
Project - 1

# Perception for Autonomous Robots

March 8, 2022

*Instructors:*
Samer Charifa

*Semester:*
Spring 2022

*Course code:*
ENPM 673

*Student:*
Joseph Pranadeer Reddy Katakam
UID: 117517958

# Contents

# List of Figures

# 1 Introduction and Project Constraints

- In this Project, a custom AR Tag is detected whose point of reference is used for Augmentation Application. To do so, the AR Tag is `Detected` from input sequence followed by `Tracking` the Tag to obtain pose using which Super Imposition of Image and a Virtual Cube is done.
- Provided Resource: Video (onto which the Augmentation has to be done), Super Imposing Image (.png), kmatrix.
- `Allowed Functions`:
  1. numpy: svd, inv, matmul, etc
  2. scipy: fft and inverse fft
  3. some cv2 functions such as Harris corner or Shi-Tomasi,goodfeaturestotrack, sobel, canny, blob detection.
- `Disallowed Functions`:
  1. cv2 findHomography
  2. cv2 warpPerspective
  3. cv2 Hough Transforms
  4. cv2 find Countours
  5. any inbuilt function which calculates the homography or warps the image for you.
  6. any other inbuilt function that solves the question in less than 5 lines.

# 2 Problem 1: AR-Tag Detection

## 2.1 Fast Fourier Transformation

- **Fourier Transformation**: In the simplest terms, a Fourier transform helps in breaking down a incoming signal into its building blocks. Example: using a High Pass Filter will yield edges. The Fast Fourier Transform is a convenient mathematical algorithm for computing the Discrete Fourier Transform. It is used for converting a signal from Spatial Domain to Frequency Domain (in Image Processing). Therefore, the FFT represents the image in both real and imaginary components. By analyzing these values, we can perform image processing routines such as blurring, edge detection, thresholding, texture analysis, and yes, even blur detection.
- **Pipeline**:
  1. Obtain the Input Image and convert it to Gray Scale Image followed by computing FFT.
  2. To filter out noise and make the detection smoother, Gaussian Blur is used. Therefore, multiply the above image with the Gaussian mask to filter out high-frequency signals and perform Inverse Transform.
  3. Threshold the blurred image to get the Binary Image.
  4. Compute FFT once again and multiply it with the Circular Mask to eliminate low frequency signals. To get back the image, performing Inverse FFT.

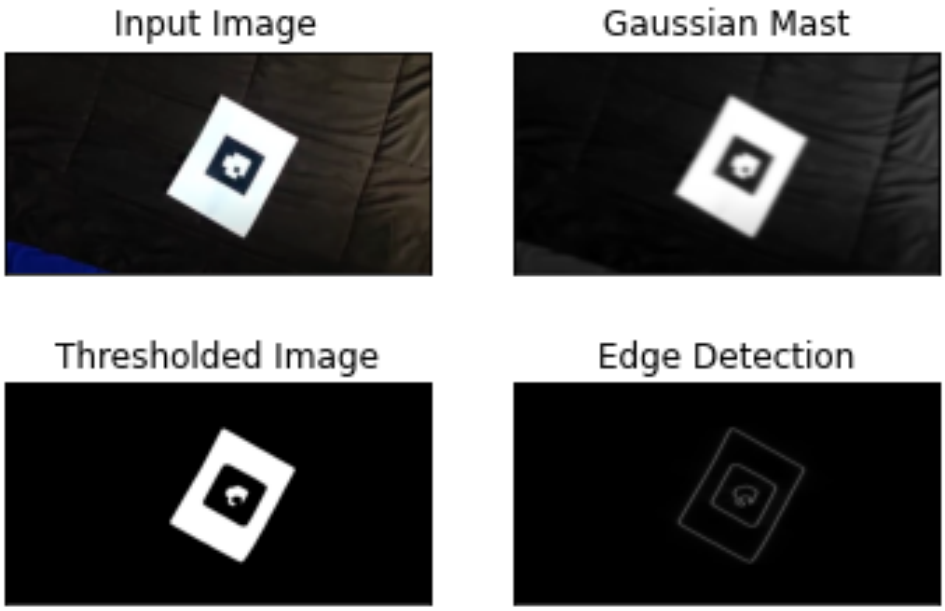


Figure 1: Fast Fourier Transformation

## 2.2 Corner Detection

- After trying a lot of allowed Corner Detection functions, I have decided to use `Harris Corner Detection Method` as the detected corners are robust to lighting and noise comparatively.
- **Pipeline**:
  1. Using a kernel, compute Erosion and Dilation for the obtained image to eliminate any present Noise.
  2. Obtain Corners using Harris Corner Detection Method. This step is followed by thresholding to eliminate any unwanted corners.
  3. Refinement is done to the obtained corners.
  4. Now, segregate the corners of the sheet and the AR Tag.
  5. Draw the lines between the corners to show successful detection of edges.

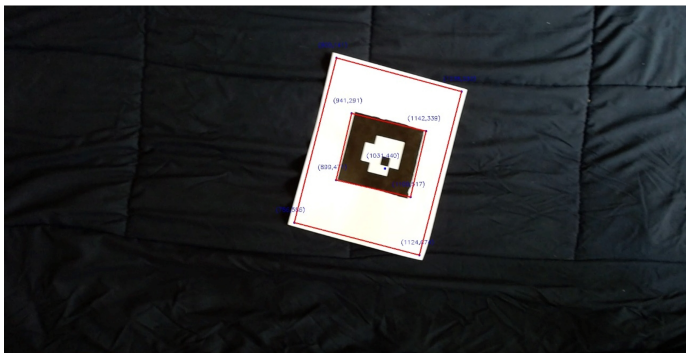

Figure 2: Plotting the Detected Corners on taken Image

## 2.3 Decode custom AR Tag

- **AR Tag**: There are many types of Fiducial Markers whose purpose is to act as a point of reference or a measure. Here the popular fiducial marker `AR Tag` has been used. Using this, the AR Tag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera.
- Decoding the Tag is done in a step process. First the Tag has to be processed and then, information has to be extracted from the Tag.
- **Pipeline to Process the Tag**:
  1. The tag can be decomposed into an 8 x 8 grid of squares, which includes a adding of 2 squares width (outer black cells in the image) along the borders. This allows easy detection of the tag when placed on any contrasting background.
  2. Take the median for each block and replace the 20 $\times$ 20 block with a single value depending on the median. It will be either 255 or 0 since the image is binary.
  3. The inner 4 x 4 grid (i.e. after removing the padding) has the orientation depicted by a white square in the lower-right corner. This represents the upright position of the tag. Extract it.
  4. Rotate the Matrix until the up-right pose is reached which is determined by the bottom right grid and track the number of rotations made to achieve this.
  5. Lastly, the inner-most 2 x 2 grid (i.e. after removing the padding and the orientation grids) encodes the binary representation of the tag's ID, which is ordered in the clockwise direction from least significant bit to most significant. So, the top-left square is the least significant bit, and the bottom-left square is the most significant bit. Extract this matrix to get Tag Information.
- **Pipeline to get Tag Info**:
  1. As discussed, from the 2 x 2 inner most grid, the data is obtained in a clockwise manner.
  2. A tag ID is created for each cell depending on whether the cell is blank or filled.
  3. These values are used to super impose an Image onto the Tag.

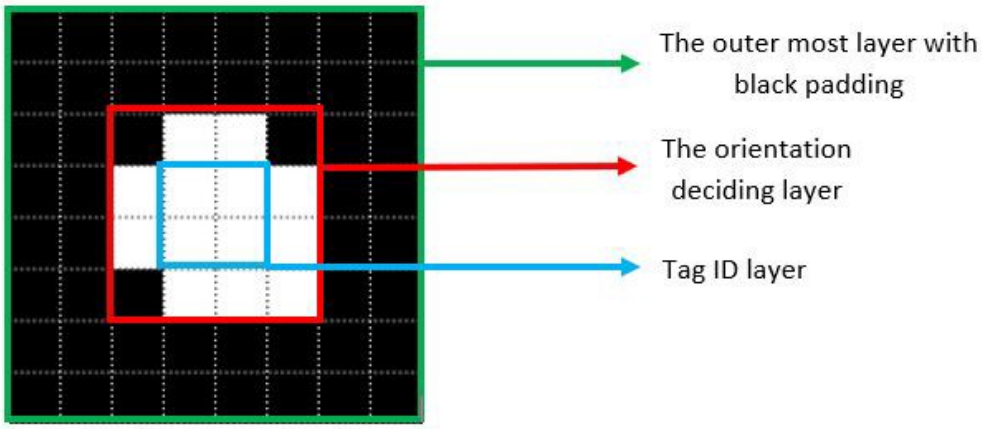


Figure 3: Grid Pattern overlayed on AR Tag

# 3 Problem 2: Tracking

## 3.1 Homography and Warping

- Once the corners of the Tag are obtained, `Homography Matrix` can be computed between the Tag corners and the desired tag corners.
- Next the image is warped using `Inverse Warping Technique` which is used to obtain the AR-tag for decoding.
- Therefore, the end result of this stage is the Pose and Information of the Tag.

Figure 4: Detected Output of AR TAG

## 3.2 Superimposing the Image

- From the above obtained data, the given Testudo image is superimposed on the AR Tag.
- First, get the corners of Testudo image and the Tag.
- Now that we have two sets of ordered points, we can compute a homography matrix and warp the Testudo image on the video frame.
- Problems that occurred when superimposing the image is that, there were many missing points. So, in order to not loose any information, I have decided to use Bi-linear Interpolation to interpolate and get the missing data.

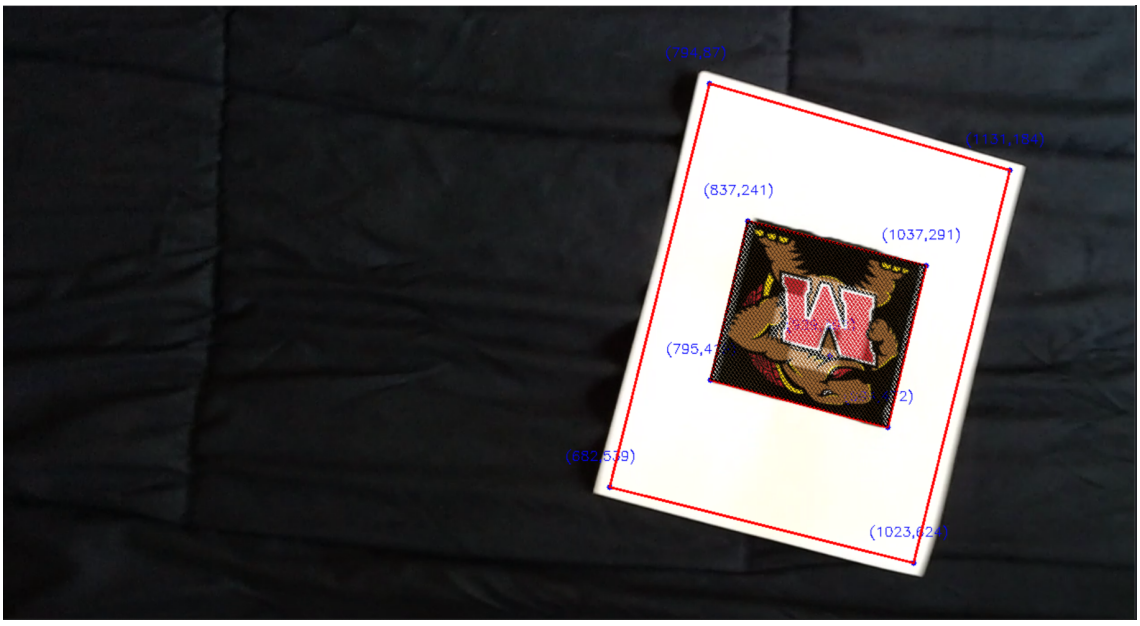


Figure 5: Computed Superimposed Image

## 3.3 Virtual Cube on Tag

- Augmented reality applications generally place 3D objects onto the real world, which maintain the three-dimensional properties such as rotation and scaling as you move around the "object" with your camera. In this part of the project you will implement a simple version of the same, by placing a 3D cube on the tag. This is the process of "projecting" a 3D shape onto a 2D image.
- Since a cube is a three dimensional entity, we need a $3 \times 4$ projection matrix to project 3D points into the image plane. The basic assumption while calculating the homography matrix is that the points are planar, i.e. $z = 0$ for all the points.
- Now, we have z values as well. Thus we need to modify this H matrix to suit our requirements.
- Projection Matrix is obtained using the Homography matrix and the Intrinsic matrix parameters.
- Goal is to obtain the projection matrix, using which we can multiply the homogeneous 3D co-ordinates of the cube and obtain its equivalent image co-ordinates.

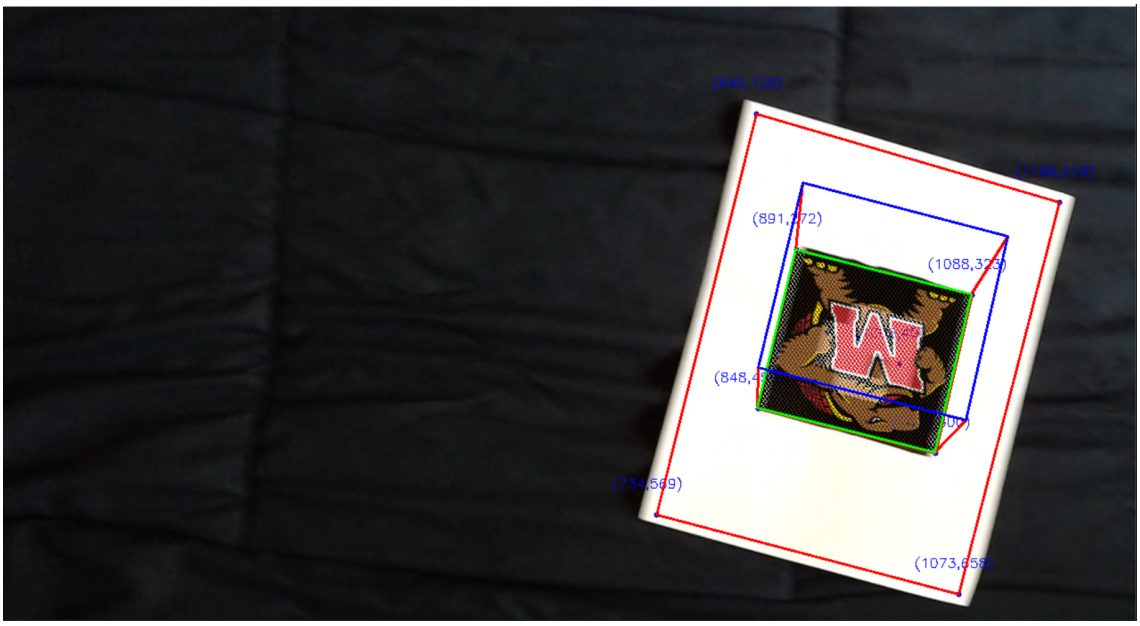


Figure 6: Computed Virtual Cube onto Tag

# 4 Resources

- Homography Estimation
- Image Warping
- Given Intrinsic Matrix
- Modifying Intrinsic Matrix

# 5 Output Links: Video and Images

- Drive Link (Press here)