**Methodology:-**

The individual part was divided into two subparts, the first of which was used to measure the defect and the second of which was used to identify the orientation. Defects are quantified using an area and polygonal curve analysis performed with the approxPolyDP function. The orientation detection algorithm is based on locating the contour's extreme points and performing logic on the coordinates. Figure 4.1 illustrates the overall methodology for each component, which includes the use of a webcam as the primary sensor for the vision detection system, the processing of real-time video captures, the application of filters and other segmented features for contour detection, and then structural analysis for defect measurement. Extreme co-ordinates were used to obtain orientation detection.

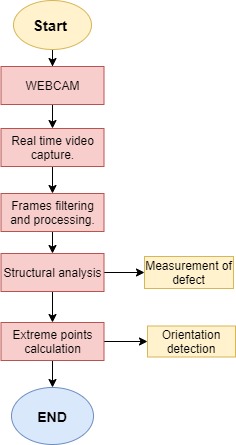


Figure 1:- Methodology

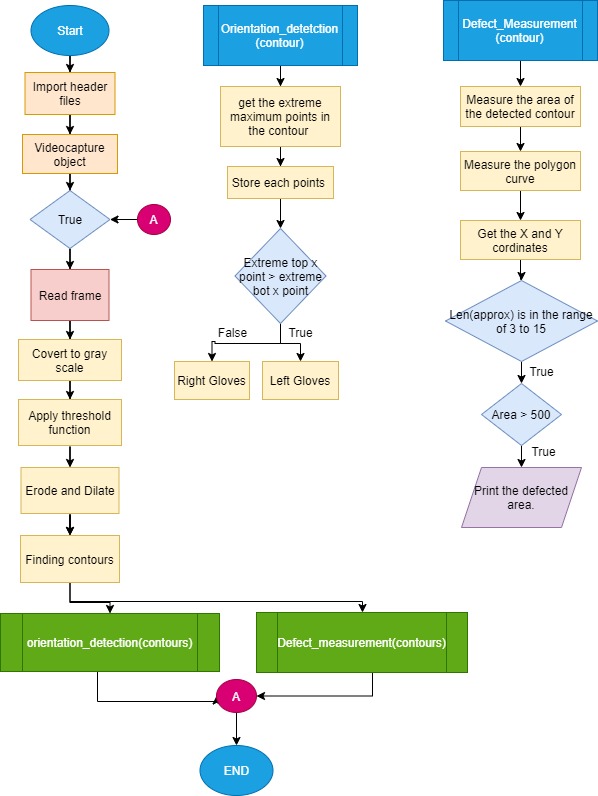


Figure 2:- Flow Chart.

The flow chart for the entire system is shown in Figure 2. After initialising the header files, the Video Capture object captures frames from the attached camera and converts each captured frame to grayscale. The transformed picture was then processed using a segmentation approach (threshold) to separate the lower layer from the gloves layer; for improved edge detection, the modified image was then submitted to the erode function and, last, to the dilate function. This processed picture was then used to determine the contours, with the resulting variable from the contour function being sent to two distinct functions for orientation detection and defect assessment.

The orientation detection user defined function received a contour as a parameter and then passed it to the max function, which returns the maximum points of the contour's edges. The output of the function is a tuple containing the contour's coordinates and each of the contour's extreme points is stored in a separate variable. The top and bottom stored variables are then compared to determine the gloves' left and right orientations.

The user-defined function for defect measuring received the contour as a parameter; the parameter was utilised to obtain the area of the detected contour. The polygonal curve function approxPolyDP was used to determine the number of sides and x and y coordinates of the contour. The nested if else condition was used to compare the length of the contour to the detected area; if the contour meets the requirement, the detected region is printed as defective and the area's measurement is displayed.

**Code and Methods Explanations and Results:-**

while True:

ret, frame = fr.read()

gray = cv.cvtColor(frame, cv.COLOR\_BGR2GRAY)

thres = cv.threshold(gray, 110, 150, cv.THRESH\_BINARY)[1]

thres = cv.erode(thres, None, iterations=1)

thres = cv.dilate(thres, None, iterations=1)

thres = cv.Canny(thres, 90, 180)

\_, contours, \_ = cv.findContours(thres, cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

Above lines of code is the processing part of the program where the image is being filtered and noise removal technique is applied along with detection of the edges of the image. The captured frame is being converted to grayscale image(Figure 3) using the COLOR\_BGR2GRAY function this was done to have a one channel image which most of the function in OpenCV uses.



Figure 3:- Gray Image.

Next line shows the thresholding of the image which was done to segment the gloves from rest of the Image, thresholding function requires four arguments which are the grayscale image, minimum threshold value for classifying the pixels, maximum threshold value and fourth argument being the type of threshold in this case it was Binary type where the segmented image is in black or white pixels(Figure 4).



Figure 4:- Threshold image

Next two lines are the erode and dilation where erode was used to remove white noise and dilation was used to add back the removed area of the object since after white noise removal the object size shrinks down(Figure 5). The erode and dilate function has three arguments where first argument is the current frame, since there was no kernel, element and anchor was passed hence the argument was NONE, the last argument was the number of times dilation and erosion will be performed which in this case was 1.



Figure 5:- Erosion and Dilation.

After the erosion and dilation was performed canny edge detection(Figure 6) was used in which several steps are built in performed for the function first being the blurring of the image in which gaussian filter is applied, next step is then finding the intensity gradient of the image in which edge gradient and direction of the pixel is found. Third step in canny edge detection is removing the unwanted pixels which may not be the edge of the detected object and the last step is the hysteresis thresholding which decides which are all edges and which are not by providing the minvalue for the thresholding and maxval for the thresholding. For function Canny there are three arguments the first being the current frame and the other two are the hysteresis thresholding parameters.

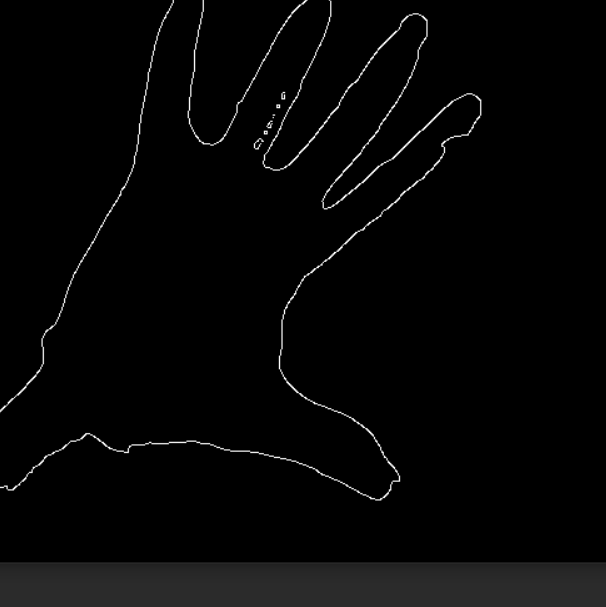


Figure 6:- Canny edge detection

Next line of code is finding the contours in the frame. Contours is basically a curve that joins all the continuous points depending on the colour and intensity.Contours are very effective in finding shapes in the image in our case it was the defect. To find contour cv.findContours was used which has three arguments, first is the image where the contour function will be performed, second is the contour retrieval mode which is used for the process of finding hierarchy inside the contours which is basically finding relationship of one contour to another inside the image, in this case RETR\_TREE was used which is often termed as the perfect mode for contours retrieval since it finds all the contour in the image and makes a list showing the relationship of one contour to another. Last argument is the Contour approximation method which is retrieving the X Y coordinates of the contours since retrieving all the coordinates of the boundary of the shape will be more processing consuming and memory consuming task, CHAIN\_APPROX\_SIMPLE method was used which saves memory by removing all the redundant points in the contour.

area = cv.contourArea(cnt)

approx = cv.approxPolyDP(cnt, 0.009 \* cv.arcLength(cnt, True), True)

x = approx.ravel()[0]

y = approx.ravel()[1]

if 15 > len(approx) > 3:

if area > 500:

print(approx)

cv.drawContours(frame, [approx], 0, (255, 255, 0), 3)

cv.putText(frame, "DEFECTED", (x, y), font, 1, (0, 255, 0))

cv.putText(frame, "AREA :-" + str(area), (x, y+25), font, 1, (0, 255, 0))

In the Above lines of codes the first line is to find the area of the current contour which was programmed with the help of inbuilt function contourArea where each contour is passed as an argument and it returns the area of the contour, next is approxPolyDp function which is used to get the approximation of curve, there are three argument in the function first being the contour second is the epsilon value which is maximum distance from contour to approximated contour. Last argument specifies whether the curve is closed or not. In the next two lines the coordinates of x and y axis returned from the approx function is stored by using the numpy ravel function which returns contiguous flattened 1dimensional array. Next lines are the conditions where the len(approx) function is used to get the total number of sides in the contour and putText function is used to put the defected area text on the frame. Figure 7 and Figure 8 shows the result of the function.



Figure 7:-Defected area for hole



Figure 8:- Defected area for hole and dirt

def orientation\_detection(count):

if len(count) == 0:

print('Connection')

else:

try:

count = imutils.grab\_contours(count)

c = max(count, key=cv.contourArea)

extLeft = tuple(c[c[:, :, 0].argmin()][0])

extRight = tuple(c[c[:, :, 0].argmax()][0])

extTop = tuple(c[c[:, :, 1].argmin()][0])

extBot = tuple(c[c[:, :, 1].argmax()][0])

if extTop[0] > extBot[0]:

cv.putText(frame, "LEFT GLOVE", (50, 50), font, 2, (255, 255, 0))

elif extTop[0] < extBot[0]:

cv.putText(frame, "RIGHT GLOVE", (50, 50), font, 2, (255, 255, 0))

except ValueError:

cv.putText(frame, "NO GLOVE DETECTED", (50, 50), font, 1, (255, 255, 0))

Above lines shows the orientation detection function which takes the countour as an argument and uses if else condition for getting the contour, try and except block was used in the begging because the function was always throwing a ValueError and the program used to exit so to tackle the issue exception was thrown where if the valueError shows which means there are no gloves detected(Figure 9). Inside the try statement imutils.grabContour function was used to get the coordinates of the contours and sort them which is then later passed as an argument to the max function which gets the maximum coordinates of the contour. Next lines shows the use of numpy function where the extLeft variable hold the smallest x coordinate value in the entire contour array which was performed by calling argmin() function similarly extRight holds the largest x coordinates. Y smallest and largest coordinates are stored in extTop and extBot variables. To fine the Left glove and right glove nested if else condition was used where if the extTop variable value is less than the extBot value then it is left glove(Figure 10), else it is Right glove(Figure 11).



Figure 9:- No Gloves Detection.

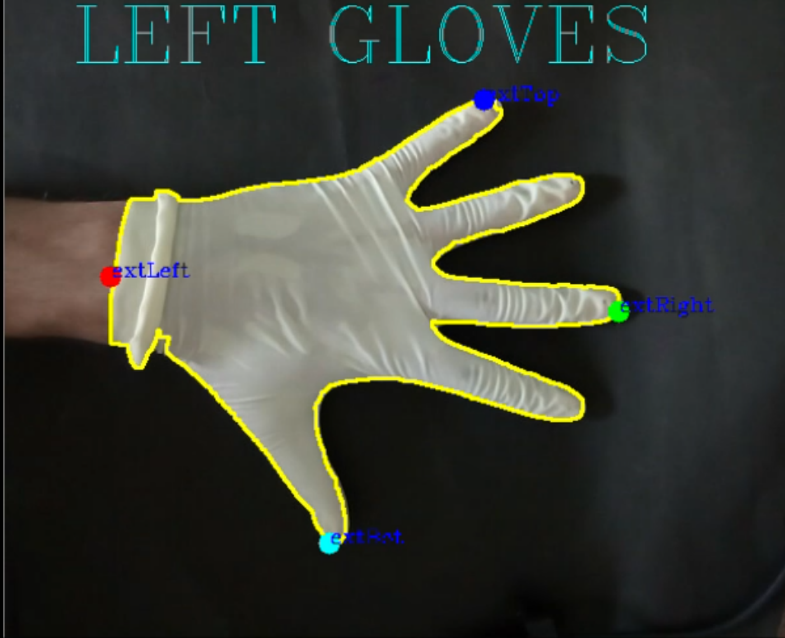


Figure 10:- Left glove orientation.



Figure 11:- Right Glove orientation.

**DISCUSSION:-**

To determine the defect, the primary objective was to detect the proper contours for the selection of the curves; since applying canny edge detection directly did not result in a correctly identified output, a few noise filters were applied, primarily erosion, and dilation was performed to fill up the shrink size of the glove. Otherwise, the contour would have been incapable of detecting the gloves and defects within the contour. The hysteresis thresholding parameters used in canny edge detection were critical in achieving a proper edge detection, which was accomplished through trial and error. When used with other techniques, the findContour function's contour approximation input produced a highly sluggish and error-filled contour, which was eventually solved by utilising the CHAIN APPROX SIMPLE method. To obtain the contour's coordinates and length, approxPolyDP was utilised, which simplified the computation for approximating the contour's side count. For the purpose of determining the glove's orientation, a graph was created to examine the gloves' finger sheaths; it was discovered that the gloves' most extreme points may be utilised to separate from one another. The sheath for the left glove's index finger is on the left side of the graph's top block, while the sheath for the right glove is on the right. Thus, the coordinates for the gloves were recorded, and it was discovered that the extreme top portion of the left glove's x point is greater than the bottom part of the right glove's x point, and vice versa for the right glove. When no gloves or objects were found in the frame, the function threw a ValueError; thus, to resolve this issue, the error was thrown using the unless statement and no gloves detected was written on the frame. The accuracy of the glove orientation detection was fairly good, since it frequently provided the proper result quickly.