IADIS International Journal on Computer Science and Information Systems
Vol. 11, No. 2, pp. 1-18
ISSN: 1646-3692

DEVELOPMENT OF GENERAL-PURPOSE PROJECTION-BASED AUGMENTED REALITY SYSTEMS

Marc Sunet[1], Marc Comino[1], Dimosthenis Karatzas[2], Antoni Chica[3] and Pere-Pau Vázquez[3]
[1]*ViRVIG Group – UPC, C/ Jordi Girona, 1-3, Barcelona*
[2]*Centre Visió per Computador, Edifici O, 08193, Bellaterra*
[3]*ViRVIG Group – UPC, C/ Jordi Girona, 1-3, Barcelona*

ABSTRACT

Despite the large amount of methods and applications of augmented reality, there is little homogenization on the software platforms that support them. An exception may be the low level control software that is provided by some high profile vendors such as Qualcomm and Metaio. However, these provide fine grain modules for e.g. element tracking. We are more concerned on the application framework, that includes the control of the devices working together for the development of the AR experience. In this paper we describe the development of a software framework for AR setups. We concentrate on the modular design of the framework, but also on some hard problems such as the calibration stage, crucial for projection-based AR. The developed framework is suitable and has been tested in AR applications using camera-projector pairs, for both fixed and nomadic setups.

KEYWORDS

Augmented Reality, Modular design, Software framework

1. INTRODUCTION

Augmented reality (AR) is a growing field that entered mainstream mainly thanks to the ubiquity of mobile devices. However, relying on the use of a mobile device for AR puts some limitations on the features that such a system may provide. Projection-based augmented reality, on the other hand, may be implemented in such a way that lets the user free hands to perform all sorts of interactions, such as virtually writing on physical documents. We are concerned on the development of systems for Human-Computer Interaction, and we will show examples of applications that provide different types of interaction using this approach. Among the different possible systems of projection-based AR, we are more interested in fixed

or nomadic setups. A key advantage is that since fixed setups do not require that the user moves cameras or projectors, such systems are easier to calibrate and less prone to accidents. Nomadic solutions [Huber et al. 2012] are in between fixed systems and mobile projected user interfaces [Huber, 2014; Willis et al., 2013]. They use pico-projectors that are placed on a fixed position for the duration of the experience. Therefore, they share the advantage of fixed systems in terms of usage: once set up, the user may have her hands free, which provides more flexibility for the interaction. On the other hand, like with mobile projected user interfaces, these are harder to calibrate. We will demonstrate our framework on a nomadic and a fixed setup.

Most setups use special purpose libraries and programs that have been developed hardware-dependent. This hinders the reproduction of an equivalent system in a different place if any of the hardware components are changed (e.g. substituting a projector because larger resolution or brightness is required).

To address this problem, we have developed a device independent, modular software framework, that abstracts the hardware layers into modules, and facilitates the substitution of the any module (camera, projector, input device) with little effort. The system also abstracts the capture and visualization modules. This way, the input can be addressed by naked hand gestures, or with other input devices, and the output can be carried out by simply drawing images or text, or with a more complex set of widgets able to simulate a full-featured virtual desktop. The key modules of our system are: *Hardware Abstraction Layers, Data Abstraction Layer, Communication Protocol, Visualization Module, Interaction Module, and Application Logic*. These components are sufficient to implement a vast amount of different setups, and most of the configurations can be achieved with little changes. Some applications will require extra modules, as we will see later when we describe some application examples.

In the following, we will describe the different parts of the system, the two different setups we built based on this framework, and demonstrate its utility using an augmented document demo application to play music. More concretely, Section 2 will introduce related work, Section 3 will describe the system, its modules, and the interaction system based on messages. In Section 4 we concentrate on two hard problems, the calibration, and the continuous update of the projected virtual widgets that form the user interface. Section 5 will introduce different setups and applications, and finally, Section 6 will discuss the achievements and conclude the paper by pointing out some lines for future research.

2. RELATED WORK

The field of augmented reality is continuously evolving. In its initial days, twenty years ago, it was a technology rarely spotted outside research centers, and most of the research was focused on the problem of being able to include a synthetic object in the real world without the object being perceived as virtual. Nowadays, with the explosion of mobile devices, augmented reality has gone mainstream, with users of all backgrounds using it for a wide variety of uses such as maps navigation, museum guides, and a bunch of professional aid applications. Most of these previous examples are commonly implemented as see-through systems. This imposes the limitation of requiring a device to be placed between the user and the reality, and sometimes its manipulation is cumbersome or poses limitations on the user freedom. For example, smartphone-mediated AR requires the user to have one hand (or two) devoted to control the system.

On the other hand, projector-based augmented reality, does not let the user freely change its location, but it may essentially free her hands so that a wider set of interactions may be available. This is one of the ideas behind the recently launched Sprout PC [Hewlett-Packard, 2015] by HP. Despite the great variety of such systems, software is far from standardized. Many examples are proprietary, and others are just research-based demonstrations, with the focus placed on the interaction or visualization features, more than the software architecture that makes them possible.

Other previous research has focused on similar problems with a lower degree of generalization, such as in the case of the CAMPAR framework [Sielhorst et al., 2006] tailored to the operating room, with a special emphasis on the synchronization of devices.

The approach by Kolomenski [Kolomenski 2013] is similar to ours in the devices used (camera, projector, IR pens…), like other systems [Linder and Maes, 2010, Mistry and Maes, 2009, Weiley and Adcock, 2013], but here we concentrate on the software modularization part. We do not focus on robot-operated systems (i. e. [Tsuji et al., 2013, Bernier et al., 2012]), since our approach is intended to be closer to a nomadic system. We also focus on projected-based AR instead of see-through approaches [Spindler et al., 2012], or systems that require external worn devices [Kim, 2012], since the environments we are interested on (e. g. public libraries), require freedom and little number of external devices. Freehand interaction promotes experimentation, and facilitates user rotation. Moreover, the lack of mobile parts improves the durability of the setups.

3. OVERVIEW OF THE SYSTEM

The system consists on a set of decentralized modules that communicate to each other with the use of a communications system (see Figure 1). In this system, several channels are open, and the modules can freely register to receive the messages of the different kinds of information.

- **Hardware abstraction layers:** A set of modules are used to hide the nitty gritty details of the hardware specific components from the rest of the system. They thus allow the substitution of a camera or projector element without affecting the rest of the system.
- **Communication protocol:** The different modules issue and listen messages that are managed by the messaging system. A key advantage of using a message passing architecture is that we can easily parallelize the system, placing different modules on different platforms (computers, mobiles, servers) in a transparent way for the system which does not need to know where each module runs from.
- **Interaction module:** The interaction module tracks the user input and issues messages corresponding to the different interactions that are detected.
- **Visualization module:** This system is in charge of the rendering of the different elements to be visualized.
- **Data abstraction layer:** It is in charge of the input and output of the data that has to be red/written from/to disk. In many cases this module will be a simple one, but in some others, it might imply working against a more complex database system.
- **Application logic:** This component is the one that defines the current running application. Again, the communication with the other components is handled via messaging.

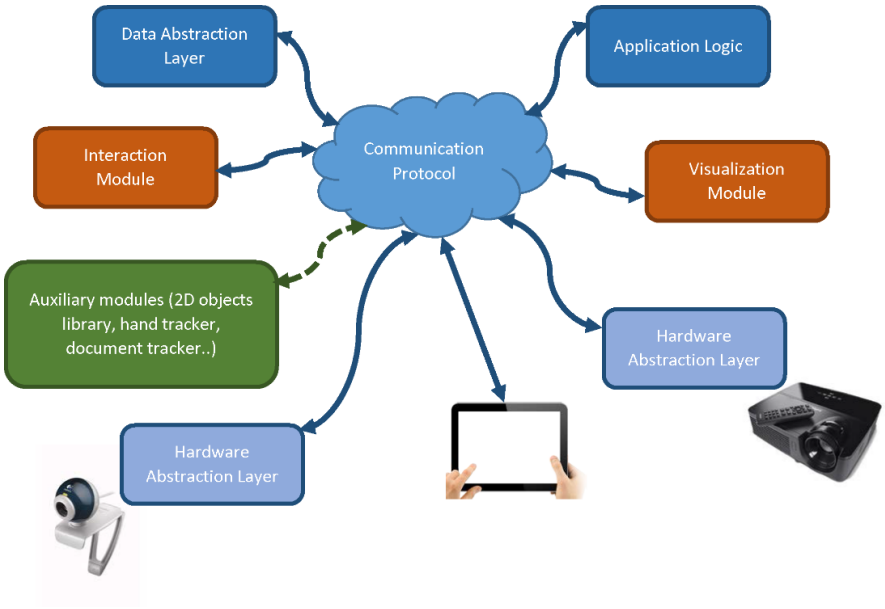


Figure 1. The architecture of our framework

The modules in blue tones are the modules that work mainly with data, while the orange ones are the ones intended for the processing of human-computer interaction. In green we illustrate potential modules that can be useful. In some of our scenarios, we may have one, none, or several of those, depending on the nature of the application. The *Communication Protocol* is central to our implementation, since it provides the means to seamlessly connect different devices and services via direct communication (as in the case of the tablet) or with an abstraction layer (projector and camera). The top-right element is the *Application Logic*, which deals with the application instructions. On top-left, we have the *Data Abstraction Layer* that provides access to persistent information.

Apart from the fixed modules, which are common for most applications, other, extra modules can be implemented. Most of these will be application-specific, and we will not deal with them in this paper. We only mention them here for completeness, and they may appear in some examples later.

3.1 Developed Modules

For the realization of our system, we developed the following modules: Projector HAL, Camera HAL, Application Logic, Communications Module, Interaction Module, Visualization Module, and the Data Abstraction Layer.

The **Hardware Abstraction Layers** are pretty simple, they abstract the access to images in the case of the camera, and the projection in the case of the projector. The camera images are queued in a buffer, and the interested modules can read them. The **Application Logic** is different in each case. However, since it makes strong use of the other modules, it commonly requires few lines of code. The **Communications Module** is the skeleton that vertebrates the whole system. All the information that is captured or generated is put into the communication

system, and the modules that require it, register to the convenient channels. It has been implemented using Google Protobuffers [Google Developers, 2015] over a ZeroMQ [iMatrix Corporation 2012] transport protocol. Protocol Buffers are a language and platform-neutral system for serializing structured data. They are also extensible, which makes them quite useful in many communication systems. ZeroMQ is a messaging transport protocol is a transport layer protocol for exchanging messages between two peers over a connected transport layer such as TCP. In our system, all the modules that may generate data or commands puts messages into a channel, and the modules that wish to read this information only have to register to those channels. This way we achieve a hardly hierarchical structure that is easy to maintain and whose modules can be replaced simply.

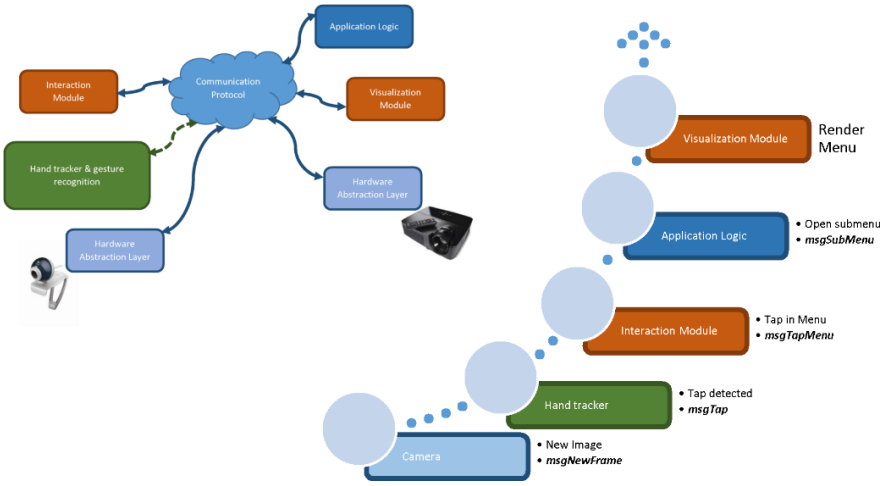


Figure 2. A full example of message passing throughout the system. Upon a hand gesture by the user, a tap, the system generates a submenu. The different modules issue messages into the system and the application logic decides the actions (e.g. opening a submenu) also through message passing

We can see a working example of this protocol in a subset of the modules in Figure 2. In this example we show how a tap in a menu can be interpreted by our system and generate a submenu. The top-left image is a clipped version of the whole system, where only the modules intervening in the tap processing and reaction are shown. The bottom right part, encodes the different modules with the same colors to facilitate the reading. First, the camera generates a new frame where the hand is tapping on a concrete region of the working space. The camera HAL, as expected, generates a message with the image as the contents. The hand tracking auxiliary module reads the image and detects a tap. This tap is then passed to the system through another message (*msgTap*). The interaction module receives this message and passes the information to the application logic, which is aware of the elements that have been rendered (this can be achieved directly or through previous check with the visualization module). Then, the application logic decides that a new submenu must be opened, and passes this information to the visualization module as a new message that carries out a command, *msgSubMenu*.

For the sake of the reader, we have avoided a thorough description on the parameters and the current format that each message may carry, but there are easy to imagine.

The **Interaction Module** is the one in charge of getting the input from the user and convert this input into commands or information that is broadcast to the interested modules. We have

implemented it in two different flavors: hand-based gestures, and IR-pen gestures. In Section 5 we provide more details on the interaction modules. The **Visualization Module** renders all the objects that are projected onto the working area. We have created a lightweight UI library built on top of SFML graphics tool [Gomila, 2015]. SFML is a multi-framework library that provides a simple interface to various multimedia components of the operative system. The principal characteristic of the UI we have developed is that it allows the 3D rendering of 2D widgets in order to correct the projection deformation induced by an arbitrarily-tilted projector. Moreover, it also serves as a pipe between the gesture module and the application, namely it detects on which widgets the gestures are performed and forwards this information. Finally, an image-based positioning algorithm has been implemented in order to adequately place the widgets in the working space, that is able to cope with document movements. As this is explained in Section 5.2, these widgets are intelligently placed according to the free space in the working region, and move accordingly if the reference anchor points are displaced.

The backoffice system, **Data Abstraction Layer**, deals with persistent data. In one of the use cases we developed, for example, we dealt with documentary information. As a result, a database was required, in this case we used an Oracle database of documents with hand generated annotations. The result of the interaction with the application also generated a set of new annotations. These were also stored along the database. This required a module to handle this data. All of this can be abstracted from the application, and in some particular cases, where the data lacks the generalization of the framework we propose, may require slightly more effort, but most common data will be treated simply by a generic data abstraction module.

These developed modules are common to all applications and only little modifications to to some of the systems may be required if we change the input or output devices. In our case, we did not have to change anything for the transition of our nomadic system to the fixed system.

Each application will use all of the previous modules, but the *Application Logic* is dependent on the application to be developed. Therefore, it will be different for every application, but the other components can be simply used as is. Together with these modules, we found that other components can be commonly required in many scenarios, these are enumerated here:

- **Rendering subsystem:** For the visualization part, several strategies can be used, in our case, we developed a library of visual objects and a library of visual feedback elements.
- **Document tracker:** When the application scenario is intended to simulate a virtual desk, the tracking of documents becomes a must. Therefore, this module may be of great utility.

Some of the scenarios we worked with throughout the development of the project dealt with documents. In some cases, the scenario consisted in augmenting the document, by adding some information on demand, and in some other cases, the document was used as input (for identifying or capturing images, etc.). In all these cases, apart from the concrete software for capturing or identifying elements, there is the need of tracking the document in the scene. Therefore, a simple document tracker was implemented and used to provide information both for the input (e.g. capturing information) and output (e.g. projecting extended information onto the document) systems.

3.2 Interaction

The interaction with our system can be carried out using two different techniques: hand-based, and with IR-pens. The most important advantage of the hand-based interaction is the lack of external elements. However, the most important limitation, is the hand segmentation. Since each user may have a different skin color, and the illumination conditions change along the day, the hand interaction lacks some degree of robustness. This is especially true, and may be a problem, for nomadic systems. Unfortunately, since in most places we are not able to control the illumination totally, recalibration may be required, that is why our self-calibration system is almost automatic and can be triggered if needed. For fixed systems illumination changes can also be a problem, although not as severe, since the conditions change less frequently. Ideal conditions should ensure the light is constant along the day, which is not common in most places, so if drastic changes occur, as said, the calibration can be run upon demand.

On the other hand, when using IR-pens, we will require a third camera with its extra calibration stage. This extra element, however, still keeps a low cost for the hardware setup. Moreover, as compared to the hand gesture interactions and detection, IR-pens do not represent a major problem, since IR is more robust to illumination changes. Moreover, the calibration stage is much simpler, due to the same reasons and therefore is quite straightforward.

In any case, the modular design permits the gestures to be implemented in the interaction module, independently on the way they are captured. That is, the same gesture can be performed by a hand or by an IR-Pen, in our case, these two trackers were implemented, but it would be easy to perform equivalent gestures with other external devices such as the MYO Armband or the Leap Motion, the only issue is the concrete gesture tracker, that is, encoding the same gesture using different devices, but the interaction module remains the same.

In order to properly determine the gestures, and to maintain uniformity, these are performed in three stages, as shown in Figure 3:

Initialization: The gesture is detected and identified. Initial visualization cues are provided.

Updating: Gesture is performed by the user. Visual cues identify and communicate the gesture to the user.

Finish: Gesture finishes. If an action is linked to the gesture, it is triggered.

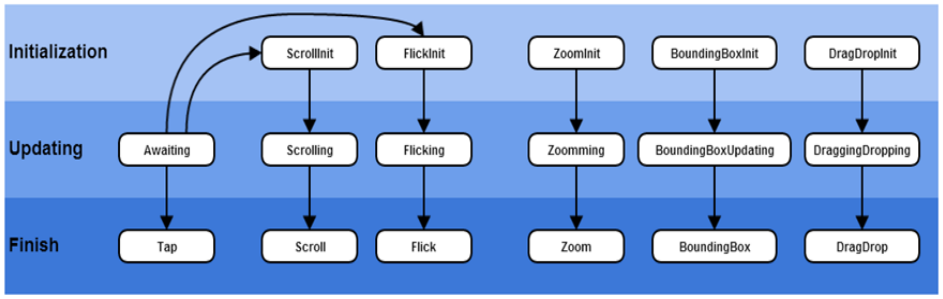


Figure 3. The different stages of the gestures that can be determined by our system

Note that the figure determines a state machine that is updated throughout the gesture tracking. Each of the boxes correspond to messages that the gesture tracker will issue to the system. Therefore, the interested modules can read them and act accordingly. In our case, the

Visualization Module is aware of the gestures being carried out and generates the appropriate visual cues to inform the user that a certain gesture is being detected and where.

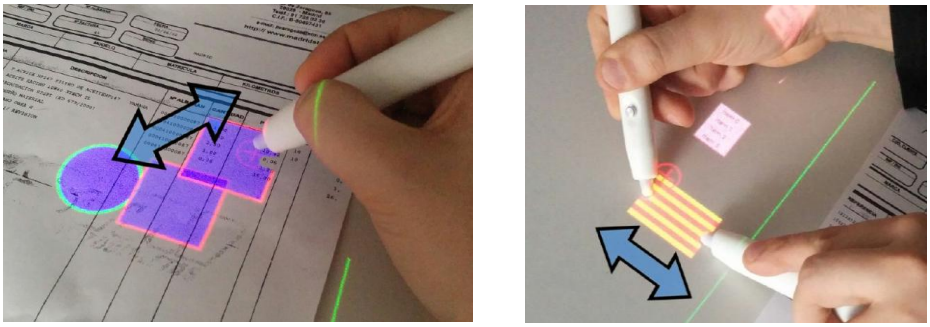


Figure 4. Interaction with the IR-pen system with one and two pens respectively

For many of the gestures, especially when they last long, the visualization system will provide some visual cue to help the user understand that the gesture has been determined. The different visual cues may go from projecting the input point as a circle, to more elaborated effects such as marking a certain button or menu entry as selected. Since in our case we have implemented a set of widgets that cover the main elements of a virtual desktop, many of these visual cues are implemented as different states of the widgets (e.g. selected vs non-selected). Other effects are simply provided with the interaction of the widgets. For example, when performing a drag-and-drop operation, the element is moved as the user drags its virtual position. This is shown for instance in Figure 4-left. In the first case, a drag-and-drop operation (indicated by arrows on the left) is being carried out by the user. The visual cue that communicates the behavior is the actual translation of the rectangle in purple. We can also perform other two-hands operations such as scaling, as shown in Figure 4-right. The displacement of the pens is also indicated here with the blue arrow, and the user will see an effective incremental resizing of the object while the gesture is not finished.

As said, the gesture management module has been implemented agnostic of the interaction element. We have designed a set of one-hand or two-hand gestures that include simple taps, swipes, and so on, that can be implemented both by hand or IR-pen. In both cases, the user can work with one hand/pen or with two, and the detected gestures are equivalent for the hand and the pen. The different gestures that are detected when operating with a single hand are shown in Figure 5. These consist in: Tap, scroll, flick, and drag-and-drop.

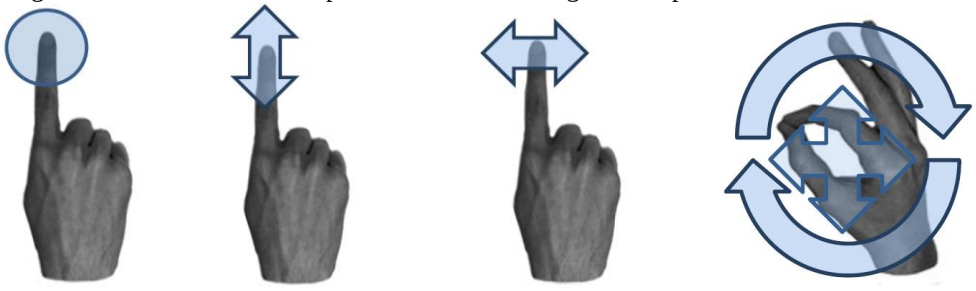


Figure 5. The one-hand gestures that can be detected by our system: tap (the user clicks gently on the working surface), slide (the user moves vertically the finger), flick (the user moves horizontally), and rotate (the user rotates the hand after the index and thumb have contacted)

Some tasks, such as zoom, are more comfortable to generate using two input points. In order to detect them, the users have to use the two hands. In Figure 6 these gestures are shown. In order to determine the input position, we track for a contact between the index and the thumb.

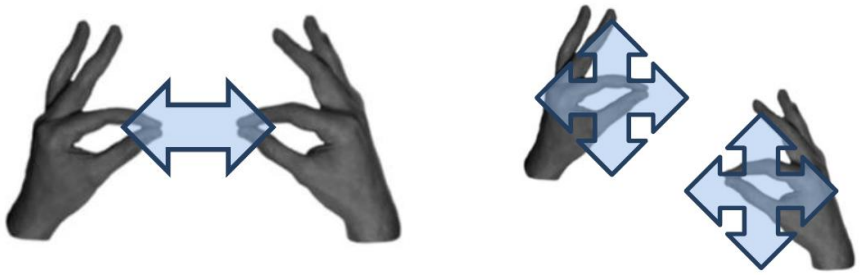


Figure 6. The two gestures that can be detected by the system when working with two hands (zoom, and drag and drop). The difference between the gestures lies in the relative position of both hands: if the contact points are oriented to each other (e.g. in the same horizontal position, as seen from above), the movement is classified as zoom, otherwise, it is classified as a dragging gesture

4. IMPLEMENTATION DETAILS

In this Section we will concentrate on the development of two crucial stages of the system: the calibration and the tracking. The calibration stage is very important because it will define a set of parameters, such as the transformation matrices, that will allow us to render the objects in place. The document tracking module is responsible of determining the position of the physical objects that will are used as input for the projection-based system. In our case, we focus on document-based systems, so the element to track is a document, and the virtual UI is rendered with respect to the document position. As a result, we need to track the document in realtime, and the elements of the UI must be placed accordingly, taking into account the limitations in the working space.

4.1 Calibration and Working Space Definition

Since we are using a camera-projector pair AR system, we need to know the relative position of the camera and the projector. Therefore, a calibration procedure is necessary. In our case, since we are not fixing the features of the camera or the projector, we cannot impose a fixed camera and projector positions. Thus, we need to calibrate one with respect the other. Of course, they must be placed in reasonable positions, such that the camera and the projector share approximately the same vision volume.

This is why we have developed a multi-stage procedure that calculates the homographies and includes the definition of the virtual working region. This way, if the camera-projector pair is repositioned, or one of the elements is changed for another with different properties, we can re-calibrate and continue working without any modification to the other framework modules.

The calibration procedure performs the following steps:

1. Calculate the homography between projector and camera
2. Estimate camera parameters
3. Detect orientation and size of the valid working area
4. [Calibrate other external gesturing elements, e.g. IR pen]
4. [Set up other input devices, e.g. tablet]
5. Communicate the homographies

For the first step, we generate three different rectangles of different colors that are projected by the projector and captured by the camera. Since the camera captures many images, we perform a background subtraction to ease the segmentation. Then, a segmentation process ensures that the rectangles are properly captured. In order to do so, we further optimize the rectangle determination by using a corner detection algorithm. The fact that we know the geometry of the objects and their positioning, lets us estimate the camera-projector homography.

Once we know the camera-projector homography, we need to determine the size and orientation of the working area, that depends on the properties of the camera and projector, and their positioning. Since we want to deal with the possibility of a nomadic system whose configuration can be changed any time it is set up, we do not perform any assumption on the working area. For a proper detection of its size and orientation, we use a template document that serves as marker. The document is only assumed to be present in the documents database. The document identification is performed on the fly using SIFT [Lowe, 2014] for feature extraction. Once the features are detected, we use FLANN library [Muja and Lowe, 2009] for indexing and retrieval of documents. The choice of SIFT was due to its efficiency and because it has been previously demonstrated as suitable for documents that contain text and images [Nakai et al. 2005]. We determine the maximum valid working space by fitting the largest rectangle in the limits of the camera viewing space and projector space.

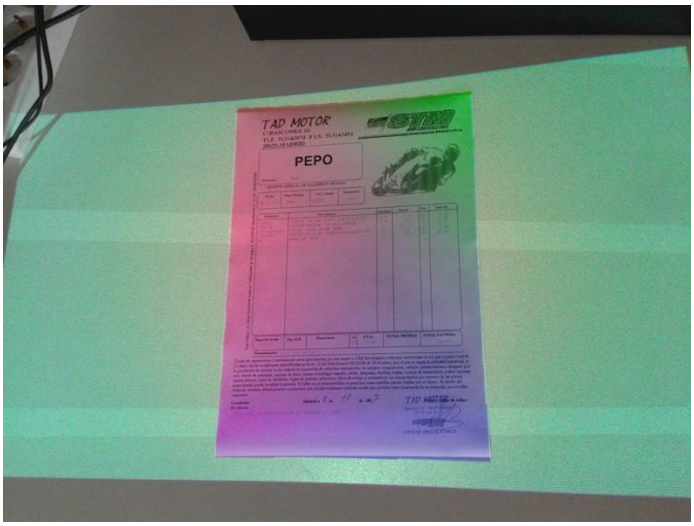


Figure 7. The calibration procedure illustrated. In this case, we render a quad over the document to show where it was detected by our system

The document template serves two purposes: the definition of the working area, and as an anchoring position for the UI projected widgets, as will be shown later.

In some systems, we have used alternative input devices such as IR pens. Due to the use of infrared light, IR pens are less prone to detection problems in varying illumination conditions. This, together with the fact that the devices are cheap, makes them widely usable in different environments. For the use of such devices, they must be calibrated too. In our case, we perform this calibration right after the working area calculation. For IR pens, we render a set of four markers (crosses) that the user must click with the IR pen. Once the user has clicked to all of them, we use their positions to calculate the corresponding homographies.

Finally, the system has generated the whole set of homographies necessary and these are communicated to the application logic through the messaging system.

4.2 Document tracking and UI Update

As already commented, we have developed augmented document systems, where the central object of interaction is a document that must be filled, completed, or analyzed using the projection-based AR system. In order to interact, we generate virtual widgets, that are projected around or on top of the document upon user interactions. For example, a tap on a concrete region may generate a menu, clicking on a menu may generate a submenu, etc. We want the user to be able to interact seamlessly, and the system to be able to react to the user interactions as well as the potential modifications of the environment. These modifications can appear in the form of the document repositioning, being for a proper reading, or just because the user accidentally (or purposely) changes the document position.

4.2.1 Algorithm Overview

Since most of the interaction tools are projected, the positions we generate them correspond to logical positions of the UI. And these are all built around the document that serves as the basis of the interaction. When a new widget is created as a result of an interaction (e.g. menu creation), the system defines a *preferred position.* However, this position may be unavailable due to lack of space or because it is already occupied by another widget. As a result, we have created a greedy algorithm to quickly determine widget locations and reposition them when the document placement changes.

Given a rectangular layout L, a set of rectangular widgets W and set of preferred locations $P = \{p_w \mid w \in W\}$, the widget positioning problem can be formulated as finding the positions $P^* = \{p^*_w \mid w \in W\}$ such that there is no overlapping between the widgets w and that they are contained within L. Note that this problem does not always have a valid solution.

Without loss of generality, we consider that the bottom-left corner of our layout L coincides with the origin of coordinates $(0,0)$ and, therefore, we characterize L by giving its width w_L and its height h_L. We characterize each widget $w \in W$ by giving its width w_w its height h_w and its preferred location $p_w=(x^*_w, y^*_w)$. The preferred location of a widget is the 2D point where we ideally want to pin the the bottom-left corner of our widget. With these definitions we can formalize the widget positioning problem as finding P^* such that:

$$argmin_{P^*} \sum_{w \in W} E(p_w, p_w^*)$$
$$subject\ to$$

$$\forall w \in W, x_w^* \geq 0, y_w^* \geq 0, x_w^* + w_w \leq w_L, y_w^* + h_L \leq h_L,$$

$$\forall w \in W, \forall w' \in W \setminus w, x_w^* + w_w < x_{w'}^* \wedge x_{w'}^* + w_{w'} < x_w^* \wedge y_w^* + h_w < y_{w'}^* \wedge y_{w'}^* + h_{w'} < y_w^*$$

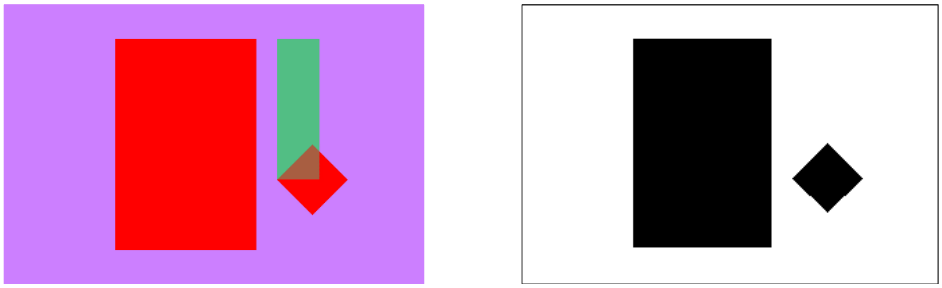
where E is a given energy function. From here on we will be basically using euclidean distance as E.

Solving the described optimization problem is indeed very time-consuming. Nevertheless, we need a real time scalable algorithm, which comes at the expense of not always finding the optimal solution. For this, we will first move into the discrete domain by converting L into a grid of w_r x h_r cells. Therefore, from now we will characterize widgets using cell units instead of continuous units. Let $\omega_w = floor(w_r * w_w / w_L)$ be the discrete width for widget w and $\lambda_w = floor(h_r * h_w / h_L)$ its discrete height. In practice, this induces a relaxation of the problem; we will tolerate overlappings of up to $w_w - \omega_w * w_L / w_r$ and $h_w - \lambda_w * h_L / h_r$.

Next, we will introduce a new class of widgets which will always remain "fixed". These widgets will always be placed at their preferred locations even if this induces overlaps with other fixed widgets. Finally, we will adopt a greedy approach to find the placing for the rest of the widgets. Given a set of fixed widgets F, we will iteratively find the best position for a given widget $w \in W \setminus F$, place it there and treat it as a fixed widget for the following iterations. Notice that the scalability of this method arises from the trade-off between time and precision when varying w_r and h_r.

4.2.2 Implementation Details

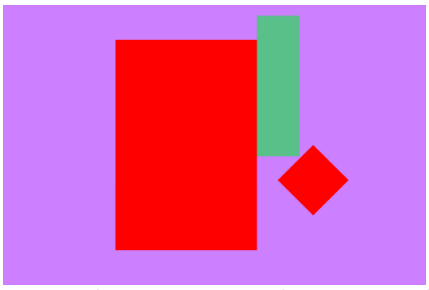
We employ an auxiliary data structure to keep track of available space. Basically, we keep a binary matrix of dimension w_r x h_r . When placing a widget, we mark the cell it occupies as unavailable. To find the optimal position for a given widget w, we erode the availability map using as structuring element the shape of the widget. Then, we select as p^*_w the cell from the eroded availability map that minimizes the euclidean distance to p^*_w. Following, we place w, update the availability map and repeat for the remaining widgets.



a) Initial position with two fixed widgets (red) and the next widget (green) to be fixed.

b) Availability map after placing the fixed widgets

c) Eroded availability map. The green widget can be placed in any of the white cells without overlapping the others



d) Final configuration with the found position for the green widget

Figure 8. The algorithm for automatic widget placement

Initially (top left), two widgets (in red) have a fixed position and we try to place the new widget (in green). The availability map is first created (top right) from the fixed widgets. Then, an eroded version (bottom left) defines the available regions for the widget. Note that these actually indicate where we can put the bottom left corner of the widget. Finally, the new position for the widget (bottom right) is determined.

In our implementation we allow positioning widgets with respect to other widgets (e.g. submenus). When the parent widget is moved, we update the children's absolute positions and use these as input of our algorithm. Also, we allow the creation of widgets which are rectangles but not necessarily aligned with the edges of the layout.

Basically, we place several widgets hanging from the main document, which is the absolute reference. As the document in the working region, the position of the widgets that hang from it are updated. This is valid also for rotations, as it is shown with a real case in Figure 9.

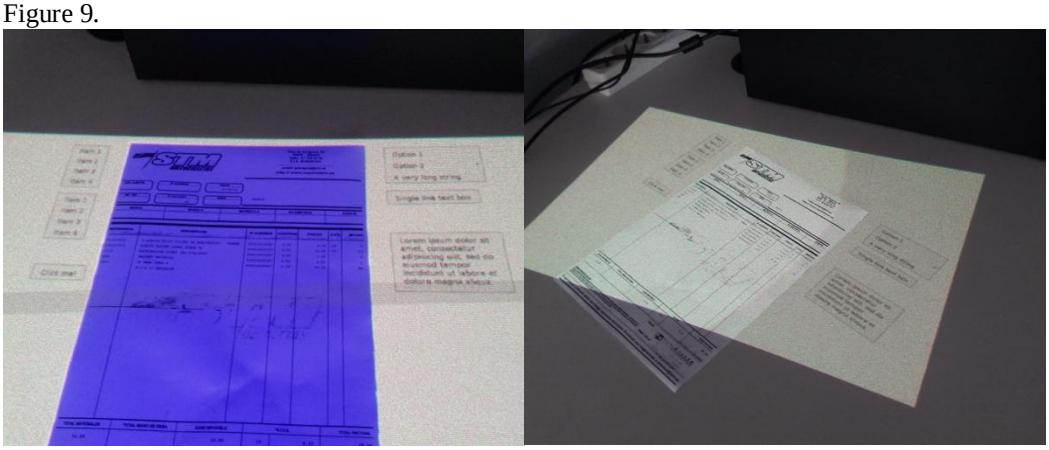


Figure 9. The widget placement algorithm in action

The left image shows the document that has been used as virtual anchoring for the projected UI widgets. This corresponds to a regular configuration of the working area. On the right we show a drastic modification of the document orientation and how the system reacts by moving the projected widgets of the user interface to the proper available locations. Note how we keep the orientation with respect to the reference document, to facilitate reading.

5. EXAMPLE SETUPS

In order to test our system, we have developed two different setups, a nomadic and a fixed system. With the aid of the previously described modules, few extra packages were required.

5.1 Nomadic System

The nomadic system, as stated previously, was intended to show the capabilities of a projection-based augmented reality system paired with a camera and with hand or IR-pen interaction. The main features of the system were the portability and the flexibility in mount therefore, several aspects had to be implemented in an adaptive way, which greatly increases the difficulty of the software development. These, were the main decisions we had to take:

- **Hardware:** The hardware, if the system has to be moved, must be light and easy to mount. We chose a small Logitech C615 and a pico-projector from MicroVision, the MicroVision Showwx, together with an articulated arm that had a heavy basis to be stable on the table.
- **Calibration:** Since the system could be moved, the calibration system may not rely on fixed environment, so several elements such as illumination, background, and so on, must be taken into account.

The selected **hardware** with the initial setup is shown in Figure 10-left, where we can see an example of the first version of the nomadic system. A second version, with a more professional look, was also created (see Figure 10-right) with the use of a shell printed using a 3D printer. The selection of the hardware was based on a thorough analysis of capabilities of the different devices (projector and camera) in the market, in terms of quality, image resolution, distance of projection/capture, and so on. Throughout all the process, we had in mind that the system should be affordable, since transporting such a system is a delicate operation and accidents might occur, devices should be simply replaced, and affordable. With this idea leading our analysis, we came out with the projector and camera selected, that offered a good balance between quality and cost, making the cost of both add up to less than 500$. A second component was a PC. In this case, a portable device, with no special features, a commodity PC is able to run the whole system with no problem. The body of the system consisted on a modified lamp arm and a couple of plastic pieces printed on a 3D printer. The total cost of the nomadic system was around $1500.

The main advantages of these two hardware components are their balance between space, weight, and quality. An important advantage of the projector is that it is always-in-focus, which facilitates the calibration process for non-fixed systems.

Figure 10. The initial setup of the nomadic system and the 3D printed shell to enclose the cameras and projector

The *calibration* was a second, important issue. Since the nomadic system can be built in different places, we need a calibration process that is able to adapt to different lighting conditions. There are two different aspects (that involve many variables) that may be taken into account when calibrating such a system: a) **Illumination:** Conditions may change between different places, so the calibration system must be as robust as possible to lighting changes, and b) **working area conditions:** The size, color, and orientation of the working area may change due to physical limitations. Although no large room is necessary to fit all the elements, the available space may change from place to place.

So we created a calibration method, described above, that can be triggered as required (e.g. if the environment where the system is running is prone to large changes of illumination along the day, we may run it in the morning in the afternoon), since it also uses the same messaging system to communicate the different found matrices. Once the homographies have been calculated, they are broadcast to the whole system, so that the application logic, the visualization system, and whoever is interested in those, can read them. From then on, however, all the communication referring to virtual elements to be projected, will be carried out in working space. The visualization system is in charge of positioning the elements properly, and even repositioning them automatically in order to avoid occlusions if necessary, as described in Section 4.2.

5.2 Fixed System

The fixed system is composed by a Basler ac2500-14gc camera and a projector InFocus IN 3138HDA, which provides HD projection with 4000 ANSI lumen. Moreover, this system also uses an IR camera, which is basically a very similar camera, a Basler ac2500-14gm with an IR longpass filter (850nm, M27 x 0,5mm) for the IR Pens. The devices here, in contrast with the nomadic system, can be of higher quality, and the distance of the projector to the surface is of

2.2m, and the area of projection is about 1.1m wide (16:9). The main difference of the fixed system with the nomadic one is the intended use. The objective in this one is to have a living lab in a public library where the users may experiment with projected augmented reality technologies. More specifically, the users will be, mainly, children, and therefore, we have developed a set of small toy applications to be used by the children.

In Figure 11-left we can see how the fixed system looks. The fixed system uses the same, previously enumerated software packages to perform all the tasks. The only difference with the previous system is that the projectors and cameras have a larger resolution and can be placed at a larger distance, so that we can build a fixed system that is less prone to accidents.

Figure 11. Left: Fixed setup in a public library. The fixed setup has the projector/camera pair fixed in a structure that is attached to the ceiling. This way, the users have free space around them to experiment and freely perform gestures to interact with the application. Right: The music toy application

One of the toy applications we have developed is a music player. In this application, the user shows a music score that the system is able to detect and interpret. The user only has to select the note, and the system plays it. The system can also change the instrument that plays the music by letting the user choosing among a set of predefined instruments. Everything happens in a very user-friendly way, by providing most of the options as icons the users may select. We can see an example of this application in Figure 11-right, where the projected elements such as the piano tiles or the instrument icons are all interaction widgets. The system tracks the document position, so if it changes, the widgets are automatically rearranged accordingly. The user can choose the instrument, play a note, or play the whole song.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a software framework tailored for the rapid development of augmented reality setups that are based on the projector-camera pair. The system is highly distributed and all components execute individually and communicate through a communication system based on Google Protocol buffers over a ZeroMQ transport layer. All the modules communicate using a protocol defined by the Communications Module that is the center of all the system. We can even attach external devices (e.g. a tablet) to the communications system. The development of a simple application with our new framework can take as few as a week if no other hardware elements have to be added. It consists basically on reprogramming the *Application Logic* module to fulfill the users' needs. Besides the general modules, we have also implemented other modules for document tracking, widget rendering, and so on, that are easily integrated and can be shared by other modules. In future we want to continue developing the system, but concentrating on new features that may be driven by new example applications, or new input devices.

ACKNOWLEDGMENTS

The authors acknowledge the support for this project provided by TIN2014-52211-C2 by the Spanish Ministerio de Economía y Competitividad with EU FEDER funds.

REFERENCES

Bernier, E., et al. 2012. The MobilAR Robot, Ubiquitous, Unobtrusive, Augmented Reality Device. *Biennial Conference on Engineering Systems Design and Analysis. American Society of Mechanical Engineers*, 375–381.

iMatix Corporation. 2007-2012. ZeroMQ Message Transport Protocol. *http://rfc.zeromq.org/spec:23*, checked June 2015.

Google Developers. 2015. Protocol Buffers, *https://developers.google.com/protocol- buffers/*, checked June 2015.

Gomila, L.. 2015. Simple and Fast Multimedia Library.. *http://www.sfml-dev.org/*, checked June 2015.

Hewlett-Packard. 2015. Sprout Official Page. *http://sprout.hp.com/us/en/*, last checked June 2015.

Huber, J., 2014. A Research Overview of Mobile Projected User Interfaces. *Informatik-Spektrum*, Vol. 37, No. 5, pp. 464–473.

Huber, J. et al., 2012. LightBeam: Nomadic Pico Projector Interaction with Real World Objects. *CHI '12 Extended Abstracts on Human Factors in Computing Systems*. ACM, New York, NY, USA, pp. 2513–2518.

Kim, D. 2012. Digits: freehand 3D interactions anywhere using a wrist-worn gloveless sensor. *ACM symposium on User interface software and technology*. ACM, pp. 167–176.

Kolomenski, A., 2013. Realization of a spatial augmented reality system– A digital whiteboard using a Kinect sensor and a PC projector. *Ph.D. Dissertation*. Texas A&M University.

Linder, N. and Maes, P., 2010. LuminAR: portable robotic augmented reality interface design and prototype. In *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, pp. 395–396.

Lowe, D. G., 2004. Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision*, Vol. 60, No. 2, pp 91-110.

Mistry, P. and Maes, P., 2009. SixthSense: a wearable gestural interface. *ACM SIGGRAPH ASIA 2009 Sketches*. ACM, 11.

Muja, M. and Lowe, D. G. FLANN, fast library for approximate nearest neighbors, in *International Conference on Computer Vision Theory and Applications (VISAPP'09)*, 2009

Nakai, T., Kise, K., and M. Iwamura. Use of affine invariants in locally likely arrangement hashing for camera-based document image retrieval, in *Document Analysis Systems VII*. Springer, 2006, pp. 541–552.

Sielhorst, T. et al., 2006. Campar: A software framework guaranteeing quality for medical augmented reality. *International Journal of Computer Assisted Radiology and Surgery*, Vol. 1, Suppl. 1, pp. 29-30.

Spindler, M., et al., 2012. Towards spatially aware tangible displays for the masses. *Workshop on Designing Collaborative Interactive Spaces for e-Creativity, e-Science and e-Learning at AVI*, Vol. 12, No. 5, pp. 1213-1225.

Tsuji, K. et al., 2013. Robust projection method for a mobile robot with a camera and a projector based on a structured-environment approach-Experiments of the modified method considering a distance between a marker and a robot. *Robot Motion and Control (RoMoCo), 9th Workshop on IEEE*, pp. 36–41.

Weiley, V. and Adcock, M., 2013. Drawing in the Lamposcope. *ACM Conference on Creativity & Cognition*. ACM, pp. 382–383.

Willis, K.D., Shiratori, T., and Mahler, M. Hideout: mobile projector interaction with tangible objects and surfaces, in *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 2013, pp. 331–338.