Basic Image Processing

Lecture

Fall 2022

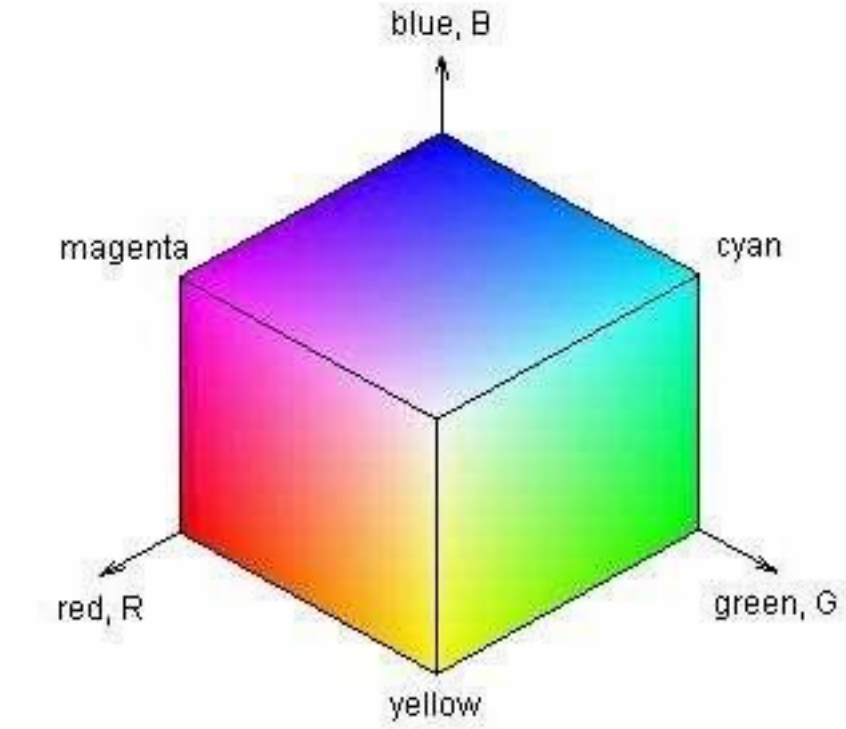
M | ROBOTICS

Sort by Size and Color!

S2T4W22

RGB color space

- Single wavelength primaries
- Good for devices (e.g., OLEDs for monitor)
- Simple to work with

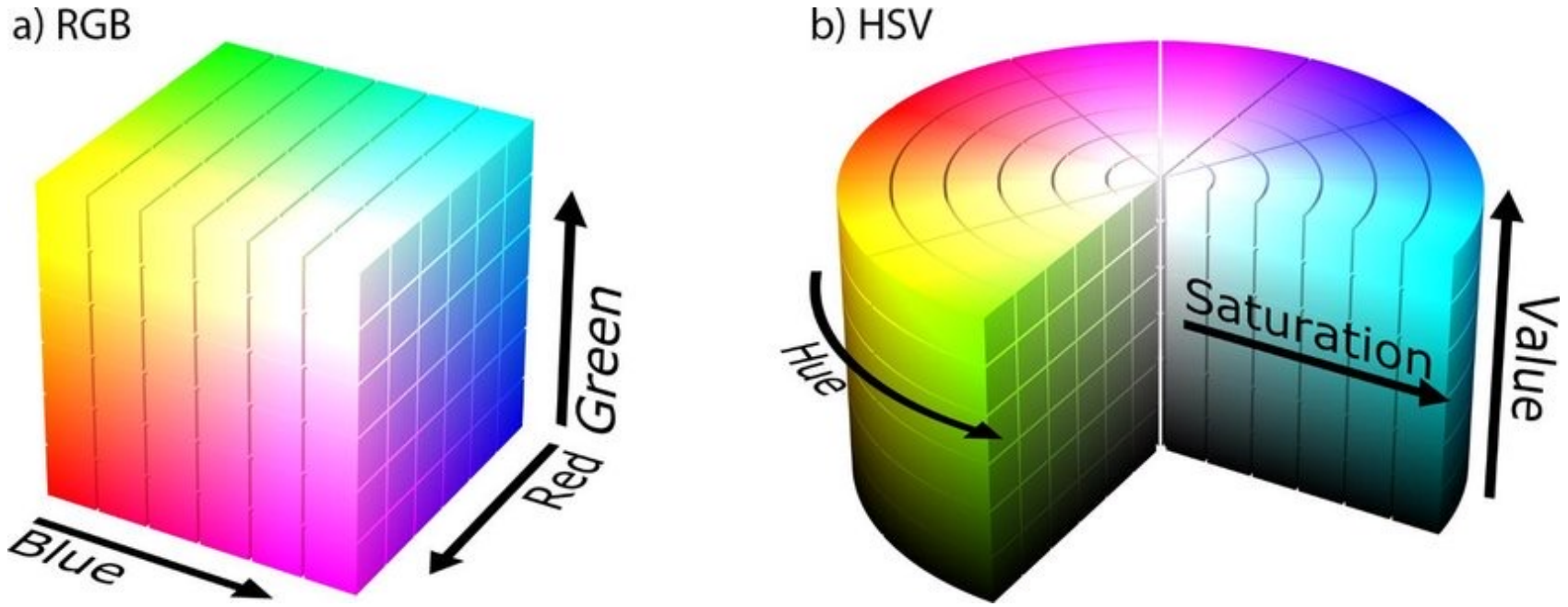


Defining RGB Colors for Computing

- specify intensity of red, green, and blue components
- typical range of intensity 0 - 255
- 256 * 256 * 256 = 16,777,216 potential colors, 24 bit color
- Higher resolution in many applications (12 to 16 bits or more per channel) “HDR” -- High Dynamic Range
- Additive color
  - (0, 0, 0) = Black
  - (255, 255, 255) = White
  - (255, 0 , 255) = ? (255, 140, 0) = ?

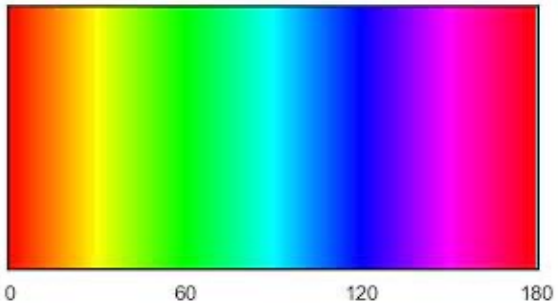
HSV color space

- Hue, Saturation, Value
- Nonlinear – reflects topology of colors by coding **hue** as an angle



HSV

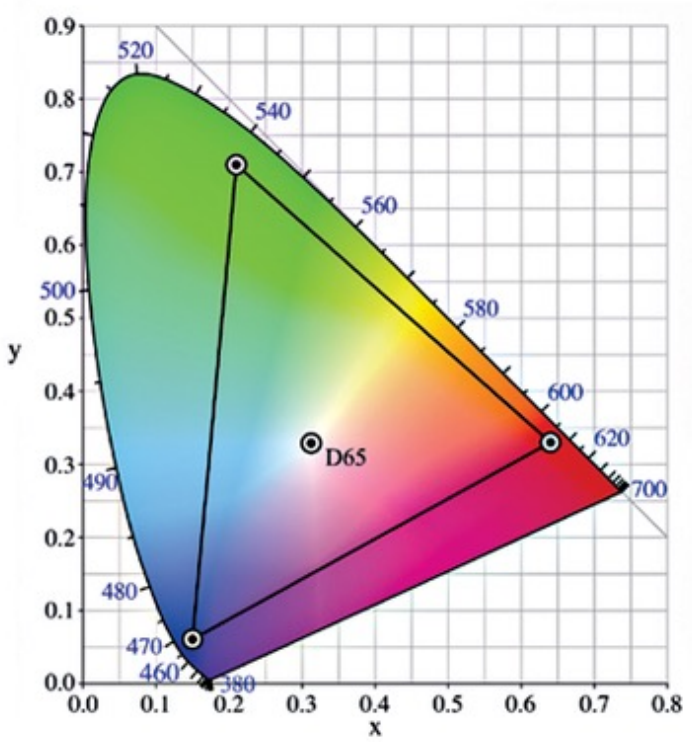
- **Hue** channel is an **angle** [0 -179]
- **Saturation** and **Value** are 8 bit [0-255]
- Need to take into account wrap around 0-180°



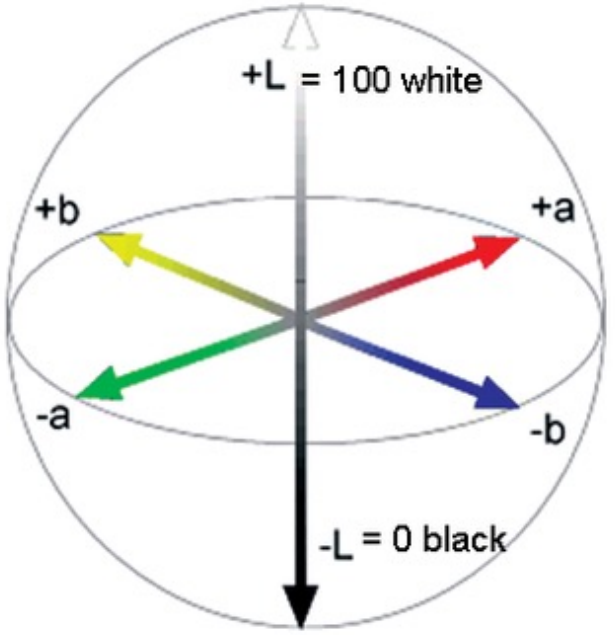
CIE L*a*b color

- Allows wider range of color than RGB model
- L – lightness a – red/green axis, b yellow/blue axis

RGB Color Model

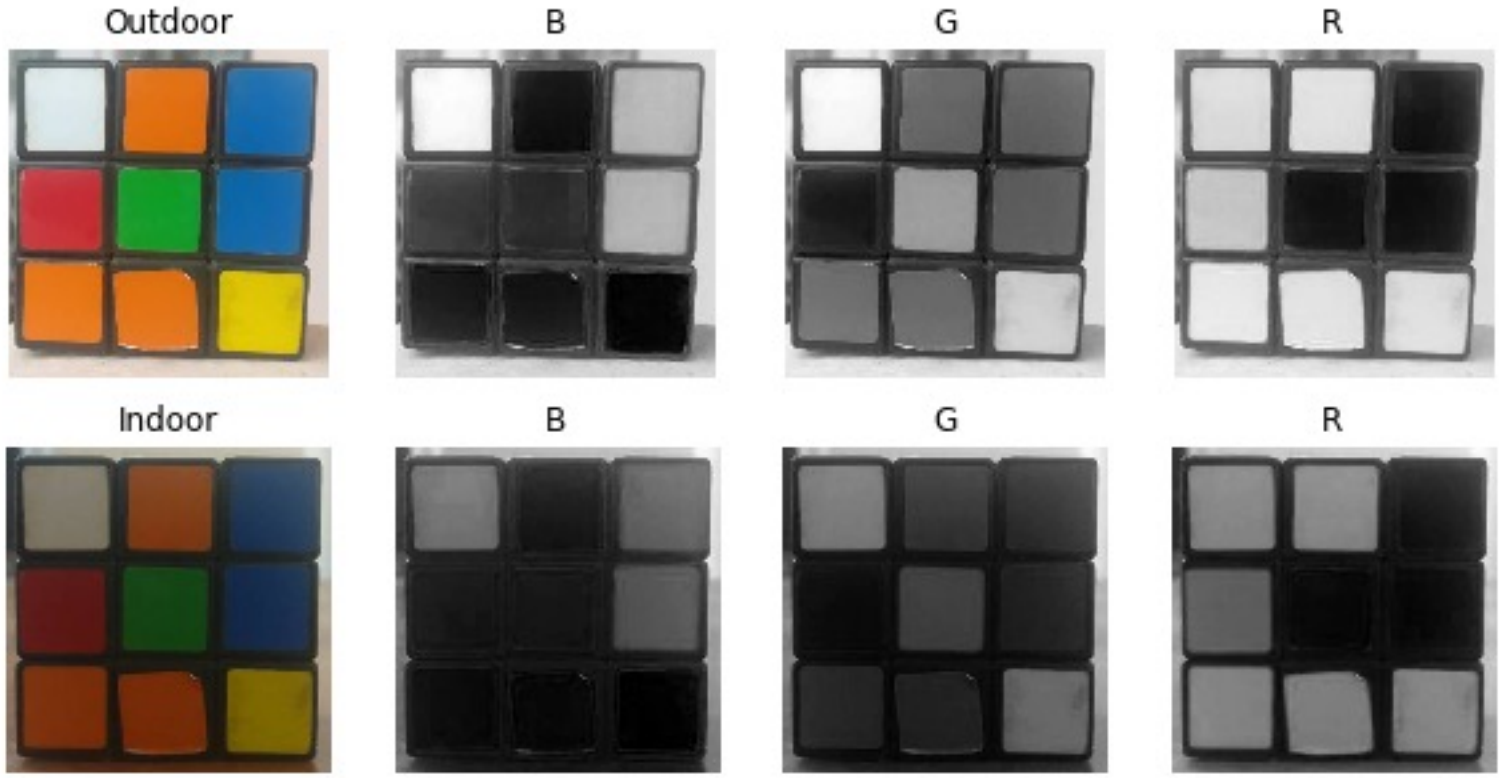


CIE L*a*b* Color Space



Tutorial on Colorspace

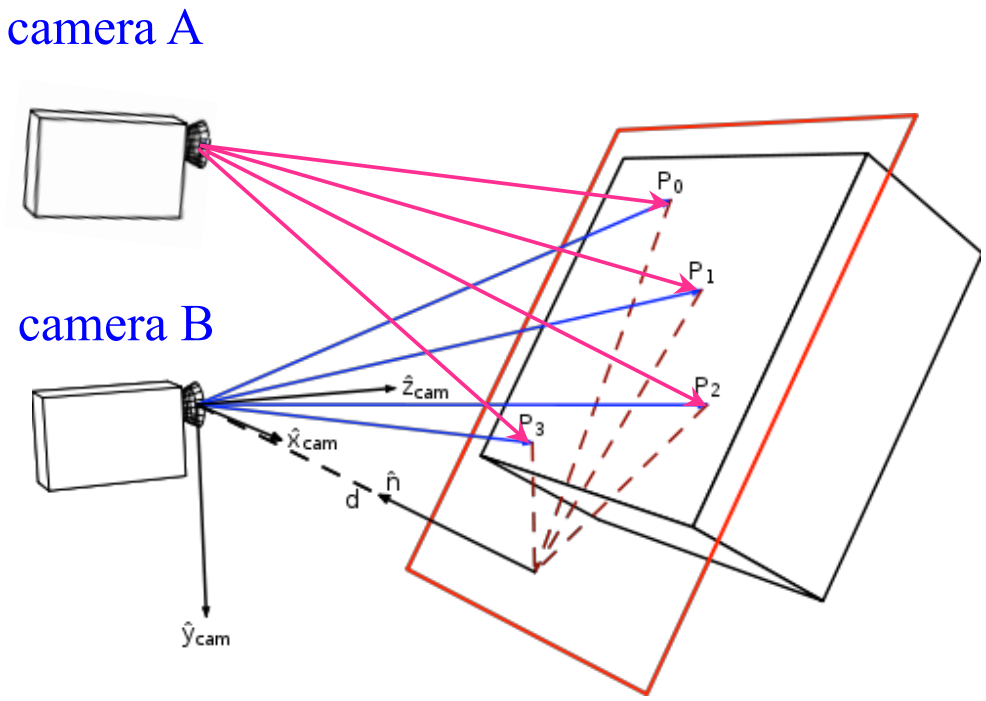
- https://learnopencv.com/color-spaces-in-opencv-cpp-python/



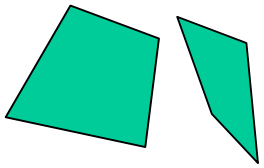
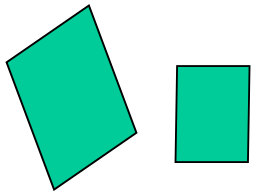
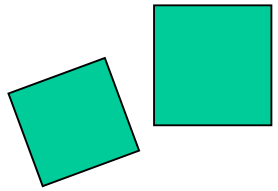
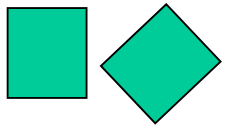
Projective Transform

- Move from one camera view to another

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} u_0 \\ v_0 \\ 1 \end{bmatrix}$$



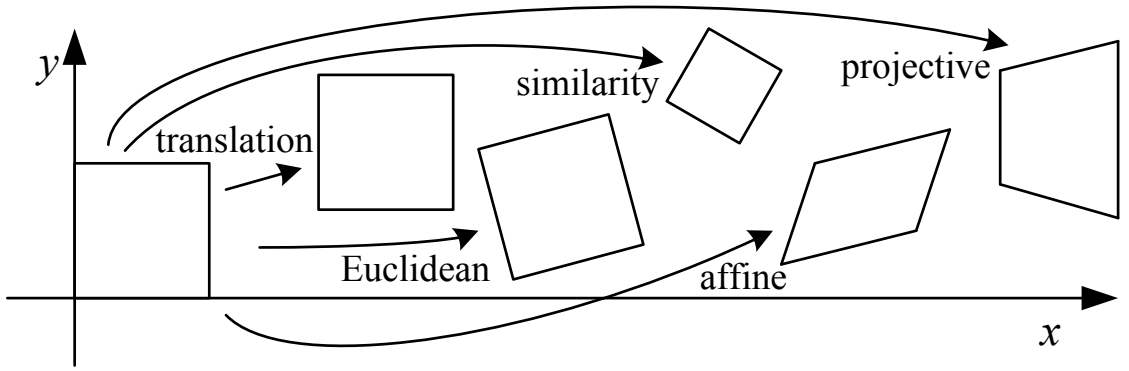
2D Transforms

		transformed squares	invariants
Projective 8dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, order of contact (intersection, tangency, inflection, etc.), cross ratio
Affine 6dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on parallel lines (e.g. midpoints), linear combinations of vectors (centroids), **The line at infinity l_∞**
Similarity 4dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratios of lengths, angles, The circular points I,J
Euclidean 3dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Absolute lengths, angles, areas

ETH

2D Transforms

Transformation	Matrix	# DoF	Preserves
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2\times 3}$	2	orientation
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2\times 3}$	3	lengths
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2\times 3}$	4	angles
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2\times 3}$	6	parallelism
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3\times 3}$	8	straight lines

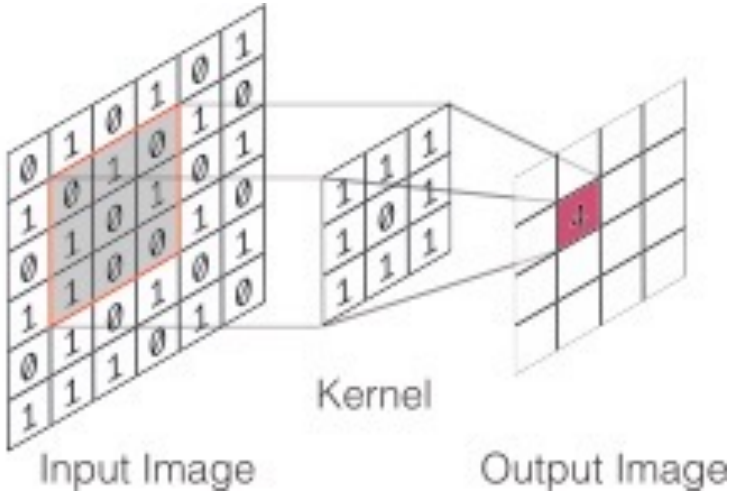


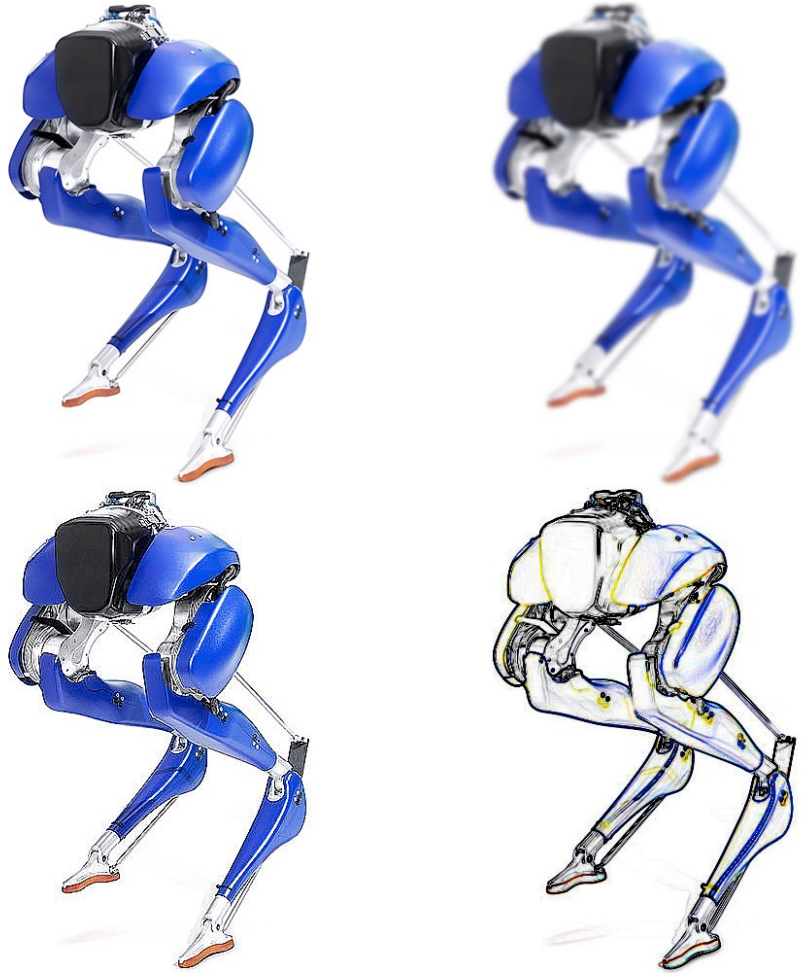
CV2 functions

- cv2.getAffineTransform()
  - 3 (u,v) point correspondances
- cv2.warpAffine()
  - `dst(x,y)=src(M11x+M12y+M13,M21x+M22y+M23)`
- cv2.getPerspectiveTransform()
  - 4 (u,v) point correspondances
- cv2.warpPerspective()
  - `dst(x,y)=src(M11x+M12y+M13M31x+M32y+M33,M21x+M22y+M23M31x+M32y+M33)`

Convolution

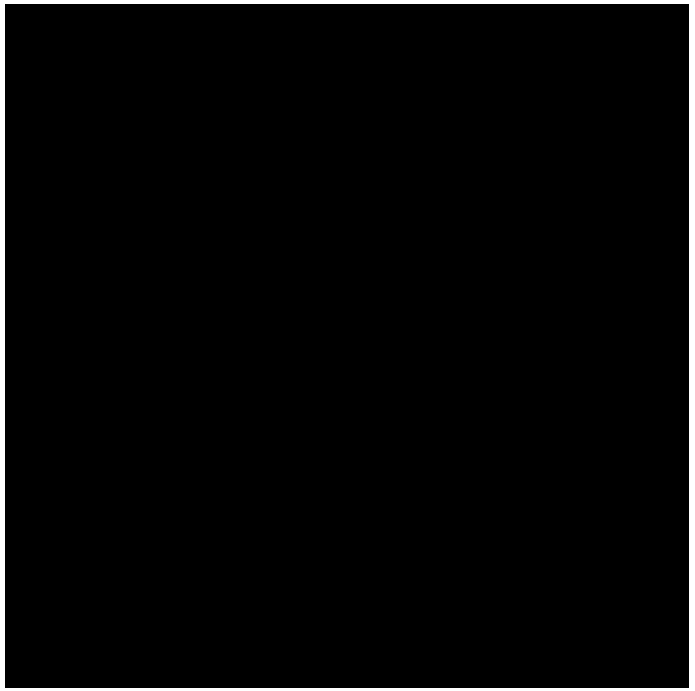
- Convolution is a fundamental image processing technique
- “Slide” kernel over input image and compute output image
- Blur, Sharpen, Edge detection





Thresholding

- cv2.threshold() ⇒ grayscale to binary image
- cv2.inRange() ⇒ binary image where white pixels meet range criteria



Morphological Filters

- Like convolution, but Boolean for binary images
- Erosion: 1 if image AND kernel == 1
- Dilation: 1 if image OR kernel == 1

Morphological Filters

- Can use consecutive erode/dilate steps to achive nois reduction or edge finding

Opening
erode then dilate

Closing
dilate then erode

Gradient
dilate minus erode

Moments

- How do we find the center of a group of pixels?
- Center of mass of set of objects: $x_{CM} = \frac{\sum_i m_i x_i}{M}$, $y_{CM} = \frac{\sum_i m_i y_i}{M}$,
- Area of pixels: $M = \sum_x \sum_y I(x, y)$
- General moment formula: $M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$
- Centroid: $(\mathrm{x}, \mathrm{y}) = (\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}})$
- Cv2.moments(image)

Contours

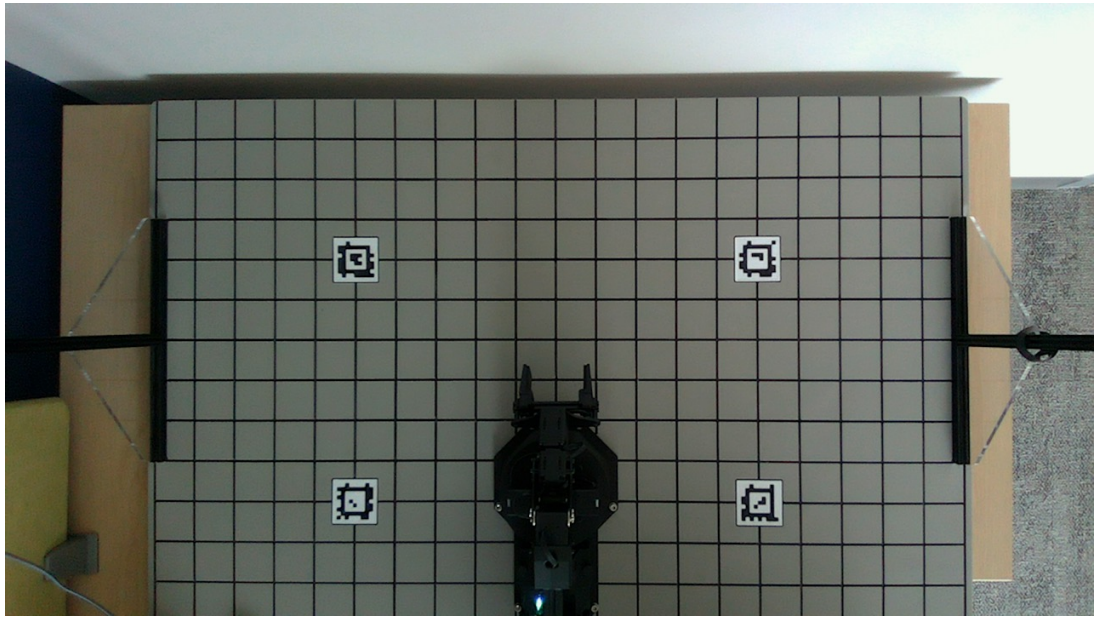
- Using moments works if we only have one blob after thresholding
- Can separate multiple blobs with contours
- cv2.contours() seperates blobs and returns a set of contours
- contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object
- Center of mass will be the same for contour as it is for blob

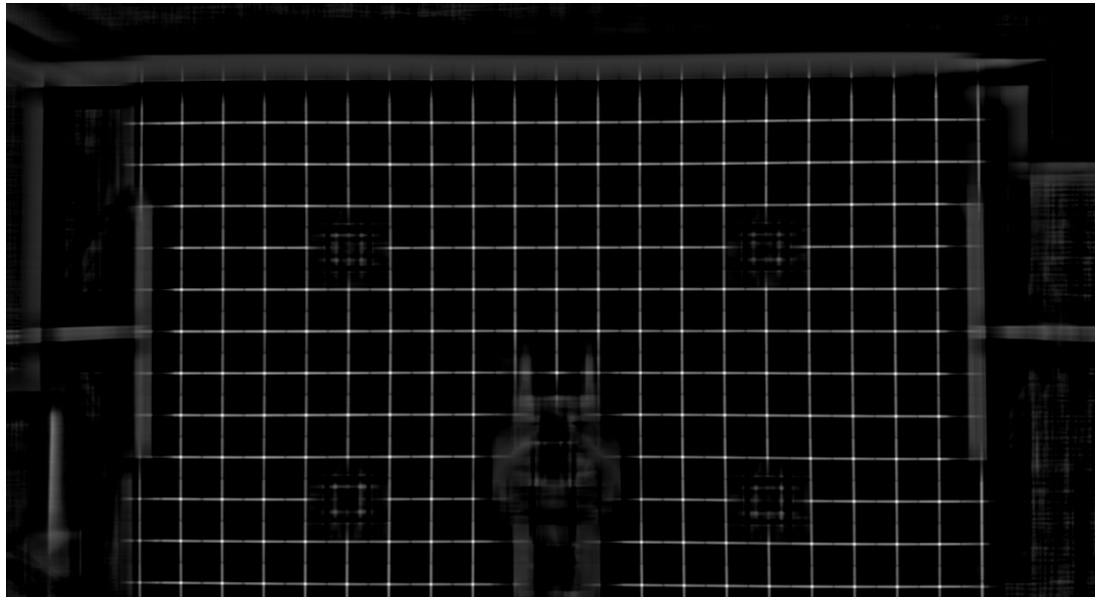
Building a Block Detector

- Segment the image based on color or depth value
- Remove noise from segmented images
- Find contours of the segments
- Calculate the moments for the contours to find the centroids of the blobs in the image
- Use cv2.minAreaRect() to find orientation of block
- Use centroid, orientation, depth value, inverse intrinsic matrix & inverse extrinsic matrix to find locations in workspace

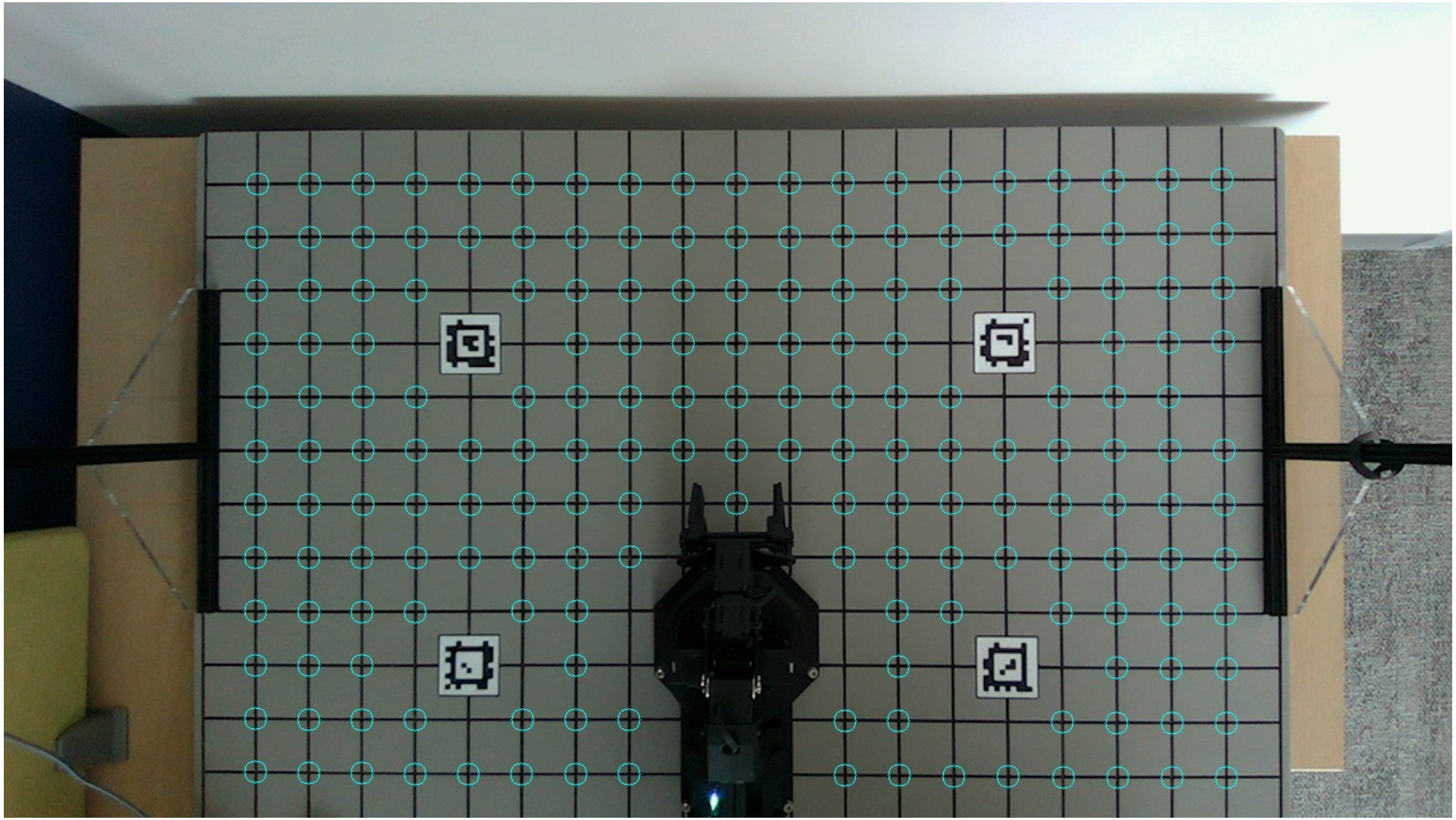
Template Matching

- cv2.matchTemplate(), cv2.minMaxLoc()

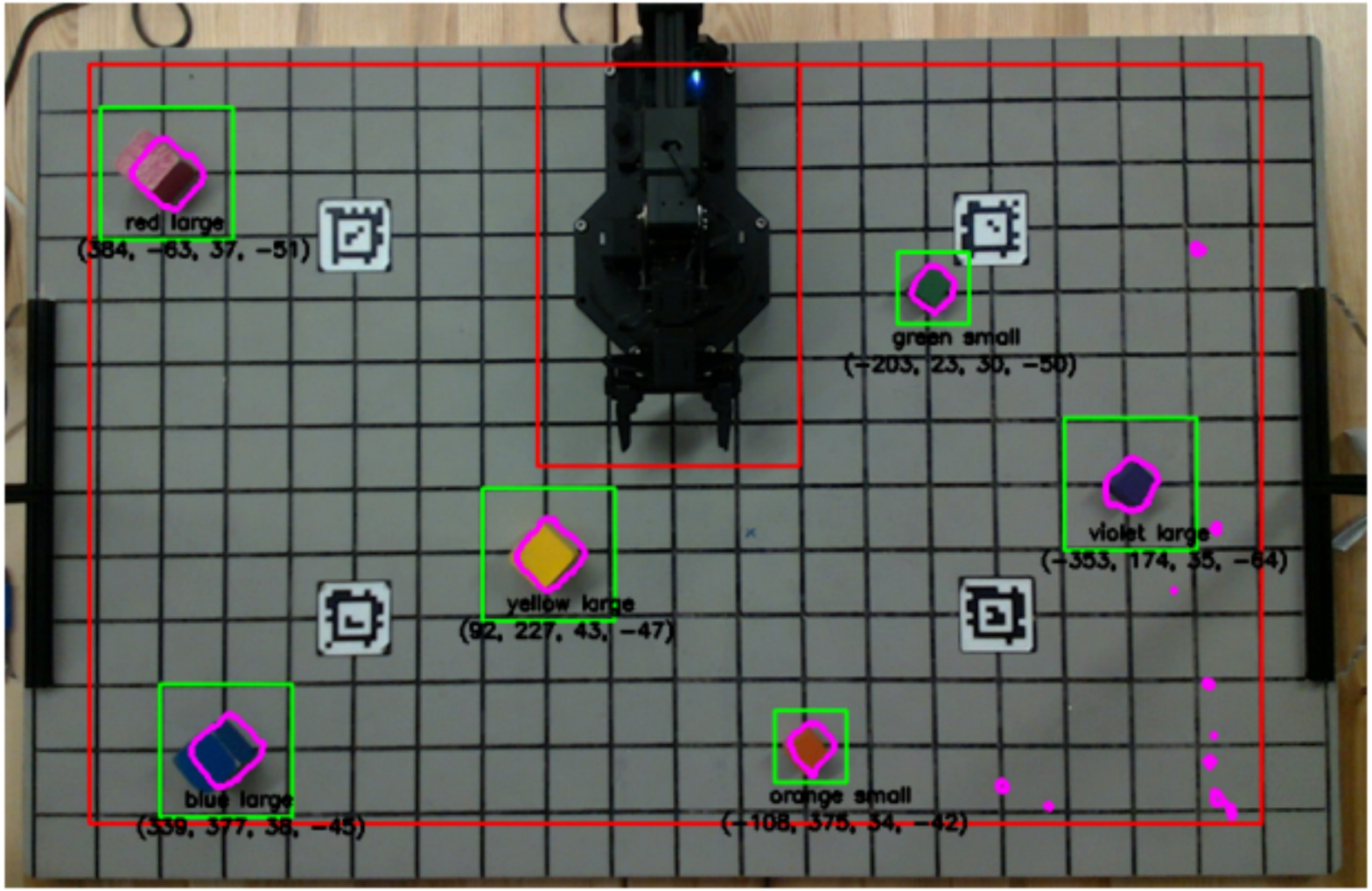




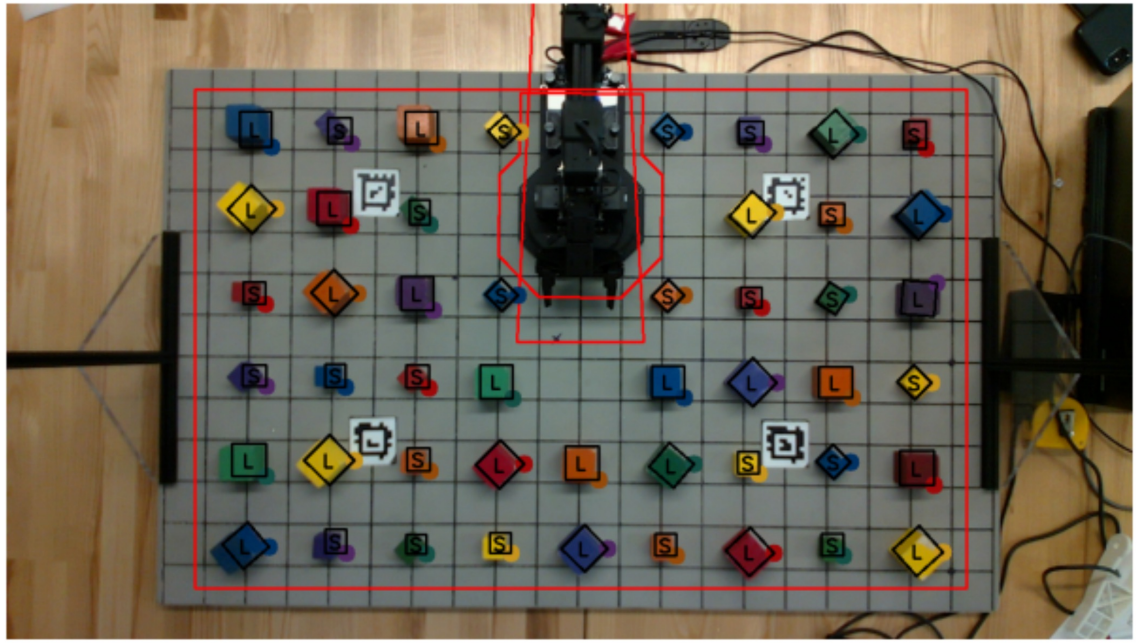
Template Matching



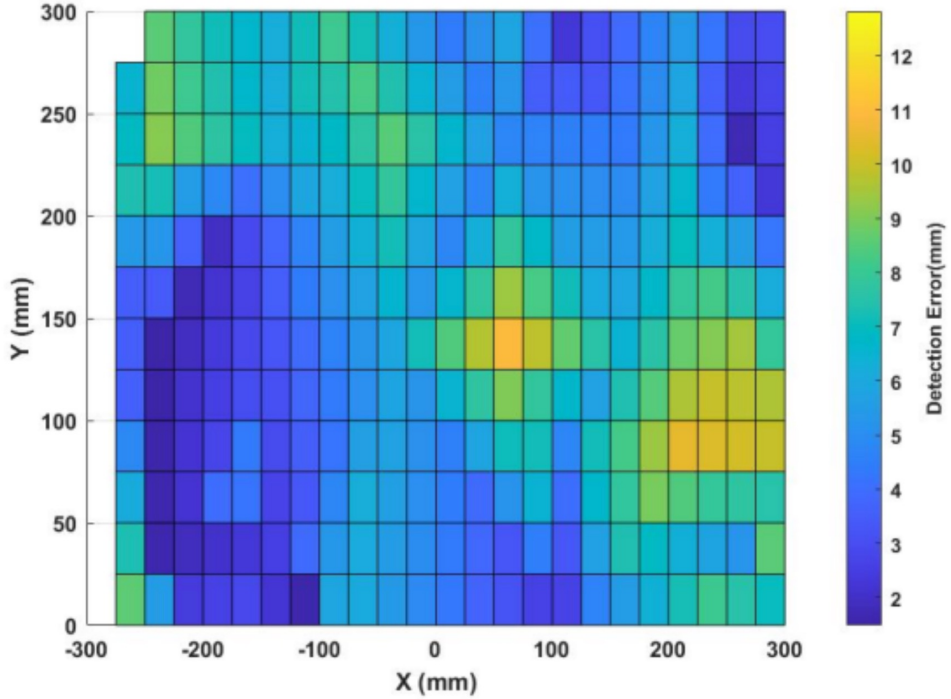
OpenCV Drawing



System Testing



S2T2W22



S2T4W22

New Calibration Objects