UNIVERSIDADE DA MADEIRA

Center of Exact Sciences and Engineering

Master in Informatics Engineering

INEXPENSIVE SOLUTION FOR REAL-TIIME VIDEO AND IMAGE STITCHING

Filipe Quintal

Advisor: Dr Mon-Chu Chen

Thesis submitted to the University of Madeira to obtain the degree of Master in Informatics Engineering.

November 2009

Acknowledgements

I want to express my appreciation for my thesis advisor, Dr. Mon-Chu Chen, for all the support, advice and patience during the development of this thesis.

I would also like to thank my family and friends for the support during this period of my study

Abstract

Image stitching is the process of joining several images to obtain a bigger view of a scene. It is used, for example, in tourism to transmit to the viewer the sensation of being in another place. I am presenting an inexpensive solution for automatic real time video and image stitching with two web cameras as the video/image sources. The proposed solution relies on the usage of several markers in the scene as reference points for the stitching algorithm. The implemented algorithm is divided in four main steps, the marker detection, camera pose determination (in reference to the markers), video/image size and 3d transformation, and image translation. Wii remote controllers are used to support several steps in the process. The built-in IR camera provides clean marker detection, which facilitates the camera pose determination. The only restriction in the algorithm is that markers have to be in the field of view when capturing the scene.

Several tests where made to evaluate the final algorithm. The algorithm is able to perform video stitching with a frame rate between 8 and 13 fps. The joining of the two videos/images is good with minor misalignments in objects at the same depth of the marker, misalignments in the background and foreground are bigger. The capture process is simple enough so anyone can perform a stitching with a very short explanation.

Although real-time video stitching can be achieved by this affordable approach, there are few shortcomings in current version. For example, contrast inconsistency along the stitching line could be reduced by applying a color correction algorithm to every source videos. In addition, the misalignments in stitched images due to camera lens distortion could be eased by optical correction algorithm.

The work was developed in Apple's Quartz Composer, a visual programming environment. A library of extended functions was developed using Xcode tools also from Apple.

Index

Table of contents

List of Images

List of Tables

List of Graphics

Key Words

Quartz Composer: (QC) Apple's visual-programming based development tool, used in the development of this thesis.

Xcode: Apple's IDE for software development for apple's products

Scene: Any location that is or is about to be captured.

Stitching: Process of joining 2 or more images, to get a wider view of a scene also called **joining or collage**.

Joining, collage: Joining or collage can also be used to name the result of a stitching.

Joiner: Algorithm that performs the joining.

Marker: Element in a scene to be use used as a reference, it's usually easy to differentiate from the scene.

Wii Remote: Nintendo's video game controller, it was released with the Nintendo Wii.

Camera Pose: Position and orientation of a camera in real space, normally expressed by 3 positions (X,Y,Z), and 3 angles.

RGB: Red, Green and Blue color space.

HSL: Hue, Saturation and Luminosity color space

LED: Light emitting diode, it can emit different types of light (for example infra red light).

Patch: Quartz Composer's routine or agglomerate of routines.

Blob detection: Process of detecting the coordinates of a point or a region in an image.

Infrared: Radiation invisible to the human eyes, which can be detected by the most of cameras.

Wii Remote pitch: The pitch value is the rotation of the Wii Remote according to coordinate system parallel to the sides of the Wii Remote and as origin the center of the Wii Remote.

Wii Remote roll: The Roll value is the rotation of the Wii Remote according to a coordinate system parallel to the front (or back) of the Wii Remote and has as origin the center of the Wii Remote.

Focal length: The distance between the center of a lens or curved mirror and its focus.

Image translation: Is the result of shifting the images according to an axle.

1 Introduction

1.1 The research problem

This thesis focuses on finding a process to accomplish video stitching in real time using available hardware and software to keep the process inexpensive, the final solution has to be cheap and flexible enough so anyone can use it.

Image stitching is a well studied problem with several solutions proposed, but if we consider the solutions that process images in real time the number of proposed solutions decrease (that is probably because real time is not a critical requirement for image stitching), so for me to accomplish video stitching I couldn't use any image stitching solution and replicate it for each frame of the video.

The final solution has to be accessible to anyone, it must not require any specific hardware or software that could be expensive or hard to get.

It's important like I have said before, to the final solution to be inexpensive In terms of hardware and software needed, but it also as to be easy to use, and it also should allow some flexibility during the capture, no complex setups should be used during the capture, so that anyone with no photography or computer vision experience could test it.

In summary in order to meet with the research problem the video stitching solution has to be cheap, and flexible.

1.2 The importance of the problem

Sometimes a image is not enough to show all the detail of a given scene as on wants, and sometimes a video can show a lot more detail that just images, video and image stitching is an obvious solution to increase the amount of information a image/video could transmit to the user.

A video can show a lot more of a scene than fixed images, it can show environmental changes, interaction between the different objects of the scene. Image stitching is already a well-studied problem with several good solutions (even commercial programs like Photoshop are capable of performing a really high quality image stitching), but there are not a lot of solutions for video stitching especially with the degree of freedom (during the capture) that I am proposing.

I want to show the result of the stitching during the actual capture, that way it's possible to avoid errors during the process, and of course the real time solution also

contribute a lot for the fun of the process, it's different to see the result during the capture than to wait for the result, a not real time solution could move away interested users.

It's important for the solution to be inexpensive because I want to everyone to test the application, it is also important to be easy to use, it should not require specific hardware, and of course it has to be cheap in terms of computational power needed, so that a real time solution can be presented.

1.3 Proposed solution

The proposed solution performs automated video stitching with 2 Web-Cameras as the video sources, the stitching algorithm relies on the usage of a marker anywhere in the scene, a Wii Remote is used as a support during the process, a pose determination algorithm is implemented to detect the position of the camera which is then used to transform the images before the stitching, the final step of the algorithm is the translation that bring the images next to each other.

The solution performs automated video and image stitching, that means that the image joining and transformation are made automatically with no need for adjustments from the user, that way is easy for everyone pick up and try with no specific experience required. I have used web-cameras as the sources for the videos they returned videos with sufficient quality for my work, and they are relatively cheap when compared with specialized equipment like fisheye lenses.

A Marker is placed in the scene before the capture, the marker is differentiated from the rest of the scene by the IR light it emits. The marker is like a reference used to calculate the different parameters of the joining like the translation, scaling and rotation that are applied to each image/video, with the usage of a marker I maintain the cost of the algorithm low (computational-wise), that decision is crucial for the algorithm to perform video stitching in real time.

Like I have said before the marker is differentiated from the rest of the scene by the IR light it emits, the Wii Remote is used to detect the IR light from the maker, I could probably have found a less expensive solution to detect the IR marker, but the main advantage of the Wii Remote is that it also provides a built in blob detection algorithm, and there's also a patch for Quartz Composer to work with the Wii Remote that handles all the peripheral-computer communication. Besides the marker detection the Wii Remote is also used in the pose determination algorithm (the rotation values returned by his accelerometers).

In order transform the images before the stitching the proposed solution first discovers the position and orientation of the camera in real space relatively to the markers, the pose algorithm implemented is a combination between a traditional pose determination

algorithm and the values from the Wii Remote accelerometers to get a complete pose estimation.

After the orientation of the camera is discovered (3 rotations and a x, y,z position), their value is used to transform the videos/images, the goal of this transformation is to prepare the videos/images for the stitching so that they are joined in a more natural way, improving the overall quality of the stitching.

After all the transformations where made the stitching itself takes place, which is actually a simple 3 axles translation that stitch the videos by a reference point (on of maker's points).

The proposed solution meets the requirements of the research problem, it performs an inexpensive and easy to operate real time video stitching algorithm, the only thing that could constrain the usage of the solution, is the requirement for 2 Wii Remotes and the construction of a marker.

1.4 Synopsis

The rest of the thesis is organized as follows. In chapter 2 I talk about all the research needed for complete my work such as background knowledge and similar solutions, in Chapter 3 I present the tools that I have used to implement my ideas, and how I have extended the Quartz Composer patch library to suit my needs in the implementation.

In Chapter 4 there is the explanation of all the steps of the stitching algorithm. The first step is the marker detection (section 4.1) I've tried some different approaches for this stage, in this section I explain their weaknesses and strengths, and I have also shown all the process that lead me to the final solution. Also in Chapter 4 I illustrate how I have implemented the pose detection algorithm (section 4.2) and how I combined the pitch and roll values from a Wii Remote with a "traditional" pose detection algorithm to get a complete camera pose determination. In section 4.3 I lay out the reason why I did the pose detection, which was to transform the images to improve the stitching. After the image transformation with the values from the pose detection algorithm the final step in the algorithm is to translate the image in the QC viewer window to complete the actual stitching (section 4.4). At the end of chapter 4 there is a summary of the stitching algorithm (section 4.5)

In chapter 5 first I explained how I've changed the image-stitching algorithm to perform video stitching (section 5.1), and lay out different ways of presenting the stitched videos to the user (section 5.2), later I explore possible application for the proposed solution (section 5.3).

Later in chapter 6 I present the final results (section 6.1), and I have also prepared a series of simple tests to evaluate the overall quality of the stitching process (section 6.2).

Finally I present the final conclusions, and future improvements in chapter 7.

The consulted bibliography is listed in chapter 8.

2 Literature Review

2.1 Background knowledge

During my work I had to accomplish several tasks most of them related to the computer vision technology.

For the goal of detecting differences in colors and variance in images, it's very important to know which color space to use, and their main features. In (H.J.C, 2008) the authors explain in a very easy to understand way the difference between the HSV and RGB color spaces, this paper focus in how to make a robot see like a human, that's a bit different from my work but, some of the problems found are the same as the ones that I have found. In (Pemberton, 2002) I have found which are the main weaknesses and strengths of each color space, (HSL and RGB), in (ChaosPro, 2008) there is a complete explanation of the HSL and RGB color space. Quartz Composer uses the RGB color space so I had to convert the values of the pixels from RGB to HSL, I've use the code in (Gattas, 2008)and modify it to Core Image kernel language.

The core of my work will be developed in quartz composer, I have never used this programming language before, this language encourages user exploration and is easy to learn by trying the different functions, at (Apple inc, 2008) I have found a complete Quartz Composer programming guide.

To be able to detect a shape it could be useful to detect the edges of an object in a image, there are several algorithm for detecting edges the most common are the sobel and the canny, an edge is normally a part of an image where it changes abruptly in (Green, Edge detection tutorial, 2002),(Green, Edge Detection Tutorial, 2002) the author talks in detail about these two algorithms. Quartz composer have already built in a sobel filter, also there is a patch for a canny edge detection filter provided by Kineme (with the open cv plug in).

One of the main tasks of the proposed solution is to estimate the position of a camera in real space according to markers this problem is called pose estimation and is has being studied for a long time now, in (Quan & Lan, 1999) the author have developed a solution for determining the pose of a camera in space with 3 points and also a linear solution for 4 and 5 to n known points, in (Martin & Robert, 1981) the author presents a analytic solution for pose estimation with 3 and 4 known points in (Kamata, Eason, Tsuji, & Kawaguchi, 1992) the authors have also developed a algorithm (with a closed form solution) for camera pose estimation using 4 points, as we can se the pose estimation problem has several solutions, in (Chandra, 1989) the author have developed a closed solution for this problem using a 4 point marker with quadrangular orientation, this solution is divided in

several steps which makes the testing easier, I have followed this paper in pose estimation part of my work.

In (Cambrige in colour, 2008) there is a study of different types of images projections, their advantages and weaknesses, and also a lot of examples.

2.2 Required technology

Like I said before I want to make the collages inexpensive to accomplish so there's not much technology required.

Two Wii Remotes and 2 webcams are required for video stitching, the webcams have to be external because they will be moving around attached to the Wii Remote, it is also advised to use two identical web-cams (same model), a computer with the operating system Mac OS X (10.5) or more recent. To build the markers are needed 12 Infra Red LEDs, an AAA battery, wires and weld, and also a rigid structure to attach the LEDs to it, I have built my own structure with aluminum bars, in a way that it occupies as least area in the image as possible.

2.3 Previous works

One of the tasks that I have to complete in my work is to figure out how to join two different pictures of the same "scene" taken from different points of view in (Lihi & Perona, Automating Joiners, 2007), there is a relatively high level technique to and some guidelines how to join two pictures, this paper points the main advantages of scene collages: add more information to a single picture; allowing multiple view points in the same picture; this paper also provides some good examples of really nice scene collages, there is also in (Lihi & Perona, 2007) more examples of collages from the same authors. In (Nomura, Zhang, & Nayar, 2007) there are also some examples (videos and pictures) of collages. One of the challenges of scene collage is to find from which position to get the pictures in (Nomura, Zhang, & Nayar, 2007) the authors used an array of cameras that could be arrange in various ways, the author have a build an application to automate the scene collage, but it give the user an interface to adjust the collage as he wishes.

One of the main features of my work is the possibility of using more than one camera, beside the obvious option of using more than one camera for providing images for a scene collage, this approach can be used for more functions (Neel, Wojciech, & Shai, 2006) the authors use an array of cameras to improve the alpha matting (used for example in cinema for background subtraction). If we use more than one camera we cloud improve the accuracy of the tracking, in (Black & Ellis, 2006)the authors uses multiple cameras to track objects in 3d using the results from 2d tracking from each camera.

The concept of using a mesh to transform images is relatively simple we have two meshes the source one (that correspond to the original image), and the result one; each vertex on the source one will be mapped to a vertex in the result, resulting in a transformed image.

The mesh approach is normally used for two task, first is useful to resolve the optical lens distortion that's present in almost every camera, in (Pintaric, Neumann, & Rizzo) the authors have used a mesh for just that, they have a "correction mesh" which when applied to an image removes the optical distortion from it. The other approach is when one needs to distort the image to simulate some type of environment, in (Bourke, 2006) the authors have built a plug in for quartz composer that supports the mesh transformations, the author then use this transformation for example to distort images projected to round or spherical surfaces.

This two approaches are useful to improve the quality of the image and adapt the image to the environment in witch the image is projected, but do not resolve the problem of joining two images.

In (Rocchini, Cignoni, & Montani) the authors have used the mesh approach to build a 3d model of an object (based on a number of images that cover all the image), the mesh is used to calculate the 3d coordinates of the object and to map the images. To map the images to the object the authors have to join images, this is similar of one of the things that I could have tried with the mesh approach, the mesh in this paper is calculated trough feature tracking in the image and it's only used to improve the mapping on the 3d object, the authors did not use the mesh to resolve the optical distortion, (since they used a lot of images they probably have a large amount of redundant parts of the object that can be removed if they aspect is not right)

The (Foote & Kimber, 2000) work combine some of the approaches of other authors, they have build a camera array with five cameras to be used for in conferences, or lectures, the authors used a mesh to remove the optical distortion, in their approach the relative positions of the cameras is the same and the stitching is straight forward, the images can be presented to the user as normal size image at the time which is only a portion of the panoramic image, then a motion detection is applied to it to pan zoom or tile according to movement. (For example the algorithm could detect that a speaker is talking, the returned image could be a zoomed image to the speaker). The main advantages of this approach are that it can be used with videos and because the cameras are fixed the stitching is fast and it solves the optical distortion. The main disadvantage is the distortion of the objects if they are closer to the cameras, and the position of the cameras cannot be changed.

In (Timo, 2009) the author have build a Virtual Reality environment with the Wii Remote, it uses some of the same Computer Vision functions as my work, but they are implemented differently, for my work the most important part of (Timo, 2009) work is how

it estimates the intrinsic values of the Wii Remote, these values are very important to estimate the camera orientation, and because Nintendo does not provide them there was no way for me to find them.

2.4 Similar Solutions

In (Kang & Yoon, 2005)the authors have built a virtual tour for the web using several images, the result is a really smooth collage to be used in a web based environment, although the final artifact the same (the collage) the authors use a different approach than mine's they have used 2d images to build a 3d scene where the user can navigate freely, that is not the only place where our works differ in (Kang & Yoon, 2005) approach spends a lot effort on preparing the images to the collage, in mine I will try to minimize that effort by providing a "clean" image detection to the collage algorithms, I will spend more effort on automating the joining of the images.

In (Lihi & Perona, 2007) the author have build a joiner for images taken from different points of view, it uses features tracking to find similarities in pairs of pictures and then based on the similarities, to each image is applied a translation, rotation and scaling to make a collage, at my work I used markers to get the points to apply the transformation functions, this approach is also different from mine in how the images are taken, this joiner have to deal with a lot more redundant portions of images than mine's, other important feature of this work is that, the application provides a interface to the user to adjust the collage when the results did not came as expected.

This work is more automated than mine, but in the paper there is no reference that this joiner will support real time video input, and that is one of the main features of my approach.

3 Tools used

I have used the same tool during the almost all of the development of my thesis but even though the development was centered on apple's Quartz Composer some other tools were used to extend Quartz Composer's functionality.

In this chapter I will present the development tools used trough the thesis, first I will present Quartz Composer and, since QC it's a bit different from other development tools I will very shortly present his main features (section 3.1). Like I've said before I needed to extend quartz composer I did that using a proper template available in apple's Xcode Tools, later in this chapter I will summarize how that extension of functionality was made (section 3.2), in the end of this chapter there is a summary of all the tools and techniques used in the development (section 3.3)

3.1 Quartz composer

Quartz composer is a visual programming development tool, based on nodes that comes as part of apple's Xcode tools since Mac OSX 10.4 and more recently in the iPhone SDK, it process and render graphical data, taking advantage of Cocoa, Quartz 2D, Core Image, OpenGL, and QuickTime frameworks**,** it allows the combination of the capabilities of those frameworks in a easy and transparent way that combined with visual programming and the no need for compiling made the development in quartz intuitive, and in a way it invites de user to explore it capabilities. The source file created by quartz it's called a composition and it's the result of the connections between patches.

During my work when I had to do some complex things to accomplish some goal, the Quartz Composer function library was not vast enough and so I had to extend it by creating my own patches.

3.1.1 Patches

The programming in quartz composer is made trough the connection of patches. A patch works like a function or routine in traditional programming languages each patch is a processing unit computing a fraction of the algorithm (for example resizing a image).

For better performance the execution Quartz Composer uses lazy evaluation approach, each patch is only computed if it results are being rendered or it's used by something that is being rendered.

The patches in quartz composer are divided by the following categories:

The **Composite** patches receive two images and uses one image to apply some type of transformation to the second one.

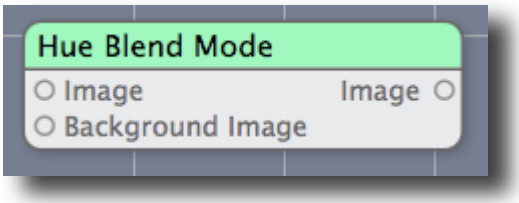


Figure 1: Hue Blend Mode patch

The **Filter** patches (which are in majority in QC), receive an image which is applied a transformation based on inputs from the user or from other patches, the result is the transformed image.

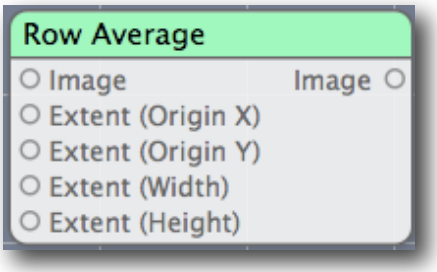


Figure 2: Row Average patch

The **Generator** patches generate images algorithmically based on the input from the user.

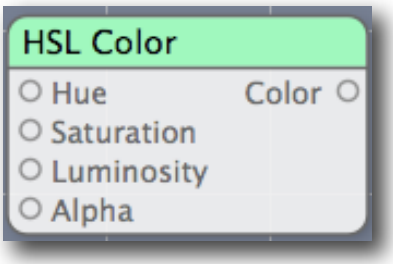


Figure 3: HSL color generator patch

The **Modifier** patches, takes the input (an image, a String...), modifies it some way, the object returned is the same type as the one received (a modification to an image will return a image).

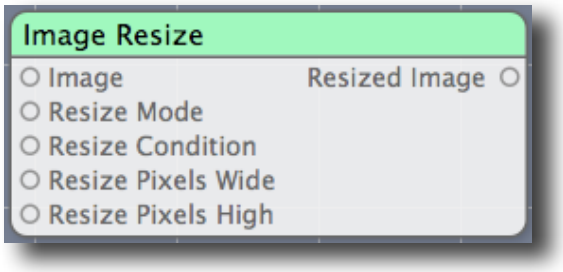


Figure 4: Image Resize patch

The **Controller** patches handle manly with external interfaces and is used normally as the input to other patches for example to make some other patch enabled based on an external event (mouse click).

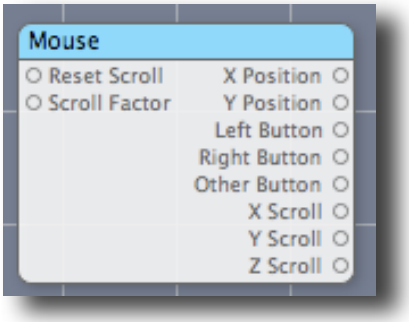


Figure 5:Mouse Control patch

The **Source** patches are used to get data from an external source, ex: image or video. The data can then be used in a composition.

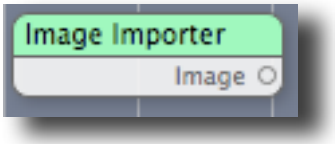


Figure 6:Image importer patch

The **Programming** patch provides text fields for the user to implement the patch behavior, there are JavaScript, OpenGL Shading Language, and Core image kernel, programming patches.

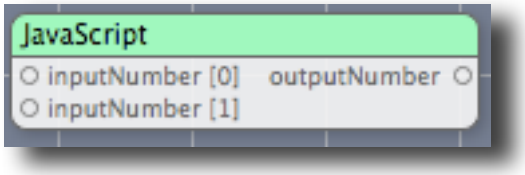


Figure 7: JavaScritp programming patch

The **Environment patches** are used normally as macros (a patch that contains other patches inside, and the changes made on a macro will affect it sub-patches). The environment patches normally are used to transform the look in some way of a composition, which as to be "inside" the environment patch.

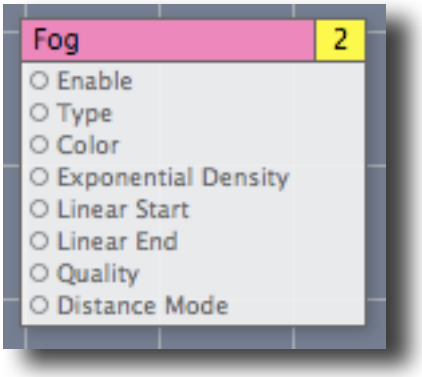


Figure 8: Fog generator patch

The **Renderer** patch renders the input to the viewer window, for example the billboard patch renders the input image to the viewer window in a customized (in width, height, positioning, rotation) rectangle.

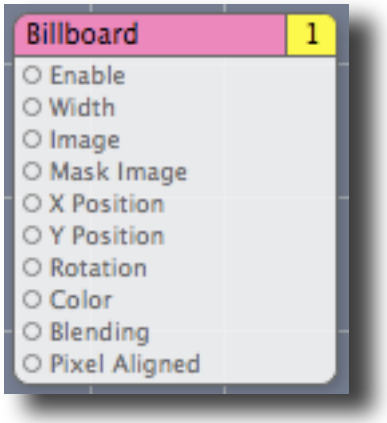


Figure 9: Billboard patch

***The renderer and Environment patches are organized in layers, elements with a higher layer number will be seen on top.

The **tool** patches are a collection of patches used to accomplish concrete goals on a composition, for example the iterator patch is useful to run an array.

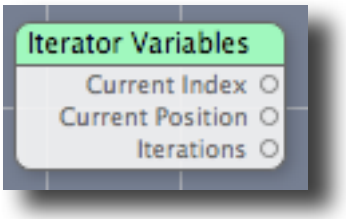


Figure 10:Iterator patch

3.1.1.1 Macros

To decrease the complexity and improve the modularity of a composition the patches can be organized in macros the macros keep the input and output necessary to process his sub-patches, a macro could be a sub-patch to another macro, a macro keeps all the behavior and the actual components of the original patches.

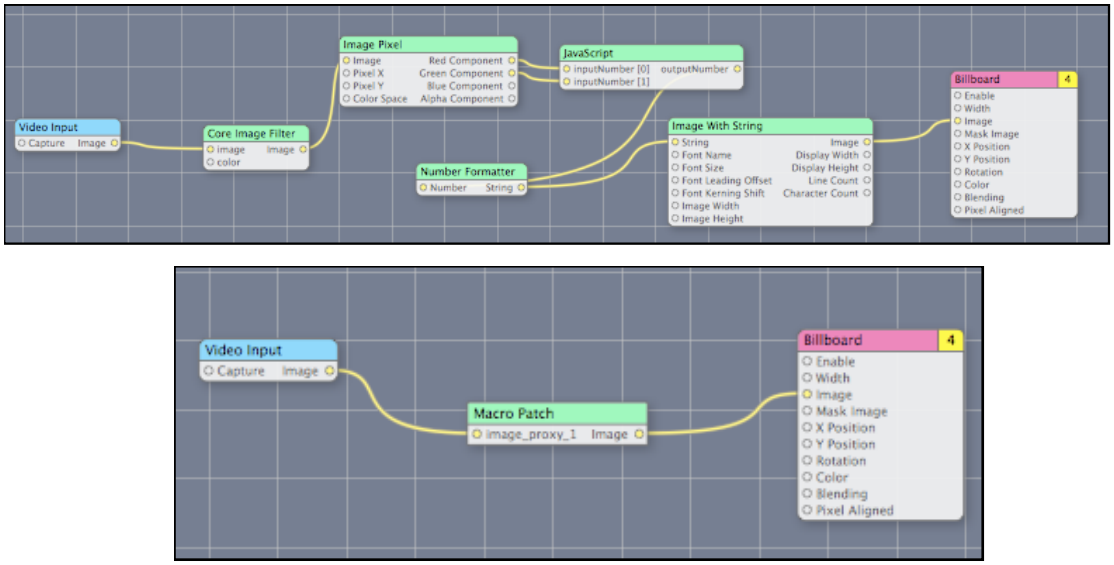
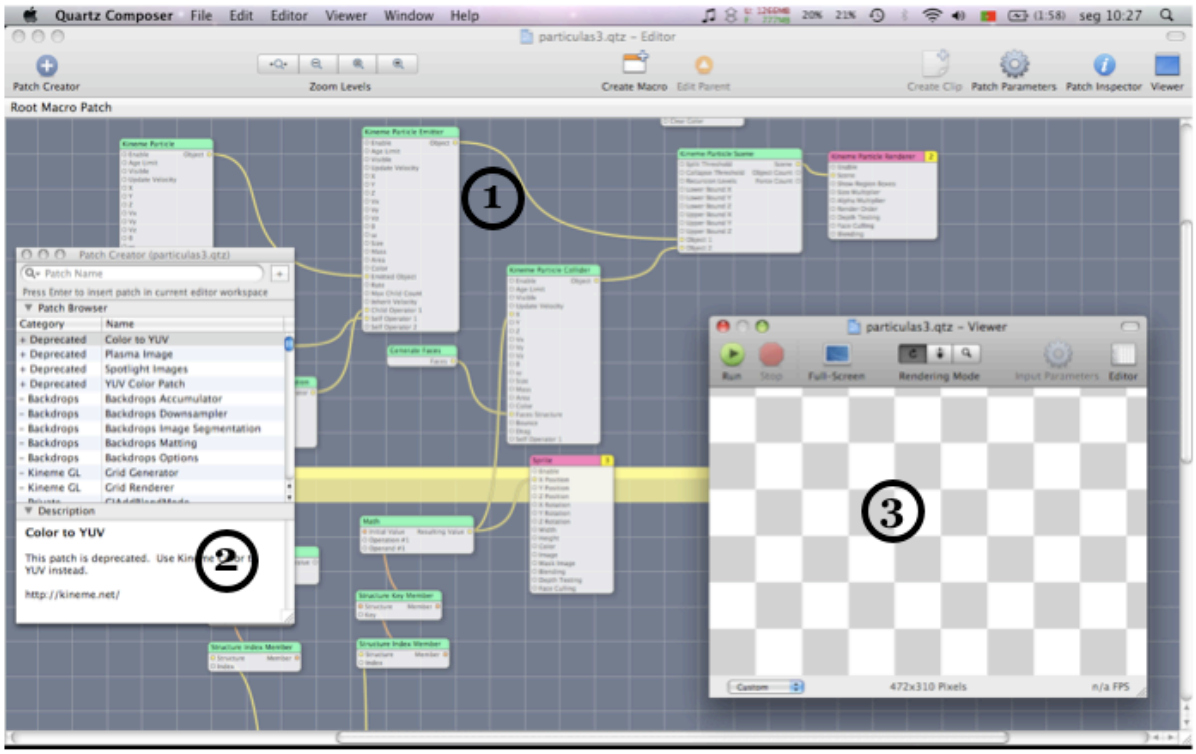


Figure 11: Macro clustering example

3.1.2 Interface



1. Composition Editor
2. Patch and Clip Library
3. Composition Viewer Window

3.1.2.1 Composition Editor

If quartz composer was a “traditional” development tool this was the code editor, but because quartz composer uses a visual programming language, this is where the user adds and connects the patches to his composition.

3.1.2.2 Patch and Clip Editor

This is where the user can find all the patches that can be used in a composition, the patches are grouped by their category, at the bottom of the window there is a description of the selected patch.

3.1.2.3 Composition Viewer Window

In the composition viewer window the user can see the result of the composition, he can stop or play the composition, at the bottom of the window there is the current frames per second info.

The compositions are rendered into an area that has 2 by 2 quartz composer units, from -1 to 1 (the visible part) in the x and y-axis, and "infinite" Z units.

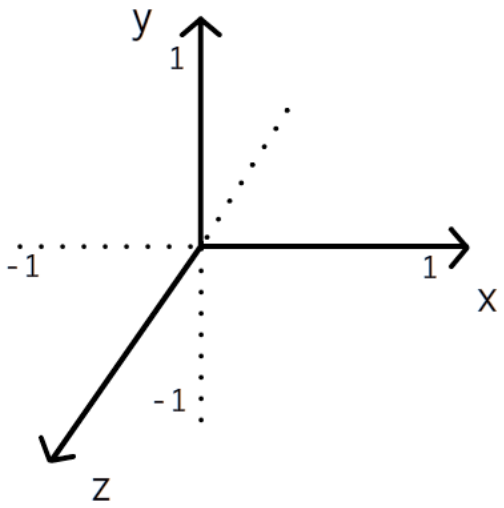


Figure 12:Quartz Composer coordinate system

3.1.3 Composition example

The next example shows the interaction between the patches to create a composition:

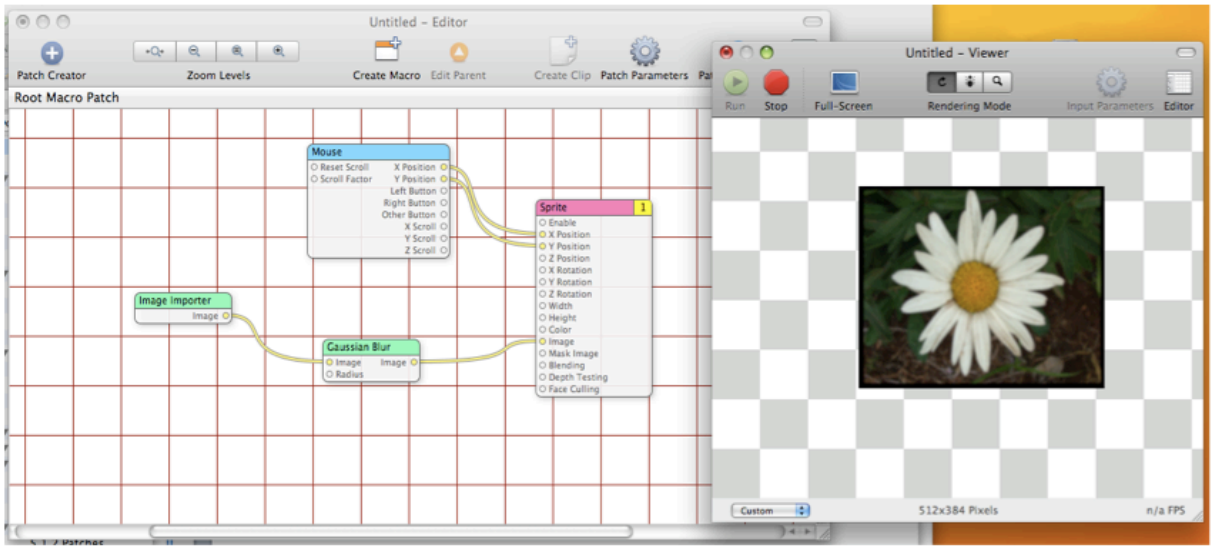


Figure 13: Composition example

The composition above, first imports an image from a file in the computer, next the image passes trough a Gaussian blur filter, the result is rendered in a sprite, the position of the sprite in the viewer window is the same as the mouse position.

3.1.4 Opportunities for extensions

In my opinion the area where Quartz Composer still is missing, and one area that was important in my work, is the complexity of the compositions in terms of communication between the user and the composition.

The compositions are static, they can respond to a user inputs but the result are still limited, and to get a decent result the composition reaches a really high complexity.

The programming patches in QC (JavaScript, OpenGL Shading Language, and Core image kernel), give the developer a good tool to extend QC.

- **JavaScritp** is useful to arithmetic, logical and cyclical operations that are quite complicated with the numeric and logical operation patches.
- **Core image Kernel language**, is an image processing language contains a sub set of Open GL shading language and Core Image functions, it allows very fast per pixel processing, but has some limitations when we need to access several pixels at the same time. Quartz composer has a patch where we can test the core image kernel code, the **core image filter patch**, it can receive as input a float, an images and tuples with 2,3,4 elements and it return always only one image, it executes the code for each pixel of the input images. The kernel image patch is executed by the GPU, which is useful to remove CPU load during the execution of the composition.
- **OpenGL Shading Language** is used to writer shaders that can simulate for example environment effects such as fog, lighting.

3.2 Quartz Composer Custom Patch Creation

During my work when I had to do some complex things to accomplish a certain goal, the Quartz Composer function library was not vast enough and so I had to extend it by creating my own patches.

The functionality of Quartz Composer can be extend using the programming patches that I have mentioned above but the most flexible way to expand QC is to write the patches

from scratch in Xcode, and that was my approach when I needed to write more complicated functions like a blob detection algorithm or the pose detection algorithm.

3.2.1 Xcode tools

Xcode tools is a collection of tools for developing on Mac OS X, I've used the Xcode IDE that is part of Xcode tools.

I used Xcode to build custom patches for quartz composer. I have used apple's template and documentation (Guide, 2008; Apple inc, 2008), the patches are written in Obj C language, the compiling produces an .plug-in file which has to be copied to the Quartz Composer Plug-ins folder at the system library, to be tested within a composition.

The task of building patches can be a bit tiring, because for every new build one as to copy the patch to right folder then open quartz composer and finally test the patch in a composition. Once I was used the language and the syntax which is a bit different from all the object oriented languages that I've know, it was easier to prevent some errors, removing that way unnecessary tests.

3.2.2 The template in Xcode

The Xcode IDE provides a template that facilitates the creations of custom patches, this is template is based on Apple's official API for custom patch creation.

To create a new plug-in in the Xcode, in the new project Assistant select Standard Apple plug-in and then Quartz Composer plug-in. Xcode automatically creates the [PlugInName].m and the [PlugInName].h files in the .h we declare the properties of the class, because we are going to build an patch the properties are the input and output ports, the .m file contains the code that is responsible of the behavior of the patch.

3.2.2.1 The .h file

When we create the project the .h file created automatically is empty, to add properties to the class, the process is similar any property in objective C 2.0 language.

The name of the property that maps an input port as to begin with input, and the same way to the ones related to the output ports.

ex:

```
@property(assign) NSDictionary* inputmeshPoints;
```

Here I have created an input port named as `inputmeshPoints` that in the implementation file will be treated as a pointer to an NSDictionary.

If we need to use some other methods (that are not created by default), we have to declare them here (and implement them in the .m file).

Ex:

```
(double) calcY: (double)x :(double) y ;
```

This method is called calcY and returns a double and receives as input two doubles.

3.2.2.2 The .m file

The .m file is were is implement the behavior of the patch, the essential methods are created by default, so to begin to build some plug-ins these are the methods one needs to understand and modify:

```
attributes
attributesForPropertyPortWithKey:
executionMode
timeMode
execute:atTime:withArguments:
```

attributes method

The attributes method return an Dictionary with all the attributes of the plug-in, then each attribute can be accessed by it is key, during my work never used this method, to access the attributes I have used the **self** keyword.

attributesForPropertyPortWithKey method

In this method we specify the ports name in the user interface, and the default, maximum and minimum values for them.

ExecutionMode method

This method is responsible for telling either the patch is a consumer, a provider or a processor.

time mode

This method returns the time mode of the patch, is important to implement this method to define if our patch depends or not on time.

execute:AtTime:WithArguments:

The execute method is where is processed the data from the input ports to the output ports. To prevent the method from became too big and complex is useful to divide it in functions implemented apart (they have to be implemented inside the @implementation tag).

3.2.3 Unofficial API

Besides apple's official API, there also another API that allows the development of custom patches for Quartz Composer.

The Unofficial API as is often called, does not have any public documentation, and is not supported by Apple in any way. The unofficial API allows to the developer to have more control over the Quartz composer (for example it allows to receive and return more types of data from and to Quartz Composer).

Some of the patches used in the development where build with the unofficial API (like the WiimoteControl patch).

3.3 Tools used summary

The development of this thesis was centered in Quartz Composer, it has a vast library of functions but most of them are related to build animations and rendering graphical data.

One way to extend Quartz Composer's functionality is trough the programming patches which allow the developer to code the behavior of the patch. But the most flexible way to extend Quartz Composer is by building patches from scratch using Apple's official API, or the undocumented and unsupported but more flexible unofficial API.

4 Stitching algorithm

Image stitching is the process of joining several different images so that these joined images became a larger representation of a given scene.

A stitching algorithm could be as simple as moving an image next to other, but normally the complexity of the algorithm increases with the degree of freedom one can get when capturing the scene, and with the quality of the result. My approach relies on the use of a marker anywhere in the field of view of the camera, and that is the only restriction to the algorithm (the marker as to be in the field of view of the camera) assuming that the marker detection is good.

To improve the quality of the stitching I have also transformed each image during the capture so all the images remain with the same aspect, resulting in collage is as seamless as possible. The process uses a Web-Camera as the "supplier" for the images and a Wii Remote as the source of all the data needed (the marker coordinates) for the algorithm itself.

The proposed stitching algorithm is divided in the following steps:

A summary of the algorithm can be found in section 4.5.

4.1 Marker detection

The decision of using marker to help me in the stitching was one of the first made in the development, a marker is "something" that could be easily differentiated from the rest of the scene (by his shape, color,...). In my work the marker works as a guide for several steps of the process, and it was really important to keep the cost (computational-wise) of the algorithm low, other approaches that use feature tracking to detect similarities in the pictures are more expensive, and it could be hard to implement real time video stitching with the same degree of freedom as in the proposed solution without the usage of the makers.

The detection of the marker coordinates on the scene is the first step of the stitching algorithm and because of that it is really important to be working well, it influences all the next steps. The proposed solution uses the Wii Remote Infra Red camera as the source of the marker coordinates, because the Wii Remote as a built in blob detection algorithm I can return to the rest of the stitching algorithm the coordinates of each dot of the marker without any additional charge in the CPU.

Next I will explain why I have chosen the Wii Remote to assist me with the process (section 4.1.1), then I lay out how the marker evolve from the firsts prototypes (section 4.1.2) to the final one (section 4.1.3), later I present the tool that allowed me to use the Wii Remote in Quartz Composer, the WiimoteControl Patch (section 4.1.4) in this section I will also explain how the Wii Remote is positioned relatively to the web-camera in order for them to be used together (section 4.1.5), then I present the results of this part of the algorithm (section 4.1.6), and lay out some test that I have made to evaluate the marker detection (section 4.1.7) at the end of this section there's a summary of this step of the algorithm (section 4.1.8).

4.1.1 Using the Wii Remote

After much type spent on building a blob detection algorithm and markers, and with some prototypes working, I have decided change my approach and use the Wii Remote to detect the coordinates needed for the joining. At first I tried the Wii Remote because the other approaches were not very stable and the coordinate detection was no consistent enough for me to build the rest of the algorithm on. And after some tests I have confirmed that the Wii Remote was in fact a big improvement to the coordinate detection.

The decision of using the Wii Remote affect some of the previous work, the marker that I have built were not good anymore, some functions had to be re-written, but the most important thing was that, after testing the Wii Remote for a while I have realized that the

quality of the coordinate detection was enough to use it to something more than the joining (I mean more than simply the image translation), that's when I saw the opportunity for implementing the pose detection algorithm (section 4.2), and using the other data from the Wii Remote to improve the joining algorithm.

Once I started the development with the Wii Remote I immediately started building new markers suitable for the Wii Remote, but before that I have tried different ideas for markers and coordinates detection, next I will briefly lay out some of those ideas, and finally present the final solution.

4.1.2 Markers

4.1.2.1 Card markers

In the early stages of development I've used a card marker that was detected by the web-camera, the marker had colors that could be easily differentiated from the rest of the scene, a highly saturated green and a pink with high hue values.

Next I will shortly present the algorithm:

The detection algorithm first converted the color space of the image received from the camera to HSL, and then it compared the pixel (the HSL value) that was being processed with the marker HSL value, if it was a match to the resulting image pixel value was assigned the white value otherwise the black value was assigned, the resulting image was a black and white image with the white parts representing the markers. Then the blob detection algorithm detected in image coordinates the position of the white parts that represent the position in the markers.

The algorithm presented above suffer a series of improvements, first it could only detect one point in the image (if I wanted to detect 2 points they had to have different colors), but it was refined and when I decided to went with Wii Remote approach it was capable of detecting 16 points (all with the same color) in a image.

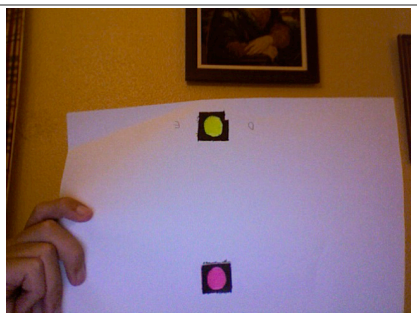
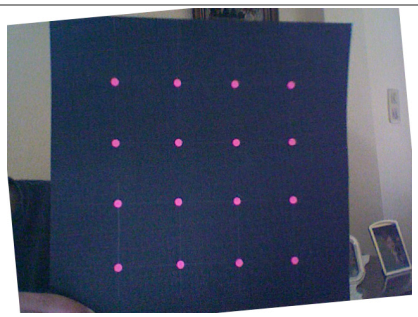
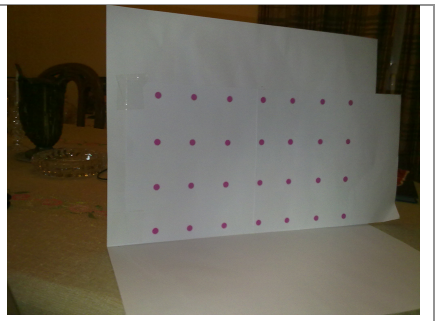




Table 1: Different versions of the card markers

Problems with the card markers approach

The markers were highly sensitive to light changes, the hsl values varied a lot during these changes it was hard to get a consistent detection, the sensitivity of the algorithm was reduced during the development but never removed.

The markers had to be really close to the camera to get a good detection, when I was using 2 cameras it was hard to position the 2 cameras apart from each other but close enough so they could both "see" the marker.

To capture the scene first we had to take a picture from each camera with the markers in position, next the marker had to be removed and the scene was captured with the cameras, the process had to be repeated each time the position of the camera was changed.

The algorithm was expensive (60% CPU charge) to run at 15~20 fps.

The use of the card markers had some advantages to, it's cheaper than the Wii Remote approach (2 Wii Remote 2x39€), and there is no need for computer-peripheral communication that was sometimes buggy.

4.1.2.2 LED light

In an attempt to remove some of the weaknesses of the card maker approach, I have built a filter (with exposed film) which was attached to the camera to filter the majority of the light, this filter was meant to use with led light that worked as a marker, the resulting image would be an image almost all black with a little dot where the led was pointing, this image was later the input for the same blob detection algorithm as the one used in the previous approach.

Figure 14: Camera with and without the filter

Figure 15: Resulting image from the web-camera with the filter

With this approach I almost removed the environmental variables that were present in the previous solution.

Since I only had to detect one point each time the algorithm was faster, but I had to take 2 pictures in each camera to perform the stitching since the rotation and scaling transformation needed 2 reference points in each image/video.

After the joining algorithm was processed I had to remove the filter from each camera, then capture the images/videos, if the position of the cameras somehow changed the all process had to be done again.

On the upside the led marker could be further away from the cameras than the card marker, making that way easier to find a common point between the 2 cameras.

4.1.3 Wii Remote Markers

It easy to notice that the two approaches presented above were never sable or good enough to build the rest of the algorithm on. And also the first prototypes needed a lot of accuracy during the capture, one had to follow a rigid procedure to perform a stitching and if any error occurred during the capture that procedure had to be repeated. That did not meet the goal of an easy to operate and flexible stitching, and those were the main constrains that lead me to research for a better way to perform the marker detection.

One of the reasons that made me choose the Wii Remote as complement to the camera was his capacity to detect up to 4 infra red (IR) sources, so obviously I had to take that into account during the design of the marker, the other constrain comes from the pose detection algorithm which could restrict the marker arrangement in a particular way, and of course one of the main goals of my thesis is a inexpensive to accomplish scene collage so the marker as to be easy and cheap to build and also it has be easy to transport and to place in a particular scene.

Considering the restrictions that I have mentioned above I will explain how the marker is composed:

- Since the pose detection algorithm assumes that the marker as a quadrangular shape the used marker is a rectangle.
- Its made by an aluminum frame and 12 IR LED's, so it cheap to build.
- And like I said before it emits IR light, so the Wii Remote can detect it.

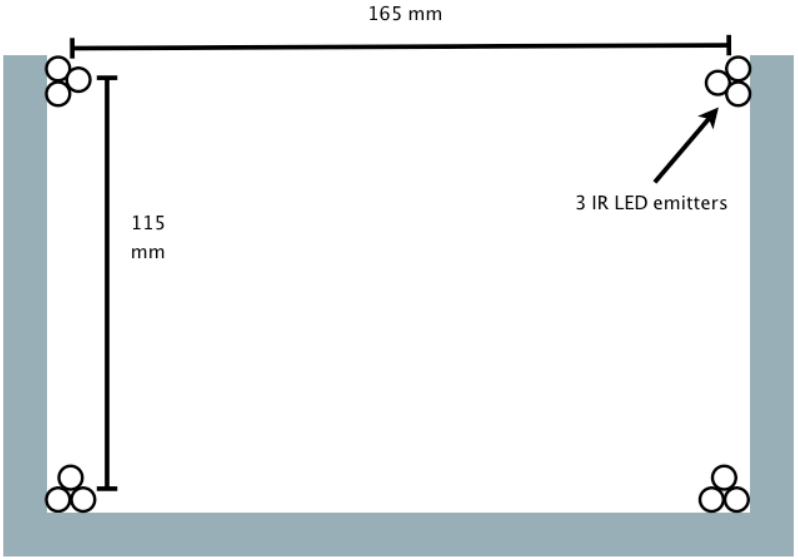


Figure 16: Marker scheme

The Wii Remote marker have also suffer some improvements, at first it only had 1 LED in each corner, the detection was good when the Wii Remote was more or less in front the marker but when the angle from the Wii Remote to the markers increased the detection started to fail. I added 1 more led in each corner which reduced the amount of detection errors, but it still was faulty when the view angles increased so I've added one more IR led to each corner, even with 3 LED's the detection was not perfect with bigger viewing angles. Then I have realized that the problem was that the LED project light to the front, even I increased the number of LED's I would be increasing the amount of light projected to the front improving the detection when the Wii Remote was in front of the camera (which was already good).

So I had to find way to reflect the projected light to sides without ruining the front projection, to do that I started looking for a material that when positioned in front of the LED could do that, and at my first attempt I've found one that was good for the job, if I put a bit of baking paper in front of the LED's the light get spread across the baking paper filter, and that way the Wii Remote could "see" the LED's from higher angles.

Figure 17 Filter positioning

The next images show how the markers evolved until the final prototype:

Figure 18: Different versions of the Wii Remote markers

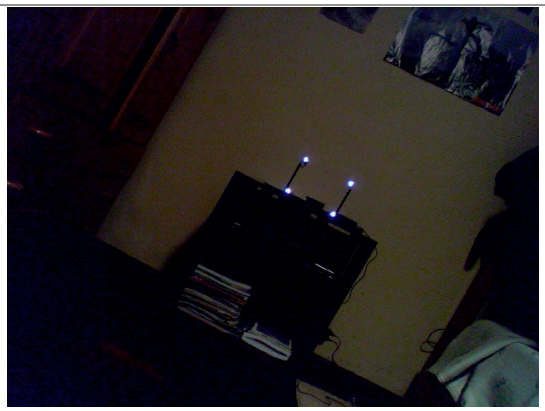
The IR light that comes from the Wii Remote could be bad for the quality of the resulting stitched images/videos, since it can be also detected by Web-Camera, this is more

noticeable if the camera is very close, or if the scene is to dark examples of this phenomena can be seen in almost all the stitching examples trough this thesis.

I have reduced the area of the marker as much as I could, the IR light detected by the camera is right now the main thing that denunciates the presence of the marker.

The IR light could be restricted from the Web-Camera if I used a filter that only allowed visible light to pass, or a infra red cut filter, this type of filters are available in the market, I have not used one in my work, that is just a simple addition to the cameras with no need for any changes in the algorithm.

The next image shows how the filter could remove the effect of the IR in an image, the example is just 2 pictures taken from the same position the first one with the maker on and in the second one off.



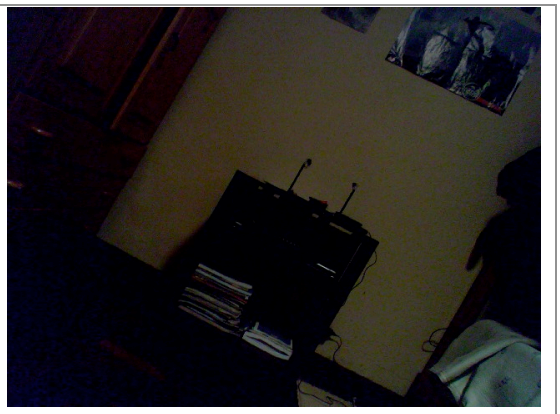


Table 2: Simulation of the effect, of an infra red filter in a scene

4.1.4 Wiimote Control Patch

All the work that I have mentioned above would be in vain if I didn't have a way to get the values from the Wii Remote to the Quartz Composer, that task is accomplished by Wiimote Control Patch.

The WiimoteControlPatch was built based on the Darwin Remote Framework (Hiroaki, 2008), which is responsible for all the computer-peripheral communication and data handling.

I have used the version 0.2 of this patch, at first this version didn't provided the values from the Infra Red Camera, so I had to made some changes to the patch to be able to get all 4 values

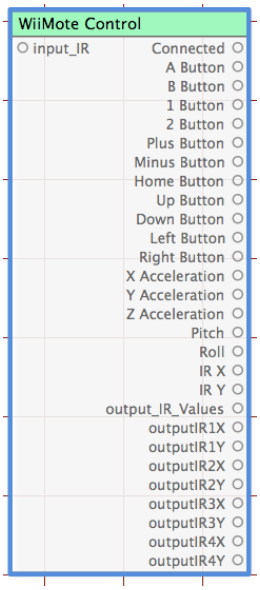


Figure 19: WiiMote Control patch

The main outputs of the Wiimote Control patch are:

OutputIR (1X~4Y): These are the coordinates of the infrared sources detected by camera, these coordinates belong to the interval XX:(0~1024) and YY (0~768) range.

Roll: The Roll value is the rotation of the Wii Remote according to a coordinate system parallel to the front (or back) of the Wii Remote and has as origin the center of the Wii Remote.

Pitch: The pitch value is the rotation of the Wii Remote according to coordinate system parallel to the sides of the Wii Remote and as origin the center of the Wii Remote.

Acceleration Values: These values came directly from the accelerometers, and they are used to calculate the roll and pitch, this calculation is made inside the patch.

Buttons: The Wiimote control patch also provides the state of all the buttons in the controller, so they can be used like any other control type of patch in Quartz composer.

4.1.5 Wii Remote positioning.

I haven't talked yet how I will use with the Wii Remote with the web-camera, while the Wii Remote is responsible for tracking the marker, the Web-Camera as to capture the scene. They should have some relationship so that they are point at the same portion of the scene at the time.

The Web-Camera is attached in top of the Wii Remote:

Figure 20: Capture method

From the picture above is obvious there will be a deviation from what the Wii Remote is "seeing" to what the web-camera is "seeing", that deviation is accentuated if remember that the image sensors in those 2 cameras are completely different and have different specifications.

The difference between the coordinates of a certain point on the 2 devices is very important because the joining is made based on a point detected from the Wii Remote, but the images are captured with the web-camera, so I have to discover where the point detected with the Wii Remote is really situated on the image in order to make the joining as seamless as possible, a solution to this problem is discussed in section 4.4.1.2.

4.1.6 Marker detection results

Next I will show different results of marker detection from different distances and angles, the images at left represent the 4 points of the marker and the image at the right show an image from the view point of the Wii Remote (using the same method as shown in Figure 20).

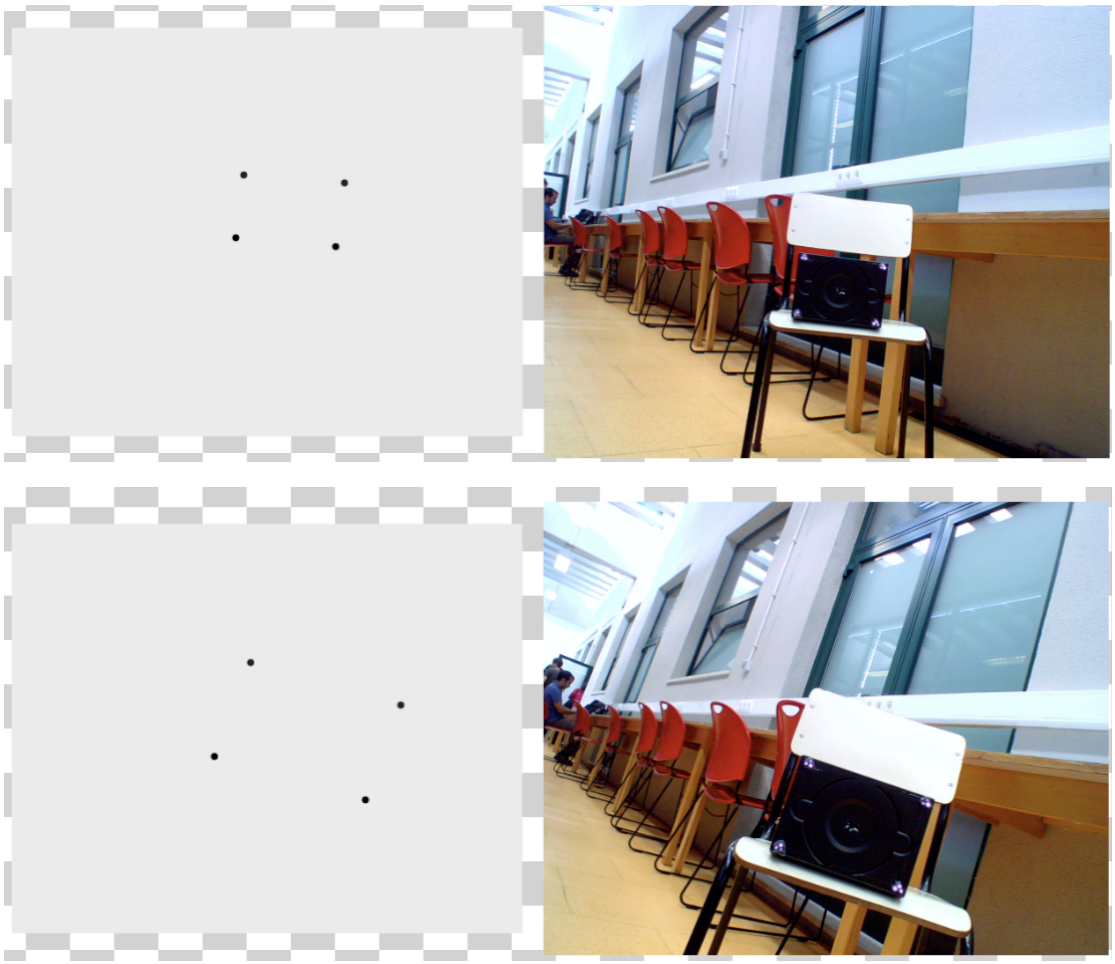


Figure 21: Different examples of the marker detection

From the examples above we can conclude that the detection was good enough for me to build the rest of the algorithm using the WiiRemote as the coordinates provider, the detection performs well in different distances and viewpoints angles.

4.1.7 Wii Remote marker detection evaluation

To evaluate the marker detection by the Wii Remote I have tried to detect the marker from different distances and I measured the errors.

In the meaning of the test an error is any point (from the 4 dots of the maker), which is not detected, or if the coordinates returned is not right. These tests were made to confirm if the baking paper used was in fact an improvement to the detection.

To perform this evaluation I have repeated the detection from the same position with the marker with and without the baking paper and registered the results.

After collecting a sufficient amount of data I have organized the results in the following table.

Camera position				Errors	
Distance (cm)	X-Rot	Y-Rot	Z-Rot	Without filter	With filter
74	0	5	0	0	0
103	12	0	51	2	0
120	2	23	9	1	0
120	22	10	8	0	0
130	3,6	5,3	4,05	0	0
140	10	30	0	2	1
140	2	40	2	3	2

From the results above, we can notice that the baking paper filter was in fact an improvement to the detection, mainly when the rotation values of the WiiRemote increased, it is also noticeable that the misdetections can still happen even with the filter.

4.1.8 Marker detection summary

From the beginning of the development I have decided to work with a marker on the scene to help in the stitching process, the marker is “something” that could be easily differentiated from the rest of the scene by some particular features (like in the final marker which is differentiated by the IR light).

In the first stages of the development I have used the same web-camera used to capture the scene to detect the marker, this detection was made by a blob detection algorithm which I built, and it among other things in rely pixel comparison, this comparison was very sensitive to luminosity changes but after some improvements in the blob detection algorithm I have accomplished my first (simple) collage.

Even tough the blob detection algorithm evolved, it was never stable enough for me to build the rest of the stitching algorithm on, so I have decided to use a Wii Remote more precisely his IR camera, to get a cleaner detection that decision influenced the construction of the markers which had emit IR light to be detected by de Wii Remote.

The usage of t a marker was really important to keep the algorithm inexpensive (computational-wise) otherwise it would have been very hard to perform video stitching in real time.

4.2 Camera Pose detection

Camera pose detection is the process of discovering the position and orientation of a camera in space relatively to reference points (the markers), this is a classic computer vision problem with several solutions proposed, using 4 dots this problems became is a closed solution meaning it can be solved analytically with no need for iteration that could be expensive computational-wise, I have implemented the method in (Chandra, 1989) which is supported by the marker and the Wii Remote, the rotation values returned from the Wii Remote were also used to improve the pose estimation.

The rest of the section is organized as follows, first I explain why I have decided to add a pose detection algorithm to the stitching process (section 4.2.1), next I lay out how I have implemented the algorithm in (Chandra, 1989) his restrictions and results (section 4.2.2), then I justify why I have combined the roll and pitch values from the Wii Remote with the algorithm that I have implemented before (section 4.2.3). The pitch a roll values received from the Wii Remote have to be adjusted in order to work better with the image transformation algorithm (section 4.2.4), later in this section I demonstrate the pose determination algorithm with some examples and evaluation (section 4.2.5), at the end of this section I have done a summary of this step of the algorithm (section 4.2.6).

4.2.1 Why in my work

The pose detection is probably the single most important step to improve the quality of the stitching (in the proposed solution) because, when one knows from where the images are taken that information can be used to transform the images before joining them, the goal of this transformation is to remove the perspective distortion of an image, the next example shows this phenomena.

The next image is the result of stitching two images, in the first one there were only applied 2d translations, to the second one there was applied a 3d transformation and translation based on the pose estimation.

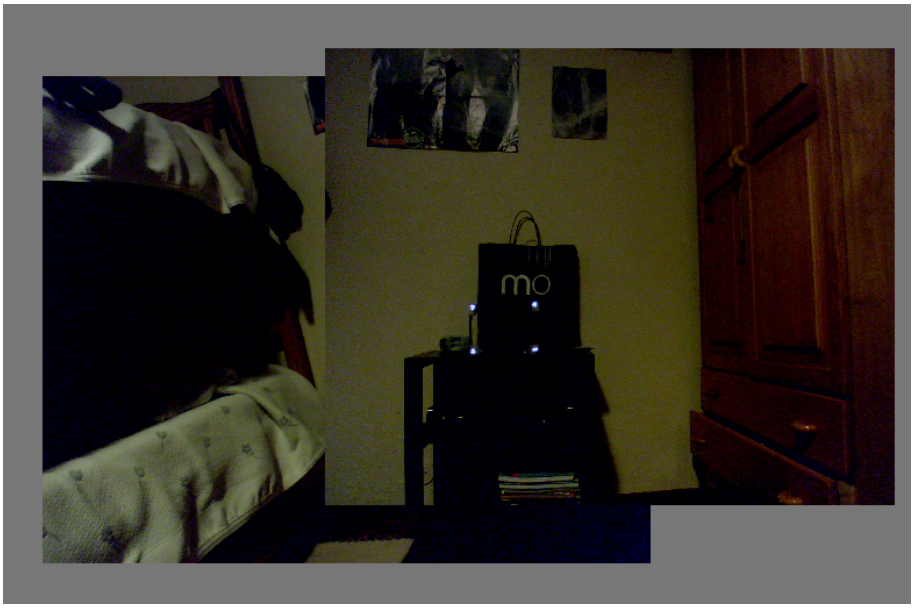


Figure 22: Image without transformation based on pose estimation

The perspective distortion is evident in this image, even thought it does not ruin the perception of the scene it surely reduces it.

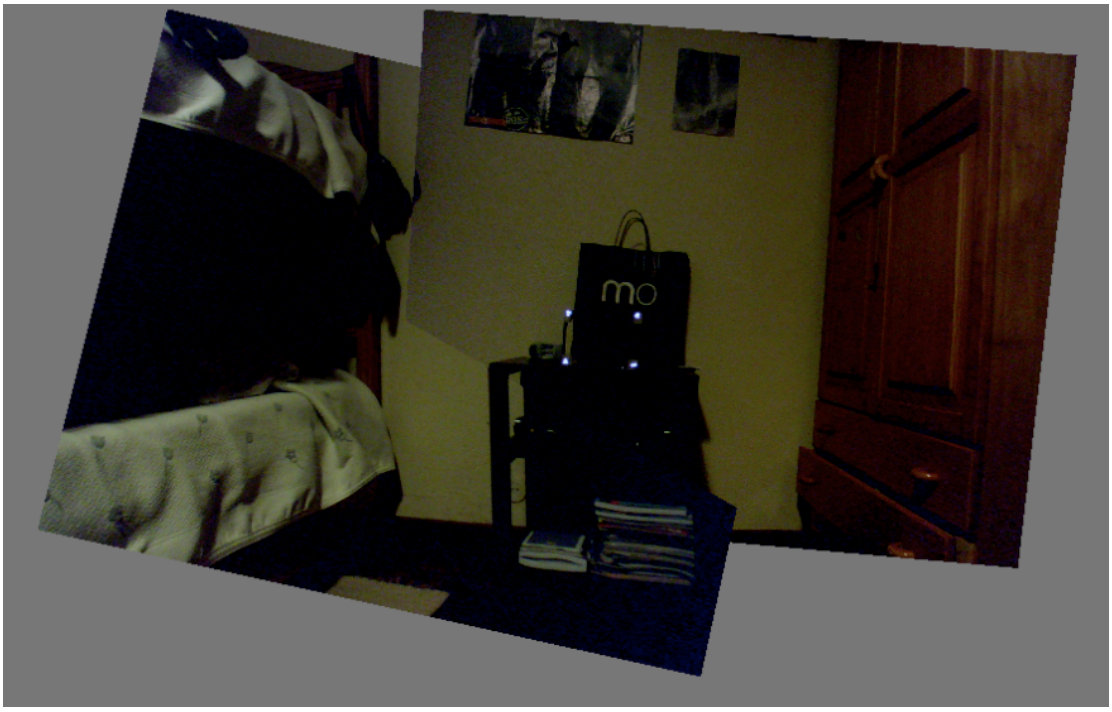


Figure 23: Image with transformation based on pose estimation

With transformations made, it's easier to view details, we can notice that the aspect of the image and the size ratios between the objects on the scene are more real than the ones in the first image.

This type of transformation is made easier once the position of the camera relatively to the scene (marker) is discovered.

The result of a complete pose detection algorithm is:

- Angles of rotation between the camera and the marker in the in X, Y and Z-axis.
- Position of the camera in the X Y and Z-axis.

From all the values returned from a complete the pose detection algorithm, in my work I only use the z position (used in the size transformation section 4.3.2), and the 3 rotations (used in the 3d transformation section 4.3.1), the image translation is calculated with the reference point from the marker.

4.2.2 Analytic solution for the camera pose detection

There are several solutions to discover the position and orientation of the camera in the real world relatively to a marker, I have used the algorithm in (Chandra, 1989), for 3 main reasons:

- It could be developed step by step, that way I was easier to detect errors and to isolate some portion of the algorithm
- In (Chandra, 1989) the authors have provided synthetic data which was useful during the coding/testing process (I could verify if the algorithm was correct using the data from the paper before continue the development)
- This is a closed form algorithm there was no need to perform expensive (in computational power needed), iterative operations.

4.2.2.1 The algorithm

Restrictions, the camera pose detection algorithm as to receive:

- **Position of each vertex of the quadrangular marker in mm:** the 4 points have to lie on the same plan. I have assumed that all the points have 0 value on the z-axis, the marker is located entirely in the xy plan. I assigned to the lower left point of the marker the origin of the referential (Figure 24).
- **The Focal length of the camera**: this had to be an approximated value since I did not have any access to the Wii Remote IR camera specifications.
- **The coordinates of the 4 points in mm:** this coordinates are the physical position of the points in the image sensor (of the Wii Remote).

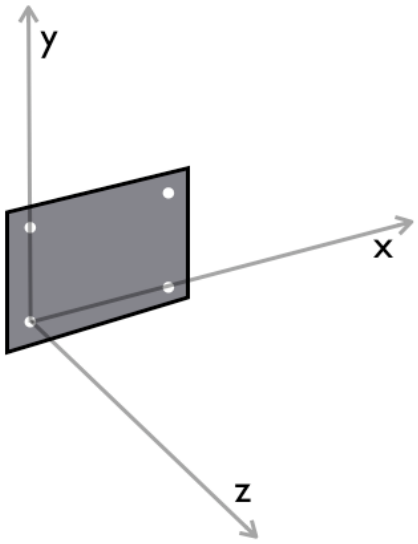


Figure 24: Real space coordinate system used for the input and output of the pose detection algorithm

Pixel to mm conversion

Since the pose detection algorithm that I have implemented required that the coordinates of the marker dots were given in mm (the physical position in the image sensor), but the WiiRemote Control patch only returned the dots coordinates in pixels (xx: 0~1024, yy:0~768), I had convert them to millimeters.

To accomplish that I had to discover the specifications (sensor size and focal length) of the Wii Remote IR camera, but since Nintendo does not provide them, I have researched through PixArt (Builder of the IR sensor in the Wii Remote) catalog, trying to find the Wii Remote IR sensor, that did not helped much because the pixel values returned from the Wii Remote control patch do not reflect the actual position of the pixel. The sensor on the Wii Remote is much smaller than 1024*768, so the only think I could rely was the relative dimensions of the sensor:

Width=(1/3)*height

From that I have assumed a 1’/4’’ sensor size, so it had 3,60mm width and 2,70mm height and I have used the same value for the focal length as (Timo, 2009), that assumption did not jeopardize the algorithm since the focal length is also an input value for the pose detection algorithm and the sensor size only have to be relative to the focal length for the pose detection algorithm to work.

Finally I have used the next 2 formulas to execute the conversion

```
(sensor_width/2)-((pixel_x*sensor_width)/1024)
(sensor_height/2)-((pixel_y*sensor_height)/768)
```

Results

The pose detection algorithm returns the following data (according to the coordinate system in Figure 24):

- X position relatively to the marker.
- Y position relatively to the marker.
- Z position relatively to the marker.
- Rotation on X-axis.
- Rotation on Y-axis.
- Rotation on Z-axis.

4.2.2.2 The pose calculator patch

The camera pose detection algorithm was implemented in a specific patch (this patch was developed using apple official API). The camera pose determination consist of a series of mathematical formulas, by grouping all this computing in one patch improves the performance the modularity of my compositions, the patch look like:

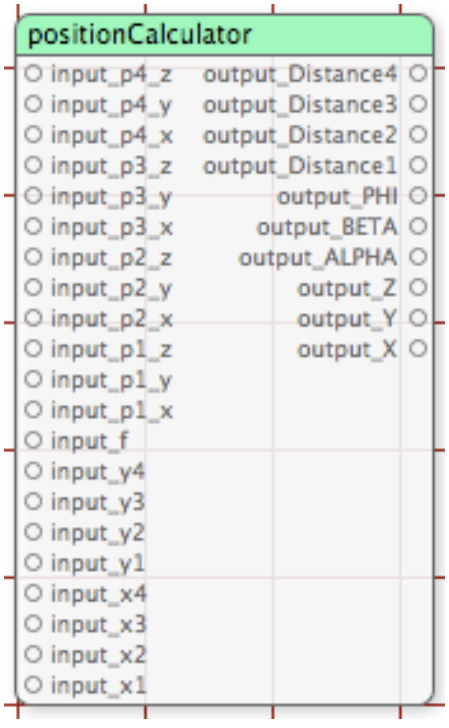


Figure 25: PositionCalculator patch

Inputs:

input_p(1~4)_(x~z): these are the physical position of the marker points in space these values are inserted in millimeters.

input_f: focal length of the camera.

input_(x~y)(1~4): positions of the marker points in the camera sensor, these values are also in millimeters.

Outputs:

output_Distance(1~4): distance from the camera to each point of the marker in millimeters.

output_(PHI/BETA/ALPHA): values of the camera rotation in the 3 different angles.

ouput_(X/Y/Z): position of the camera in real space also in millimeters.

4.2.3 Wii Remote approach for the pose detection

Even though we cannot obtain the all results as the ones mentioned above, with the Wii Remote we can obtain some of the values needed for the pose detection, the Wii Remote returns de Roll and pitch (the Z and X-rotation).

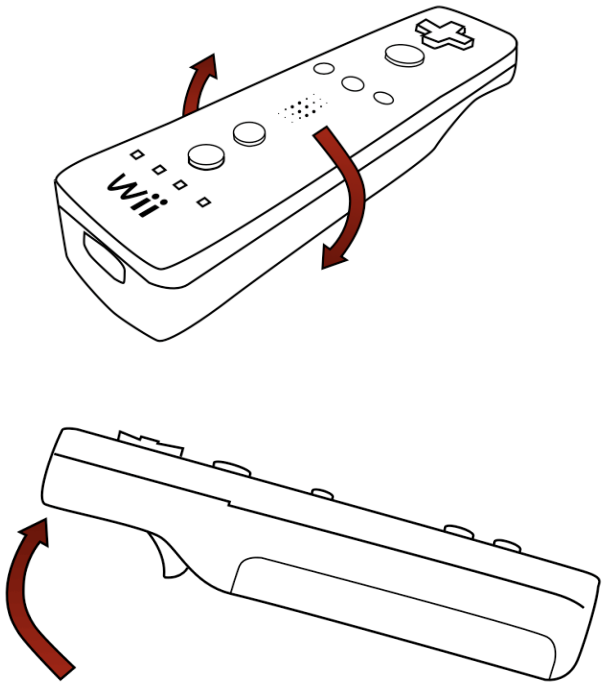


Figure 26: Different types of rotation detected by the Wii Remote on top the roll (Z-axis rotation) and at the bottom the pitch (X-axis rotation).

So instead of using the Z and X-rotation from the pose detection algorithm I have decided to use those rotation values from the Wii Remote, there was no need to calculate those rotation in the pose detection algorithm and also the values from the Wii Remote

Control patch are processed even if they are not being used, so there's no extra CPU charge needed to get those values.

During the tests I have also noticed that the values from the Wii Remote are more stable than the ones from pose detection algorithm, mainly because one of the inputs of the pose detection algorithm are the points coordinates that are "read" from the Wii Remote IR camera which sometimes fail (if there are no IR source near by, if the source is too weak, if there are too much sunlight in the area,...), if we add to that the fact that the pitch and roll values from the Wii Remote only fail when de connection is lost, the final result much more stable.

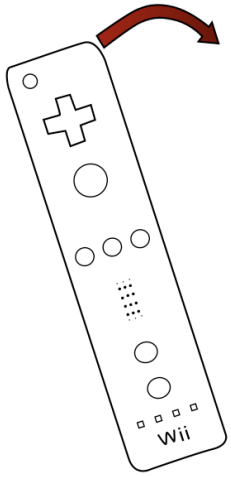


Figure 27: The Y-Axis rotation it's the only rotation that I have used from the pose detection algorithm

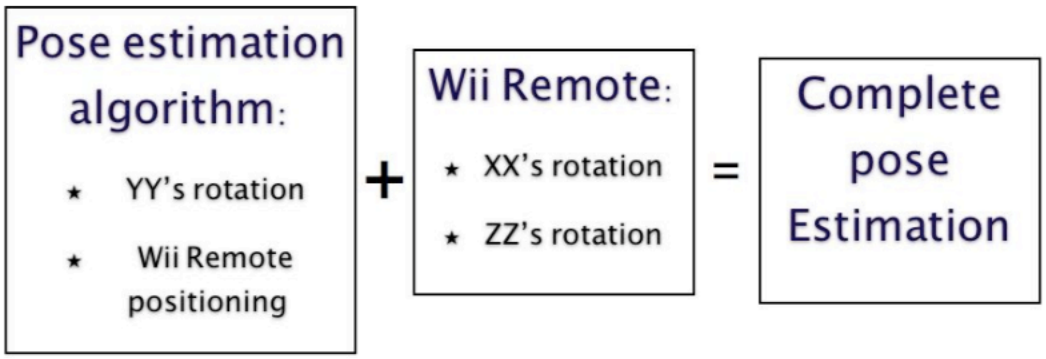


Figure 28: Pose detection process

4.2.4 Pitch and Roll values

When I have started to test my "hybrid" pose detection algorithm I have come across a problem right at the beginning, the roll value from the Wii Remote was not the same as the camera, mainly because the center of rotation of the Wii Remote is different

(the Wii Remote is on top of the camera). Later I have also noticed that the pitch value (from the WiiMote control patch) did not reflect accurately the X-Rotation of the WiiRemote.

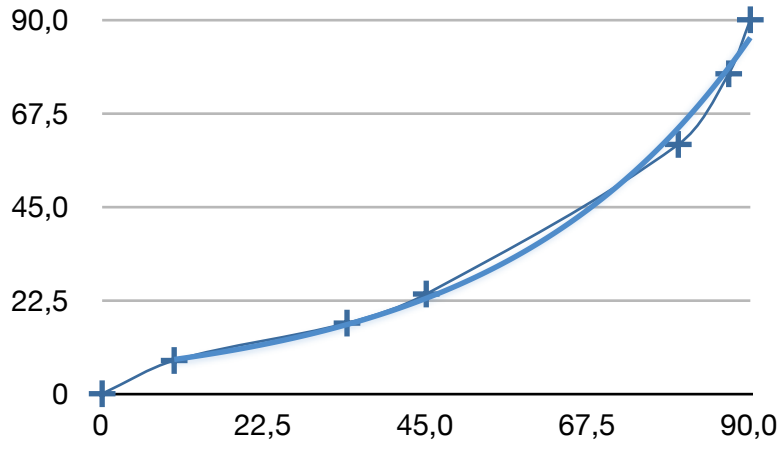
I have decided to build two tests, to collect enough data to help me to analyze the relationship between the real and measured Rotations

4.2.4.1 Roll

The test consisted in attaching the Wii Remote on top of the camera, and then pointing the camera to a horizontal line or shape. If the Roll values from the Wii Remote matched the ZZ rotation of the camera, when I added to the image from the camera the inverse of the Z rotation measured from the Wii Remote, the horizontal line should have stayed horizontal, that did not happened so I manually changed the angle of the image until the line was horizontal again, I did that for set of values, my goal was to find a pattern in the relations between the Wii Remote values and the manually attributed ones, that could be expressed in a function.

After some tests to collect a sufficient amount of data I built the next graphic that represents the relationship between the Wii Remote roll value and the camera Z-Rotation.

```
Zrot = 4.1563*e^(0.0293*Roll)   *
```



Graphic 1: Relationship between the roll and the Wii Remote rotation values

*The function was only used for rotation values bigger than 4.1563 and smaller than -4.1563.

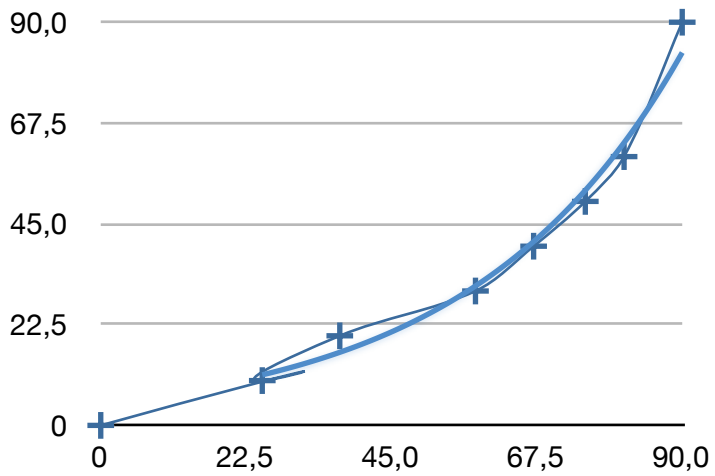
4.2.4.2 Pitch

I had to analyze how the pitch value from the WiiMote patch was related with the real pitch of the Wii Remote (and thus the camera, since the camera is located on top o the

Wii Remote). So I have built a simple test that consisted in measuring the real pitch of the Wii Remote and then compare with the returned value.

After I have collected some samples I have organized and in a spreadsheet and I got the following graph and respective formula:

```
Xrot = 5.231*e^(0.0307*pitch)    *
```



Graphic 2: Relationship between the pitch and Wii Remote X-rotation

*The function was only used for rotation values bigger than 5.231 and smaller than -5.231.

4.2.5 Pose detection evaluation

Next I will compare the real position and orientation of the WiiRemote, with the values that I got from the Pose detection algorithm that I have used. The real position of the Wii Remote was easy to measure, but the real orientation (3 angles) was a bit harder.

With the following tests I have tried to evaluate how the algorithm behave, I was mainly concerned about the traditional pose detection because it was sensitive because, I had to estimate the value of the sensor size and focal length, and also because from previous tests I had noticed that the rotation values that I got from the Wii Remote were accurate.

Because it's hard to get the real values of parameters of the pose detection I have decided to collect individually each one, I have only tested for the values that I have used in the video/image transformation the Z position, and 3 rotations.

4.2.5.1 Z position results (in mm)

The Z position is related with the size of the image in the image transformation algorithm, this test was by simply measuring the "perpendicular" distance from the Wii Remote to the marker.

Real position	Algorithm Result	Difference
155	160	-5
144	152	-8
132	135	-3
110	114	-4
97	94	3
80	75	5
70	72	-2
60	63	-3
42	48	-6

Table 3: Z position evaluation results

Average Difference **4,333 mm**

The pose detection algorithm returns the Z position of the camera rather well, I got 8 mm as the biggest difference, besides the accuracy of the algorithm the other thing that is also important is the stability of the values returned, the results were consistently right.

4.2.5.2 X Rotation

The pitch received from the WiiRemote is used as the input for a function which adjust his value to the real X rotation of the WiiRemote, since I have built that function is expected for the next test to be good, but I did the test so that I could confirm my expectations:

Measured Angle	Calculated Angle	Difference
0	0	0
5	6,02	-1,02
10	11,34	-1,34
20	20,42	-0,420
25	25,2	-0,2

30	31,6	-1,6
35	34,2	0,8
40	41,49	-1,49
50	56	-6

Table 4: X-rotation evaluation results

Average difference **1,43 degrees**

As it was expected the errors in this tests were small, that happens mainly because I have built a function (section 4.2.4.2) that adjusts the X rotation values returned from the Wii Remote to the real values, this evaluation confirmed that the mapping function was returning the expecting values.

4.2.5.3 Y rotation (in degrees)

The Y rotation is the only rotation value that I've used from the traditional pose determination algorithm implemented, and I was particularly worried about the results in this test, because in early test this parameter was unstable. The next table lists the results of the Y rotation evaluation.

Measured Angle	Calculated Angle	Difference
-17	-19	2
-11	-12	1
-10	-10,891	0,891
0	0,15	-0,15
6	8	-2
28	31,6	-3,6
10	11,3	-1,3
20	18,496	1,504
25	23,643	1,357

Table 5: Y-rotation evaluation results

Average Difference **1,533 degrees**

The difference in the Y rotation is also small and when the detection is good the returned result is accurate, the main problems happens when the detection fail even a bit the results vary a lot that's why I had to use Quartz Composer's smooth patch to smooth the Y rotation value from the pose detection algorithm.

4.2.5.4 Z Rotation

I've also did a test to evaluate the detected Z rotation, the rotation value is not used straight from the Wii Remote, first it's the input of a function that adjusts his value to work better with the image transformation algorithm (section 4.2.4.1).

Measured Angle	Calculated Angle	Difference
0	0,08	-0,08
10	9,6	0,4
20	19,8	0,2
30	28,6	1,4
40	39,5	0,5
50	52	-2
60	61,02	-1,02
70	71,6	-1,6

Table 6: Z-rotation evaluation results

Average difference **0,9 degrees**

Like in the X-Rotation test, this test was made mostly to test how well the conversion function (section 4.2.4.1) mapped the real Z-Rotation of the Wii Remote. The results showed the expected, the mapping function is working well.

4.2.6 Pose Detection summary

The pose detection algorithm is the process of discovering the position and orientation of a camera in real space based in a series of previously known points correspondences (between the image in the camera and the real world), with 4 points this is pose detection becomes a closed form problem meaning that it does not need iteration to be solved.

I have implemented pose detection in the proposed solution because the results returned from it (3 rotation and X, Y, Z position) helped me to transform the images before they were stitched, so that the overall quality of the stitching increased. The pose detection algorithm used, is as combination from a traditional pose detection algorithm, and the rotation values from the Wii Remote (pitch and roll).

Tests proved that the implemented pose determination is returning a result which is accurate enough to be used by the rest of the algorithm.

4.3 Images transformation

The purpose of the pose detection algorithm is to transform the image that is being captured, according to the algorithm output. So once one knows the position and orientation of the camera relatively to the scene, we can transform the image.

The goal of this transformation is to change the general aspect and look of the images, so it can be presented the user in a more natural way.

Once the orientation of the camera is received from the pose detection algorithm the images are transformed so that the marker in the resulting image remained with same aspect even when the scene is captured from different positions, to accomplish that the algorithm rotates the images in the 3 axis. And a size transformation is also applied to each image so that the size of the components of a scene have the same relative size even if the images/videos are being captured from different distances

In this section, first I lay out how I applied the 3d rotation to an image from the camera based on the results from the pose determination algorithm (section 4.3.1), the second transformation applied to each image is the size transformation, and later in this section I explain how that transformation is applied (section 4.3.2). In section 4.3.3 I illustrate the transformation algorithm with some examples.

At the end of this section there's a summary of this step of the algorithm (section 4.3.4).

4.3.1 3d Rotation

All the images suffer the same transformation, once I get the orientation of the camera I rotate the image in the opposite angle as the one received from the pose detection algorithm that way, the images remain the same aspect.

Since I'm rendering the images in a Sprite patch, and because it provides the inputs to change the X, Y and Z-axis rotation I can directly change the image rotation with no need for pixel-by-pixel calculation. Other approaches like using 3d affine transformation to change the aspect of the image could return best results, but they are also more complex and need more computational power so, my decision to go with simple 3d transformation even though is not the best to accomplish a really high quality collage, is in my opinion a good commitment between performance and results.

4.3.2 Size transformation

Because the 2 images could be captured from different distances I have to transform the images according to their distance, there were 2 obvious solution from this transformation, I could move the image in depth according to the distance from the camera or I could change the size of the image maintaining that way his depth.

I've decided to go with the second solution because it's easier to join 2 images if they have the same depth (same Z values), if I want to present the scene in any other perspective than right in the front of the viewer. The different depths of the images could reduce the smoothness of the stitching as the next example shows.

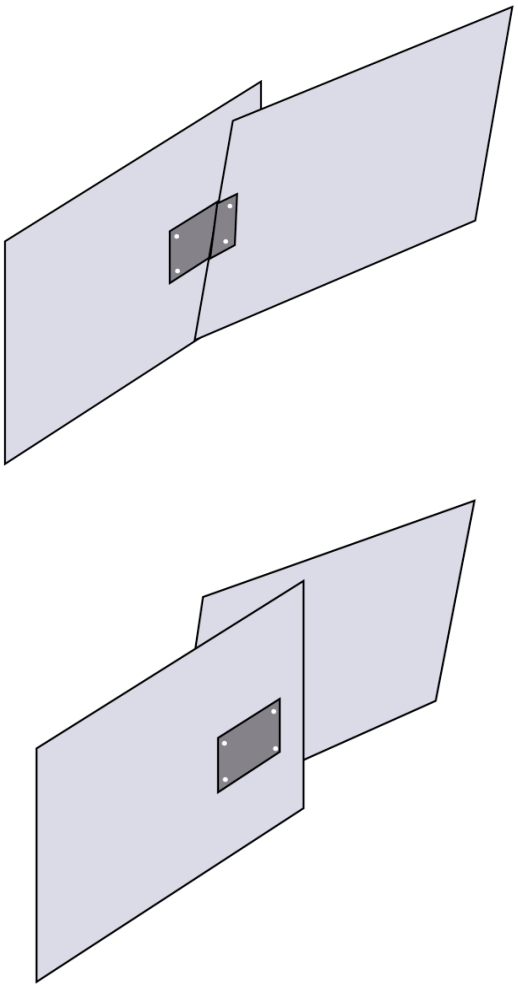


Figure 29: On top stitching with the distance represented in the size of the image and at the bottom represented with the z position of the image

4.3.2.1 Calculation

Before I could calculate the new width and height of each image I had to assign a width value to a distance, which worked as a reference value for the rest of the calculation, after a few tests I have assigned the width value of 1 to images captured from 1,50m of distance the new width and height are calculated as follows:

```
newWidth = 1/(1,50/Distance);

*newHeight = 0.8*newWidth;
```

*The new height is calculated just by this multiplication in order to maintain the image aspect.

4.3.3 Image transformation results

Figure 30: Image straight out of the web-cam

Figure 31: Image transformed based on the pose detection result

Figure 32: Two pictures of the same scene taken from different distances and viewpoints

The transformation made in the first example (Figure 30 and 31) is just a simple 2d rotation so that the marker in the transformed image is horizontal after the translation, that way the image is ready to be stitched with another one.

In the second example the transformation is more complex, the pictures were taken from different distances and the view point have also changed, so in this example all the variables associated with an image (position, 3 axis rotation, and scaling) have changed in order to maintain the general position and aspect of the marker, the transformation in this example have prepared the images for an eventual stitching with any other one.

4.3.4 Image transformation summary

To each image were applied several transformation with the goal of preparing the images before the stitching itself, so that the images are presented with a more natural aspect, and the stitching becomes smoother.

Once the position and orientation of the camera is discovered, each image suffers a 3d transformation. They are rotated in the opposite angles as the ones returned from the pose detection algorithm, that way some of the perspective distortion is removed. To each image is also applied a size transformation to simulate the distance from where the scene is being captured, the size transformation levels the size the 2 images so that the stitching is good even tough the cameras are at different distances from the marker.

4.4 Images translation

This is the final step of the stitching algorithm and at this stage the images are already transformed, I just have to join them i.e. apply a translation in the X and Y and Z so that the images are stitched according to a reference point.

First in this section I lay out some constrains that I had to take in account before the translation, next I explain how I overcome those constrains with several coordinates conversions (section 4.4.1), later I show how I did translation and why I choose the image positioning used (section 4.4.2), then I present some examples and results of the image translation algorithm (section 4.4.3). To evaluate the image translation and the coordinates conversions I have built 2 tests (section 4.4.4). Finally at the end of this section I have made a summary to summarize this step of the algorithm (section 4.4.5).

4.4.1 Restrictions

The reference point for the joining is selected from the Wii Remote IR camera, but if I only moved the images according to these values my collage would became bad, for 2 main reasons:

- The images suffer a series of transformation so obviously the reference points are not at the same position as before the transformation, to solve that I have applied to the Wii Remote points the same transformation as the one I made in the image points.
- The Field of View, resolution and position of the cameras is not the same as the Wii Remote, so the reference points are not exactly at the same position in both devices.

4.4.1.1 Pixel to Quartz Composer units conversion

Since all the transformations are made in quartz composer I had to convert the markers positions and all the other necessary data to QC units.

The visible area in the QC viewer area goes from -1 to 1, in the xx and yy axis, so once a got a value from the Wii Remote I had to do the following conversion

```
qcX = (((wiiX * 2)/1024) - 1)
qcY = (((wiiY * 1.3333)/768) - 0.6666)
```

The X's conversions is straightforward, in the Y's conversion the only think that I had to take in account was that the aspect of the Wii Remote sensor in 4/3, that means that if in QC units the Wii Remote X's values will change from -1 to 1 covering 2 units, then the Y's values had go from -0.66666 to 0.66666 (covering 1.333333 units).

4.4.1.2 Wii Remote pixels to camera pixels conversion

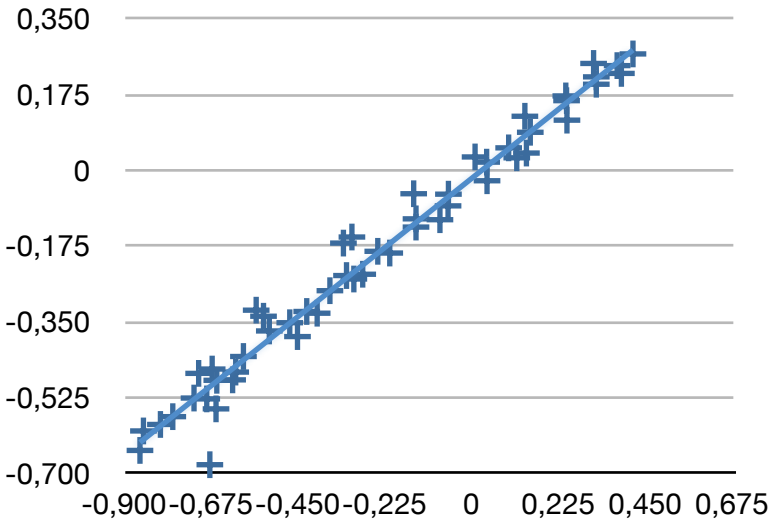
I have to stitch 2 pictures captured from 2 web-cameras, but the all the data used in the stitching is received trough the Wii Remote IR camera. These two cameras have different resolutions, pixels size, field of view, and if we add to that the fact the position from where the scene is captured is different (the camera is on top of the Wii Remote), at least a deviation in the Y's position was expected, so it's obvious that the coordinates of the points in the two peripherals it's different.

My experience in the early tests showed me that the relationship between the coordinates in the two devices seamed to be linear. So I made some further tests and collect a sufficient amount of data to confirm my suspicions, the relationship between the 2 coordinates are expressed in the next graphics.

Later I have used the resulting equations to convert the coordinates from Wii Remote to Camera).

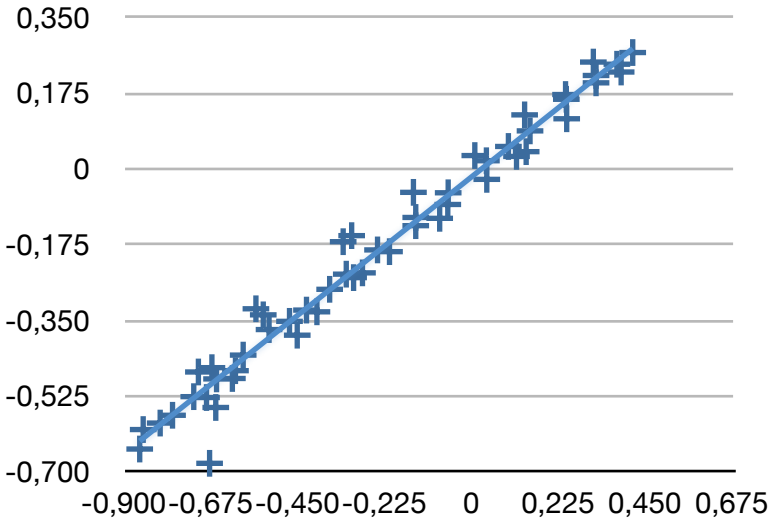
With the Wii Remote and web-camera pointing in the same direction, I collect the coordinates in the 2 devices of the same point, the process is repeated trough all the X's and Y's axis, 53 samples were taken, the next graphics represent the relation between the camera coordinates and the Wii Remote coordinates.

$$CamX = 0,7099*WiiX - 0,0175$$



Graphic 3: Relationship between the X-axis coordinates in the Wii Remote and the web-camera.

CamY = 0,7922*WiiY -0,125



Graphic 4: Relationship between the Y-axis coordinates in the Wii Remote and the web-camera.

After building those 2 graphics I got 2 functions that allowed me to convert between camera pixels and Wii Remote pixels.

4.4.1.3 Points Rotation

Like seen in section 4.3.1 to all images/videos were applied 3 transformations (one rotation in each axle), this rotation will affect all the points in the images so, it will also affect the point that I'm using as a reference for the translation. The point used as a reference is received from the Wii Remote IR camera his coordinates are converted o camera coordinates and then finally they are converted to QC coordinates, to find the new position of the point after the transformations I have followed the next procedure:

If θ is the rotation in the xx axis, and x, y, z are the current coordinates the new position of is calculated by the following multiplication:

$$\text{Rotation}_{zz} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

The new positions after the yy and xx rotation are calculated the same way

$$\text{Rotation}_{yy} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\text{Rotation}_{xx} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

To calculate the above equations I have build a macro which I added to the Quartz Composer patch library, that way it's easier to perform this task anytime I want, this macro contribute also to a cleaner look of the composition.

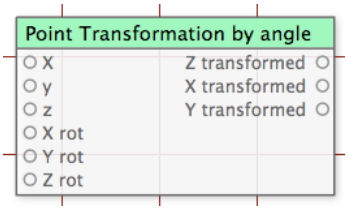


Figure 33: Point Transformation by angle patch

4.4.1.4 Size transformation

To the images/videos was also applied a size transformation, like the rotation transformation this one also changes the coordinates of the points. By default the conversion from pixels to QC units it's made assuming that the image has the same width as the viewer window (2 QC units).

The goal of the following conversion is to change the reference point coordinate as the image increases or decreases.

```
imageRatio = newWidth/2
transformedX = X/imageRatio
transformedY = Y/imageRatio
```

4.4.2 Image translation

After I got the right reference points I have to move a image next to the other that way finishing the stitching.

In the first tests the translation was made by fixing the first image and then moving the second towards the first one, that method was not really good during the capture process, because the joining was not always made in same area of the viewer window, it was harder to track errors, and even the capture process itself was harder.

The final translation process is made in a way that the reference point in both images is always positioned in the middle of the viewer window, don't matter from where the image/video is captured, that way it's easier to detect any error during the capture because the part where the majority of the errors could occur is always at the same position.

The stitching is accomplished by a simple translation in the 3 axis, but first I will present the QC unit system and how an image is positioned in the viewer window:

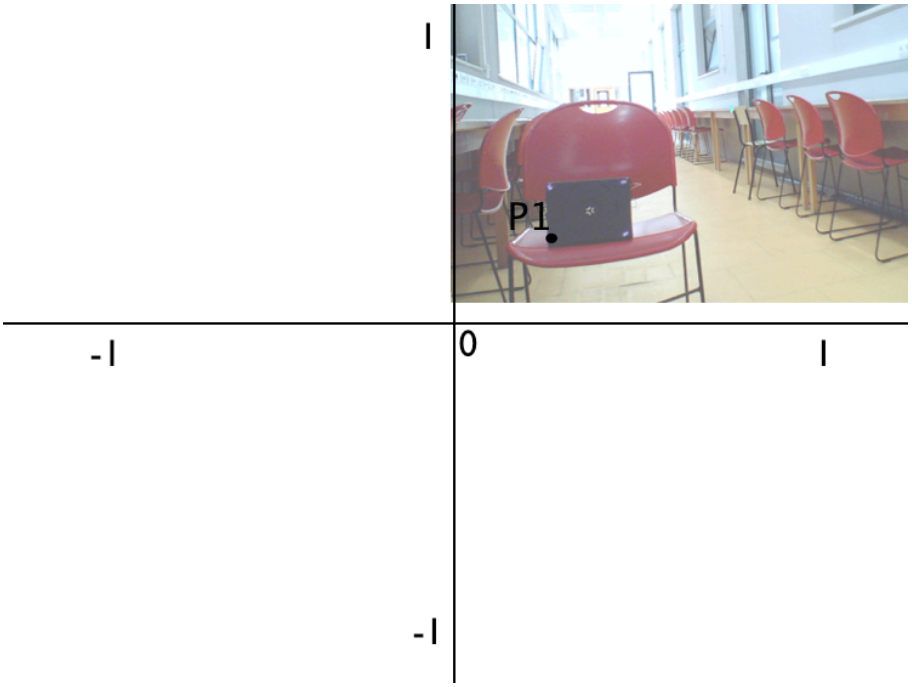


Figure 34: Image presentation in the viewer window

To “bring” the images to the middle I have to shift the images the same amount as the opposite of his X’s Y’s Z’s coordinates the translation is made using a point from detected from the marker (the point used in the translation is the P1 represented in the images).

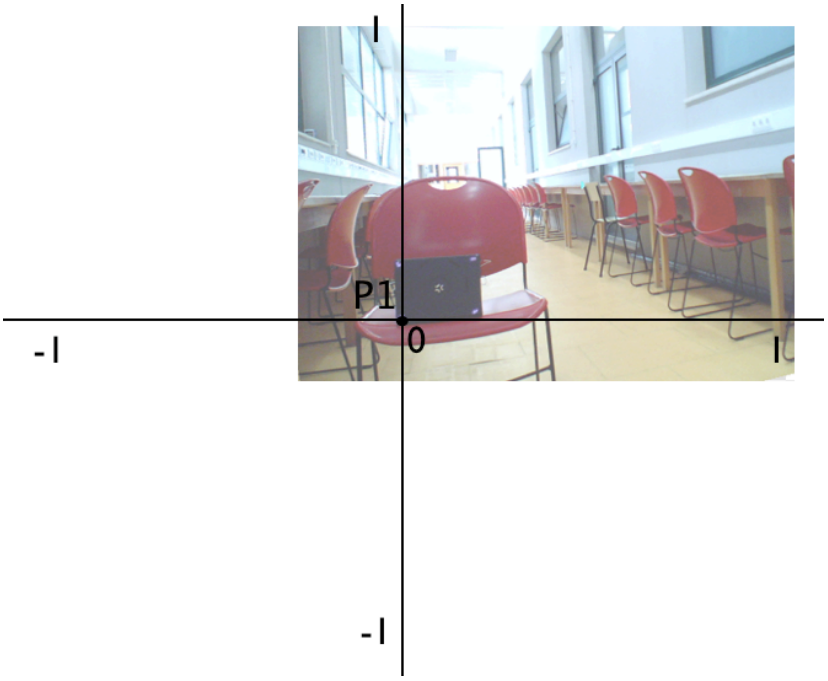


Figure 35:Image presentation I the viewer window after the translation

4.4.3 Image translation results

4.4.3.1 Wii Remote coordinates to camera coordinates conversion

The next example shows how the 4 points used in the entire algorithm are mapped between the camera and the Wii Remote. In the upper left image there are the points straight from the Wii Remote, the upper right is the image from the Web-Camera, in the lower right one are represented the points already converted, and in the lower left those points are overlapped with the image from the web-camera.

Wii Remote points	**Web_cam image**
Converted Wii Remote points	**Converted Wii Remote points + Web_cam image**

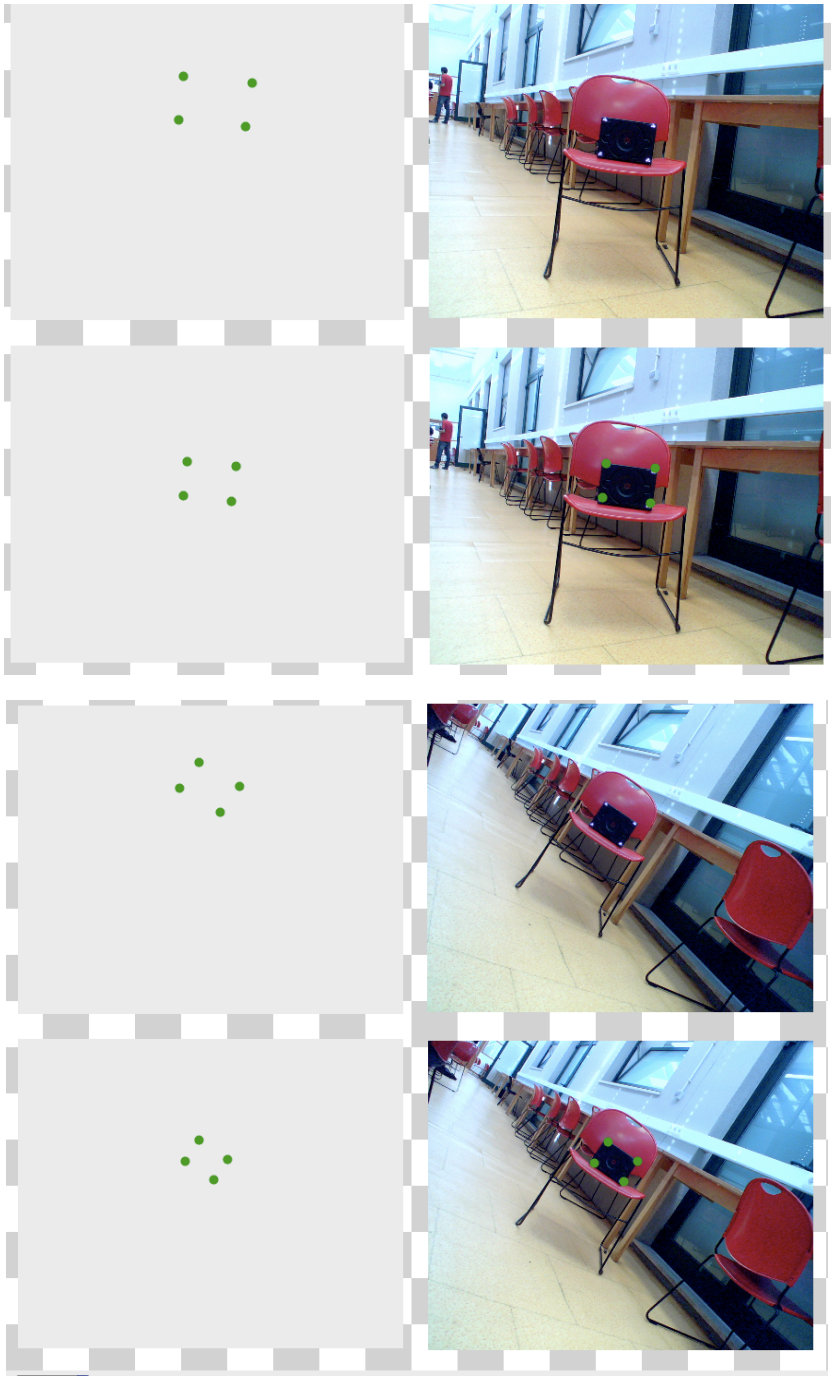


Figure 36: Wii Remote to web-camera coordinates system, demonstration.

In the previous example it's easy to notice the difference in the position of the marker points in the 2 devices. It also shows that the mapping function do the conversion rather well otherwise even if the previous steps were perfectly performed the stitching would be bad.

4.4.3.2 Image translation

The next example illustrate how the translation should work, the marker (actually the reference point which is the lower left dot) is positioned at the middle of the viewer window, independently from where the image was captured, the red dot represents de middle of the screen:

Figure 37: Image translation demonstration.

We can notice that the marker is not positioned right on the center in all the images this happens because the marker coordinates suffer a series of transformations before the translation, because of that the accuracy suffer a bit, the main deviation occurs on the Wii Remote to camera coordinates transformation function which like I have explained before was build by test, but that error is not enough to ruin the stitching.

4.4.4 Image translation evaluation

4.4.4.1 Wii Remote to camera coordinates conversion evaluation

To measure how well the algorithm was performing (more on the algorithm on section 4.4.1), I had developed a test which consist in taking a picture with the Wii Remote points overlapped on the image from the Web-Camera, then the distance between the

calculated points and the real ones was measured (in pixels). The camera's position varies during the test. To each capture I have collected each one of the 4 points as a different result.

The next image shows how the measure was done, the black point represent the real position of the marker, and the red one represent the result of the conversion from WiiRemote to Web-Camera coordinates.

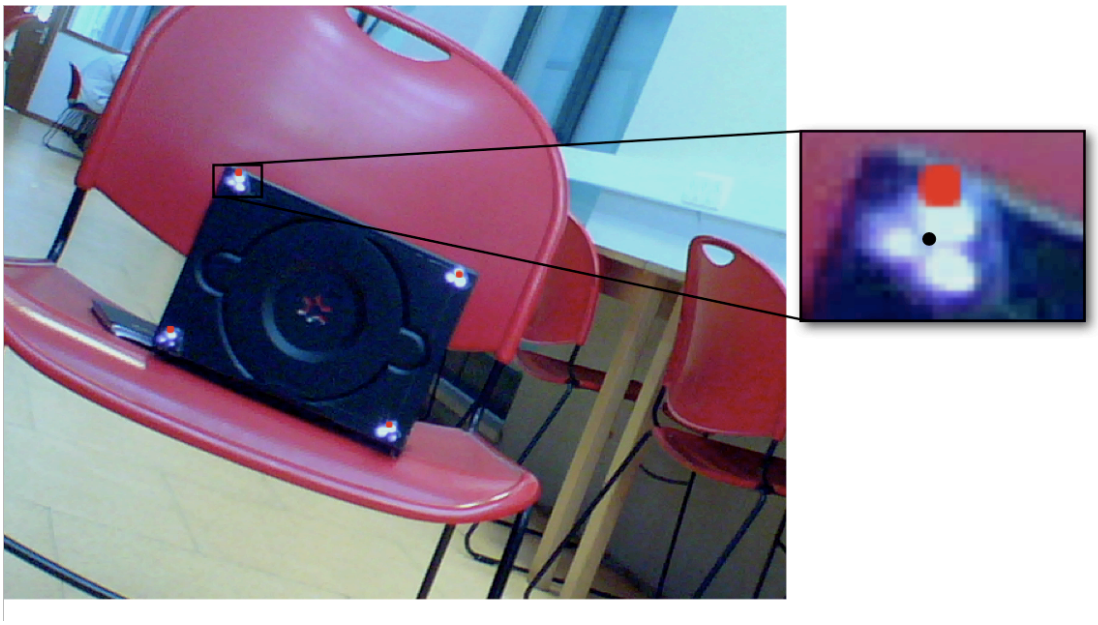


Figure 38: Coordinates conversion evaluation demonstration

The results of the evaluation are in the next table:

Coordinates				**Camera position**				
Wii Remote (x,y)		Camera (x,y)		**Difference**	Distance	X Rot	Y Rot	Z Rot
446	421	445	426	5,099	1,049	3,66	12,54	7,89
449	554	448	562	8,062	1,049	3,66	12,54	7,89
636	415	640	421	7,211	1,049	3,66	12,54	7,89
641	550	641	557	7	1,049	3,66	12,54	7,89
841	314	841	314	0	1,054	8,583	-11,3	8.883
846	448	844	448	2	1,054	8,583	-11,3	8.883
1047	441	1044	446	5,831	1,054	8,583	-11,3	8.883
1045	229	1042	303	74,061	1,054	8,583	-11,3	8.883
260	379	251	387	12,042	1.051	4,933	-7,646	2,157
255	519	249	529	11,662	1.051	4,933	-7,646	2,157

455	524	448	532	10,63	1.051	4,933	-7,646	2,157
461	387	451	393	11,662	1.051	4,933	-7,646	2,157
334	248	333	256	8,062	1,059	4,242	-5,422	-19,457
277	377	276	386	9,055	1,059	4,242	-5,422	-19,457
458	456	458	463	7	1,059	4,242	-5,422	-19,457
517	333	517	335	2	1,059	4,242	-5,422	-19,457
751	334	749	339	5,385	1.057	8,871	-8,871	-10,945
722	464	719	468	5	1.057	8,871	-8,871	-10,945
908	508	908	512	4	1.057	8,871	-8,871	-10,945
944	372	941	375	4,243	1.057	8,871	-8,871	-10,945
230	307	233	309	3,606	1,32	13,78	-4,82	-9,845
211	419	217	422	6,708	1,32	13,78	-4,82	-9,845
373	447	375	448	2,236	1,32	13,78	-4,82	-9,845
391	337	391	337	0	1,32	13,78	-4,82	-9,845
486	399	486	399	0	1,37	16,60	6,77	5,82
489	502	490	495	7,071	1,37	16,60	6,77	5,82
633	504	637	496	8,944	1,37	16,60	6,77	5,82
634	400	636	386	14,142	1,37	16,60	6,77	5,82
574	394	574	388	6	1,26	4,977	-2,969	3,593
577	504	572	500	6,403	1,26	4,977	-2,969	3,593
729	504	732	502	3,606	1,26	4,977	-2,969	3,593
734	394	734	389	5	1,26	4,977	-2,969	3,593
464	537	466	562	25,08	0,831	-17,018	0	-8,193
436	711	439	738	27,166	0,831	-17,018	0	-8,193
682	744	686	770	26,306	0,831	-17,018	0	-8,193
706	568	710	592	24,331	0,831	-17,018	0	-8,193
384	493	383	518	25,02	0,821	-21,47	-10,16	-23,449
291	648	293	673	25,08	0,821	-21,47	-10,16	-23,449
600	610	599	634	24,021	0,821	-21,47	-10,16	-23,449
513	767	513	794	27	0,821	-21,47	-10,16	-23,449

Table 7: Coordinate conversion evaluation results

Average error **11 pixels**, in a 1280*800 pixels image

From the previous table we can see that the biggest cause of errors is not the variation of the angles of view but the distance from where the scene is being captured, when it's to close to the marker the error increase, there are two reasons why that error did not affect the quality of the stitching:

1. Even though the error increases when the capture is closer to the maker, it never increases to a value that could compromise the stitching 27 pixels is the biggest deviation of a real point from the Web-Camera and the converted one from the WiiRemote.
2. Because the biggest errors occurs always from a certain distance on, I can easily avoid those distances, which in fact are to close if one wants to cover a wider view in the capture.

Some of the errors could also be explained by the configuration of the marker, like I have shown in section 4.1.3 in each corner of the maker there are 3 LEDs, during the capture the WiiRemote could detect one LED as stronger than the other 2, that point will be returned as the center of that corner even tough the center is in the middle of the LEDs, resulting in a small error in the detection.

4.4.4.2 Image translation evaluation

To evaluate the image translation algorithm (section 4.4.2), which should keep the marker always in the middle, I have took a picture with a dot marking the center of the viewer window and then I have measured (in pixels) the distance from the center to the lower left dot of the marker, I have repeated that process for several camera positions.

The next image shows how I did the evaluation:

Figure 39: Image translation evaluation example

The next table shows the results of this evaluation:

Reference point position		Real Center			Camera Position			
X	Y	X	Y	Difference	Distance	X Rot	Y Rot	Z Rot
646	400	640	400	6	1,28	5,911	1,15	3,38
645	400	640	400	5	1,23	4,243	3,08	4,44
640	400	640	400	0	1,22	0	0	4,97
645	398	640	400	5,385	1,21	8,495	0	9,13
644	396	640	400	5,657	1,21	18,55	0	9,40
640	393	640	400	7	1,347	16,83	3,66	-3,49
646	397	640	400	6,708	1,366	-1,65	-2,63	2,36
649	397	640	400	9,487	1,368	-5,72	4,073	9,26
640	400	640	400	0	1,2	-17,88	0	-3,53
644	398	640	400	4,472	1,22	-15,8	-1,65	-4,45
644	403	640	400	5	1,22	19,40	1,73	9,65
642	400	640	400	2	1,21	3,581	-4,91	-0,5
636	398	640	400	4,472	1,19	3,061	0	-0,5
630	383	640	400	19,723	1,44	16,372	1,52	14,66
645	399	640	400	5,099	1,19	16,82	-2,501	-0,33

Table 8: Image translation evaluation results

Average error **6 pixels**, in a 1280*800 pixels image.

As the previous results show the image translation is working well, I got 19 pixels as my highest deviations, which if we consider the size of the image the deviation values are small enough not to ruin the quality of the stitching. Some of the errors in the image translation also came from the previous step of the algorithm.

4.4.5 Image translation summary

The last step of the stitching algorithm is the image translation, which is simply just bringing each image next to each other.

In the translation process it is used a reference point (the lower left point of the marker), this point is detected by the Wii Remote, but the translation is applied to images from the web-camera so I had to discover where a point detected by the Wii Remote is positioned in a picture from the web-camera. In the previous step of the algorithm were applied several transformations to the images/videos, those transformations affected the position of the reference point, so his new position had to be calculated.

The translation is made by shifting both images so that the reference point becomes positioned in the middle of the viewer window, don't matter what's camera position or orientation the reference point (in both images) is always at the (0,0,0) position, because that part of the image is fixed in the center of the screen the process of doing a collage (from the view point of the user) is easier.

After the some tests to evaluate the translation process, it was noticeable that the image translation (and of course all the coordinates conversions needed in this step of the algorithm), is working well.

4.5 Stitching algorithm summary

In summary the 4 steps of the stitching algorithm can be explained as follows:

1. **Marker detection:** Detection of the 4 point of the marker using the WiiRemote.
2. **Camera pose Determination:** Using the previous detected coordinates and the pitch a roll values from the WiiRemote find the orientation of the web-camera combining the pitch and roll value from the WiiRemote with a "traditional" pose detection algorithm
3. **Image transformation:** Based on the orientation of the camera transform the received image from the web-cam with 3d rotation and scaling.
4. **Image translation:** Translate the image so that the reference point, in both images is situated in the middle of the viewer window.

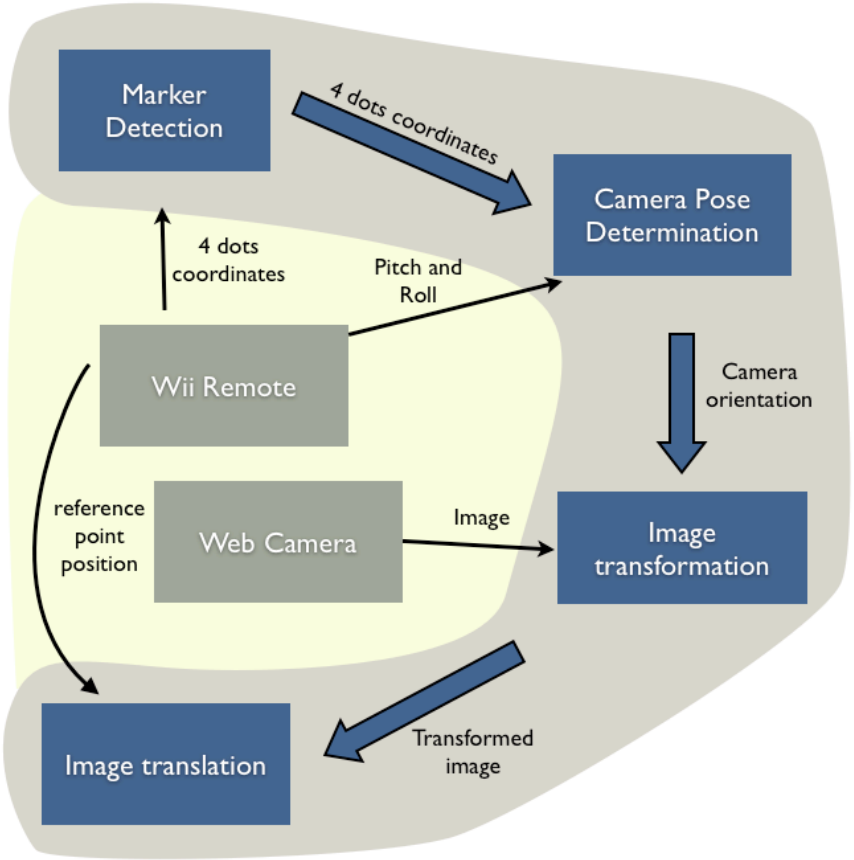


Figure 40: Stitching algorithm summary

5 Video Stitching

In this chapter, first, I will present how the proposed algorithm was optimized to perform real time video stitching (section 5.1). Second, I lay out how I present the stitching to the user as well as other possible presentation (section 5.2). Last, I lay out some possible applications for real time video stitching (section 5.3).

5.1 Video Stitching

Video stitching is probably the most important feature of my thesis. The algorithm is the same as the one for image stitching, except the sources are motion videos rather than static images. The image stitching algorithm was developed on the basis that it would be applied to real-time video stitching. As a result, only minor changes were needed to joining videos.

The algorithm for video stitching almost the same as the one for image stitching. However, there are two main differences that I will explain later in this chapter.

5.1.1 Data Input

In video stitching, the algorithm always receives new data. In image stitching, after one image was processed, the position, rotation, and size of the image were saved. These values did not have to be re-calculated again. However, with videos the stitching as to be made in real time, these values have to be constantly updated. As a result, additional computation is required.

5.1.2 Video Capture

The other major difference between image and video stitching is that, with video stitching the image capture has to be continuous. Because the images are shown in real time, their position and rotation have to be calculated in real time too.

In early tests, the results of video stitching were not good enough because of the sensitivity of the Wii Remote data. Data such as pitch, roll and IR marker coordinates are fluctuant. Even when the camera is relatively fixed, the video position and rotation result on the screen were unstable and always trembling. In order to overcome this problem, a build-in QC function, the Smooth patch, was introduced. The patch smoothes a numerical value in real-time and was appended to every patches that process data from the Wii Remote, in particular when the position and rotation values of the image was assigned.

5.1.3 Image Positioning

The decision of the using the proposed translation algorithm (section 4.4.2) was made when I started stitching videos. The images are transformed in that way that part of the source image always remains in the center of the screen. By doing so, it really helped at the early testing state when the algorithm was not finalized yet.

5.2 Presentation to the user

After the stitching process, it is time to present the images to the user. This presentation should be made according to type the scene and environment. A wider and open world scene could be better presented in a cylindrical shape like in a panoramic theater. A smaller indoor scene could be presented in a plane or solid volume. The stitched images/videos could also be projected as a texture onto a 3d model.

5.2.1 Plane

Presenting the images on a plane is the easiest way for coding, it requires least amount of computational power to process. This type of presentation could not be the most realistic or immersive to the user but it is good for tracking errors and testing the stitching algorithm.

5.2.2 Cylindrical Projection

Images presented in a cylindrical projection maintain a better relative size and position of the objects than a simple projection on plane. This type of presentation is used to represent open space.

5.2.3 3D Model Mapping

To get a more realistic presentation to the user, a 3d model of a given scene has to be built. And, the stitched images are mapped onto the 3d model.

This type of presentation is good for situations when one wants to show the same scene several times. Or if the cameras will be somehow restricted to an area so that the, used 3d model is always the same with no need for a new one in real time. The process of building a 3d model for the scene is very tedious and time consuming. One has to build a 3d model in every capture (for each different scene). Besides, this will require lots of

computing power to achieve in real-time. It somehow conflicts with the goal of my thesis work to provide an inexpensive stitching solution, therefore it is decided not to realize it.

5.3 Applications

With an additional tool for exporting the stitched videos to a video file or to another application, users could use the outputs for many applications:

A better sensation of exploring the world could be one potential application of the image translation algorithm that keeps a portion of the scene fixed (section 4.4.2). If the user covers a large area of the environment when capturing the scene, the resulting stitched video could transmit to the viewer the feeling of exploring the scene by him/herself. That feeling could be useful for showing different locations to a tourist, or presenting a historical place in a museum as well.

A wide representation of a scene could be made, using the proposed algorithm with a modification, a type of accumulator that saved every frame and position of the video in the viewer window, resulting in a dragging effect almost like if the user is painting the viewer window with the images that are being captured in real-time.

Mapping the stitched video to a 3d representation of a given scene could also provide a more immersive visualization experience for users.

6 Results & Evaluation

6.1 Results

During the development process I did some tests to verify how the result was evolving, to plan what I had to do next, and of course to verify if everything was going as planed.

Next I will present the results I got at different stages of the development process. In the results from section 6.1.1 to 6.1.2 the translation algorithm used is still the first one (more on section 4.4.2). From section 6.1.3 on, the collage was made in real time.

In the next results, is noticeable the evolution not only of the algorithms but also of the marker, since I will present the results chronologically (from when the tests were made), the results also show the evolution of the maker.

The complexity of the tests also evolves, as the algorithm and maker evolved.

6.1.1 Still Images with 1 WiiRemote and 1 Web-Camera

At this stage the algorithm was detecting the camera pose to transform the images in 3d according to the camera orientation, the images were not transformed in size, they were only rotated in 3d.

Figure 41: Still Images with 1 WiiRemote and 1 Web-Camera

We can notice here the main function of the pose detection algorithm the image at left was taken from a different angle than the one at right but, apart from some adjustments in image transformations (angles) and translation needed this joining is good.

This was a very preliminary test, some of the coordinates conversions are not present in this joining

6.1.2 Still Images with 2 WiiRemote and 2 Web-Cameras

At this stage I was using 2 cameras and 2 Wii Remotes, but only used one pair at the time. All of the algorithms are the same as in the previous result except from the calculations of the shifting in the Z-axis and Y-axis that were improved. All the shifting calculation was made using Wii Remote IR camera coordinates instead of trying to match the Web-Camera coordinates (more on section 4.4.1.2).

Figure 42: Still Images with 2 WiiRemote and 2 Web-Cameras

Figure 43: Still Images with 2 WiiRemote and 2 Web-Cameras

In this step I had already calculated the resulting coordinates after the rotation, and the translation algorithm was using those coordinates, I think it's visible an increase in the quality of the joining.

6.1.3 Still Images with 2 WiiRemote and 2 Web-Cameras (WiiMote to Web-camera coordinates match)

The algorithm is the same as in the previous step except that in this one there is a match between the Wii Remote and Web-Cameras coordinates, and also the result of applying a rotation to the markers points was improved.

Figure 44: Still Images with 2 WiiRemote and 2 Web-Cameras (WiiMote to Web-camera coordinates match).

6.1.4 Still Images with 2 Wii Remote and 2 (WiiMote to Web-Camera coordinates match)

In the next results the algorithm was at his final stage of development, every step of the algorithm was improved from the previous test to this one. The red dot on the center of the screen is still present in the first example (it is used as a reference for the translation), but it was removed in the final tests.

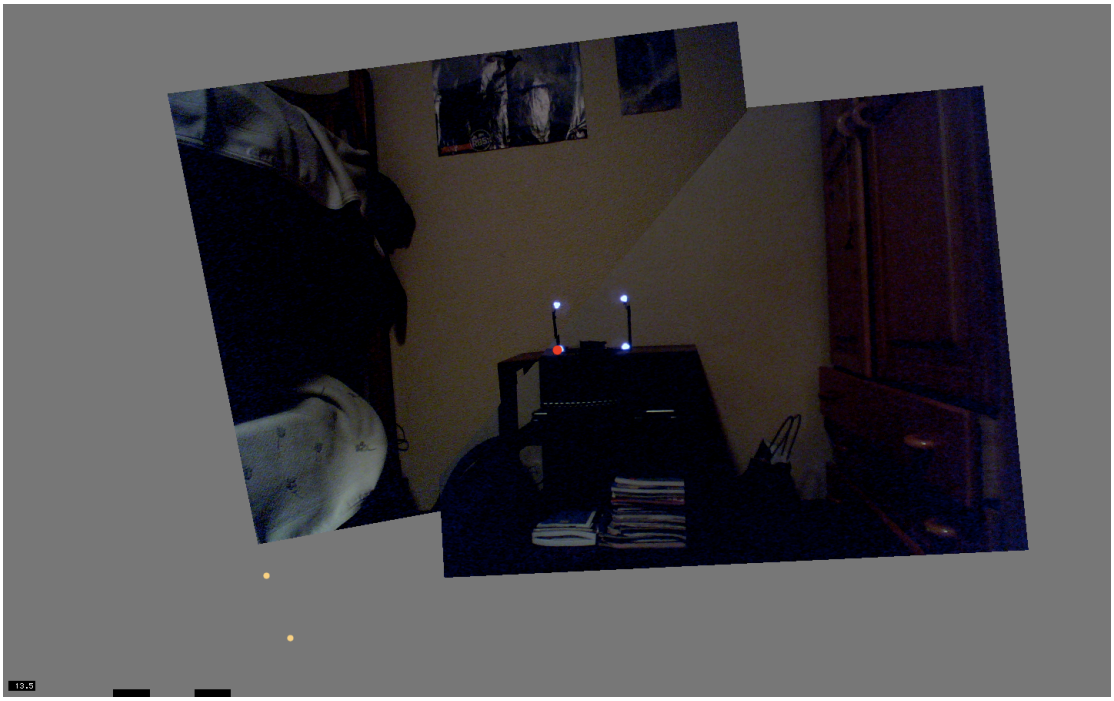


Figure 45: Final algorithm result 1

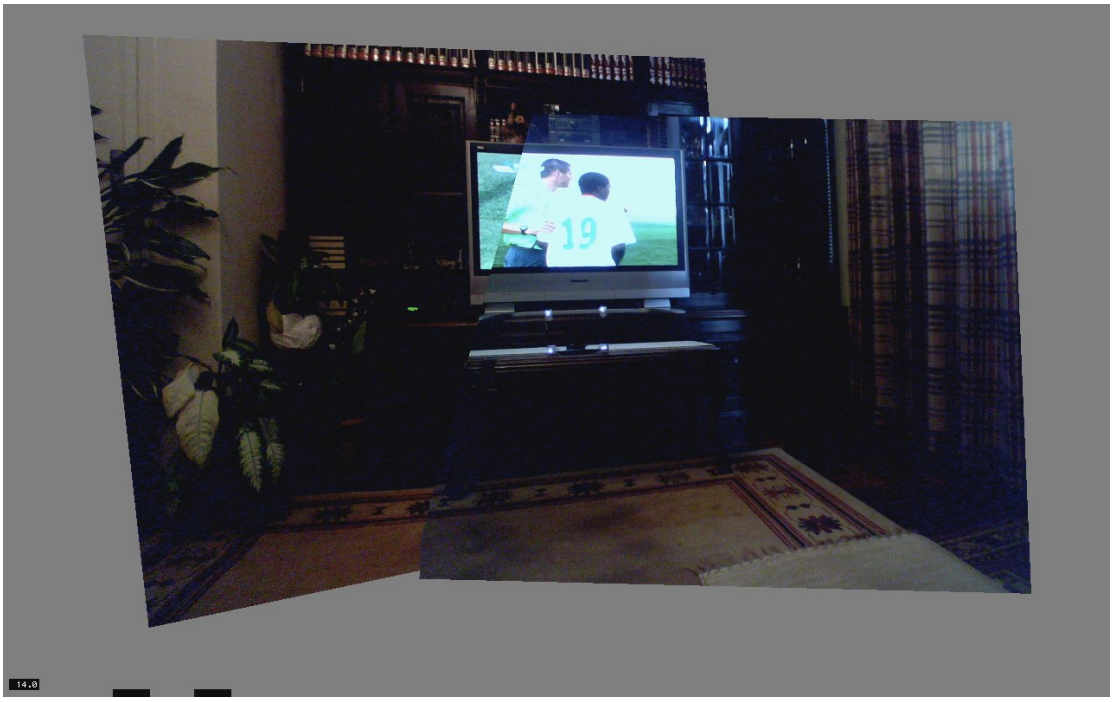


Figure 46: Final algorithm result 2

Figure 47: Final algorithm result 3.

Figure 48: Final algorithm result 4.

Figure 49: Final algorithm result 5.

7 Evaluation

It's difficult to measure how well a joining it made, it's simple for one to look at the result and evaluate if it's "good" or "not good", but it's hard to assign a value to that evaluation. I have developed a method that removes some of that subjectivity from the algorithm evaluation.

7.1.1 Overall stitching evaluation

I preformed a stitching with a grid as background that allowed me to measure the misalignment between the images, like in the other tests all the measurements are made in pixels.

I have tried to cover a wide variety of angles and positions with this test, the next image show the used setup for the test.

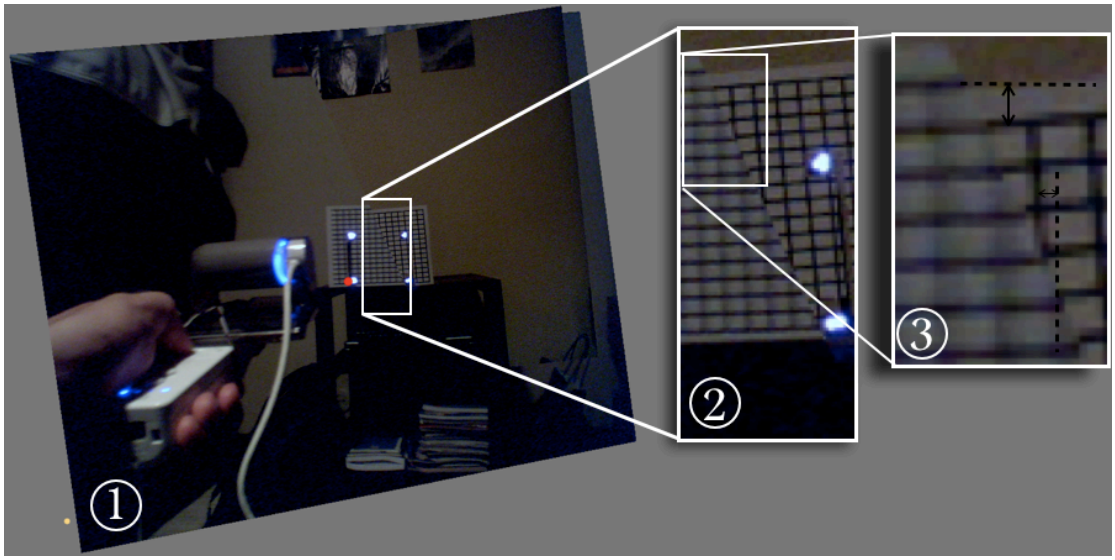


Figure 50: Example of the evaluation process

1. Stitched image used for evaluation.
2. Border between the 2 joined images.
3. Representation of the measurements taken in each image.

In each image I have measured as much correspondences as possible, because the border between the 2 images is a vertical line, I could perform more measurements on the Y-axis than on the X-axis.

Test 1

X1	X2	**Difference**	Y1	Y2	**Difference**
758	748	10	451	455	-4
756	756	0	445	442	3
762	764	-2	437	434	3
770	772	-2	430	427	3
777	780	-3	424	422	2
785	788	-3	417	415	2
			410	408	2
			403	401	2
			396	394	2
			389	388	1
			382	381	1
			376	374	2
			369	368	1
			362	362	0
			355	355	0
			350	350	0

Table 9: Overall stitching evaluation results (test1)

Average difference X 0; Y 1,25 **pixels**

Test 2

X1	X2	**Difference**	Y1	Y2	**Difference**
747	752	-5	348	348	0
756	761	-5	356	356	0
765	770	-5	363	363	0
774	779	-5	369	369	0
783	788	-5	376	376	0
793	797	-4	382	382	0
802	805	-3	389	390	-1
812	815	-3	396	396	0
823	823	0	403	404	-1
830	834	-4	410	411	-1

			417	418	-1
			423	425	-2
			430	432	-2
			437	438	-1
			443	445	-2
			440	438	2

Table 10: Overall stitching evaluation results (test2)

Average difference X 0,56; Y 3,9 **pixels**

Test 3

X1	X2	**Difference**	Y1	Y2	**Difference**
744	749	-5	346	349	-3
743	749	-6	356	356	0
743	748	-5	362	362	0
744	748	-4	370	370	0
			376	376	0
			395	395	0
			403	403	0
			409	409	0
			416	416	0
			423	423	0
			430	430	0
			437	437	0
			443	442	1
			450	448	2

Table 11: Overall stitching evaluation results (test3)

Average difference X 5; Y0 **pixels**

Test 4

X1	X2	**Difference**	Y1	Y2	**Difference**
764	768	-4	348	354	-6
776	775	1	354	361	-7
787	787	0	361	373	-12
798	798	0	375	380	-5
		-0,75	383	386	-3

			388	393	-5
			395	400	-5
			403	406	-3
			410	412	-2
			416	419	-3
			423	426	-3
			430	433	-3
			436	439	-3
			443	445	-2
			450	452	-2

Table 12: Overall stitching evaluation results (test4).

From the result presented before, I can observe that the error in the stitching algorithm is small (it is never bigger than 15 pixels). The main cause of that error is the differences in the alignment of the cameras with the Wii Remotes. Because the top of the camera is not fixed it is hard get the same position every time, although it is easy to align both cameras manually.

One of the reasons for the success of the evaluation was, the actual evaluation setup, the grid is positioned at the same plan as the marker, and the algorithm is optimized for the joining at that level, misalignments in the foreground or background are bigger and also harder to evaluate.

8 Conclusions

8.1 Summary

The video/image stitching implementation was made possible with the use of the combination of four markers and the Wii Remote. The final solution does not need any difficult-to-acquire hardware or software for testing. It does not require special knowledge to operate. The algorithm is capable of performing real time video stitching with an average frame rate between 8~16 fps. The final results of the evaluation show the quality of the stitching except some misalignments in special cases.

During the development of this thesis, I have tried many different approaches in order to accomplish my goal stitching video in real-time. My ideas got clearer when I started to use the Wii remote and realized the possibilities that it could provide. In fact, the Wii Remote and the markers have been helpful in all stages of the algorithm.

The final solution uses two web-cameras and two Wii Remotes. Since they are commercially available hardware, the implementation is inexpensive and affordable for general public to test it. The capture process also meets another important requirement, which is flexibility to use. Anyone without specific photography or computer vision knowledge can make a stitching with very short and simple explanation.

The proposed solution performs video stitching in real time with an average frame rate between 8-16 fps. Those results are achieved because I have made some decisions in the development to simplify the process. Those decisions include: the 3d transformation applied to each image is done through a built in Quartz Composer patch; the usage of a marker removed the need for expensive feature tracking in the images, and of course the use of Wii Remotes which were really helpful to detect the maker and in the pose determination algorithm.

Evaluations were made through all steps in the implementation that helped me to verify how the decisions made during the development process affected the algorithm. Besides the individual evaluation, I have also measured the overall quality of the stitching (alignment between the two videos/images). The result of that evaluation showed that the quality of the stitching is good with objects at same level as the marker, misalignments in the foreground or background are bigger.

Some simplifications were made in the algorithm not only to increase the performance of the process but also to add more flexibility to the capture process. At the end, the final algorithm balances trade-off and reaches a good compromise between performance, quality and flexibility.

8.2 Issues

As mentioned earlier, the final solution is, in my opinion, a good commitment between quality, flexibility and performance. Nonetheless, there are some issues with the final solution that could compromise the quality, flexibility and performance of the algorithm. Sometimes the marker detection was not as good as it was expected. That inaccuracy was reflected on the output of the very sensitive pose determination algorithm. The overall quality of the stitching was also affected by the quality of IR marker detection done by the camera on a Wii Remote.

Even though the marker detection obtained a big improvement with the usage of the Wii Remote along with the IR light markers. The detection still fails mainly in outdoors environments. One reason is that the sunlight also consists of wide range of IR light in his radiation that lead to failure of the detection.

The implemented pose determination algorithm (the analytic solution) was very sensitive. The issue was enlarged because some of inputs provided to the algorithm can only be estimated, such as the focal length and sensor size of the Wii Remote IR camera. This issue was reduced with the combination of the pose determination algorithm with the rotation values from the Wii Remote.

The more common issue during the execution of the proposed solution is the frequent connection error between the Wii Remote and the computer. It is sometimes hard to establish a connection over Bluetooth between the Wii Remote and the computer. However, once the connection is made it is sable and the connection almost never gets lost.

All the issues listed above do not occur that often to ruin the stitching experience, still they could be further improved in the future.

8.3 Future work

In order to solve some of the issues that I have mentioned above, or to improve the quality of the result, there are solutions that could be applied in future work.

Various color correction algorithms could be applied to each image to level the colors, brightness and contrast of source images, so that the blended part in the collage could be smoother.

To remove some of misalignments and distortion in the stitched images/videos due to camera lens distortion, optical correction algorithm could be applied to the source video/image.

In current implementation, the stitching line that divides the joining between the two images is a straight line. Techniques, such as feature tracking or color matching

between the source images, could improve the quality of the stitching result. However, those techniques impose a non-linear stitching line, which certainly increase the demands for real-time computing power.

A different (more stable) pose determination algorithm could be implemented, so that the capture process could be more stable. The result could also gain some improvements from a better pose determination.

In order to expand the potential applications of real-time video stitching, it is important to export the stitched result to a video file or source, so that output could be used for any purpose users request.

A simple addition that would result in visible improvements to the result is the usage of an IR filter in the web-cameras so that the IR light from the marker would be removed from the joining.

9 Bibliography

Apple inc. (2008). *Introduction to Quartz Composer Programming Guide*. From Mac OSX Reference library: http://developer.apple.com/mac/library/documentation/GraphicsImaging/Conceptual/QuartzComposer/qc_intro/qc_intro.html

Black, J., & Ellis, T. (2006). Multi camera image tracking. *Image and Vision Computing* (24), 1256–1267.

Bourke, P. (2006, February). *Mesh format for image warping*. ( ) Retrieved 03 22, 2009 from http://local.wasp.uwa.edu.au/~pbourke/dataformats/meshwarp/

Chandra, C. T. (1989). *Landmark Tracking and camera Calibration.* University of Tenesse, Knoxville.

ChaosPro. (2008). *HSL Colorspace*. Retrieved 10 25, 2008 from http://www.chaospro.de/documentation/html/paletteeditor/colorspace_hsl.htm

Foote, J., & Kimber, D. ".-1. (2000). FlyCam: practical panoramic video and automatic camera control. *Multimedia and Expo. 3*, pp. 1419-1422. IEEE International Conference.

Gattas, M. (2008, 10 26). *RGB to HSL*. From http://www.tecgraf.puc-rio.br/~mgattass/color/RGBtoHSL.htm

Green, B. (2002). *Edge detection tutorial*. Retrieved 10 21, 2008 from Autonomous systems lab: http://www.pages.drexel.edu/~weg22/edge.html

Green, B. (2002). *Edge Detection Tutorial*. Retrieved 10 21, 2008 from Autonomous systems Lab: http://www.pages.drexel.edu/~weg22/can_tut.html

Guide, I. t. (2008). Retrieved 11 20, 2008 from Mac OS X Reference Library: http://developer.apple.com/mac/library/documentation/GraphicsImaging/Conceptual/QuartzComposer_Patch_PlugIn_ProgGuide/Introduction/Introduction.html

H.J.C, L. (2008). *Basics of color based computer vision implemented in Matlab.* Eindhoven.

Hiroaki. (2008). Retrieved March 13, 2009 from sourceforge: http://darwiin-remote.sourceforge.net/

Kamata, S., Eason, R., Tsuji, M., & Kawaguchi, E. (1992). A camera calibration using four point-targets. *: Pattern Recognition, 1992. Vol.I. Conference A: Computer Vision and Applications, Proceedings., 11th IAPR International Conference* , (pp. 550-553). The Hague,.

Kang, E.-Y., & Yoon, I. (2005). Smooth Scene Transition for Virtual Tour on the World Wide Web. *IEEE* .

Lihi, Z.-M., & Perona, P. (2007). Retrieved 10 27, 2008 from http://www.vision.caltech.edu/lihi/Demos/AutoJoiners.html

Lihi, Z.-M., & Perona, P. (2007). Automating Joiners. *NPAR* .

Martin, A. F., & Robert, C. B. (1981). Random Sample Consensus: A Paradigm for Model Fitting with applications to image analysis and Automated Cartography". *SRI International*, *24.*

Neel, J., Wojciech, M., & Shai, A. (2006). Natural Video Matting using Camera Arrays. *ACM Transactions on Graphics* (25).

Nomura, Y., Zhang, L., & Nayar, S. (2007). *CAVE|Projects: Scene Collages and Flexible Camera Arrays*. Retrieved 11 27, 2008 from Computer vision laboratory Columbia University: http://www1.cs.columbia.edu/CAVE/projects/scene_collage

Pemberton, S. (2002). *HSL*. Retrieved 10 24, 2008 from http://homepages.cwi.nl/~steven/css/hsl.html

Pintaric, T., Neumann, U., & Rizzo, A. *mmersive Panoramic Video.* University of Southern California, Integrated Media Systems Center, Los Angeles.

Quan, L., & Lan, Z. (1999). Linear N-Point Camera Pose Determination. *IEEE Trans* , pp. 774-780.

Rocchini, C., Cignoni, P., & Montani, C. *Multiple Textures Stitching and Blending on 3D Objects.* Istituto di Elaborazione dell'Informazione.

Timo, F. (2009, 2 11). *Wiimote Virtual Reality Desktop*. Retrieved 5 6, 2009 from Coding4Fun: http://blogs.msdn.com/coding4fun/archive/2009/02/11/9412433.aspx