Hand Segmentation

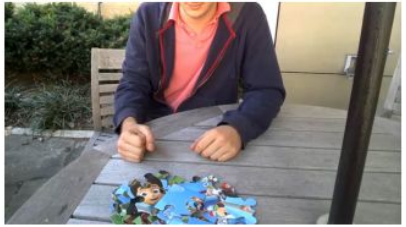
Advanced Image Processing

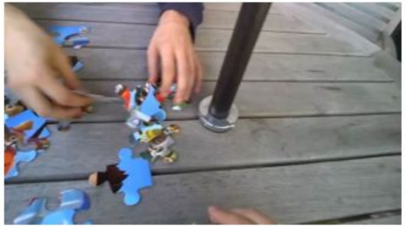
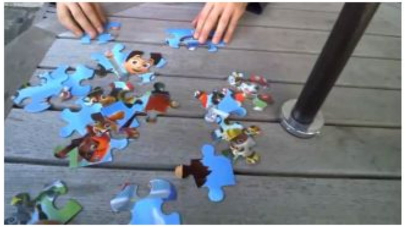
Marek Lichvár, Michal Švec

https://github.com/sinthoras50/AIP-project/tree/main

Dataset - Ego Hands

- Sourced from https://vision.soic.indiana.edu/projects/egohands/
- 48 google class videos / 4800 frames total
- Each video has 100 labelled ground-truth frames
- Each frame contains 1-4 hands in various positions performing various activities
- Matlab API only

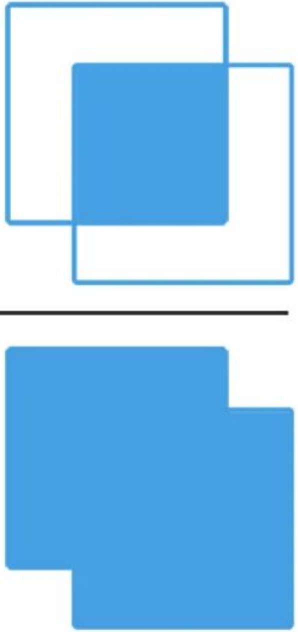


Intersection over Union Metric

- test dataset consist of 20 images

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Color-based segmentation

1. ORIGINAL

Color-based segmentation

1. ORIGINAL
2. **SKIN COLOR EXTRACTION (HSV)**

Color-based segmentation

1. ORIGINAL
2. SKIN COLOR EXTRACTION (HSV)
3. **3x EROSION**

Color-based segmentation

1. ORIGINAL
2. SKIN COLOR EXTRACTION (HSV)
3. 3x EROSION
4. **1x DILATION**

Color-based segmentation

1. ORIGINAL
2. SKIN COLOR EXTRACTION (HSV)
3. 3x EROSION
4. 1x DILATION
5. **FILLED CONTOURS WITH AREA > 10000**

Color-based segmentation

IOU: 72.95%

RESULT

GROUND TRUTH

Edge-based segmentation

1. ORIGINAL

Edge-based segmentation

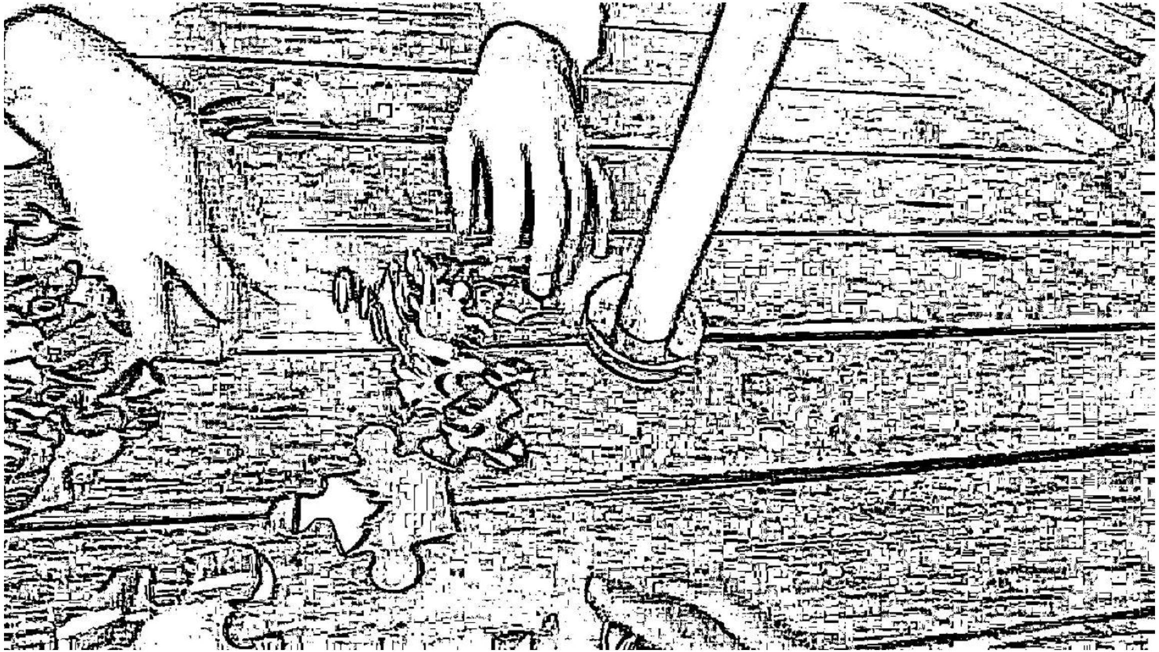
1. ORIGINAL
2. **GRAYSCALE**

Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. **HISTOGRAM EQUALIZATION**

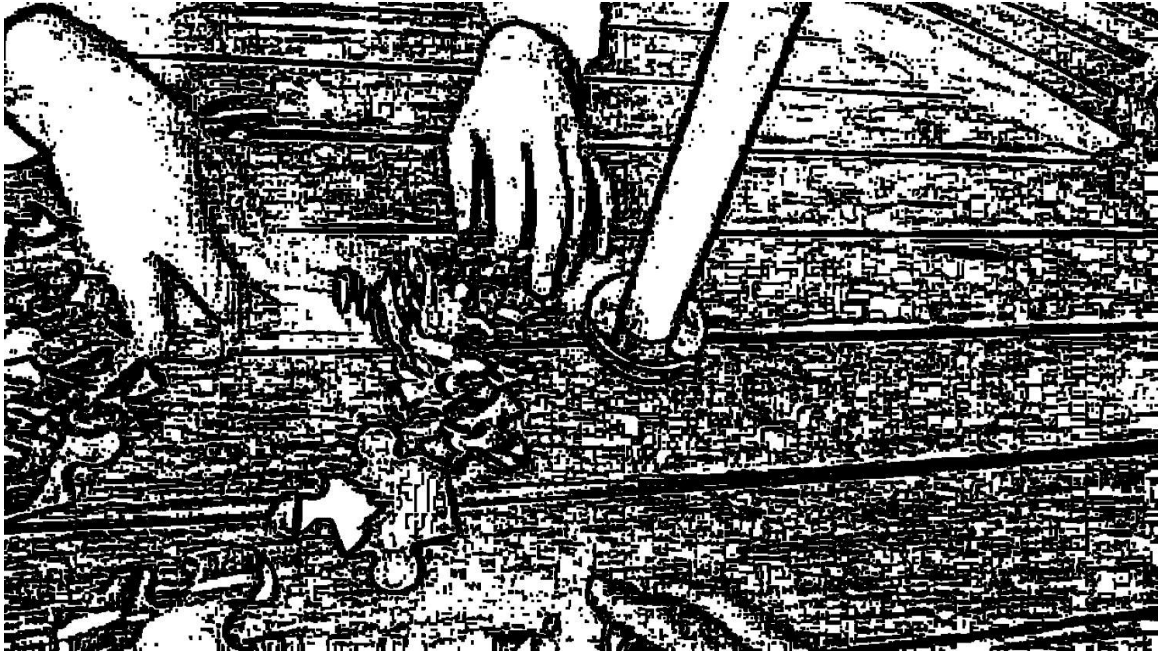
Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. **MEAN ADAPTIVE THRESHOLDING**



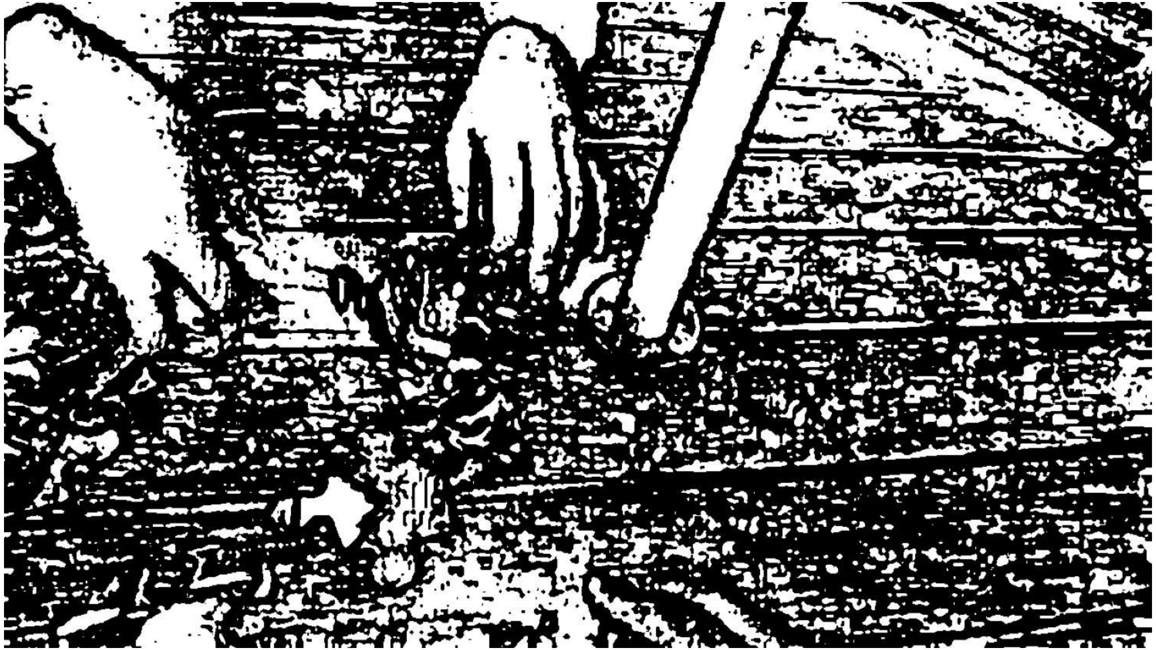
Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. MEAN ADAPTIVE THRESHOLDING
5. **EROSION**



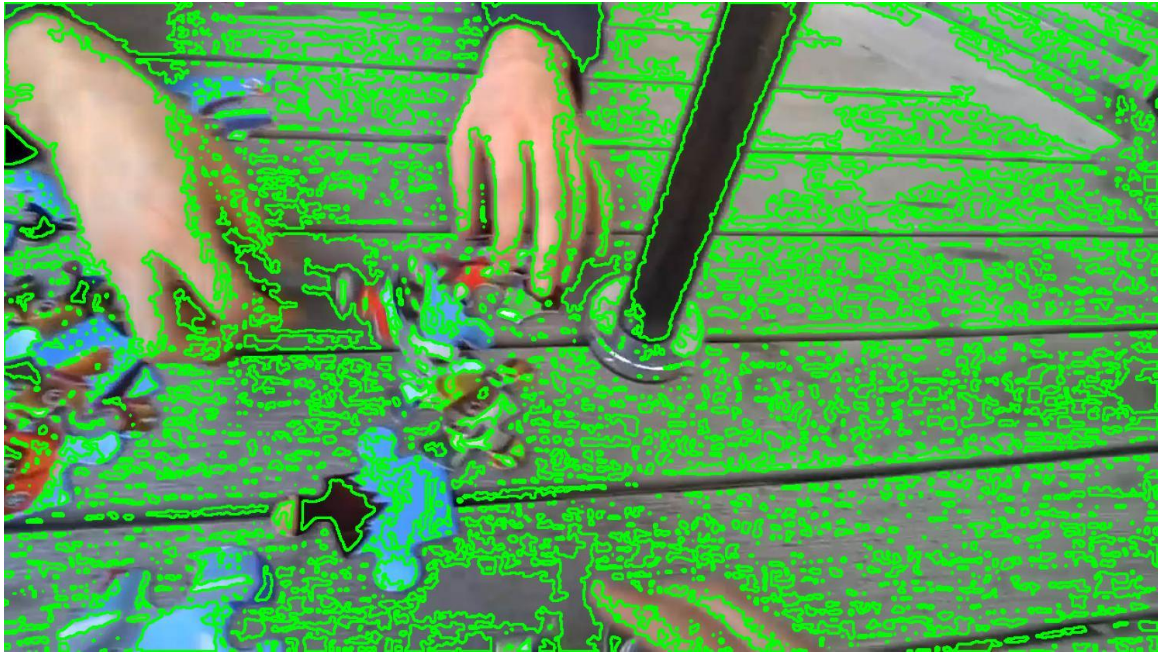
Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. MEAN ADAPTIVE THRESHOLDING
5. EROSION
6. **MEDIAN BLUR**



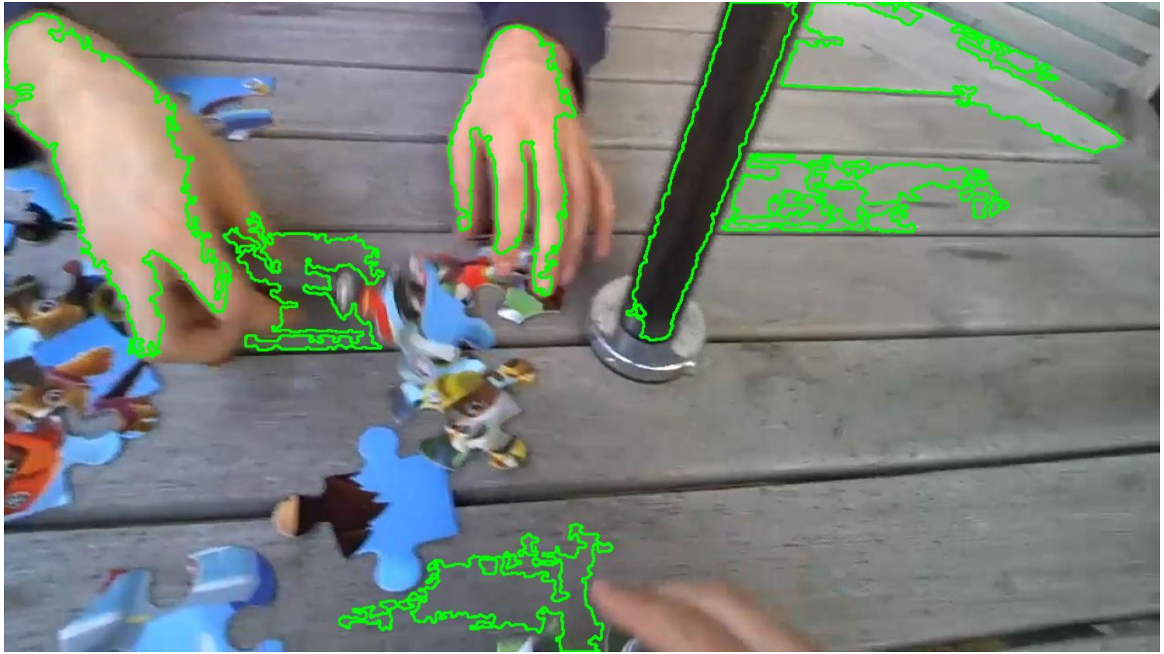
Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. MEAN ADAPTIVE THRESHOLDING
5. EROSION
6. MEDIAN BLUR
7. **CONTOURS**



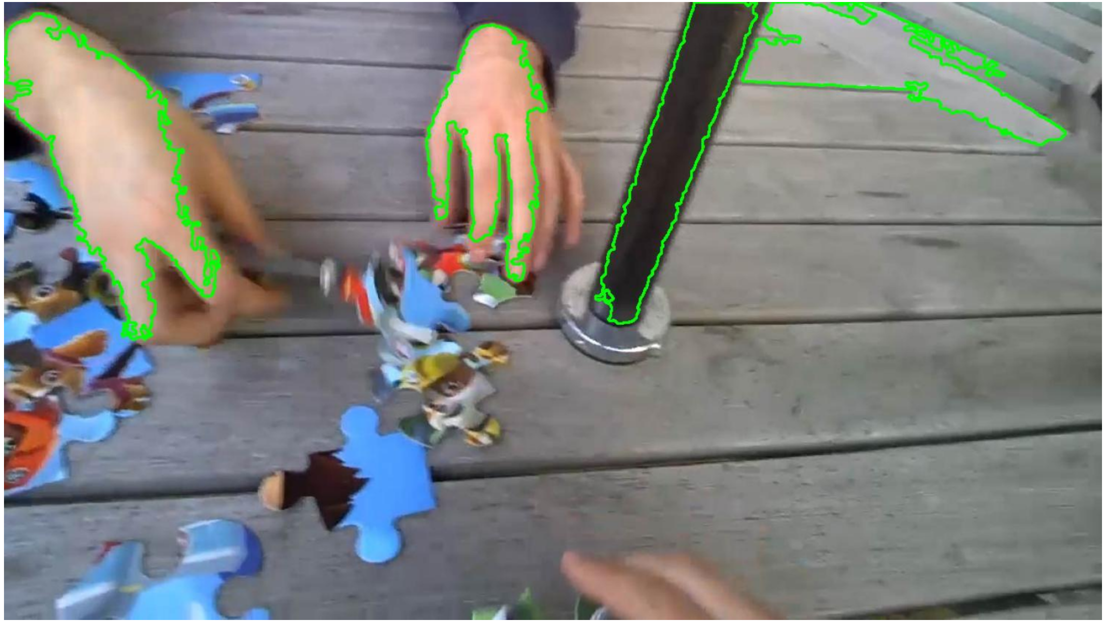
Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. MEAN ADAPTIVE THRESHOLDING
5. EROSION
6. MEDIAN BLUR
7. CONTOURS
8. **LARGE CONTOURS**



Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. MEAN ADAPTIVE THRESHOLDING
5. EROSION
6. MEDIAN BLUR
7. CONTOURS
8. LARGE CONTOURS
9. **SMOOTH CONTOURS**



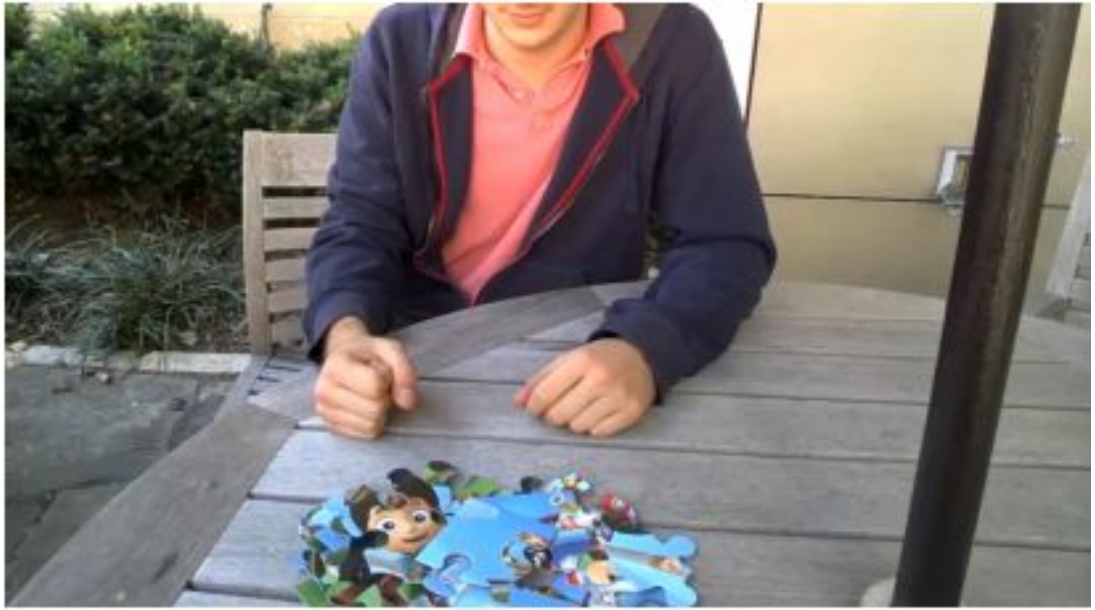
Edge-based segmentation

1. ORIGINAL
2. GRAYSCALE
3. HISTOGRAM EQUALIZATION
4. MEAN ADAPTIVE THRESHOLDING
5. EROSION
6. MEDIAN BLUR
7. CONTOURS
8. LARGE CONTOURS
9. SMOOTH CONTOURS
10. **RECTANGLE CONTOURS**

IoU: 6-12%

Color + contour based segmentation

1. ORIGINAL



Color + contour based segmentation

1. ORIGINAL
2. **SKIN EXTRACTION (HSV)**

Color + contour based segmentation

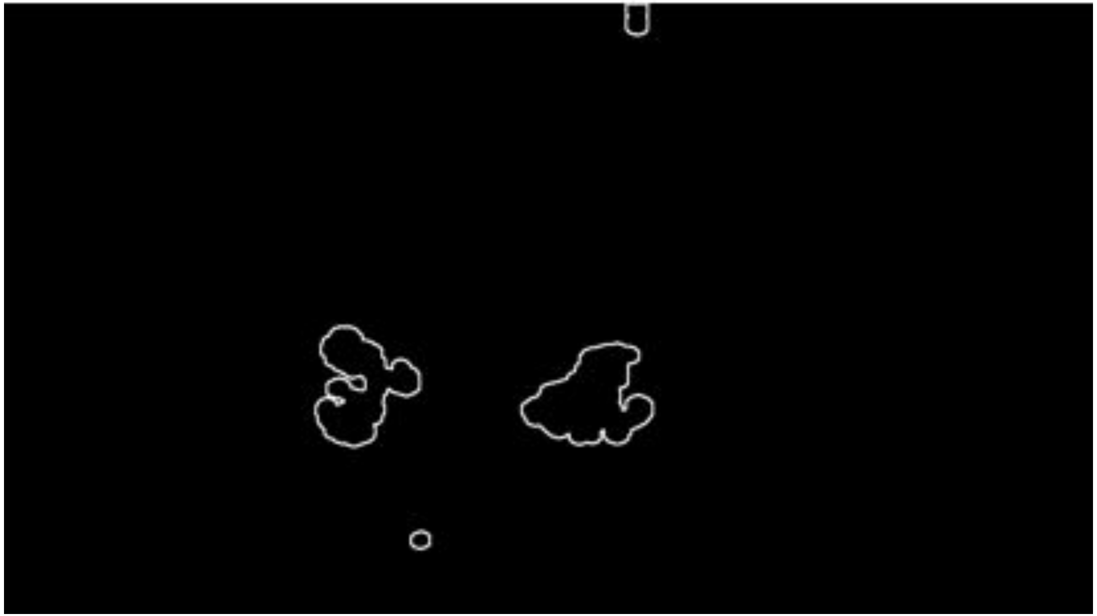
1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. **OPENED MASK**

Color + contour based segmentation

1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. OPENED MASK
4. **DILATED MASK**

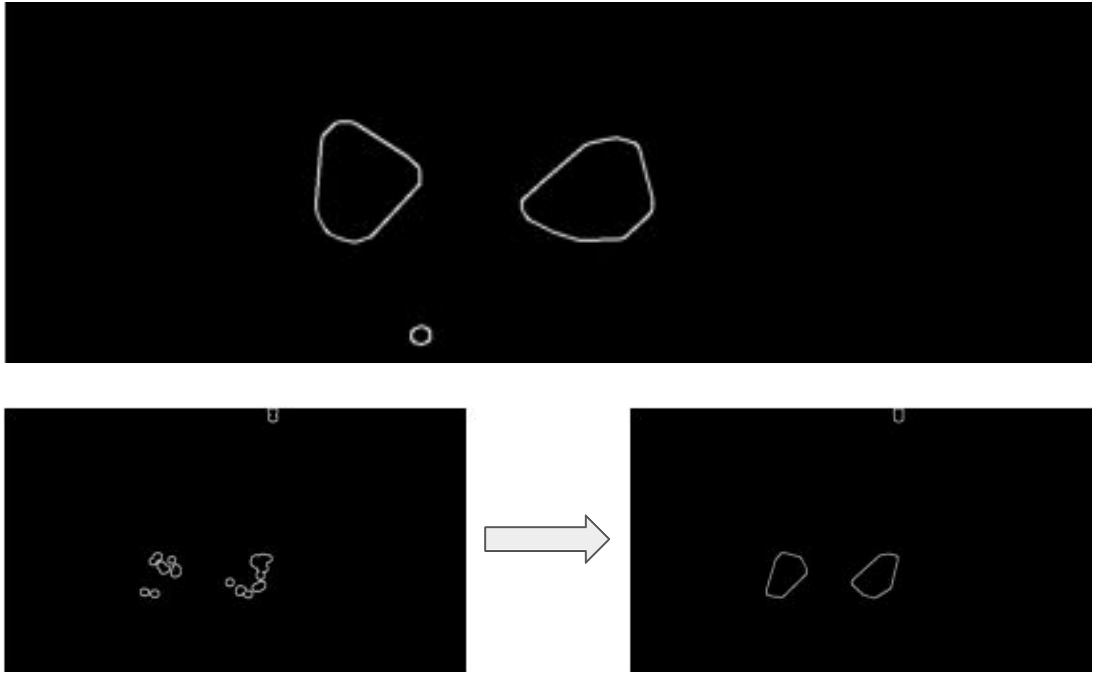
Color + contour based segmentation

1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. OPENED MASK
4. DILATED MASK
5. **DETECT CONTOURS**



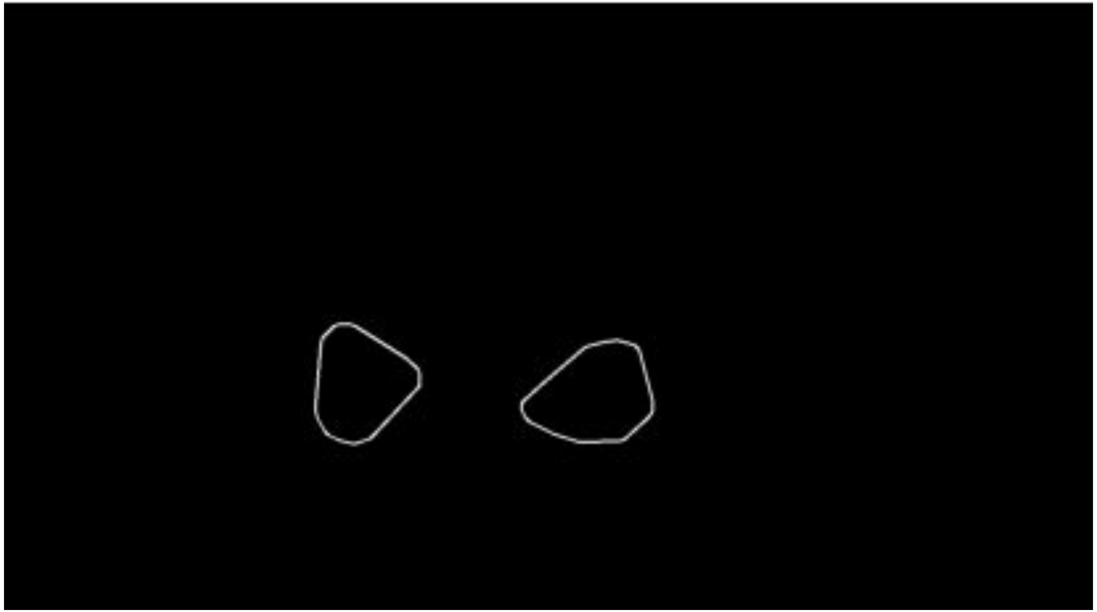
Color + contour based segmentation

1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. OPENED MASK
4. DILATED MASK
5. DETECT CONTOURS
6. **MERGE BASED ON DISTANCE & CONVEX HULL**



Color + contour based segmentation

1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. OPENED MASK
4. DILATED MASK
5. DETECT CONTOURS
6. MERGE BASED ON DISTANCE & CONVEX HULL
7. **FILTER BY AREA**

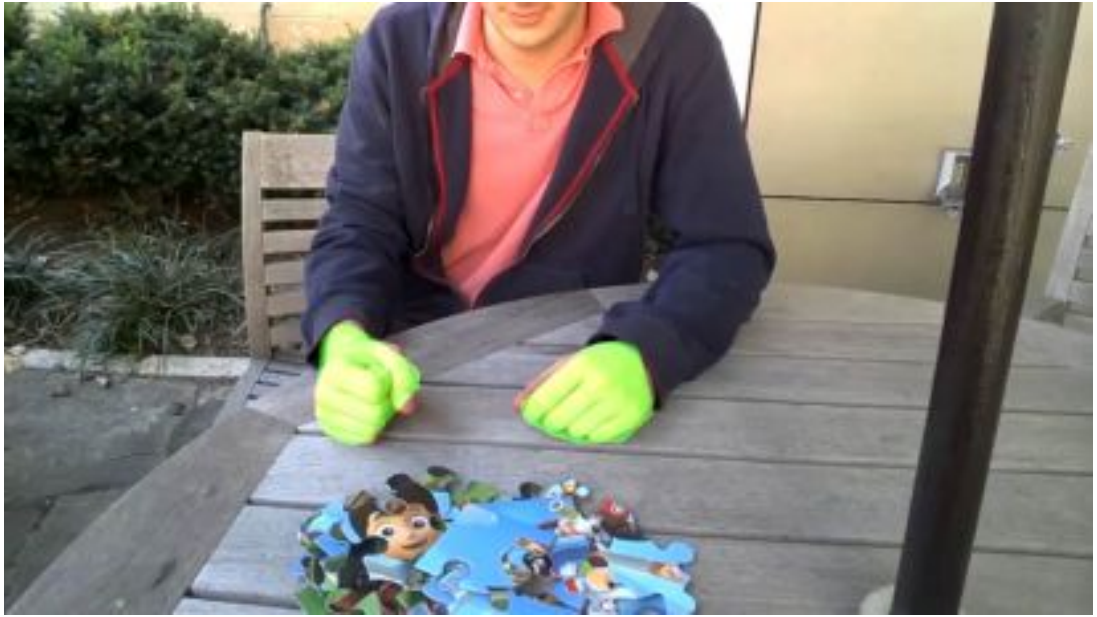


Color + contour based segmentation

1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. OPENED MASK
4. DILATED MASK
5. DETECT CONTOURS
6. MERGE BASED ON DISTANCE & CONVEX HULL
7. FILTER BY AREA
8. **CREATE MASK**

Color + contour based segmentation

1. ORIGINAL
2. SKIN EXTRACTION (HSV)
3. OPENED MASK
4. DILATED MASK
5. DETECT CONTOURS
6. MERGE BASED ON DISTANCE & CONVEX HULL
7. FILTER BY AREA
8. **CREATE MASK**



IoU: ~65%

K-means based segmentation

1. ORIGINAL

K-means based segmentation

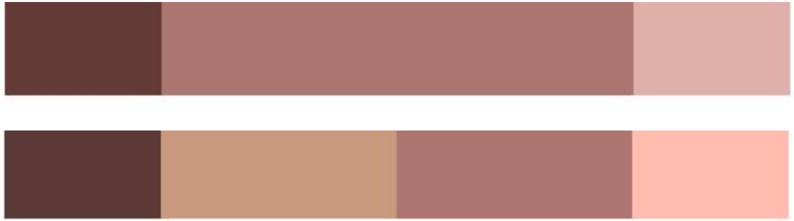
1. ORIGINAL
2. **EQUALIZE V IN HSV**

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. **REDUCE COLORS WITH K-MEANS**

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. REDUCE COLORS WITH K-MEANS
4. **FIND CLOSEST COLOR TO SKIN**



```python
# use L2 distance in LAB color space to determine wether pixels should be in the mask or not
def in_range(img, color, cutoff):
  img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
  distances = np.linalg.norm(img_lab - color, axis=-1)
  mask = (distances < cutoff).astype(np.uint8) * 255
  return mask

# convert BGR color into HSV, attempt to calculate lower and upper tones of the same color
def calculate_color_bounds(color):
  color = cv2.cvtColor(np.array([[color]]), cv2.COLOR_BGR2HSV)[0][0]

  min_hsv = color.copy()
  min_hsv[2] = 100
  min_hsv[1] *= 1.3
  min_hsv = np.clip(min_hsv, 0, 255).astype(np.uint8)

  max_hsv = color.copy()
  max_hsv[2] = 245
  max_hsv[1] *= 0.7
  max_hsv = np.clip(max_hsv, 0, 255).astype(np.uint8)

  min_bgr = cv2.cvtColor(np.array([[min_hsv]]), cv2.COLOR_HSV2BGR)[0][0]
  max_bgr = cv2.cvtColor(np.array([[max_hsv]]), cv2.COLOR_HSV2BGR)[0][0]
  return [min_bgr, max_bgr]

# attempt to find closest color within an image using L2 distance
def find_closest_color(img, color):
  img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
  distances = np.linalg.norm(img_lab - color, axis=-1)
  min_distance = np.min(distances)
  closest_pixel_indices = np.unravel_index(np.argmin(distances), distances.shape)
  closest_color = img[closest_pixel_indices]

  return [closest_color, closest_pixel_indices, min_distance]
```

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. REDUCE COLORS WITH K-MEANS
4. FIND CLOSEST COLOR TO SKIN
5. **COMBINE COLORS IN RANGE INTO A MASK**

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. REDUCE COLORS WITH K-MEANS
4. FIND CLOSEST COLOR TO SKIN
5. COMBINE COLORS IN RANGE INTO A MASK
6. **CLOSE MASK AND DETECT CONTOURS**

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. REDUCE COLORS WITH K-MEANS
4. FIND CLOSEST COLOR TO SKIN
5. COMBINE COLORS IN RANGE INTO A MASK
6. CLOSE MASK AND DETECT CONTOURS
7. **FILTER CONTOURS BASED ON FEATURES**
    - compare against precomputed features
    - weighted sum of differences to measure a score

```python
# calculate features from the polygon specifying a detected contour
def detect_features(polygon, image):
  pts = np.array(polygon, dtype=np.int32)

  hull = cv2.convexHull(pts)
  hull_area = cv2.contourArea(hull)
  polygon_area = cv2.contourArea(pts)

  rect = cv2.minAreaRect(pts) # [[cx, cy], [w, h], [rotation]]
  box = cv2.boxPoints(rect)
  box = box.astype(np.int32)
  width, height = rect[1]
  aspect_ratio = width / height if height > 0 else 0
  bbox_area = width * height
  solidity = polygon_area / hull_area if hull_area > 0 else 0
  extent = polygon_area / bbox_area if bbox_area > 0 else 0
  rectangularity = bbox_area / polygon_area if polygon_area > 0 else 0

  img_mask = np.zeros(image.shape[:2], dtype=np.uint8)
  cv2.fillPoly(img_mask, [pts], 255)
  masked_img = cv2.bitwise_and(image, image, mask=img_mask)
  masked_pixels = masked_img[img_mask == 255]
  color = np.mean(masked_pixels, axis=0).astype(np.uint8)
  color = cv2.cvtColor(np.array([[color]]), cv2.COLOR_BGR2LAB)[0][0]

  return {
      'c_hull_area': hull_area,
      'polygon_area': polygon_area,
      'aspect_ratio': aspect_ratio,
      'bbox_area': bbox_area,
      'solidity': solidity,
      'extent': extent,
      'rectangularity': rectangularity,
      'color': color
  }
```

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. REDUCE COLORS WITH K-MEANS
4. FIND CLOSEST COLOR TO SKIN
5. COMBINE COLORS IN RANGE INTO A MASK
6. CLOSE MASK AND DETECT CONTOURS
7. **FILTER CONTOURS BASED ON FEATURES**
    - compare against precomputed features
    - weighted sum of differences to measure a score

C. hull area

Polygon area

Aspect ratio - width / height

Bbox area - width * height

Solidity - poly / hull

Extent - poly / bbox

Rectangularity - bbox / poly

Color - normalized [L, A, B]

K-means based segmentation

1. ORIGINAL
2. EQUALIZE V IN HSV
3. REDUCE COLORS WITH K-MEANS
4. FIND CLOSEST COLOR TO SKIN
5. COMBINE COLORS IN RANGE INTO A MASK
6. CLOSE MASK AND DETECT CONTOURS
7. FILTER CONTOURS BASED ON FEATURES
8. **CREATE A MASK FROM THE CONTOURS**

IoU: ~38%

Notes & observations

- Equalizing V channel in HSV drastically improved accuracy
- Subject to clusters getting initialized badly
- Attempts to initialize own centers unsuccessful
- Not performant - images on average require k > 9
- Not so great with cv2.inRange
- Contour detection could likely be improved
- Unresolved problem with finding closest color using L2 distance

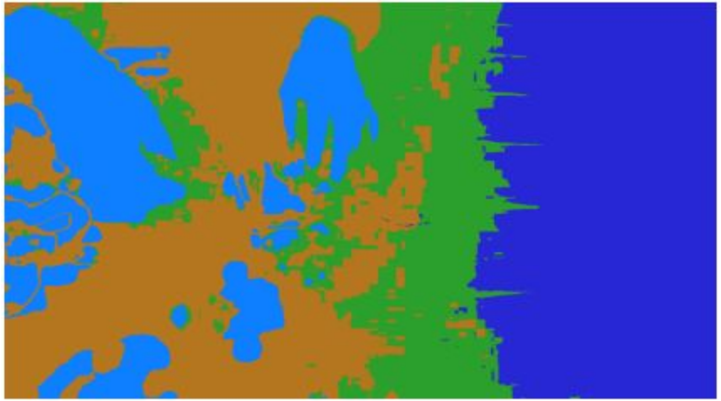
Other features

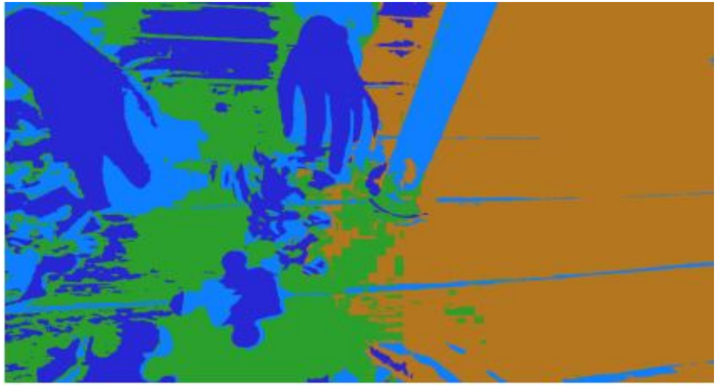
```
features_obj = {
  'convex_hull_areas': convex_hull_areas,
  'polygon_areas': polygon_areas,
  'bbox_areas': bbox_areas,
  'solidities': solidities,
  'extents': extents,
  'rectangularities': rectangularities,
  'aspect_ratios': aspect_ratios,
  'avg_c_hull_area': np.mean(flat_c_hull_areas),
  'avg_polygon_area': np.mean(flat_polygon_areas),
  'avg_solidity': np.mean(flat_solidities),
  'avg_bbox_area': np.mean(flat_bbox_areas),
  'avg_extent': np.mean(flat_extents),
  'avg_rectangularity': np.mean(flat_rectangularities),
  'avg_aspect_ratio': np.mean(flat_aspect_ratios),
  'max_c_hull_area': np.max(flat_c_hull_areas),
  'max_polygon_area': np.max(flat_polygon_areas),
  'max_solidity': np.max(flat_solidities),
  'max_bbox_area': np.max(flat_bbox_areas),
  'max_extent': np.max(flat_extents),
  'max_rectangularity': np.max(flat_rectangularities),
  'max_aspect_ratio': np.max(flat_aspect_ratios),
  'min_c_hull_area': np.min(flat_c_hull_areas),
  'min_polygon_area': np.min(flat_polygon_areas),
  'min_solidity': np.min(flat_solidities),
  'min_bbox_area': np.min(flat_bbox_areas),
  'min_extent': np.min(flat_extents),
  'min_rectangularity': np.min(flat_rectangularities),
  'min_aspect_ratio': np.min(flat_aspect_ratios),
  'min_color_BGR': averaged_colors[2],
  'max_color_BGR': averaged_colors[0],
  'avg_minmax_color_BGR': averaged_colors[1],
  'avg_pixels_color_BGR': averaged_colors[3],
  'min_color_LAB': cv2.cvtColor(np.array([[averaged_colors[2]]]), cv2.COLOR_BGR2LAB)[0][0],
  'max_color_LAB': cv2.cvtColor(np.array([[averaged_colors[0]]]), cv2.COLOR_BGR2LAB)[0][0],
  'avg_minmax_color_LAB': cv2.cvtColor(np.array([[averaged_colors[1]]]), cv2.COLOR_BGR2LAB)[0][0],
  'avg_pixels_color_LAB': cv2.cvtColor(np.array([[averaged_colors[3]]]), cv2.COLOR_BGR2LAB)[0][0]
}
```

[0.391, 0.393, 0.802, 0.392, 0.740, 0.137, 0.264, 0.615, 0.552, 0.564]

Additional attempts

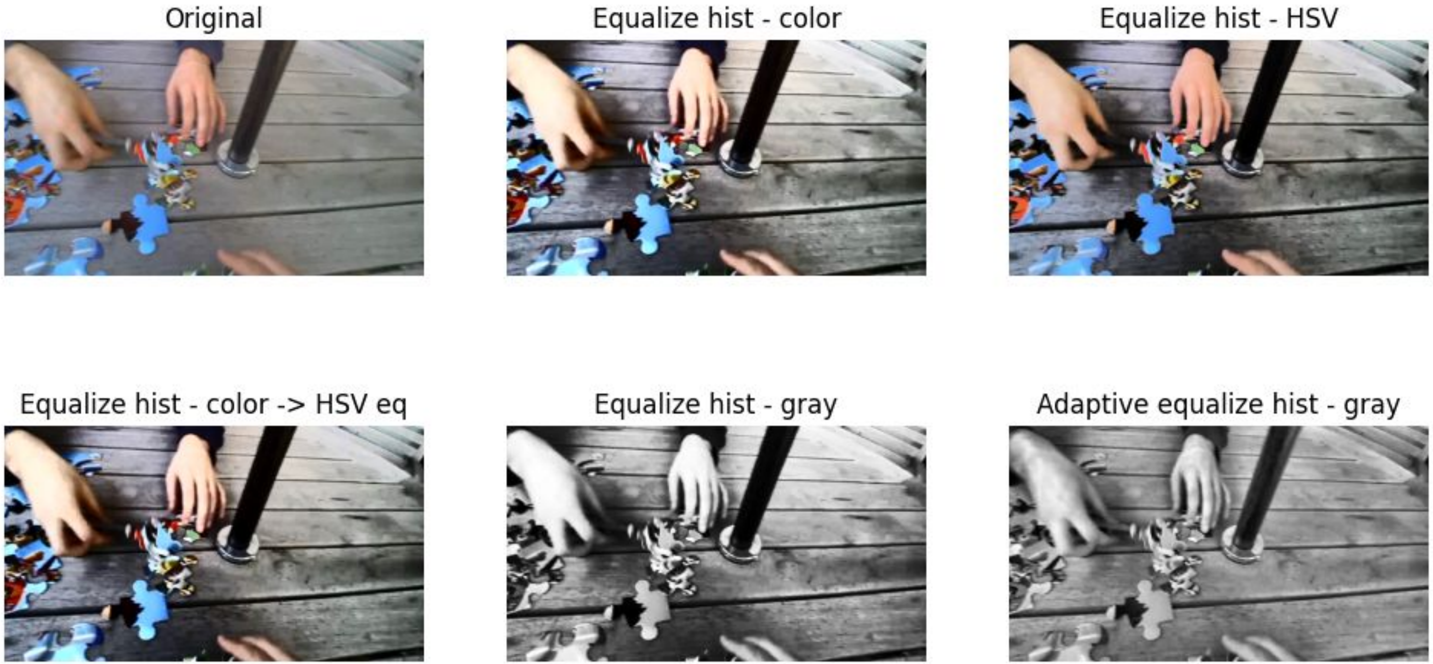
- Segmentation using K-Means with weighted coords
- Segmentation using K-Means with weighted gradients
    - Vertical & horizontal gradients found using sobel filter





Color enhancement

Segmentation methods were tested using the following enhancements



Summary

Algorithm	IoU score
color-based segmentation	72.95%
edge-based segmentation	6 - 12%
color/contours based segmentation	64.66%
K-means based segmentation	30 - 38%