To calculate the angular displacement of an object, I followed these steps:

1. **Finding the Center**: I used object detection models like YOLO and torchvision to locate the center of the object.
2. **Cropping the Image**: I cropped the image to perfectly fit the object, ensuring the center of the image matched the object's center, creating a center of mass frame.
3. **Calculating Angular Displacement**:
    1. I identified a dominant feature in the template image and found the same feature in the test image, relative to the center.
    2. I calculated the dot product of unit vectors to get the cosine of the object's rotation angle.
    3. Taking the cosine inverse gave me the rotation angle of the object in the test image.

For detecting and matching features, I used three methods:

1. **Canny Edge Detection**: This method reduced noise in contour detection. When the object faced up, two close circles were detected using the Hough Circles method.
2. **FLANN Algorithm**: A popular method for matching features between two images. Its accuracy depends on two parameters: sigma and the coefficient of n.distance in the following condition:

```
for m, n in matches:
  if m.distance < a * n.distance
```

3. **Template Matching**: This method finds the best fit of a template from the template image in the test image. However, since the object is quite symmetrical, template matching was not very effective.

Convention: -

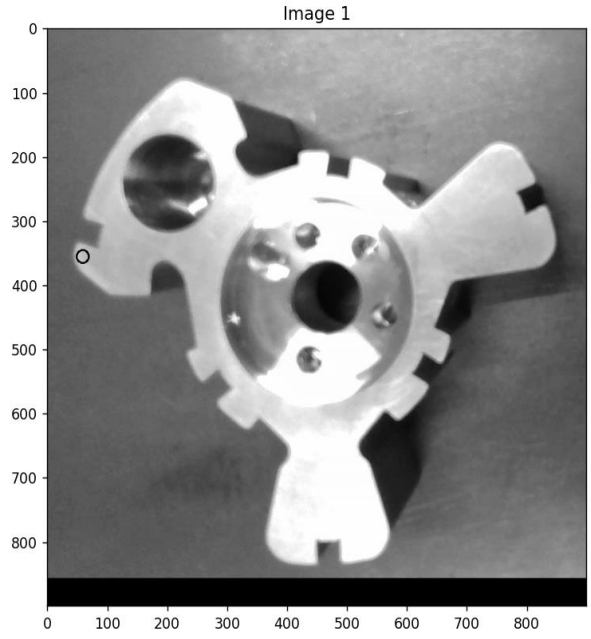
1. Object facing up

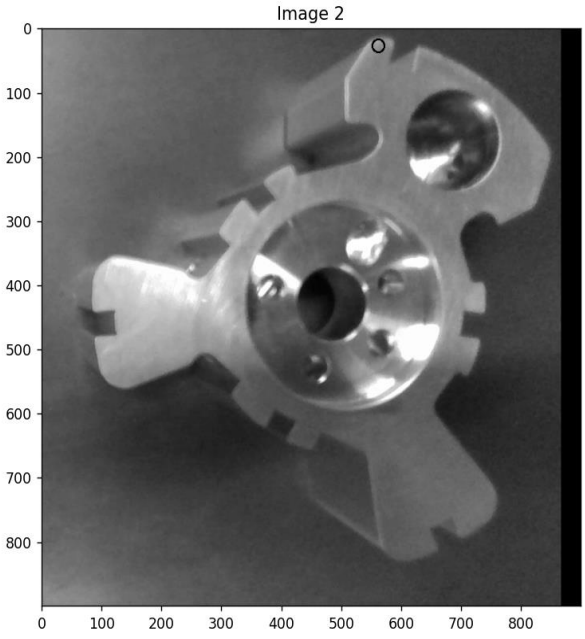
2. Object facing down

Below are the correct outputs for different ways of feature matching: -

1. Using Circle.py:

2. Using FLANN: A Black circle resembles the same feature in both the images.

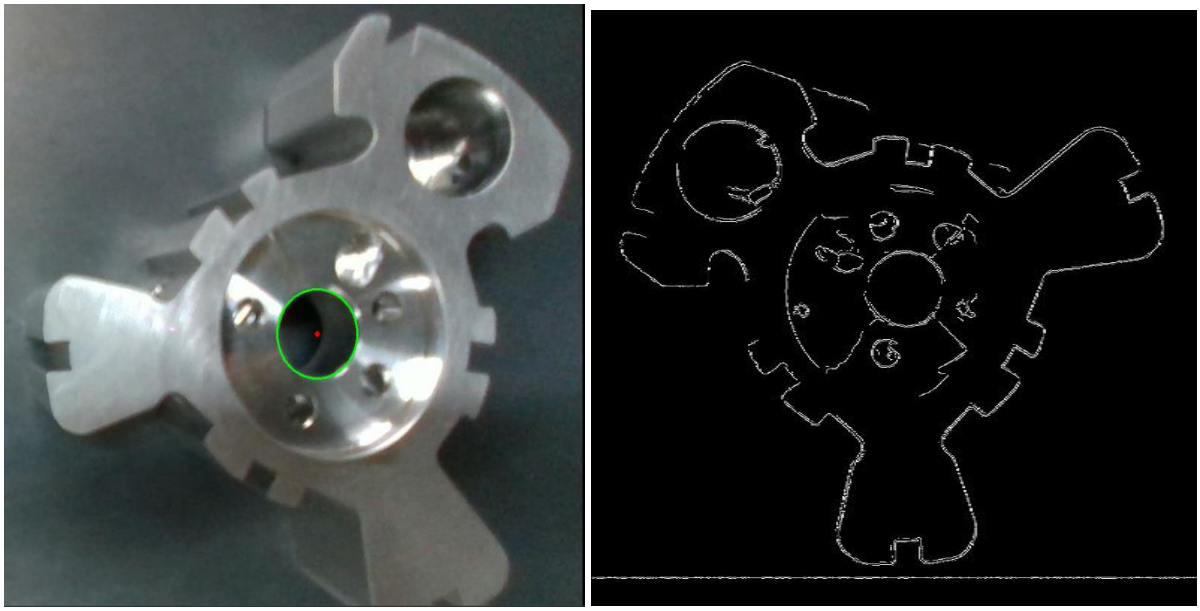




3. Using Template matching: (Not that accurate)

Below are the incorrect outputs for different ways of feature matching: -

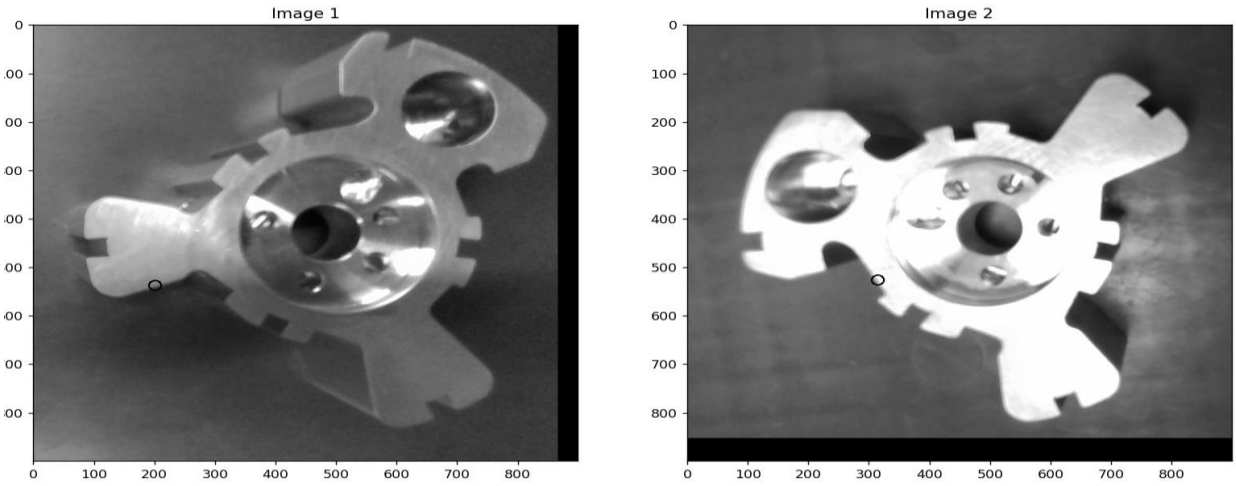
1. Using Circle.py: (This method doesn't work when the object is facing down)



Here the major issue is in canny edge detection. If the outer circle is blurred or somewhat distorted, the canny edge detection method will make it an incomplete circle which will get eliminated in the code (remove the semi-circle section).

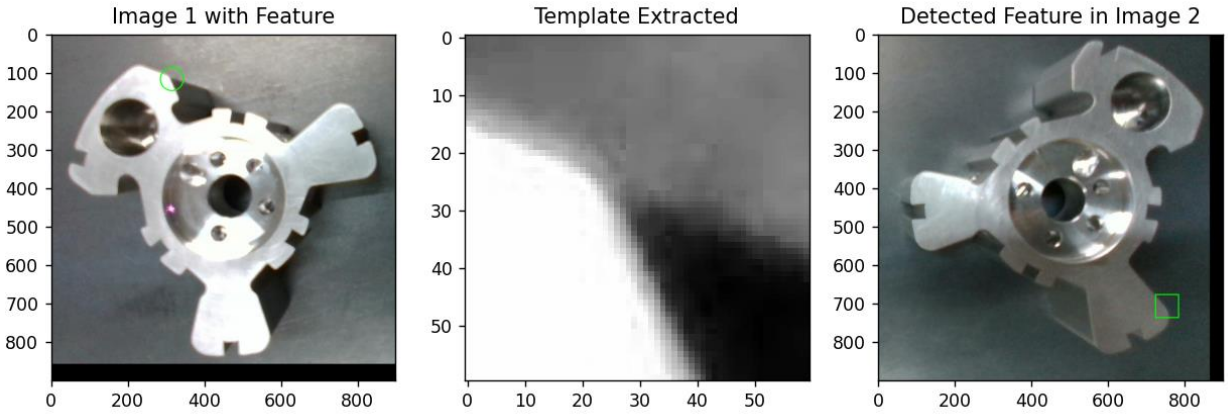
The canny image is attached to the previous working example, you can notice both the outer and inner circles here are quite sharp, but in canny edge detection of the current image is incomplete thus circle is not detected here.

2. Using FLANN:



You can notice above that the highlighted part of both images is quite similar. This algorithm thus gives incorrect output when symmetries are present at high amounts in the image.

3. Using template matching:



Template matching faces the same issue faced by FLANN; the extracted template has high probabilities at various locations for the given object due to symmetries.

By observation, I found accuracies for feature matching as below:
FLANN > Canny's edge circle >Template matching.

Ways to improve the project:

Currently YOLO model is not able to differentiate between facing up and facing down objects. To do that, more inputs are required which can be achieved by image augmentation i.e. feeding facing up and facing down images at multiple orientations of the same images.

After properly distinguishing the images, FLANN accuracy can be improved by considering various cases. This will improve accuracy in the case when the object is facing up.

To make FLANN and template matching more accurate, we should focus on selecting valid dominant features. These are features that appear in only one place in both images. Another way to improve accuracy could be increasing the template size, but this might lead to inaccuracies in finding the feature's location in the test image.