- ORIGINAL ARTICLE -

Implementation of an Open Source Based Augmented Reality Engine for Cloud Authoring Frameworks

Implementación de un Motor de Realidad Aumentada Basado en Código Abierto para Frameworks de Authoring en la Nube

Nahuel Mangiarua[1], Jorge Ierache[1], and María José Abasolo[2,3]

[1] *DIIT, National University of La Matanza, San Justo, Buenos Aires, Argentina*
{nmangiarua, ierache}@unlam.edu.ar
[2] *Faculty of Informatics, National University of La Plata, La Plata, Buenos Aires, Argentina*
[3] *Scientific Research Commission of the Buenos Aires Province, Buenos Aires, Argentina*

Abstract

In recent years the technology around Augmented Reality has grown considerably, in particular, cloud based always online solutions. In this paper we present a pipeline model and sample implementation that shows how an Augmented Reality engine can be built by leveraging advances in open source algorithm implementations. We also show how such an engine can be effectively integrated with cloud authoring tools to take advantage of the network connectivity and its computing power without an always online requirement.

Keywords: Augmented Reality, Cloud Authoring Framework.

Resumen

En tiempos recientes la tecnologia al rededor de la Realidad Aumentada a crecido considerablemente, en particular, las soluciones con conexion constante basadas en la nube. En este trabajo presentamos un modelo de pipeline y una implementacion de ejemplo que muestra como un motor de Realidad Aumentada puede ser construido aprovechando los avances en algoritmos de implementacion abierta. Tambien mostramos como dicho motor puede ser efectivamente integrado con herramientas de autor en la nube para sacar ventaja de su conectividad y poder de computo, pero sin requerir una conexion continua.

Palabras claves: Realidad Aumentada, Framework de Autoria en la Nube.

1 Introduction

Augmented Reality (AR) consists in the creation of an environment in which information and virtual objects are superimposed to reality, offering the user an enriched experience without interfering with his natural perception. It can be used to expand our senses, to define a direct or indirect view of a real world environment, whose elements are combined with virtual elements, such as texts, images, audios or videos to create a mixed reality in real time [1]. AR does not always add elements to the real world, but it can also be used to remove parts of it, occluding a physical object from the view and replacing it with relevant information [2]. As the AR Gartner hype cycle described by Lens-Fitzgerald [3] states, AR can be categorized in several types or levels of increasing availability and seamlessness. From type 0 to 3, the first type refers to linking real world elements to digital content by reading an special code like a QR or barcode. Type 1 instead can effectively augment the reality by detecting a marker and projecting virtual elements relative to its spatial position. On the other hand, type 2 and 3 augment the physical reality without relaying in markers, making use of precise gps, environment mapping and wearable devices focusing on providing a seamless experience In recent years AR technology has seen a wide amount of progress from both academic research and business. There are various libraries and frameworks for exploiting augmented reality environments such as ARToolKit [4], Vuforia [5], Layar [6], ARCore [7], ARKit [8]. Furthermore, several cloud based authoring platforms have been developed such as the recently acquired by IBM Aurasma [9], Augment [10], Aumentaty [11], Zappar [12]. These authoring tools allow users to upload virtual content to augment the reality, linking it to an AR marker, GPS position or trigger image. Another example of an authoring tool is our Virtual Catalogs System [13][14] which focus on reducing the required technological knowledge of the final user to effectively augment content. Within this frameworks and authoring tools, a considerable fraction of the effort has been put on improving pure cloud solutions and pushing towards AR of type 2 and 3. In particular, markerless AR that maps the environment into some 3D representation. By anchoring virtual objects at a point in this mapped space, the reality can be augmented when that point is seen through a

wearable or screen. Despite the huge potential of this type of AR and the incredible improvement in network connectivity and latency, we can still find a considerable number of applications where type 1 AR on a mixed cloud-local platform offers the best solution. In parallel with the advances discussed previously, open source algorithms and techniques have continue to be implemented and improved making it possible to put together a type 1 AR engine able to work offline when needed, while taking advantage of the cloud when available. In this paper we present a sample implementation of such and engine and its integration with our existing cloud authoring framework. At the same time we will be measuring the performance of different algorithms available on the OpenCV library [15] relevant to each step of the proposed pipeline. The rest of the paper is organized as follows: in Section 2 we describe the architecture of our existing cloud based authoring tool, a proposed pipeline for a type 1 AR engine and their integration; in Section 3 we proceed to test the pipeline to identify the most promising algorithms for its implementation. We finally conclude with our findings and a discussion about scalability issues and related future work.

2 Augmented Reality Cloud Authoring Framework

2.1 Framework Architecture

An AR authoring framework consists of set of tools specifically designed to assist non-expert individuals in effectively augmenting their reality with virtual content. In particular, the Augmented Catalog's System, consists of a cloud based application that works closely in hand with a secondary mobile application. As seen on the architecture diagram in figure 1, the cloud or server portion of the system provides a web based editor working as the front end to the final user. In this front end augmented catalogs can be created from scratch or by making use of existing templates. Additionally, a RESTful API exposes the serialized and compressed catalogs of user generated content to the mobile application though HTTP as detailed in figure 2.

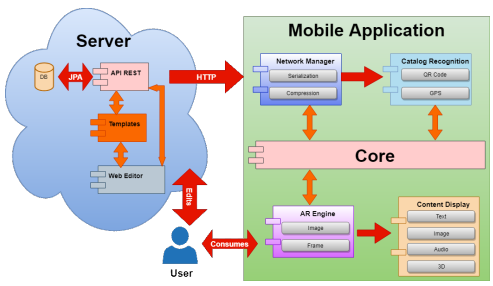

Figure 1: Architecture overview of the Augmented Catalogs System. The proposed AR engine can be integrated as the AR Engine module.

Once a catalog is acquired by the mobile application, it can be exploited completely offline by leveraging the capabilities of a content display module and a marker recognition module. This last one is the entry point where the proposed AR engine can be plug into the system as previously demonstrated in our Virtual Catalogs System using a commercial implementation.

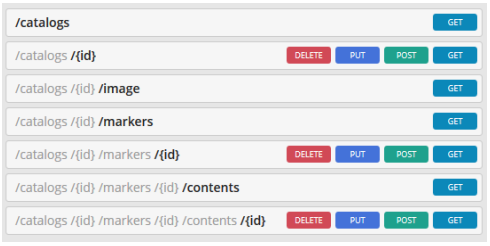

Figure 2: Structure of the Rest API connecting the server application in the cloud and the client.

This authoring framework architecture, as exposed by the Rest API, was previously integrated with a commercial AR engine and an inference engine in the context of a smart augmented medical card system [16] and with an augmented template framework [17].

2.2 Augmented Reality Engine

Augmented Reality of type 1, in terms of computer science, can be described as the problem of applying computer vision algorithms to detect planar objects, estimate a perspective transform relative to the camera, and tracking the object in question. While this have been all well-known and solved tasks for years, the constraint to perform them on real time has a significant impact that required thoughtful engineering and novel methods that are yet an active field of research. A proposed pipeline from which to build a sample engine is shown on Figure 3. Here we divide the problem in seven individual tasks grouped in two sub processes, starting from the detection of the planar object and followed by tracking. Detection of the trigger image or marker on a bigger picture or video stream is the first and also the most computationally expensive task of the pipeline. The detection sub-process shares elements to that presented by [18] and includes the following steps: finding of key-points, extraction of descriptors for those key-points, matching the current key-points with the pre-computed ones of the marker, finding inliers to the match and finally fitting a perspective transformation that satisfies the matching pairs. The first two steps in the proposed approach are to find and compute descriptors for key points of an image of the wanted object, the marker, and of the scene or video frame. To achieve this several algorithms are currently available and freely implemented, from classic blob and edge detection to more complex and robust methods like SURF [19], FAST [20], BRISK [21], STAR [22] and ORB [23]. While some of this meth-

ods include both the detection and description steps, other algorithms only provide a means to describe a point that must be found with a different one. The following step in the detection sub process is to match the descriptors obtained from the marker and the scene. This matches will allow us to find an homography and retrieve the transform matrix. For the task of matching, the time complexity of a naive approach is O(NxM) were N is the number of key-points of the marker and M the number of key-points in the scene. This means that the amount of key-points produced by each algorithm becomes an important factor regardless of the time it takes to actually find them. Once matches have been determined, it is necessary to filter out unwanted, low quality ones leaving us with the inliers of a potential affine projection. This process is performed by the RANSAC algorithm at the time of fitting a perspective matrix using the findHomography method provided by the opencv library. This matrix will allow us to project virtual content over the captured image or video stream with the correct perspective to make it look as if part of the real environment. In order to reduce the processing power required, the pipeline for an AR engine must switch to a tracking loop once the marker has been detected. From this point further, we can use the found key-points, inliers to the homography computation, to track the object and avoid re-computing them on each frame of a video stream. For this purpose, the already classic Lucas-Kanade algorithm based on optical flow can be used.

Figure 3: Pipeline for an AR application.

2.3 Integration

A type 1 AR engine built with the proposed pipeline and the described open source tools can work offline provided the markers have been already processed and its key-points associated with the virtual content to display as augmentations. It is precisely here where the integration with a cloud based authoring framework comes into play. Figure 1 shows the high level architecture of our Augmented Catalogs System including the integration point with the proposed engine. After a user uploads a trigger image or marker, we can take advantage of the high computational power of the cloud to quickly generate and process several variations of the marker. By applying known synthetic illumination changes and affine transformations we can simulate the conditions at which each marker may be found in the real world. By computing key-points on this simulated conditions, combining and ranking then we can increase the robustness of the detection step. Once computed, this key-points can be easily stored along with the augmentation contents and distributed as a whole catalog using the serialization and compression modules.

3 Testing Marker Recognition Algorithms for the Augmented Reality Engine

We proceeded to test and compare the capabilities of some of the available algorithms in the opencv library when under the constraint of real time computing.

3.1 Tests description

The tests consist of running the detection sub-process for each of several combinations of detection and description methods, measuring their time to completion on a fixed scene. The following tests were recorded running our code in a Core i7 4810MQ CPU at 2.8Ghz. In all cases, the various parameters of each algorithm were set to achieve the best speed while maintaining an equal level of accuracy as perceived by a human. Figure 4.a shows the marker, a png picture with a resolution of 800x560 chosen to be robust and with a good distribution of potential key-points. Figure 4.b illustrates the fixed test scene, a png file with a resolution of 1200x675 pixels made such that it represents the border line scenarios on which some of the current AR engine would still be able to work: a rotation of near 45 degrees (although we want complete rotation invariance), partial occlusion, poor illumination and a perspective of around 30 degrees from the top.

3.2 Finding of key-points and extraction of descriptors performance

We tested a combination of five key-point finder algorithms and four descriptors, SURF, FAST, ORB, BRISK and FREAK [24]. Some algorithms include their own descriptors while others do not. In the case of the FAST algorithm, the built in descriptor is covariant to rotations so it was immediately discarded. The ORB and BRIEF descriptors performed reasonably well but they were not robust enough to scale so FREAK or SURF were used on the final rounds of testing as shown in figure 5.

Table 1: Key point finding and descriptor extraction times in ms.

Algorithm	Finding Time	Extraction Time	Total Time
Fast+Freak	00.79	4.16	04.95
Brisk	10.99	0.00	10.99
Orb+Freak	10.18	2.77	12.95
Star+Freak	14.31	1.58	15.89
Surf	86.61	35.91	122.52

Figure 4: a) The marker images used for the tests. b) The simulated, fixed scene where the marker must be detected by the tests.

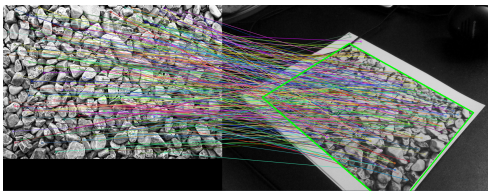

Figure 5: Example of the test's visual output using SURF features.

A usual camera ranges between 15 and 30 frames per second which means we have at most 50ms to run the whole pipeline without dropping frames. From the data in Table 1 we can immediately notice that SURF is not very appropriate for real time applications. Further parameter tweaking (a Hessian threshold of around 30000) allowed SURF to pass the test with significant lower times but cross validating with other marker confirmed that such fine tuning over-fits the original marker and does not work for others.

3.3 Matching descriptors and finding an homography

After finding and describing the key-points of the video frame, the next step it to match them with the key-points of the marker we computed in the cloud. While the minimum amount of points to compute an homography is as low as 4, we found that, given the accuracy of this key-points, an average of 20 good matches are needed. With this number of points the RANSAC algorithm can reduce the error of a rectangle fit to the marker borders to levels hardly noticeable by bare sight, as seen in figure 6. Note that finding the exact error in the retrieved transform is not of interest for this application since the results are only to be perceived by humans.

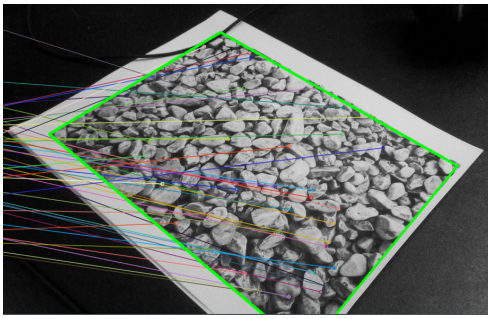

Figure 6: Overlaid rectangle with calculated perspective transform applied.

Table 2: Key point matching times in milliseconds.

Algorithm	Matching Time	Homography Time	Total Time
Orb+Freak	04.61	00.39	05.00
Star+Freak	10.71	03.59	14.30
Brisk	16.49	03.53	20.02
Surf	11.29	35.76	47.05
Fast+Freak	413.33	05.10	418.43

Table 3: Key points amount and quality.

	Surf	Brisk	Fast	Star	Orb
Scene Features	267	349	1524	503	500
Filtered matches	317	48	32	43	88
Inlier matches	255	40	29	39	64
Max dist.	0.54	202	183	167	168
Min dist.	0.09	51	24	29	30
% of inliers	95.5%	11.5%	1.9%	7.7%	12.8%

Given the complexity of a näive, brute force matching algorithm, the number of key-pints needed to achieve the wanted level of quality will have a tremendous impact in this step of the pipeline. As seen in table 5 and in line with results presented in [25] and [18], the ORB key point detection and FREAK descriptors produces the best results allowing for accurate planar object detection under the time limit of 20 fps,

or 50 milliseconds. ORB, Oriented FAST and Rotated BRIEF, is a rather new approach that builds up from the previous algorithms to amend for their weak points. This algorithm allows for a high level of parameterization but since we want to ensure the most general approach possible, the only values changed from its default was the maximum number of features (500) and the patch size (16). This number of features ensures a low maximum and average computing time as reported in Table 1 while yielding enough good quality matches. The smaller patch size benefits small features and reduces the margin left out from the marker in order to capture a more uniformly distributed set of key points. The descriptor chosen, Fast Retina Key point or FREAK for short, is a novel technique which tries to emulate the retinal response in our eyes to build a robust binary descriptor. It can be parameterized to be rotation and scale invariant and so it was set. Paradoxically, even if SURF produces only an average number of features, since most of them are of high quality, the homography calculation using RANSAC has to be run with a high number of candidates which raises its processing time significantly more than its gains to accuracy. Further filtering of the SURF features or RANSAC parameters tuning could be performed but the previous step already discarded this algorithm. Table 3 clearly shows why the matching step with FAST features is so slow with over 1500 key points just for the scene. We can also appreciate the relative quality of the features as a percentage of the inliers after running RANSAC. As stated before, SURF produces high quality key points with more than 95% of inliers and really low distances. The rest of the algorithms perform really close on average distance with ORB leaving in amount of inliers.

3.4 Detection Results

Table 4: Total times for each tested combination of algorithms.

Algorithm	Match & Homography	Find & Extract	Total Time
Orb+Freak	5	7.77	12.77
Star+Freak	14.3	15.88	30.18
Brisk	20.02	20.02	40.04
Surf	47.05	82.96	130.01
Fast+Freak	418.43	422.59	841.02

Table 6 shows the final times for this algorithms when running over a video stream of 720p without sight of the marker. It is evident that a great part of the processing effort in the detection loop is spent on matching the descriptors when using a naive, brute force nearest neighbor finder. OpenCV provides an implementation of more advanced matching algorithms

Table 5: Final times for ORB + FREAK + NAIVE NN over a video stream.

Times in ms	Max	Avg
Detect loop	44.00	14.68
Match only	19.34	06.03
% of total time	43.95%	41.07%

grouped in the Fast Library for Approximate Nearest Neighbors. The FLANN [26] matcher performs fast approximate nearest neighbor searches in high dimensional spaces by automatically choosing the best algorithm and optimum parameters depending on the dataset. While powerful, the lack of documentation for this implementation makes it hard to use for features other than numerical, like SURF. Running some experiments with this feature, it turns out that for a low amount of features (around 500 or less in this case) the naive brute force NN algorithm works at the same speed or even faster. This is because FLANN advanced model has to be retrained for every frame since we want to match the marker static features to the scene ones. If we did the opposite, the model could benefit from the fact that the markers features does not change and be trained only once. This requires some additional step to filter out detected features that are not part of the object but, since SURF features are very accurate, even without this filtering the homography calculation is good enough and the matching times reduces from the 11ms reported on Table 2 to roughly 2ms. This proves FLANN matcher is of utility for this particular problem and further work to use it with binary descriptors would yield significant performance improvements. Even if this algorithms work within the margins, we want to reduce the processing time even further to have enough time to do the rendering of the augmentation elements, that is, leave as much time as possible to the computer graphics side of this problem.

3.5 Tracking performance

In this area the Lucas-Kanade [27] method implementation in OpenCV was tested. The algorithm provides an internal measurement of the tracking error for each point and returns a mask for the ones lost. To further increase the accuracy, a threshold was set to discard points with an error over 15. Since the quality of the tracking degrades over time, a lower bound on the number of remaining key points was set. Some quick testing showed that having less than 15 points produces a noticeable error on the homography computation. When the number of key points is then lower than 15, the object is considered lost and we switch back to the detection steps.

This method of switching between loops for detection and tracking showed to work reasonable well even

Table 6: Final times when a Tracking sub-process is used.

	Times in ms
Max Frame	37.00
Avg. Frame	10.54
Avg. Track loop	10.47
Improvement	28.67%

under a highly un-optimized setup as our demo with a reduction of 28% in average frame time. Furthermore, having a separate and fast tracking once the marker is recognized, allows for the detection loop to be loosen up, trading off increased computation time to gain accuracy. Nevertheless, the switching does not take advantage of the last know position of the object when falling back. Instead, a mixed setup were a small number of new features is calculated on each step to regenerate and keep the tracking accuracy was implemented.

3.6 Simple Optimizations

3.6.1 Geometric constraints

Since the markers used for this demo, and the vast majority of type 1 AR applications use mainly squares or rectangles, some geometric constraints can be applied to ensure proper detection. Each time a homography is estimated, we can exploit the fact that, when correct, applying it to a rectangle of the same proportions as the marker should preserve the rectangular shape. This simple technique allows for quick false positive pruning without adding almost any computational cost by just checking whether the two inner diagonals of the figure intercept. The area can also be easily computed and further filter false positives whenever it gets unreasonably small.

3.6.2 Two-step detection

Over this work we have stated that the ORB key point detection algorithm along with FREAK descriptors is one of the most efficient and robust ways to perform planar object detection under heavy time constrains. The reason of this efficiency comes from the fact that this methods pro- vide enough medium quality features very quickly, but this means that, when the object is detected, only a few high quality points, the inliers, are left. This small number of points is not the ideal initialization for a tracking algorithm that would benefit from a bigger number of uniformly distributed features. Taking this into account, a two-step detection was implemented such that, after the marker is located, a new set of GFTT is calculated for that area and fed to the tracking loop. This showed to offer better robustness to occlusions and local shadows during the tracking process.

3.6.3 Feature regeneration

Rather than tracking a set of initial points until the object is completely lost and then reverting to the detection loop, a regenerative approach was implemented. This simple optimization checks whether the tracked points drop under a certain threshold, but while the object position is still well known, and finds a small number of new features only in- side the object's area. Since we know the new points belong to the object, no matching needs to be performed, and if an algorithm where the number of wanted features like GFTT or ORB is used, the overall time for this operation lies below the 2ms.

4 Conclusions

While most of the recent research effort and progress in the field of AR has been in pushing towards type 2 and type 3 variants, we have shown how a type 1 AR engine can be put together and integrated to a cloud authoring framework with considerable little effort by leveraging state of the art open source and freely available algorithms. We proposed and tested a complete AR pipeline by adjusting the pipeline in [16] and extending it, adding a tracking sub-process. We tested the detection sub-process with several combinations of key-point detection and description algorithms and demonstrated the ORB+FREAK pair to be the most promising for this particular, time constrained task. We also demonstrate it is very desirable to use tracking methods such as local optical flow in a separate subprocess once a trigger image is found. An integration point of the AR engine in the architecture of a cloud based authoring tool was explored. Grouping user generated contents by marker/trigger in the context of a catalog allows us to expose a very simple RESTfull API to communicate with clients. This integration allows the use of hardware resources available on the cloud, ensures bottom-up emerging content generation while taking advantage of online distribution and offline exploitation when needed.

5 Future Work

Even if the Augmented Catalogs System in which the sample AR engine is integrated allows for the creation of a manageable amount of markers or image triggers per catalog we are not exempt of the reduced scalability problem of the proposed engine or other commercial available ones. To alleviate this problem our effort will be focused on incrementing the scalability, the number of trigger images that can be detected without external help, of this model engine. Searching and identifying images in big volumes of data is a very active field or research with a flourishing cloud based ecosystem of related services such as reverse image search engines. In this context the most common

approaches are, developing predictable approximate search algorithms, or reducing the dimensionality of the input or query. The first type of methods can accept queries of a considerable size by predictably sacrificing the precision of the result. On the other hand, dimensionality reduction methods seek to discard the least relevant data from the input in order to allow for the use of more costly distance functions that yield more precise results. Our focus will be on integrating existing techniques, adapting them to work within the time constraints of an AR engine yet incrementing the scalability factor of the system in terms of the amount of trigger images or markers than can be detected offline. In particular, regardless of the difficulties adapting FLANN for the available types of inputs, its approach of building a hierarchical structure, a dendogram, to perform approximate NN results of great interest in this context. While building such structures have relatively high computational complexities of up to O(n2), the hardware capabilities available in the cloud make it possible to exploit them. Leveraging this indexing techniques we plan to integrate human face detection and recognition algorithms, incorporating them as another type of AR trigger into the Augmented Catalogs System. This research line seeks to be the starting point for future developments seeking to exploit the implicit information available on human faces such as biometric parameters and emotion elicitation.

Competing interests

The authors have declared that no competing interests exist.

References

[1] C. Manresa Yee, M. J. Abásolo, R. Mas Sansó, and M. Vénere, "Realidad virtual y realidad aumentada. Interfaces avanzadas", 2011.

[2] R. T. Azuma. "A survey of augmented reality", Presence: Teleoperators and Virtual Environments, 6(4):355–385, 1997.

[3] C. Boonstra, R. V. D. Klein, and M. Lens-Fitzgerald. "The Augmented Reality Hype Cycle". Available at: https://huguesrey.wordpress.com-/2009/09/08/the-augmented-reality-hype-cycle-sprxmobile-mobile-service-architects/. Accessed on 2019-9-1.

[4] Artoolkit. Available at: http://www.hitl.washington.edu/artoolkit/. Accessed on 2019-9-1.

[5] Vuforia. Available at: https://developer.vuforia.com/. Accessed on 2019-9-1.

[6] Layar. Available at: https://www.layar.com/. Accessed on 2019-9-1.

[7] Google ARCore. Available at: https://developers.google.com/ar/. Accessed on 2019-9-1.

[8] ARKit. Available at: https://developer.apple.com/arkit/. Accessed on 2019-9-1.

[9] Aurasma. Available at: https://www.aurasma.com. Accessed on 2019-9-1.

[10] Augment. Available at: https://www.augment.com/. Accessed on 2019-9-1.

[11] Aumentary. Available at: http://www.aumentaty.com/index.php. Accessed on 2019-9-1.

[12] Zappar. Available at: https://www.zappar.com/. Accessed on 2019-9-1.

[13] J. Ierache, N. Mangiarua, N. Verdicchio, M. Becerra, N. Duarte, S. Igarza, "Sistema de Catálogo para la Asistencia a la Creación, Publicación, Gestión y Explotación de Contenidos Multimedia y Aplicaciones de Realidad Aumentada". XVIII Argentine Congress of Computer Science, 2014.

[14] J. Ierache, N. Mangiarua, S. Bevacqua, N. Verdicchio, M. Becerra, D. Sanz, M. Sena, F. Ortiz, N. Duarte, S. Igarza, "Development of a Catalogs System for Augmented Reality Applications". World Academy of Science, Engineering and Technology, International Science Index 97, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 9(1), 1-7, 2015.

[15] OpenCV. Available at: www.opencv.org

[16] C. Montalvo, F. Petrolo, D. Sanz, N. Mangiarua, N. Verdicchio, S. Igarza, J. Ierache, "Knowledge Based Augmented Card System for Medical Assistance Over Mobile Devices". Selected Paper, XXI Argentine Congress of Computer Science, pages 257-265, La Plata, Argentina, 2017.

[17] N. Mangiarua, J. Ierache, M. Becerra, H. Maurice, S. Igarza, O. Spositto, "Templates Framework for the Augmented Catalog System". XXIV Argentine Congress of Computer Science, Tandil, Argentina, Pages 267-276, Revised Selected Papers, Springer Nature Switzerland, Springer, Computer Series Online, 2018.

[18] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK". International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, 2018, pp. 1-10, 2018.

[19] H. Bay, A. Ess, T. Tuytelaars, and L.V. Gool, "Speeded-up robust features (surf)". Comput. Vis. Image Underst., 110(3):346–359, 2008.

[20] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection". In Proceedings of the 9th European Conference on Computer Vision - Volume Part I, ECCV'06, pages 430–443, Berlin, Heidelberg, 2006.

[21] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints". In Proceedings of the 2011 International Conference on Computer Vision, ICCV '11, pages 2548– 2555, Washington, DC, USA, 2011.

[22] M. Agrawal, K. Konolige, and M. Rufus Blas, "Censure: Center surround extremas for real-time feature detection and matching". In David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, editors, ECCV (4), volume 5305 of Lecture Notes in Computer Science, pages 102–115. Springer, 2008.

[23] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf". In Proceedings of the 2011 International Conference on Computer Vision, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011.

[24] P. Vandergheynst, R. Ortiz, and A. Alahi, "Freak: Fast retina keypoint". 2013 IEEE Conference on Computer Vision and Pattern Recognition, 0:510–517, 2012.

[25] E. Karami, S. Prasad, M. S. Shehata, "Image matching using sift, surf, BRIEF and ORB: performance comparison for distorted images". CoRR abs/1710.02726, 2017.

[26] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration". In International Conference on Computer Vision Theory and Application, VISSAPP'09, pages 331–340. INSTICC Press, 2009.

[27] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision". In Proceedings of the 7th International Joint Conference on Artificial Intelligence, Volume 2, IJ- CAI'81, pages 674-679, San Francisco, CA, USA, 1981.

Citation: N. Mangiarua, J. Ierache and M.J. Abásolo. *Implementation of an Open Source Based Augmented Reality Engine for Cloud Authoring Frameworks*. Journal of Computer Science & Technology, vol. 19, no. 2, pp. 175–182, 2019.
DOI: 10.24215/16666038.19.e16
Received: April 13, 2019 **Accepted:** September 09, 2019.