Technical Documentation

Table 1. Document Change Control

Version	Date	Authors	Summary of Changes
1.0	14/10/2018	Shenal Samarasinghe Dineth Gunawardena	Initial draft created
1.1	14/10/2018	Shenal Samarasinghe Dineth Gunawardena Ayub Khan	Data was added to certain sections
1.2	28/10/2018	Lyndon	Updating makeHandMask, constants, getHandContour,getObjectCentre,getTraceCoordinates, calibrateHSV, log

FrameTrace.js

The of the this class has been deprecated and transferred to the client side

base64toMat(base64)

```
function base64toMat(base64) {
    var split = base64.split(",")[1];
    return cv.imdecode(Buffer.from(split, "base64"));
}
```

Definition: Converts a base64 value to a Mat value
Parameter: Base64
Return: Mat

Gesture.js

This file has the code that is used to extract the hand from the frame

Variable

lH- Low Hue
lS- Low Saturation
lV- Low Value
uH- Upper Hue
uS- Upper Saturation
uV- Upper Value
hueVariance- Acceptable range of Hue
satVariance- Acceptable range of Saturation
valVariance- Acceptable range of Value

Constant

transparentPixel

grabHand(handFrame)

Definition: Extracts hands from frame using makeHandMask()
Parameter: *Mat* handFrame
Return: *Mat* with the original frame with inverted hand mask

```
function grabHand(handFrame) {
      let src = handFrame;
      src = src.cvtColor(cv.COLOR_RGB2RGBA);
      const handMask = makeHandMask(src);

      for (let i = 0; i < handMask.rows; i++) {
           for (let j = 0; j < handMask.cols; j++) {
                if (handMask.at(i, j) == 0) {
                     src.set(i, j, transparentPixel);
                }
           }
      }
      return src;
}
```

makeHandMask(img)

Definition: Create a Mat Frame that consists of bits with values of 0s or 1s to signify the threshold of the color that is selected i.e. if the raw image pixel is the selected color, return a 1 else if not the color return a 0. This function generates the mask for one frame to be used for hand extraction.
Parameter: Mat of Vec3 RGB values
Return: Mat of Binary values

```
function makeHandMask(img) {
      // Denoising the color
      for(var i = 0; i < 2; i++){
         img = img.blur(new cv.Size(10,10));
      }
      // filter by skin color
      const imgHLS = img.cvtColor(cv.COLOR_BGR2HLS);
      var rangeMask = imgHLS.inRange(new cv.Vec(lH,lS,lV), new cv.Vec(uH, uS,uV));
      //close gaps
     rangeMask = rangeMask.morphologyEx(kernel,cv.MORPH_OPEN);
     rangeMask = rangeMask.morphologyEx(kernelClose,cv.MORPH_CLOSE);
     //rangeMask = rangeMask.dilate(new cv.Mat([[1, 1],[1, 1]], cv.CV_8U), new cv.Vec(-1, -1), 2);
      // remove noise
      var blurred = rangeMask.blur(new cv.Size(10, 10));
      const thresholded = blurred.threshold(75, 255, cv.THRESH_BINARY);
      return thresholded;
};
```

base64toMat(base64)

Definition: Splits base64 string and converts it to Mat
Parameter: *String* base64 string
Return: *Mat*

```
function base64toMat(base64) {
    var split = base64.split(",")[1];
    return cv.imdecode(Buffer.from(split, "base64"));
}
```

calibrateHSV(hsv)

Definition: Calibrate upper and lower HSV values
Parameter: Float [3]
Return: void

```
function calibrateHSV(hsv){
    //calibrate lower hue value
    if(hsv[0] >= hueVariance)
    {
        lH = (hsv[0] - hueVariance)*180;

    }
    else // hsv[0] < hueVariance
    {
        lH = 0.00;
    }
    //calibrate lower saturation value
    if(hsv[1] >= satVariance)
    {
        lS = (hsv[1] -satVariance)*255;
    }
    else //hsv[1] < satVariance
    {
        lS = 0.00;
    }
    //calibrate lower value value
    if(hsv[2] >= valVariance)
    {
        lV = (hsv[2] - valVariance)*255;
    }
    else //hsv[2] < valVariance
    {
        lV = 0.00;
    }
    //calibrate upper hue value
    if(hsv[0] <= (1 - hueVariance))
    {
        uH = (hsv[0] + hueVariance)*180;
    }
```

GetTraceCoordinates.js

This file has the code needed to get the coordinates of the center of the pointer

Variables

lH- Low Hue
lS- Low Saturation
lV- Low Value
uH- Upper Hue
uS- Upper Saturation
uV- Upper Value
hueVariance- Acceptable range of Hue
satVariance- Acceptable range of Saturation
valVariance- Acceptable range of Value

Constants

kernel = Morphology opening - Deprecated no longer used (used for dilation and erosion)
kernelClose = Morphology closing - Deprecated no longer used (used for dilation and erosion)

makeHandMask(img)

```
const makeHandMask = (img) => {
    // Denoising the color
    for(var i = 0; i < 2; i++){
        img = img.blur(new cv.Size(2,2));
    }
    // filter by skin color
    const imgHLS = img.cvtColor(cv.COLOR_BGR2HLS);
    var rangeMask = imgHLS.inRange(new cv.Vec(lH, lS,lV), new cv.Vec(uH, uS,uV));
    //close gaps
    rangeMask = rangeMask.morphologyEx(kernel,cv.MORPH_OPEN);
    rangeMask = rangeMask.morphologyEx(kernelClose,cv.MORPH_CLOSE);
    // remove noise
    var blurred = rangeMask.blur(new cv.Size(10, 10));
    const thresholded = blurred.threshold(75, 255, cv.THRESH_BINARY);
    cv.imshow('getTraceCoordinates',thresholded);
    cv.waitKey(2);
    return thresholded;
};
```

Definition: Create a Mat Frame that consists of bits with values of 0s or 1s to signify the threshold of the color that is selected i.e. if the raw image pixel is the selected color, return a 1

else if not the color return a 0. This function generates the mask for one frame to be used for hand extraction.
Parameter: Mat of Vec3 RGB values
Return: Mat of Binary values

getHandContour (handMask)

```
const getHandContour = (handMask) => {
    const mode = cv.RETR_EXTERNAL;
    const method = cv.CHAIN_APPROX_SIMPLE;
    const contours = handMask.findContours(mode, method);
    // largest contour
    return contours.sort((c0, c1) => c1.area - c0.area)[0];
};
```

Definition: Uses the hand mask to get the hand contour
Parameter: Mat of Binary values
Return: Contour[]

getObjectCenter(contour)

```
const getObjectCenter = (contour) => {
    // get hull indices and hull points
    const hullIndices = contour.convexHullIndices();
    const contourPoints = contour.getPoints();
    const hullPointsWithIdx = hullIndices.map(idx => ({
        pt: contourPoints[idx],
        contourIdx: idx
    }));
    const hullPoints = hullPointsWithIdx.map(ptWithIdx => ptWithIdx.pt);

  // get the x and y values of the center of the object
    var xpt = 0;        //contains the x cordinates
    var ypt = 0     // contains the y cordinates
    for (var i=0;i<hullPoints.length;i++) {
      xpt = xpt+(hullPoints[i].x)
      ypt = ypt+(hullPoints[i].y)
    }
    xpt =  (xpt/(hullPoints.length)).toFixed(0)
    ypt = (ypt/(hullPoints.length)).toFixed(0)
    console.log("getObjectCenter: xpt - ", xpt, ", ypt - ",ypt);
    return [xpt, ypt]; // returns an array with the x and y cordinates
    // return new cv.Point(xpt, ypt); // returns an array with the x and y cordinates
};
```

Definition: It takes the hull points from the contour and uses the hull points to calculate the center points of the pointer.
Parameter: Contour[]
Return: Point2d

GetTraceCoordinates(frame)

```
const GetTraceCoordinates = (frame) => {
    //console.log("GetTraceCoordinates - frame: ", frame);
    //console.log("frame type: ", typeof(frame));
    // const resizedImg = frame.resizeToMax(640);
    const handMask = makeHandMask(frame);
    const handContour = getHandContour(handMask);
    //cv.imshow('handContour',handContour);
    if (!handContour) {
        return null;
    }
    const objectCenter = getObjectCenter(handContour);
    console.log("GetTraceCoordinates - objectCenter", objectCenter);
    return objectCenter;
}
```

Definition: It acts as the main function,
Parameter: Mat (Raw image)
Return: Point2d

calibrateHSV(hsv)

```
function calibrateHSV(hsv) {
    //calibrate lower hue value
    if (hsv[0] >= hueVariance) {
        lH = (hsv[0] - hueVariance) * 180;

    }
    else // hsv[0] < hueVariance
    {
        lH = 0.00;
    }
    //calibrate lower saturation value
    if (hsv[1] >= satVariance) {
        lS = (hsv[1] - satVariance) * 255;
    }
    else //hsv[1] < satVariance
    {
        lS = 0.00;
    }
    //calibrate lower value value
    if (hsv[2] >= valVariance) {
        lV = (hsv[2] - valVariance) * 255;
    }
    else //hsv[2] < valVariance
    {
        lV = 0.00;
    }
    //calibrate upper hue value
    if (hsv[0] <= (1 - hueVariance)) {
        uH = (hsv[0] + hueVariance) * 180;
    }
    else //hsv[0] > 1 - hueVariance
    {
        uH = 1 * 180;
    }
    //calibrate upper saturation value
    if (hsv[1] <= 1 - satVariance) {
        uS = (hsv[1] + satVariance) * 255;
    }
```

```
    else  //hsv[1] > 1 - satVariance
    {
        uS = 1 * 255;
    }
    //calibrate upper value value
    if (hsv[2] <= 1 - valVariance) {
        uV = (hsv[2] + valVariance) * 255;
    }
    else //hsv[2] > 1 - valVariance
    {
        uV = 1 * 255;
    }

    console.log("lH - ", lH);
    console.log("lS - ", lS);
    console.log("lv - ", lV);
    console.log("uH - ", uH);
    console.log("uS - ", uS);
    console.log("uV - ", uV);
}
```

Definition: It calibrates the HSV values
Parameter:float[3]
Return: Void

Utils.js

webRTC.js

log()

```
function log() {
    var array = ['Message from server:'];
    array.push.apply(array, arguments);
    socket.emit('log', array);
}
```

Definition: emits message from server
Parameter: void
Return: void