Augmented Reality with Homographies

Alex Szilagyi, Avichel Verma
Illinois Institute of Technology

***Abstract*—In this paper, a computer vision concept is presented which provides real-time tracking and application of the concepts around Augmented Reality. This system utilizes ORB detection, homography, and image transformations which work to map each frame in a video capture to that of a predefined image file. Based on the transformations performed and features calculated, a 3D model is rendered in real-time. This paper will discuss the computations that were performed during this process and discuss a variety of use cases for this software.**

Keywords—Augmented Reality, Computer Vision, Homography, Oriented FAST and Rotated BRIEF, BFMatcher, SIFT, Projection

I. Introduction

This paper will explores the use case of Homography and pose transformations in Augmented Reality. Many scenarios exist in which could benefit from the introduction of virtual content. Emerging examples of this can be seen throughout a variety of fields such as Education. Interactive media is continually introduced into Educational settings as a means of producing a visual learning experience to students. As the technologies surround Computer Vision grow, the introduction of Augmented Reality becomes a viable solution.

Despite the increase in technologies that surround Augmented Reality, algorithms are still unsuited for real-time applications. Existing methods, which generally batch-process the entire data sequence at once, are often computationally excessive and time-consuming for real-time uses. This paper offers an explanation of concepts that attempt to resolve this problem. Through applying techniques of ORB feature detection, Homography, and projective transformations, we are able to provide an application that showcases some use cases of AR on a computationally limited systems.

The concepts and progress of applying each of these computations, as well as the experimentations that were performed will be detailed in this paper. Code examples will be provided that portray real-time tracking of features in notable images, as well as the rendering of 3D models upon recognition. As each of these concepts is explored, the underlying choices will be analyzed on why each method were chosen and how they were implemented. In particular, this paper will discuss Oriented FAST and Rotated BRIEF (ORB) and showcase some examples of it with real-time feature detection. In examining the use cases of ORB detection, a comparison between ORB and SIFT will be made. Comparisons will be made throughout the paper that regard a number of methods as well as a comparison of their efficiency and accuracy. In the initial stages of research, it was determined that utilizing Homography, feature detection, and projection transformations would be required.

The code that was developed will utilize each of these concepts in its AR process. The program was designed to recognize 3 different ID cards in real-time and display a corresponding object for each card. The program first identifies the features using ORB detection. It then uses BFMatcher to provide a number of matches between the recorded video frame and the reference image. Depending on which reference image received the highest number of matches, a corresponding object will be displayed. The object that is rendered will undergo transformations that utilizes Homography as well as other transformations to accurately display on the card in real-time.

II. Homography

A. Overview

The first of the concepts explored in this coding process was Homography. In simple terms, Homography represents a one-to-one mapping between two images. This concept is important in Augmented Reality, in which the concept resolves around mapping a predefined 2D (image) feature space to the same 3D (world) feature space. In our case, this represents the transformation that must be found between a 2D reference image and the 3D (world) captured image in a video frame. In order to maintain proper transformations within a video capture, the homography must be constantly calculated as transformations are performed on each frame within the video. By doing these transformations on each frame, we can achieve an approximation of an object's movements position and of its features.

B. Homography Computation

In order to fully understand the relationship that is created by homography, it is important to take a look at the computations that are being performed. In the example that will be presented later in the paper, homography is performed in an attempt to relate a 2D reference image to a video frame capture of the same 3D image. The attempt is made to recognize which transformations need to be made in order to render the 3D model that is used in AR. In this scenario, two sets of points exist. The first set of points are used to represent the 2D reference image. These set of points can be represented via homogeneous coordinates as (x, y, z). Alternatively, the

second set of points is that of the 3D image (captured via video frame) in which the set of points should be mapped to. These coordinates can be held in similar fashion via homogeneous coordinates as (u, v, w). The relationship between the two images is a homography if the following equation holds true:

$$\begin{pmatrix} u_i \\ v_i \\ w_i \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$

To solve for the elements of H, we must note that H can be broken down into a set of equations. By finding at least four general point correspondence, we can provide solutions to the unknown values in this matrix [8]. As shown above and explained previously, it can be seen that a homography matrix (represented with h's above) can be used to transform the initial points of the image (x, y, z) to the points that are reflected in the video capture (u, v, w). In order to find these corresponding points, we must perform some feature detection. These feature points were chosen using ORB (Oriented FAST and Rotated BRIEF) keypoint detection, which performs a combination of FAST keypoint detection and BRIEF descriptor detection.

C. Oriented FAST and Rotated BRIEF (ORB)

ORB (Oriented FAST and Rotated BRIEF) is a modified synthesis of FAST keypoint detector and BRIEF descriptor to increase efficiency and performance. Although FAST provides efficient keypoint detection, it does not produce measures of cornerness and gives relatively large responses along edges. In order to improve this, Harris corner measure is applied to FAST key points. This is done by selecting a large number of keypoints (N) through a low threshold value. With this large number of threshold values, we select K number of keypoints, where K < N. The keypoints are then ordered according to Harris measure and the top values are chosen.

It should also be noted that FAST cannot produce multi-scale features. This can be solved by implementing a scale pyramid of the image, and producing FAST features filtered by Harris measure at each level of the pyramid.

In order to overcome the problem of rotation invariance, FAST keypoints detection uses an *intensity weighted centroid*. This method assumes that a corner's intensity is offset from its center and hence this vector is used for computing the orientation. To improve the rotation invariance, moments are computed with x and y which should be in a circular region of radius r, where r is the size of the patch[1].

ORB additionally uses a modified version of BRIEF. This is due to the fact that BRIEF performs poorly with rotation. To resolve this, ORB runs a greedy search among the possible binary tests and filters out the values with both high variance and mean close to 0.5. The result is called as **rBRIEF**.

Once the above calculations are performed utilizing ORB, we are left with a number of feature points on the 2D reference image, and a number of feature points on the 3D world image. In order to determine if the 3D world image is in frame, the number of probable matches must be determined by comparing these two feature spaces. In order to do this, the application utilized Brute-Force Matcher (BFMatcher).

D. Brute-Force Matcher (BFMatcher)

As discussed above, in order to compute the homography matrix used to transform the 2D reference image to the 3D video capture, a number of feature points were determined using ORB. In order to obtain our goal of rendering an object onto this 3D video capture, it is important to recognize when this 3D video capture represents a good match to the reference image. Additionally, by defining different reference images to be matched with different objects, we are able to present a scalable application that can openly display a variety of objects.

Brute-Force matcher [9] begins by taking the descriptor of one feature in the first set and comparing it with all other features in the second set using a distance calculation. In determining which distance calculation to use, it is important to note which descriptor previously analyzed the images. The normalized L2 distance (or euclidean distance) provides a good option for SIFT, SURF, etc. This Normalized L2 distance is found by taking the square root of the sum of squares. For binary string descriptors like ORB, BRIEF, etc, it is better to use normalized Hamming distance. In Hamming distance, the distance is computed between two values by measuring the minimum number of substitutions required to change one value to another. This is done by minimizing the number of errors that are created during this transformation. The second attribute that can be modified in BFMatcher is called crossCheck. If this value is set to true, the matcher returns only matches with values that represent the two features in both sets that should match each other.

Once these parameters are decided, the BFMatcher can be passed the image descriptors and the number of matches are returned. This number of matches can be checked against some parameter in order to determine whether or not to render an object. It is important to recognize this threshold of matches, as a busier scene will tend to have much higher matches than that of a plain background. Now that a set of matches has been obtained and we have recognized the possibility for erroneous feature points within our detection, the homography matrix can be estimated.

In order to correctly find the matches that will calculate the homography correctly, we can utilize Random Sample Consensus (RANSAC). This RANSAC algorithm allows for an estimation to be made from a set of observed data that

contains outliers. As described above, in order to obtain better results and eliminate excess noise from our video capture, we must utilize this algorithm in finding correct matches.

E. Random Sample Consensus (RANSAC)

RANSAC (RANdom SAmple Consensus) is a resampling technique that uses the minimum number of data points required to estimate the model parameter. It uses as much data points to plot a fitted line to a model and prune outliers from the model.

It randomly selects a subset of data points and uses it to estimate the model parameters. It then defines the data points that are within the error tolerance of the generated model. These data points are considered as agreed with the generated model and are hence called consensus. The data points in the consensus are stated as inliners and the rest as outliers by RANSAC. If the count of the data points in the consensus is high enough, it trains the final model using them. The process is repeated for a number of iterations, to find the threshold iteration the equation stated below is used:

$$\widehat{T_{iter}} = \left[\frac{log\ \varepsilon}{log(1-q)} \right];$$

where ε = probability threshold ;(1-q) = subset with outliers.

This returns the model which has the smallest average error among the generated model.

As RANSAC is a randomized algorithm, it does not guarantee to find the optimal model with respect to inliers. But, the probability of reaching the best-fitted solution is possible with assigning suitable values to algorithm parameters.

III. Comparison of Algorithms

ORB: - As stated above, ORB is a fusion of FAST keypoints detector and BRIEF descriptor with improved modification. It uses FAST to determine keypoints and then Harris corner measure is applied to find top K points. Fast lacks orientation computation and rotation variant. Further, it computes Intensity weighted centroid of the patch with located at the center. This helps in finding determining the direction of the vector from corner points to centroid that gives its orientation. Additional moments are computed to improve rotation invariance. To add to this, the BRIEF descriptor performs poorly in case of an in-plane rotation. A rotation matrix is computed using orientation of patch and then the BRIEF descriptors are steered in accordance to orientation.

SIFT: - SIFT is the most renowned feature detection-description algorithm. The algorithm is computed in 4 basic steps. First, SIFT estimates the scale space extrema using Difference of Gaussian(DoG). Secondly, keypoints are localized and refined by eliminating low contrast points. Thirdly, a description method extracts a 16x16 neighborhood around each detected features and further segments the regions into sub-blocks, render a total of 128 bin values. Finally, descriptor generator is used to compute the local image descriptor for each key point based on image gradient magnitude and orientation. Although SIFT is proven to be very efficient in feature recognition but it requires a large computational complexity which is a major drawback for real-time applications.

The following equation shows convolution of difference of two Gaussians with Image *I (x, y)*.

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y),$$

where G is Gaussian Function.[4]

SURF: - SURF approximated DoG using box filters. Squares are used for approximation since the convolution with square is much faster if integral image is used. SURF detector is based on determinant of Hessian Matrix and it exploits integral images to improve feature-detection speed. A neighborhood around the key points is selected and divided into sub-regions. SURF uses *Haar wavelet responses* for feature-description within these sub-regions. The sign of Laplacian which is already computed in detection is used for underlying interest points. The sign of Laplacian distinguishes bright blobs on dark backgrounds and vice-versa. These types of contrasts which are based on sign allows faster matching.

Equation represents the Hessian Matrix in point "x = (x, y)" at scale "σ".[4]

$$H(x,\sigma) = \begin{pmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{pmatrix}$$

Where "Lxx (x, σ)" is the convolution of Gaussian second order derivative with the image "I" in point "x", and similarly for "Lxy (x, σ)" and "Lyy (x, σ)".

ORB is the fastest algorithm while SIFT performs the best in most scenarios(Fig 3.1). For Special cases, when angle of rotation is proportional to 90 degrees, ORB and SURF outperforms SIFT(Fig 3.2). In noisy images, ORB and SIFT shows almost similar performances(Fig 3.3). In ORB, Features are mostly concentrated on objects at the center of the image while in SURF, SIFT and FAST key points detectors are distributed over the image.[5]

	Time (sec)	Kpnts1	Kpnts2	Matches	Match rate (%)
SIFT	0.13	248	229	183	76.7
SURF	0.04	162	166	119	72.6
ORB	0.03	261	267	168	63.6

Fig 3.1 Results of comparing the images with varying intensity

Angle →	0	45	90	135	180	225	270
SIFT	100	65	93	67	92	65	93
SURF	99	51	99	52	96	51	95
ORB	100	46	97	46	100	46	97

Fig 3.2 Matching rate versus the rotation angle

	Time (sec)	Kpnts1	Kpnts2	Matches	Match rate (%)
SIFT	0.115	248	242	132	53.8
SURF	0.059	162	385	108	39.48
ORB	0.027	261	308	155	54.48

Fig 3.3 Results of image matching rate with noise

Order of feature-detector-descriptors to detect high quantity of features.

ORB>SURF>SIFT

Efficiency in order of feature matching.

SIFT>SURF>ORB

ORB being low computational requirement than SIFT. By exploring the depths of each of these concepts, it enabled us to ensure that the complexity of real-time Augmented Reality did not detract from the feasibility and performance of our application.

IV. Application Implementation

The application that was created attempts to utilize the concepts discussed in the previous section to demonstrate some use cases of Augmented Reality in a real-time video capture. Our implemented will utilize Python 3.6.5 as well as OpenCV. OpenCV will be utilized to perform key point detection (ORB) as well as image transformation and homography.

The application relies on a prerequisites in order to begin the computations. These prerequisites are:

1. Saved reference image (for feature detection and Homography)
2. A variety of .obj file for later rendering use
3. A defined minimum number of matches (for BFMatcher)
4. A set of camera parameters, used in the transformations that were performed

Each of the concepts discussed in the previous section will be revisited, with an explanation included on how the resulting code was affected.

The following section will relate the theoretical explanations described previously to the code that was written.

1. Feature Detection

A. Oriented FAST and Rotated BRIEF (ORB)

As described in the Homography section previously, the first step in our process was to perform ORB Keypoint detection. This provided us with a number of features from both the reference image and the .obj image, both of which are prerequisites to calculating the features. The .obj images were made online via tinkercad[3].

These keypoint values will be passed later to BFMatcher, in an attempt to recognize when the reference image matches values inside the frame of video.

ORB Example Code:

```
orb = cv2.ORB_create()

model, descriptors = orb.detectAndCompute(model, None)
k_frame, descr = orb.detectAndCompute(frame, None)
```

B. BF Matcher

As explained previously, BF Matches takes in values from the orb detector and provides a list of matches that was detected from the sets of feature points. As described in the previous section, since we are using ORB detection, it is important to utilize the normalized Hamming distance as opposed to the Euclidean Distance. This helps obtain better results and provides matches which more closely represent the features we wish to discover.

It is important to recognize that in order to determine which model most closely matches that of the frame, we compute the matches of each set of data. In our set of reference images ({imgA, imgB, … imgZ}), we map each image with an associated object from a similar set ({objA, objB, …, objZ}). Additionally, the ORB keypoint detector is performed on each of these and provides a set of keypoint and descriptor set for each

(image, object) pairings. In other words, when we analyze the frame of the video capture and find that imgC most closely matches, we want to render the associated objC. In order to maintain scalability throughout the project, it was necessary to create this mapping of (Image, Object) pairings.

In the application, the BFMatcher is performed iteratively using the current video frame ORB descriptors and the set of reference ORB descriptors obtained from each image that we wish to compare. Out of each of these computations, the one with the most matches is determined as the most closely related image. With this information, we can display the correct object rendering based on a variety of object/image pairings in our dataset.

BFMatcher Example code:

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING,
crossCheck=True)
matches = bf.match(model, descriptors)
```

C. *Homography and Transformations*

After the feature points are represented via ORB in their respective sets and matched to the model accordingly, we can begin to compute the Homography matrix and associated transformations required to render our AR Object. We only perform these computations if the total number of matches exceeds the threshold that we set initially. Once again, this threshold may vary depending on the amount of noise in the background of the image. Additionally, the reference images chosen should be those that have relatively unique attributes. Although any reference image can be chosen, it is possible that the wrong object will be rendered if BFMatcher cannot correctly decide the correct matched image.

First, we must differentiate between the points in our model and those in our frame. As described in the above section, these points are utilized in estimating the Homography matrix which will be used for transformations on the images. Once we have these set of points, we can call the following code to compute the homography matrix:

Initial

```
cv2.findHomography(source, destination,
cv2.RANSAC, 5.0)
```

We determined it important to utilize the RANSAC algorithm in order to eliminate excess outliers in our image. This will allow for a set of inliers, which represent the good matches, and a set of outliers, which represent excess noise that has been mistaken for good matches. Once we have obtained these points, we can use them to correctly create our projection matrix which is utilized in rendering the object.

D. Reading of .obj Files

An additional library was utilized that reads .obj files and makes them easy to use. This library was obtained from an Open Source project known as pygame[2]. The essence of this code implements the opening of .obj files and conversion into easily workable class objects.

2. Application Rendering

Once the homography matrix has been obtained, the final step is to create the projection matrix and render the 3D models onto the image. Lastly, the frame is modified and displayed in real time as it is captured. The following steps describe how the process occurred in the code.

A. Projection Matrix

First, the projection matrix was created based on the camera parameters and the homography matrix previously calculated. In order to obtain the correct calibration matrix, we must know some attributes about the camera. This calibration matrix can be displayed as the following matrix:

$$\begin{pmatrix} f_u & 0 & u_o \\ 0 & f_v & v_o \\ 0 & 0 & 1 \end{pmatrix}$$

In the above matrix, the focal length is represented by *f* in (u, v) coordinates while u and v are representative of the optical center of the camera.

Once we have obtained both the above matrix and the homography matrix, we can calculate the projection matrix. This projection matrix can be calculated by working backwards from out Homography matrix. We can recover the external matrix values through the following computations:

$$G = \begin{pmatrix} R_1 & R_2 & t \end{pmatrix} = A^{-1} \cdot H$$

Fig 4.1 - Estimation of Projection Matrix
Where G represents parts of the external calibration matrix amongst two different reference frames.

In obtaining the projection matrix, it is important to note that the values obtained from G are based off of an estimated Homography matrix. In *Fig 4.2*, the values of G_1 and G_2 are the similarly represented values obtained from *Fig 4.1* (R_1 and R_2 respectively). Since we know that the external calibration matrix [R1' R2' R3' t] represents the final homogeneous transformation matrix that is required, we simply need to obtain accurate representations of each value based on their estimated comparisons.

$$l = \sqrt{||G_1|| \cdot ||G_2||}$$

$$C = \frac{G_1}{l} + \frac{G_2}{l}$$

$$p = \frac{G_1}{l} \times \frac{G_2}{l}$$

$$d = C \times p$$

$$R_1' = \frac{1}{\sqrt{2}}\left(\frac{C}{||C||} + \frac{d}{||d||}\right)$$

$$R_2' = \frac{1}{\sqrt{2}}\left(\frac{C}{||C||} - \frac{d}{||d||}\right)$$

$$R_3' = R_1' \times R_2'$$

$$R = [R_1' \; R_2' \; R_3' \; t]$$

$$Proj = A \cdot R$$

Fig 4.2 - Projection Matrix Calculations [6]

First, we can compute the rotation and translation along the x and y axis by taking the dot product between the estimated Homography matrix and the inverse of the camera parameters (*Fig 4.*1). Once these are obtained, we normalize each of the vectors and compute the orthonormal basis of each. Lastly, the 3D projection matrix is applied to the current frame utilizing the camera parameters initially defined. This projection matrix is passed on to the rendering of the model onto the frame and represents transformations that must be applied to later frame and object attributes. Each of these steps can be seen in (*Fig 4.2)*, in which *G* represents estimated values that correlate to real values *R.* These come together to allow for a Projection matrix that can be used in the rendering of the object onto frame.

B. Rendering of Models

The final stage of rendering our model is to project the 3D model using the calculations we have obtained thus far. Since we already have our model and our projection matrix, all that's left is to project our object onto the image frame of the live video. This can be done by taking each point in the model and scale it according to the projection matrix. This will take care of the scale and transformations that need to be done and return an image in which is projected onto the frame. This is done since we do not initially know the size of the object. If this was controlled in the creation of the objects, this would not have to be done manually.

C. Scaling and Handling of Models

Since each .obj file that is downloaded has a different size of model, we provided a parameter that allows manually scaling of the object. This is determined when analyzing each object and has to be passed in as part of the pairing. Lastly, we can choose to add textures to the model if it comes with one. For efficiency, we chose to simply cover the image in a standard color in order to allow for faster performance. In order to ensure that the program is able to modify frames live, it is important to choose objects that have relatively little complexity. With larger, more complex models, the program requires much more computational power.

V. RESULTS

As mentioned in the feature selection section above *(III & IV),* a variety of methods were tested in an attempt to obtain the highest accuracy possible. Each of these provided unique benefits and drawbacks that were weighted carefully in the underlying results.

While examining the number of matches on each of the cards under different scenarios, we noticed common themes with each of the results obtained. These results were based on the feedback obtained from each frame of the video and data was recorded regarding the number of matches that each card obtained, with respect to their reference image. In a dark environment or white background with no movement, the number of features that are matched with each of the cards was 0. As additional noise and movement was added to the frame, this number of matches rose. Upon examining the threshold for each of the matches, it was determined that a good match was found if the threshold reached ~200 Matches. This was a common occurrence no matter the card or background in view. In backgrounds with much noise or those without clear view of the card, the program was often unable to determine which model had the highest matches. In order to limit the error in our process, it was decided that all cards would be tested on the same simple background with little variance in the environment.

The cards that we chose as reference images can be found in the figures below. Each of these cards were chosen due to their unique features, which allowed the ORB descriptor to clearly distinguish the card features while still displaying similar features between them. In each of the cards, there

exists some form of clear pattern. Additionally, each of the three cards has some characters of text that is clearly displayed. Images of the three cards matching features and rendering objects are displayed below. Fig 5.1 matches a gift card and displays the first 5 digits of Pi. Fig 5.2 matches a ventra card and displays an image of a CO Molecule. Lastly, Fig 5.3 matches a metra pass to that of the letters "Co2".

Fig 5.1 - Pi Reference Card

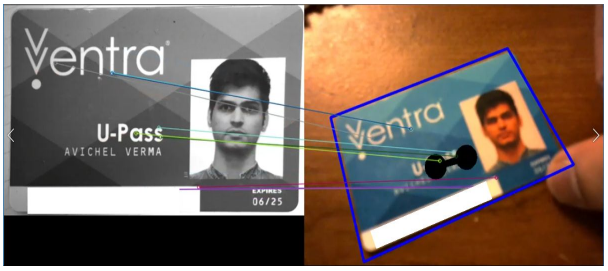

Fig 5.2 - Molecule Reference Card

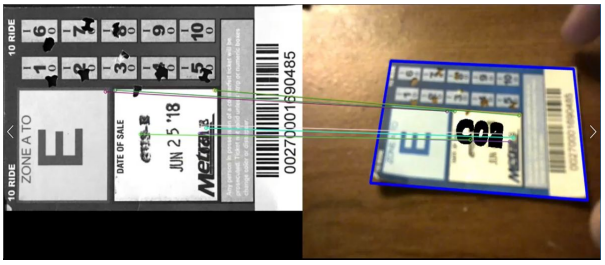

Fig 5.3 - C02 Reference Card

Each of these cards were uploaded to the computer via scanner in a clearly displayed manner. These scanned images were used in the initial phase of ORB's feature detection (image-left), and were compared to the video captured frames that can be seen in the right side of the images above. The lines attaching the image were drawn using openCV and simply display the highest correlating matches that BFMatcher found. Analyzing these allow for a recognition of the highest weighted features and lets us perform some error analysis on each of the cards.

The models that were chosen to match each of the cards represent some form of real-life use case for the image. In an education setting, a set of cards with unique features could be printed out that represent match equations or chemical compositions. A card can be matched to display an image of a chemical composition in either text format (*Fig 5.3)* or molecular structure *(Fig 5.2)*. If an image of a mathematical formula or symbol is displayed (such as that of Pi), a corresponding number can be displayed in 3D (*Fig 5.1)*.

These cards showed to provide the best results in recognizing matches for their respective objects. As mentioned, each of these obtained over 200 matches when shown in video format to their reference image counterpart. The other cards obtained values that corresponded to ~50 to ~200 matches depending on their clarity and orientation to the same reference image. In some cases, the BFMatcher determined that the number of features from Card A had a higher correlation to Object B than Card B did. During the analysis, it became clear which features in the cards caused this error. Although the results were not perfect, the program nearly always matched the correct card with the correct model.

VI. Error Analysis

After a thorough evaluation of the results obtained by performing various experiments on the model, it was observed that BFMatcher finds higher correlation between various cards depending on the similar feature points and patterns. In certain scenarios, BFMatcher had some difficulty matching the features from the reference image onto the capture frame. In these scenarios, some small error was observed which results in the application switching back and forth between various models.

Firstly, It was also observed that font similarity played a vast role in between the Pi and Molecule reference cards (Fig 5.1 and 5.2 respectively). Upon examining the features chosen when selecting the best match, it was found that in the cases where text was weighted highly that the error in choice rose. One example of this is that the letter 'n' exists on both cards in a similar font. This resulted in BFMatcher weighing the cards equally, as the ORB descriptors matched the reference image with the same number of matches.

Secondly, similar patterns and features on various cards were integrally important in determining the percentage of error. In the Pi and Molecule reference cards (Fig 5.1 and 5.2 respectively), the patterns that exist have areas of large gradients and sharp edges. In the errors observed, these edges were often found as notable features that matched the reference image.

It was examined that the number of errors that existed between Fig 5.3 and the other cards was significantly lower than the other tests. This can be mostly attributed to the unique number of features that do not exist on the other cards. In particular, this card has a variety of text that is in a unique font and size than the other cards. Additional features such as the number of separate boxes also set it apart from the others. In an attempt to eliminate as much error as possible, these three

cards were chosen from a set of 8 as they provided a unique analysis on feature similarity while still providing accurate results.

VII. Future Works

As mentioned in previous section *Comparison of Algorithms (III),* one of the many changes that could be made is experimentation with different feature detection libraries. Although ORB was chosen for good reason, it would be interesting to do a comparison of efficiencies with the other models. Additionally, due to the need to calculate and render 3D models on to frames in real time, we were limited with the complexity that could be displayed. In order to improve the opportunity for real-life scenarios, it is important to allow for complex enough models to exist. For this to happen, the efficiency of our program must improve. Since this was not the nature of our experimentation, the efficiency of ORB and other algorithms chosen was sufficient enough.

In order to allow for the exploration of other feature detection algorithms such as SIFT, future works can explore GPU optimization. Such an optimization will allow for heavy computations to be performed in real time and allow for exploration of computationally heavy algorithms. In the exploration of these algorithms, it can be expected based on results from outside works that the accuracy in detection of feature points will improve. Additionally, GPU Optimization will allow for more complex models to be introduced that can open the software to additional use cases.

This paper represents an example of the real life applications of Augmented Reality. The focus is not to provide working examples of real life scenarios. In future works, these concepts could be applied in warehouses, educational classrooms, and online learning sites. In order to accompany for the needs of these settings, the code can be ported onto a mobile device to allow for ease of use. The efficiency of the software will have to be carefully analyze in order to ensure that this transition is possible. In addition to the efficiency, any environment that adds extra noise to the image requires additional complexity. Since this experimentation focuses on a definitive set of features, it would most likely have to be heavily modified for scenarios with large noise.

Lastly, this paper serves to provide examples for a small set of use cases and is designed for an educational setting. In the event that this software is determined to be used commercially, the code will have to be restructured in order to encompass a more scalable environment. As it is, many of the models and objects are limited in scalability and serve as to provide examples of what is possible with these concepts.

VIII. Lessons Learned

In order to have a successful application with a robust and scalable nature, it is important to recognize the variety of opportunities that exist in the field of Computer Vision. In this paper, we discussed the opportunity to explore a multitude of algorithms such as ORB and SIFT. Each of these provide a variety of values in unique scenarios.

While experimenting with a variety of frame captures, it was noticed that feature detection varied widely with the amount of noise in the frame. In order to ensure that the results were accurate, a shift had to be made in the way the recording was captured in order to ensure a relatively noise-free environment that allowed for definitive feature capturing. Similarly to exploring the amount of noise in the background of an image, the uniqueness of the features of reference image was also explored. We found that those that had the most unique features provided for the best results and those that contained similar characters and features replicated each other's objects more often.

IX. Conclusion

Initial research provided unique information on the use cases of Homography and pose transformations in Augmented Reality. By examining the nature of these methods, we were able to design the goals and functionality behind this application. The application developed focuses on eliminating the need to batch-process data while applying AR technologies. In doing so, the process that was created looks at introducing ORB feature detection, Homography, and projection transformations to apply Augmented Reality in real-time. This paper examines the optional approaches that currently exist for AR on computationally limited systems.

The overall nature of the project was extensive in depth, as we attempted to make our solution unique by comparing the benefits and drawbacks of each of the algorithms used. Throughout this project, an understanding was made that the solution must be able to exist in real-world scenarios. This helped with our approach in allowing us to analyze which stage of the solution could be better optimized, while not impeding on the accuracy of the program. Some obstacles still remain that exist error into the process, such as a limited number of matching features in the cards. With further optimization, it is possible that the type of solution implemented can have even greater accuracy in the detection of features and rendering of objects. We hope that there are continued approaches that provide similar solutions in exploring the field of Augmented Reality.

X. Acknowledgement

The researchers would like to recognize and thank the overseeing professor, Gady Agam, for his beneficial insights and support throughout the course of this project. Without his assistance, this research could not have reached its fullest potential.

XI. References

[1] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html

[2] http://www.pygame.org/wiki/OBJFileLoader

[3] https://www.tinkercad.com/

[4] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8346440

[5] https://arxiv.org/ftp/arxiv/papers/1710/1710.02726.pdf

[6] https://link.springer.com/content/pdf/10.1007%2Fs11263-008-0152-6.pdf

[7] http://www.dtic.mil/dtic/tr/fulltext/u2/a460585.pdf

[8] https://pdfs.semanticscholar.org/d901/eb960d723aa878a6bdaa2d3944aedb12ad86.pdf

[9] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html