Production 1

Webcam.pde

```
// Step 1. Import the video library.
import processing.video.*;


//Step 2. Declare a capture object.
Capture video;
PImage image1,image2,image3,image4,image5;
float threshold = 100f;
float threshold1 = 255f;
int videoHeight = 480;
int videoWidth = 640;

int[] area, xc, yc, averagex, averagey;
int labelCounter = 150;

// Step 5. Read from the camera when a new image is available!
void captureEvent(Capture video) {
  video.read();
}

void setup() {
  size(1920, 960);//640x480 default

  // Step 3. Initialize Capture object.
  video = new Capture(this, videoWidth, videoHeight);

  // Step 4. Start the capturing process.
  video.start();
```

```
  image1 = createImage(video.width, video.height, RGB);
  image2 = createImage(video.width, video.height, RGB);
  image3 = createImage(video.width, video.height, RGB);
  image4 = createImage(video.width, video.height, RGB);
  image5 = createImage(video.width, video.height, RGB);

  area = new int[width*height];
  xc = new int[area.length];
  yc = new int[area.length];
  averagex = new int[area.length];
  averagey = new int[area.length];
}

// Step 6. Display the image.
void draw() {
  if (video.available() == true) {
    video.read();
  }
  image(video, 0, 0);
  image(image1, videoWidth, 0);
  image(image2, videoWidth*2, 0);
  image(image3, 0, videoHeight);
  image(image4, videoWidth, videoHeight);
  image(image5, videoWidth*2, videoHeight);
  video.loadPixels();

  blur(video,image1,27);
  greyScale(image1,image2);
```

```
 image1.updatePixels();
 image2.updatePixels();
 image3.updatePixels();
 image4.updatePixels();
 image5.updatePixels();
}
```

fliters.pde

```
void skinDetection(PImage selectedImage,PImage changedImage )
{
 for (int y = 0; y < videoHeight; y++)
 {
  for (int x = 0; x < videoWidth; x++)
  {
   int loc = x+y*videoWidth;

   float R = red(selectedImage.pixels[loc]);
   float G = green(selectedImage.pixels[loc]);
   float B = blue(selectedImage.pixels[loc]);

    if (R > 95 & G >40  & B > 20  & R > B & ( R -G ) > 15)
    {
     changedImage.pixels[loc] = color(255);
    } else
    {
     changedImage.pixels[loc] = color(0);
    }
  }
 }
```

```
}

//Median
void median (PImage selectedImage,PImage changedImage )
{
  for (int y = 1; y < videoHeight -1; y++) {
    for (int x = 1; x < videoWidth -1; x++) {
      float[] list = new float[9];
      int kernelCounter = 0;



      for (int ky = -1; ky <= 1; ky++) {
        for (int kx = -1; kx <= 1; kx++) {
          int pos = (y + ky)*videoWidth + (x + kx);
          list[kernelCounter] = brightness(selectedImage.pixels[pos]);
          kernelCounter++;
        }
      }
      // printArray(list);
      list = sort(list);
      changedImage.pixels[y*videoWidth + x] = color(list[5]); // take median value as value
    }
  }
}

//dilation
void dilation(PImage selectedImage,PImage changedImage,int kernels)
{
  for (int y = kernels; y < videoHeight-kernels; y++)
  { // Skip top and bottom edges
```

```
   for (int x = kernels; x < videoWidth-kernels; x++)
   { // Skip left and right edges
     float sum = 0; // Kernel sum for this pixel
     for (int ky = kernels*-1; ky <= kernels; ky++)
     {
       for (int kx = kernels*-1; kx <= kernels; kx++)
       {
         // Calculate the adjacent pixel for this kernel point
         int pos = (y + ky)*videoWidth + (x + kx);
         // Multiply adjacent pixels based on the kernel values
         sum += brightness(selectedImage.pixels[pos])/255;
       }
     }
     if (sum >= 1)
     {
       changedImage.pixels[y*videoWidth + x] = color(255);
     } else
     {
       changedImage.pixels[y*videoWidth + x] = color(0);
     }
   }
  }
}

//Erosion
void erosion(PImage selectedImage,PImage changedImage,int kernels)
{
  for (int y = kernels; y < videoHeight-kernels; y++) { // Skip top and bottom edges
    for (int x = kernels; x < videoWidth-kernels; x++) { // Skip left and right edges
      float sum = 0; // Kernel sum for this pixel
```

```
    for (int ky = kernels*-1; ky <= kernels; ky++) {
      for (int kx = kernels*-1; kx <= kernels; kx++) {
        // Calculate the adjacent pixel for this kernel point
        int pos = (y + ky)*videoWidth + (x + kx);
        // Multiply adjacent pixels based on the kernel values
        sum += brightness(selectedImage.pixels[pos])/255;
      }
    }
    if (sum == 25)
    {
      changedImage.pixels[y*videoWidth + x] = color(255, 255, 255);
    } else
    {
      changedImage.pixels[y*videoWidth + x] = color(0, 0, 0);
    }
   }
  }
}

void blur(PImage selectedImage,PImage changedImage,float blurness)
{
  float v = 1.0 / blurness;
  float[][] kernel = {{ v, v, v },
    { v, v, v },
    { v, v, v }};
  // Loop through every pixel in the image
  for (int y = 1; y < videoHeight-1; y++) {   // Skip top and bottom edges
    for (int x = 1; x < videoWidth-1; x++) {  // Skip left and right edges
      float sum = 0; // Kernel sum for this pixel
      for (int ky = -1; ky <= 1; ky++) {
```

```
      for (int kx = -1; kx <= 1; kx++) {
        // Calculate the adjacent pixel for this kernel point
        int pos = (y + ky)*videoWidth + (x + kx);
        // Image is grayscale, red/green/blue are identical
        float val = red(selectedImage.pixels[pos]);
        // Multiply adjacent pixels based on the kernel values
        sum += kernel[ky+1][kx+1] * val;
      }
    }
    // For this pixel in the new image, set the gray value
    // based on the sum from the kernel
    changedImage.pixels[y*videoWidth + x] = color(sum);
   }
  }
}

//greyscale filter
void greyScale(PImage selectedImage,PImage changedImage)
{
  for (int x =0; x<videoWidth; x++)
  {
    for (int y =0; y<videoHeight; y++)
    {
      int loc = x+y*videoWidth;

      float R = red(selectedImage.pixels[loc]);
      float G = green(selectedImage.pixels[loc]);
      float B = blue(selectedImage.pixels[loc]);
      float average = (R+G+B)/3;
      changedImage.pixels[loc] = color(average);
```

```
    }
  }
}
```

blob.pde

```
void grassfire(int x, int y,PImage blobedImage)
{
  blobedImage.pixels[y*videoWidth+x] = labelCounter;
  area[labelCounter]++;
  xc[labelCounter]+=x;
  yc[labelCounter]+=y;
  int negativY = y-1;
  int positivY = y+1;

  if (x+1< blobedImage.width &&  blobedImage.pixels[y*videoWidth+x+1] == color(255))
  {
    grassfire(x+1, y,blobedImage);
  }
  if (y+1< blobedImage.height &&  blobedImage.pixels[positivY*videoWidth+x] == color(255))
  {
    grassfire(x, y+1,blobedImage);
  }
  if (x-1>=0 && blobedImage.pixels[y*videoWidth+x-1] == color(255))
  {
    grassfire(x-1, y,blobedImage);
  }
  if (y-1>=0 &&  blobedImage.pixels[negativY*videoWidth+x] == color(255))
  {
    grassfire(x, y-1,blobedImage);
  }
}
```