameToFrame	0,0459464
FrameToFrame	358,36	FrameToFrame	21,4372	FrameToFrame	1,8629	FrameToFrame	0,0459425	FrameToFrame	0,045948
FrameToFrame	372,534	FrameToFrame	21,2803	FrameToFrame	1,814	FrameToFrame	0,0469797	FrameToFrame	0,0469223
FrameToFrame	380,519	FrameToFrame	22,0164	FrameToFrame	1,90623	FrameToFrame	0,0459903	FrameToFrame	0,04693
FrameToFrame	350,986	FrameToFrame	22,1565	FrameToFrame	1,81252	FrameToFrame	0,0469631	FrameToFrame	0,0469121
FrameToFrame	359,049	FrameToFrame	21,9999	FrameToFrame	1,82796	FrameToFrame	0,0469855	FrameToFrame	0,0479127
FrameToFrame	365,411	FrameToFrame	22,0461	FrameToFrame	1,93761	FrameToFrame	0,0479727	FrameToFrame	0,0459072
FrameToFrame	380,423	FrameToFrame	22,0162	FrameToFrame	1,85938	FrameToFrame	0,0469775	FrameToFrame	0,0469438
FrameToFrame	366,065	FrameToFrame	21,9064	FrameToFrame	1,90613	FrameToFrame	0,0469749	FrameToFrame	0,0469701
FrameToFrame	359,268	FrameToFrame	21,8263	FrameToFrame	1,82809	FrameToFrame	0,0449758	FrameToFrame	0,0467764
FrameToFrame	365,956	FrameToFrame	21,5481	FrameToFrame	1,8124	FrameToFrame	0,0469807	FrameToFrame	0,0459784
FrameToFrame	366,128	FrameToFrame	21,8738	FrameToFrame	1,85941	FrameToFrame	0,0479766	FrameToFrame	0,0470089
FrameToFrame	365,675	FrameToFrame	21,8289	FrameToFrame	1,82815	FrameToFrame	0,0459557	FrameToFrame	0,0474249
FrameToFrame	367,784	FrameToFrame	22,1566	FrameToFrame	1,7969	FrameToFrame	0,0479605	FrameToFrame	0,0469611
FrameToFrame	381,741	FrameToFrame	21,3282	FrameToFrame	1,82829	FrameToFrame	0,0459717	FrameToFrame	0,0469932
FrameToFrame	374,391	FrameToFrame	21,2327	FrameToFrame	1,76553	FrameToFrame	0,0479615	FrameToFrame	0,0459861
FrameToFrame	381,363	FrameToFrame	21,7505	FrameToFrame	1,81252	FrameToFrame	0,0449713	FrameToFrame	0,0469839
FrameToFrame	359,833	FrameToFrame	21,8118	FrameToFrame	1,7814	FrameToFrame	0,0469663	FrameToFrame	0,0469679
FrameToFrame	367,673	FrameToFrame	21,9226	FrameToFrame	1,81241	FrameToFrame	1,87593	FrameToFrame	0,0459823
FrameToFrame	382,069	FrameToFrame	22,0935	FrameToFrame	1,81252	FrameToFrame	0,0469659	FrameToFrame	0,0469746
FrameToFrame	367,158	FrameToFrame	22,2819	FrameToFrame	1,87503	FrameToFrame	0,0470064	FrameToFrame	0,0469784
FrameToFrame	368,346	FrameToFrame	22,0001	FrameToFrame	1,78126	FrameToFrame	0,0469409	FrameToFrame	0,04699
FrameToFrame	374,677	FrameToFrame	22,046	FrameToFrame	1,79705	FrameToFrame	0,0469833	FrameToFrame	0,0479682
FrameToFrame	355,485	FrameToFrame	21,3135	FrameToFrame	1,7655	FrameToFrame	0,0449771	FrameToFrame	0,0459858
FrameToFrame	355,052	FrameToFrame	21,0925	FrameToFrame	1,7813	FrameToFrame	0,0470047	FrameToFrame	0,0459579
FrameToFrame	362,406	FrameToFrame	20,9371	FrameToFrame	1,73438	FrameToFrame	0,0479737	FrameToFrame	0,0479592

FrameToFrame	369,612	FrameToFrame	21,2653	FrameToFrame	1,81252	FrameToFrame	0,0459807	FrameToFrame	0,0469663
FrameToFrame	341,237	FrameToFrame	21,1268	FrameToFrame	1,75004	FrameToFrame	0,0469794	FrameToFrame	0,0459787
FrameToFrame	348,517	FrameToFrame	21,3589	FrameToFrame	1,79689	FrameToFrame	0,0479778	FrameToFrame	0,046982
FrameToFrame	362,237	FrameToFrame	21,0774	FrameToFrame	1,79691	FrameToFrame	0,0460538	FrameToFrame	0,0459778
FrameToFrame	355,551	FrameToFrame	20,5316	FrameToFrame	1,71886	FrameToFrame	0,0459598	FrameToFrame	0,0479724
FrameToFrame	354,922	FrameToFrame	21,3745	FrameToFrame	1,76558	FrameToFrame	0,046981	FrameToFrame	0,0459784
FrameToFrame	355,003	FrameToFrame	20,5943	FrameToFrame	1,76565	FrameToFrame	0,0479791	FrameToFrame	0,0479788
FrameToFrame	370,049	FrameToFrame	21,4363	FrameToFrame	1,75	FrameToFrame	0,0469858	FrameToFrame	0,0459903
FrameToFrame	355,877	FrameToFrame	21,3903	FrameToFrame	1,75002	FrameToFrame	0,0459707	FrameToFrame	0,0479576
FrameToFrame	348,861	FrameToFrame	21,1579	FrameToFrame	1,7188	FrameToFrame	0,0459768	FrameToFrame	0,0459682
FrameToFrame	348,268	FrameToFrame	21,3427	FrameToFrame	1,73439	FrameToFrame	1,81351	FrameToFrame	0,0459768
FrameToFrame	348,627	FrameToFrame	21,2033	FrameToFrame	1,68752	FrameToFrame	0,0469509	FrameToFrame	0,0479769
FrameToFrame	341,517	FrameToFrame	21,2501	FrameToFrame	1,73438	FrameToFrame	0,0469922	FrameToFrame	0,0469826
FrameToFrame	341,58	FrameToFrame	20,9214	FrameToFrame	1,68773	FrameToFrame	0,0459361	FrameToFrame	0,0459791
FrameToFrame	349,299	FrameToFrame	20,938	FrameToFrame	1,68738	FrameToFrame	0,046957	FrameToFrame	0,0479615
FrameToFrame	342,72	FrameToFrame	20,7341	FrameToFrame	1,75	FrameToFrame	0,0469727	FrameToFrame	0,0459935
FrameToFrame	362,661	FrameToFrame	21,5785	FrameToFrame	1,70329	FrameToFrame	0,046973	FrameToFrame	0,0459653
FrameToFrame	335,187	FrameToFrame	20,4531	FrameToFrame	1,64065	FrameToFrame	0,0459951	FrameToFrame	0,0479628
FrameToFrame	349,971	FrameToFrame	20,9213	FrameToFrame	1,73426	FrameToFrame	0,0479541	FrameToFrame	0,0469611
FrameToFrame	328,235	FrameToFrame	20,3276	FrameToFrame	1,71879	FrameToFrame	0,0459666	FrameToFrame	0,0459852
FrameToFrame	348,989	FrameToFrame	20,7356	FrameToFrame	1,67191	FrameToFrame	0,0479666	FrameToFrame	0,0479701
FrameToFrame	349,891	FrameToFrame	20,5769	FrameToFrame	1,67188	FrameToFrame	1,7332	FrameToFrame	0,0459842
FrameToFrame	356,675	FrameToFrame	20,797	FrameToFrame	1,68754	FrameToFrame	0,0483082	FrameToFrame	0,0459784
FrameToFrame	336,688	FrameToFrame	19,8747	FrameToFrame	1,71876	FrameToFrame	0,0459826	FrameToFrame	0,0469849
FrameToFrame	328,206	FrameToFrame	20,3435	FrameToFrame	1,71891	FrameToFrame	0,0470012	FrameToFrame	0,0469733
FrameToFrame	335,999	FrameToFrame	20,3911	FrameToFrame	1,73427	FrameToFrame	0,0459621	FrameToFrame	0,0469781
FrameToFrame	336,689	FrameToFrame	20,2962	FrameToFrame	1,64064	FrameToFrame	0,0482197	FrameToFrame	0,0479826
FrameToFrame	344,222	FrameToFrame	20,1735	FrameToFrame	1,65642	FrameToFrame	0,044971	FrameToFrame	0,0459913
FrameToFrame	343,05	FrameToFrame	20,6859	FrameToFrame	1,64065	FrameToFrame	0,0470022	FrameToFrame	0,0479596
FrameToFrame	336,811	FrameToFrame	20,3438	FrameToFrame	1,73425	FrameToFrame	0,0479846	FrameToFrame	0,0459803
FrameToFrame	349,83	FrameToFrame	20,7974	FrameToFrame	1,73438	FrameToFrame	0,0469791	FrameToFrame	0,0459569
FrameToFrame	328,314	FrameToFrame	20,3595	FrameToFrame	1,67204	FrameToFrame	0,045972	FrameToFrame	0,0479887
FrameToFrame	342,767	FrameToFrame	20,7958	FrameToFrame	1,71864	FrameToFrame	0,0469768	FrameToFrame	0,0469675
FrameToFrame	355,33	FrameToFrame	21,1265	FrameToFrame	1,76564	FrameToFrame	0,0469759	FrameToFrame	0,0610048
FrameToFrame	354,502	FrameToFrame	21,5141	FrameToFrame	1,71879	FrameToFrame	0,0469659	FrameToFrame	0,04794
FrameToFrame	356,393	FrameToFrame	20,8443	FrameToFrame	1,79706	FrameToFrame	0,0459922	FrameToFrame	0,0459803
FrameToFrame	354,877	FrameToFrame	21,4997	FrameToFrame	1,79677	FrameToFrame	0,0469669	FrameToFrame	0,0479506
FrameToFrame	354,346	FrameToFrame	21,1731	FrameToFrame	1,7344	FrameToFrame	0,0479772	FrameToFrame	0,0469922
FrameToFrame	368,128	FrameToFrame	21,9824	FrameToFrame	1,76577	FrameToFrame	0,0469926	FrameToFrame	0,0469768
FrameToFrame	362,065	FrameToFrame	21,6264	FrameToFrame	1,82806	FrameToFrame	0,0469784	FrameToFrame	0,0449717
FrameToFrame	362,081	FrameToFrame	21,5933	FrameToFrame	1,78136	FrameToFrame	0,0449813	FrameToFrame	0,0479727
FrameToFrame	370,191	FrameToFrame	21,4207	FrameToFrame	1,75002	FrameToFrame	0,0469416	FrameToFrame	0,0459945

FrameToFrame	347,689	FrameToFrame	21,2982	FrameToFrame	1,76553	FrameToFrame	0,047983	FrameToFrame	0,0479801
FrameToFrame	342,689	FrameToFrame	20,5299	FrameToFrame	1,76565	FrameToFrame	0,0459803	FrameToFrame	0,046981
FrameToFrame	356,16	FrameToFrame	21,4388	FrameToFrame	1,76577	FrameToFrame	0,0479775	FrameToFrame	0,0469801
FrameToFrame	356,517	FrameToFrame	20,4993	FrameToFrame	1,79682	FrameToFrame	0,0459938	FrameToFrame	0,0459781
FrameToFrame	348,679	FrameToFrame	21,0304	FrameToFrame	1,73438	FrameToFrame	0,0459589	FrameToFrame	0,0459935
FrameToFrame	356,154	FrameToFrame	21,0791	FrameToFrame	1,70316	FrameToFrame	0,0479746	FrameToFrame	0,047964
FrameToFrame	362,971	FrameToFrame	20,9367	FrameToFrame	1,73437	FrameToFrame	0,0469621	FrameToFrame	0,0459836
FrameToFrame	328,063	FrameToFrame	20,7974	FrameToFrame	1,75005	FrameToFrame	0,0459739	FrameToFrame	0,0470147
FrameToFrame	348,473	FrameToFrame	21,0311	FrameToFrame	1,70313	FrameToFrame	0,047973	FrameToFrame	0,0479984
FrameToFrame	349,797	FrameToFrame	20,9226	FrameToFrame	1,70314	FrameToFrame	0,0459877	FrameToFrame	0,0459547
FrameToFrame	356,332	FrameToFrame	20,905	FrameToFrame	1,75005	FrameToFrame	0,0459608	FrameToFrame	0,0459582
FrameToFrame	343,033	FrameToFrame	20,5162	FrameToFrame	1,71878	FrameToFrame	0,0479733	FrameToFrame	0,0469807
FrameToFrame	349,145	FrameToFrame	20,8905	FrameToFrame	1,71874	FrameToFrame	0,0469749	FrameToFrame	0,0469829
FrameToFrame	355,44	FrameToFrame	21,093	FrameToFrame	1,81253	FrameToFrame	0,046972	FrameToFrame	0,047983
FrameToFrame	341,955	FrameToFrame	20,8291	FrameToFrame	1,78142	FrameToFrame	1,78041	FrameToFrame	0,0459685
FrameToFrame	347,752	FrameToFrame	21,28	FrameToFrame	1,76549	FrameToFrame	0,0479689	FrameToFrame	0,0479932
FrameToFrame	335,049	FrameToFrame	20,7981	FrameToFrame	1,67192	FrameToFrame	0,0459547	FrameToFrame	0,0459425
FrameToFrame	355,877	FrameToFrame	21,2181	FrameToFrame	1,76562	FrameToFrame	0,0459787	FrameToFrame	0,0469845
FrameToFrame	335,689	FrameToFrame	20,8911	FrameToFrame	1,71879	FrameToFrame	0,0470121	FrameToFrame	0,0469788
FrameToFrame	349,393	FrameToFrame	20,6408	FrameToFrame	1,67187	FrameToFrame	0,0469473	FrameToFrame	0,046981
FrameToFrame	343,017	FrameToFrame	20,5773	FrameToFrame	1,73441	FrameToFrame	0,0479621	FrameToFrame	0,0459864
FrameToFrame	348,83	FrameToFrame	20,7493	FrameToFrame	1,75003	FrameToFrame	0,0469717	FrameToFrame	0,0479615
FrameToFrame	342,768	FrameToFrame	20,6884	FrameToFrame	1,70328	FrameToFrame	0,0459585	FrameToFrame	0,0459637
FrameToFrame	341,751	FrameToFrame	21,1093	FrameToFrame	1,73427	FrameToFrame	0,0459714	FrameToFrame	0,0469579
FrameToFrame	336,205	FrameToFrame	20,421	FrameToFrame	1,65643	FrameToFrame	1,79806	FrameToFrame	0,045982
FrameToFrame	343,033	FrameToFrame	20,9683	FrameToFrame	1,7655	FrameToFrame	0,0459181	FrameToFrame	0,0469826
FrameToFrame	321,923	FrameToFrame	20,4858	FrameToFrame	1,71879	FrameToFrame	0,047965	FrameToFrame	0,0479743
FrameToFrame	342,392	FrameToFrame	20,9049	FrameToFrame	1,70317	FrameToFrame	0,0469669	FrameToFrame	0,062008
FrameToFrame	350,067	FrameToFrame	20,7672	FrameToFrame	1,78126	FrameToFrame	0,0459727	FrameToFrame	0,0459723
FrameToFrame	336,234	FrameToFrame	20,6561	FrameToFrame	1,71898	FrameToFrame	0,0469759	FrameToFrame	0,0469663
FrameToFrame	350,705	FrameToFrame	20,8748	FrameToFrame	1,6873	FrameToFrame	0,0459637	FrameToFrame	0,0479676
FrameToFrame	322,675	FrameToFrame	20,2649	FrameToFrame	1,70316	FrameToFrame	0,0479618	FrameToFrame	0,0469541
FrameToFrame	329,702	FrameToFrame	20,3441	FrameToFrame	1,70328	FrameToFrame	0,0459614	FrameToFrame	0,0469839
FrameToFrame	337,22	FrameToFrame	20,6081	FrameToFrame	1,64067	FrameToFrame	0,0473739	FrameToFrame	0,0449781
FrameToFrame	337,613	FrameToFrame	20,3601	FrameToFrame	1,76549	FrameToFrame	1,70339	FrameToFrame	0,0479817
FrameToFrame	343,593	FrameToFrame	20,9852	FrameToFrame	1,65628	FrameToFrame	0,0459592	FrameToFrame	0,0459884
FrameToFrame	337,548	FrameToFrame	20,7175	FrameToFrame	1,70314	FrameToFrame	0,0469717	FrameToFrame	0,0469688
FrameToFrame	322,938	FrameToFrame	20,4527	FrameToFrame	1,75003	FrameToFrame	0,0469756	FrameToFrame	0,0469733
FrameToFrame	336,704	FrameToFrame	20,953	FrameToFrame	1,70324	FrameToFrame	0,0479721	FrameToFrame	0,0469804
FrameToFrame	336,988	FrameToFrame	20,797	FrameToFrame	1,67178	FrameToFrame	0,0459646	FrameToFrame	0,0459794
FrameToFrame	335,577	FrameToFrame	20,9531	FrameToFrame	1,65626	FrameToFrame	1,71802	FrameToFrame	0,048509
FrameToFrame	343,284	FrameToFrame	20,8134	FrameToFrame	1,71879	FrameToFrame	0,0469765	FrameToFrame	0,0459643

FrameToFrame	343,032	FrameToFrame	20,9687	FrameToFrame	1,60939	FrameToFrame	0,047982	FrameToFrame	0,0459848
FrameToFrame	336,532	FrameToFrame	21,2188	FrameToFrame	1,68752	FrameToFrame	0,0459784	FrameToFrame	0,0469775
FrameToFrame	342,533	FrameToFrame	21,1704	FrameToFrame	1,70314	FrameToFrame	0,0469752	FrameToFrame	0,0480122
FrameToFrame	345,252	FrameToFrame	20,6406	FrameToFrame	1,6563	FrameToFrame	0,0459842	FrameToFrame	0,0459354
FrameToFrame	335,595	FrameToFrame	21,0002	FrameToFrame	1,75012	FrameToFrame	0,0469817	FrameToFrame	0,0479785
FrameToFrame	343,77	FrameToFrame	20,8295	FrameToFrame	1,65616	FrameToFrame	0,0479596	FrameToFrame	0,0459781
FrameToFrame	349,359	FrameToFrame	21,6234	FrameToFrame	1,73438	FrameToFrame	0,045964	FrameToFrame	0,0469784
FrameToFrame	342,189	FrameToFrame	21,298	FrameToFrame	1,70327	FrameToFrame	0,0479746	FrameToFrame	0,0469817
FrameToFrame	349,392	FrameToFrame	21,6097	FrameToFrame	1,70303	FrameToFrame	0,0459771	FrameToFrame	0,0469708
FrameToFrame	342,752	FrameToFrame	21,4514	FrameToFrame	1,65626	FrameToFrame	0,0469826	FrameToFrame	0,0460374
FrameToFrame	342,348	FrameToFrame	21,3448	FrameToFrame	1,76567	FrameToFrame	0,0459775	FrameToFrame	0,0469188
FrameToFrame	336,358	FrameToFrame	21,0461	FrameToFrame	1,73436	FrameToFrame	0,0479801	FrameToFrame	0,0479903
FrameToFrame	336,017	FrameToFrame	21,2348	FrameToFrame	1,84381	FrameToFrame	1,75028	FrameToFrame	0,0469791
FrameToFrame	342,705	FrameToFrame	21,2022	FrameToFrame	1,71878	FrameToFrame	0,0450169	FrameToFrame	0,0610269
FrameToFrame	344,314	FrameToFrame	21,2807	FrameToFrame	1,65634	FrameToFrame	0,0479217	FrameToFrame	0,046938
FrameToFrame	343,611	FrameToFrame	21,0015	FrameToFrame	1,70317	FrameToFrame	0,0469534	FrameToFrame	0,0479567
FrameToFrame	350,161	FrameToFrame	21,4517	FrameToFrame	1,64063	FrameToFrame	0,0469788	FrameToFrame	0,045971
FrameToFrame	330,608	FrameToFrame	20,7346	FrameToFrame	1,68738	FrameToFrame	0,045981	FrameToFrame	0,0469919
FrameToFrame	351,408	FrameToFrame	21,3292	FrameToFrame	1,76566	FrameToFrame	0,0469643	FrameToFrame	0,045971
FrameToFrame	337,376	FrameToFrame	20,8427	FrameToFrame	1,67189	FrameToFrame	0,0469839	FrameToFrame	0,0479749
FrameToFrame	350,221	FrameToFrame	21,3588	FrameToFrame	1,75004	FrameToFrame	0,0469724	FrameToFrame	0,0459829
FrameToFrame	337,222	FrameToFrame	21,0005	FrameToFrame	1,73441	FrameToFrame	0,0459733	FrameToFrame	0,0479493
FrameToFrame	344,063	FrameToFrame	21,1089	FrameToFrame	1,64061	FrameToFrame	0,047982	FrameToFrame	0,0459807
FrameToFrame	344,189	FrameToFrame	20,9999	FrameToFrame	1,65629	FrameToFrame	0,046981	FrameToFrame	0,0459797
FrameToFrame	356,721	FrameToFrame	21,5936	FrameToFrame	1,70313	FrameToFrame	0,0459646	FrameToFrame	0,0479794
FrameToFrame	349,833	FrameToFrame	21,5155	FrameToFrame	1,7345	FrameToFrame	0,0469756	FrameToFrame	0,0459864
FrameToFrame	355,64	FrameToFrame	21,8908	FrameToFrame	1,76555	FrameToFrame	0,0459877	FrameToFrame	0,0469592
TotalTime	53526,351	TotalTime	3226,4485	TotalTime	266,96878	TotalTime	23,1238344	TotalTime	8,6162813

Simple scenario - Translation

Original		**Original_Window**		**Original_Window_Pyramid**		**Final (Camshift)**		**Final (Optical Flow)**	
FrameToFrame	273,703	FrameToFrame	15,6661	FrameToFrame	1,42882	FrameToFrame	1,4756	FrameToFrame	1,39165
FrameToFrame	287,125	FrameToFrame	16,563	FrameToFrame	1,39418	FrameToFrame	0,061974	FrameToFrame	0,0467719
FrameToFrame	286,812	FrameToFrame	15,5616	FrameToFrame	1,42005	FrameToFrame	0,0460057	FrameToFrame	0,047966
FrameToFrame	285,985	FrameToFrame	16,4523	FrameToFrame	1,3597	FrameToFrame	0,0479612	FrameToFrame	0,0459367
FrameToFrame	272,17	FrameToFrame	15,7986	FrameToFrame	1,49003	FrameToFrame	0,046014	FrameToFrame	0,0459486
FrameToFrame	286,424	FrameToFrame	15,7648	FrameToFrame	1,37596	FrameToFrame	0,0469582	FrameToFrame	0,0479365
FrameToFrame	229,248	FrameToFrame	15,718	FrameToFrame	1,32811	FrameToFrame	0,0479666	FrameToFrame	0,0459277
FrameToFrame	279,438	FrameToFrame	15,954	FrameToFrame	1,37706	FrameToFrame	0,0459842	FrameToFrame	0,047949
FrameToFrame	271,767	FrameToFrame	15,9831	FrameToFrame	1,37658	FrameToFrame	0,0476327	FrameToFrame	0,0459406
FrameToFrame	287,014	FrameToFrame	16,0791	FrameToFrame	1,31812	FrameToFrame	0,0459592	FrameToFrame	0,0479419
FrameToFrame	287,033	FrameToFrame	15,4367	FrameToFrame	1,32812	FrameToFrame	0,0460637	FrameToFrame	0,0459412
FrameToFrame	280,245	FrameToFrame	15,7666	FrameToFrame	1,34862	FrameToFrame	0,0479573	FrameToFrame	0,045946
FrameToFrame	285,767	FrameToFrame	16,0765	FrameToFrame	1,39248	FrameToFrame	0,0459832	FrameToFrame	0,0469053
FrameToFrame	278,982	FrameToFrame	15,7815	FrameToFrame	1,28411	FrameToFrame	0,0469858	FrameToFrame	0,0479458
FrameToFrame	258,201	FrameToFrame	14,5785	FrameToFrame	1,24285	FrameToFrame	0,0469643	FrameToFrame	0,0459467
FrameToFrame	264,857	FrameToFrame	15,3899	FrameToFrame	1,26559	FrameToFrame	0,0469554	FrameToFrame	0,0479345
FrameToFrame	285,656	FrameToFrame	16,0624	FrameToFrame	1,41066	FrameToFrame	0,0459749	FrameToFrame	0,0459354
FrameToFrame	263,92	FrameToFrame	14,9216	FrameToFrame	1,33036	FrameToFrame	0,0479891	FrameToFrame	0,0459505
FrameToFrame	264,888	FrameToFrame	15,0371	FrameToFrame	1,37907	FrameToFrame	0,0459662	FrameToFrame	0,0479195
FrameToFrame	271,405	FrameToFrame	15,2453	FrameToFrame	1,3598	FrameToFrame	0,04694	FrameToFrame	0,0459781
FrameToFrame	279,624	FrameToFrame	15,3126	FrameToFrame	1,25687	FrameToFrame	0,047982	FrameToFrame	0,0469018
FrameToFrame	279,686	FrameToFrame	15,2331	FrameToFrame	1,39868	FrameToFrame	0,0459752	FrameToFrame	0,0469451
FrameToFrame	279,923	FrameToFrame	15,2351	FrameToFrame	1,28149	FrameToFrame	0,0472543	FrameToFrame	0,0469438
FrameToFrame	280,092	FrameToFrame	15,7178	FrameToFrame	1,31352	FrameToFrame	0,0459694	FrameToFrame	0,0459544
FrameToFrame	273,387	FrameToFrame	15,0797	FrameToFrame	1,26778	FrameToFrame	0,048008	FrameToFrame	0,0478569
FrameToFrame	266,06	FrameToFrame	14,67	FrameToFrame	1,2343	FrameToFrame	0,0469538	FrameToFrame	0,045988
FrameToFrame	265,264	FrameToFrame	15,0633	FrameToFrame	1,29671	FrameToFrame	0,0449883	FrameToFrame	0,0479624
FrameToFrame	280,639	FrameToFrame	14,937	FrameToFrame	1,26576	FrameToFrame	0,0479602	FrameToFrame	0,0469653
FrameToFrame	265,748	FrameToFrame	15,0945	FrameToFrame	1,29698	FrameToFrame	0,045973	FrameToFrame	0,045981
FrameToFrame	273,264	FrameToFrame	15,0929	FrameToFrame	1,25261	FrameToFrame	0,0479977	FrameToFrame	0,0479756
FrameToFrame	273,313	FrameToFrame	15,2661	FrameToFrame	1,25001	FrameToFrame	0,045946	FrameToFrame	0,0459547
FrameToFrame	265,761	FrameToFrame	14,8904	FrameToFrame	1,31249	FrameToFrame	0,0479762	FrameToFrame	0,0471276
FrameToFrame	266,311	FrameToFrame	15,4213	FrameToFrame	1,34388	FrameToFrame	0,0459816	FrameToFrame	0,0459842
FrameToFrame	259,607	FrameToFrame	14,3912	FrameToFrame	1,29673	FrameToFrame	0,0469878	FrameToFrame	0,0479612
FrameToFrame	266,656	FrameToFrame	15,3586	FrameToFrame	1,26573	FrameToFrame	0,0459701	FrameToFrame	0,0459601
FrameToFrame	259,527	FrameToFrame	14,3288	FrameToFrame	1,2968	FrameToFrame	0,0470666	FrameToFrame	0,0469576
FrameToFrame	252,344	FrameToFrame	14,7193	FrameToFrame	1,34376	FrameToFrame	0,0480134	FrameToFrame	0,0469765
FrameToFrame	252,713	FrameToFrame	15,0931	FrameToFrame	1,29692	FrameToFrame	0,0459643	FrameToFrame	0,0459929
FrameToFrame	267,686	FrameToFrame	14,3911	FrameToFrame	1,31253	FrameToFrame	0,046964	FrameToFrame	0,0479746
FrameToFrame	259,516	FrameToFrame	14,4062	FrameToFrame	1,28111	FrameToFrame	0,0459566	FrameToFrame	0,0469643
FrameToFrame	259,528	FrameToFrame	15,1092	FrameToFrame	1,29704	FrameToFrame	0,0479942	FrameToFrame	0,0469637

FrameToFrame	252,514	FrameToFrame	14,4835	FrameToFrame	1,24986	FrameToFrame	0,0459877	FrameToFrame	0,0449774
FrameToFrame	260	FrameToFrame	14,9845	FrameToFrame	1,25001	FrameToFrame	0,0479666	FrameToFrame	0,0479814
FrameToFrame	266,485	FrameToFrame	15,328	FrameToFrame	1,34376	FrameToFrame	0,045956	FrameToFrame	0,0459861
FrameToFrame	259,594	FrameToFrame	15,2035	FrameToFrame	1,34381	FrameToFrame	0,045965	FrameToFrame	0,0479705
FrameToFrame	267,186	FrameToFrame	15,359	FrameToFrame	1,28121	FrameToFrame	0,0479926	FrameToFrame	0,0469656
FrameToFrame	259,029	FrameToFrame	15,2976	FrameToFrame	1,26556	FrameToFrame	0,0469817	FrameToFrame	0,0469605
FrameToFrame	258,591	FrameToFrame	15,139	FrameToFrame	1,31249	FrameToFrame	0,0459794	FrameToFrame	0,0469765
FrameToFrame	259,688	FrameToFrame	15,0472	FrameToFrame	1,29688	FrameToFrame	1,29826	FrameToFrame	0,0449726
FrameToFrame	258,843	FrameToFrame	15,1262	FrameToFrame	1,375	FrameToFrame	0,0449758	FrameToFrame	0,0479804
FrameToFrame	273,934	FrameToFrame	15,3595	FrameToFrame	1,26563	FrameToFrame	0,047948	FrameToFrame	0,0459832
FrameToFrame	274,548	FrameToFrame	14,8591	FrameToFrame	1,46874	FrameToFrame	0,0469852	FrameToFrame	0,0479682
FrameToFrame	274,436	FrameToFrame	14,4051	FrameToFrame	1,32816	FrameToFrame	0,0469631	FrameToFrame	0,0459672
FrameToFrame	261,045	FrameToFrame	14,6558	FrameToFrame	1,2656	FrameToFrame	0,0459659	FrameToFrame	0,0479621
FrameToFrame	247,044	FrameToFrame	14,4068	FrameToFrame	1,26562	FrameToFrame	0,0469643	FrameToFrame	0,0459701
FrameToFrame	254,795	FrameToFrame	14,2811	FrameToFrame	1,25001	FrameToFrame	0,0469624	FrameToFrame	0,0460371
FrameToFrame	261,701	FrameToFrame	14,468	FrameToFrame	1,23437	FrameToFrame	0,0459762	FrameToFrame	0,0479079
FrameToFrame	262,188	FrameToFrame	14,4384	FrameToFrame	1,21867	FrameToFrame	0,0469797	FrameToFrame	0,0620956
FrameToFrame	269,531	FrameToFrame	14,796	FrameToFrame	1,20314	FrameToFrame	0,0479807	FrameToFrame	0,0458883
FrameToFrame	247,796	FrameToFrame	14,4389	FrameToFrame	1,25	FrameToFrame	0,04598	FrameToFrame	0,0479778
FrameToFrame	241,169	FrameToFrame	14,3272	FrameToFrame	1,2031	FrameToFrame	1,25068	FrameToFrame	0,0469538
FrameToFrame	255,816	FrameToFrame	14,1553	FrameToFrame	1,21878	FrameToFrame	0,0459553	FrameToFrame	0,0459781
FrameToFrame	249,419	FrameToFrame	14,0952	FrameToFrame	1,2031	FrameToFrame	0,0479939	FrameToFrame	0,0479778
FrameToFrame	242,012	FrameToFrame	13,8738	FrameToFrame	1,20312	FrameToFrame	0,0459653	FrameToFrame	0,0469531
FrameToFrame	242,323	FrameToFrame	13,8594	FrameToFrame	1,15634	FrameToFrame	0,046982	FrameToFrame	0,047025
FrameToFrame	242,514	FrameToFrame	13,8124	FrameToFrame	1,17179	FrameToFrame	0,0469871	FrameToFrame	0,044921
FrameToFrame	220,76	FrameToFrame	13,4379	FrameToFrame	1,17187	FrameToFrame	0,04598	FrameToFrame	0,0479775
FrameToFrame	248,422	FrameToFrame	14,1554	FrameToFrame	1,21889	FrameToFrame	0,0479467	FrameToFrame	0,0469894
FrameToFrame	249,246	FrameToFrame	14,094	FrameToFrame	1,20313	FrameToFrame	0,0469768	FrameToFrame	0,046974
FrameToFrame	241,889	FrameToFrame	13,8122	FrameToFrame	1,17172	FrameToFrame	0,0469198	FrameToFrame	0,0469781
FrameToFrame	256,311	FrameToFrame	14,2034	FrameToFrame	1,24999	FrameToFrame	0,0459646	FrameToFrame	0,0469772
FrameToFrame	255,451	FrameToFrame	14,2513	FrameToFrame	1,2656	FrameToFrame	0,0459589	FrameToFrame	0,0459887
FrameToFrame	255,467	FrameToFrame	14,5149	FrameToFrame	1,20312	FrameToFrame	0,0479753	FrameToFrame	0,0459617
FrameToFrame	248,75	FrameToFrame	14,3746	FrameToFrame	1,20313	FrameToFrame	0,0459624	FrameToFrame	0,0479663
FrameToFrame	248,812	FrameToFrame	14,0153	FrameToFrame	1,2031	FrameToFrame	0,0479923	FrameToFrame	0,0469833
FrameToFrame	254,793	FrameToFrame	14,6419	FrameToFrame	1,28138	FrameToFrame	0,0459653	FrameToFrame	0,0469691
FrameToFrame	255,735	FrameToFrame	13,9686	FrameToFrame	1,18736	FrameToFrame	0,0479647	FrameToFrame	0,0459887
FrameToFrame	248,31	FrameToFrame	14,6855	FrameToFrame	1,2188	FrameToFrame	0,0449585	FrameToFrame	0,0459906
FrameToFrame	254,859	FrameToFrame	14,3918	FrameToFrame	1,23442	FrameToFrame	0,0479814	FrameToFrame	0,0469621
FrameToFrame	254,781	FrameToFrame	14,4059	FrameToFrame	1,24991	FrameToFrame	0,0469784	FrameToFrame	0,0479778
FrameToFrame	255,498	FrameToFrame	14,5308	FrameToFrame	1,2187	FrameToFrame	0,0459909	FrameToFrame	0,0469817
FrameToFrame	261,857	FrameToFrame	14,8123	FrameToFrame	1,20314	FrameToFrame	0,0479573	FrameToFrame	0,045879
FrameToFrame	254,794	FrameToFrame	14,7655	FrameToFrame	1,28139	FrameToFrame	0,0459707	FrameToFrame	0,0469541

FrameToFrame	253,466	FrameToFrame	15,1249	FrameToFrame	1,3125	FrameToFrame	0,0479823	FrameToFrame	0,0479682
FrameToFrame	261,937	FrameToFrame	14,4384	FrameToFrame	1,2031	FrameToFrame	0,0449752	FrameToFrame	0,0449566
FrameToFrame	254,654	FrameToFrame	15,1402	FrameToFrame	1,26549	FrameToFrame	0,0479826	FrameToFrame	0,0469691
FrameToFrame	255,608	FrameToFrame	14,828	FrameToFrame	1,23435	FrameToFrame	0,0459807	FrameToFrame	0,0479669
FrameToFrame	247,921	FrameToFrame	14,7807	FrameToFrame	1,20314	FrameToFrame	0,0469759	FrameToFrame	0,0461648
FrameToFrame	255,219	FrameToFrame	14,8593	FrameToFrame	1,23437	FrameToFrame	0,0479855	FrameToFrame	0,047981
FrameToFrame	262,482	FrameToFrame	15	FrameToFrame	1,26565	FrameToFrame	0,045982	FrameToFrame	0,0459791
FrameToFrame	254,859	FrameToFrame	14,9221	FrameToFrame	1,24998	FrameToFrame	0,0469666	FrameToFrame	0,0469775
FrameToFrame	262,559	FrameToFrame	14,7816	FrameToFrame	1,26562	FrameToFrame	1,25021	FrameToFrame	0,0459762
FrameToFrame	262,874	FrameToFrame	14,8436	FrameToFrame	1,23452	FrameToFrame	0,0469724	FrameToFrame	0,0479798
FrameToFrame	254,732	FrameToFrame	14,8599	FrameToFrame	1,26547	FrameToFrame	0,0459678	FrameToFrame	0,0459759
FrameToFrame	262,53	FrameToFrame	15,0308	FrameToFrame	1,21873	FrameToFrame	0,0479567	FrameToFrame	0,0479935
FrameToFrame	262,124	FrameToFrame	14,5616	FrameToFrame	1,20313	FrameToFrame	0,0459861	FrameToFrame	0,0450037
FrameToFrame	262,39	FrameToFrame	14,8129	FrameToFrame	1,32809	FrameToFrame	0,0459633	FrameToFrame	0,0469348
FrameToFrame	254,656	FrameToFrame	14,8589	FrameToFrame	1,18755	FrameToFrame	0,0479682	FrameToFrame	0,0479932
FrameToFrame	261,83	FrameToFrame	15,3283	FrameToFrame	1,26557	FrameToFrame	0,0459784	FrameToFrame	0,045973
FrameToFrame	261,918	FrameToFrame	15,2193	FrameToFrame	1,24995	FrameToFrame	0,0479926	FrameToFrame	0,0469647
FrameToFrame	261,406	FrameToFrame	15,0472	FrameToFrame	1,28135	FrameToFrame	0,0459781	FrameToFrame	0,0479631
FrameToFrame	261,248	FrameToFrame	15,4359	FrameToFrame	1,31244	FrameToFrame	0,0469605	FrameToFrame	0,0469669
FrameToFrame	253,935	FrameToFrame	15,1891	FrameToFrame	1,24995	FrameToFrame	0,0459861	FrameToFrame	0,0460188
FrameToFrame	261,044	FrameToFrame	15,2797	FrameToFrame	1,28128	FrameToFrame	0,0470352	FrameToFrame	0,046948
FrameToFrame	267,813	FrameToFrame	15,6409	FrameToFrame	1,29685	FrameToFrame	0,0469624	FrameToFrame	0,0459666
FrameToFrame	260,671	FrameToFrame	15,5157	FrameToFrame	1,29687	FrameToFrame	0,0479692	FrameToFrame	0,0479727
FrameToFrame	268,642	FrameToFrame	15,7025	FrameToFrame	1,31249	FrameToFrame	0,0459749	FrameToFrame	0,0469672
FrameToFrame	259,996	FrameToFrame	15,7517	FrameToFrame	1,2969	FrameToFrame	0,0479826	FrameToFrame	0,0459669
FrameToFrame	268,421	FrameToFrame	15,4679	FrameToFrame	1,29701	FrameToFrame	0,044979	FrameToFrame	0,0469653
FrameToFrame	239,511	FrameToFrame	15,4213	FrameToFrame	1,26545	FrameToFrame	0,0479618	FrameToFrame	0,0459794
FrameToFrame	261,735	FrameToFrame	15,3903	FrameToFrame	1,25016	FrameToFrame	0,0469756	FrameToFrame	0,0479583
FrameToFrame	254,01	FrameToFrame	15,1421	FrameToFrame	1,24984	FrameToFrame	0,0459925	FrameToFrame	0,045965
FrameToFrame	275,297	FrameToFrame	15,6085	FrameToFrame	1,28129	FrameToFrame	0,0479801	FrameToFrame	0,0469621
FrameToFrame	254,105	FrameToFrame	15,0146	FrameToFrame	1,32808	FrameToFrame	0,0459646	FrameToFrame	0,047974
FrameToFrame	253,779	FrameToFrame	15,25	FrameToFrame	1,29688	FrameToFrame	0,0479612	FrameToFrame	0,0469791
FrameToFrame	253,856	FrameToFrame	15,282	FrameToFrame	1,28123	FrameToFrame	0,0449842	FrameToFrame	0,045981
FrameToFrame	275,186	FrameToFrame	15,7344	FrameToFrame	1,25002	FrameToFrame	0,0469563	FrameToFrame	0,0469778
FrameToFrame	267,781	FrameToFrame	15,7179	FrameToFrame	1,32821	FrameToFrame	0,0469704	FrameToFrame	0,0469797
FrameToFrame	268,468	FrameToFrame	15,4387	FrameToFrame	1,2656	FrameToFrame	0,0469566	FrameToFrame	0,0469752
FrameToFrame	261,355	FrameToFrame	15,0301	FrameToFrame	1,28114	FrameToFrame	0,0479762	FrameToFrame	0,045989
FrameToFrame	246,669	FrameToFrame	15,4381	FrameToFrame	1,29688	FrameToFrame	0,0459778	FrameToFrame	0,0479576
FrameToFrame	260,731	FrameToFrame	15,531	FrameToFrame	1,28133	FrameToFrame	0,0459816	FrameToFrame	0,0459752
FrameToFrame	267,092	FrameToFrame	15,9522	FrameToFrame	1,34368	FrameToFrame	0,0479278	FrameToFrame	0,0459624
FrameToFrame	274,142	FrameToFrame	16,047	FrameToFrame	1,26563	FrameToFrame	0,0459759	FrameToFrame	0,0469752
FrameToFrame	260,401	FrameToFrame	15,7353	FrameToFrame	1,31252	FrameToFrame	0,0470317	FrameToFrame	0,0479846

FrameToFrame	267,795	FrameToFrame	15,6094	FrameToFrame	1,23445	FrameToFrame	0,0479586	FrameToFrame	0,0459675
FrameToFrame	260,984	FrameToFrame	15,484	FrameToFrame	1,28115	FrameToFrame	0,0459598	FrameToFrame	0,047981
FrameToFrame	268,106	FrameToFrame	15,7349	FrameToFrame	1,29696	FrameToFrame	0,0479637	FrameToFrame	0,0449457
FrameToFrame	260,044	FrameToFrame	15,733	FrameToFrame	1,26552	FrameToFrame	0,0455021	FrameToFrame	0,0480057
FrameToFrame	260,404	FrameToFrame	15,3296	FrameToFrame	1,29686	FrameToFrame	0,0469813	FrameToFrame	0,0459563
FrameToFrame	281,327	FrameToFrame	15,952	FrameToFrame	1,28128	FrameToFrame	0,0469768	FrameToFrame	0,0479669
FrameToFrame	260,092	FrameToFrame	15,5792	FrameToFrame	1,28122	FrameToFrame	0,0470423	FrameToFrame	0,0459573
FrameToFrame	273,873	FrameToFrame	16,2173	FrameToFrame	1,29687	FrameToFrame	0,045947	FrameToFrame	0,0479836
FrameToFrame	259,482	FrameToFrame	15,7809	FrameToFrame	1,29689	FrameToFrame	0,0469675	FrameToFrame	0,0469788
FrameToFrame	259,295	FrameToFrame	15,7518	FrameToFrame	1,26561	FrameToFrame	0,0480012	FrameToFrame	0,0459781
FrameToFrame	280,046	FrameToFrame	16,4825	FrameToFrame	1,32812	FrameToFrame	0,0469647	FrameToFrame	0,0459951
FrameToFrame	259,016	FrameToFrame	15,8905	FrameToFrame	1,34376	FrameToFrame	0,0459608	FrameToFrame	0,0479583
FrameToFrame	266,839	FrameToFrame	16,0628	FrameToFrame	1,34375	FrameToFrame	0,045965	FrameToFrame	0,046955
FrameToFrame	260,125	FrameToFrame	15,5943	FrameToFrame	1,3125	FrameToFrame	0,0469653	FrameToFrame	0,0469852
FrameToFrame	266,593	FrameToFrame	15,8287	FrameToFrame	1,29686	FrameToFrame	0,0479862	FrameToFrame	0,045982
FrameToFrame	266,058	FrameToFrame	16,1078	FrameToFrame	1,31253	FrameToFrame	0,0459743	FrameToFrame	0,0459678
FrameToFrame	252,372	FrameToFrame	15,4686	FrameToFrame	1,28135	FrameToFrame	0,046982	FrameToFrame	0,0479814
FrameToFrame	266,155	FrameToFrame	16,1575	FrameToFrame	1,39052	FrameToFrame	0,0469768	FrameToFrame	0,045982
FrameToFrame	266,172	FrameToFrame	16,4054	FrameToFrame	1,40639	FrameToFrame	0,0469974	FrameToFrame	0,0479839
FrameToFrame	273,908	FrameToFrame	16,3126	FrameToFrame	1,32798	FrameToFrame	0,0469611	FrameToFrame	0,0469797
FrameToFrame	267,464	FrameToFrame	16,265	FrameToFrame	1,32812	FrameToFrame	0,0459605	FrameToFrame	0,0460018
FrameToFrame	279,845	FrameToFrame	16,5633	FrameToFrame	1,3594	FrameToFrame	0,0469653	FrameToFrame	0,0479769
FrameToFrame	280,2	FrameToFrame	16,469	FrameToFrame	1,34392	FrameToFrame	0,0479817	FrameToFrame	0,0459624
FrameToFrame	265,764	FrameToFrame	16,3745	FrameToFrame	1,32793	FrameToFrame	0,0459573	FrameToFrame	0,0469602
FrameToFrame	273,421	FrameToFrame	16,3901	FrameToFrame	1,35936	FrameToFrame	0,0480734	FrameToFrame	0,140472
FrameToFrame	279,766	FrameToFrame	16,5933	FrameToFrame	1,37504	FrameToFrame	0,0458598	FrameToFrame	0,0459614
FrameToFrame	273,141	FrameToFrame	16,3602	FrameToFrame	1,34378	FrameToFrame	0,0459682	FrameToFrame	0,0479782
FrameToFrame	265,233	FrameToFrame	16,4216	FrameToFrame	1,31249	FrameToFrame	0,0479817	FrameToFrame	0,0459803
FrameToFrame	272,67	FrameToFrame	16,5782	FrameToFrame	1,31252	FrameToFrame	0,0459646	FrameToFrame	0,0479929
FrameToFrame	265,873	FrameToFrame	16,3129	FrameToFrame	1,3437	FrameToFrame	0,046991	FrameToFrame	0,046955
FrameToFrame	287,422	FrameToFrame	16,6095	FrameToFrame	1,375	FrameToFrame	1,37575	FrameToFrame	0,0449858
FrameToFrame	272,502	FrameToFrame	16,4993	FrameToFrame	1,34376	FrameToFrame	0,0459666	FrameToFrame	0,0479557
FrameToFrame	271,906	FrameToFrame	16,5934	FrameToFrame	1,375	FrameToFrame	0,0469672	FrameToFrame	0,04599
FrameToFrame	272,014	FrameToFrame	16,7181	FrameToFrame	1,37498	FrameToFrame	0,0459627	FrameToFrame	0,0479778
FrameToFrame	278,358	FrameToFrame	16,9387	FrameToFrame	1,37501	FrameToFrame	0,0479753	FrameToFrame	0,0459573
FrameToFrame	285,952	FrameToFrame	16,7178	FrameToFrame	1,34377	FrameToFrame	0,046948	FrameToFrame	0,0469801
FrameToFrame	263,889	FrameToFrame	16,8909	FrameToFrame	1,35952	FrameToFrame	0,0459653	FrameToFrame	0,0459566
FrameToFrame	285,733	FrameToFrame	16,7338	FrameToFrame	1,39049	FrameToFrame	0,0479852	FrameToFrame	0,0479769
FrameToFrame	278,392	FrameToFrame	17,0317	FrameToFrame	1,32811	FrameToFrame	0,0459839	FrameToFrame	0,0459832
FrameToFrame	278,589	FrameToFrame	16,7651	FrameToFrame	1,35935	FrameToFrame	0,0469762	FrameToFrame	0,047982
FrameToFrame	285,458	FrameToFrame	17,313	FrameToFrame	1,35941	FrameToFrame	0,0459874	FrameToFrame	0,0459258
FrameToFrame	270,962	FrameToFrame	16,797	FrameToFrame	1,40623	FrameToFrame	0,0468704	FrameToFrame	0,046974

FrameToFrame	272,438	FrameToFrame	16,7491	FrameToFrame	1,39064	FrameToFrame	0,0479589	FrameToFrame	0,0468826
FrameToFrame	272,621	FrameToFrame	16,3916	FrameToFrame	1,34375	FrameToFrame	0,0469775	FrameToFrame	0,0469961
FrameToFrame	272,717	FrameToFrame	16,8277	FrameToFrame	1,32813	FrameToFrame	0,0459759	FrameToFrame	0,0459486
FrameToFrame	265,42	FrameToFrame	16,1873	FrameToFrame	1,29687	FrameToFrame	0,0479894	FrameToFrame	0,0479679
FrameToFrame	272,92	FrameToFrame	16,5937	FrameToFrame	1,37501	FrameToFrame	0,0449835	FrameToFrame	0,0469708
FrameToFrame	265,561	FrameToFrame	16,1573	FrameToFrame	1,34385	FrameToFrame	0,0479782	FrameToFrame	0,0469865
FrameToFrame	279,358	FrameToFrame	16,7019	FrameToFrame	1,34362	FrameToFrame	0,0459752	FrameToFrame	0,0469435
FrameToFrame	273,28	FrameToFrame	16,2654	FrameToFrame	1,375	FrameToFrame	0,0479868	FrameToFrame	0,044963
FrameToFrame	273,389	FrameToFrame	16,4385	FrameToFrame	1,3125	FrameToFrame	0,046964	FrameToFrame	0,0479878
FrameToFrame	264,657	FrameToFrame	16,6401	FrameToFrame	1,3908	FrameToFrame	0,0469813	FrameToFrame	0,0469784
FrameToFrame	273,543	FrameToFrame	16,422	FrameToFrame	1,32797	FrameToFrame	0,045947	FrameToFrame	0,046964
FrameToFrame	265,686	FrameToFrame	16,4064	FrameToFrame	1,32812	FrameToFrame	0,0459643	FrameToFrame	0,0459787
FrameToFrame	280,251	FrameToFrame	16,7816	FrameToFrame	1,35938	FrameToFrame	0,0479656	FrameToFrame	0,0479615
FrameToFrame	265,136	FrameToFrame	16,4679	FrameToFrame	1,39055	FrameToFrame	0,045981	FrameToFrame	0,0459896
FrameToFrame	271,092	FrameToFrame	16,9833	FrameToFrame	1,37512	FrameToFrame	0,0469804	FrameToFrame	0,0459406
FrameToFrame	273,03	FrameToFrame	16,7191	FrameToFrame	1,35926	FrameToFrame	0,0469653	FrameToFrame	0,0469521
FrameToFrame	286,296	FrameToFrame	16,7338	FrameToFrame	1,35934	FrameToFrame	0,0479621	FrameToFrame	0,0479596
FrameToFrame	272,389	FrameToFrame	16,6258	FrameToFrame	1,35939	FrameToFrame	0,0459666	FrameToFrame	0,0459781
FrameToFrame	265,373	FrameToFrame	16,5006	FrameToFrame	1,32814	FrameToFrame	0,0459601	FrameToFrame	0,0479842
FrameToFrame	272,232	FrameToFrame	16,7029	FrameToFrame	1,31247	FrameToFrame	0,0479746	FrameToFrame	0,045973
FrameToFrame	264,86	FrameToFrame	16,6246	FrameToFrame	1,35939	FrameToFrame	1,34301	FrameToFrame	0,0459823
FrameToFrame	279,044	FrameToFrame	16,9217	FrameToFrame	1,48436	FrameToFrame	0,0479676	FrameToFrame	0,0479791
FrameToFrame	264,326	FrameToFrame	16,9072	FrameToFrame	1,37517	FrameToFrame	0,0449768	FrameToFrame	0,047007
FrameToFrame	271,733	FrameToFrame	16,9685	FrameToFrame	1,37487	FrameToFrame	0,0479759	FrameToFrame	0,0469464
FrameToFrame	272,764	FrameToFrame	16,7639	FrameToFrame	1,34372	FrameToFrame	0,0469653	FrameToFrame	0,0469634
FrameToFrame	278,546	FrameToFrame	17,1253	FrameToFrame	1,40626	FrameToFrame	0,0469688	FrameToFrame	0,0459775
FrameToFrame	270,72	FrameToFrame	17,0476	FrameToFrame	1,35938	FrameToFrame	0,0459585	FrameToFrame	0,0459858
FrameToFrame	291,375	FrameToFrame	17,6551	FrameToFrame	1,39062	FrameToFrame	0,0469708	FrameToFrame	0,0469695
FrameToFrame	277,636	FrameToFrame	17,3295	FrameToFrame	1,3751	FrameToFrame	0,046955	FrameToFrame	0,0469797
FrameToFrame	263,529	FrameToFrame	16,719	FrameToFrame	1,39052	FrameToFrame	0,0459662	FrameToFrame	0,0469762
FrameToFrame	278,202	FrameToFrame	17,2654	FrameToFrame	1,34387	FrameToFrame	0,0479903	FrameToFrame	0,0479923
FrameToFrame	285,202	FrameToFrame	17,4207	FrameToFrame	1,40615	FrameToFrame	0,046965	FrameToFrame	0,0459617
TotalTime	52862,844	TotalTime	3097,5281	TotalTime	259,69944	TotalTime	17,0419084	TotalTime	10,7699744

Simple scenario - Scaling

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	227,978	FrameToFrame	11,9202	FrameToFrame	0,984197	FrameToFrame	1,03023	FrameToFrame	0,772184
FrameToFrame	222,497	FrameToFrame	11,7664	FrameToFrame	0,893854	FrameToFrame	0,0458415	FrameToFrame	0,047838
FrameToFrame	222,735	FrameToFrame	11,1704	FrameToFrame	0,895293	FrameToFrame	0,0469624	FrameToFrame	0,0459826
FrameToFrame	223,562	FrameToFrame	11,0466	FrameToFrame	0,925281	FrameToFrame	0,0479846	FrameToFrame	0,046965
FrameToFrame	222,048	FrameToFrame	11,4855	FrameToFrame	0,909058	FrameToFrame	0,0459868	FrameToFrame	0,0459775
FrameToFrame	221,893	FrameToFrame	11,3903	FrameToFrame	0,877575	FrameToFrame	0,0696493	FrameToFrame	0,0469784
FrameToFrame	226,104	FrameToFrame	11,5787	FrameToFrame	0,893729	FrameToFrame	0,0557641	FrameToFrame	0,047981
FrameToFrame	222,813	FrameToFrame	11,592	FrameToFrame	0,937475	FrameToFrame	0,0459787	FrameToFrame	0,04598
FrameToFrame	229,534	FrameToFrame	11,6735	FrameToFrame	0,893263	FrameToFrame	0,047983	FrameToFrame	0,0469993
FrameToFrame	222,688	FrameToFrame	11,4218	FrameToFrame	1,08569	FrameToFrame	0,0459964	FrameToFrame	0,0459322
FrameToFrame	223,635	FrameToFrame	11,4985	FrameToFrame	1,00011	FrameToFrame	0,0459415	FrameToFrame	0,0479698
FrameToFrame	225,272	FrameToFrame	11,6264	FrameToFrame	0,910103	FrameToFrame	0,0479852	FrameToFrame	0,0469765
FrameToFrame	224,531	FrameToFrame	11,4672	FrameToFrame	0,924136	FrameToFrame	0,0459637	FrameToFrame	0,0469634
FrameToFrame	224,241	FrameToFrame	11,4546	FrameToFrame	0,938257	FrameToFrame	0,0479637	FrameToFrame	0,0469768
FrameToFrame	226,467	FrameToFrame	11,7955	FrameToFrame	0,90708	FrameToFrame	0,0459829	FrameToFrame	0,0469727
FrameToFrame	225,211	FrameToFrame	11,1418	FrameToFrame	0,89719	FrameToFrame	0,04699	FrameToFrame	0,0449787
FrameToFrame	224,604	FrameToFrame	11,6713	FrameToFrame	0,877352	FrameToFrame	0,047982	FrameToFrame	0,0479814
FrameToFrame	147,012	FrameToFrame	10,9996	FrameToFrame	0,923133	FrameToFrame	0,0449781	FrameToFrame	0,0469775
FrameToFrame	227,104	FrameToFrame	11,5785	FrameToFrame	0,891785	FrameToFrame	0,0468434	FrameToFrame	0,0469855
FrameToFrame	226,452	FrameToFrame	11,6716	FrameToFrame	0,908255	FrameToFrame	0,0469784	FrameToFrame	0,0469961
FrameToFrame	228,918	FrameToFrame	11,8286	FrameToFrame	0,940944	FrameToFrame	0,0470124	FrameToFrame	0,0459303
FrameToFrame	231,88	FrameToFrame	11,0464	FrameToFrame	0,96961	FrameToFrame	0,0479612	FrameToFrame	0,105223
FrameToFrame	224,39	FrameToFrame	11,6878	FrameToFrame	0,907622	FrameToFrame	0,0469752	FrameToFrame	0,0512659
FrameToFrame	224,497	FrameToFrame	11,6715	FrameToFrame	0,945817	FrameToFrame	0,0469701	FrameToFrame	0,0469948
FrameToFrame	225,859	FrameToFrame	10,9367	FrameToFrame	0,888281	FrameToFrame	0,0449588	FrameToFrame	0,0459547
FrameToFrame	226,191	FrameToFrame	11,7803	FrameToFrame	0,922402	FrameToFrame	0,0469887	FrameToFrame	0,0469922
FrameToFrame	226,28	FrameToFrame	11,9079	FrameToFrame	0,937532	FrameToFrame	0,0469396	FrameToFrame	0,0479705
FrameToFrame	227,466	FrameToFrame	11,4829	FrameToFrame	0,908141	FrameToFrame	0,0479775	FrameToFrame	0,0459781
FrameToFrame	164,368	FrameToFrame	11,4851	FrameToFrame	0,937476	FrameToFrame	0,0469579	FrameToFrame	0,0469791
FrameToFrame	226,107	FrameToFrame	11,7504	FrameToFrame	0,953105	FrameToFrame	0,0469727	FrameToFrame	0,0459733
FrameToFrame	234,932	FrameToFrame	11,7175	FrameToFrame	0,95313	FrameToFrame	0,0459646	FrameToFrame	0,0479846
FrameToFrame	233,481	FrameToFrame	11,6107	FrameToFrame	0,984365	FrameToFrame	0,0469631	FrameToFrame	0,0459579
FrameToFrame	239,264	FrameToFrame	11,7798	FrameToFrame	0,937458	FrameToFrame	0,0469582	FrameToFrame	0,0469916
FrameToFrame	235,655	FrameToFrame	11,9697	FrameToFrame	0,984373	FrameToFrame	0,0459797	FrameToFrame	0,0469714
FrameToFrame	239,295	FrameToFrame	12,1403	FrameToFrame	1,00396	FrameToFrame	0,048109	FrameToFrame	0,0469573
FrameToFrame	229,81	FrameToFrame	11,5476	FrameToFrame	0,964398	FrameToFrame	0,0458501	FrameToFrame	0,0459573
FrameToFrame	237,637	FrameToFrame	11,8589	FrameToFrame	1,00013	FrameToFrame	0,0470198	FrameToFrame	0,0469752
FrameToFrame	240,591	FrameToFrame	15,9531	FrameToFrame	1,01546	FrameToFrame	0,0469242	FrameToFrame	0,0469942
FrameToFrame	244,669	FrameToFrame	12,14	FrameToFrame	0,984352	FrameToFrame	0,0459759	FrameToFrame	0,0479583
FrameToFrame	236,903	FrameToFrame	13,5007	FrameToFrame	0,984491	FrameToFrame	0,0479814	FrameToFrame	0,0469627
FrameToFrame	242,825	FrameToFrame	12,2491	FrameToFrame	0,953148	FrameToFrame	0,0469784	FrameToFrame	0,0459592

FrameToFrame	247,138	FrameToFrame	12,2351	FrameToFrame	0,984351	FrameToFrame	0,0459845	FrameToFrame	0,0459909
FrameToFrame	250,013	FrameToFrame	12,6254	FrameToFrame	0,999828	FrameToFrame	0,0479788	FrameToFrame	0,0469679
FrameToFrame	249,513	FrameToFrame	12,2802	FrameToFrame	1	FrameToFrame	0,0459797	FrameToFrame	0,0469772
FrameToFrame	255,795	FrameToFrame	12,6875	FrameToFrame	1,03125	FrameToFrame	0,0469884	FrameToFrame	0,0469858
FrameToFrame	252,781	FrameToFrame	12,9697	FrameToFrame	0,968821	FrameToFrame	0,0469765	FrameToFrame	0,0479663
FrameToFrame	254,688	FrameToFrame	12,6544	FrameToFrame	1,0155	FrameToFrame	0,0459704	FrameToFrame	0,0449762
FrameToFrame	257	FrameToFrame	12,5635	FrameToFrame	1,01563	FrameToFrame	0,0469566	FrameToFrame	0,0479814
FrameToFrame	261,627	FrameToFrame	12,3758	FrameToFrame	1,21876	FrameToFrame	0,0479804	FrameToFrame	0,0459762
FrameToFrame	263,514	FrameToFrame	12,6856	FrameToFrame	1,07811	FrameToFrame	0,047015	FrameToFrame	0,0469791
FrameToFrame	270,311	FrameToFrame	13,5784	FrameToFrame	1,14071	FrameToFrame	0,0459351	FrameToFrame	0,046974
FrameToFrame	275,139	FrameToFrame	13,3128	FrameToFrame	1,09376	FrameToFrame	0,045965	FrameToFrame	0,0479791
FrameToFrame	277,858	FrameToFrame	13,6875	FrameToFrame	1,09362	FrameToFrame	0,0479859	FrameToFrame	0,0459871
FrameToFrame	282,233	FrameToFrame	13,4989	FrameToFrame	1,14078	FrameToFrame	0,0459672	FrameToFrame	0,0469675
FrameToFrame	273,015	FrameToFrame	13,3919	FrameToFrame	1,06234	FrameToFrame	0,0479785	FrameToFrame	0,0469791
FrameToFrame	276,013	FrameToFrame	13,6711	FrameToFrame	1,07811	FrameToFrame	0,0462247	FrameToFrame	0,0469951
FrameToFrame	281,952	FrameToFrame	13,7651	FrameToFrame	1,20313	FrameToFrame	0,0459803	FrameToFrame	0,0469653
FrameToFrame	284,905	FrameToFrame	14,4504	FrameToFrame	1,15623	FrameToFrame	0,0479823	FrameToFrame	0,0469804
FrameToFrame	288,39	FrameToFrame	14,6547	FrameToFrame	1,23438	FrameToFrame	0,0460984	FrameToFrame	0,0459852
FrameToFrame	297,687	FrameToFrame	15,1093	FrameToFrame	1,21876	FrameToFrame	0,0468482	FrameToFrame	0,0459778
FrameToFrame	299,218	FrameToFrame	15,2817	FrameToFrame	1,25007	FrameToFrame	0,0469788	FrameToFrame	0,0469714
FrameToFrame	310,547	FrameToFrame	15,4535	FrameToFrame	1,2344	FrameToFrame	0,0459784	FrameToFrame	0,0479579
FrameToFrame	319,955	FrameToFrame	16,1717	FrameToFrame	1,32824	FrameToFrame	0,0479794	FrameToFrame	0,0458447
FrameToFrame	314,125	FrameToFrame	15,8897	FrameToFrame	1,31253	FrameToFrame	0,0459813	FrameToFrame	0,0479836
FrameToFrame	321,081	FrameToFrame	16,0792	FrameToFrame	1,32813	FrameToFrame	0,0479852	FrameToFrame	0,0469881
FrameToFrame	328,187	FrameToFrame	16,3438	FrameToFrame	1,34372	FrameToFrame	0,0471696	FrameToFrame	0,0459707
FrameToFrame	339,939	FrameToFrame	16,8588	FrameToFrame	1,35943	FrameToFrame	0,0459803	FrameToFrame	0,0469621
FrameToFrame	341,83	FrameToFrame	16,8597	FrameToFrame	1,42183	FrameToFrame	0,0459582	FrameToFrame	0,0469791
FrameToFrame	345,503	FrameToFrame	16,515	FrameToFrame	1,45416	FrameToFrame	0,0469813	FrameToFrame	0,0469813
FrameToFrame	356,427	FrameToFrame	17,5475	FrameToFrame	1,3663	FrameToFrame	0,0479817	FrameToFrame	0,045982
FrameToFrame	366,595	FrameToFrame	18,0767	FrameToFrame	1,46108	FrameToFrame	0,0459836	FrameToFrame	0,0469884
FrameToFrame	361,222	FrameToFrame	17,6253	FrameToFrame	1,45792	FrameToFrame	0,0479717	FrameToFrame	0,0469932
FrameToFrame	365,456	FrameToFrame	18,3599	FrameToFrame	1,4881	FrameToFrame	0,0459826	FrameToFrame	0,0469582
FrameToFrame	375,879	FrameToFrame	18,9059	FrameToFrame	1,53458	FrameToFrame	0,0469679	FrameToFrame	0,0459787
FrameToFrame	382,738	FrameToFrame	18,5318	FrameToFrame	1,52795	FrameToFrame	0,0459771	FrameToFrame	0,0479823
FrameToFrame	387,176	FrameToFrame	18,7024	FrameToFrame	1,61645	FrameToFrame	0,0479952	FrameToFrame	0,0469823
FrameToFrame	390,647	FrameToFrame	19,6878	FrameToFrame	1,64252	FrameToFrame	0,0469881	FrameToFrame	0,0459733
FrameToFrame	406,034	FrameToFrame	20,3113	FrameToFrame	1,67193	FrameToFrame	0,0459338	FrameToFrame	0,0469499
FrameToFrame	412,445	FrameToFrame	20,079	FrameToFrame	1,65966	FrameToFrame	0,0469836	FrameToFrame	0,0469817
FrameToFrame	413,113	FrameToFrame	20,093	FrameToFrame	1,65629	FrameToFrame	0,0460936	FrameToFrame	0,0469807
FrameToFrame	423,837	FrameToFrame	20,8914	FrameToFrame	1,73443	FrameToFrame	0,0468505	FrameToFrame	0,046982
FrameToFrame	450,38	FrameToFrame	21,8128	FrameToFrame	1,76563	FrameToFrame	0,0469666	FrameToFrame	0,0469765
FrameToFrame	451,726	FrameToFrame	22,3895	FrameToFrame	1,81254	FrameToFrame	0,0469797	FrameToFrame	0,0459765

FrameToFrame	460,883	FrameToFrame	22,3278	FrameToFrame	1,7813	FrameToFrame	0,0469897	FrameToFrame	0,0479788
FrameToFrame	465,352	FrameToFrame	22,7815	FrameToFrame	1,78119	FrameToFrame	0,0469675	FrameToFrame	0,0459816
FrameToFrame	474,853	FrameToFrame	23,2345	FrameToFrame	1,87607	FrameToFrame	0,0459813	FrameToFrame	0,0469775
FrameToFrame	493,372	FrameToFrame	23,6563	FrameToFrame	1,9114	FrameToFrame	0,0479923	FrameToFrame	0,0469791
FrameToFrame	501,976	FrameToFrame	24,0779	FrameToFrame	2,04998	FrameToFrame	0,0469784	FrameToFrame	0,0469756
FrameToFrame	501,245	FrameToFrame	24,5786	FrameToFrame	2,06585	FrameToFrame	0,046014	FrameToFrame	0,0469865
FrameToFrame	506,26	FrameToFrame	25,0152	FrameToFrame	2,02016	FrameToFrame	0,0479316	FrameToFrame	0,0459925
FrameToFrame	517,402	FrameToFrame	25,3745	FrameToFrame	2,14544	FrameToFrame	0,046007	FrameToFrame	0,0469983
FrameToFrame	524,996	FrameToFrame	25,0938	FrameToFrame	2,0862	FrameToFrame	0,0459983	FrameToFrame	0,0469396
FrameToFrame	533,967	FrameToFrame	25,3133	FrameToFrame	2,23797	FrameToFrame	0,0479438	FrameToFrame	0,0469845
FrameToFrame	535,027	FrameToFrame	25,9367	FrameToFrame	2,14832	FrameToFrame	0,0459803	FrameToFrame	0,0469701
FrameToFrame	549,371	FrameToFrame	26,1883	FrameToFrame	2,18013	FrameToFrame	0,0479817	FrameToFrame	0,045981
FrameToFrame	549,391	FrameToFrame	26,7973	FrameToFrame	2,17549	FrameToFrame	0,046981	FrameToFrame	0,0479826
FrameToFrame	565,512	FrameToFrame	27,2013	FrameToFrame	2,28924	FrameToFrame	0,046956	FrameToFrame	0,0459775
FrameToFrame	566,545	FrameToFrame	27,7517	FrameToFrame	2,34402	FrameToFrame	0,0469781	FrameToFrame	0,0469813
FrameToFrame	584,873	FrameToFrame	27,7028	FrameToFrame	2,36215	FrameToFrame	0,0450089	FrameToFrame	0,0470673
FrameToFrame	589,14	FrameToFrame	28,5929	FrameToFrame	2,36324	FrameToFrame	0,0470047	FrameToFrame	0,0468758
FrameToFrame	592,499	FrameToFrame	27,7179	FrameToFrame	2,35313	FrameToFrame	0,0479907	FrameToFrame	0,0459797
FrameToFrame	602,328	FrameToFrame	29,0016	FrameToFrame	2,40117	FrameToFrame	0,0471898	FrameToFrame	0,0469775
FrameToFrame	605,891	FrameToFrame	29,1563	FrameToFrame	2,46893	FrameToFrame	0,0449531	FrameToFrame	0,0469881
FrameToFrame	608,081	FrameToFrame	29,4532	FrameToFrame	2,4621	FrameToFrame	0,0479714	FrameToFrame	0,0479785
FrameToFrame	615,514	FrameToFrame	29,9043	FrameToFrame	2,56992	FrameToFrame	0,0459768	FrameToFrame	0,0469656
FrameToFrame	625,64	FrameToFrame	30,0944	FrameToFrame	2,58722	FrameToFrame	0,0479775	FrameToFrame	0,0459916
FrameToFrame	628,598	FrameToFrame	30,1554	FrameToFrame	2,55186	FrameToFrame	0,0459784	FrameToFrame	0,0469691
FrameToFrame	643,001	FrameToFrame	31,1263	FrameToFrame	2,54361	FrameToFrame	0,047134	FrameToFrame	0,046982
FrameToFrame	644,485	FrameToFrame	31,4845	FrameToFrame	2,53877	FrameToFrame	0,045981	FrameToFrame	0,0459989
FrameToFrame	648,174	FrameToFrame	30,5453	FrameToFrame	2,61813	FrameToFrame	0,048016	FrameToFrame	0,0469313
FrameToFrame	658,721	FrameToFrame	32,0161	FrameToFrame	2,77207	FrameToFrame	0,0459861	FrameToFrame	0,0479794
FrameToFrame	663,378	FrameToFrame	32,0788	FrameToFrame	2,7857	FrameToFrame	0,0464021	FrameToFrame	0,0449829
FrameToFrame	667,613	FrameToFrame	32,3425	FrameToFrame	2,69133	FrameToFrame	0,047964	FrameToFrame	0,0469784
FrameToFrame	676,759	FrameToFrame	32,2356	FrameToFrame	2,69305	FrameToFrame	0,0469669	FrameToFrame	0,0469765
FrameToFrame	685,862	FrameToFrame	33,4051	FrameToFrame	2,80123	FrameToFrame	0,0459755	FrameToFrame	0,0469765
FrameToFrame	688,395	FrameToFrame	33,1723	FrameToFrame	2,78834	FrameToFrame	0,0469624	FrameToFrame	0,046982
FrameToFrame	710,865	FrameToFrame	34,4383	FrameToFrame	2,91456	FrameToFrame	0,0479769	FrameToFrame	0,0475991
FrameToFrame	713,427	FrameToFrame	34,2022	FrameToFrame	2,91155	FrameToFrame	0,0459675	FrameToFrame	0,0459582
FrameToFrame	730,054	FrameToFrame	35,0303	FrameToFrame	2,94028	FrameToFrame	0,0469576	FrameToFrame	0,0479865
FrameToFrame	733,607	FrameToFrame	35,2518	FrameToFrame	2,89176	FrameToFrame	0,0459922	FrameToFrame	0,0469813
FrameToFrame	741,398	FrameToFrame	35,5459	FrameToFrame	3,03866	FrameToFrame	0,0469461	FrameToFrame	0,0459762
FrameToFrame	750,646	FrameToFrame	36,7648	FrameToFrame	3,04704	FrameToFrame	0,0479814	FrameToFrame	0,0469813
FrameToFrame	762,492	FrameToFrame	36,86	FrameToFrame	3,14128	FrameToFrame	0,0469813	FrameToFrame	0,0459848
FrameToFrame	767,542	FrameToFrame	37,4534	FrameToFrame	3,12517	FrameToFrame	0,0459566	FrameToFrame	0,0470612
FrameToFrame	779,397	FrameToFrame	36,6706	FrameToFrame	3,18748	FrameToFrame	0,0470824	FrameToFrame	0,047152

FrameToFrame	741,616	FrameToFrame	38,1574	FrameToFrame	3,14084	FrameToFrame	0,0459986	FrameToFrame	0,0468377
FrameToFrame	817,857	FrameToFrame	39,0308	FrameToFrame	3,14663	FrameToFrame	3,57903	FrameToFrame	0,0459714
FrameToFrame	829,259	FrameToFrame	39,9993	FrameToFrame	3,36774	FrameToFrame	0,0469355	FrameToFrame	0,0469656
FrameToFrame	841,7	FrameToFrame	39,6564	FrameToFrame	3,3965	FrameToFrame	0,0469752	FrameToFrame	0,0469659
FrameToFrame	866,062	FrameToFrame	41,61	FrameToFrame	3,63372	FrameToFrame	0,0469586	FrameToFrame	0,0479599
FrameToFrame	857,932	FrameToFrame	40,921	FrameToFrame	3,43959	FrameToFrame	0,0459855	FrameToFrame	0,0469775
FrameToFrame	889,608	FrameToFrame	42,2671	FrameToFrame	3,454	FrameToFrame	0,0469595	FrameToFrame	0,0469807
FrameToFrame	897,593	FrameToFrame	42,5611	FrameToFrame	3,57864	FrameToFrame	0,0469695	FrameToFrame	0,0449816
FrameToFrame	899,75	FrameToFrame	43,2186	FrameToFrame	3,57825	FrameToFrame	0,0469653	FrameToFrame	0,0469791
FrameToFrame	910,047	FrameToFrame	43,5952	FrameToFrame	3,71889	FrameToFrame	0,0469974	FrameToFrame	0,0469868
FrameToFrame	918,86	FrameToFrame	44,1406	FrameToFrame	3,67395	FrameToFrame	0,0459736	FrameToFrame	0,0469675
FrameToFrame	932,939	FrameToFrame	44,9051	FrameToFrame	3,66999	FrameToFrame	0,0479034	FrameToFrame	0,0469672
FrameToFrame	940,689	FrameToFrame	45,2195	FrameToFrame	3,74165	FrameToFrame	0,0449563	FrameToFrame	0,0479782
FrameToFrame	951,036	FrameToFrame	45,0454	FrameToFrame	3,80059	FrameToFrame	0,0469845	FrameToFrame	0,0449861
FrameToFrame	964,036	FrameToFrame	46,2667	FrameToFrame	3,81979	FrameToFrame	0,0470086	FrameToFrame	0,0479749
FrameToFrame	969,831	FrameToFrame	46,1095	FrameToFrame	3,85807	FrameToFrame	0,0479708	FrameToFrame	0,0469784
FrameToFrame	1003,57	FrameToFrame	47,8745	FrameToFrame	4,05707	FrameToFrame	0,046982	FrameToFrame	0,0469801
FrameToFrame	1014,66	FrameToFrame	48,1408	FrameToFrame	4,14873	FrameToFrame	0,0469685	FrameToFrame	0,0459678
FrameToFrame	1033,51	FrameToFrame	49,0478	FrameToFrame	4,1073	FrameToFrame	0,0451567	FrameToFrame	0,0469797
FrameToFrame	1053,98	FrameToFrame	50,2476	FrameToFrame	4,16524	FrameToFrame	0,0479833	FrameToFrame	0,047982
FrameToFrame	1065,9	FrameToFrame	50,8138	FrameToFrame	4,2346	FrameToFrame	0,0469784	FrameToFrame	0,045965
FrameToFrame	1086,7	FrameToFrame	52,3273	FrameToFrame	4,23775	FrameToFrame	0,0469987	FrameToFrame	0,0469829
FrameToFrame	1099,82	FrameToFrame	52,5629	FrameToFrame	4,3077	FrameToFrame	0,0459473	FrameToFrame	0,0469589
FrameToFrame	1112,31	FrameToFrame	53,3742	FrameToFrame	4,54626	FrameToFrame	0,0479814	FrameToFrame	0,0459685
FrameToFrame	1129,17	FrameToFrame	53,9231	FrameToFrame	4,45327	FrameToFrame	0,0459807	FrameToFrame	0,0469611
FrameToFrame	1149,17	FrameToFrame	53,7179	FrameToFrame	4,5314	FrameToFrame	0,0459807	FrameToFrame	0,0469881
FrameToFrame	1161,82	FrameToFrame	54,9063	FrameToFrame	4,57848	FrameToFrame	0,0479788	FrameToFrame	0,0469579
FrameToFrame	1183,33	FrameToFrame	55,5621	FrameToFrame	4,73454	FrameToFrame	0,0469759	FrameToFrame	0,046972
FrameToFrame	1174,61	FrameToFrame	55,7356	FrameToFrame	4,7348	FrameToFrame	0,045981	FrameToFrame	0,0459864
FrameToFrame	1185,53	FrameToFrame	56,1243	FrameToFrame	4,67275	FrameToFrame	0,0469964	FrameToFrame	0,0469695
FrameToFrame	1200,25	FrameToFrame	57,0784	FrameToFrame	4,78052	FrameToFrame	0,0460968	FrameToFrame	0,0469794
FrameToFrame	1220,28	FrameToFrame	57,7645	FrameToFrame	4,95066	FrameToFrame	0,0480125	FrameToFrame	0,0469871
FrameToFrame	1225,95	FrameToFrame	58,0161	FrameToFrame	4,86185	FrameToFrame	0,0458934	FrameToFrame	0,0469691
FrameToFrame	1231,67	FrameToFrame	57,8754	FrameToFrame	4,93776	FrameToFrame	0,0479451	FrameToFrame	0,0469813
FrameToFrame	1237,48	FrameToFrame	58,2805	FrameToFrame	5,00036	FrameToFrame	0,0460006	FrameToFrame	0,0469813
FrameToFrame	1238,91	FrameToFrame	59,0003	FrameToFrame	4,97972	FrameToFrame	0,0475907	FrameToFrame	0,0469801
FrameToFrame	1247,2	FrameToFrame	59,0005	FrameToFrame	5,06596	FrameToFrame	0,0620783	FrameToFrame	0,0469858
FrameToFrame	1246,42	FrameToFrame	59,0616	FrameToFrame	4,99763	FrameToFrame	0,0459701	FrameToFrame	0,0469669
FrameToFrame	1237,3	FrameToFrame	58,5627	FrameToFrame	4,95277	FrameToFrame	0,0479801	FrameToFrame	0,0469605
FrameToFrame	1245,81	FrameToFrame	58,7342	FrameToFrame	4,98053	FrameToFrame	0,0459771	FrameToFrame	0,0459781
FrameToFrame	1247,25	FrameToFrame	59,4994	FrameToFrame	4,89658	FrameToFrame	0,0479121	FrameToFrame	0,0468758
FrameToFrame	1235,75	FrameToFrame	58,4686	FrameToFrame	4,95009	FrameToFrame	0,0450121	FrameToFrame	0,0459855

FrameToFrame	1239,43	FrameToFrame	58,9075	FrameToFrame	4,99252	FrameToFrame	0,0479288	FrameToFrame	0,0469701
FrameToFrame	1235,22	FrameToFrame	59,1554	FrameToFrame	5,01054	FrameToFrame	0,0469656	FrameToFrame	0,0469967
FrameToFrame	1225,68	FrameToFrame	58,4536	FrameToFrame	5,03662	FrameToFrame	0,0459743	FrameToFrame	0,0479695
FrameToFrame	1224,02	FrameToFrame	58,6251	FrameToFrame	5,02024	FrameToFrame	0,0470237	FrameToFrame	0,0459633
FrameToFrame	1211,28	FrameToFrame	58,593	FrameToFrame	4,84379	FrameToFrame	0,0469759	FrameToFrame	0,0469621
FrameToFrame	1217,86	FrameToFrame	57,86	FrameToFrame	4,89092	FrameToFrame	0,0459589	FrameToFrame	0,0459659
FrameToFrame	1208,48	FrameToFrame	57,8426	FrameToFrame	4,92197	FrameToFrame	0,0487181	FrameToFrame	0,0469659
FrameToFrame	1209,67	FrameToFrame	57,501	FrameToFrame	5,00032	FrameToFrame	0,0450255	FrameToFrame	0,0479653
FrameToFrame	1192,49	FrameToFrame	56,9838	FrameToFrame	4,81262	FrameToFrame	0,0469595	FrameToFrame	0,0459691
FrameToFrame	1165,88	FrameToFrame	55,453	FrameToFrame	4,71903	FrameToFrame	0,046981	FrameToFrame	0,0479547
TotalTime	105292,738	TotalTime	5098,1705	TotalTime	425,485191	TotalTime	12,8472529	TotalTime	9,0757843

Medium scenario - Rotation

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	570,462	FrameToFrame	31,7045	FrameToFrame	2,65615	FrameToFrame	2,72213	FrameToFrame	2,38056
FrameToFrame	585,985	FrameToFrame	30,8344	FrameToFrame	2,57885	FrameToFrame	0,046632	FrameToFrame	0,0621992
FrameToFrame	516,547	FrameToFrame	30,1229	FrameToFrame	2,39627	FrameToFrame	0,0530222	FrameToFrame	0,0780206
FrameToFrame	515,922	FrameToFrame	29,6135	FrameToFrame	2,41485	FrameToFrame	0,0468229	FrameToFrame	0,0624336
FrameToFrame	524,595	FrameToFrame	30,1052	FrameToFrame	2,53445	FrameToFrame	0,0468161	FrameToFrame	0,0624792
FrameToFrame	567,046	FrameToFrame	30,4702	FrameToFrame	2,41539	FrameToFrame	0,0624577	FrameToFrame	0,0624153
FrameToFrame	532,469	FrameToFrame	30,4389	FrameToFrame	2,44557	FrameToFrame	0,0468267	FrameToFrame	0,0624496
FrameToFrame	551,873	FrameToFrame	30,1889	FrameToFrame	2,44673	FrameToFrame	0,0467834	FrameToFrame	0,0624602
FrameToFrame	551,689	FrameToFrame	30,8453	FrameToFrame	2,3364	FrameToFrame	0,142925	FrameToFrame	0,0624484
FrameToFrame	536,922	FrameToFrame	30,7984	FrameToFrame	2,45876	FrameToFrame	0,0446484	FrameToFrame	0,0624695
FrameToFrame	543,639	FrameToFrame	31,017	FrameToFrame	2,43986	FrameToFrame	0,046887	FrameToFrame	0,062441
FrameToFrame	557,281	FrameToFrame	31,4547	FrameToFrame	2,47722	FrameToFrame	0,046821	FrameToFrame	0,0780321
FrameToFrame	584,5	FrameToFrame	31,1421	FrameToFrame	2,51929	FrameToFrame	0,0468338	FrameToFrame	0,0624541
FrameToFrame	529,845	FrameToFrame	30,8617	FrameToFrame	2,48644	FrameToFrame	0,0467164	FrameToFrame	0,0624545
FrameToFrame	557,905	FrameToFrame	30,3915	FrameToFrame	2,36653	FrameToFrame	0,0468338	FrameToFrame	0,0624423
FrameToFrame	522,266	FrameToFrame	30,6577	FrameToFrame	2,47676	FrameToFrame	0,0467635	FrameToFrame	0,0624387
FrameToFrame	532,612	FrameToFrame	30,2982	FrameToFrame	2,37655	FrameToFrame	0,0469621	FrameToFrame	0,0624609
FrameToFrame	536,795	FrameToFrame	31,0015	FrameToFrame	2,58719	FrameToFrame	0,0468238	FrameToFrame	0,0623996
FrameToFrame	546,218	FrameToFrame	31,5327	FrameToFrame	2,37593	FrameToFrame	0,0468094	FrameToFrame	0,0624333
FrameToFrame	533,234	FrameToFrame	30,1578	FrameToFrame	2,39061	FrameToFrame	0,0468367	FrameToFrame	0,0624384
FrameToFrame	528,64	FrameToFrame	30,8765	FrameToFrame	2,46882	FrameToFrame	0,0467353	FrameToFrame	0,062424
FrameToFrame	558,063	FrameToFrame	30,6577	FrameToFrame	2,43752	FrameToFrame	0,0468024	FrameToFrame	0,0624307
FrameToFrame	524,157	FrameToFrame	30,5952	FrameToFrame	2,43925	FrameToFrame	0,0468729	FrameToFrame	0,0624384
FrameToFrame	545,655	FrameToFrame	30,5327	FrameToFrame	2,60677	FrameToFrame	0,0467241	FrameToFrame	0,0624811
FrameToFrame	553,593	FrameToFrame	30,3765	FrameToFrame	2,336	FrameToFrame	0,0467982	FrameToFrame	0,0624156
FrameToFrame	546,596	FrameToFrame	29,9545	FrameToFrame	2,40139	FrameToFrame	0,0469271	FrameToFrame	0,0624782
FrameToFrame	540,92	FrameToFrame	29,9703	FrameToFrame	2,29599	FrameToFrame	0,0467058	FrameToFrame	0,0624365
FrameToFrame	547,656	FrameToFrame	30,2357	FrameToFrame	2,36403	FrameToFrame	0,0468222	FrameToFrame	0,0679084
FrameToFrame	532,923	FrameToFrame	29,5328	FrameToFrame	2,34391	FrameToFrame	0,0469563	FrameToFrame	0,0624269
FrameToFrame	547,39	FrameToFrame	29,7983	FrameToFrame	2,43951	FrameToFrame	0,0512589	FrameToFrame	0,0624468
FrameToFrame	520,186	FrameToFrame	29,6109	FrameToFrame	2,31888	FrameToFrame	0,0469197	FrameToFrame	0,062482
FrameToFrame	541,017	FrameToFrame	29,4858	FrameToFrame	2,34521	FrameToFrame	0,0467039	FrameToFrame	0,0624124

FrameToFrame	513,031	FrameToFrame	29,3452	FrameToFrame	2,28567	FrameToFrame	0,0468627	FrameToFrame	0,0624355
FrameToFrame	521,689	FrameToFrame	28,83	FrameToFrame	2,2594	FrameToFrame	0,0467985	FrameToFrame	0,0624551
FrameToFrame	521,998	FrameToFrame	28,7978	FrameToFrame	2,33746	FrameToFrame	0,0467934	FrameToFrame	0,0624442
FrameToFrame	515,925	FrameToFrame	28,5014	FrameToFrame	2,42523	FrameToFrame	0,0468501	FrameToFrame	0,0624301
FrameToFrame	501,092	FrameToFrame	28,9399	FrameToFrame	2,38029	FrameToFrame	0,0467648	FrameToFrame	0,0624615
FrameToFrame	501,499	FrameToFrame	28,6722	FrameToFrame	2,28675	FrameToFrame	0,0468264	FrameToFrame	0,0624346
FrameToFrame	515,141	FrameToFrame	28,564	FrameToFrame	2,28783	FrameToFrame	0,0468184	FrameToFrame	0,0517149
FrameToFrame	501,453	FrameToFrame	28,7357	FrameToFrame	2,29888	FrameToFrame	0,0468049	FrameToFrame	0,0625818
FrameToFrame	502,548	FrameToFrame	28,564	FrameToFrame	2,31264	FrameToFrame	0,0468161	FrameToFrame	0,062288
FrameToFrame	488,66	FrameToFrame	27,8607	FrameToFrame	2,29686	FrameToFrame	0,0468184	FrameToFrame	0,0624413
FrameToFrame	481,637	FrameToFrame	28,5015	FrameToFrame	2,28135	FrameToFrame	0,0468684	FrameToFrame	0,0624541
FrameToFrame	496,832	FrameToFrame	28,1108	FrameToFrame	2,1875	FrameToFrame	0,0467613	FrameToFrame	0,0624477
FrameToFrame	482,887	FrameToFrame	28,0014	FrameToFrame	2,18753	FrameToFrame	0,0468274	FrameToFrame	0,0624516
FrameToFrame	496,187	FrameToFrame	28,2982	FrameToFrame	2,18763	FrameToFrame	0,0468184	FrameToFrame	0,0624468
FrameToFrame	496,234	FrameToFrame	27,8295	FrameToFrame	2,24996	FrameToFrame	0,046845	FrameToFrame	0,0624702
FrameToFrame	512,562	FrameToFrame	28,314	FrameToFrame	2,20706	FrameToFrame	2,30334	FrameToFrame	0,0624112
FrameToFrame	490,768	FrameToFrame	27,8139	FrameToFrame	2,24033	FrameToFrame	0,04565	FrameToFrame	0,0516412
FrameToFrame	499,389	FrameToFrame	27,4231	FrameToFrame	2,10255	FrameToFrame	0,0468232	FrameToFrame	0,0623938
FrameToFrame	507,891	FrameToFrame	27,0014	FrameToFrame	2,19239	FrameToFrame	0,0469332	FrameToFrame	0,0624166
FrameToFrame	513,938	FrameToFrame	27,2514	FrameToFrame	2,19867	FrameToFrame	0,0468126	FrameToFrame	0,0624551
FrameToFrame	487,562	FrameToFrame	26,9075	FrameToFrame	2,10055	FrameToFrame	0,0467068	FrameToFrame	0,0624217
FrameToFrame	509,047	FrameToFrame	26,3919	FrameToFrame	2,06485	FrameToFrame	0,0469425	FrameToFrame	0,0624477
FrameToFrame	502,358	FrameToFrame	27,0169	FrameToFrame	2,17799	FrameToFrame	0,0467292	FrameToFrame	0,0624753
FrameToFrame	494,548	FrameToFrame	26,9544	FrameToFrame	2,13203	FrameToFrame	0,0468267	FrameToFrame	0,0624808
FrameToFrame	481,031	FrameToFrame	26,9544	FrameToFrame	2,05089	FrameToFrame	0,0469294	FrameToFrame	0,0624112
FrameToFrame	467,376	FrameToFrame	26,2982	FrameToFrame	2,08257	FrameToFrame	0,0468299	FrameToFrame	0,0624349
FrameToFrame	446,031	FrameToFrame	26,5483	FrameToFrame	2,08737	FrameToFrame	0,046734	FrameToFrame	0,0624208
FrameToFrame	481,842	FrameToFrame	26,5169	FrameToFrame	2,13256	FrameToFrame	0,0469031	FrameToFrame	0,0624397
FrameToFrame	475,237	FrameToFrame	26,2826	FrameToFrame	2,05062	FrameToFrame	0,0467542	FrameToFrame	0,0624535
FrameToFrame	454,076	FrameToFrame	26,1732	FrameToFrame	2,12858	FrameToFrame	0,0468229	FrameToFrame	0,0624442
FrameToFrame	456,407	FrameToFrame	25,5169	FrameToFrame	2,07845	FrameToFrame	0,0467626	FrameToFrame	0,0624721
FrameToFrame	456,171	FrameToFrame	25,7043	FrameToFrame	2,06262	FrameToFrame	0,0468271	FrameToFrame	0,0624096
FrameToFrame	463,891	FrameToFrame	25,5638	FrameToFrame	2,03121	FrameToFrame	0,0468328	FrameToFrame	0,0624589

FrameToFrame	449,422	FrameToFrame	25,5792	FrameToFrame	1,9845	FrameToFrame	0,0467908	FrameToFrame	0,0624477
FrameToFrame	450,081	FrameToFrame	25,7669	FrameToFrame	2,03131	FrameToFrame	0,0468194	FrameToFrame	0,0624112
FrameToFrame	465,388	FrameToFrame	25,642	FrameToFrame	1,86382	FrameToFrame	0,0469403	FrameToFrame	0,0624458
FrameToFrame	466	FrameToFrame	25,3293	FrameToFrame	2,00799	FrameToFrame	0,0467126	FrameToFrame	0,0624737
FrameToFrame	458,094	FrameToFrame	25,2201	FrameToFrame	1,97107	FrameToFrame	0,0468331	FrameToFrame	0,0624387
FrameToFrame	465,905	FrameToFrame	25,2979	FrameToFrame	1,96492	FrameToFrame	0,0468043	FrameToFrame	0,0624067
FrameToFrame	452,657	FrameToFrame	24,6107	FrameToFrame	1,88129	FrameToFrame	0,0469191	FrameToFrame	0,0624564
FrameToFrame	467,406	FrameToFrame	24,6261	FrameToFrame	1,90628	FrameToFrame	0,0467562	FrameToFrame	0,0624638
FrameToFrame	439,109	FrameToFrame	24,4231	FrameToFrame	2,03995	FrameToFrame	0,0467863	FrameToFrame	0,0624429
FrameToFrame	480,802	FrameToFrame	24,97	FrameToFrame	1,89607	FrameToFrame	0,0468537	FrameToFrame	0,062423
FrameToFrame	463,117	FrameToFrame	24,8762	FrameToFrame	1,92084	FrameToFrame	0,0467876	FrameToFrame	0,062458
FrameToFrame	462,76	FrameToFrame	24,8293	FrameToFrame	2,04694	FrameToFrame	0,0468104	FrameToFrame	0,0624606
FrameToFrame	436,27	FrameToFrame	24,0792	FrameToFrame	1,94106	FrameToFrame	0,0468068	FrameToFrame	0,0624051
FrameToFrame	463,779	FrameToFrame	24,5482	FrameToFrame	1,85978	FrameToFrame	0,0468094	FrameToFrame	0,0624375
FrameToFrame	450,292	FrameToFrame	25,1729	FrameToFrame	2,00541	FrameToFrame	0,0468271	FrameToFrame	0,0624516
FrameToFrame	436,294	FrameToFrame	24,4545	FrameToFrame	1,86341	FrameToFrame	0,0468152	FrameToFrame	0,0624718
FrameToFrame	424,323	FrameToFrame	23,8136	FrameToFrame	1,84687	FrameToFrame	0,046972	FrameToFrame	0,0624487
FrameToFrame	444,728	FrameToFrame	24,3292	FrameToFrame	1,80126	FrameToFrame	0,0466885	FrameToFrame	0,0624073
FrameToFrame	430,068	FrameToFrame	24,017	FrameToFrame	1,86309	FrameToFrame	0,0468174	FrameToFrame	0,062491
FrameToFrame	422,467	FrameToFrame	24,2979	FrameToFrame	1,83092	FrameToFrame	0,0468194	FrameToFrame	0,0624112
FrameToFrame	423,854	FrameToFrame	23,7513	FrameToFrame	1,75213	FrameToFrame	0,0468248	FrameToFrame	0,0624221
FrameToFrame	430,586	FrameToFrame	24,2668	FrameToFrame	1,71953	FrameToFrame	0,0468274	FrameToFrame	0,0624503
FrameToFrame	430,982	FrameToFrame	23,8917	FrameToFrame	1,77082	FrameToFrame	0,0468158	FrameToFrame	0,0624362
FrameToFrame	429,261	FrameToFrame	24,2669	FrameToFrame	1,8786	FrameToFrame	0,0545543	FrameToFrame	0,0624496
FrameToFrame	418,901	FrameToFrame	23,2512	FrameToFrame	1,81432	FrameToFrame	0,0468187	FrameToFrame	0,0624622
FrameToFrame	423,202	FrameToFrame	24,0636	FrameToFrame	1,8129	FrameToFrame	0,0468501	FrameToFrame	0,0624849
FrameToFrame	417,633	FrameToFrame	23,548	FrameToFrame	1,78126	FrameToFrame	1,89378	FrameToFrame	0,0624256
FrameToFrame	422,592	FrameToFrame	24,0637	FrameToFrame	1,81255	FrameToFrame	0,0468392	FrameToFrame	0,0624567
FrameToFrame	410,226	FrameToFrame	23,5949	FrameToFrame	1,81253	FrameToFrame	0,0466972	FrameToFrame	0,0624211
FrameToFrame	432,543	FrameToFrame	23,7668	FrameToFrame	1,82816	FrameToFrame	0,0468254	FrameToFrame	0,0624525
FrameToFrame	417,146	FrameToFrame	23,4855	FrameToFrame	1,87502	FrameToFrame	0,046819	FrameToFrame	0,0624753
FrameToFrame	424,397	FrameToFrame	23,6574	FrameToFrame	1,89067	FrameToFrame	0,0468245	FrameToFrame	0,0624493
FrameToFrame	416,131	FrameToFrame	23,8761	FrameToFrame	1,78131	FrameToFrame	0,0468303	FrameToFrame	0,0624522

FrameToFrame	431,741	FrameToFrame	23,6887	FrameToFrame	1,79688	FrameToFrame	0,0469297	FrameToFrame	0,0624031
FrameToFrame	423,725	FrameToFrame	23,7354	FrameToFrame	1,82809	FrameToFrame	0,0467081	FrameToFrame	0,0624538
FrameToFrame	429,709	FrameToFrame	24,2513	FrameToFrame	1,8125	FrameToFrame	0,0468081	FrameToFrame	0,0624654
FrameToFrame	416,178	FrameToFrame	24,3293	FrameToFrame	1,84389	FrameToFrame	0,046828	FrameToFrame	0,062433
FrameToFrame	435,994	FrameToFrame	24,3293	FrameToFrame	1,84382	FrameToFrame	0,0468158	FrameToFrame	0,0624343
FrameToFrame	430,319	FrameToFrame	24,1263	FrameToFrame	1,76561	FrameToFrame	0,0469435	FrameToFrame	0,0624727
FrameToFrame	443,395	FrameToFrame	24,1887	FrameToFrame	1,9062	FrameToFrame	0,046709	FrameToFrame	0,0625353
FrameToFrame	437,647	FrameToFrame	23,9386	FrameToFrame	1,81264	FrameToFrame	0,0468457	FrameToFrame	0,0623656
FrameToFrame	422,209	FrameToFrame	24,1261	FrameToFrame	1,87503	FrameToFrame	0,0467937	FrameToFrame	0,0624291
FrameToFrame	436,569	FrameToFrame	24,4388	FrameToFrame	1,87505	FrameToFrame	0,0468206	FrameToFrame	0,0624535
FrameToFrame	437,509	FrameToFrame	24,4543	FrameToFrame	1,84375	FrameToFrame	0,0468264	FrameToFrame	0,0624391
FrameToFrame	428,238	FrameToFrame	24,2199	FrameToFrame	1,85943	FrameToFrame	0,0468139	FrameToFrame	0,0624153
FrameToFrame	448,463	FrameToFrame	24,5637	FrameToFrame	1,82809	FrameToFrame	0,0468245	FrameToFrame	0,0624336
FrameToFrame	423,599	FrameToFrame	23,6573	FrameToFrame	1,84383	FrameToFrame	0,0468129	FrameToFrame	0,0624852
FrameToFrame	449,419	FrameToFrame	24,0169	FrameToFrame	1,81245	FrameToFrame	0,0545107	FrameToFrame	0,0670912
FrameToFrame	450,319	FrameToFrame	24,4386	FrameToFrame	1,90636	FrameToFrame	0,0468338	FrameToFrame	0,0624814
FrameToFrame	396,256	FrameToFrame	23,173	FrameToFrame	1,75006	FrameToFrame	0,0468405	FrameToFrame	0,0666072
FrameToFrame	438,809	FrameToFrame	23,47	FrameToFrame	1,81253	FrameToFrame	0,0467969	FrameToFrame	0,0624391
FrameToFrame	422,852	FrameToFrame	23,8918	FrameToFrame	1,78125	FrameToFrame	0,0468149	FrameToFrame	0,0624554
FrameToFrame	429,76	FrameToFrame	23,798	FrameToFrame	1,81254	FrameToFrame	0,0468213	FrameToFrame	0,0518015
FrameToFrame	436,788	FrameToFrame	23,8449	FrameToFrame	1,84499	FrameToFrame	0,0468213	FrameToFrame	0,0605054
FrameToFrame	425,085	FrameToFrame	23,8451	FrameToFrame	1,79942	FrameToFrame	0,0468187	FrameToFrame	0,06244
FrameToFrame	406,755	FrameToFrame	23,298	FrameToFrame	1,78127	FrameToFrame	0,0469646	FrameToFrame	0,0625532
FrameToFrame	434,444	FrameToFrame	23,1729	FrameToFrame	1,81252	FrameToFrame	0,0468136	FrameToFrame	0,0623419
FrameToFrame	410,255	FrameToFrame	23,3919	FrameToFrame	1,81278	FrameToFrame	0,0467097	FrameToFrame	0,0624333
FrameToFrame	431,024	FrameToFrame	23,6729	FrameToFrame	1,84549	FrameToFrame	0,0469399	FrameToFrame	0,0624381
FrameToFrame	423,146	FrameToFrame	23,3606	FrameToFrame	1,79953	FrameToFrame	0,0467235	FrameToFrame	0,0625542
FrameToFrame	417,507	FrameToFrame	23,3136	FrameToFrame	1,82939	FrameToFrame	0,0469355	FrameToFrame	0,0623573
FrameToFrame	417,055	FrameToFrame	23,8136	FrameToFrame	1,78519	FrameToFrame	0,0468017	FrameToFrame	0,0624394
FrameToFrame	446,151	FrameToFrame	23,7043	FrameToFrame	1,87518	FrameToFrame	0,0467244	FrameToFrame	0,0624734
FrameToFrame	426,027	FrameToFrame	23,548	FrameToFrame	1,79847	FrameToFrame	0,0469229	FrameToFrame	0,0624227
FrameToFrame	451,696	FrameToFrame	23,8293	FrameToFrame	1,8308	FrameToFrame	0,0467427	FrameToFrame	0,0624375
FrameToFrame	430,262	FrameToFrame	23,7823	FrameToFrame	1,86197	FrameToFrame	0,0469079	FrameToFrame	0,0624368

FrameToFrame	423,275	FrameToFrame	23,8762	FrameToFrame	1,8518	FrameToFrame	0,06234	FrameToFrame	0,062433
FrameToFrame	437,45	FrameToFrame	23,9701	FrameToFrame	1,79911	FrameToFrame	0,0625619	FrameToFrame	0,0624644
FrameToFrame	416,405	FrameToFrame	23,6261	FrameToFrame	1,82191	FrameToFrame	0,0467302	FrameToFrame	0,0624275
FrameToFrame	452,887	FrameToFrame	23,6885	FrameToFrame	1,75621	FrameToFrame	0,0467953	FrameToFrame	0,0625407
FrameToFrame	424,338	FrameToFrame	23,5949	FrameToFrame	1,86668	FrameToFrame	0,0469159	FrameToFrame	0,0623085
FrameToFrame	438,832	FrameToFrame	23,8449	FrameToFrame	1,80186	FrameToFrame	0,0468405	FrameToFrame	0,0624484
FrameToFrame	419,615	FrameToFrame	23,1887	FrameToFrame	1,75257	FrameToFrame	0,0468309	FrameToFrame	0,0624641
FrameToFrame	425,209	FrameToFrame	23,1105	FrameToFrame	1,67567	FrameToFrame	0,046819	FrameToFrame	0,0624442
FrameToFrame	435,475	FrameToFrame	24,1576	FrameToFrame	1,85238	FrameToFrame	0,0468309	FrameToFrame	0,062448
FrameToFrame	463,195	FrameToFrame	24,6731	FrameToFrame	1,83872	FrameToFrame	0,046702	FrameToFrame	0,0624593
FrameToFrame	459,617	FrameToFrame	25,6886	FrameToFrame	2,03121	FrameToFrame	0,0468319	FrameToFrame	0,0624461
FrameToFrame	459,179	FrameToFrame	25,6888	FrameToFrame	1,93762	FrameToFrame	0,0468129	FrameToFrame	0,0624509
FrameToFrame	443,571	FrameToFrame	25,9232	FrameToFrame	1,92182	FrameToFrame	0,0468245	FrameToFrame	0,0623874
FrameToFrame	472,054	FrameToFrame	26,22	FrameToFrame	2,10943	FrameToFrame	2,05539	FrameToFrame	0,0624509
FrameToFrame	470,227	FrameToFrame	26,8294	FrameToFrame	2,03127	FrameToFrame	0,0468386	FrameToFrame	0,0624792
FrameToFrame	468,243	FrameToFrame	26,6888	FrameToFrame	2,10952	FrameToFrame	0,0467036	FrameToFrame	0,0624233
FrameToFrame	467,743	FrameToFrame	26,9076	FrameToFrame	2,09391	FrameToFrame	0,0468251	FrameToFrame	0,0624253
FrameToFrame	495,715	FrameToFrame	26,8451	FrameToFrame	2,11382	FrameToFrame	0,046821	FrameToFrame	0,0625074
FrameToFrame	468,616	FrameToFrame	26,5794	FrameToFrame	2,10026	FrameToFrame	0,0468133	FrameToFrame	0,0623733
FrameToFrame	501,985	FrameToFrame	27,4545	FrameToFrame	2,14804	FrameToFrame	0,0468341	FrameToFrame	0,0624618
FrameToFrame	479,749	FrameToFrame	27,4076	FrameToFrame	2,11414	FrameToFrame	0,0468145	FrameToFrame	0,062408
FrameToFrame	447,587	FrameToFrame	26,8294	FrameToFrame	2,06683	FrameToFrame	0,046828	FrameToFrame	0,0624759
FrameToFrame	453,568	FrameToFrame	26,5013	FrameToFrame	2,20337	FrameToFrame	0,0468043	FrameToFrame	0,077791
FrameToFrame	441,882	FrameToFrame	25,6732	FrameToFrame	2,07945	FrameToFrame	0,0468296	FrameToFrame	0,0623964
FrameToFrame	486,419	FrameToFrame	26,9856	FrameToFrame	2,09392	FrameToFrame	0,0469319	FrameToFrame	0,0624294
FrameToFrame	469,23	FrameToFrame	25,7669	FrameToFrame	2,07818	FrameToFrame	0,0467135	FrameToFrame	0,0624503
FrameToFrame	469,016	FrameToFrame	26,2045	FrameToFrame	2,1096	FrameToFrame	0,0468238	FrameToFrame	0,0538615
FrameToFrame	443,79	FrameToFrame	25,2511	FrameToFrame	1,93818	FrameToFrame	0,0468197	FrameToFrame	0,0624586
FrameToFrame	455,421	FrameToFrame	26,1575	FrameToFrame	2,1409	FrameToFrame	0,0469297	FrameToFrame	0,0624349
FrameToFrame	451,231	FrameToFrame	25,6108	FrameToFrame	2,05324	FrameToFrame	0,0468322	FrameToFrame	0,0624785
FrameToFrame	431,777	FrameToFrame	24,8449	FrameToFrame	1,99049	FrameToFrame	0,0466943	FrameToFrame	0,0624147
FrameToFrame	477,592	FrameToFrame	24,5637	FrameToFrame	2,02133	FrameToFrame	0,0469268	FrameToFrame	0,0624282
FrameToFrame	489,831	FrameToFrame	25,8763	FrameToFrame	2,08196	FrameToFrame	0,0468216	FrameToFrame	0,0624384

FrameToFrame	448,499	FrameToFrame	25,5325	FrameToFrame	2,00268	FrameToFrame	0,0467417	FrameToFrame	0,0624439
FrameToFrame	470,711	FrameToFrame	25,345	FrameToFrame	2,03588	FrameToFrame	0,0467215	FrameToFrame	0,0779773
FrameToFrame	453,295	FrameToFrame	24,6262	FrameToFrame	2,08178	FrameToFrame	0,046939	FrameToFrame	0,0623913
FrameToFrame	371,551	FrameToFrame	22,2197	FrameToFrame	1,75449	FrameToFrame	1,78263	FrameToFrame	0,0624519
FrameToFrame	304,596	FrameToFrame	19,7823	FrameToFrame	1,50365	FrameToFrame	0,0467052	FrameToFrame	0,0624609
FrameToFrame	301,59	FrameToFrame	20,3761	FrameToFrame	1,61412	FrameToFrame	0,0468101	FrameToFrame	0,062414
FrameToFrame	322,299	FrameToFrame	21,5634	FrameToFrame	1,6446	FrameToFrame	0,0468315	FrameToFrame	0,0624436
FrameToFrame	310,782	FrameToFrame	21,0947	FrameToFrame	1,62924	FrameToFrame	0,0468222	FrameToFrame	0,0624375
FrameToFrame	262,686	FrameToFrame	16,1103	FrameToFrame	1,3013	FrameToFrame	1,26874	FrameToFrame	0,0624878
FrameToFrame	227,26	FrameToFrame	14,4069	FrameToFrame	1,06327	FrameToFrame	0,0467488	FrameToFrame	0,0624118
FrameToFrame	203,151	FrameToFrame	13,7663	FrameToFrame	1,191	FrameToFrame	0,0468668	FrameToFrame	0,0624057
FrameToFrame	210,901	FrameToFrame	14,3601	FrameToFrame	1,12932	FrameToFrame	0,04676	FrameToFrame	0,0624439
FrameToFrame	238,731	FrameToFrame	16,157	FrameToFrame	1,30597	FrameToFrame	0,0468113	FrameToFrame	0,0673167
FrameToFrame	219,418	FrameToFrame	16,2038	FrameToFrame	1,30064	FrameToFrame	0,0467963	FrameToFrame	0,0624647
FrameToFrame	272,623	FrameToFrame	17,1414	FrameToFrame	1,3601	FrameToFrame	1,35986	FrameToFrame	0,0624724
FrameToFrame	266,014	FrameToFrame	18,6102	FrameToFrame	1,38692	FrameToFrame	1,41649	FrameToFrame	0,0624753
FrameToFrame	338,533	FrameToFrame	22,8293	FrameToFrame	1,65681	FrameToFrame	1,80295	FrameToFrame	0,0623954
FrameToFrame	471,29	FrameToFrame	26,6418	FrameToFrame	2,14583	FrameToFrame	0,0468216	FrameToFrame	0,0624403
FrameToFrame	560,313	FrameToFrame	31,2984	FrameToFrame	2,44933	FrameToFrame	0,046829	FrameToFrame	0,0624426
FrameToFrame	594,937	FrameToFrame	32,533	FrameToFrame	2,67879	FrameToFrame	0,0468469	FrameToFrame	0,0624394
FrameToFrame	643,404	FrameToFrame	36,7517	FrameToFrame	3,0625	FrameToFrame	0,0467825	FrameToFrame	0,0624205
FrameToFrame	671,581	FrameToFrame	35,1424	FrameToFrame	2,90643	FrameToFrame	3,18522	FrameToFrame	0,0624448
FrameToFrame	545,095	FrameToFrame	31,6109	FrameToFrame	2,48446	FrameToFrame	0,0467199	FrameToFrame	0,0624811
FrameToFrame	511,225	FrameToFrame	30,1423	FrameToFrame	2,37519	FrameToFrame	0,0468271	FrameToFrame	0,0624153
FrameToFrame	460,211	FrameToFrame	27,6106	FrameToFrame	2,24996	FrameToFrame	0,0467854	FrameToFrame	0,0624593
FrameToFrame	529,215	FrameToFrame	30,2359	FrameToFrame	2,35943	FrameToFrame	0,0468434	FrameToFrame	0,0625186
FrameToFrame	525,09	FrameToFrame	31,6422	FrameToFrame	2,59897	FrameToFrame	0,046804	FrameToFrame	0,0780334
FrameToFrame	568,763	FrameToFrame	31,314	FrameToFrame	2,43878	FrameToFrame	0,0468283	FrameToFrame	0,0624275
FrameToFrame	591,108	FrameToFrame	33,6579	FrameToFrame	2,64343	FrameToFrame	0,046794	FrameToFrame	0,0624201
FrameToFrame	519,153	FrameToFrame	31,8298	FrameToFrame	2,55731	FrameToFrame	0,0467937	FrameToFrame	0,062465
FrameToFrame	611,577	FrameToFrame	33,1266	FrameToFrame	2,54715	FrameToFrame	0,046802	FrameToFrame	0,0624535
FrameToFrame	454,117	FrameToFrame	27,8452	FrameToFrame	2,11427	FrameToFrame	2,24492	FrameToFrame	0,0624464
FrameToFrame	556,528	FrameToFrame	32,8922	FrameToFrame	2,71881	FrameToFrame	0,0469027	FrameToFrame	0,0780706

FrameToFrame	515,467		FrameToFrame	28,767		FrameToFrame	2,26603		FrameToFrame	0,0467247		FrameToFrame	0,0511915
FrameToFrame	506,601		FrameToFrame	29,439		FrameToFrame	2,43775		FrameToFrame	0,0468024		FrameToFrame	0,0625882
TotalTime	92435,899		TotalTime	5209,0649		TotalTime	409,28852		TotalTime	31,0027173		TotalTime	14,8027376

Medium scenario - Translation

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	368,803	FrameToFrame	24,3991	FrameToFrame	2,02151	FrameToFrame	2,21574	FrameToFrame	1,72996
FrameToFrame	374,659	FrameToFrame	23,8916	FrameToFrame	1,97577	FrameToFrame	0,0556012	FrameToFrame	0,0466612
FrameToFrame	360,716	FrameToFrame	23,7192	FrameToFrame	1,90923	FrameToFrame	0,0468755	FrameToFrame	0,0467366
FrameToFrame	387,996	FrameToFrame	24,2019	FrameToFrame	1,91144	FrameToFrame	0,0467276	FrameToFrame	0,0469194
FrameToFrame	381,478	FrameToFrame	24,2199	FrameToFrame	2,00905	FrameToFrame	0,0469146	FrameToFrame	0,0468149
FrameToFrame	373,493	FrameToFrame	24,3668	FrameToFrame	1,89794	FrameToFrame	0,046828	FrameToFrame	0,0468267
FrameToFrame	373,597	FrameToFrame	24,1861	FrameToFrame	1,99222	FrameToFrame	0,0468158	FrameToFrame	0,0468338
FrameToFrame	367,474	FrameToFrame	23,934	FrameToFrame	1,90818	FrameToFrame	0,0467122	FrameToFrame	0,0467238
FrameToFrame	380,825	FrameToFrame	24,5382	FrameToFrame	1,94135	FrameToFrame	0,0468306	FrameToFrame	0,0468954
FrameToFrame	360,19	FrameToFrame	23,9005	FrameToFrame	1,92578	FrameToFrame	0,04682	FrameToFrame	0,0468392
FrameToFrame	367,507	FrameToFrame	23,6758	FrameToFrame	1,94945	FrameToFrame	0,0468075	FrameToFrame	0,0468142
FrameToFrame	368,143	FrameToFrame	23,5793	FrameToFrame	1,92199	FrameToFrame	0,0468117	FrameToFrame	0,0468466
FrameToFrame	360,62	FrameToFrame	23,6886	FrameToFrame	2,04674	FrameToFrame	0,0468245	FrameToFrame	0,0468351
FrameToFrame	374,513	FrameToFrame	24,1105	FrameToFrame	1,92196	FrameToFrame	0,0468364	FrameToFrame	0,0467956
FrameToFrame	380,291	FrameToFrame	24,548	FrameToFrame	1,96862	FrameToFrame	0,0467417	FrameToFrame	0,046845
FrameToFrame	380,54	FrameToFrame	24,5324	FrameToFrame	1,8751	FrameToFrame	0,0469313	FrameToFrame	0,0467963
FrameToFrame	374,701	FrameToFrame	23,5636	FrameToFrame	1,8594	FrameToFrame	0,0468328	FrameToFrame	0,046889
FrameToFrame	387,476	FrameToFrame	24,5325	FrameToFrame	1,90622	FrameToFrame	0,0468245	FrameToFrame	0,0467629
FrameToFrame	395,629	FrameToFrame	24,3448	FrameToFrame	1,90627	FrameToFrame	0,0467549	FrameToFrame	0,0467979
FrameToFrame	402,363	FrameToFrame	24,5794	FrameToFrame	1,90619	FrameToFrame	0,0467898	FrameToFrame	0,0468315
FrameToFrame	409,446	FrameToFrame	24,7667	FrameToFrame	1,93759	FrameToFrame	0,0468899	FrameToFrame	0,0468248
FrameToFrame	373,024	FrameToFrame	24,47	FrameToFrame	1,89064	FrameToFrame	0,0467109	FrameToFrame	0,0468197
FrameToFrame	386,894	FrameToFrame	24,4542	FrameToFrame	1,93758	FrameToFrame	0,0468296	FrameToFrame	0,0467975
FrameToFrame	387,518	FrameToFrame	23,7198	FrameToFrame	1,90623	FrameToFrame	0,0467854	FrameToFrame	0,0469335
FrameToFrame	380,723	FrameToFrame	24,5637	FrameToFrame	1,90627	FrameToFrame	0,046853	FrameToFrame	0,0467093
FrameToFrame	388,51	FrameToFrame	23,7043	FrameToFrame	1,92173	FrameToFrame	0,0469053	FrameToFrame	0,0468229
FrameToFrame	373,082	FrameToFrame	23,8762	FrameToFrame	1,93764	FrameToFrame	0,0468126	FrameToFrame	0,0468017
FrameToFrame	373,111	FrameToFrame	24,7356	FrameToFrame	1,92192	FrameToFrame	0,0467542	FrameToFrame	0,0468229
FrameToFrame	354,332	FrameToFrame	23,6105	FrameToFrame	1,89066	FrameToFrame	0,0467963	FrameToFrame	0,0467712
FrameToFrame	367,667	FrameToFrame	23,9386	FrameToFrame	1,82817	FrameToFrame	0,0518807	FrameToFrame	0,0624599
FrameToFrame	376,121	FrameToFrame	23,8293	FrameToFrame	1,84377	FrameToFrame	0,0467959	FrameToFrame	0,0468113
FrameToFrame	388,42	FrameToFrame	24,4387	FrameToFrame	1,92196	FrameToFrame	0,0468431	FrameToFrame	0,0468008
FrameToFrame	368,529	FrameToFrame	23,4386	FrameToFrame	1,8125	FrameToFrame	1,91616	FrameToFrame	0,0470163
FrameToFrame	381,501	FrameToFrame	24,1418	FrameToFrame	1,89063	FrameToFrame	0,0467706	FrameToFrame	0,0466256
FrameToFrame	368,652	FrameToFrame	23,7199	FrameToFrame	1,81257	FrameToFrame	0,0469759	FrameToFrame	0,046804
FrameToFrame	375,62	FrameToFrame	24,0795	FrameToFrame	1,9531	FrameToFrame	0,0466603	FrameToFrame	0,0468008
FrameToFrame	380,263	FrameToFrame	24,2824	FrameToFrame	1,90635	FrameToFrame	0,0468129	FrameToFrame	0,0468178
FrameToFrame	381,592	FrameToFrame	24,2512	FrameToFrame	1,95315	FrameToFrame	0,0468097	FrameToFrame	0,0468113
FrameToFrame	353,883	FrameToFrame	23,9542	FrameToFrame	2,0157	FrameToFrame	0,046888	FrameToFrame	0,0468335
FrameToFrame	361,096	FrameToFrame	23,6574	FrameToFrame	1,87502	FrameToFrame	0,0468367	FrameToFrame	0,0468492
FrameToFrame	361,33	FrameToFrame	23,673	FrameToFrame	1,91154	FrameToFrame	0,046837	FrameToFrame	0,0468091

FrameToFrame	368,194	FrameToFrame	23,6574	FrameToFrame	1,79705	FrameToFrame	0,0496225	FrameToFrame	0,046795
FrameToFrame	357,248	FrameToFrame	22,7669	FrameToFrame	1,81481	FrameToFrame	0,0467703	FrameToFrame	0,0468004
FrameToFrame	353,264	FrameToFrame	23,8448	FrameToFrame	1,90712	FrameToFrame	1,92395	FrameToFrame	0,046845
FrameToFrame	347,143	FrameToFrame	23,4699	FrameToFrame	1,89385	FrameToFrame	0,0467523	FrameToFrame	0,0468065
FrameToFrame	374,982	FrameToFrame	24,0481	FrameToFrame	1,95319	FrameToFrame	0,046786	FrameToFrame	0,0468232
FrameToFrame	389,466	FrameToFrame	24,1262	FrameToFrame	1,87781	FrameToFrame	0,0469306	FrameToFrame	0,0468489
FrameToFrame	381,52	FrameToFrame	24,3918	FrameToFrame	1,96197	FrameToFrame	0,0468678	FrameToFrame	0,04677
FrameToFrame	389,289	FrameToFrame	23,6576	FrameToFrame	1,92843	FrameToFrame	0,0466558	FrameToFrame	0,0468206
FrameToFrame	382,237	FrameToFrame	23,9855	FrameToFrame	1,78974	FrameToFrame	0,0468145	FrameToFrame	0,0467873
FrameToFrame	382,377	FrameToFrame	24,3137	FrameToFrame	1,89168	FrameToFrame	0,0468072	FrameToFrame	0,0468222
FrameToFrame	353,546	FrameToFrame	24,0169	FrameToFrame	1,94643	FrameToFrame	0,0468222	FrameToFrame	0,0468126
FrameToFrame	347,028	FrameToFrame	23,501	FrameToFrame	1,94111	FrameToFrame	0,0468178	FrameToFrame	0,0468354
FrameToFrame	360,291	FrameToFrame	23,8293	FrameToFrame	1,97085	FrameToFrame	0,0468149	FrameToFrame	0,0469406
FrameToFrame	361,851	FrameToFrame	23,345	FrameToFrame	1,87932	FrameToFrame	1,81727	FrameToFrame	0,0629086
FrameToFrame	362,494	FrameToFrame	23,4229	FrameToFrame	1,80533	FrameToFrame	0,0467174	FrameToFrame	0,046912
FrameToFrame	362,404	FrameToFrame	23,5168	FrameToFrame	1,85055	FrameToFrame	0,0469502	FrameToFrame	0,0467077
FrameToFrame	343,027	FrameToFrame	22,3761	FrameToFrame	1,8949	FrameToFrame	0,0468271	FrameToFrame	0,0468267
FrameToFrame	349,013	FrameToFrame	23,0637	FrameToFrame	1,81481	FrameToFrame	0,0467975	FrameToFrame	0,0468165
FrameToFrame	364,347	FrameToFrame	23,126	FrameToFrame	1,86589	FrameToFrame	0,0467209	FrameToFrame	0,0468065
FrameToFrame	364,794	FrameToFrame	23,0636	FrameToFrame	1,89314	FrameToFrame	0,0468357	FrameToFrame	0,0468457
FrameToFrame	356,433	FrameToFrame	23,298	FrameToFrame	1,78626	FrameToFrame	0,0469114	FrameToFrame	0,046795
FrameToFrame	357,94	FrameToFrame	22,8293	FrameToFrame	1,83367	FrameToFrame	0,0468383	FrameToFrame	0,0469531
FrameToFrame	351,453	FrameToFrame	22,4387	FrameToFrame	1,78494	FrameToFrame	0,0468562	FrameToFrame	0,0466879
FrameToFrame	336,399	FrameToFrame	22,798	FrameToFrame	1,73673	FrameToFrame	0,0467953	FrameToFrame	0,0468357
FrameToFrame	350,269	FrameToFrame	23,1104	FrameToFrame	1,84951	FrameToFrame	0,046702	FrameToFrame	0,0468075
FrameToFrame	335,245	FrameToFrame	22,8294	FrameToFrame	1,78535	FrameToFrame	0,0468373	FrameToFrame	0,0468335
FrameToFrame	337,818	FrameToFrame	22,6261	FrameToFrame	1,7294	FrameToFrame	0,0469165	FrameToFrame	0,0468158
FrameToFrame	336,304	FrameToFrame	22,7979	FrameToFrame	1,75394	FrameToFrame	0,0468354	FrameToFrame	0,0467985
FrameToFrame	330,439	FrameToFrame	22,3449	FrameToFrame	1,74218	FrameToFrame	0,046718	FrameToFrame	0,0468681
FrameToFrame	344,374	FrameToFrame	22,2353	FrameToFrame	1,75723	FrameToFrame	0,046906	FrameToFrame	0,0467645
FrameToFrame	364,101	FrameToFrame	22,8605	FrameToFrame	1,82647	FrameToFrame	0,046829	FrameToFrame	0,0624894
FrameToFrame	336,662	FrameToFrame	22,7824	FrameToFrame	1,74491	FrameToFrame	0,0467225	FrameToFrame	0,0467805
FrameToFrame	330,763	FrameToFrame	22,3291	FrameToFrame	1,76566	FrameToFrame	0,046811	FrameToFrame	0,0468383
FrameToFrame	335,664	FrameToFrame	22,9074	FrameToFrame	1,81257	FrameToFrame	0,0469223	FrameToFrame	0,0624378
FrameToFrame	344,64	FrameToFrame	22,673	FrameToFrame	1,81248	FrameToFrame	0,0466917	FrameToFrame	0,0467847
FrameToFrame	336,409	FrameToFrame	22,7354	FrameToFrame	1,76577	FrameToFrame	0,0469143	FrameToFrame	0,0468328
FrameToFrame	337,023	FrameToFrame	22,7825	FrameToFrame	1,7813	FrameToFrame	0,0468665	FrameToFrame	0,0624596
FrameToFrame	323,137	FrameToFrame	22,6261	FrameToFrame	1,81855	FrameToFrame	0,0468075	FrameToFrame	0,0468078
FrameToFrame	342,189	FrameToFrame	23,2512	FrameToFrame	1,83036	FrameToFrame	0,0468235	FrameToFrame	0,0468367
FrameToFrame	343,441	FrameToFrame	23,0011	FrameToFrame	1,84424	FrameToFrame	0,0466773	FrameToFrame	0,0467889
FrameToFrame	358,34	FrameToFrame	23,0635	FrameToFrame	1,76843	FrameToFrame	0,0469608	FrameToFrame	0,0624413
FrameToFrame	356,775	FrameToFrame	23,2354	FrameToFrame	1,84802	FrameToFrame	0,0467321	FrameToFrame	0,0515385

FrameToFrame	350,269	FrameToFrame	23,1106	FrameToFrame	1,78824	FrameToFrame	0,0468091	FrameToFrame	0,0468165
FrameToFrame	356,356	FrameToFrame	23,3137	FrameToFrame	1,85154	FrameToFrame	1,89702	FrameToFrame	0,0468489
FrameToFrame	370,335	FrameToFrame	23,3496	FrameToFrame	1,88236	FrameToFrame	0,0367671	FrameToFrame	0,0624176
FrameToFrame	363,192	FrameToFrame	23,016	FrameToFrame	1,77032	FrameToFrame	0,0456628	FrameToFrame	0,0468511
FrameToFrame	337,58	FrameToFrame	21,8762	FrameToFrame	1,75419	FrameToFrame	0,0468187	FrameToFrame	0,0624019
FrameToFrame	357,595	FrameToFrame	23,2511	FrameToFrame	1,78175	FrameToFrame	0,0468088	FrameToFrame	0,0469239
FrameToFrame	363,913	FrameToFrame	23,4278	FrameToFrame	1,74343	FrameToFrame	0,0467039	FrameToFrame	0,0467068
FrameToFrame	343,85	FrameToFrame	22,6916	FrameToFrame	1,72603	FrameToFrame	0,0469207	FrameToFrame	0,0624724
FrameToFrame	351,646	FrameToFrame	22,7966	FrameToFrame	1,75475	FrameToFrame	1,81366	FrameToFrame	0,0468011
FrameToFrame	373,091	FrameToFrame	23,0123	FrameToFrame	1,78342	FrameToFrame	0,0467982	FrameToFrame	0,0624583
FrameToFrame	366,267	FrameToFrame	22,9543	FrameToFrame	1,82838	FrameToFrame	0,0468303	FrameToFrame	0,0467921
FrameToFrame	372,729	FrameToFrame	23,1946	FrameToFrame	1,82842	FrameToFrame	0,0467212	FrameToFrame	0,0468129
FrameToFrame	338,029	FrameToFrame	22,5503	FrameToFrame	1,76595	FrameToFrame	0,0469127	FrameToFrame	0,0468319
FrameToFrame	303,151	FrameToFrame	22,6667	FrameToFrame	1,7344	FrameToFrame	1,76413	FrameToFrame	0,046802
FrameToFrame	352,939	FrameToFrame	22,6869	FrameToFrame	1,76579	FrameToFrame	0,0467174	FrameToFrame	0,0624541
FrameToFrame	325,997	FrameToFrame	21,8917	FrameToFrame	1,65808	FrameToFrame	0,0469079	FrameToFrame	0,0468511
FrameToFrame	348,303	FrameToFrame	21,9386	FrameToFrame	1,65843	FrameToFrame	0,0467202	FrameToFrame	0,0468136
FrameToFrame	340,738	FrameToFrame	21,8761	FrameToFrame	1,67484	FrameToFrame	0,046819	FrameToFrame	0,0468591
FrameToFrame	354,592	FrameToFrame	21,9855	FrameToFrame	1,73648	FrameToFrame	0,0468117	FrameToFrame	0,0468598
FrameToFrame	332,745	FrameToFrame	21,7978	FrameToFrame	1,69351	FrameToFrame	1,7747	FrameToFrame	0,0623342
FrameToFrame	340,103	FrameToFrame	22,1731	FrameToFrame	1,72224	FrameToFrame	0,046692	FrameToFrame	0,0468463
FrameToFrame	332,598	FrameToFrame	21,7978	FrameToFrame	1,78422	FrameToFrame	0,0469034	FrameToFrame	0,046812
FrameToFrame	339,899	FrameToFrame	21,8137	FrameToFrame	1,7357	FrameToFrame	0,046736	FrameToFrame	0,046794
FrameToFrame	347,509	FrameToFrame	21,9385	FrameToFrame	1,6566	FrameToFrame	0,0469444	FrameToFrame	0,0468271
FrameToFrame	346,908	FrameToFrame	22,0167	FrameToFrame	1,83075	FrameToFrame	0,0466731	FrameToFrame	0,0624599
FrameToFrame	346,829	FrameToFrame	22,3135	FrameToFrame	1,74002	FrameToFrame	0,0468258	FrameToFrame	0,0468197
FrameToFrame	347,475	FrameToFrame	21,8918	FrameToFrame	1,76592	FrameToFrame	0,0468768	FrameToFrame	0,0468008
FrameToFrame	360,871	FrameToFrame	22,1573	FrameToFrame	1,79691	FrameToFrame	0,0468867	FrameToFrame	0,046819
FrameToFrame	333,129	FrameToFrame	21,5009	FrameToFrame	1,71881	FrameToFrame	1,81866	FrameToFrame	0,046804
FrameToFrame	333,565	FrameToFrame	21,5167	FrameToFrame	1,73439	FrameToFrame	0,046642	FrameToFrame	0,0624638
FrameToFrame	347,346	FrameToFrame	21,923	FrameToFrame	1,9688	FrameToFrame	0,0468261	FrameToFrame	0,0468142
FrameToFrame	346,913	FrameToFrame	22,0322	FrameToFrame	1,76573	FrameToFrame	0,0468149	FrameToFrame	0,0468261
FrameToFrame	325,528	FrameToFrame	21,5635	FrameToFrame	1,68743	FrameToFrame	0,0468296	FrameToFrame	0,0467805
FrameToFrame	340,758	FrameToFrame	21,5168	FrameToFrame	1,62513	FrameToFrame	0,0468242	FrameToFrame	0,0468155
FrameToFrame	347,539	FrameToFrame	22,1103	FrameToFrame	1,65626	FrameToFrame	0,0468248	FrameToFrame	0,04682
FrameToFrame	319,885	FrameToFrame	21,0479	FrameToFrame	1,70316	FrameToFrame	0,0468893	FrameToFrame	0,0468296
FrameToFrame	326,321	FrameToFrame	21,3606	FrameToFrame	1,71878	FrameToFrame	0,0467334	FrameToFrame	0,0468149
FrameToFrame	333,503	FrameToFrame	21,7041	FrameToFrame	1,71879	FrameToFrame	0,0529796	FrameToFrame	0,0468389
FrameToFrame	332,994	FrameToFrame	21,6729	FrameToFrame	1,71878	FrameToFrame	0,0466956	FrameToFrame	0,0468476
FrameToFrame	333,643	FrameToFrame	21,5635	FrameToFrame	1,6719	FrameToFrame	1,70639	FrameToFrame	0,0526277
FrameToFrame	333,508	FrameToFrame	21,6104	FrameToFrame	1,76566	FrameToFrame	0,0468139	FrameToFrame	0,0468424
FrameToFrame	341,09	FrameToFrame	21,4231	FrameToFrame	1,68757	FrameToFrame	0,0467347	FrameToFrame	0,0467943

FrameToFrame	334,078	FrameToFrame	21,2509	FrameToFrame	1,73438	FrameToFrame	0,0467834	FrameToFrame	0,0468777
FrameToFrame	334,428	FrameToFrame	21,7043	FrameToFrame	1,67186	FrameToFrame	0,0468501	FrameToFrame	0,0467635
FrameToFrame	327,986	FrameToFrame	21,2823	FrameToFrame	1,625	FrameToFrame	0,0469306	FrameToFrame	0,0468142
FrameToFrame	326,719	FrameToFrame	21,5479	FrameToFrame	1,85947	FrameToFrame	0,0468065	FrameToFrame	0,0469425
FrameToFrame	334,278	FrameToFrame	21,8448	FrameToFrame	1,65627	FrameToFrame	0,0467106	FrameToFrame	0,0467049
FrameToFrame	340,24	FrameToFrame	21,7666	FrameToFrame	1,76564	FrameToFrame	0,0468206	FrameToFrame	0,0468104
FrameToFrame	327,926	FrameToFrame	21,3136	FrameToFrame	1,71882	FrameToFrame	0,0467956	FrameToFrame	0,0467921
FrameToFrame	328,061	FrameToFrame	21,2198	FrameToFrame	1,67175	FrameToFrame	0,046838	FrameToFrame	0,0468402
FrameToFrame	327,77	FrameToFrame	21,5168	FrameToFrame	1,67201	FrameToFrame	0,0467985	FrameToFrame	0,0468036
FrameToFrame	340,84	FrameToFrame	21,9386	FrameToFrame	1,75001	FrameToFrame	0,0468402	FrameToFrame	0,0468216
FrameToFrame	341,29	FrameToFrame	21,7042	FrameToFrame	1,68754	FrameToFrame	0,0468046	FrameToFrame	0,0601491
FrameToFrame	314,549	FrameToFrame	21,1415	FrameToFrame	1,76565	FrameToFrame	0,0468043	FrameToFrame	0,046804
FrameToFrame	348,966	FrameToFrame	21,7042	FrameToFrame	1,67175	FrameToFrame	0,046922	FrameToFrame	0,0468367
FrameToFrame	341,761	FrameToFrame	21,6262	FrameToFrame	1,65639	FrameToFrame	0,0468197	FrameToFrame	0,0468027
FrameToFrame	325,412	FrameToFrame	22,0478	FrameToFrame	1,65619	FrameToFrame	0,0623342	FrameToFrame	0,0467982
FrameToFrame	320,355	FrameToFrame	21,173	FrameToFrame	1,70322	FrameToFrame	0,0468187	FrameToFrame	0,0468309
FrameToFrame	312,986	FrameToFrame	21,2198	FrameToFrame	1,62504	FrameToFrame	0,0468101	FrameToFrame	0,0468001
FrameToFrame	320,451	FrameToFrame	21,1259	FrameToFrame	1,65616	FrameToFrame	0,0468309	FrameToFrame	0,0468235
FrameToFrame	319,626	FrameToFrame	21,3917	FrameToFrame	1,64062	FrameToFrame	0,046912	FrameToFrame	0,0468088
FrameToFrame	299,593	FrameToFrame	20,876	FrameToFrame	1,61617	FrameToFrame	1,59656	FrameToFrame	0,0468495
FrameToFrame	352,832	FrameToFrame	22,5794	FrameToFrame	1,75919	FrameToFrame	0,0466901	FrameToFrame	0,0467802
FrameToFrame	330,202	FrameToFrame	22,1103	FrameToFrame	1,69363	FrameToFrame	0,0469355	FrameToFrame	0,0468078
FrameToFrame	346,206	FrameToFrame	22,6887	FrameToFrame	1,73601	FrameToFrame	0,0467225	FrameToFrame	0,0468338
FrameToFrame	323,592	FrameToFrame	20,9072	FrameToFrame	1,56266	FrameToFrame	0,0467754	FrameToFrame	0,0468607
FrameToFrame	311,37	FrameToFrame	19,6415	FrameToFrame	1,54434	FrameToFrame	0,0468501	FrameToFrame	0,0467805
FrameToFrame	303,851	FrameToFrame	19,7821	FrameToFrame	1,50771	FrameToFrame	0,0469136	FrameToFrame	0,0468178
FrameToFrame	289,61	FrameToFrame	19,4072	FrameToFrame	1,51565	FrameToFrame	0,0468405	FrameToFrame	0,0468354
TotalTime	53548,9	TotalTime	3478,99	TotalTime	283,647	TotalTime	37,4988	TotalTime	22,33

Medium scenario - Scaling

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	303,219	FrameToFrame	18,7233	FrameToFrame	1,53272	FrameToFrame	1,49411	FrameToFrame	1,40866
FrameToFrame	274,032	FrameToFrame	17,813	FrameToFrame	1,45258	FrameToFrame	0,0456416	FrameToFrame	0,0466292
FrameToFrame	295,572	FrameToFrame	18,2812	FrameToFrame	1,49983	FrameToFrame	0,0469727	FrameToFrame	0,0624217
FrameToFrame	287,981	FrameToFrame	18,1401	FrameToFrame	1,45358	FrameToFrame	0,0459858	FrameToFrame	0,0468415
FrameToFrame	266,772	FrameToFrame	17,7972	FrameToFrame	1,43589	FrameToFrame	0,0479733	FrameToFrame	0,0468149
FrameToFrame	308,675	FrameToFrame	18,9066	FrameToFrame	1,48456	FrameToFrame	0,0469483	FrameToFrame	0,0468261
FrameToFrame	266,157	FrameToFrame	17,8591	FrameToFrame	1,43861	FrameToFrame	0,0469675	FrameToFrame	0,0467866
FrameToFrame	309,726	FrameToFrame	18,5304	FrameToFrame	1,49891	FrameToFrame	0,0459787	FrameToFrame	0,0468665
FrameToFrame	309,805	FrameToFrame	18,5474	FrameToFrame	1,42123	FrameToFrame	0,0475464	FrameToFrame	0,0467757
FrameToFrame	308,204	FrameToFrame	18,9989	FrameToFrame	1,54703	FrameToFrame	0,0460964	FrameToFrame	0,044004
FrameToFrame	309,347	FrameToFrame	18,6102	FrameToFrame	1,4848	FrameToFrame	0,0478745	FrameToFrame	0,0504936
FrameToFrame	309,395	FrameToFrame	18,7178	FrameToFrame	1,5322	FrameToFrame	0,0459765	FrameToFrame	0,0468235
FrameToFrame	288,573	FrameToFrame	18,2031	FrameToFrame	1,45309	FrameToFrame	0,0460137	FrameToFrame	0,046906
FrameToFrame	302,603	FrameToFrame	18,5006	FrameToFrame	1,40519	FrameToFrame	0,046981	FrameToFrame	0,0467087
FrameToFrame	296,008	FrameToFrame	18,1864	FrameToFrame	1,43742	FrameToFrame	0,0479538	FrameToFrame	0,0468222
FrameToFrame	287,897	FrameToFrame	17,8594	FrameToFrame	1,51653	FrameToFrame	0,0459572	FrameToFrame	0,0468248
FrameToFrame	302,071	FrameToFrame	18,6111	FrameToFrame	1,45277	FrameToFrame	0,0479781	FrameToFrame	0,0468171
FrameToFrame	302,299	FrameToFrame	18,5303	FrameToFrame	1,46816	FrameToFrame	0,0459896	FrameToFrame	0,0468444
FrameToFrame	287,282	FrameToFrame	18,0613	FrameToFrame	1,51465	FrameToFrame	0,047966	FrameToFrame	0,0467905
FrameToFrame	315,347	FrameToFrame	18,7509	FrameToFrame	1,61012	FrameToFrame	0,0459646	FrameToFrame	0,046819
FrameToFrame	308,537	FrameToFrame	18,828	FrameToFrame	1,53073	FrameToFrame	0,0459951	FrameToFrame	0,0467988
FrameToFrame	307,972	FrameToFrame	19,2352	FrameToFrame	1,53208	FrameToFrame	0,0469589	FrameToFrame	0,0468168
FrameToFrame	272,187	FrameToFrame	19,1562	FrameToFrame	1,57785	FrameToFrame	0,0479842	FrameToFrame	0,0468396
FrameToFrame	306,076	FrameToFrame	19,6714	FrameToFrame	1,54568	FrameToFrame	0,0459813	FrameToFrame	0,0468072
FrameToFrame	313,241	FrameToFrame	19,4517	FrameToFrame	1,6098	FrameToFrame	0,0479643	FrameToFrame	0,0468367
FrameToFrame	334,01	FrameToFrame	20,0465	FrameToFrame	1,67194	FrameToFrame	0,0459659	FrameToFrame	0,0468213
FrameToFrame	319,153	FrameToFrame	19,9217	FrameToFrame	1,57815	FrameToFrame	0,0459601	FrameToFrame	0,0468627
FrameToFrame	339,657	FrameToFrame	20,3593	FrameToFrame	1,62495	FrameToFrame	0,0479432	FrameToFrame	0,0468033
FrameToFrame	323,512	FrameToFrame	20,2656	FrameToFrame	1,75046	FrameToFrame	0,0459659	FrameToFrame	0,0468036
FrameToFrame	330,41	FrameToFrame	20,781	FrameToFrame	1,70246	FrameToFrame	0,047974	FrameToFrame	0,0468941
FrameToFrame	329,964	FrameToFrame	20,5481	FrameToFrame	1,6873	FrameToFrame	0,0459608	FrameToFrame	0,0467587
FrameToFrame	337,69	FrameToFrame	21,0146	FrameToFrame	1,67285	FrameToFrame	0,0479672	FrameToFrame	0,0468476
FrameToFrame	343,618	FrameToFrame	21,1414	FrameToFrame	1,78131	FrameToFrame	0,0459659	FrameToFrame	0,0517659
FrameToFrame	342,17	FrameToFrame	21,0467	FrameToFrame	1,79647	FrameToFrame	0,0469682	FrameToFrame	0,0468428
FrameToFrame	342,194	FrameToFrame	21,2037	FrameToFrame	1,79585	FrameToFrame	0,0459665	FrameToFrame	0,0467748
FrameToFrame	377,678	FrameToFrame	21,7952	FrameToFrame	1,73561	FrameToFrame	0,047965	FrameToFrame	0,0468104
FrameToFrame	346,155	FrameToFrame	21,8602	FrameToFrame	1,78035	FrameToFrame	0,0459662	FrameToFrame	0,0468408
FrameToFrame	359,364	FrameToFrame	22,7353	FrameToFrame	1,87403	FrameToFrame	0,0479794	FrameToFrame	0,0468101
FrameToFrame	373,288	FrameToFrame	23,0294	FrameToFrame	1,87533	FrameToFrame	0,0459566	FrameToFrame	0,0513435
FrameToFrame	351,066	FrameToFrame	22,6267	FrameToFrame	1,89179	FrameToFrame	0,0459624	FrameToFrame	0,0468351
FrameToFrame	364,687	FrameToFrame	23,0614	FrameToFrame	1,84315	FrameToFrame	0,0479666	FrameToFrame	0,0468543

FrameToFrame	370,991	FrameToFrame	23,1711	FrameToFrame	1,81222	FrameToFrame	0,0459633	FrameToFrame	0,0468056
FrameToFrame	397,104	FrameToFrame	24,1103	FrameToFrame	1,95365	FrameToFrame	0,0479666	FrameToFrame	0,046811
FrameToFrame	404,302	FrameToFrame	24,0613	FrameToFrame	1,87456	FrameToFrame	0,045971	FrameToFrame	0,0468149
FrameToFrame	396,48	FrameToFrame	24,0168	FrameToFrame	1,85991	FrameToFrame	0,0459608	FrameToFrame	0,0468161
FrameToFrame	409,153	FrameToFrame	25,1085	FrameToFrame	2,04694	FrameToFrame	0,0479993	FrameToFrame	0,0468396
FrameToFrame	409,089	FrameToFrame	24,8908	FrameToFrame	1,96821	FrameToFrame	0,0469646	FrameToFrame	0,0468351
FrameToFrame	402,831	FrameToFrame	25,3278	FrameToFrame	1,96764	FrameToFrame	0,0459614	FrameToFrame	0,0468376
FrameToFrame	401,902	FrameToFrame	26,5633	FrameToFrame	2,0314	FrameToFrame	0,0480381	FrameToFrame	0,0468206
FrameToFrame	421,708	FrameToFrame	25,6245	FrameToFrame	2,047	FrameToFrame	0,04592	FrameToFrame	0,046735
FrameToFrame	420,038	FrameToFrame	25,9842	FrameToFrame	2,09378	FrameToFrame	0,04699	FrameToFrame	0,0468428
FrameToFrame	424,265	FrameToFrame	26,3449	FrameToFrame	2,15725	FrameToFrame	0,0469794	FrameToFrame	0,0468078
FrameToFrame	444,805	FrameToFrame	27,0764	FrameToFrame	2,24877	FrameToFrame	0,0469865	FrameToFrame	0,0468514
FrameToFrame	435,266	FrameToFrame	27,2036	FrameToFrame	2,21866	FrameToFrame	0,0460021	FrameToFrame	0,0468081
FrameToFrame	440,541	FrameToFrame	27,5779	FrameToFrame	2,32927	FrameToFrame	0,0479695	FrameToFrame	0,0468572
FrameToFrame	439,395	FrameToFrame	27,8753	FrameToFrame	2,40611	FrameToFrame	0,0459653	FrameToFrame	0,0467789
FrameToFrame	451,788	FrameToFrame	28,5477	FrameToFrame	2,40631	FrameToFrame	0,0479695	FrameToFrame	0,0468197
FrameToFrame	461,57	FrameToFrame	29,7495	FrameToFrame	2,49879	FrameToFrame	0,0459588	FrameToFrame	0,0468527
FrameToFrame	461,101	FrameToFrame	29,4838	FrameToFrame	2,48549	FrameToFrame	0,0460002	FrameToFrame	0,0467972
FrameToFrame	466,742	FrameToFrame	30,3125	FrameToFrame	2,51523	FrameToFrame	0,0469736	FrameToFrame	0,0468418
FrameToFrame	471,071	FrameToFrame	30,4693	FrameToFrame	2,5779	FrameToFrame	0,0479627	FrameToFrame	0,0468107
FrameToFrame	499,496	FrameToFrame	30,9993	FrameToFrame	2,62424	FrameToFrame	0,0459813	FrameToFrame	0,0468623
FrameToFrame	536,6	FrameToFrame	32,891	FrameToFrame	2,73395	FrameToFrame	0,0479477	FrameToFrame	0,0468203
FrameToFrame	533,827	FrameToFrame	33,6239	FrameToFrame	2,75058	FrameToFrame	0,0459762	FrameToFrame	0,0467857
FrameToFrame	526,472	FrameToFrame	33,5328	FrameToFrame	2,87466	FrameToFrame	0,0459893	FrameToFrame	0,0468261
FrameToFrame	530,369	FrameToFrame	34,4686	FrameToFrame	2,84455	FrameToFrame	0,0469672	FrameToFrame	0,0467783
FrameToFrame	563,909	FrameToFrame	35,0452	FrameToFrame	2,96777	FrameToFrame	0,0479589	FrameToFrame	0,0468447
FrameToFrame	581,899	FrameToFrame	35,9535	FrameToFrame	3,07921	FrameToFrame	0,0459944	FrameToFrame	0,0467934
FrameToFrame	604,814	FrameToFrame	37,2963	FrameToFrame	3,07668	FrameToFrame	0,0479701	FrameToFrame	0,0468546
FrameToFrame	623,485	FrameToFrame	38,3592	FrameToFrame	3,21951	FrameToFrame	0,0469772	FrameToFrame	0,0468062
FrameToFrame	612,467	FrameToFrame	38,7972	FrameToFrame	3,26438	FrameToFrame	0,0449819	FrameToFrame	0,0467328
FrameToFrame	643,189	FrameToFrame	40,5011	FrameToFrame	3,37653	FrameToFrame	0,0479868	FrameToFrame	0,046837
FrameToFrame	659,783	FrameToFrame	41,7014	FrameToFrame	3,49878	FrameToFrame	0,0459697	FrameToFrame	0,0468309
FrameToFrame	658,768	FrameToFrame	41,7042	FrameToFrame	3,54761	FrameToFrame	0,0479605	FrameToFrame	0,0468251
FrameToFrame	642,85	FrameToFrame	41,7807	FrameToFrame	3,56264	FrameToFrame	0,0469781	FrameToFrame	0,0468107
FrameToFrame	651,091	FrameToFrame	41,2809	FrameToFrame	3,53075	FrameToFrame	0,0459774	FrameToFrame	0,046871
FrameToFrame	655,174	FrameToFrame	42,4228	FrameToFrame	3,65597	FrameToFrame	0,0459935	FrameToFrame	0,0467648
FrameToFrame	674,427	FrameToFrame	43,281	FrameToFrame	3,67279	FrameToFrame	0,0469804	FrameToFrame	0,046821
FrameToFrame	663,643	FrameToFrame	43,5632	FrameToFrame	3,64062	FrameToFrame	0,0479512	FrameToFrame	0,0468392
FrameToFrame	707,535	FrameToFrame	45,0457	FrameToFrame	3,71727	FrameToFrame	0,0459848	FrameToFrame	0,0468001
FrameToFrame	741,833	FrameToFrame	49,2509	FrameToFrame	4,18901	FrameToFrame	0,0479605	FrameToFrame	0,0468158
FrameToFrame	846,434	FrameToFrame	55,4671	FrameToFrame	4,70155	FrameToFrame	0,0469422	FrameToFrame	0,0467988
FrameToFrame	870,905	FrameToFrame	57,9234	FrameToFrame	4,68857	FrameToFrame	0,0469679	FrameToFrame	0,046862

FrameToFrame	886,032	FrameToFrame	59,1238	FrameToFrame	4,99838	FrameToFrame	0,0459787	FrameToFrame	0,0467818
FrameToFrame	691,387	FrameToFrame	44,8276	FrameToFrame	3,64162	FrameToFrame	0,0459768	FrameToFrame	0,0468396
FrameToFrame	661,503	FrameToFrame	43,4228	FrameToFrame	3,56218	FrameToFrame	0,047974	FrameToFrame	0,0467889
FrameToFrame	682,801	FrameToFrame	44,7957	FrameToFrame	3,70334	FrameToFrame	0,0469614	FrameToFrame	0,0468174
FrameToFrame	730,785	FrameToFrame	46,892	FrameToFrame	3,82697	FrameToFrame	0,0469605	FrameToFrame	0,0468335
FrameToFrame	746,647	FrameToFrame	48,6078	FrameToFrame	4,01742	FrameToFrame	0,0459807	FrameToFrame	0,0468181
FrameToFrame	758,507	FrameToFrame	49,2359	FrameToFrame	4,03116	FrameToFrame	0,0469826	FrameToFrame	0,0468655
FrameToFrame	709,583	FrameToFrame	47,0307	FrameToFrame	3,84293	FrameToFrame	0,0459797	FrameToFrame	0,0467834
FrameToFrame	695,348	FrameToFrame	45,3908	FrameToFrame	3,39093	FrameToFrame	0,0479801	FrameToFrame	0,0468428
FrameToFrame	1047,08	FrameToFrame	69,4685	FrameToFrame	5,82707	FrameToFrame	0,0469906	FrameToFrame	0,0468078
FrameToFrame	1130,13	FrameToFrame	73,5934	FrameToFrame	6,0472	FrameToFrame	0,0469749	FrameToFrame	0,0468299
FrameToFrame	1397,73	FrameToFrame	88,9837	FrameToFrame	7,68764	FrameToFrame	0,0609618	FrameToFrame	0,0624827
FrameToFrame	1470,53	FrameToFrame	99,5322	FrameToFrame	8,5927	FrameToFrame	0,0629914	FrameToFrame	0,0467947
FrameToFrame	1588,33	FrameToFrame	105,86	FrameToFrame	9,4064	FrameToFrame	9,26616	FrameToFrame	0,0467924
FrameToFrame	1724,74	FrameToFrame	119,343	FrameToFrame	10,1247	FrameToFrame	0,0469611	FrameToFrame	0,0468232
FrameToFrame	1673,27	FrameToFrame	113,625	FrameToFrame	9,75036	FrameToFrame	0,0469839	FrameToFrame	0,0469412
FrameToFrame	1723,29	FrameToFrame	118,11	FrameToFrame	10,2035	FrameToFrame	0,045973	FrameToFrame	0,0467116
FrameToFrame	1836,4	FrameToFrame	128,296	FrameToFrame	11,1085	FrameToFrame	0,046955	FrameToFrame	0,0468142
FrameToFrame	1884,38	FrameToFrame	132,265	FrameToFrame	11,4862	FrameToFrame	0,0469691	FrameToFrame	0,0468155
TotalTime	57551,179	TotalTime	3712,2004	TotalTime	309,46392	TotalTime	15,4717886	TotalTime	6,1787929

Complex scenario - Rotation

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	315,738	FrameToFrame	12,6901	FrameToFrame	1,88703	FrameToFrame	1,64631	FrameToFrame	0,875905
FrameToFrame	307,502	FrameToFrame	12,5617	FrameToFrame	1,44234	FrameToFrame	0,0466908	FrameToFrame	0,0621979
FrameToFrame	307,898	FrameToFrame	12,7803	FrameToFrame	1,4268	FrameToFrame	0,104804	FrameToFrame	0,0685515
FrameToFrame	279,5	FrameToFrame	12,3601	FrameToFrame	1,44539	FrameToFrame	0,051366	FrameToFrame	0,0624542
FrameToFrame	315,201	FrameToFrame	12,2346	FrameToFrame	1,43751	FrameToFrame	0,0466975	FrameToFrame	0,078003
FrameToFrame	308,994	FrameToFrame	12,235	FrameToFrame	1,4456	FrameToFrame	0,0468235	FrameToFrame	0,0624413
FrameToFrame	300,607	FrameToFrame	12,1089	FrameToFrame	1,41595	FrameToFrame	0,0468428	FrameToFrame	0,0624615
FrameToFrame	313,122	FrameToFrame	12,4064	FrameToFrame	1,45759	FrameToFrame	0,0467709	FrameToFrame	0,0624173
FrameToFrame	293,384	FrameToFrame	12,4988	FrameToFrame	1,36612	FrameToFrame	0,0490064	FrameToFrame	0,0624567
FrameToFrame	322,359	FrameToFrame	12,6415	FrameToFrame	1,34794	FrameToFrame	0,0468213	FrameToFrame	0,0624214
FrameToFrame	321,502	FrameToFrame	12,3578	FrameToFrame	1,44217	FrameToFrame	0,0443764	FrameToFrame	0,0624282
FrameToFrame	286,436	FrameToFrame	12,8591	FrameToFrame	1,44516	FrameToFrame	0,0503769	FrameToFrame	0,0624433
FrameToFrame	278,745	FrameToFrame	12,7362	FrameToFrame	1,51816	FrameToFrame	0,046845	FrameToFrame	0,0631482
FrameToFrame	277,804	FrameToFrame	12,8889	FrameToFrame	1,48435	FrameToFrame	0,0468174	FrameToFrame	0,0624593
FrameToFrame	299,465	FrameToFrame	12,8447	FrameToFrame	1,46259	FrameToFrame	0,0467924	FrameToFrame	0,0624192
FrameToFrame	320,997	FrameToFrame	12,9058	FrameToFrame	1,37509	FrameToFrame	0,0468203	FrameToFrame	0,0624173
FrameToFrame	266,22	FrameToFrame	12,6558	FrameToFrame	1,34848	FrameToFrame	0,0468405	FrameToFrame	0,0624461
FrameToFrame	265,758	FrameToFrame	13,0624	FrameToFrame	1,36386	FrameToFrame	0,0467915	FrameToFrame	0,0624275
FrameToFrame	294,52	FrameToFrame	12,9848	FrameToFrame	1,35938	FrameToFrame	0,046812	FrameToFrame	0,0624526
FrameToFrame	300,376	FrameToFrame	12,6569	FrameToFrame	1,42742	FrameToFrame	0,0468405	FrameToFrame	0,0624221
FrameToFrame	308,445	FrameToFrame	12,7652	FrameToFrame	1,40286	FrameToFrame	0,0468447	FrameToFrame	0,0624651
FrameToFrame	307,603	FrameToFrame	12,9839	FrameToFrame	1,33346	FrameToFrame	0,0467604	FrameToFrame	0,0567061
FrameToFrame	306,284	FrameToFrame	12,8281	FrameToFrame	1,5047	FrameToFrame	0,0468117	FrameToFrame	0,0502088
FrameToFrame	313,45	FrameToFrame	12,8276	FrameToFrame	1,32819	FrameToFrame	0,0468492	FrameToFrame	0,056421
FrameToFrame	299,709	FrameToFrame	12,9535	FrameToFrame	1,52023	FrameToFrame	0,0467389	FrameToFrame	0,062313
FrameToFrame	313,252	FrameToFrame	13,4057	FrameToFrame	1,5079	FrameToFrame	0,0468335	FrameToFrame	0,0625568
FrameToFrame	278,255	FrameToFrame	13,0316	FrameToFrame	1,43743	FrameToFrame	0,0467944	FrameToFrame	0,0623477
FrameToFrame	326,49	FrameToFrame	12,6558	FrameToFrame	1,56678	FrameToFrame	0,0468444	FrameToFrame	0,0625372
FrameToFrame	297,291	FrameToFrame	12,8904	FrameToFrame	1,5269	FrameToFrame	0,0467568	FrameToFrame	0,0624683
FrameToFrame	298,61	FrameToFrame	12,8903	FrameToFrame	1,45653	FrameToFrame	0,046913	FrameToFrame	0,0624384
FrameToFrame	325,458	FrameToFrame	12,9542	FrameToFrame	1,50581	FrameToFrame	0,0467212	FrameToFrame	0,0623605
FrameToFrame	346,666	FrameToFrame	13,7183	FrameToFrame	1,53809	FrameToFrame	0,0468123	FrameToFrame	0,0781483
FrameToFrame	332,571	FrameToFrame	12,984	FrameToFrame	1,59804	FrameToFrame	0,0468319	FrameToFrame	0,0624686
FrameToFrame	324,766	FrameToFrame	13,2817	FrameToFrame	1,50934	FrameToFrame	0,0468133	FrameToFrame	0,0623195
FrameToFrame	338,996	FrameToFrame	13,4673	FrameToFrame	1,4959	FrameToFrame	0,0468566	FrameToFrame	0,0624949
FrameToFrame	330,853	FrameToFrame	13,7186	FrameToFrame	1,53289	FrameToFrame	0,0467648	FrameToFrame	0,0623807
FrameToFrame	308,332	FrameToFrame	13,6267	FrameToFrame	1,52387	FrameToFrame	0,0468274	FrameToFrame	0,0624211
FrameToFrame	350,046	FrameToFrame	13,5925	FrameToFrame	1,5185	FrameToFrame	0,0468357	FrameToFrame	0,0625433
FrameToFrame	330,632	FrameToFrame	13,2816	FrameToFrame	1,62281	FrameToFrame	0,0468142	FrameToFrame	0,0623438
FrameToFrame	349,524	FrameToFrame	13,4842	FrameToFrame	1,60695	FrameToFrame	0,0468168	FrameToFrame	0,0697815
FrameToFrame	351,187	FrameToFrame	13,8446	FrameToFrame	1,51881	FrameToFrame	0,0467629	FrameToFrame	0,0623923

FrameToFrame	322,369	FrameToFrame	13,967	FrameToFrame	1,54692	FrameToFrame	0,0469294	FrameToFrame	0,062457
FrameToFrame	307,719	FrameToFrame	13,8607	FrameToFrame	1,56277	FrameToFrame	0,0468274	FrameToFrame	0,0624131
FrameToFrame	400,155	FrameToFrame	14,0468	FrameToFrame	1,55656	FrameToFrame	0,0468293	FrameToFrame	0,0624359
FrameToFrame	373,663	FrameToFrame	13,5312	FrameToFrame	1,53658	FrameToFrame	0,046804	FrameToFrame	0,0624372
FrameToFrame	330,577	FrameToFrame	13,2973	FrameToFrame	1,55514	FrameToFrame	0,046829	FrameToFrame	0,0624458
FrameToFrame	340,341	FrameToFrame	13,4515	FrameToFrame	1,72684	FrameToFrame	0,0468245	FrameToFrame	0,078172
FrameToFrame	309,986	FrameToFrame	13,8608	FrameToFrame	1,56367	FrameToFrame	0,0467055	FrameToFrame	0,0623493
FrameToFrame	338,749	FrameToFrame	13,5608	FrameToFrame	1,43032	FrameToFrame	0,0469435	FrameToFrame	0,0624224
FrameToFrame	331,969	FrameToFrame	13,86	FrameToFrame	1,51981	FrameToFrame	0,04671	FrameToFrame	0,0624795
FrameToFrame	337,57	FrameToFrame	13,9525	FrameToFrame	1,56833	FrameToFrame	0,0468418	FrameToFrame	0,0624112
FrameToFrame	351,861	FrameToFrame	13,8599	FrameToFrame	1,62431	FrameToFrame	0,0621572	FrameToFrame	0,0624907
FrameToFrame	336,494	FrameToFrame	13,9214	FrameToFrame	1,54686	FrameToFrame	0,0423273	FrameToFrame	0,0625161
FrameToFrame	366,55	FrameToFrame	13,7967	FrameToFrame	1,63251	FrameToFrame	0,0542339	FrameToFrame	0,0623669
FrameToFrame	317,424	FrameToFrame	13,0312	FrameToFrame	1,55951	FrameToFrame	0,0468072	FrameToFrame	0,0625052
FrameToFrame	303,323	FrameToFrame	13,5007	FrameToFrame	1,48681	FrameToFrame	0,0468094	FrameToFrame	0,0623316
FrameToFrame	323,381	FrameToFrame	13,7965	FrameToFrame	1,5418	FrameToFrame	0,0468037	FrameToFrame	0,0624433
FrameToFrame	344,724	FrameToFrame	13,5147	FrameToFrame	1,50511	FrameToFrame	0,0468453	FrameToFrame	0,0624404
FrameToFrame	366,457	FrameToFrame	13,6564	FrameToFrame	1,49987	FrameToFrame	0,0468162	FrameToFrame	0,0624811
FrameToFrame	331,934	FrameToFrame	13,2508	FrameToFrame	1,4945	FrameToFrame	0,0469441	FrameToFrame	0,0625462
FrameToFrame	326,22	FrameToFrame	12,9059	FrameToFrame	1,5521	FrameToFrame	0,0467321	FrameToFrame	0,0623252
FrameToFrame	303,542	FrameToFrame	13,5478	FrameToFrame	1,54082	FrameToFrame	0,0467883	FrameToFrame	0,0624352
FrameToFrame	330,925	FrameToFrame	13,4982	FrameToFrame	1,55443	FrameToFrame	0,0467931	FrameToFrame	0,0624811
FrameToFrame	358,222	FrameToFrame	13,625	FrameToFrame	1,49283	FrameToFrame	0,0468261	FrameToFrame	0,0623961
FrameToFrame	336,267	FrameToFrame	13,7506	FrameToFrame	1,57588	FrameToFrame	0,0468229	FrameToFrame	0,06244
FrameToFrame	343,426	FrameToFrame	13,0936	FrameToFrame	1,53125	FrameToFrame	0,0468203	FrameToFrame	0,0671743
FrameToFrame	321,248	FrameToFrame	13,5148	FrameToFrame	1,62143	FrameToFrame	0,0468591	FrameToFrame	0,0624968
FrameToFrame	341,143	FrameToFrame	14,1411	FrameToFrame	1,55759	FrameToFrame	0,0467976	FrameToFrame	0,0624346
FrameToFrame	320,818	FrameToFrame	13,7036	FrameToFrame	1,53144	FrameToFrame	0,0467876	FrameToFrame	0,0623615
FrameToFrame	348,545	FrameToFrame	13,905	FrameToFrame	1,59124	FrameToFrame	0,0468232	FrameToFrame	0,0624307
FrameToFrame	389,57	FrameToFrame	13,8766	FrameToFrame	1,64694	FrameToFrame	0,0468232	FrameToFrame	0,0624282
FrameToFrame	389,267	FrameToFrame	13,8585	FrameToFrame	1,69229	FrameToFrame	1,73444	FrameToFrame	0,0625901
FrameToFrame	359,271	FrameToFrame	13,7023	FrameToFrame	1,63079	FrameToFrame	0,0469188	FrameToFrame	0,0624044
FrameToFrame	366,36	FrameToFrame	13,5625	FrameToFrame	1,5752	FrameToFrame	0,0468181	FrameToFrame	0,0623721
FrameToFrame	364,128	FrameToFrame	13,8439	FrameToFrame	1,44474	FrameToFrame	0,0467174	FrameToFrame	0,062434
FrameToFrame	335,41	FrameToFrame	13,8127	FrameToFrame	1,64788	FrameToFrame	0,046811	FrameToFrame	0,0624509
FrameToFrame	335,725	FrameToFrame	13,2963	FrameToFrame	1,5416	FrameToFrame	0,0468396	FrameToFrame	0,0624295
FrameToFrame	370,714	FrameToFrame	13,0156	FrameToFrame	1,57688	FrameToFrame	0,046777	FrameToFrame	0,0624336
FrameToFrame	335,598	FrameToFrame	13,6109	FrameToFrame	1,55031	FrameToFrame	0,046837	FrameToFrame	0,0624433
FrameToFrame	343,499	FrameToFrame	13,5919	FrameToFrame	1,44938	FrameToFrame	0,0469233	FrameToFrame	0,0624513
FrameToFrame	356,383	FrameToFrame	13,4854	FrameToFrame	1,51783	FrameToFrame	0,0467078	FrameToFrame	0,0624134
FrameToFrame	321,247	FrameToFrame	13,4537	FrameToFrame	1,52374	FrameToFrame	0,0467995	FrameToFrame	0,0501745
FrameToFrame	356,665	FrameToFrame	13,1236	FrameToFrame	1,57411	FrameToFrame	0,0468203	FrameToFrame	0,0636344

FrameToFrame	328,192	FrameToFrame	13,5476	FrameToFrame	1,61094	FrameToFrame	0,0468069	FrameToFrame	0,0624474
FrameToFrame	349,61	FrameToFrame	13,7494	FrameToFrame	1,53953	FrameToFrame	0,046813	FrameToFrame	0,0625433
FrameToFrame	335,5	FrameToFrame	13,4369	FrameToFrame	1,5375	FrameToFrame	0,046829	FrameToFrame	0,0623332
FrameToFrame	342,799	FrameToFrame	13,47	FrameToFrame	1,42588	FrameToFrame	0,0468457	FrameToFrame	0,0625501
FrameToFrame	350,252	FrameToFrame	13,8901	FrameToFrame	1,41811	FrameToFrame	0,0469092	FrameToFrame	0,0623255
FrameToFrame	364,678	FrameToFrame	13,2977	FrameToFrame	1,54998	FrameToFrame	0,0467013	FrameToFrame	0,0624545
FrameToFrame	343,245	FrameToFrame	13,8266	FrameToFrame	1,53068	FrameToFrame	0,0468056	FrameToFrame	0,0624526
FrameToFrame	322,396	FrameToFrame	14,048	FrameToFrame	1,6017	FrameToFrame	0,0468165	FrameToFrame	0,0780492
FrameToFrame	350,195	FrameToFrame	13,6868	FrameToFrame	1,50736	FrameToFrame	0,0468264	FrameToFrame	0,0624356
FrameToFrame	328,717	FrameToFrame	13,8593	FrameToFrame	1,54127	FrameToFrame	0,0468274	FrameToFrame	0,0624352
FrameToFrame	357,676	FrameToFrame	14,4224	FrameToFrame	1,60247	FrameToFrame	0,0469557	FrameToFrame	0,0625074
FrameToFrame	314,566	FrameToFrame	13,9844	FrameToFrame	1,67515	FrameToFrame	0,0467138	FrameToFrame	0,0624115
FrameToFrame	335,952	FrameToFrame	13,968	FrameToFrame	1,53802	FrameToFrame	0,0448896	FrameToFrame	0,0624035
FrameToFrame	363,052	FrameToFrame	13,5156	FrameToFrame	1,58652	FrameToFrame	0,0564165	FrameToFrame	0,0780527
FrameToFrame	341,031	FrameToFrame	14,2654	FrameToFrame	1,59017	FrameToFrame	0,0467437	FrameToFrame	0,0625699
FrameToFrame	312,938	FrameToFrame	14,171	FrameToFrame	1,64678	FrameToFrame	0,0467857	FrameToFrame	0,0623458
FrameToFrame	299,314	FrameToFrame	14,9695	FrameToFrame	1,6326	FrameToFrame	0,0467055	FrameToFrame	0,0624099
FrameToFrame	332,874	FrameToFrame	14,3753	FrameToFrame	1,69608	FrameToFrame	0,0614965	FrameToFrame	0,062595
FrameToFrame	327,675	FrameToFrame	14,0003	FrameToFrame	1,54964	FrameToFrame	0,0466757	FrameToFrame	0,0623255
FrameToFrame	355,25	FrameToFrame	14,0313	FrameToFrame	1,5901	FrameToFrame	0,0469287	FrameToFrame	0,0625302
FrameToFrame	354,175	FrameToFrame	14,0301	FrameToFrame	1,65486	FrameToFrame	1,70461	FrameToFrame	0,0656489
FrameToFrame	346,789	FrameToFrame	13,8446	FrameToFrame	1,64936	FrameToFrame	0,0468274	FrameToFrame	0,062433
FrameToFrame	351,468	FrameToFrame	14,2022	FrameToFrame	1,60818	FrameToFrame	0,046921	FrameToFrame	0,0624789
FrameToFrame	364,519	FrameToFrame	14,938	FrameToFrame	1,6977	FrameToFrame	0,0468101	FrameToFrame	0,0623823
FrameToFrame	406,49	FrameToFrame	14,7028	FrameToFrame	1,65465	FrameToFrame	0,0467078	FrameToFrame	0,0625828
FrameToFrame	363,409	FrameToFrame	15,3593	FrameToFrame	1,74333	FrameToFrame	0,0468405	FrameToFrame	0,0624384
FrameToFrame	376,91	FrameToFrame	14,9382	FrameToFrame	1,6718	FrameToFrame	0,0468053	FrameToFrame	0,06233
FrameToFrame	349,737	FrameToFrame	14,5617	FrameToFrame	1,67197	FrameToFrame	0,0468033	FrameToFrame	0,0624506
FrameToFrame	313,471	FrameToFrame	15,9065	FrameToFrame	1,69505	FrameToFrame	0,0467947	FrameToFrame	0,0624439
FrameToFrame	352,524	FrameToFrame	14,7329	FrameToFrame	1,73101	FrameToFrame	0,0468162	FrameToFrame	0,0780816
FrameToFrame	373,415	FrameToFrame	14,4062	FrameToFrame	1,67189	FrameToFrame	0,046855	FrameToFrame	0,062534
FrameToFrame	349,711	FrameToFrame	15,0633	FrameToFrame	1,70559	FrameToFrame	0,0469008	FrameToFrame	0,0624272
FrameToFrame	375,661	FrameToFrame	15,3597	FrameToFrame	1,6644	FrameToFrame	0,0467158	FrameToFrame	0,0623894
FrameToFrame	340,562	FrameToFrame	15,531	FrameToFrame	1,72115	FrameToFrame	0,0468027	FrameToFrame	0,0623974
FrameToFrame	339,548	FrameToFrame	14,7959	FrameToFrame	1,60547	FrameToFrame	0,0508785	FrameToFrame	0,0624978
FrameToFrame	339,299	FrameToFrame	15,2047	FrameToFrame	1,61913	FrameToFrame	0,0468235	FrameToFrame	0,078003
FrameToFrame	367,378	FrameToFrame	15,0935	FrameToFrame	1,71707	FrameToFrame	0,0468053	FrameToFrame	0,0624173
FrameToFrame	355,88	FrameToFrame	14,7958	FrameToFrame	1,62784	FrameToFrame	1,72488	FrameToFrame	0,0624394
FrameToFrame	362,112	FrameToFrame	14,8291	FrameToFrame	1,63443	FrameToFrame	0,0467308	FrameToFrame	0,062586
FrameToFrame	362,527	FrameToFrame	14,8592	FrameToFrame	1,6814	FrameToFrame	0,046829	FrameToFrame	0,0624211
FrameToFrame	362,42	FrameToFrame	14,7027	FrameToFrame	1,62112	FrameToFrame	0,0467812	FrameToFrame	0,0624673
FrameToFrame	397,883	FrameToFrame	15,0004	FrameToFrame	1,66617	FrameToFrame	0,0468537	FrameToFrame	0,0623377

FrameToFrame	389,598	FrameToFrame	15,0151	FrameToFrame	1,73328	FrameToFrame	0,046896	FrameToFrame	0,0625484
FrameToFrame	368,597	FrameToFrame	15,2332	FrameToFrame	1,73759	FrameToFrame	0,0467248	FrameToFrame	0,0624221
FrameToFrame	372,8	FrameToFrame	16,5311	FrameToFrame	1,83971	FrameToFrame	0,0467918	FrameToFrame	0,0624519
FrameToFrame	398,38	FrameToFrame	16,5315	FrameToFrame	1,78119	FrameToFrame	0,0468072	FrameToFrame	0,0624064
FrameToFrame	398,396	FrameToFrame	15,5002	FrameToFrame	1,9138	FrameToFrame	0,0468184	FrameToFrame	0,0527381
FrameToFrame	405,691	FrameToFrame	14,8119	FrameToFrame	1,643	FrameToFrame	0,0468171	FrameToFrame	0,0624519
FrameToFrame	432,868	FrameToFrame	14,9383	FrameToFrame	1,59895	FrameToFrame	0,046922	FrameToFrame	0,0625228
FrameToFrame	417,8	FrameToFrame	15,3902	FrameToFrame	1,61691	FrameToFrame	0,0467074	FrameToFrame	0,0623974
FrameToFrame	406,63	FrameToFrame	15,6566	FrameToFrame	1,6536	FrameToFrame	0,046836	FrameToFrame	0,06251
FrameToFrame	419,849	FrameToFrame	15,391	FrameToFrame	1,62722	FrameToFrame	0,0467886	FrameToFrame	0,0624336
FrameToFrame	411,943	FrameToFrame	15,2019	FrameToFrame	1,58205	FrameToFrame	0,0468194	FrameToFrame	0,0624436
FrameToFrame	412,444	FrameToFrame	15,719	FrameToFrame	1,77142	FrameToFrame	0,0468633	FrameToFrame	0,0623544
FrameToFrame	447,431	FrameToFrame	15,4222	FrameToFrame	1,81293	FrameToFrame	0,0467706	FrameToFrame	0,062424
FrameToFrame	377,673	FrameToFrame	15,2186	FrameToFrame	1,67313	FrameToFrame	0,0468777	FrameToFrame	0,0624359
FrameToFrame	433,866	FrameToFrame	14,9993	FrameToFrame	1,62052	FrameToFrame	0,0468729	FrameToFrame	0,0625651
FrameToFrame	391,897	FrameToFrame	14,9859	FrameToFrame	1,60418	FrameToFrame	0,0467029	FrameToFrame	0,0623284
FrameToFrame	392,158	FrameToFrame	15,0148	FrameToFrame	1,74164	FrameToFrame	0,0468283	FrameToFrame	0,062424
FrameToFrame	351,643	FrameToFrame	14,9376	FrameToFrame	1,65108	FrameToFrame	0,0468707	FrameToFrame	0,062551
FrameToFrame	385,363	FrameToFrame	14,9531	FrameToFrame	1,76983	FrameToFrame	0,0467334	FrameToFrame	0,0623445
FrameToFrame	400,755	FrameToFrame	14,9057	FrameToFrame	1,64394	FrameToFrame	0,0468239	FrameToFrame	0,0625398
FrameToFrame	405,867	FrameToFrame	15,2823	FrameToFrame	1,66404	FrameToFrame	0,0469329	FrameToFrame	0,0623753
FrameToFrame	405,94	FrameToFrame	15,1402	FrameToFrame	1,74214	FrameToFrame	0,0468248	FrameToFrame	0,0624131
FrameToFrame	414,884	FrameToFrame	14,9526	FrameToFrame	1,74739	FrameToFrame	0,0468543	FrameToFrame	0,0624474
FrameToFrame	409,549	FrameToFrame	14,7345	FrameToFrame	1,65814	FrameToFrame	0,0466802	FrameToFrame	0,0624429
FrameToFrame	410,789	FrameToFrame	15,1882	FrameToFrame	1,6298	FrameToFrame	0,0468255	FrameToFrame	0,0625715
FrameToFrame	382,363	FrameToFrame	15,3271	FrameToFrame	1,61019	FrameToFrame	0,0468155	FrameToFrame	0,0623018
FrameToFrame	409,41	FrameToFrame	15,3276	FrameToFrame	1,66247	FrameToFrame	1,80295	FrameToFrame	0,0624288
FrameToFrame	423,99	FrameToFrame	15,2964	FrameToFrame	1,67231	FrameToFrame	0,0468117	FrameToFrame	0,0624198
FrameToFrame	362,151	FrameToFrame	15,0166	FrameToFrame	1,69697	FrameToFrame	0,0467315	FrameToFrame	0,0623826
FrameToFrame	403,18	FrameToFrame	15,7645	FrameToFrame	1,75103	FrameToFrame	0,0468986	FrameToFrame	0,0624506
FrameToFrame	404,418	FrameToFrame	15,61	FrameToFrame	1,72834	FrameToFrame	0,0467081	FrameToFrame	0,0624471
FrameToFrame	413,232	FrameToFrame	15,6242	FrameToFrame	1,723	FrameToFrame	0,046922	FrameToFrame	0,0625549
FrameToFrame	433,672	FrameToFrame	15,5793	FrameToFrame	1,74426	FrameToFrame	0,0467129	FrameToFrame	0,0624125
FrameToFrame	397,443	FrameToFrame	15,6864	FrameToFrame	1,75626	FrameToFrame	0,0469204	FrameToFrame	0,062365
FrameToFrame	405,815	FrameToFrame	15,8131	FrameToFrame	1,6451	FrameToFrame	0,0467427	FrameToFrame	0,0624718
FrameToFrame	348,84	FrameToFrame	15,5628	FrameToFrame	1,65386	FrameToFrame	0,0467992	FrameToFrame	0,06251
FrameToFrame	392,53	FrameToFrame	15,2656	FrameToFrame	1,67052	FrameToFrame	0,0469143	FrameToFrame	0,0668462
FrameToFrame	399,656	FrameToFrame	15,374	FrameToFrame	1,75028	FrameToFrame	0,0462113	FrameToFrame	0,0624436
FrameToFrame	400,239	FrameToFrame	15,016	FrameToFrame	1,74189	FrameToFrame	0,0548597	FrameToFrame	0,06245
FrameToFrame	378,169	FrameToFrame	15,1556	FrameToFrame	1,70809	FrameToFrame	0,0467052	FrameToFrame	0,0624433
FrameToFrame	415,081	FrameToFrame	15,5155	FrameToFrame	1,6674	FrameToFrame	0,0468149	FrameToFrame	0,0624186
FrameToFrame	358,546	FrameToFrame	15,3138	FrameToFrame	1,68381	FrameToFrame	0,0467068	FrameToFrame	0,0624067

FrameToFrame	330,773
FrameToFrame	360,132
FrameToFrame	374,85
FrameToFrame	387,379
FrameToFrame	387,616
FrameToFrame	353,437
FrameToFrame	368,878
FrameToFrame	319,782
TotalTime	61440,18

FrameToFrame	14,6713
FrameToFrame	14,7964
FrameToFrame	14,2195
FrameToFrame	14,5935
FrameToFrame	14,5776
FrameToFrame	14,8286
FrameToFrame	14,2348
FrameToFrame	14,8737
TotalTime	2467,3643

FrameToFrame	1,60703
FrameToFrame	1,70045
FrameToFrame	1,65643
FrameToFrame	1,58763
FrameToFrame	1,59421
FrameToFrame	1,55402
FrameToFrame	1,61325
FrameToFrame	1,68145
TotalTime	278,23562

FrameToFrame	0,0468187
FrameToFrame	0,0468139
FrameToFrame	0,0469169
FrameToFrame	0,0467225
FrameToFrame	0,0467876
FrameToFrame	0,0468184
FrameToFrame	0,0468309
FrameToFrame	0,0468181
TotalTime	16,6891491

FrameToFrame	0,0623769
FrameToFrame	0,0624429
FrameToFrame	0,0624368
FrameToFrame	0,0625449
FrameToFrame	0,0623364
FrameToFrame	0,0624676
FrameToFrame	0,0625052
FrameToFrame	0,0623717
TotalTime	11,8314158

Complex scenario - Translation

Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	296,8	FrameToFrame	15,8995	FrameToFrame	2,06588	FrameToFrame	1,7326	FrameToFrame	1,03517
FrameToFrame	325,046	FrameToFrame	15,9672	FrameToFrame	1,70957	FrameToFrame	0,0468498	FrameToFrame	0,0618554
FrameToFrame	290,341	FrameToFrame	16,1254	FrameToFrame	1,68766	FrameToFrame	0,0469031	FrameToFrame	0,0619638
FrameToFrame	333,202	FrameToFrame	16,0461	FrameToFrame	1,63442	FrameToFrame	0,0449242	FrameToFrame	0,061924
FrameToFrame	369,265	FrameToFrame	15,5017	FrameToFrame	1,7966	FrameToFrame	0,046675	FrameToFrame	0,062925
FrameToFrame	309,672	FrameToFrame	16,4519	FrameToFrame	1,79283	FrameToFrame	0,0469412	FrameToFrame	0,0634131
FrameToFrame	323,782	FrameToFrame	16,0622	FrameToFrame	1,67703	FrameToFrame	0,0469393	FrameToFrame	0,0619048
FrameToFrame	309,843	FrameToFrame	16,4536	FrameToFrame	1,78599	FrameToFrame	0,0479538	FrameToFrame	0,061966
FrameToFrame	375,485	FrameToFrame	15,8112	FrameToFrame	1,68756	FrameToFrame	0,0468992	FrameToFrame	0,0619471
FrameToFrame	303,187	FrameToFrame	16,1268	FrameToFrame	1,80699	FrameToFrame	0,0469377	FrameToFrame	0,06395
FrameToFrame	340,581	FrameToFrame	16,1241	FrameToFrame	1,79688	FrameToFrame	0,0467613	FrameToFrame	0,0619551
FrameToFrame	346,374	FrameToFrame	16,2815	FrameToFrame	1,80454	FrameToFrame	0,0459435	FrameToFrame	0,0629465
FrameToFrame	316,966	FrameToFrame	16,0313	FrameToFrame	1,66101	FrameToFrame	0,046938	FrameToFrame	0,0619522
FrameToFrame	338,188	FrameToFrame	16,0158	FrameToFrame	1,8398	FrameToFrame	0,0469368	FrameToFrame	0,0609698
FrameToFrame	344,875	FrameToFrame	16,1547	FrameToFrame	1,94541	FrameToFrame	0,0460592	FrameToFrame	0,063899
FrameToFrame	323,53	FrameToFrame	16,4215	FrameToFrame	1,78994	FrameToFrame	0,0467921	FrameToFrame	0,0619602
FrameToFrame	353,39	FrameToFrame	16,0952	FrameToFrame	1,72621	FrameToFrame	0,0479801	FrameToFrame	0,0629208
FrameToFrame	310,36	FrameToFrame	16,2343	FrameToFrame	1,82032	FrameToFrame	0,0469169	FrameToFrame	0,0609615
FrameToFrame	338,186	FrameToFrame	16,4985	FrameToFrame	1,62298	FrameToFrame	0,0459024	FrameToFrame	0,0629423
FrameToFrame	332,079	FrameToFrame	16,1105	FrameToFrame	1,75151	FrameToFrame	0,0469428	FrameToFrame	0,0622745
FrameToFrame	367,109	FrameToFrame	16,2807	FrameToFrame	1,82219	FrameToFrame	0,04694	FrameToFrame	0,0619054
FrameToFrame	338,42	FrameToFrame	15,9837	FrameToFrame	1,86603	FrameToFrame	0,0469566	FrameToFrame	0,06396
FrameToFrame	330,954	FrameToFrame	16,1252	FrameToFrame	1,8571	FrameToFrame	0,0459146	FrameToFrame	0,0619577
FrameToFrame	352,438	FrameToFrame	15,9843	FrameToFrame	1,75807	FrameToFrame	0,0469538	FrameToFrame	0,061957
FrameToFrame	309,329	FrameToFrame	16,016	FrameToFrame	1,77464	FrameToFrame	0,0468826	FrameToFrame	0,0618996
FrameToFrame	303,561	FrameToFrame	15,9064	FrameToFrame	1,64523	FrameToFrame	0,0459832	FrameToFrame	0,0629853
FrameToFrame	325,751	FrameToFrame	16,2495	FrameToFrame	1,77247	FrameToFrame	0,0479788	FrameToFrame	0,0629907
FrameToFrame	324,077	FrameToFrame	16,2177	FrameToFrame	1,7974	FrameToFrame	0,0460073	FrameToFrame	0,0629936
FrameToFrame	338,296	FrameToFrame	16,1108	FrameToFrame	1,73432	FrameToFrame	1,60952	FrameToFrame	0,0619894
FrameToFrame	303,064	FrameToFrame	16,0459	FrameToFrame	1,60368	FrameToFrame	0,0471379	FrameToFrame	0,0620023
FrameToFrame	339,174	FrameToFrame	15,7498	FrameToFrame	1,73436	FrameToFrame	0,0459913	FrameToFrame	0,0629856
FrameToFrame	345,997	FrameToFrame	16,0003	FrameToFrame	1,70317	FrameToFrame	0,047974	FrameToFrame	0,0619913
FrameToFrame	309,577	FrameToFrame	16,1718	FrameToFrame	1,65611	FrameToFrame	0,0459752	FrameToFrame	0,0619798
FrameToFrame	317,017	FrameToFrame	15,999	FrameToFrame	1,73853	FrameToFrame	0,0479756	FrameToFrame	0,0629914
FrameToFrame	352,967	FrameToFrame	15,9384	FrameToFrame	1,67587	FrameToFrame	0,0459948	FrameToFrame	0,0629904
FrameToFrame	317,598	FrameToFrame	16,0468	FrameToFrame	1,66391	FrameToFrame	0,0459201	FrameToFrame	0,0619869
FrameToFrame	346,105	FrameToFrame	16,0311	FrameToFrame	1,67612	FrameToFrame	0,0479852	FrameToFrame	0,0629875
FrameToFrame	310,609	FrameToFrame	15,8274	FrameToFrame	1,8351	FrameToFrame	0,0469752	FrameToFrame	0,0609891
FrameToFrame	289,563	FrameToFrame	15,7502	FrameToFrame	1,81688	FrameToFrame	0,0469746	FrameToFrame	0,0629914
FrameToFrame	331,626	FrameToFrame	15,8761	FrameToFrame	1,88552	FrameToFrame	0,0459829	FrameToFrame	0,0620016
FrameToFrame	324,234	FrameToFrame	15,967	FrameToFrame	1,64263	FrameToFrame	0,0479807	FrameToFrame	0,0629769

FrameToFrame	330,687	FrameToFrame	16,2673	FrameToFrame	1,76531	FrameToFrame	0,0459916	FrameToFrame	0,0629651
FrameToFrame	301,938	FrameToFrame	16,1241	FrameToFrame	1,81188	FrameToFrame	0,0459662	FrameToFrame	0,0619875
FrameToFrame	300,531	FrameToFrame	16,1397	FrameToFrame	1,854	FrameToFrame	1,70299	FrameToFrame	0,0629731
FrameToFrame	309,065	FrameToFrame	15,8765	FrameToFrame	1,71907	FrameToFrame	0,0470044	FrameToFrame	0,0609685
FrameToFrame	329,403	FrameToFrame	15,9369	FrameToFrame	1,73116	FrameToFrame	0,0469566	FrameToFrame	0,0629904
FrameToFrame	338,656	FrameToFrame	15,9381	FrameToFrame	1,71045	FrameToFrame	0,0479814	FrameToFrame	0,0629907
FrameToFrame	359,345	FrameToFrame	15,8421	FrameToFrame	1,8813	FrameToFrame	0,046982	FrameToFrame	0,0619881
FrameToFrame	322,437	FrameToFrame	16,11	FrameToFrame	1,78345	FrameToFrame	0,0460025	FrameToFrame	0,0619843
FrameToFrame	321,798	FrameToFrame	16,1086	FrameToFrame	1,73677	FrameToFrame	0,0459454	FrameToFrame	0,0629939
FrameToFrame	315,983	FrameToFrame	15,5312	FrameToFrame	1,71675	FrameToFrame	0,0479599	FrameToFrame	0,0619997
FrameToFrame	315,891	FrameToFrame	15,9073	FrameToFrame	1,6613	FrameToFrame	0,0459755	FrameToFrame	0,0629811
FrameToFrame	323,561	FrameToFrame	16,1407	FrameToFrame	1,67071	FrameToFrame	0,0479826	FrameToFrame	0,0629702
FrameToFrame	353,13	FrameToFrame	16,2336	FrameToFrame	1,59744	FrameToFrame	0,0459842	FrameToFrame	0,0619792
FrameToFrame	367,089	FrameToFrame	16,1257	FrameToFrame	1,68177	FrameToFrame	0,0469682	FrameToFrame	0,0629949
FrameToFrame	281,501	FrameToFrame	15,5308	FrameToFrame	1,83732	FrameToFrame	0,0469647	FrameToFrame	0,0609987
FrameToFrame	325,421	FrameToFrame	15,7959	FrameToFrame	1,65876	FrameToFrame	1,68723	FrameToFrame	0,0629436
FrameToFrame	274,296	FrameToFrame	15,516	FrameToFrame	1,78944	FrameToFrame	0,0479381	FrameToFrame	0,0619888
FrameToFrame	317,641	FrameToFrame	15,8439	FrameToFrame	1,81151	FrameToFrame	0,045989	FrameToFrame	0,0639911
FrameToFrame	282	FrameToFrame	15,9059	FrameToFrame	1,75225	FrameToFrame	0,0469669	FrameToFrame	0,061008
FrameToFrame	325,11	FrameToFrame	15,8597	FrameToFrame	1,6912	FrameToFrame	0,0460044	FrameToFrame	0,0629593
FrameToFrame	303,014	FrameToFrame	15,4844	FrameToFrame	1,77011	FrameToFrame	0,0469595	FrameToFrame	0,0619974
FrameToFrame	311,363	FrameToFrame	15,8445	FrameToFrame	1,66457	FrameToFrame	0,0469708	FrameToFrame	0,0639863
FrameToFrame	319,246	FrameToFrame	15,5612	FrameToFrame	1,64799	FrameToFrame	0,0469801	FrameToFrame	0,0610115
FrameToFrame	318,437	FrameToFrame	15,2494	FrameToFrame	1,70517	FrameToFrame	0,0459823	FrameToFrame	0,0629657
FrameToFrame	325,267	FrameToFrame	15,4069	FrameToFrame	1,77909	FrameToFrame	0,0479791	FrameToFrame	0,0629907
FrameToFrame	332,361	FrameToFrame	15,5948	FrameToFrame	1,66808	FrameToFrame	0,0459771	FrameToFrame	0,0619891
FrameToFrame	331,937	FrameToFrame	15,4357	FrameToFrame	1,74224	FrameToFrame	0,046991	FrameToFrame	0,0630167
FrameToFrame	318,405	FrameToFrame	15,5787	FrameToFrame	1,73125	FrameToFrame	0,046991	FrameToFrame	0,0619679
FrameToFrame	290,092	FrameToFrame	15,172	FrameToFrame	1,66913	FrameToFrame	0,046989	FrameToFrame	0,0629939
FrameToFrame	311,046	FrameToFrame	15,3134	FrameToFrame	1,62583	FrameToFrame	0,0469906	FrameToFrame	0,0609827
FrameToFrame	304,079	FrameToFrame	15,4672	FrameToFrame	1,63629	FrameToFrame	0,04699	FrameToFrame	0,063
FrameToFrame	296,5	FrameToFrame	15,4232	FrameToFrame	1,78109	FrameToFrame	0,0469663	FrameToFrame	0,0629458
FrameToFrame	367,328	FrameToFrame	15,358	FrameToFrame	1,64841	FrameToFrame	0,0469881	FrameToFrame	0,0629917
FrameToFrame	303,671	FrameToFrame	15,5634	FrameToFrame	1,7343	FrameToFrame	1,54701	FrameToFrame	0,062001
FrameToFrame	309,517	FrameToFrame	15,5923	FrameToFrame	1,53652	FrameToFrame	0,0459653	FrameToFrame	0,0612861
FrameToFrame	331,766	FrameToFrame	15,3287	FrameToFrame	1,70792	FrameToFrame	0,0469714	FrameToFrame	0,0629776
FrameToFrame	346,999	FrameToFrame	15,4686	FrameToFrame	1,64497	FrameToFrame	0,0459723	FrameToFrame	0,0619798
FrameToFrame	289,297	FrameToFrame	15,5152	FrameToFrame	1,63419	FrameToFrame	0,0469829	FrameToFrame	0,0640661
FrameToFrame	311,235	FrameToFrame	15,3287	FrameToFrame	1,51566	FrameToFrame	0,0479807	FrameToFrame	0,0608986
FrameToFrame	305,047	FrameToFrame	15,3759	FrameToFrame	1,60938	FrameToFrame	0,0459762	FrameToFrame	0,0639638
FrameToFrame	333,499	FrameToFrame	15,5145	FrameToFrame	1,691	FrameToFrame	0,0479769	FrameToFrame	0,0609785
FrameToFrame	325,532	FrameToFrame	15,6408	FrameToFrame	1,72762	FrameToFrame	0,0459765	FrameToFrame	0,0629747

FrameToFrame	319,358	FrameToFrame	15,4057	FrameToFrame	1,63335	FrameToFrame	0,0459669	FrameToFrame	0,0629856
FrameToFrame	325,906	FrameToFrame	15,5628	FrameToFrame	1,64065	FrameToFrame	1,61058	FrameToFrame	0,0609666
FrameToFrame	318,674	FrameToFrame	15,4528	FrameToFrame	1,70764	FrameToFrame	0,0621812	FrameToFrame	0,063976
FrameToFrame	338,592	FrameToFrame	16,0622	FrameToFrame	1,76609	FrameToFrame	0,0470157	FrameToFrame	0,0609862
FrameToFrame	339,983	FrameToFrame	15,7349	FrameToFrame	1,7449	FrameToFrame	0,0468899	FrameToFrame	0,0640017
FrameToFrame	333,203	FrameToFrame	15,4064	FrameToFrame	1,8248	FrameToFrame	0,0469666	FrameToFrame	0,06195
FrameToFrame	313,265	FrameToFrame	14,9835	FrameToFrame	1,57812	FrameToFrame	0,0469618	FrameToFrame	0,062976
FrameToFrame	334,844	FrameToFrame	15,1724	FrameToFrame	1,74447	FrameToFrame	0,0469765	FrameToFrame	0,0609769
FrameToFrame	326,579	FrameToFrame	15,2654	FrameToFrame	1,74994	FrameToFrame	0,0460015	FrameToFrame	0,062968
FrameToFrame	319,468	FrameToFrame	15,4687	FrameToFrame	1,72993	FrameToFrame	0,0459335	FrameToFrame	0,0619718
FrameToFrame	284,234	FrameToFrame	15,1567	FrameToFrame	1,53561	FrameToFrame	0,0479746	FrameToFrame	0,0629782
FrameToFrame	298,093	FrameToFrame	15,14	FrameToFrame	1,61222	FrameToFrame	0,0459736	FrameToFrame	0,0630254
FrameToFrame	341,11	FrameToFrame	15,3438	FrameToFrame	1,78609	FrameToFrame	0,0469656	FrameToFrame	0,0619458
FrameToFrame	275,781	FrameToFrame	16,0005	FrameToFrame	1,80421	FrameToFrame	0,0469804	FrameToFrame	0,0629686
FrameToFrame	332,954	FrameToFrame	15,7488	FrameToFrame	1,73059	FrameToFrame	1,60961	FrameToFrame	0,0609762
FrameToFrame	300,501	FrameToFrame	15,4547	FrameToFrame	1,80355	FrameToFrame	0,046956	FrameToFrame	0,0629856
FrameToFrame	327,703	FrameToFrame	15,8586	FrameToFrame	1,70226	FrameToFrame	0,0469714	FrameToFrame	0,0629878
FrameToFrame	326,217	FrameToFrame	15,4683	FrameToFrame	1,69393	FrameToFrame	0,0459913	FrameToFrame	0,0629927
FrameToFrame	332,516	FrameToFrame	15,6713	FrameToFrame	1,56875	FrameToFrame	0,0479596	FrameToFrame	0,0619689
FrameToFrame	305,595	FrameToFrame	15,1093	FrameToFrame	1,72469	FrameToFrame	0,0459858	FrameToFrame	0,0629708
FrameToFrame	298,797	FrameToFrame	15,2978	FrameToFrame	1,78966	FrameToFrame	0,0469784	FrameToFrame	0,0609968
FrameToFrame	313,672	FrameToFrame	15,0935	FrameToFrame	1,71605	FrameToFrame	0,0469801	FrameToFrame	0,0639802
FrameToFrame	313,89	FrameToFrame	14,8126	FrameToFrame	1,74179	FrameToFrame	0,046989	FrameToFrame	0,0619901
FrameToFrame	282,985	FrameToFrame	15,2505	FrameToFrame	1,69518	FrameToFrame	0,046974	FrameToFrame	0,0619939
FrameToFrame	306,171	FrameToFrame	14,9055	FrameToFrame	1,70903	FrameToFrame	0,0469627	FrameToFrame	0,0629898
FrameToFrame	336,141	FrameToFrame	14,6403	FrameToFrame	1,72487	FrameToFrame	0,0459768	FrameToFrame	0,0609878
FrameToFrame	307,877	FrameToFrame	14,9059	FrameToFrame	1,77655	FrameToFrame	1,56281	FrameToFrame	0,0629939
FrameToFrame	316,107	FrameToFrame	14,8758	FrameToFrame	1,62823	FrameToFrame	0,0459823	FrameToFrame	0,062985
FrameToFrame	321,391	FrameToFrame	15,0609	FrameToFrame	1,6314	FrameToFrame	0,0479775	FrameToFrame	0,0619958
FrameToFrame	306,906	FrameToFrame	15,1422	FrameToFrame	1,74117	FrameToFrame	0,0459662	FrameToFrame	0,0629728
FrameToFrame	299,874	FrameToFrame	14,7343	FrameToFrame	1,67694	FrameToFrame	0,0479746	FrameToFrame	0,0629741
FrameToFrame	292,453	FrameToFrame	14,5149	FrameToFrame	1,78045	FrameToFrame	0,0469708	FrameToFrame	0,0609807
FrameToFrame	328,094	FrameToFrame	15,0932	FrameToFrame	1,62457	FrameToFrame	0,0449697	FrameToFrame	0,0629891
FrameToFrame	302,032	FrameToFrame	15,1096	FrameToFrame	1,70021	FrameToFrame	0,0469788	FrameToFrame	0,0629901
FrameToFrame	323,499	FrameToFrame	15,1246	FrameToFrame	1,67192	FrameToFrame	1,51555	FrameToFrame	0,0629894
FrameToFrame	321,784	FrameToFrame	15,8119	FrameToFrame	1,81938	FrameToFrame	0,0470237	FrameToFrame	0,0609913
FrameToFrame	338,481	FrameToFrame	15,6885	FrameToFrame	1,76757	FrameToFrame	0,0479515	FrameToFrame	0,061991
FrameToFrame	336,626	FrameToFrame	15,4383	FrameToFrame	1,78217	FrameToFrame	0,045965	FrameToFrame	0,0629827
FrameToFrame	344,906	FrameToFrame	15,8262	FrameToFrame	1,70247	FrameToFrame	0,0469589	FrameToFrame	0,0619692
FrameToFrame	338,252	FrameToFrame	15,7817	FrameToFrame	1,72072	FrameToFrame	1,60889	FrameToFrame	0,0639827
FrameToFrame	316,202	FrameToFrame	15,8751	FrameToFrame	1,74328	FrameToFrame	0,0478451	FrameToFrame	0,061992
FrameToFrame	330,328	FrameToFrame	15,9385	FrameToFrame	1,72119	FrameToFrame	0,0469926	FrameToFrame	0,0619872

FrameToFrame	331,86
FrameToFrame	330,204
FrameToFrame	336,686
FrameToFrame	343,485
FrameToFrame	357,875
FrameToFrame	300,545
FrameToFrame	286,689
FrameToFrame	322,217
FrameToFrame	343,908
FrameToFrame	285,686
TotalTime	43449,938

FrameToFrame	15,2639
FrameToFrame	15,969
FrameToFrame	15,9689
FrameToFrame	15,8288
FrameToFrame	15,859
FrameToFrame	15,8442
FrameToFrame	15,8435
FrameToFrame	15,4689
FrameToFrame	15,811
FrameToFrame	15,439
TotalTime	2119,3274

FrameToFrame	1,7134
FrameToFrame	1,68813
FrameToFrame	1,70015
FrameToFrame	1,77928
FrameToFrame	1,78945
FrameToFrame	1,71565
FrameToFrame	1,73564
FrameToFrame	1,67531
FrameToFrame	1,61876
FrameToFrame	1,72141
TotalTime	232,87888

FrameToFrame	0,0459659
FrameToFrame	0,0469817
FrameToFrame	0,0459759
FrameToFrame	1,61043
FrameToFrame	0,0459367
FrameToFrame	0,0480186
FrameToFrame	0,0460192
FrameToFrame	0,0470012
FrameToFrame	0,0459887
FrameToFrame	0,0479403
TotalTime	23,6139193

FrameToFrame	0,0619776
FrameToFrame	0,0629779
FrameToFrame	0,0619978
FrameToFrame	0,0629615
FrameToFrame	0,062975
FrameToFrame	0,0619888
FrameToFrame	0,0629975
FrameToFrame	0,0619743
FrameToFrame	0,0629827
FrameToFrame	0,0609702
TotalTime	9,4000035

Complex scenario - Scaling

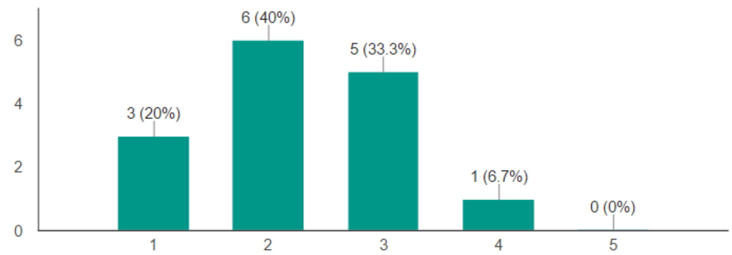
Original		Original_Window		Original_Window_Pyramid		Final (Camshift)		Final (Optical Flow)	
FrameToFrame	246,209	FrameToFrame	16,3895	FrameToFrame	1,18049	FrameToFrame	1,43199	FrameToFrame	0,839467
FrameToFrame	231,344	FrameToFrame	16,3123	FrameToFrame	1,25302	FrameToFrame	0,0468646	FrameToFrame	0,0619464
FrameToFrame	245,172	FrameToFrame	16,6085	FrameToFrame	1,29236	FrameToFrame	0,0469072	FrameToFrame	0,0619679
FrameToFrame	245,765	FrameToFrame	16,1717	FrameToFrame	1,22826	FrameToFrame	0,0611619	FrameToFrame	0,0629638
FrameToFrame	237,751	FrameToFrame	16,6243	FrameToFrame	1,2154	FrameToFrame	0,0479563	FrameToFrame	0,0630728
FrameToFrame	237,141	FrameToFrame	16,3124	FrameToFrame	1,25129	FrameToFrame	0,0459072	FrameToFrame	0,061993
FrameToFrame	193,874	FrameToFrame	16,75	FrameToFrame	1,20965	FrameToFrame	0,0462972	FrameToFrame	0,0619679
FrameToFrame	237,173	FrameToFrame	16,5942	FrameToFrame	1,21132	FrameToFrame	0,0479451	FrameToFrame	0,0619843
FrameToFrame	245,219	FrameToFrame	16,7181	FrameToFrame	1,22958	FrameToFrame	0,0469445	FrameToFrame	0,0639988
FrameToFrame	244,905	FrameToFrame	16,8598	FrameToFrame	1,25699	FrameToFrame	0,0459492	FrameToFrame	0,0611809
FrameToFrame	245,327	FrameToFrame	16,4075	FrameToFrame	1,31918	FrameToFrame	0,0469505	FrameToFrame	0,0630016
FrameToFrame	243,5	FrameToFrame	16,6231	FrameToFrame	1,24775	FrameToFrame	0,0459361	FrameToFrame	0,0629776
FrameToFrame	221,157	FrameToFrame	16,5635	FrameToFrame	1,13364	FrameToFrame	0,047949	FrameToFrame	0,0609647
FrameToFrame	243,128	FrameToFrame	17,0933	FrameToFrame	1,3107	FrameToFrame	0,0459079	FrameToFrame	0,063994
FrameToFrame	279,513	FrameToFrame	16,9223	FrameToFrame	1,34503	FrameToFrame	0,0479191	FrameToFrame	0,0619769
FrameToFrame	257,734	FrameToFrame	16,8585	FrameToFrame	1,32058	FrameToFrame	0,046904	FrameToFrame	0,061976
FrameToFrame	250,375	FrameToFrame	16,7182	FrameToFrame	1,2294	FrameToFrame	0,0469461	FrameToFrame	0,0619756
FrameToFrame	257,844	FrameToFrame	16,8606	FrameToFrame	1,35445	FrameToFrame	0,0449293	FrameToFrame	0,0630048
FrameToFrame	229,562	FrameToFrame	17,1552	FrameToFrame	1,26725	FrameToFrame	0,0469461	FrameToFrame	0,0629462
FrameToFrame	236,031	FrameToFrame	17,4059	FrameToFrame	1,24152	FrameToFrame	0,0469422	FrameToFrame	0,0609669
FrameToFrame	242,906	FrameToFrame	17,0174	FrameToFrame	1,22878	FrameToFrame	0,0469441	FrameToFrame	0,0640007
FrameToFrame	262,781	FrameToFrame	16,9199	FrameToFrame	1,41056	FrameToFrame	0,0479448	FrameToFrame	0,0612803
FrameToFrame	227,704	FrameToFrame	16,9077	FrameToFrame	1,33232	FrameToFrame	0,0469406	FrameToFrame	0,0620167
FrameToFrame	247,641	FrameToFrame	17,3112	FrameToFrame	1,32249	FrameToFrame	0,0449479	FrameToFrame	0,0639869
FrameToFrame	262,749	FrameToFrame	17,8755	FrameToFrame	1,40418	FrameToFrame	0,047873	FrameToFrame	0,0609698
FrameToFrame	262,063	FrameToFrame	17,1562	FrameToFrame	1,2856	FrameToFrame	0,0469784	FrameToFrame	0,0639792
FrameToFrame	276,953	FrameToFrame	17,7977	FrameToFrame	1,25416	FrameToFrame	0,0460047	FrameToFrame	0,062026
FrameToFrame	266,166	FrameToFrame	18,5294	FrameToFrame	1,3242	FrameToFrame	0,0473547	FrameToFrame	0,0619493
FrameToFrame	250,528	FrameToFrame	18,6576	FrameToFrame	1,31189	FrameToFrame	0,0459752	FrameToFrame	0,0623028
FrameToFrame	286,78	FrameToFrame	19,4207	FrameToFrame	1,49854	FrameToFrame	0,0469775	FrameToFrame	0,0619971
FrameToFrame	263,53	FrameToFrame	18,7352	FrameToFrame	1,41184	FrameToFrame	0,046981	FrameToFrame	0,0639706
FrameToFrame	242,183	FrameToFrame	18,2809	FrameToFrame	1,44989	FrameToFrame	0,0469711	FrameToFrame	0,061993
FrameToFrame	276,608	FrameToFrame	18,7344	FrameToFrame	1,47618	FrameToFrame	0,0469624	FrameToFrame	0,0629792
FrameToFrame	275,998	FrameToFrame	19,2812	FrameToFrame	1,38806	FrameToFrame	0,0469823	FrameToFrame	0,061974
FrameToFrame	289,734	FrameToFrame	19,6248	FrameToFrame	1,43761	FrameToFrame	0,0459659	FrameToFrame	0,0619808
FrameToFrame	282,048	FrameToFrame	19,3755	FrameToFrame	1,6246	FrameToFrame	0,0479644	FrameToFrame	0,0619942
FrameToFrame	295,889	FrameToFrame	19,4843	FrameToFrame	1,47344	FrameToFrame	0,0459704	FrameToFrame	0,0629779
FrameToFrame	266,606	FrameToFrame	19,5934	FrameToFrame	1,4716	FrameToFrame	0,0469582	FrameToFrame	0,0629766
FrameToFrame	330,253	FrameToFrame	20,3278	FrameToFrame	1,50692	FrameToFrame	0,0469967	FrameToFrame	0,0609772
FrameToFrame	302,577	FrameToFrame	19,9993	FrameToFrame	1,5161	FrameToFrame	0,0469743	FrameToFrame	0,0629821
FrameToFrame	328,657	FrameToFrame	20,4998	FrameToFrame	1,44214	FrameToFrame	0,0469425	FrameToFrame	0,0619676

FrameToFrame	349,237	FrameToFrame	20,7509	FrameToFrame	1,57195	FrameToFrame	0,0476388	FrameToFrame	0,0639831
FrameToFrame	298,281	FrameToFrame	21,124	FrameToFrame	1,56579	FrameToFrame	0,0456965	FrameToFrame	0,0619631
FrameToFrame	354,174	FrameToFrame	21,1112	FrameToFrame	1,67698	FrameToFrame	0,047973	FrameToFrame	0,0619888
FrameToFrame	331,564	FrameToFrame	21,0616	FrameToFrame	1,55966	FrameToFrame	0,0449784	FrameToFrame	0,0619779
FrameToFrame	346,783	FrameToFrame	20,9989	FrameToFrame	1,61878	FrameToFrame	0,0469839	FrameToFrame	0,0639757
FrameToFrame	323,126	FrameToFrame	21,8289	FrameToFrame	1,77073	FrameToFrame	0,0469679	FrameToFrame	0,0619776
FrameToFrame	322,688	FrameToFrame	21,844	FrameToFrame	1,62943	FrameToFrame	0,0469582	FrameToFrame	0,0619885
FrameToFrame	321,204	FrameToFrame	22,6881	FrameToFrame	1,7916	FrameToFrame	0,0469887	FrameToFrame	0,0629721
FrameToFrame	333,829	FrameToFrame	22,7338	FrameToFrame	1,50211	FrameToFrame	0,0469672	FrameToFrame	0,0629734
FrameToFrame	311,766	FrameToFrame	22,9368	FrameToFrame	1,61236	FrameToFrame	0,0459675	FrameToFrame	0,061975
FrameToFrame	333,017	FrameToFrame	22,968	FrameToFrame	1,65378	FrameToFrame	0,0479599	FrameToFrame	0,0619718
FrameToFrame	323,86	FrameToFrame	23,6737	FrameToFrame	1,73845	FrameToFrame	0,0469868	FrameToFrame	0,0619901
FrameToFrame	351,643	FrameToFrame	23,9218	FrameToFrame	1,77585	FrameToFrame	0,0469486	FrameToFrame	0,0630029
FrameToFrame	309,406	FrameToFrame	23,3425	FrameToFrame	1,70544	FrameToFrame	0,0459749	FrameToFrame	0,0619708
FrameToFrame	286,936	FrameToFrame	23,4849	FrameToFrame	1,78799	FrameToFrame	0,0469656	FrameToFrame	0,0619788
FrameToFrame	335,473	FrameToFrame	24,0161	FrameToFrame	1,75935	FrameToFrame	0,0466744	FrameToFrame	0,062348
FrameToFrame	348,625	FrameToFrame	24,3892	FrameToFrame	1,78739	FrameToFrame	0,0469759	FrameToFrame	0,0629481
FrameToFrame	370,099	FrameToFrame	24,2196	FrameToFrame	1,78673	FrameToFrame	0,0469826	FrameToFrame	0,0620051
FrameToFrame	332,797	FrameToFrame	24,4996	FrameToFrame	1,7236	FrameToFrame	0,0470182	FrameToFrame	0,0629834
FrameToFrame	387,614	FrameToFrame	25,2508	FrameToFrame	1,90161	FrameToFrame	0,045931	FrameToFrame	0,0631434
FrameToFrame	394,647	FrameToFrame	25,312	FrameToFrame	1,86822	FrameToFrame	0,0469656	FrameToFrame	0,0629721
FrameToFrame	343,406	FrameToFrame	25,4674	FrameToFrame	1,89597	FrameToFrame	0,0479689	FrameToFrame	0,0609843
FrameToFrame	379,597	FrameToFrame	26,0311	FrameToFrame	1,89256	FrameToFrame	0,0459624	FrameToFrame	0,0629872
FrameToFrame	355,253	FrameToFrame	26,1566	FrameToFrame	2,13287	FrameToFrame	0,0479631	FrameToFrame	0,0620003
FrameToFrame	375,519	FrameToFrame	26,6714	FrameToFrame	2,06928	FrameToFrame	0,0449729	FrameToFrame	0,062975
FrameToFrame	402,883	FrameToFrame	27,4851	FrameToFrame	1,95701	FrameToFrame	0,047981	FrameToFrame	0,0629827
FrameToFrame	423,897	FrameToFrame	27,876	FrameToFrame	1,95839	FrameToFrame	0,0469813	FrameToFrame	0,0609724
FrameToFrame	403,601	FrameToFrame	28,0306	FrameToFrame	2,02668	FrameToFrame	0,0459768	FrameToFrame	0,0629789
FrameToFrame	380,302	FrameToFrame	29,0472	FrameToFrame	2,02021	FrameToFrame	0,046998	FrameToFrame	0,0619888
FrameToFrame	396,928	FrameToFrame	29,5608	FrameToFrame	2,13012	FrameToFrame	0,0469727	FrameToFrame	0,0639879
FrameToFrame	429,68	FrameToFrame	29,9543	FrameToFrame	2,26622	FrameToFrame	0,0469807	FrameToFrame	0,060982
FrameToFrame	393,541	FrameToFrame	30,0455	FrameToFrame	2,1753	FrameToFrame	0,0459797	FrameToFrame	0,0629725
FrameToFrame	487,808	FrameToFrame	30,0628	FrameToFrame	2,21827	FrameToFrame	0,0474092	FrameToFrame	0,0620112
FrameToFrame	406,273	FrameToFrame	30,188	FrameToFrame	2,20227	FrameToFrame	0,0469627	FrameToFrame	0,0619503
TotalTime	22814,535	TotalTime	1574,212	TotalTime	117,40191	TotalTime	4,9104766	TotalTime	5,4584723

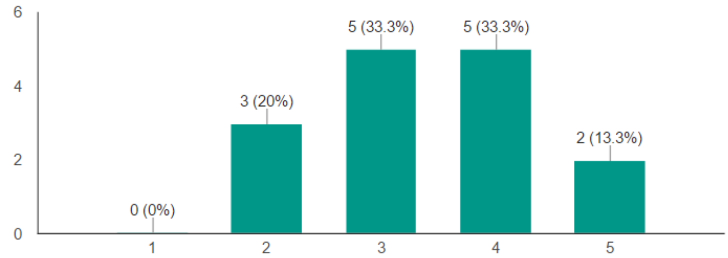
Quality tests

Post-it rotation

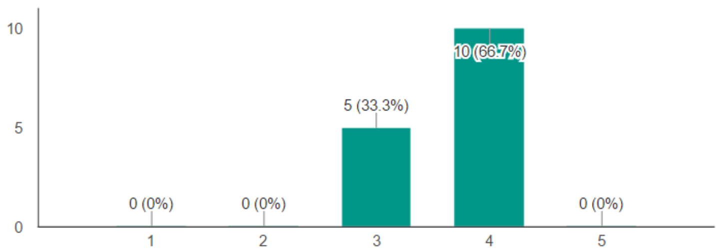
Original (15 responses)



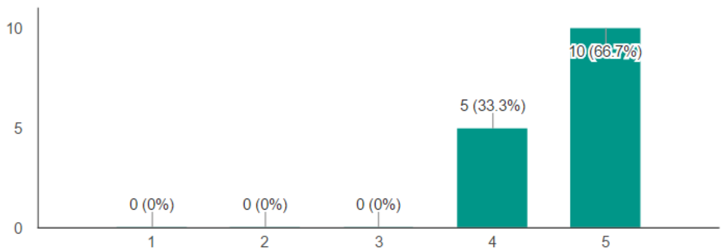
Original_Window (15 responses)



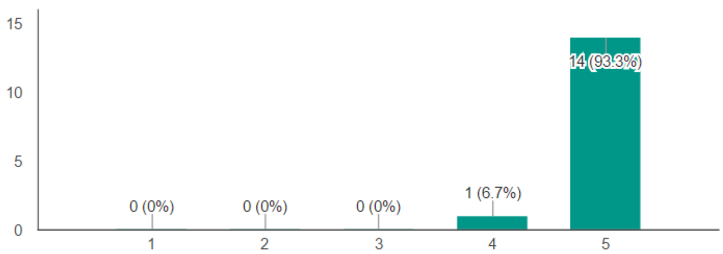
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

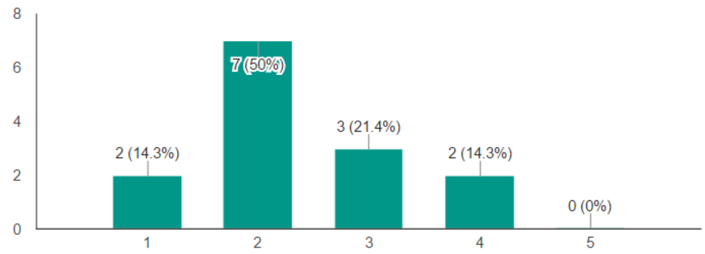


Final (Optical Flow) (15 responses)

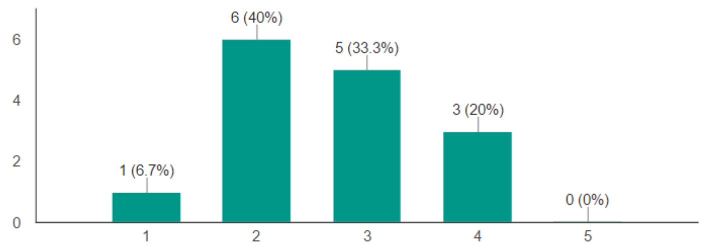


Post-it translation

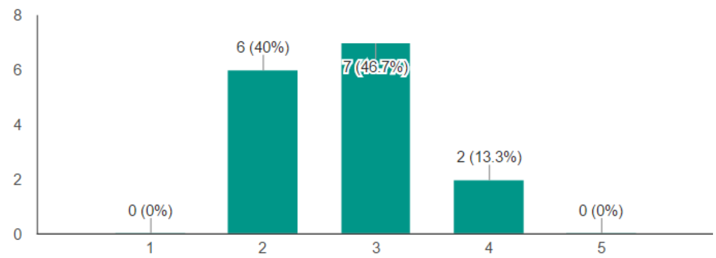
Original (14 responses)



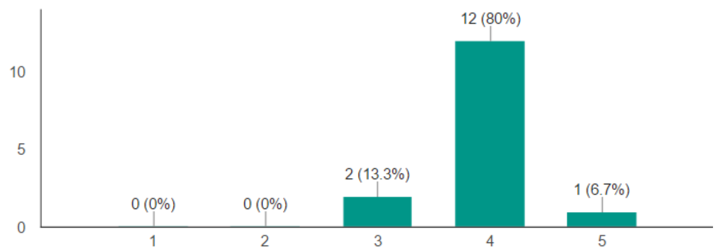
Original_Window (15 responses)



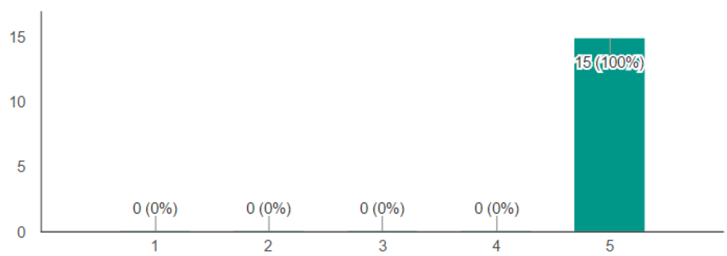
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

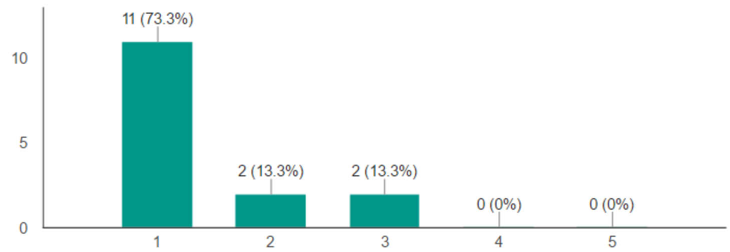


Final (Optical Flow) (15 responses)

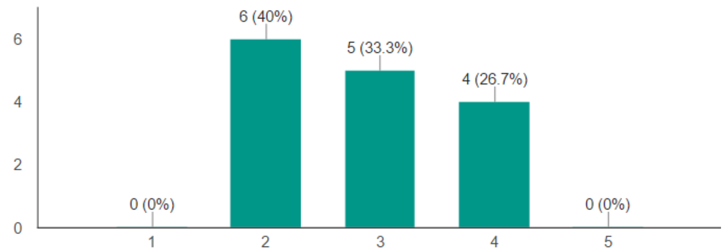


Post-it scaling

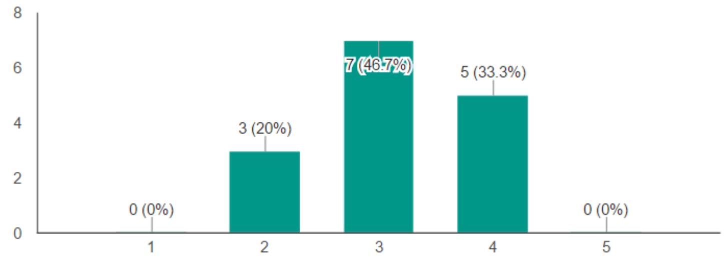
Original (15 responses)



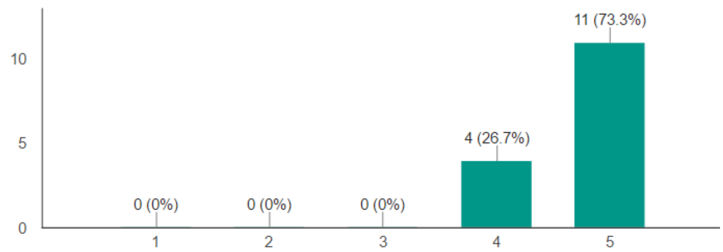
Original_Window (15 responses)



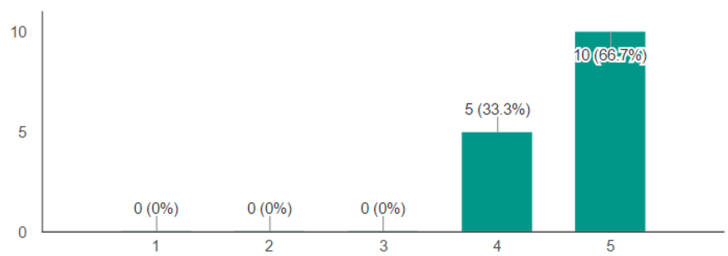
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

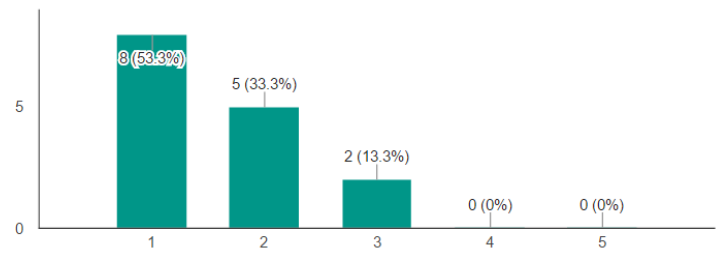


Final (Optical Flow) (15 responses)

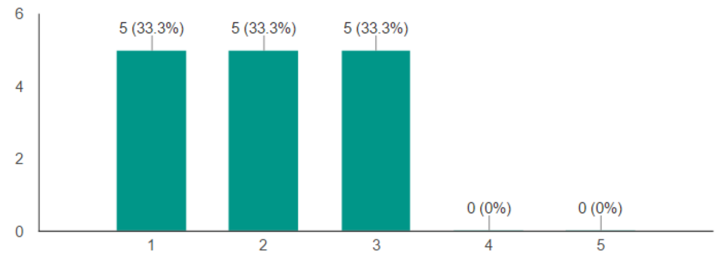


Phone rotation

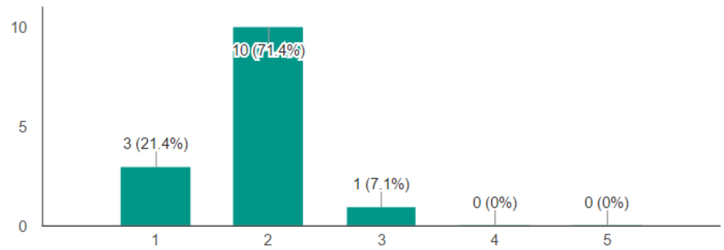
Original (15 responses)



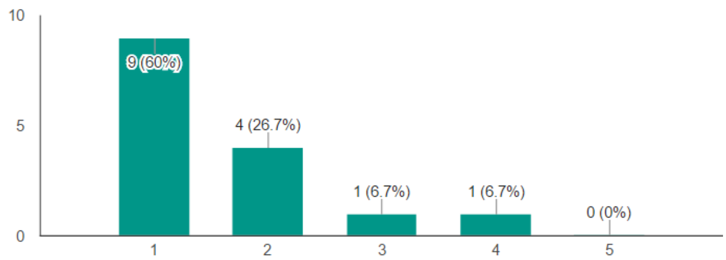
Original_Window (15 responses)



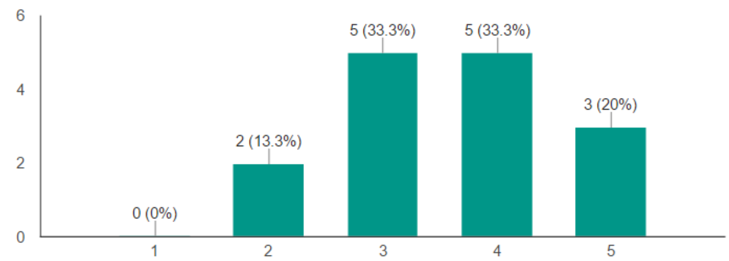
Original_Window_Pyramid (14 responses)



Final (Camfhit) (15 responses)

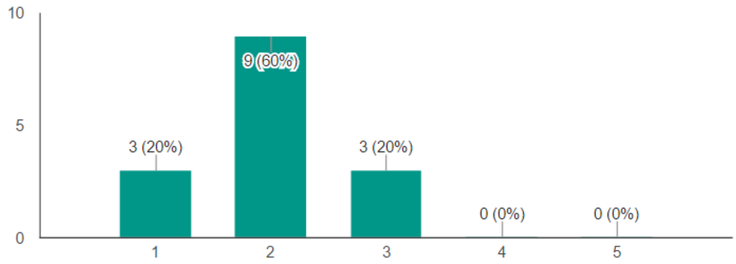


Final (Optical Flow) (15 responses)

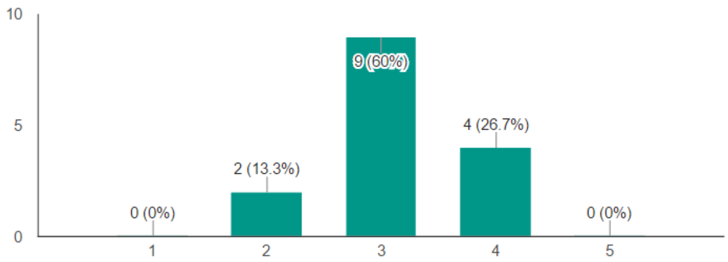


Phone translation

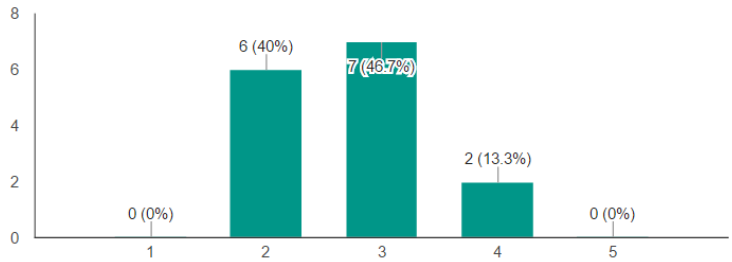
Original (15 responses)



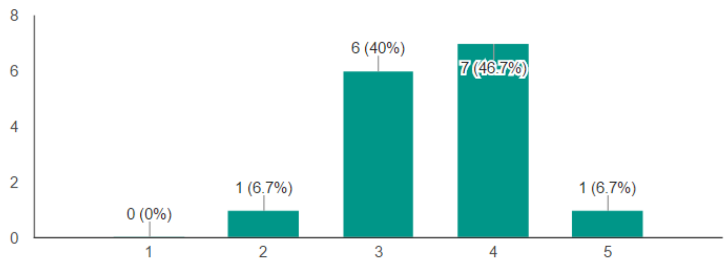
Original_Window (15 responses)



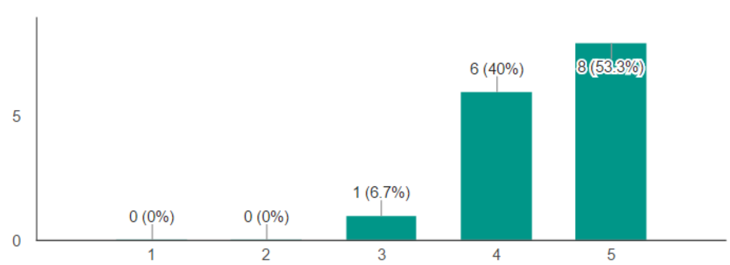
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

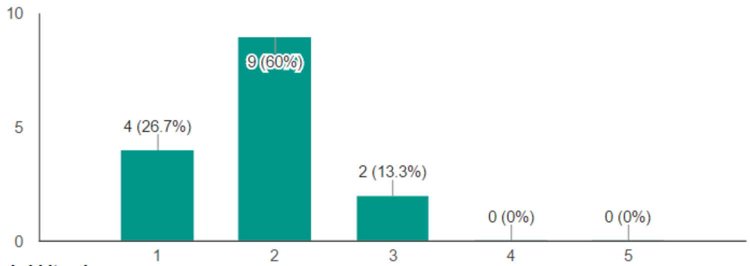


Final (Optical Flow) (15 responses)

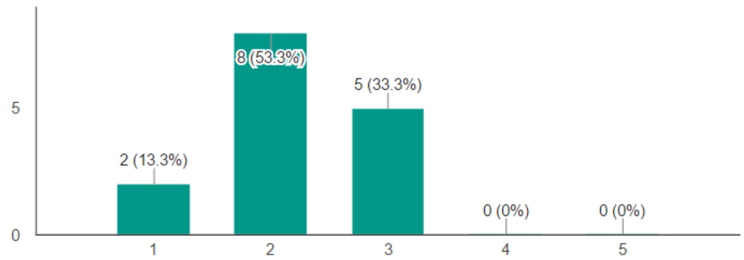


Phone scaling

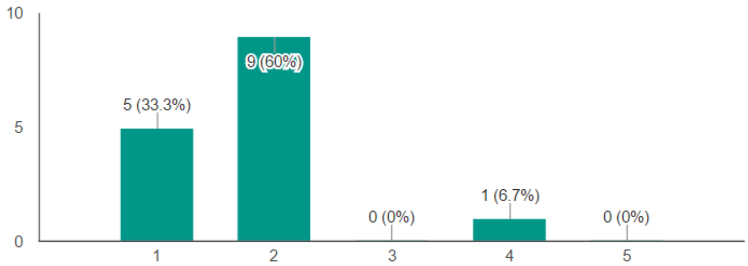
Original (15 responses)



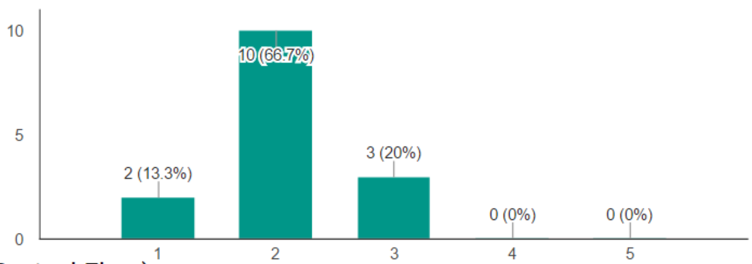
Original_Window (15 responses)



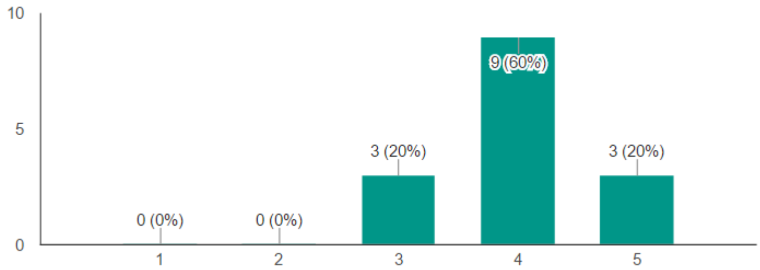
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

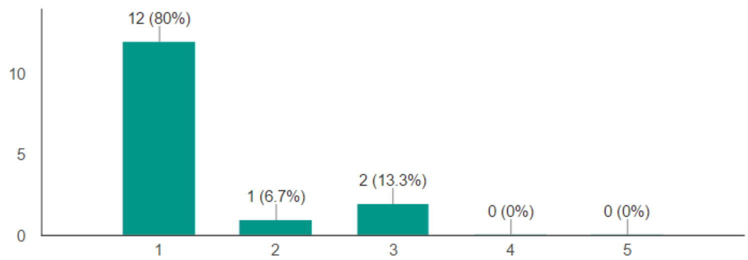


Final (Optical Flow) (15 responses)

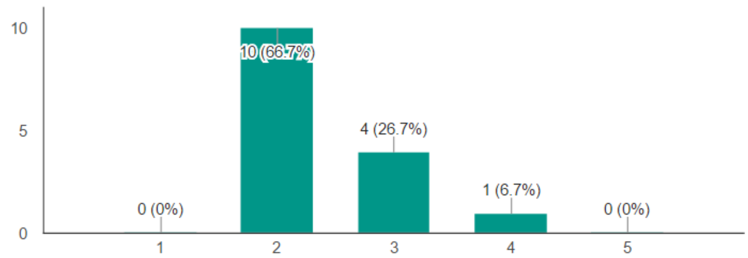


Building rotation

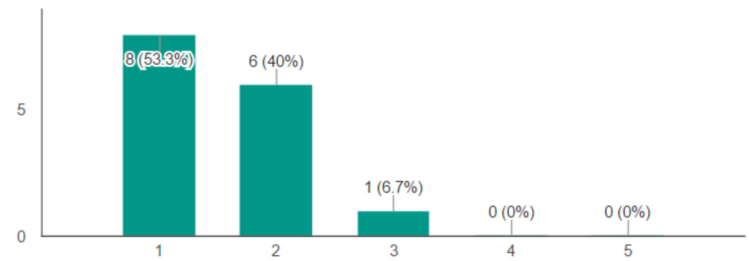
Original (15 responses)



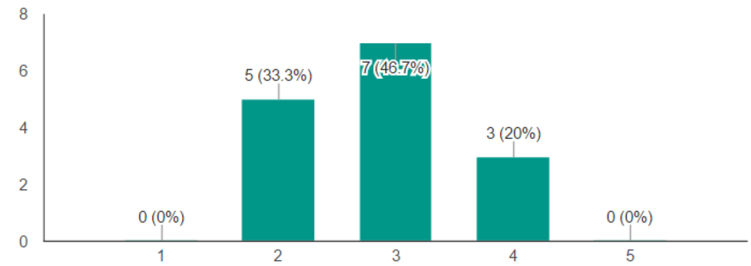
Original_Window (15 responses)



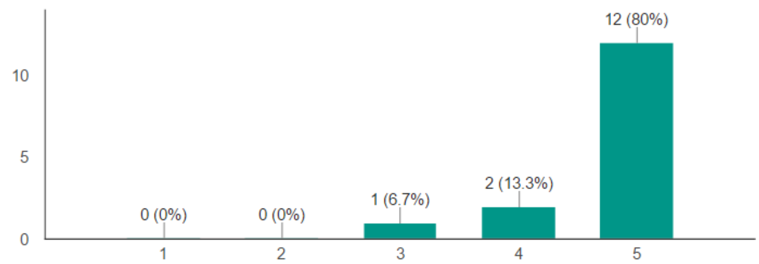
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

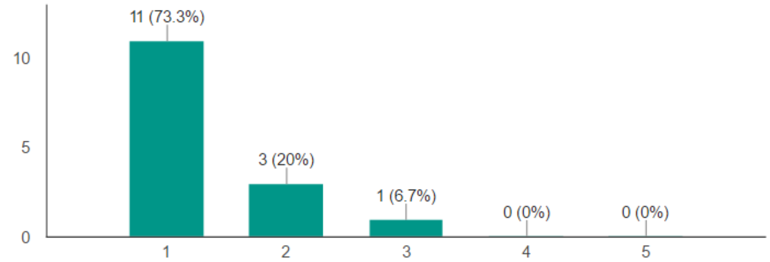


Final (Optical Flow) (15 responses)

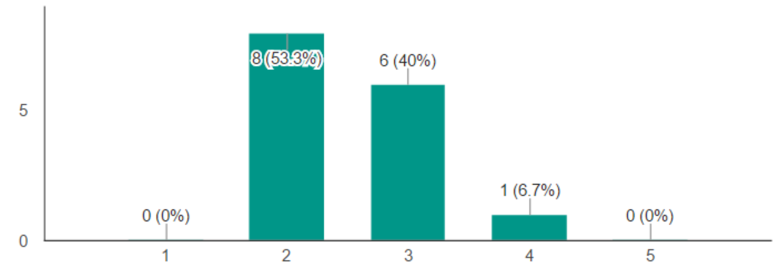


Building translation

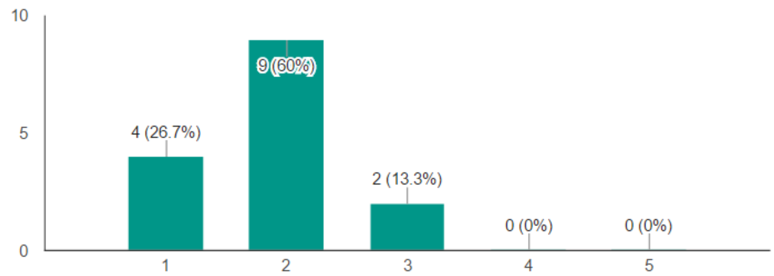
Original (15 responses)



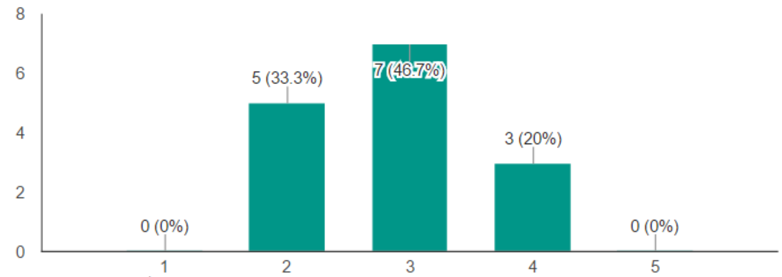
Original_Window (15 responses)



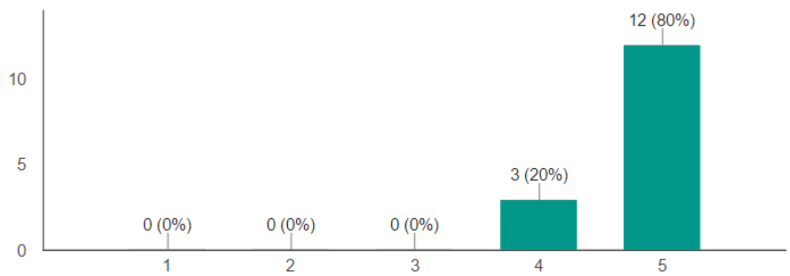
Original_Window_Pyramid (15 responses)



Final (Camshift) (15 responses)

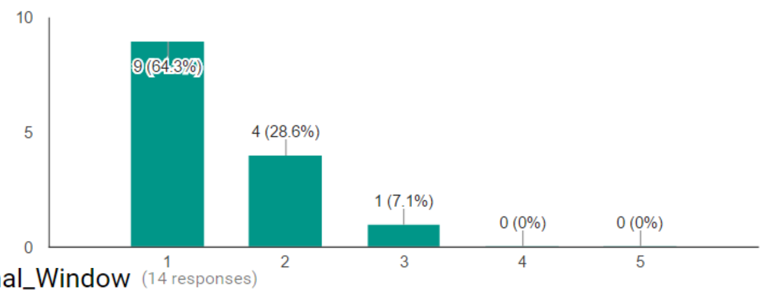


Final (Optical Flow) (15 responses)

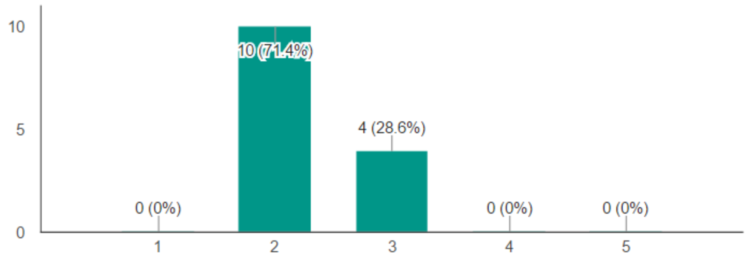


Building scaling

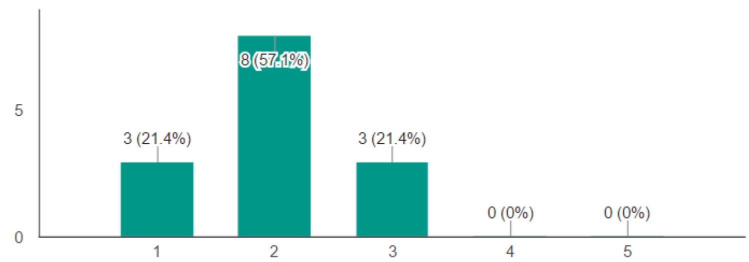
Original (14 responses)



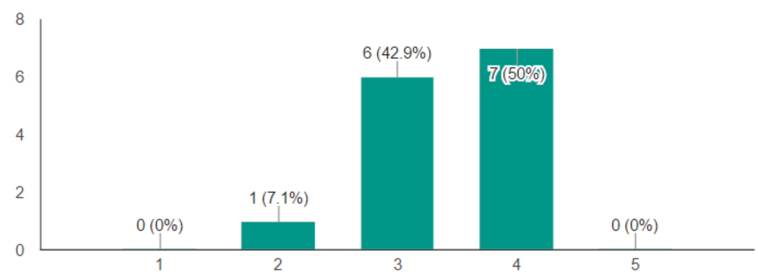
Original_Window (14 responses)



Original_Window_Pyramid (14 responses)



Final (Camshift) (14 responses)



Final (Optical Flow) (14 responses)