Pipeline Pré-processamento e Detecção de Defeitos

Módulos utilizados

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
from copy import deepcopy
```

Abrindo as imagens para o teste

In [7]:
```
folders = os.listdir('resultados-pre-processamento')
sub_folders = os.listdir('resultados-pre-processamento/'+folders[0])
imgs = {}

imgs['sem'] = {}
for sub_key in sub_folders:
    imgs['sem'][sub_key] = []
    path = 'defeitos/'+sub_key
    for img in os.listdir(path):
        imgs['sem'][sub_key].append(cv2.imread(path+'/'+img))
```

1. Pré-processamento

In [2]:
```
def preprocessing(img):
    # conversao da imagem para escala de cinza
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # aplicacao de filtro gaussiano
    imgblur = cv2.GaussianBlur(img_gray, (3,3), 0)
    # aplicacao de filtro laplaciano
    imglap = cv2.Laplacian(imgblur, cv2.CV_16S, ksize=3)
    # conversao para uint8
    imgabs = cv2.convertScaleAbs(imglap)
    # equalizacao de histograma
    imgeq = cv2.equalizeHist(imgabs)
    # filtro de media
```

```
    imgblur = cv2.blur(imgeq, (5,5))
    # limiarizacao binaria
    ret,thresh1 = cv2.threshold(imgblur,120,255,cv2.THRESH_BINARY_INV)
    # opening
    kernel = np.ones((13,13),np.uint8)
    opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
    # dilatacao
    kernel_dl = np.ones((3,3),np.uint8)
    dilation = cv2.dilate(opening,kernel_dl,iterations = 1)

    return dilation
```

2. Detecção de Defeitos

In [8]:

```
def detect_threshold(img):
    output = deepcopy(img)
    jeans = cv2.blur(output, (13,13))
    _, jeans = cv2.threshold(jeans, 20, 255, cv2.THRESH_BINARY)
    edges = cv2.Canny(jeans, 50, 100)
    contours, _ = cv2.findContours(edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
    coordinates = []
    for idx, contour in enumerate(contours):
        length = cv2.arcLength(contour, closed=True)
        if length < 260: continue

        x,y,w,h = cv2.boundingRect(contour)
        coordinates.append((x+(w//2),y+(h//2)))

    return coordinates
```

3. Pré-processamento + Detecção

In [11]:

```
def pipeline_defect_detection(img):
    img_pross = preprocessing(img)
    coordinates = detect_threshold(img_pross)
    return coordinates
```

4. Testes

In [12]:
```
result = pipeline_defect_detection(imgs['sem']['tecido1'][0])
```

In [13]:
```
print(result)
```

```
[(1078, 555), (604, 549), (915, 73), (915, 73)]
```