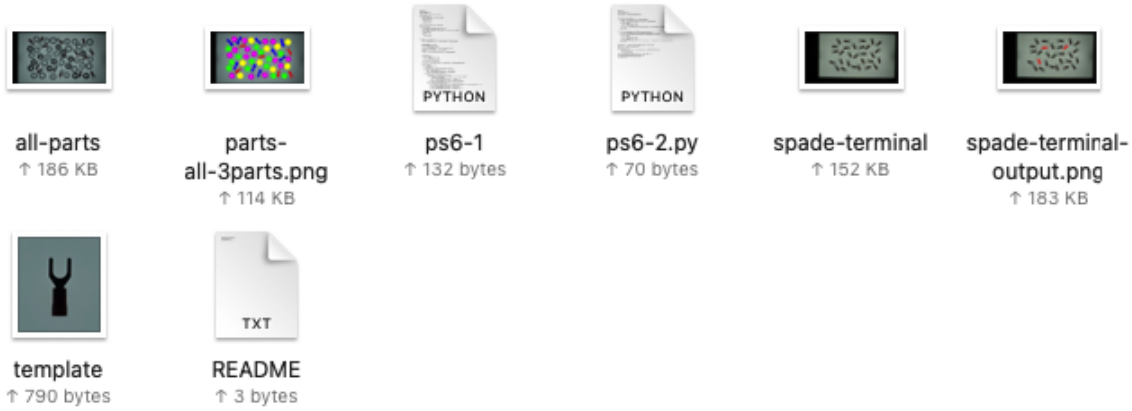


PS6-1

```python
# check size (bounding box) is square
def isSquare(siz):
    ratio = abs(siz[0] - siz[1]) / siz[0]
    #print (siz, ratio)
    if ratio < 0.1:
        return True
    else:
        return False

# chekc circle from the arc length ratio
def isCircle(cnt):
    (x,y),radius = cv2.minEnclosingCircle(cnt)
    len = cv2.arcLength(cnt, True)
    ratio = abs(len - np.pi * 2.0 * radius) / (np.pi * 2.0 * radius)
    #print(ratio)
    if ratio < 0.1:
        return True
    else:
        return False
```

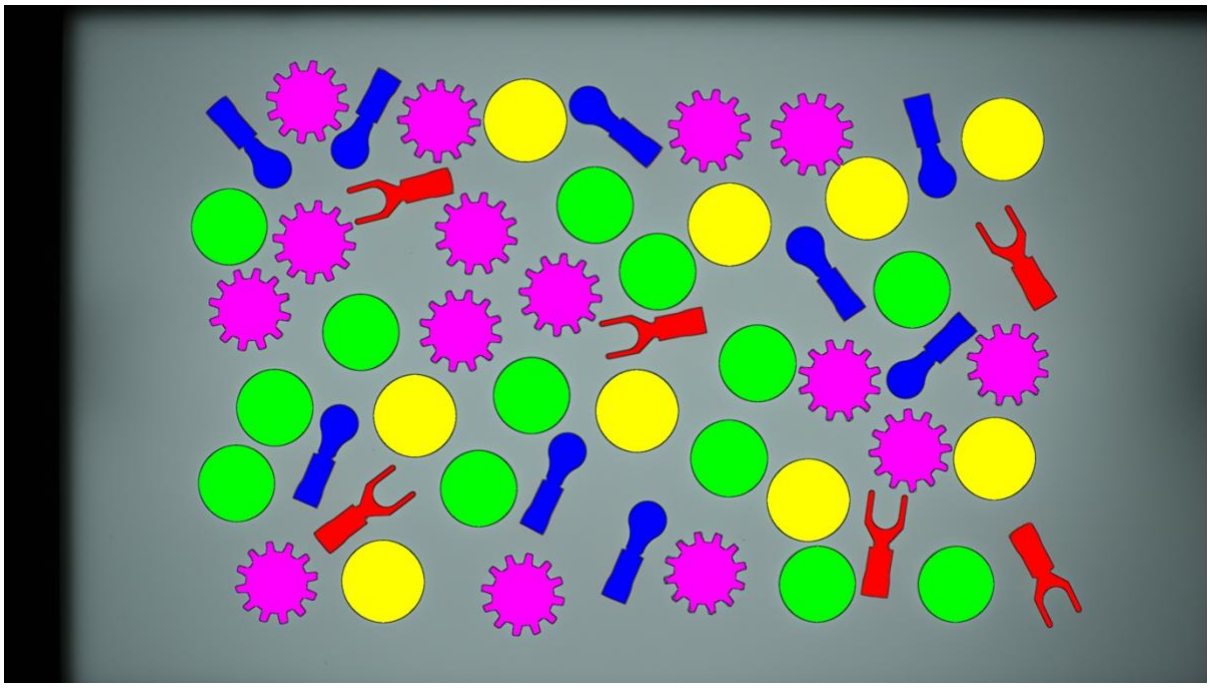
First, we wrote two functions to determine whether the contour is square or circle. Then getting into the main programme, we need to do erosion and dilation to close the balck pixel and eliminate the noise.

```python
for i in range(1):
    dst = cv2.erode(dst, None)
for i in range(3):
    dst = cv2.dilate(dst, None)
```

Then the detection program will detect the properties of the contour using hierarchy output from cv2.findContours function. The hierarchy value contains the information about last contour, next contour, parent contour and child contour. This will help us to determine the shape of the mechanical more easily. First, we create a if and elif condition, the if condition will tell the contour have no child contour so the only component with no child contour is spade. Then we can directly determine it is spade if the contour go into this block of program. The in the elif condition, it tells us the contour has child contour, the there are three possibilities: ring external lock washer and internal lock washer. To further distinguish, we find out the shape of contour itself. If it is a circle, then it maybe ring or internal lock

washer. Then we further determine the shape of child contour and finally it tells us which component it is. If the contour itself is not a circle but with child contour, then the component may be ring terminal or external lock washer. Then we used similar method mentioned above to distinguish them.

```
# find contours with hierachy
cont, hier = cv2.findContours(dst, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# each contoure
for i in range(len(cont)):
    c = cont[i]
    h = hier[0,i]
    # h[i][2],h[i][3], check child and parent contour
    if h[2] == -1 and h[3] == 0:
        # no child and parent is image outer, then spade
        img = cv2.drawContours(img, cont, i, (0,0,255),-1)
    elif h[3] == 0 and hier[0,h[2]][2] == -1:
        # with child
        if isCircle(c):# contour itself is circle
            if isCircle(cont[h[2]]):# Child contour is circle then is washer
                # double circle #washer
                img = cv2.drawContours(img, cont, i, (0,255,0),-1)
            elif not isCircle(cont[h[2]]):  # check internal lock washer
                img = cv2.drawContours(img, cont, i, (0, 255, 255), -1)
        else:# contour itself is not circle
            # 1 child and shape bounding box is not square, ring terminal
            if not isSquare(cv2.minAreaRect(c)[1]) and hier[0,h[2]][0] == -1 and hier[0,h[2]][1] == -1:
                img = cv2.drawContours(img, cont, i, (255,0, 0),-1)
            elif isCircle(cont[h[2]]):  # external lock washer
                img = cv2.drawContours(img, cont, i, (255, 0, 255), -1)
```



The result is very accurate. It successfully distinguish all the components.

PS6-2

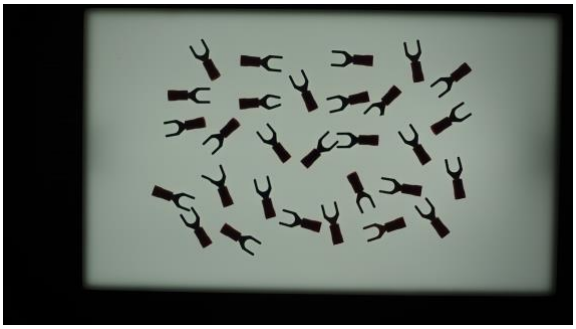
In this task we try to find the defect spade by using a template.

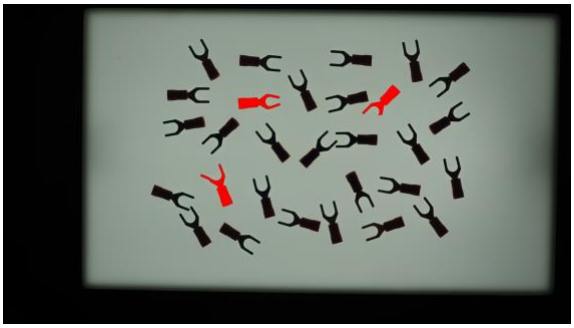
```python
def defective_contour(img):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    thr, dst = cv2.threshold(img_gray, 60, 255, cv2.THRESH_BINARY)
    for i in range(1):
        dst = cv2.erode(dst, None)
    for i in range(1):
        dst = cv2.dilate(dst, None)
    cont, hier = cv2.findContours(cv2.bitwise_not(dst), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    return cont


def complete_contour(img):  # template
    img = cv2.GaussianBlur(img, (3, 3), cv2.BORDER_DEFAULT)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to gray-scale
    thr, dst = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)  # Binary
    for i in range(1):
        dst = cv2.erode(dst, None)
    for i in range(1):
        dst = cv2.dilate(dst, None)
    cont, hier = cv2.findContours(cv2.bitwise_not(dst), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    return cont
```

First we wrote two function to extract the contour of template and every contour in the image we want to examine.

```python
spade_contour = complete_contour(template_img)  # 1 ref contour
def_contour = defective_contour(img)
def_list = []
for i in def_contour:
    diff = cv2.matchShapes(i, spade_contour[0], cv2.CONTOURS_MATCH_I2, 0)
    if diff > 2:
        def_list.append(i)
```

Then we compare the template contour with each contour one by one by using cv2.matchShape function. This function return the discrepancy magnitude measured by Hu's moment. If the magnitude exceed the threshold, we will store this contour and mark it as defective.





The result after processing is accurate.