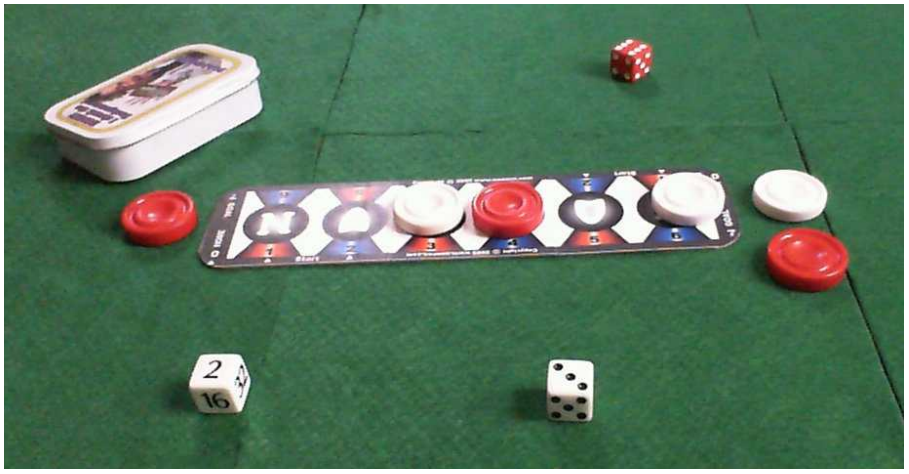
CS/RBE 549 - Final Report
Honey, I Shrunk the Backgammon

Michael Brauckmann
Daniel Miller
Chris Winsor

12/22/2014



Introduction

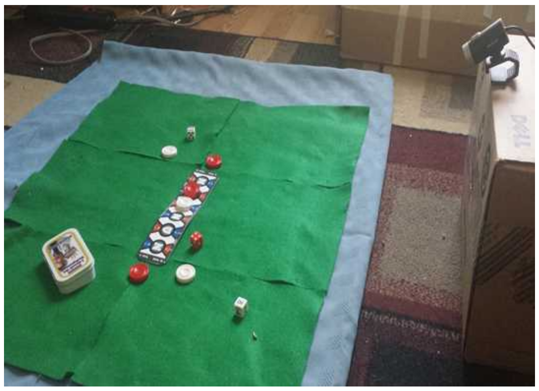
As many processes in our modern world become more automated, the need for robust sensing systems has increased. Passive sensors, such as color cameras, are smaller, less expensive, and more rugged than their active sensor (LIDAR) counterparts. As such, this report explores the use of digital color camera images for detection, manipulation, and classification of common objects. Specifically we explore the detection, segmentation, and classification of dice, as well as techniques for rectification of playing surfaces.

Methodology

This document focuses on the two tasks targeted by our computer vision team. The first, Die Segmentation and Classification, strives to isolate standard 6 sided dice from an image, and then classify them with a value 1-6, corresponding to the value shown on their top face. The second task, is rectification of the captured image, transforming the ground plane such that it is normal to the imaging plane. This makes the future task of locating the game board in space much simpler.

Die Segmentation

The first task of die segmentation involves locating the dice within an image, and isolating the pixels containing each die. These now isolated die images can be easily passed through any of a variety of machine learning algorithms for classification.



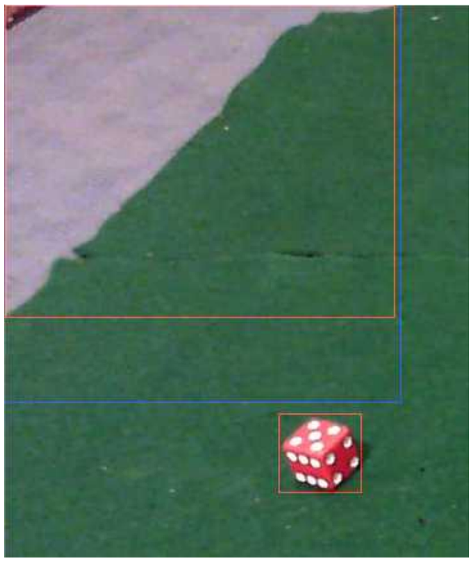
Camera Setup and Initial Image Processing

As shown in the image to the left, the camera has been positioned above the playing surface, looking down at an approximately 35 degree angle. This angle presents numerous difficulties for computer vision, as the objects' scales change as they move about the table. In an effort to inhibit further complexities, the scene lighting was fixed with a single source behind the camera. Captured images were first transformed to greyscale, with each channel's weight determined experimentally. The result of this greyscaling, and it's relation to an original image may be seen in the first two images of Appendix A.

Thresholding and Binary Operations

The now greyscaled image is thresholded using a standard binary threshold operation. The pixel intensity threshold was determined experimentally, and will vary with lighting conditions. In order to remove extremely small regions, and to connect small gaps in otherwise continuous regions, binary image operations were applied. Specifically, the binary closing operation was performed. The result of this closing can be seen in the fourth image in Appendix A.

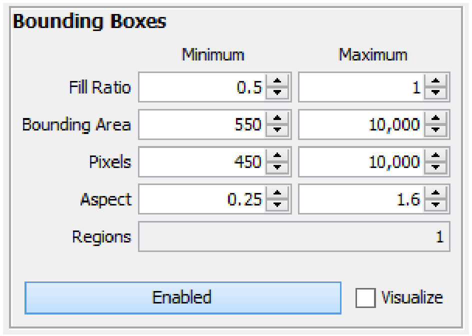
In order to slightly grow the region, while also closing and connecting neighboring regions, circular kernels were used, with differing sizes for both the erosion and dilation. Specifically, a circular kernel of size 9 was used for the dilation process, whereas a circular kernel of size 7 was used for erosion. This results in the slightly enlarged, yet connected, binary region shown in Appendix A, image four.



Region Labeling and Bounding Box Filtering

With the binarized image processed, it was time for region labeling. Unfortunately, OpenCV does not include a two-pass region labeling algorithm in its libraries. The team experimented with writing a custom two-pass labeling algorithm. However, the overhead of calling OpenCV from Java so many times over an image proved to be too much overhead. In an effort to increase processing speed, this two-pass algorithm was abandoned, in favor of a simple flood-fill algorithm. Although theoretically slower in processing time, this flood-fill algorithm leveraged OpenCV's highly optimized native C code, resulting in much faster run times.

With the regions properly labeled, certain erroneous regions still needed eliminating. As shown at the right, the non-green upper corners of the image were frequently above the threshold, passing them through the algorithm as possible dice candidates. In an effort to remove these false positives, a bounding box filter was introduced, which allowed or removed regions based on their properties. A rendering of the filtering component is shown here, for reference.



Bounding Boxes

	Minimum	Maximum
Fill Ratio	0.5	1
Bounding Area	550	10,000
Pixels	450	10,000
Aspect	0.25	1.6
Regions	1	

Four different region properties were measured, and compared to a user-set allowable range. "Fill Ratio" limits the ratio of bounding box area to region pixel count. As shown in the image above, regions which did not fill at least half of their bounding box were removed from consideration. The "Bounding Area" and "Pixels" filters bounded the allowable bounding box area and region pixel counts, respectively. Finally, the bounding box's aspect ratio was calculated, and limited to a range of 0.25 to 1.6. These filters proved to be quite robust at removing erroneously detected regions, reliably allowing only regions which contained dice.

Image Padding and Resampling

The final steps of die segmentation involved padding the bounding boxes of the detected regions, and then scaling the images to a fixed size. First, the bounding boxes were made square, and then given some padding, specified by the user. A typical value of 5 pixels was used. The images were then resized using OpenCV's `Imgproc.resize()` function, and then saved to disk.

The above process proved quite robust, as it was able to correctly detect, segment, and save more than 1800 die roll sample images, with no more than a handful of errors. Even so, these errors were usually in the form of false positives, which were easily removed with small tweaks to the above mentioned filter settings.

Die Classification

Once a die has been located in an image the next step is to identify the value of the top surface. The multilayer perceptron model commonly known as an Artificial Neural Net (ANN) was used for this purpose. building this classifier requires a sizable training set, and an implementation of an ANN.

Capturing Training Image Set

The Training set was created using the same camera setup and image segmentation process described above. however, instead of attempting to classify the segmented die image, the die was placed on a known value and the correct classification given for training. The training set consisted of 120 images of each of the 6 possible die values, with the die placement evenly distributed around the visual field of the camera. This allows all the noise and variation in the visual field to be accounted for in training data. Each training sample was resized to a square image with a given number of pixels.

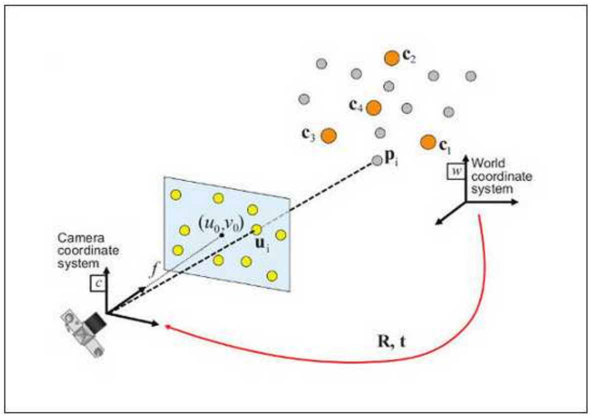
Neural Net Topology and Training

The ANN implementation used came from the OpenCV multilayer perceptron library. This library allows for a variety of network topologies. These option were made available in the graphical interface created for the project, which allowed experimentation to determine the most accurate Topology. The most accurate networks produced consisted of two internal layers of 100 nodes, and accurately classified the dice in around 68% of testing samples. This result show considerable improvement over a guess (16%) but could likely be improved with a larger training set or more robust back propagation training. The output layer consists of 6 nodes one for each possible die value and can be normalized to produce a probability of each value. currently the classification is simply the maximum value from the set but further steps could be taken to improve accuracy when two values have high probability.

Image Rectification

We are given the gameboard placed arbitrarily in the field of play and viewed from a perspective. The challenge is to unwarp the perspective to facilitate easier detection of the board.

Intrinsic / Extrinsic Camera Parameters



To unwarp the board we could have used a perspective transform and be done with it like we did in CS545 Image Processing. Instead we decided to go the formal route of camera calibration and 3D transform matrix. As a result we end up with a system that preserves units - from millimeters in the real world to pixels in the image. And because it separates intrinsic and extrinsic matrices, movement of the viewer or target don't affect the camera parameters and vice versa. Anything that needs to interact with the world, like a robot, is going to need this transform so the technique here is reusable.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

To calibrate the camera we went with OpenCV's Calib3c.findChessboardCorners() and Calib3c.calibrateCamera(). These are time-tested, meaning reliable, and a clean solution, meaning it focuses on just the millimeter-to-pixel translation and rotation of the camera. It explicitly avoids what I consider to be the eye-candy attempt to resolve z-dimension components at this stage which does nothing but unnecessarily introduce error.

To get the extrinsic matrix we used Calib3c's solvePnP(). The method takes a paired set of points in 3D physical space and 2D image plane and matches them up to produce the rotation matrix. There are several matching algorithms to choose from - here we went with the most simple - 4 points and the Gao/Hou/Tang/Chang algorithm.

solvePnP doesn't specify **HOW** you arrive at the 3D and 2D point pairings - some techniques include stereo vision, shape from shading, distance from haze, or the popular laser rangefinder. This is a huge topic and one we surveyed briefly in class - certainly plenty of opportunity for future study here.

solvePnP has output in the form of a rotation and translation vectors - the former can be converted to a rotation matrix using Calib3c.Rodrigues(). solvePnP also gives up a distortion coefficient matrix - a helpful indicator of quality on the point pairs.

Image Warping

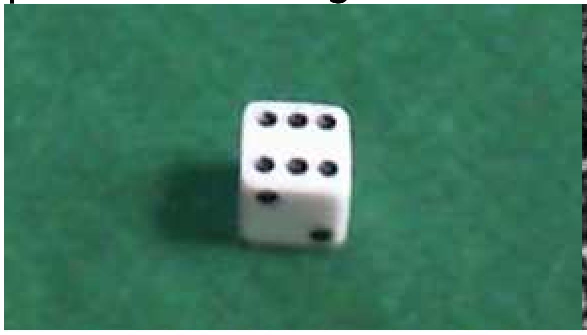
With the intrinsic camera and extrinsic rotation matrix in hand we can now convert an arbitrary 3D point to an image point on the 2D image matrix. An example is creating a warping (image-to-image) transform to "flatten" the image. The process here is to pick four arbitrary points in 3D space, convert these to 2D image, then use Calib3d.findHomography() to create the perspective transform matrix. Applying that to our image using Imgproc.warpPerspective() gives us the 2D image with perspective removed.

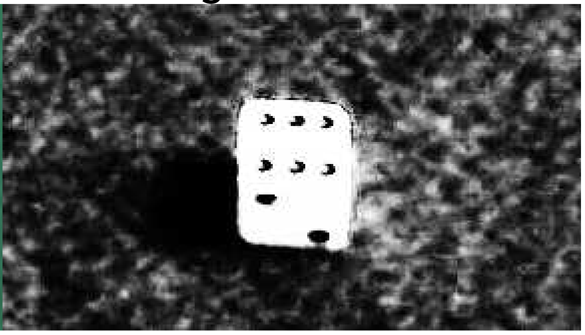
Summary

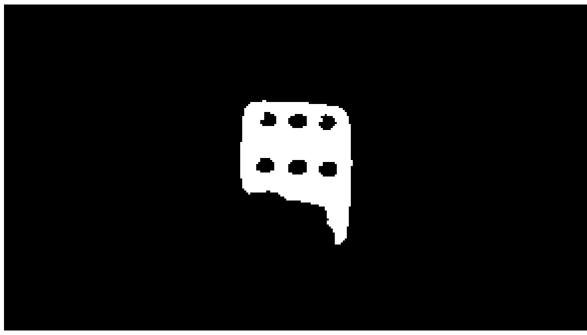
With the above we have a "flattened" image, but more importantly established:

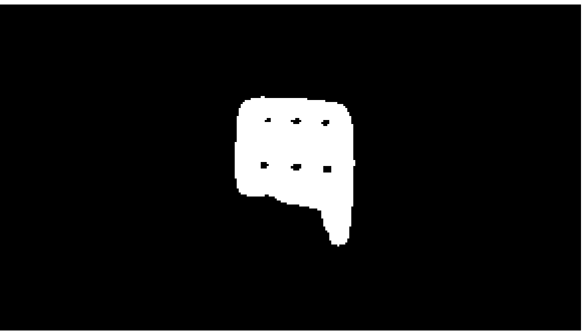
- the ability to convert an arbitrary 3D point to 2D space
- isolated the intrinsic camera components from those of extrinsic pose
- preserved unit conversion so we are able to work in 3D space (millimeters), 2D projected image space (millimeters), and 2D camera image (pixels)

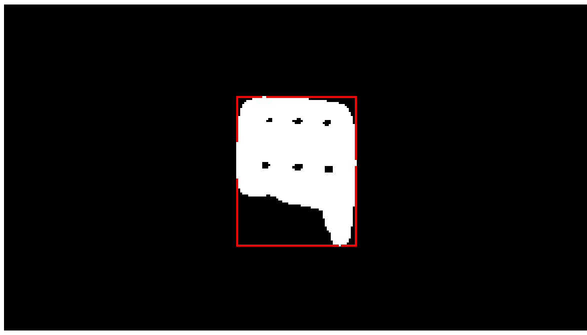
Appendix A - Die Segmentation and Labeling

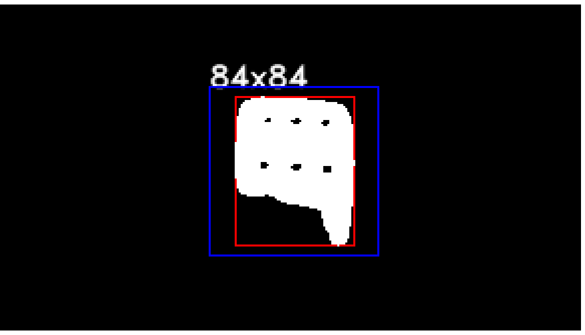

Original Image


Channel Weighting

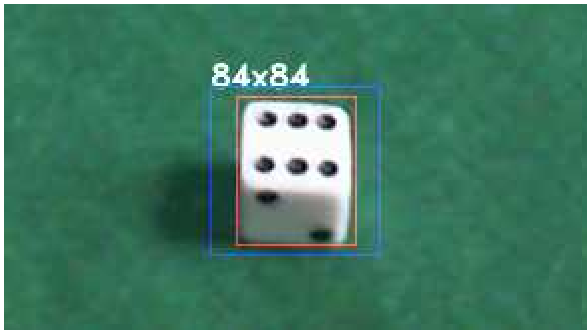

Thresholdinf


Binary Operations

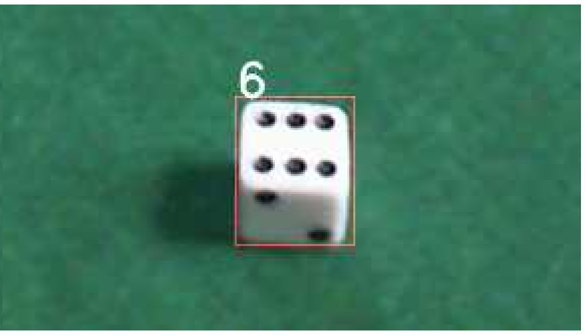

Bounding Box Filtering



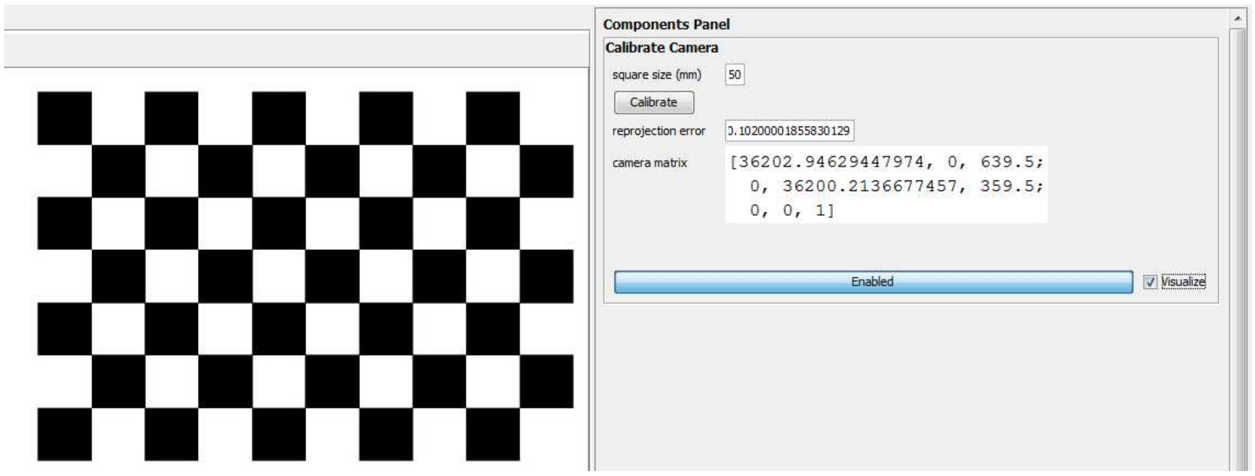
Padding and Squaring



Segmented Region



Classified Region



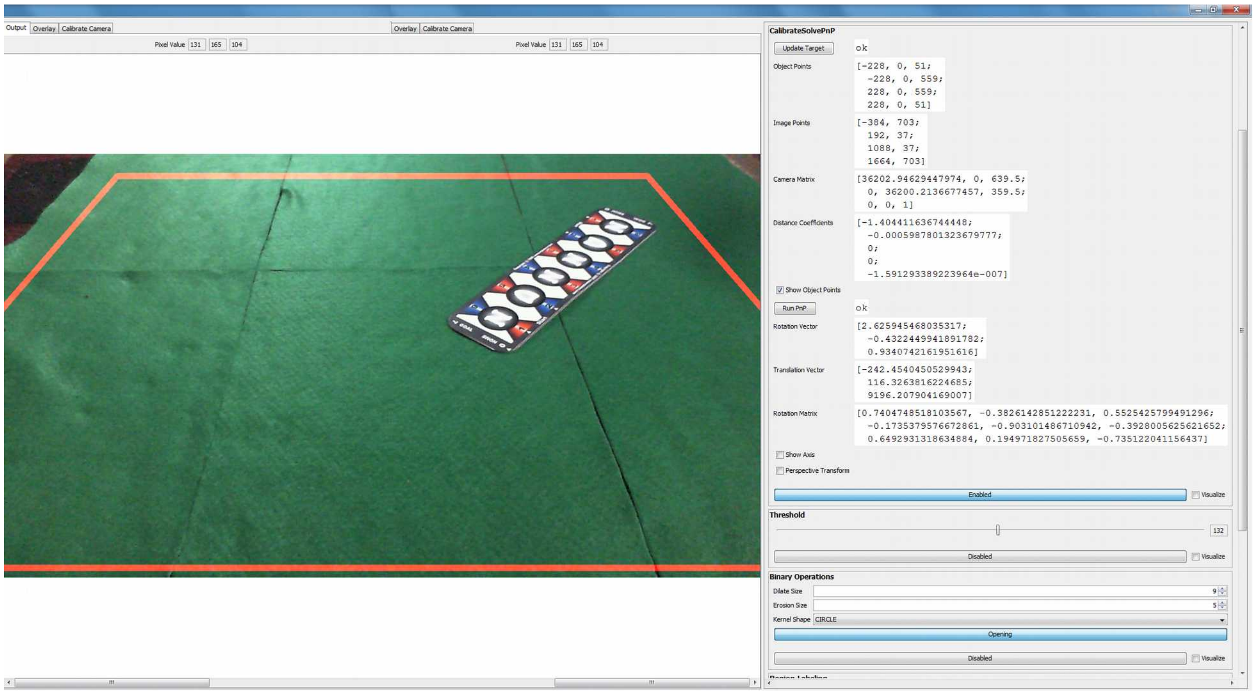
square size (mm) 50

reprojection error 0.10200001855830129

camera matrix
```
[36202.94629447974, 0, 639.5;
  0, 36200.2136677457, 359.5;
  0, 0, 1]
```

Appendix B - Camera Intrinsics and Pose

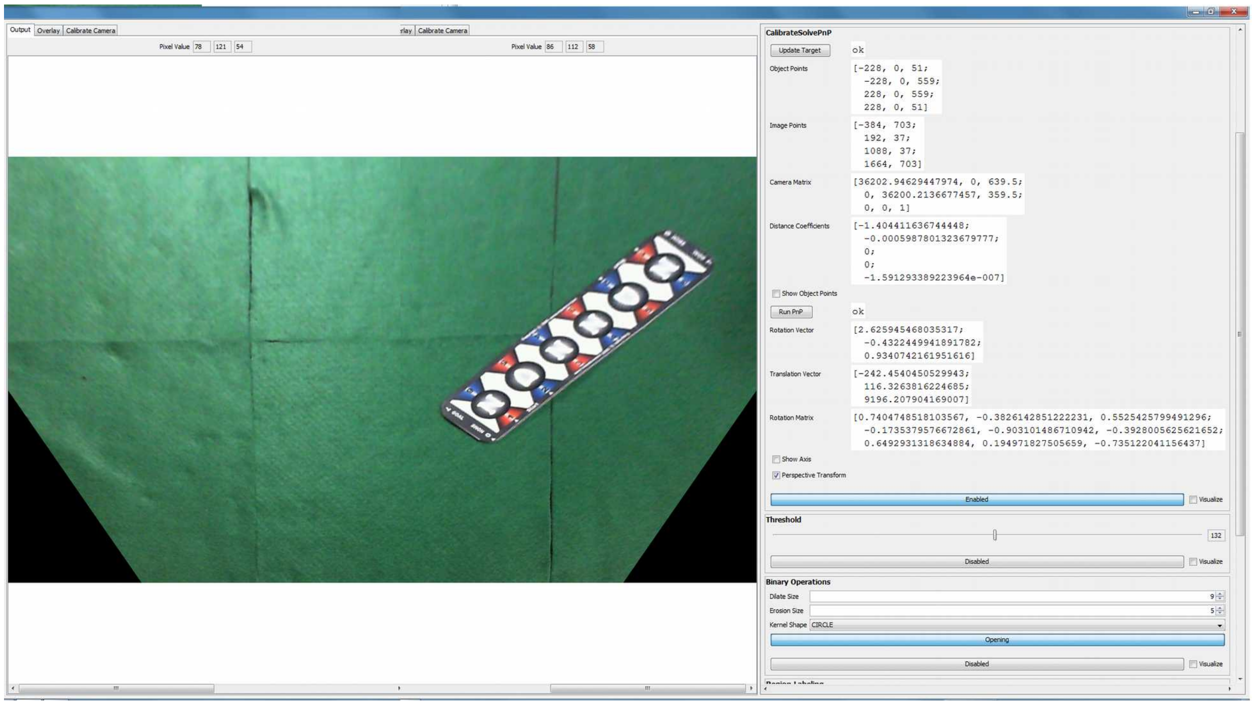
Camera Intrinsics - translation, rotation and scale from 2D projected image coordinate system in millimeters, to 2D camera image coordinate system in pixels.



```
Update Target          ok
Object Points          [-228, 0, 51;
                         -228, 0, 559;
                         228, 0, 559;
                         228, 0, 51]
Image Points           [-384, 703;
                         192, 37;
                         1088, 37;
                         1664, 703]
Camera Matrix          [36202.94629447974, 0, 639.5;
                         0, 36200.2136677457, 359.5;
                         0, 0, 1]
Distance Coefficients  [-1.404411636744448;
                         -0.0005987801323679777;
                         0;
                         0;
                         -1.591293389223964e-007]
[✓] Show Object Points
Run PnP                ok
Rotation Vector        [2.625945468035317;
                         -0.4322449941891782;
                         0.9340742161951616]
Translation Vector     [-242.4540450529943;
                         116.3263816224685;
                         9196.207904169007]
Rotation Matrix        [0.7404748518103567, -0.3826142851222231, 0.5525425799491296;
                         -0.1735379576672861, -0.903101486710942, -0.3928005625621652;
                         0.6492931318634884, 0.194971827505659, -0.735122041156437]
[ ] Show Axis
[ ] Perspective Transform
Threshold              132
Dilate Size            9
Erosion Size           5
Kernel Shape           CIRCLE
```

Extrinsics Matrix - transform from 3D physical world coordinate system in millimeters to 2D project image in millimeters.

Flattened image as a result of combining 3D-2D point conversion, Calib3d.findHomography(), and Imgproc.warpPerspective().



CalibrateSolvePnP

```
Update Target      ok
Object Points      [-228, 0, 51;
                     -228, 0, 559;
                     228, 0, 559;
                     228, 0, 51]
Image Points       [-384, 703;
                     192, 37;
                     1088, 37;
                     1664, 703]
Camera Matrix      [36202.94629447974, 0, 639.5;
                     0, 36200.2136677457, 359.5;
                     0, 0, 1]
Distance Coefficients [-1.404411636744448;
                     -0.0005987801323679777;
                     0;
                     0;
                     -1.591293389223964e-007]
[ ] Show Object Points
Run PnP            ok
Rotation Vector    [2.625945468035317;
                     -0.4322449941891782;
                     0.9340742161951616]
Translation Vector [-242.4540450529943;
                     116.3263816224685;
                     9196.207904169007]
Rotation Matrix    [0.7404748518103567, -0.3826142851222231, 0.5525425799491296;
                     -0.1735379576672861, -0.903101486710942, -0.3928005625621652;
                     0.6492931318634884, 0.194971827505659, -0.735122041156437]
[ ] Show Axis
[✓] Perspective Transform
Enabled            [ ] Visualize
```

Threshold

```
132
Disabled           [ ] Visualize
```

Binary Operations

```
Dilate Size    9
Erosion Size   5
Kernel Shape   CIRCLE
Opening
Disabled           [ ] Visualize
```