Markerless Affine Region Tracking and Augmentation Using MSER and SIFT

A Senior Project
presented to
the Faculty of the Computer Engineering Department
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in Computer Engineering

by
Greg Eddington
June, 2011

Abstract

Due to the advancements in mobile computing hardware and the inclusion of cameras in many computing platforms, augmented reality systems have become widely available. This paper presents a real-time implementation of a novel markerless augmented reality algorithm which is able to track two-dimensional affine regions without a priori information of the environment or computing a world model. The implementation consists of the MAR library; a modular software library which performs the region detection, identification, and tracking; and the Lighthouse application; a program which uses the MAR library to allow the user to augment scenes viewed from a camera. The algorithm used in this paper is able to track textured affinely invariant planar regions using the maximally stable extremal region (MSER) detector to detect the regions and scale-invariant feature transform (SIFT) detector to identify the regions. If three or more points from the region are matched across frames, an affine transformation is calculated which can be used to augment the scene by transforming a two-dimensional virtual image, causing it to appear to be stationary with respect to the region being tracked. Testing found that the system is robust at tracking translation, scale, and rotational changes in regions at a frame-rate adequate for real-time applications and is able to handle changes in angle and illumination.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Augmented reality (AR) is a field of computer vision which incorporates virtual images into real-world images. An augmented reality system is designed to blend computer generated images into camera footage in real time. AR has become increasingly popular in the field of mobile electronics, as the increase in power and the addition of high-resolution cameras have allowed for mobile augmented reality to become feasible in consumer electronics. Wikitude is an example of a mobile augmented reality application, able to combine computer generated landmark information over a physical image of landmarks (1).

Figure 1.1: An image from Wikitude, a mobile augmented reality application, augmenting a scene of a landmark with information about the landmark (1).

An augmented reality system is generally compromised of three core components: one or more cameras, a display, and a computer. Most cameras can be used for augmented reality systems, although higher quality images are preferred as they allow for

easier object recognition and tracking. Three forms of displays are often used in AR systems. Head mounted displays are worn over the users eyes and augment what the user sees. A magic mirror setup involves a screen which is placed across from a user with a camera pointing at the user, creating an setup similar to a mirror which displays an augmented scene. A magic lens setup uses a handheld screen with a camera mounted on the back, allowing the user to view "through" the device to see an augmented scene. These three display styles are shown in figure 1.2.

(a) Head Mounted Display (4)

(b) Magic Mirror (6)

(c) Magic Lens (7)

Figure 1.2: Three common display setups used in augmented reality systems.

Chapter 2

Background

Several algorithms exist for augmenting reality. Two key features which distinguish various augmented reality algorithms are whether or not they require a priori information and whether or not they require a world view. A common characteristic of augmented reality systems which require a priori information is the use of markers. The marker's appearance and dimensions are known by the AR system beforehand. The algorithms are designed to look for that specific marker, and use the a priori information given about it to augment the scene with a virtual image. Table 2.1 describes the advantages and disadvantages of using markers in an AR system. When using markers in an AR system, the system knows which surfaces to augment via a priori information and regions are identified by matching them to known patterns. When markers are not used in an AR system, the system learns which surfaces to augment via user input and regions are detected and tracked by looking for distinguishing keypoints.

ARToolKit is an example of an AR system which makes use markers to allow virtual objects to be augmented into a scene (shown in figure 2.1) (8), while Golfscape is an example of an AR system which uses spatial awareness in order to augment the distance to various golf course features into a view of the course (shown in figure 2.2) (3). The algorithm described and implemented in this paper is built on work by Matt Marano in his thesis Affine Region Tracking and Augmentation Using MSER and Adaptive SIFT Model Generation (4). This approach does not use markers or spatial awareness to create an AR system, creating an algorithm flexible for many environments, requiring only a commodity camera with no additional hardware and no alterations to the scene which will be augmented.

Characteristic	Advantages	Disadvantages
Markers	Three dimensional information can be extracted knowing the dimensions of the marker. Markers can cue the system to augment the space differently based on physical patterns.	Requires the user to add unnatural markers to the physical scene. Markers may require extra work from the user to create. Only regions where markers can be placed can be augmented.
Spatial Awareness	Three dimensional information can be extracted knowing the location of the camera relative to the region. Markers can cue the system to augment the space differently based on spatial location.	Systems require additional sensors, such as gyroscopes and GPS sensors. Prohibits augmentation using existing cameras without position sensors, such as webcams.

Table 2.1: Advantages and Disadvantages of AR Characteristics - Advantages and disadvantages of common characteristics of augmented reality systems.

2.1 Project Requirements

The goal of this project is to implement a markerless augmented reality algorithm which can track and augment two-dimensional affine regions. The following were the requirements the project aimed to fulfill:

Affine Region Detection The ability to detect affine surfaces which can be used as surfaces for augmentation.

Affine Region Tracking and Augmentation The ability track a selected affine surface and calculate an affine transformation matrix which can transform any point on the surface from an initial camera frame to subsequent camera frames.

No A Priori Information The ability to augment a scene with no information given a priori, such as marker patterns.

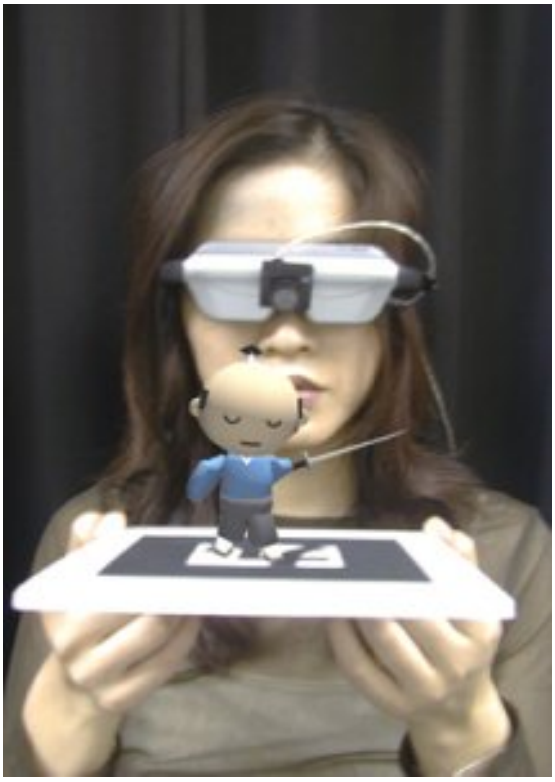


Figure 2.1: An image from ARToolKit, an augmented reality library, augmenting a scene using markers physically placed into the environment (2).

Figure 2.2: An image from Golfscape, a mobile augmented reality application which uses a GPS and solid state compass to augment golf course information onto images of the course streamed from the phone's camera (3).

No World Model The ability to augment a scene without using a world model to store the location of objects in the environment or the use of GPS or gyroscopic information.

Implementation An implementation of this algorithm in C/C++ to allow for use on

multiple platforms.

Reusable Code The creation of a software library which can be reused in future projects which wish to use this markerless augmentation algorithm.

Real-Time Augmentation The ability to augment reality in real time and not only in post-processing.

Modular Camera Support A modular design which can allow for the support of different types of camera hardware.

Configurable Environment A configurable design which allows for the augmentation to be fine-tuned to different environments and use cases.

Chapter 3

Design

3.1 Computer Vision Algorithms

The design of the markerless augmented reality algorithm presented in this paper relies on a two phase approach. The affine region detection phase involves using the maximally stable extremal regions (MSER) computer vision algorithm to detect affine regions in the camera frame. The region tracking and affine transform calculation phase involves using the scale-invariant feature transform (SIFT) computer vision algorithm to track a chosen affine region and calculate a transform to map model space to camera space for the region, which can later be used for augmentation.

As the MSER and SIFT algorithms are key to the augmentated reality system, a brief description of both is provided. A summary of the two computer vision algorithms is presented in table 3.1.

3.1.1 Maximally Stable Extremal Region (MSER)

The maximally stable extremal region (MSER) detector computer vision algorithm was created to establish correspondence between two images of a scene from different viewpoints (9). The detector finds regions of continuously connected pixel "blobs" which are both extremal and maximally stable. The detector can be used to find affine regions. These regions are robust to changes in viewpoint, rotation, scale, and lighting when compared to other affine region detectors (10). While the MSER are blobs, they are often represented with an ellipse as shown in figure 3.1.

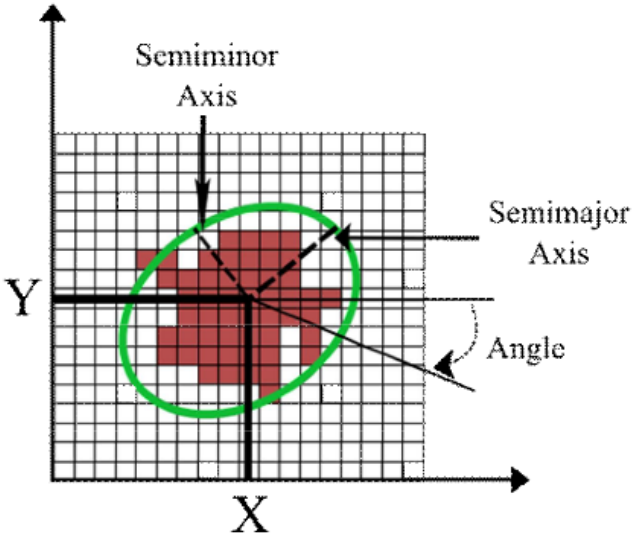


Figure 3.1: The features which describe maximally stable extremal regions (4).

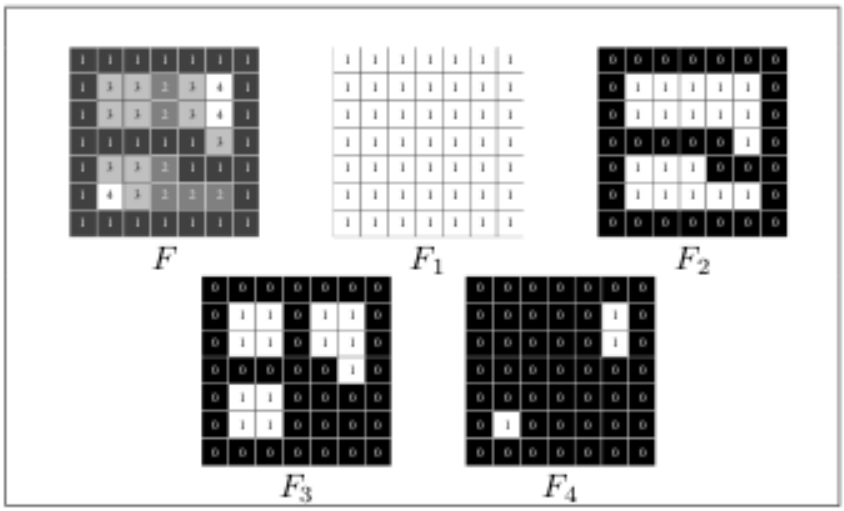


Figure 3.2: The progression of thresholding performed in the MSER algorithm. (5).

The MSERs of a frame are found by thresholding the gray-scale of the image at all possible thresholds. This creates a sequence of images which begin as all white and gradually changes to an image of all black pixels, as shown in figure 3.2. Maximal regions are the set of all connected regions of all frames. Maximally stable regions are ones where the relative area change as a function of relative change of threshold is at a local minimum (9).

3.1.2 Scale-Invariant Feature Transform

The scale-invariant feature transform (SIFT) detector is a computer vision algorithm which was created to find and describe local features in an image (11). It finds keypoints in a camera frame that are affinely invariant to changes in translation, rotation, and scale. These keypoints are represented as circular regions, shown in figure 3.3.

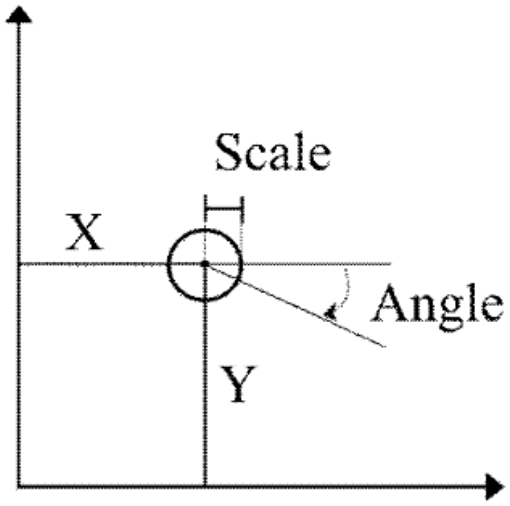


Figure 3.3: The features which describe SIFT keypoints (4).

The SIFT algorithm begins by identifying regions of interest. It does this by Gaussian blurring the image at multiple levels in make the method invariant to scale. It then performs a difference of Gaussians (DoG) by subtracting adjacent levels of the Gaussian blurred images in order to find stable regions, shown in figure 3.4. Regions of interest are found by locating the local maxima and minima in the image. If a pixel is less than or greater than all 26 neighbors (the 8 surrounding it on the current scale, the 9 on the previous scale, and the 9 on the following scale, as shown in figure 3.5) it is considered a region of interest.

The second phase of the SIFT algorithm filters out noisy points. Noisy points are those with low contrast and those which are poorly localized. Points with a low value in the difference of Gaussians function are considered to be low contrast and are filtered out, as shown in 3.6b. Poorly localized points are detected by finding keypoints which have a large principle curvature perpendicular to an edge and a small principle

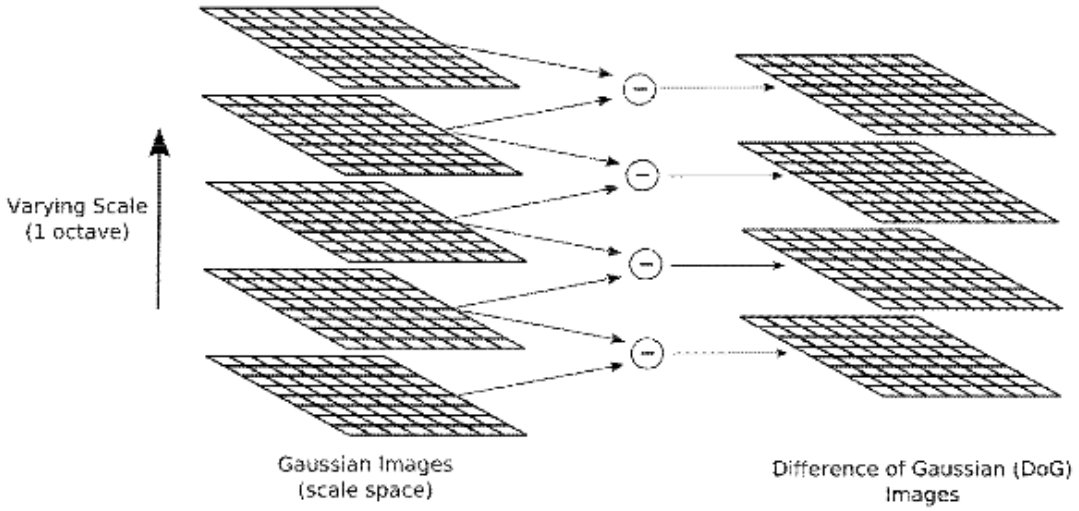


Figure 3.4: A difference of Gaussians is performed on several different levels of Gaussian blurring on an image in the SIFT filter (4).

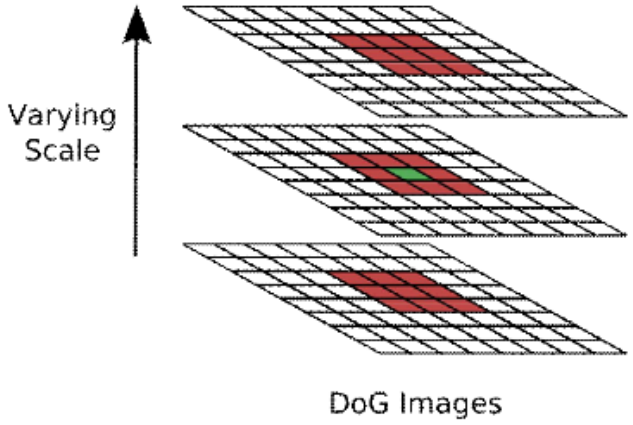


Figure 3.5: The extrema are found by comparing a pixel with its 26 neighbors, with 9 being from the previous level of DoG and 9 being from the next level (4).

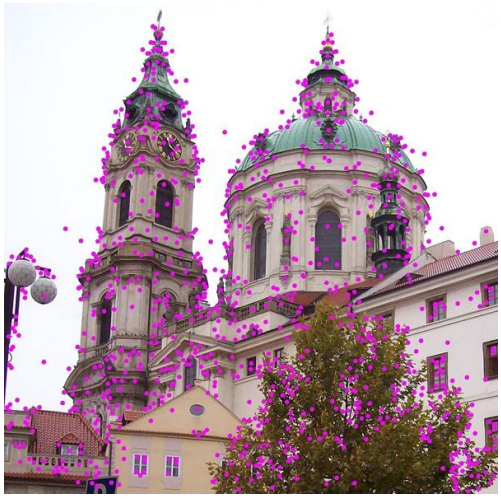
curvature in the direction of the edge. These points are filtered out as shown in figure 3.6c.

3.2 Design Overview

A diagram of the flow of the algorithm is presented in Figure 3.7. As mentioned before, the algorithm is divided into a affine region detection phase and a region tracking and

(a) Local extrema found from the DoG of an image are considered regions of interest.

(b) Regions of interest with low contrast are filtered out.



(c) Regions of interests with poor localization are filtered out, leaving the keypoints.

Figure 3.6: Regions of interest being filtered in the SIFT algorithm to the final keypoints.

transform calculation phase.

Name	Maximally Stable Extremal Region (MSER)	Scale-Invariant Feature Transform (SIFT)
Regions	Continuously connected pixel regions	Circular keypoint
Region Representation	Ellipse	Circle
Detector Criteria	Maximum or minimum image intensity relative to overall frame	Maximum or minimum image intensity relative to local area
Region Description	Center_x, Center_y, Variance_x, Variance_y, Mean	Center_x, Center_y, Scale, Angle, 128 Element Descriptor

Table 3.1: Vision Detector Summary - A summary of the MSER and SIFT detectors (4).

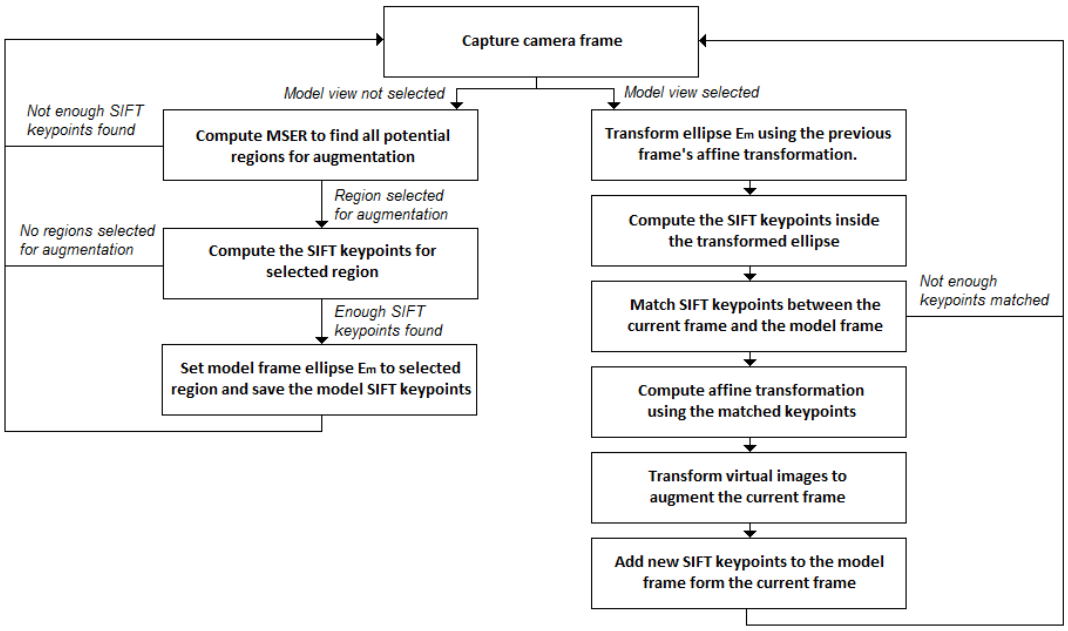


Figure 3.7: A flow diagram of the algorithm presented in this paper.

3.3 Affine Region Detection

The algorithm will identify potential candidates for augmentation by calculating the MSER in a camera frame. An example of the MSER detected in a camera frame is shown in figure 3.8. The user can select one of these regions for augmentation. The MSER is then used to determine the region to augment.

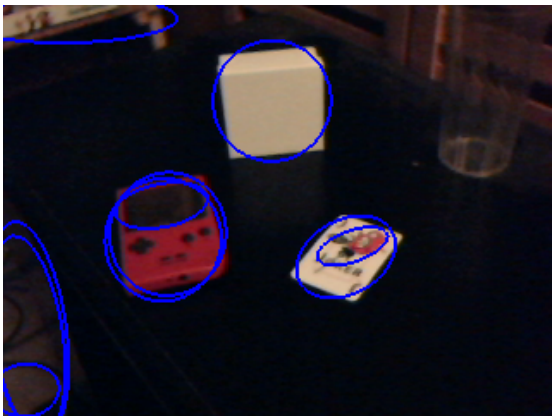


Figure 3.8: An example of several ellipse representing MSER found in a camera frame. These are potential candidates for augmentation.

The SIFT algorithm is then used to identify all keypoints within the frame. The SIFT keypoints within the selected MSER are recorded as the model keypoints and the selected MSER ellipse is recorded as the model ellipse. A visual representation of SIFT keypoints found in a camera frame is shown in figure 3.9.

3.4 Affine Region Tracking

Once an affine region has been selected and the model view frame has been recorded, the region is tracked in subsequent frames. The model view ellipse is transformed using the previous frame's affine transformation in order to transform the ellipse into the last frame's space. This transforms the search space into the region of the last frame. The SIFT keypoints within the transformed ellipse are then matched with SIFT keypoints in the model view ellipse. Assuming camera space does not change much between frames, limiting keypoints to the ellipse of the last frame can help reduce matching keypoints with different but similar appearing targets in the frame. This also reduces the search space the SIFT algorithm needs to cover per frame. As a backup, if the region cannot

Figure 3.9: An example of several SIFT keypoints found in a camera frame. These are used for region identification and tracking.

be matched within the search space of the ellipse transformed with the previous frame's affine transformation, the whole camera frame is then used as a search space for the SIFT algorithm.

In order to match SIFT keypoints between the model view and the current camera view, the distance between all keypoints in the model view and all keypoints in the current camera view is calculated using 3.1, where k_1 and k_2 are keypoints. The keypoint with the lowest distance is chosen as the best match, but in order to prevent matching between ambiguous or dissimilar keypoints, 3.2 and 3.3 are used to check if the match is valid. σ controls how unambiguous the matching must be, and δ_{max} controls the maximum distance threshold for a pair of keypoints to be considered matching.

$$distance = \sum_{i=0}^{128} |k_1 descriptor_i - k_2 descriptor_i| \tag{3.1}$$

$$distance_{best} * \sigma < distance_{secondbest} \tag{3.2}$$

$$distance_{best} < \delta_{max} \tag{3.3}$$

A minimum number of matching keypoints, greater than 3, can be defined as a threshold to consider a region matched between the model frame and current camera

frame. If at least this many keypoints are matched, then the algorithm proceeds to calculate the affine transform, otherwise it proceeds to the next frame without augmenting the scene.

3.5 Affine Transformation Calculation

After finding matching keypoints between the model view frame and the current frame, the affine transformation is calculated which can be used to augment the scene. An affine transformation can be calculated from the model view frame to the current frame using three or more distinct points.

An affine transformation will map a point (x, y) to a transformed point (u, v) using translation, rotation, and scaling. The affine transformation can be written as a matrix product of a 3x3 affine transformation matrix as shown in 3.4, where m_1-m_4 are rotational and scaling values and t_x and t_y are translation values.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & t_x \\ m_3 & m_4 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.4}$$

Given at least three matching points from the model-view frame (x_n, y_n) and the current frame (u_n, v_n), the unknowns m_1-m_4, t_x, and t_y can be solved in 3.4 using 3.5 (4).

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ & & & ... & & \\ & & & ... & & \\ x_n & y_n & 0 & 0 & 1 & 0 \\ 0 & 0 & x_n & y_n & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ ... \\ ... \\ u_n \\ v_n \end{bmatrix} \tag{3.5}$$

3.6 Augmentation

Using the affine transformation, the scene can be augmented with a virtual image. Because the affine transformation is able to transform any point on the affine surface from model view space to the space in the current camera frame, a virtual image can be drawn into the frame to appear as if it is part of the affine surface. Points which

compose a virtual image in the initial model view frame are transformed using the affine transformation and drawn in subsequent frames to augment reality.

3.7 Updating Model View Frame

In order account for gradual changes in the region being tracked, new keypoints are added to the model space and old keypoints are removed from the model space. This can allow the system to have more robust affine region tracking through gradual changes in scene, such as a gradual change in lighting or camera focus. Adding keypoints can also help reduce the effect of noise in the SIFT filter which can cause keypoints to not appear during the frame when the model view was initially recorded.

In order to gradually add new keypoints, after an affine transformation is successfully computed, the model view MSER ellipse is transformed to the space of the current frame. All SIFT keypoints within the ellipse are recorded. In order to reduce the chance of adding keypoints which aren't part of the affine region (for example, if a mismatch occurs between SIFT keypoints and the affine transform is not valid), a keypoint is required to appear in the transformed ellipse in a paramerizable number of consecutive frames. Once a keypoint has appeared in the specified number of frames, it is transformed into model space from the current camera frame's space using the inverse of the affine transformation. Old keypoints can be removed when adding new keypoints using various priorities, such as least recently used or round robin.

Chapter 4

Implementation

The markerless affine region tracking augmented reality algorithm presented in this paper was implemented in two components: the Markerless Augmented Reality (MAR) library and the Lighthouse application. These two programs are described in table 4.1

Name	**Type**	**Language(s)**	**Summary**
MAR	Library	C and C++	Contains camera interfacing code, computer vision algorithms, and the augmented reality computations
Lighthouse	Application	C	An example and test application which uses the MAR library. It contains code to display the augmented scene and handle user input.

Table 4.1: Summary of Code - Summary of the pieces of code which make up the augmented reality implementation.

4.1 MAR Library

The MAR library contains the camera interfacing code, computer vision algorithms, and code which computes affine transformations. It is the portion of software that user can use to add the capability to augment reality to their applications.

4.1.1 Camera Interface

The hardware interface to the camera is incorporated into the MAR library. The camera interface is modular so that new camera support can be added to the MAR library. This design approach was chosen to allow for simpler use of the library as users would not have to interface with the camera directly and to allow code written for using a specific camera in order to aid in augmenting reality to be incorporated into the library once and not be reproduced multiple times for different projects.

At the time of writing this paper, the MAR library currently supports Video4Linux2 cameras using memory mapping and a YUYV pixel format. The library has been tested with a Logitech Webcam C160, which is the camera used in all images in this paper.

4.1.2 Computer Vision

The maximally stable extremal region (MSER) and scale-invariant feature transform (SIFT) computer vision algorithms were added to the MAR library from the VLFeat library. The VLFeat library includes an implementation of both the MSER and SIFT algorithms designed for efficiency and compatibility (12). This library is wrapped in the MAR library to abstract away unnecessary information from the VLFeat algorithm and create data structures which encapsulate data needed by the MAR library.

4.1.3 Library Interface

The MAR library contains two key interfaces necessary for utilization: the API and the configuration file.

4.1.3.1 Application Programming Interface

The MAR library API is designed with simplicity in mind, requiring a minimal amount of calls to successfully augment a scene, but providing a larger array of optional calls to fine tune augmentation. The flow of function calls to use the MAR library is outlined in figure 4.1. The API is designed to use keys to identify library objects in function calls and have all function calls which can fail return an error code to signal whether or not they were successful and, if they failed, the reason why.

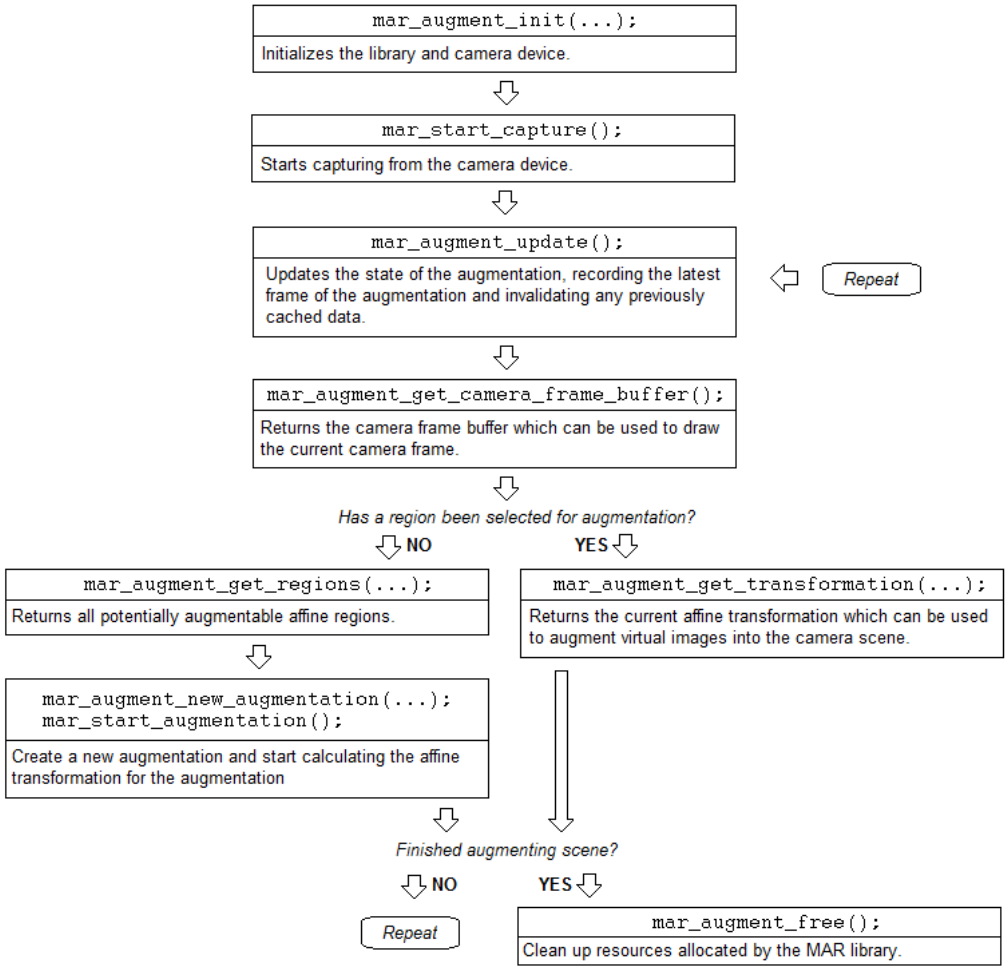


Figure 4.1: A flow diagram of the minimal API calls needed to use the MAR library.

4.1.3.2 Configuration File

The configuration file is used to parameterize many of the variables in the MSER, SIFT, and MAR algorithms. The configuration file is also the location where the camera module is configured by the user. A configuration file was incorporated into the library because it provided a simple interface for setting up the AR and camera environment across multiple programs. Having a single unified configuration file format also allows for the possibility of using a calibration program to calibrate an environment for multiple programs which use the library.

The libconfig library was chosen to manage loading the configuration file as it

provided a compact and readable file format which could be edited via a text editor (13). Parameters which are configured via the configuration file can also be configured in software via the MAR library API.

4.2 Lighthouse Application

Lighthouse is an example application which uses the MAR library. It was designed to be used as an application to demonstrate the functionality of the MAR library and to test users' configurations. It was also written to be used as a template for creating applications which use the MAR library.

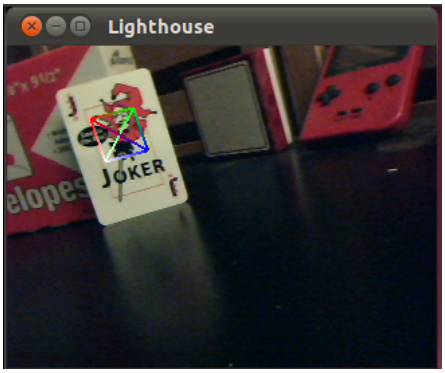
Initially, Lighthouse presents the user with areas which are potential candidates for augmentation. The user may then select an affine surface to augment using the mouse. A virtual image will be placed onto the surface coplanar to the viewing plane. The program then calls the update function from the MAR library to compute affine transformations for subsequent frames. This affine transformation is used to augment the subsequent camera frames with the virtual image. An example of a normal usage of the Lighthouse application is presented in figure 4.2.

The Lighthouse application also allows users to configure the MAR library computer vision algorithm parameters. This allows users to fine tune the augmented reality system to work better with their specific environment and desired use case. An example of a user configuring the MSER and SIFT parameters is shown in figure 4.3.

(a) Green points displayed in the frame mark regions which are potential candidates for augmentation.

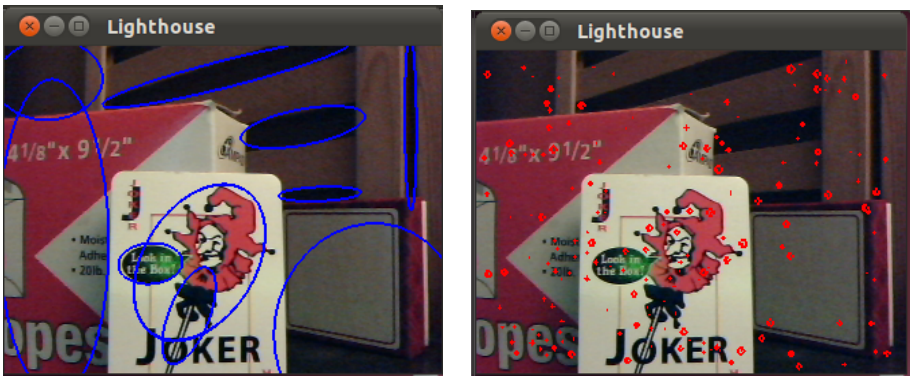
(b) The user selects a playing card as the affine region to augment.

(c) A virtual, four-colored squared is augmented into the scene in the initial frame.



(d) Using the MAR library, the affine region is tracked and the virtual image is augmented into subsequent frames.

Figure 4.2: Example usage of the Lighthouse application.



(a) A user configuring parameters for the MSER computer vision algorithm.

(b) A user configuring parameters for the SIFT computer vision algorithm.

Figure 4.3: Using Lighthouse to configure computer vision parameters for the augmentation algorithm.

Chapter 5

Results

All results were gathered running the Lighthouse application using the MAR library on a 2.2GHz Intel Celeron 900 processor running Ubuntu 11.04 using the 2.6.38 Linux kernel with a Logitech C160 webcam.

Several test cases were run to test various situations which would likely occur when augmenting a scene. They are outlined in table 5.1. The scene being augmented is shown in figure 5.1 and was chosen due to the fact it included multiple objects of different shapes in order to check that the algorithm could handle a busy scene. The region being targeted for augmentation is the flat, green coupon booklet. Images of the results are shown in figures 5.2, 5.4, 5.3, 5.5, and 5.6. The tests scenes were augmented with a rate of 12 frames per second.

For a affine region with distinguishable features, the MAR library performs well when augmenting a scene. For translational and rotational changes the algorithm was able to augment all frames. When changes in the scale of the region being augmented occurred, for the majority of the frames were able to be augmented. The ability to augment the scene decreased as the region being augmented became smaller, causing the features of the region to be indistinguishable and unrecognizable. This effect decreased when the change in size is gradual, as new keypoints are added for the smaller resolution region. Augmentation often failed when moving the camera quickly, as the film time of the camera would cause a blurred frame from which the region being augmented was unable to be identified or tracked. An example of this is shown in figure 5.7.

An additional test case was created to verify that only a recognizable portion of the region being augmented needs to be visible in order to augment the scene. This test

Name	Figure	Description
Translation	5.2	Tested the ability to augment the scene when the region being augmented is translated across the camera frame.
Rotation	5.4	Tested the ability to augment the scene when the region being augmented is rotated within the camera frame.
Scaling	5.3	Tested the ability to augment the scene when the region being augmented is moved towards and away from the camera lens.
Angle	5.5	Tested the ability to augment the scene when the angle between the camera and the region being augmented is increased.
Obscured	5.6	Tested the ability to augment the scene when the region being tracked is obscured.

Table 5.1: Test Cases - A list and description of the various test cases performed using Lighthouse.

demonstrated that the algorithm requires only a portion of the region with at least 3 matching SIFT keypoints needs to be visible in order to be augmented successfully.

Figure 5.1: A scene used to test the MAR library. The region being targeted for augmentation is a green coupon book.

Figure 5.2: A virtual, four-colored square is augmented into the scene with a camera moving across the augmented region.

Figure 5.3: A virtual, four-colored square is augmented into the scene with a camera moving away and towards the augmented region.

Figure 5.4: A virtual, four-colored square is augmented into the scene with a rotating camera.

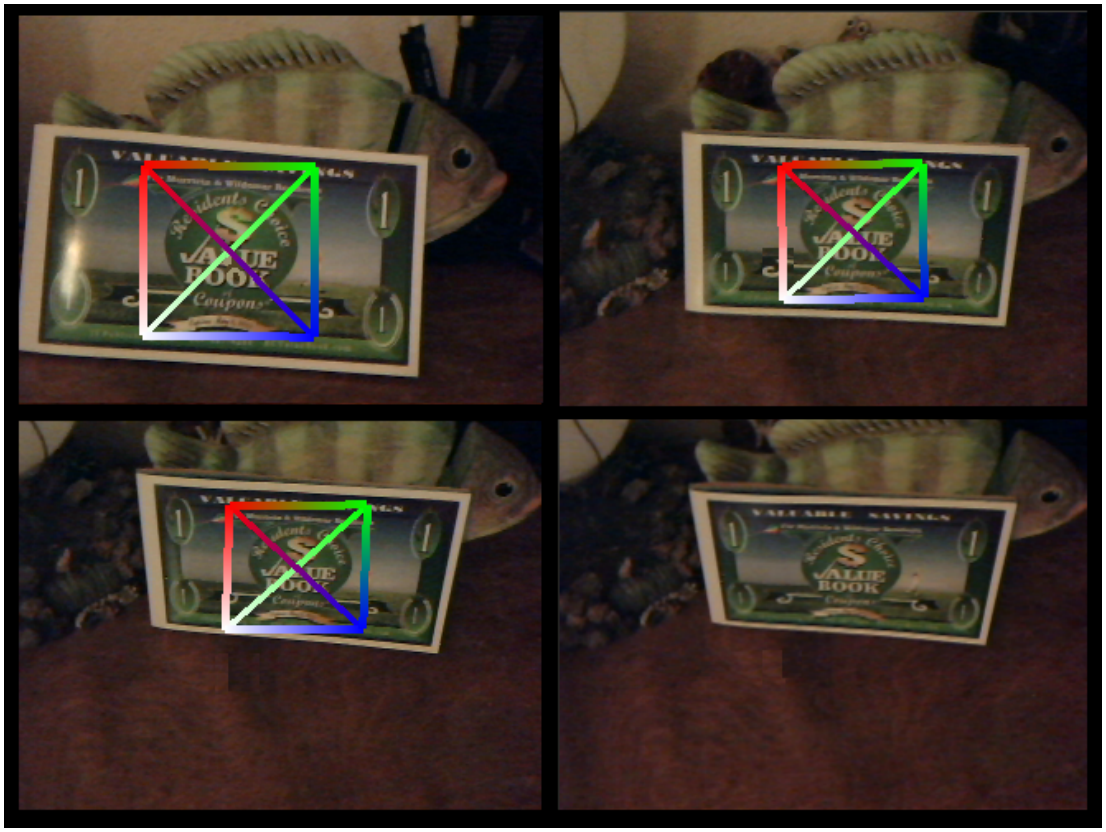


Figure 5.5: A virtual, four-colored square is augmented into the scene with the angle the augmented region is being viewed at increasing. At the last frame, the algorithm is unable to augment the region.

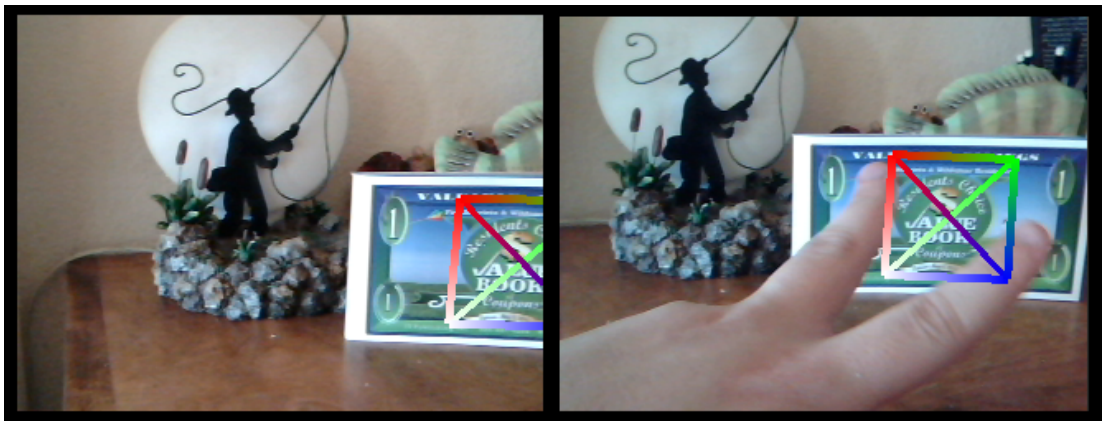


Figure 5.6: A virtual, four-colored square is augmented into the scene with the augmented region being partially obscured.

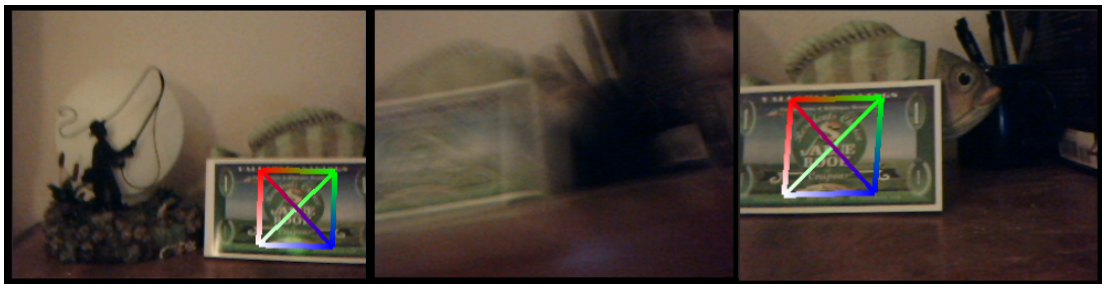


Figure 5.7: A virtual, four-colored square is augmented into the scene with a moving camera. When the camera moves fast and creates a blurred image, the implementation is unable to augment the scene as shown in the middle frame.

Chapter 6

Conclusion

The augmented reality system presented in this paper successfully augments scenes in many different environments. The algorithms is robust at augmenting regions when the region varies in translation, rotation, and scale. The algorithm is also successful at augmenting regions which are partially obscured as long as a sufficient number of SIFT keypoints are matched between the model view frame and the current frame. The algorithm is most sensitive to blurring and changes in illumination, which cause SIFT keypoints from the model view frame to differ greatly from the current frame and prevent matching. The system is also sensitive to changes in angle, as perspective effects become noticeable and negate the assumptions made about the ability to ignore the effect of perspective on the affine region.

The algorithm was successfully implemented in C and C++ with the Markerless Augmented Reality (MAR) library and Lighthouse application. The MAR library was created with the ability to be reused in multiple augmented reality projects with a modular design and simple interface. The implementation supported real-time augmentation at an average frame rate of 12 frames per second on a 2.2GHz Intel Celeron 900 processor running Ubuntu 11.04 using the 2.6.38 Linux kernel with a Logitech C160 webcam. The library was successfully designed to have the ability to support multiple cameras and have an interface for configuring the AR environment.

Appendix A

Resources and Examples

The MAR library and Lighthouse are released under the GPLv3 license. Project source code, documentation, and video examples of Lighthouse running can be found at `http://code.google.com/p/mar-library/`.

References

[1] Martin Herdina. **Wikitude**. http://www.wikitude.com/, 2011. iii, 1

[2] Philip Lamb. **ARToolKit**. http://www.hitl.washington.edu/artoolkit/, 2011. iii, 5

[3] Shotzoom Software. **Golfscape Augmented Reality Rangefinder**. http://www.golfscapeapp.com/, 2010. iii, 3, 5

[4] Matt Marano. **AFFINE REGION TRACKING AND AUGMENTATION USING MSER AND ADAPTIVE SIFT MODEL GENERATION**. 2009. iii, 2, 3, 8, 9, 10, 12, 15

[5] L. Najman and M. Couprie. **Building the Component Tree in Quasi-Linear Time**. **15**, 2006. iii, 8

[6] **ARTag**. http://www.artag.net, 2009. 2

[7] **Augmented Reality - Layar Reality Browser**. http://www.layar.com, 2011. 2

[8] Hirokazu Kato and Mark Billinghurst. **Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System**. *2nd International Workshop on Augmented Reality*, 1999. 3

[9] J. Matas, O. Chum, M. Urban, and T. Pajdla. **Robust Wide Baseline Stereo from Maximally Stable Extremal Regions**. 2002. 7, 8

[10] K. Mikolajczyk, T. Tuytelaars, C. Schmid, J. Matas A. Zisserman, F. Schaffalitzky, T. Kadir, , and L. V. Gool. **A Comparison of Affine Region Detectors**. **65**, 2005. 7

[11] David Lowe. **Object Recognition from Local Scale-Invariant Features**. **2**, 1999. 9

[12] A. Vedaldi and B. Fulkerson. **VLFeat: An Open and Portable Library of Computer Vision Algorithms**. http://www.vlfeat.org/, 2008. 18

[13] Mark A. Lindner. **libconfig C/C++ Configuration File Library**. http://www.hyperrealm.com/libconfig/, 2011. 20