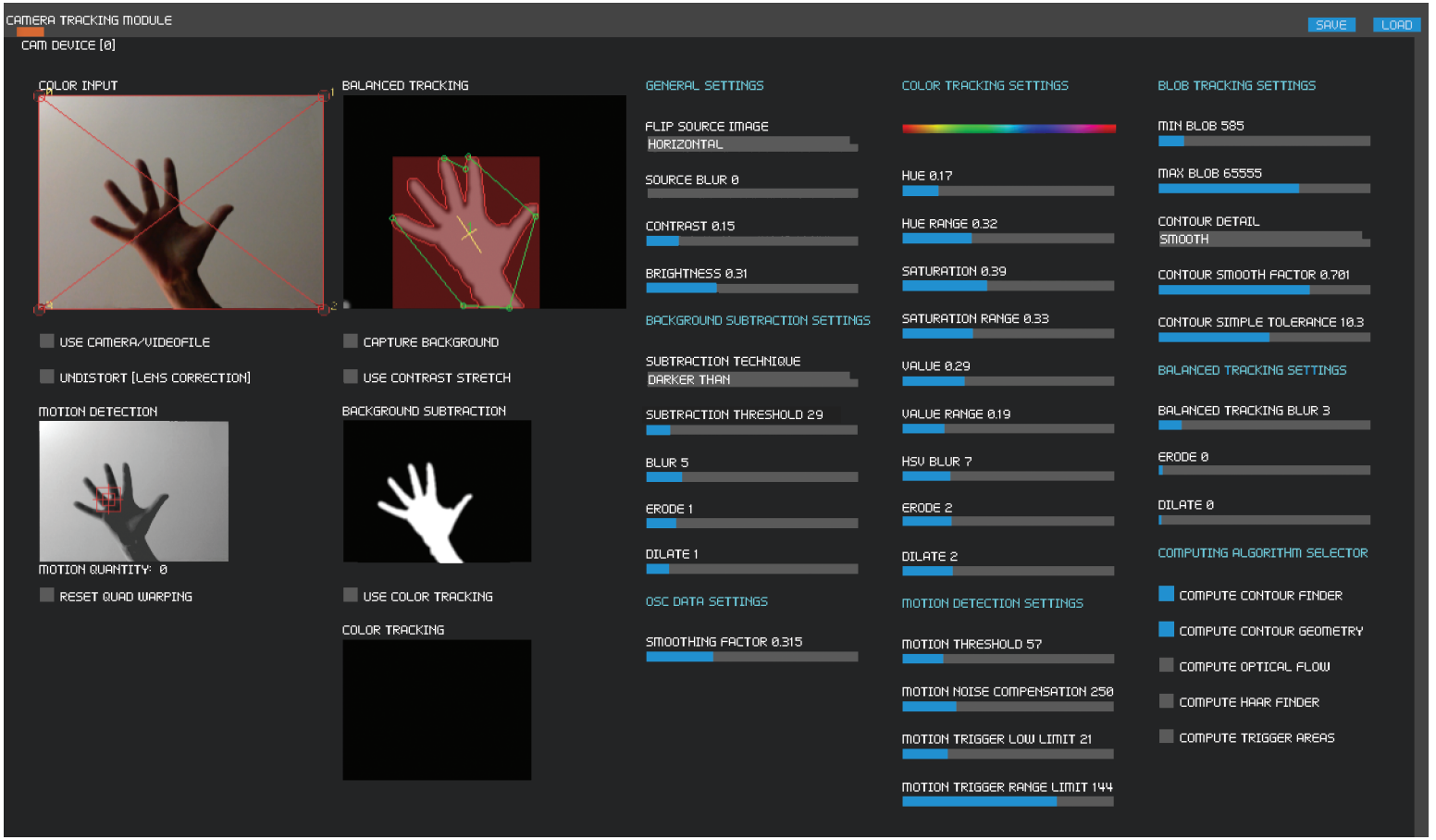


Panel Tracking Video

9. Tracking video

En GAmuza se pueden hacer scripts para tracking video utilizando las clases y métodos propios de opneFrameworks: `ofVideoGrabber()` u `ofxKinect()` para detectar la fuente y los addons de openCV `ofxCvColorImage()`, `ofxCvContourFinder()`, `ofxCvFloatImage()`, `ofxCvGrayscaleImage()`, `ofxCvShortImage()`, `ofxCvHaarFinder()`, `ofxOpticalFlowLK()`, o utilizar el panel de Computer visión con interface GUI que ha sido preconfigurado principalmente para facilitar este proceso a aquellos que no sean expertos en programación.

A diferencia de los módulos, los paneles se generan por programación en la ventana de salida usando las clases de GAmuza `gaCameraTracking()` o `gaKinectTracking()`, se pueden activar desde los ejemplos Computer Vision/camTrackingPanel para cámaras de video o Computer Vision/kinectTrackingPanel cuando se trabaja con el sensor kinect.

9.1. Panel Tracking: por cámara video

En este apartado se describe la interface del panel de Tracking para cámaras de video y los métodos de la clase de GAmuza `gaCameraTracking()`, explicándolo a partir del código que genera su carga.

```
/*
 GAmuza 043                E-9-1
 ---------------------------------
 Panel de Tracking  video
 creado por n3m3da | www.d3cod3.org
*/
camPanel = gaCameraTracking()       // asocia variable global a la clase de GAmuza
drawGUI = true
camID = 0

captureWidth = 320                  // tamaño captura de video
captureHeight = 240
cam = ofTexture()                   // textura para acoger la imagen video

function setup()                    // para guardar el archivo de configuración XML
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(camID,captureWidth,captureHeight)     // inicializa panel
                                    // inicializa textura openGL para imagen video
   cam:allocate(captureWidth,captureHeight,GL_RGB)
end
```

```
function update()
   cam = camPanel:getCameraTextureMod() // la textura actualiza los frames de video
   camPanel:update()                    // actualiza el panel de tracking
end

function draw()
   gaBackground(0.1,1.0)
      // para mostrar la imagen en directo de la cámara a pantalla completa
   ofSetColor(255)
   scaleH = OUTPUT_H
   scaleW = scaleH* captureWidth / captureHeight
      cam:draw(OUTPUT_W/2 - scaleW/2,0, scaleW,scaleH)

      ofSetColor(255)
      if drawGUI then
         camPanel:draw()            // muestra el panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI         // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

Como en otros trabajos con clases, inicialmente se asocia una variable global, `camPanel`, a la clase `gaCameraTracking()`; la segunda variable, `drawGUI`, es booleana y permitirá visualizar/ocultar el panel en la ventana de salida; `camID` asigna el ID de la cámara y las dos siguientes variables, `captureWidth` y `captureHeight`, definen las dimensiones del video capturado; la última variable, `cam`, está vinculada a la clase `ofTexture()` como soporte de la imagen capturada por la cámara de video

En el bloque `setup()` se inicializa el panel con el método `setGuiSettingsFile` cuyo parámetro indica la ruta y nombre del archivo XML que guardará la configuración del panel. Al activar el script aparece el panel en la ventana de salida, tras realizar los ajustes que requiera el proyecto, debe guardarse esa configuración clicando el botón "save" situado en el vértice superior derecho del panel. Al guardarlo se genera el archivo `camTrackingSettings.xml` en la carpeta `data` del script. El método `setup` tiene como parámetros el ID de la cámara y el tamaño del frame. Después, con el método `allocate` de la clase `ofTexture()` se asignan el tamaño y tipo de color de openGL del frame de vídeo.

En el bloque `update()`, se identifica la textura con el panel a través del método `getCameraTextureMod()` [devuelve la captura de la cámara con el eventual mirror aplicado en horizontal/vertical] o con `getCameraTexture()` [devuelve la captura desde la cámara en "raw"], con cualquiera de ellos se relaciona la clase de openFrameworks con la de GAmuza, y después con el método `update` se actualiza, para cada frame, la información de la imagen capturada.

En el bloque `draw()`, se dibuja la imagen en directo de la cámara adaptada a pantalla completa, y después, para que quede encima de la imagen, se dibuja el panel cuando la variable booleana `drawGUI` es verdadera.

En el bloque `keyReleased()`, se establece una condicional para que al teclear la letra "g" la variable `drawGUI` pase de verdadera a falsa y viceversa, construyendo un switch para ver u ocultar el panel.

Los bloques `mouseDragged()`, `mousePressed()` y `mouseReleased()` permiten manipular con el ratón los sliders y botones de la interface del panel.

9.1.1. Descripción panel

El panel Tracking video permite analizar características visuales del campo que registra la cámara, recoger determinados datos como la luminosidad, color o movimiento, seleccionar y/o ajustar esos datos, para después generar con ellos otra situación que responda en tiempo real a sus cambios, facilitando la aplicación de los algoritmos para tracking video al regular con sliders, menús y botones las condiciones básicas de entrada de datos al sistema.

GAmuza recoge el reconocimiento que el ordenador ha hecho de la cámara o cámaras conectadas, asignando un ID a cada una de ellas, empezando por el `0`, es decir, si solo hay una cámara conectada su id es `0`, si hay dos la primera es `0` y la segunda `1` y así sucesivamente.

Las interface del panel está distribuida en función de las distintas técnicas básicas para tracking video:

Blob detection, reconocimiento de regiones o áreas. Puede detectar los blobs por Background subtraction o por Color tracking. Para el reconocimiento del contorno se debe activar el algoritmo Compute Contour Finder y para la geometría del contorno hay que añadir el algoritmo Compute Contour geometry

Motion detection, reconoce la cantidad de movimiento y localiza el punto medio de la masa, siempre está activo aunque ningún algoritmo haya sido seleccionado

Haar Finder, para reconocimiento de partes del cuerpo humano, el algoritmo Compute Haar Finder debe estar seleccionado

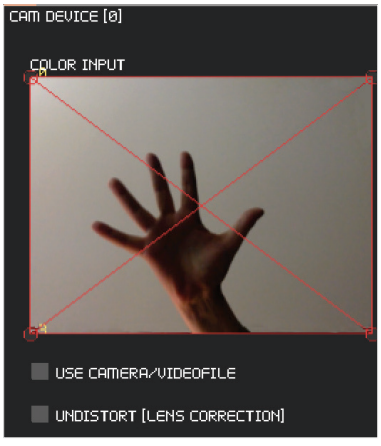
Optical Flow, dirección del movimiento representado vectorialmente, debe estar seleccionado el algoritmo Compute Optical Flow

Trigger Areas, son nueve áreas de activación ajustables en tamaño y posición desde el módulo Computer vision. Se debe activar el algoritmo Compute Trigger Areas

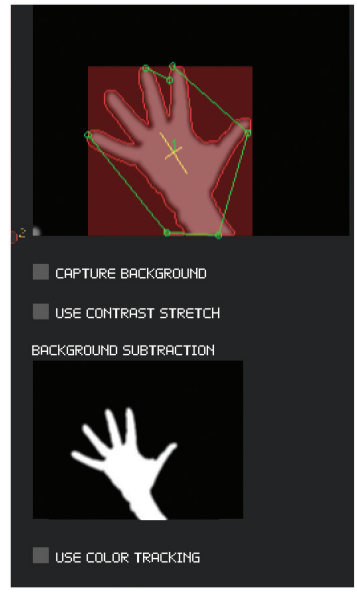
Análisis detallado de la interface

El monitor de entrada de la señal video, Color input, permite reducir el área a trackear, este *crop* se ajusta arrastrando con el ratón los puntos resaltados de los vértices. Los demás monitores muestran solo la zona seleccionada. El crop se resetea con Reset Quad Warping.

USE CAMERA/VIDEOFILE. Activa/desactiva la posibilidad de trabajar con un archivo de video pregrabado (video file) o con la imagen capturada por la cámara en directo.



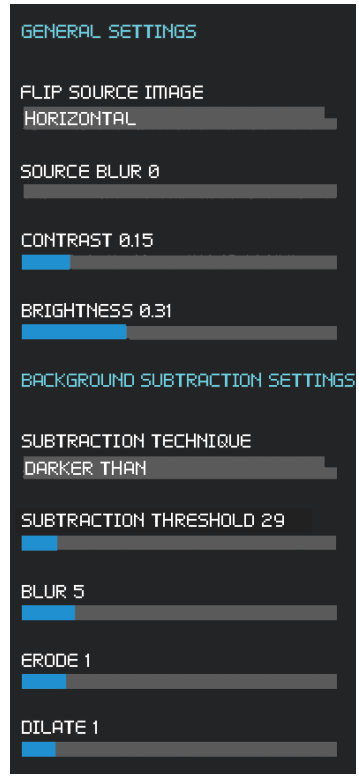
El monitor **Balanced Tracking** muestra el resultado de todos los algoritmos de tracking seleccionados y ajustados con los sliders, menús y botones del bloque derecho.



Capture Background, al clicar el botón se captura un frame de la imagen video que debe corresponder con el fondo. Esta imagen se toma como referencia para comparar los cambios que se producen entre el valor de sus píxeles y el de los frames de la imagen a trackear, aplicando el algoritmo Background Subtraction. Es el primer nivel para el cálculo de Tracking, y siempre está activo. Según las condiciones de iluminación, puede mejorarse la imagen activando **Use Contrast Stretch**: incrementa el rango de los valores más intensos de la imagen en escala de grises.

El resultado de **Background subtraction** se ve en el monitor inferior.

GENERAL SETTINGS



Flip Source Image voltea la imagen-video que proviene de la cámara, su ajuste depende de las condiciones de exhibición:

- **Off**: la imagen se percibe sin ningún cambio
- **Horizontal**: se selecciona cuando los movimientos ante la imagen responden al efecto de estar frente a un espejo.
- **Vertical**: imagen video cabeza abajo.
- **Horizontal + Vertical**: voltea en los dos sentidos.

Source Blur: desenfoque de la imagen de entrada de video.
Contrast: contraste.
Brightness: brillo.

BACKGROUND SUBTRACTION SETTINGS

Subtraction Technique, seleccionar entre:

- **Color ABS**: color absoluto
- **B&W ABS**: blanco y negro absoluto
- **Lighter than**. Cuando la figura a trackear es más clara que el fondo.
- **Darken than**. Cuando La figura a trackear es más oscura que el fondo.

Subtraction Threshold, ajusta el umbral de detección. Cuanto menor es el valor más sensible. Si se trabaja con Tracking Color se pone el valor máximo para anular los datos de substracción del fondo

Blur, desenfoca la imagen procesada.

Erode, reduce las superficies detectadas.

Dilate, expande las superficies detectadas

OSC DATA SETTINGS

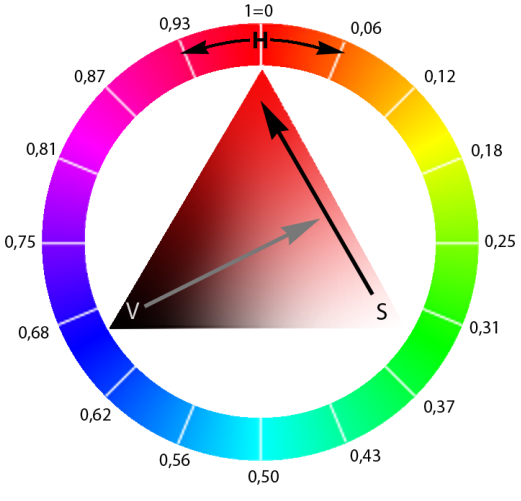
Smoothing factor: ajusta el rango de suavizado.

USE COLOR TRACKING

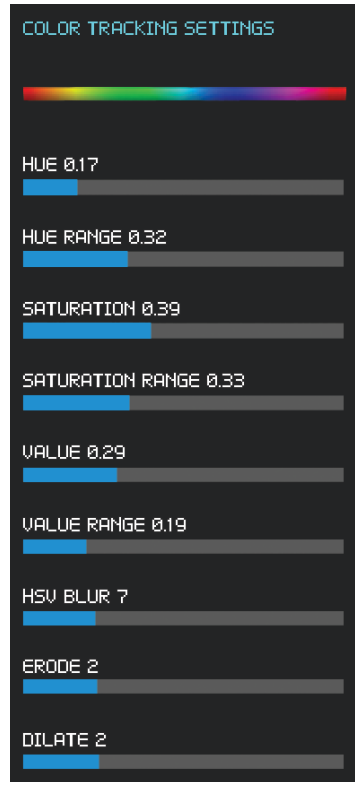
Color Tracking es un algoritmo de reconocimiento del color. Requiere condiciones de iluminación estables y la elección de un color de seguimiento muy diferenciado del contexto. **Use Color Tracking**, activa/desactiva su uso y los resultados se ven en el monitor inferior. Como se ha mencionado antes, cuando se trabaja con **COLOR TRACKING** debe subirse al máximo el valor del slider **Subtraction Threshold** para anular los datos del **BACKGROUND SUBTRACTION.**

COLOR TRACKING SETTINGS

La configuración de este bloque solo se ajusta si está activado **Use Color Tracking**. El Tracking Color transforma el espacio de color RGB al sistema HSV, porque asigna los valores del tono del color de una forma más estable. HSV identifica cada tono con un número que va de 0ª a 360ª según su posición en el círculo cromático, pero en GAmuza el rango va de `0.0` a `1.0`, quedando su equivalencia tal como muestra la imagen.



Las opciones de ajuste son:



Hue, selecciona el tono a trackear según la escala de valores de GAmuza entre `0.0` a `1.0`. La franja de color sirve de orientación.

Hue Range, amplía el rango del tono seleccionado.

Saturation, asigna el nivel de saturación del tono (hue) seleccionado. El valor `0.0` es igual a blanco, `1.0` es el tono puro, dependiendo del nivel de luminosidad (value).

Saturation Range, amplia el rango de saturación seleccionado.

Value, regula el nivel de luminosidad (value) del tono. El valor `0.0` es igual a negro, `1.0` es igual al tono puro, dependiendo del valor de saturación.

Value Range, amplía el rango de luminosidad seleccionado.

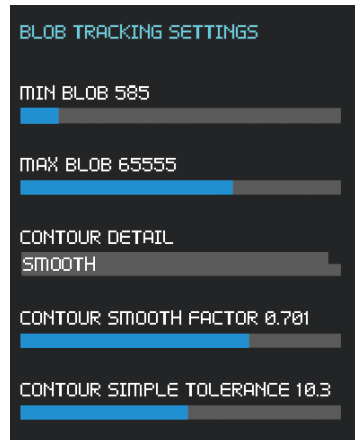
HSV Blur, filtro de desenfoque de la imagen en HSV.

Erode, reduce las superficies detectadas.

Dilate, expande las superficies detectadas.

BLOB TRACKING SETTINGS

Los blob son las zonas independientes que el tracking detecta según los algoritmos seleccionados, sus opciones de configuración son:



Min Blob, tamaño mínimo de los blobs en píxeles.

Max Blob, tamaño máximo.

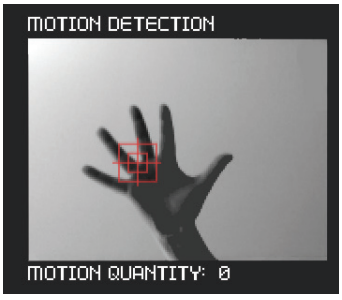
Contour Detail, el detalle del contorno se puede ajusta en:

- Raw, todo el perímetro tal cual es
- Smooth, suavizado
- Simple, adaptado a formas geométricas simples

CONTOUR SMOOTH FACTOR, si se ha seleccionado smooth, ajusta el nivel de suavizado

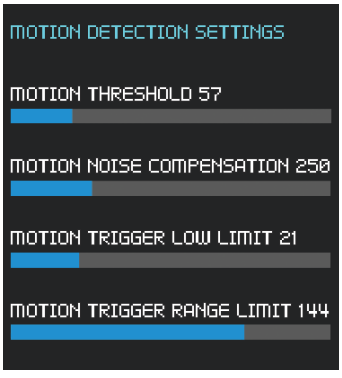
CONTOUR SIMPLE TOLERANCE, si se ha seleccionado simple, ajusta el rango de vértices que articulan el contorno.

MOTION DETECTION SETTINGS



El algoritmo para detección de movimiento está siempre activo.

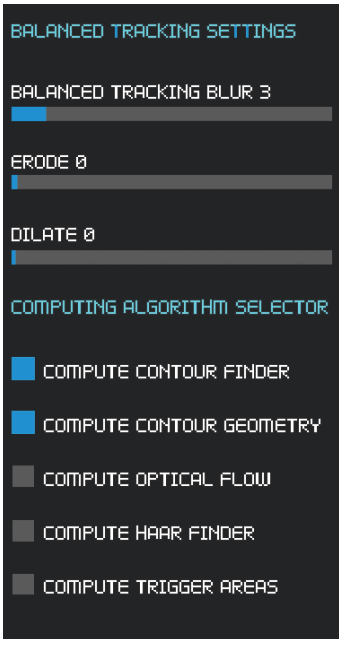
El monitor **MOTION DETECTION** muestra el cálculo de la cantidad de movimiento que se registra en la imagen y señala la posición media de ese movimiento. Los ajustes para su configuración son:



- **Motion Threshold**, regula la sensibilidad del sistema para activar la detección, si es poco sensible se debe ampliar el umbral (Threshold)
- **Motion Noise Compensation**, compensa el ruido de la señal de imagen video
- **Motion Trigger Low Limit**, nivel más bajo para activar la detección de movimiento
- **Motion Trigger Range Limit**, limita los niveles de movimiento, como si bajara la sensibilidad de detección.

BALANCED TRACKING SETTINGS

Vuelve a ajustar, en conjunto, los valores dados en cada sección.



- **Balanced Tracking Blur**, ajusta nivel de desenfoque global
- **Erode**, reduce las superficies detectadas
- **Dilate**, expande las superficies detectadas

COMPUTING ALGORITHM SELECTOR

En este bloque se seleccionan otros algoritmos específicos que se pueden aplicar, además de los mencionados hasta ahora.

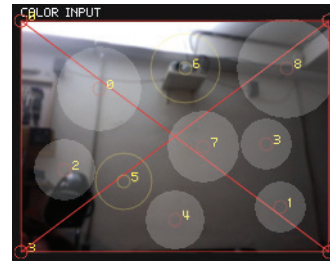
Compute Contour Finder, para procesar el contorno de las zonas detectadas. Sus valores se ajustan con **Blob Tracking Settings**.

Compute Contour geometry, para procesar el contorno de las zonas detectadas con formas geométricas.

Compute Optical Flow, calcula el flujo óptico del movimiento y lo describe con vectores que indican la dirección e intensidad.

Compute Haar Finder, activa el algoritmo para detectar partes del cuerpo humano. Hay distintos algoritmos posibles. Por defecto, está seleccionado **frontalface_alt** (Ver página 201).

Compute Trigger Areas. Activa la posibilidad de utilizar 9 zonas circulares que tienen asignado un ID. Su posición y tamaño pueden ajustarse con el ratón; con ellas se pueden determinar las áreas en las que debe responder el tracking utilizando funciones específicas de GAmuza (ver página 229)

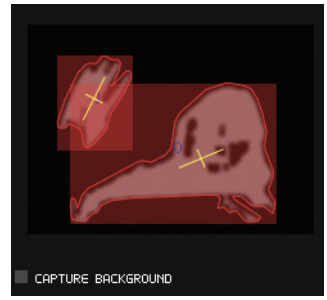


Si hay varias cámaras conectadas al ordenador, se pueden abrir distintas interfaces del panel para cada una de ellas que pueden ajustarse independientemente, por eso en las funciones de programación hay que señalar el id de la cámara.

9.1.2. Blob detection

El reconocimiento de **blobs** se realiza por métodos matemáticos que pueden detectar en la imagen zonas que tienen determinadas características constantes, o que varían dentro de un rango de valores predeterminado (tolerancia), tales como el nivel de luminosidad o color, en comparación con las áreas que les rodean, de manera que el sistema los considera similares entre sí. En GAmuza hay dos técnicas para la detección de **blobs**: **Background subtraction** y **Color Tracking**.

En el primero, sustracción del fondo, el sistema guarda una imagen tomada del fondo antes de que los objetos o personas a detectar entren en el campo visual de la cámara, para ir después comparando los cambios que se producen en el valor de cada uno de los píxeles de los frames de video, de modo que si detecta un cambio significativo en una región de la imagen, respecto del modelo de fondo, representa ese área delimitando la zona o el perímetro. El panel **Computer vision** tiene un botón para capturar el fondo situado bajo del monitor **Balanced Tracking**.



Para trabajar con la técnica COLOR TRACKING, se debe activar el botón **Use Color Tracking** y subir al máximo el valor del slider **Subtraction Threshold** en el módulo **Computer vision**, para anular los datos de BACKGROUND SUBTRACTION. El área a detectar debe tener un color muy diferente al resto de colores que entren en el campo visual de la cámara y mantener un nivel de iluminación lo más uniforme posible. El Color Tracking utiliza el espacio de color HSV, porque asigna los valores del tono del

color de una forma más estable, ver en la página 182 los parámetros de color para seleccionar sus ajustes en la interface del panel.

La clase de GAmuza `gaCameraTracking()` tiene una serie de métodos para trabajar con blobs que se comunican con el panel de **computer vision**, recogiendo en tiempo real el ajuste que hagamos en su menú, por ejemplo, mediante los sliders del menú BLOB TRACKING SETTINGS podemos desestimar los blobs menores o mayores a un determinado tamaño, en BACKGROUND SUBTRACTION SETTINGS podemos seleccionar si los blobs buscados son más claros o más oscuros que el fondo, también se puede reducir o ampliar la dimensión de los blobs detectados, etc. Pasamos a describir cada uno de los métodos y sus parámetros, para analizarlos en conjunto posteriormente a través de algunos ejemplos. Recordad que para que funcionen debe estar seleccionado, al menos, el algoritmo Compute Contour Finder:

Con el método `getNumBlobs()`, se obtiene el número de blobs que detecta una cámara específica en cada frame de GAmuza.

`getBlobX(blobID)` y `getBlobY(blobID)`, devuelven la coordenada `x` e `y`, respectivamente, de un determinado blob detectado desde una cámara específica. GAmuza identifica los blobs por orden de aparición, el blob con id `0` será el que entró primero en el encuadre de la cámara y mantendrá ese id hasta que salga. Si dos áreas inicialmente identificadas con dos id se juntan, se convierten en un solo blob y el id de todos los blobs restantes puede cambiar. El parámetro de estas funciones es: id del blob.

`getBlobW(blobID)` y `getBlobH(blobID)`, devuelven la anchura y altura, en píxeles, de un determinado blob detectado desde una cámara específica.

`getBlobContourSize(blobID)`, devuelve la posición de todos los puntos del contorno de un blob detectado por una cámara específica.

`getBlobCPointX(blobID,x)` y `getBlobCPointY(blobID,y)`, devuelven las coordenadas `x` e `y`, respectivamente, de un punto determinado del contorno.

`getBlobGeometrySize(blobID)`, devuelve las líneas geométricas que devienen del contorno de un blob desde una cámara específica. Para que funcione debe estar seleccionado el algoritmo Compute Contour geometry.

`getBlobGLineX1(blobID,x1)`, `getBlobGLineY1(blobID,y1)`, `getBlobGLineX2(blobID,x2)` y `getBlobGLineY2(blobID,y2)`, devuelven las coordenadas `x1`, `y1`, `x2`, `y2` de una línea determinada del contorno geométrico de un blob. Para que funcione debe estar seleccionado el algoritmo Compute Contour geometry.

El siguiente ejemplo básico muestra cómo utilizar los datos obtenidos por los métodos `getNumBlobs()`, `getBlobX()`, `getBlobY()`, `getBlobW()` y `getBlobH()` desde el algoritmo Compute Contour Finder para detección de blobs y remarcarlos con formas rectangulares.

```
/*
 GAmuza 043                  E-9-2
 --------------------------------
 ComputerVision - Blobs
 creado por n3m3da | www.d3cod3.org
*/
camPanel = gaCameraTracking()    // asocia variable global a la clase de GAmuza
drawGUI = true                   // booleano para activar/desactivar el panel
camID = 0                        // id de la cámara
runningBlobs = 0                 // variable para el nº de blobs activos en cada frame
cam = ofTexture()                // textura para mostrar imagen de la cámara
captureWidth = 320               // anchura y altura frame captura video
captureHeight = 240

function setup()                 // para guardar el archivo de configuración XML
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(camID,captureWidth,captureHeight)
   cam:allocate(captureWidth,captureHeight,GL_RGB)
end

function update()
   cam = camPanel:getCameraTextureMod()   // la textura actualiza los frames de video
   runningBlobs = camPanel:getNumBlobs()  // devuelve el nº de blobs detectados
   camPanel:update()                      // actualiza los datos del panel
end

function draw()
   gaBackground(0.0,0.8)
   ofSetColor(255)
   scaleH = OUTPUT_H                //para dibujar imagen cámara a pantalla completa
   scaleW = scaleH* captureWidth/captureHeight
   cam:draw(OUTPUT_W/2 - scaleW/2, 0, scaleW,scaleH)
   ofSetLineWidth(3)                // inicio dibujo de blobs, grosor líneas
   ofNoFill()                       // no rellenar
   ofSetColor(31,165,210)           // color líneas  rectángulos
   ofPushMatrix()                   // activa la opción escalar
   ofScale(OUTPUT_W,OUTPUT_H,1.0)   // para escalar la imagen a pantalla completa
   for j=0, runningBlobs-1 do       // para reconocer todos los blobs
      x = camPanel:getBlobX(j)      // coordenada X de cada blob
      y = camPanel:getBlobY(j)      // coordenada X
      w = camPanel:getBlobW(j)      // anchura de cada blob
      h = camPanel:getBlobH(j)      // altura
      ofRect(x,y,w,h)               // dibuja rectángulos con esos datos
   end
   ofPopMatrix()                    // desactiva la opción escalar
   ofSetColor(255)
```

```
    if drawGUI then
        camPanel:draw()             // muestra el panel si drawGUI está activo
    end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI         // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

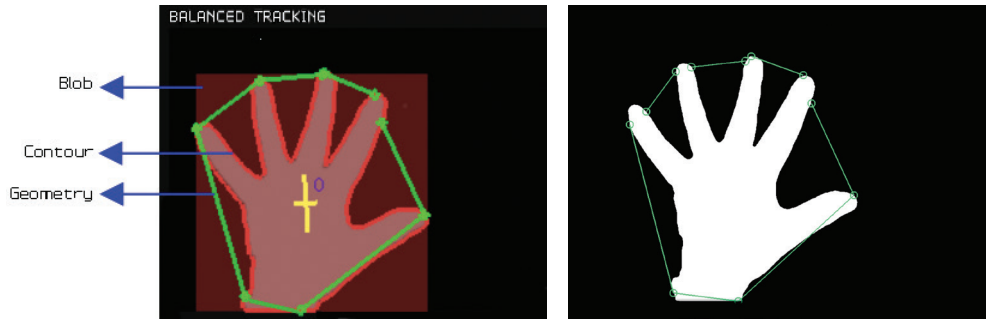
Analizaremos las diferencias que aporta este código, respecto al visto anteriormente para cargar el panel de tracking video. La variable global `runningBlobs` va a gestionar el número de blobs que detecta el sistema, como es un dato que puede cambiar según lo que acontezca delante de la cámara, en el bloque `update()` se iguala el valor de esa variable al que devuelve el método `getNumBlobs()`, así en cada frame se actualiza su valor al número de blobs detectados.

En el bloque `draw()`, se vincula la variable `cam` con el método `:draw()` de la clase, cuyos parámetros señalan las coordenadas `x`,`y`, y la anchura y altura de la imagen, en este caso, pantalla completa. Si no queremos que la ventana de salida muestre la imagen de la cámara, no se pone este método. Por ejemplo, si comentamos la siguiente línea de código del bloque `draw()`: `// cam:draw(OUTPUT_W/2 -scaleW/2, 0, scaleW,scaleH)`, para desactivar su acción, se seguirán viendo los rectángulos superpuestos a los blob sin visualizar la imagen de la cámara.

Después, la función `ofScale()` tiene como parámetros el incremento en los ejes `X`, `Y` y `Z`, los valores utilizados son pantalla completa para `X` e `Y`, y `1` para `Z` porque no se está trabajando en 3D. Después una estructura `for` asigna valores a unas variables vinculadas con las funciones `getBlobX()`, `getBlobY()`, `getBlobW()` y `getBlobH()` tantas veces en cada frame como número de blobs se hayan detectado, menos `1`, porque el primero tiene un id `0`, y asigna los valores obtenidos por esas funciones a los

parámetros de tantos rectángulos como blobs detectados, los parámetros son: coordenadas x, y, anchura y altura de cada blob.

En el siguiente ejemplo, sin que se muestre la imagen de la cámara,vamos a analizar el uso del método `getBlobContourSize()` relacionado con las funciones de detección de los puntos del contorno, y el método `getBlobGeometrySize()` relacionada con las de detección de las líneas de geometría, pare ello tiene que estar activado el algoritmo **Compute Contour geometry**.




```
/*
 GAmuza 043 ejemplos          E-9-3
 -------------------------------------
 ComputerVision - contorno y geometría
 creado por n3m3da | www.d3cod3.org
*/
camPanel = gaCameraTracking()    // asocia variable global a la clase de GAmuza
drawGUI = true                   // booleano para activar/desactivar el panel
camID = 0                        // id de la cámara
runningBlobs = 0                 // nº de blobs activos en cada frame
cam = ofTexture()                // textura para mostrar imagen de la cámara
captureWidth = 320               // anchura y altura frame captura video
captureHeight = 240

function setup()                 // para guardar el archivo de configuración XML
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(camID,captureWidth,captureHeight)     // inicializa el panel
   cam:allocate(captureWidth,captureHeight, GL_RGB)     // inicializa textura
end
```

```
function update()
   cam = camPanel:getCameraTextureMod()   // la textura actualiza los frames de video
   runningBlobs = camPanel:getNumBlobs()  //devuelve el n° de blobs detectados
   camPanel:update()                      //actualiza los datos del panel
end

function draw()
   gaBackground(0.0,1.0)

   scaleH = OUTPUT_H                // para escalar imagen video a pantalla completa
   scaleW = scaleH* captureWidth / captureHeight
   ofPushMatrix()
   ofTranslate(OUTPUT_W/2 - scaleW/2,0,0)
   ofScale(scaleW/captureWidth,scaleH/captureHeight,1.0)

   for j=0, runningBlobs-1 do       // recorre todos los blobs
      ofSetLineWidth(1)
      ofSetColor(255)
      ofFill()
      ofBeginShape()                // inicia dibujo contorno de cada blob detectado
                                    // recorre todos los puntos del contorno
      for i=0, camPanel:getBlobContourSize(j)-1 do
         x = camPanel:getBlobCPointX(j,i)       // coordenadas X de cada punto
         y = camPanel:getBlobCPointY(j,i)       // coordenadas Y
         ofVertex(x,y)                             // dibuja el contorno
      end
      ofEndShape(false)             // finaliza el dibujo sin cerrar las formas

      ofSetLineWidth(3)             // atributos del dibujo de la geometría
      ofSetColor(9,245,160)
      ofNoFill()
                                    // recorre todos los puntos de la geometría
      for z=0, camPanel:getBlobGeometrySize(j)-1 do
         x1 = camPanel:getBlobGLineX1(j,z)  // coordenadas X 1er punto de la línea
         y1 = camPanel:getBlobGLineY1(j,z)  // coordenadas Y 1er punto
         x2 = camPanel:getBlobGLineX2(j,z)  // coordenadas X 2° punto
         y2 = camPanel:getBlobGLineY2(j,z)  // coordenadas Y 2° punto
         ofLine(x1,y1,x2,y2)        //dibuja las líneas geométricas de cada blob
         ofCircle(x1,y1,3)          //dibuja círculos en cada extremo de las líneas
         ofCircle(x2,y2,3)
      end

   end
   ofPopMatrix()                    // cierra la opción escalar

    ofSetColor(255)
```

```
    if drawGUI then
        camPanel:draw()             // muestra el panel si drawGUI está activo
    end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI         // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

En este ejemplo, vemos que en el bloque del `update()` el valor de la variable global `runningBlobs` está de nuevo vinculado al que devuelve el método `getNumBlobs()`, para saber el número de blobs que se detectan en cada frame.

En el bloque `draw()`, las funciones `ofPushMatrix()` y `ofPopMatrix()` establecen el inicio y fin del código afectado por `ofScale()` que, como en el ejemplo anterior, escala los valores obtenidos de la cámara a pantalla completa. Después, para dibujar el contorno de los blobs establece un doble `for`, el primero se repite en cada frame tantas veces como blobs se hayan detectado, obteniendo el valor de la variable `runningBlobs-1`, mientras que el segundo `for` se repite tantas veces como número de puntos tenga el contorno de cada blob, obtenido por el valor de la función `getBlobContourSize(j)`, cuyo parámetro es el id del blob, que se lo asigna el primer `for`.

Para realizar el contorno utiliza el código de dibujo de formas irregulares por vértices `ofVertex(x,y)` que precisa tener antes y después, las funciones `ofBeginShape()` y `ofEndShape(false)` respectivamente. El valor de las coordenadas de los distintos vértices lo obtiene de las funciones `getBlobCPointX(j,i)` y `getBlobCPointY(j,i)` cuyos parámetros son el id del blob, que se obtiene del primer `for` y el id del punto que se obtiene del segundo `for`.

El código siguiente del bloque `draw()` inicia el dibujo de la geometría de cada blob, con un tercer `for` que se repetirá tantas veces como líneas tenga esa geometría, ese valor se obtiene de la función `getBlobGeometrySize(j)` cuyo parámetro, id del blob, viene dado por el primer `for` (`runningBlobs-1`). Los parámetros de las líneas se obtienen de las cuatro funciones que nos dan las coordenadas de los puntos de inicio y fin de cada una de ellas, `getBlobGLineX1(j,z)`, `getBlobGLineY1(j,z)`, `getBlobGLineX2(j,z)`, `getBlobGLineY2(j,z)`, cuyo segundo parámetro es el id de la línea que es asignado por este tercer `for`, con estos datos se dibujan también pequeños círculos en los extremos de las líneas para visualizar mejor las articulaciones y aproximarlo a la representación que tiene GAmuza en el panel **Computer vision**.

El siguiente ejemplo vinculado con la detección blobs utiliza Color tracking, y está planteado para la detección de un sólo blob, una forma de color diferenciado que el espectador mueve delate de la cámara para modificar el volumen y velocidad de reproducción (pitch) de un archivo de audio. Para ajustar el reconocimiento del color, en el bloque **color tracking settings** del panel **computer vision**, el color seleccionado es **hue**: 0,28, **Saturation**: 0.75 y **Value**: 0.45, y el slider **Subtraction Threshold** se ha desplazado a su máximo valor para anular la aparición de otros blobs detectados por **BACKGROUND SUBTRACTION.** En el bloque de algoritmos, debe mantenerse seleccionado **Compute Contour Finder** para la detección del blob.

```
/*
 GAmuza 043          E-9-4
 ------------------------------
 ComputerVision - Color tracking
 creado por mj
*/
                              //Variables tracking
camPanel = gaCameraTracking()
drawGUI = true
captureWidth = 320
captureHeight = 240
cam = ofTexture()
                              // Variables sonido
mySound = ofSoundPlayer()
posX = OUTPUT_W/2
posY = OUTPUT_H/2
posxVol= 0.0
posySpeed= 0.0

function setup()              // inicializa panel de tracking
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(0,captureWidth,captureHeight)
   cam:allocate(captureWidth,captureHeight,GL_RGB)
```

```
   //sonido
   mySound:loadSound(gaImportFile("hypno00.wav"),true)
   mySound:setLoop(true)
   mySound:play()
end

function update()
   cam = camPanel:getCameraTextureMod() // la textura actualiza los frames de video
   camPanel:update()                    // actualiza los datos del panel
   ofSoundUpdate()                      // actualiza sonido
end

function draw()
   gaBackground(0.0,1.0)
   posX = OUTPUT_W * camPanel:getBlobX(0) //otro modo de escalar a pantalla completa
   posY = OUTPUT_H * camPanel:getBlobY(0) // solo se busca posición del blob 0
   ofSetColor(0,255,0)                    // color verde
   ofCircle(posX, posY, 50)               // dibuja círculo en la posición del blob
   posxVol= ofMap(posX, 0, OUTPUT_W, 0.1, 1.0, true)  // mapea el valor de posición
   posySpeed= ofMap(posY, 0, OUTPUT_H, 0.1, 2.0, true) // al del volumen y speed
   mySound:setVolume(posxVol)             // volumen según posición X del blob
   mySound:setSpeed(posySpeed)            // velocidad según posición Y del blob
   ofSetColor(255)
      if drawGUI then
         camPanel:draw()                  // muestra panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI               // con tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

Los bloques `setup()` y `update()` contienen código destinado a la reproducción del sonido, e inicialización y actualización del panel tracking video, pero nada para la detección de blobs porque no se requiere saber el número de blobs, ni actualizar ese valor, dado que solo se debe reconocer uno.

En el bloque `draw()`, sólo hay que escalar dos coordenadas, se simplifica multiplicando el valor de la coordenada que devuelven las funciones `getBlobX(0)` y `getBlobY(0)` por el ancho y alto de la ventana de salida, el parámetro de estas funciones es el id del único blob buscado, el `0`. El volumen del archivo de audio se regula mapeando los valores de la posición `X` entre `0.1` y `1.0`; de forma similar, la velocidad de reproducción se mapea respecto a la posición `Y`, entre `0.1` y `2.0` . Y en función de los movimientos que la forma de ese color detectado realice ante la cámara de vídeo, el sonido se va modulando al acelerar o ralentizar la velocidad de reproducción y variar la intensidad de volumen.

9.1.3. Motion

Otro tipo de datos que obtiene la clase `gaCameraTracking()` de GAmuza es la cantidad de movimiento y localización media de la masa de píxeles que reflejan ese movimiento. Para este tipo de datos no es necesario activar ningún algoritmo en el panel **computer vision**.

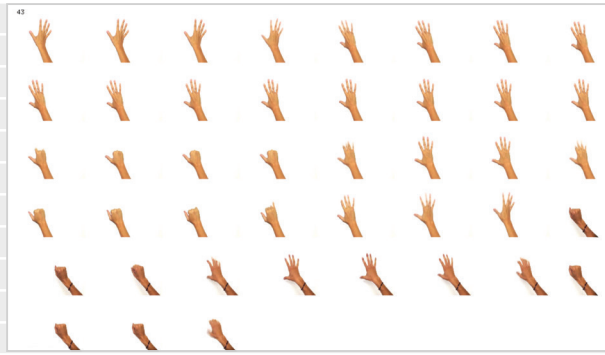
Los métodos vinculados a Motion son: `getMotionQ()`, `getMotionX()` y `getMotionY()`, y la sensibilidad del sistema para los datos que devuelven se regula en la interface del panel con los sliders del grupo MOTION DETECTION SETTINGS.

El método `getMotionQ()`, devuelve la cantidad de movimiento que se produce ante la cámara. Su valor refleja la cantidad de píxeles que se han modificado en cada frame respecto a la imagen de fondo que tiene almacenada, realmente ese valor oscila entre `0` y el ancho por alto de la imagen capturada, si bien GAmuza lo escala entre `0.0` y `1.0`. Los métodos `getMotionX()` y `getMotionY()`, devuelven el valor de las coordenadas `x` e `y`, respectivamente, del punto medio de la masa de movimiento, el desplazamiento de la representación de ese punto puede indicar hacia donde se traslada mayoritariamente el conjunto de píxeles que se han movido en el frame de la imagen video, respecto al modelo de fondo que ha guardado el sistema.

El siguiente ejemplo muestra la utilización del método `gaMotionQ()` en un proyecto básico de imágenes interactivas. Según la cantidad de movimiento registrada por la cámara de video, el sistema selecciona y reproduce un número mayor o menor de archivos de imagen que se despliegan por la pantalla siguiendo una retícula cuadrangular.

Para trabajar con la base de datos de imágenes se utiliza la clase de openFrameworks `ofDirectory()` que gestiona el directorio de los archivos guardados en una carpeta llamada `imag`, situada dentro de la carpeta `data` del script.

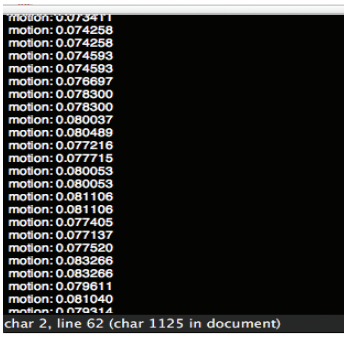
```
/*
 GAmuza 043                    E-9-5
 ------------------------------
 Interactividad  motion tracking
 creado por mj
*/

// variables panel tracking video
camPanel = gaCameraTracking()
drawGUI = true
captureWidth = 320
```



```
captureHeight = 240
                                    // variables imágenes y movimiento
dir= ofDirectory()
numImagenes = 0
imagesCount = 0
imagenBase = ofImage()
imagenes = {}                       // tabla archivos de imagen
motion = {}                         // tabla valores de movimiento
motionScaler = 0.133132             //ajustar según lecturas de movimiento en cada caso

function setup()                    //inicializa panel tracking
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(0,captureWidth,captureHeight)

   motion[0]={}                     //inicializa tabla movimiento
   motion[0].q = 0                  // valores cantidad de movimiento
   motion[0].factor = 0             // factor a aplicar para el número de imágenes
   imagenBase:loadImage(gaImportFile("mano0.png"))    // carga imagen base
   dir:listDir(gaImportFile("imag/"))                 // carga directorio de imágenes
   numImagenes = dir:size()                           // calcula n° archivos

   if dir:size()> 0 then
      for i=0,numImagenes-1 do         // inicializa la tabla de archivos de imagen
         imagenes[i] = ofImage()       // asigna la clase a cada objeto de la tabla
         imagenes[i]:(dir:getPath(i))  //carga todas las imágenes
      end
   end
end

function update()
   camPanel:update()                        // actuaiza los datos del panel
   motion[0].q = camPanel:getMotionQ()      // detecta cantidad de movimiento
   motion[0].factor= math.floor(ofMap(motion[0].q,0.0,motionScaler,0,numImagenes,true))
                            // escala la cantidad de movimiento al número de imágenes
   //gaLog(string.format("motion:%f",motion[0].q))
   //descomentar gaLog para ver en la consola el valor de movimiento máximo = motionScaler
end

function draw()
   gaBackground(1.0,1.0)
   ofSetColor(255)
   ancho = imagenBase:getWidth()/4  // calcula el ancho de la imagen base
   alto = imagenBase:getHeight()/4  // calcula el alto de la imagen base
   cols = math.floor(OUTPUT_W/ancho) // número de columnas de imágenes
   c=0                              // para hacer una retícula 2D con un solo for
```

```
   r=0
   for i=0, numImagenes-1 do
      if i < motion[0].factor then
         imagenes[i]:draw(c*ancho,r*alto,ancho,alto)
         c += 1

         if c == cols then          // si las imágenes llegan al ancho de pantalla
            c= 0
            r +=1                   // se ubican en una fila inferior
         end
      end
   end

   ofSetColor(0)
                                    //dibuja en pantalla valor actual de numImagenes
   ofDrawBitmapString(tostring(motion[0].factor),20,20)

   ofSetColor(255)
      if drawGUI then
         camPanel:draw()               // muestra el panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI            // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

El ejemplo plantea que cuanto mayor sea el movimiento que se registra delante de la cámara, mayor será el número de imágenes reproducidas. Sabemos que los valores del método `getMotionQ()` oscilan ente `0.0` y `1.0`, pero ese valor máximo depende de las condiciones de presentación del proyecto por lo que para ajustarlo se utiliza el siguiente código en el bloque `update()` para monitorizar los valores:




```
gaLog(string.format("motion: %f", motion[0].q))
```

La consola del IDE los muestra, y el mayor de esos valores sirve para delimitar cuál va a ser el de la variable `motionScaler`.

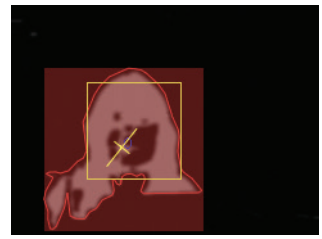
Para gestionar los archivos de imagen, se vincula la variable global `dir` a la clase de openFrameworks `ofDirectory()`, se declara una tabla de imágenes y otra para los datos de la cantidad de movimiento.

En el bloque `setup()`, después de inicializar el panel de tracking, se crea una nueva tabla en `motion[0]` con dos elementos: `motion[0].q` albergará los datos de lectura del movimiento y `motion[0].factor` se utilizará para mapear el número de imágenes en función del valor de movimiento. Dado que tablas solo guardarán un valor que va cambiando, su index es 0 e inicialmente se les asigna un valor `0` que se actualizará en el bloque `update()`. La siguiente línea de código con la variable `imagenBase` es para cargar la imagen que se toma como referencia. Después, viene el código dirigido a la gestión de los archivos de imagen, aplicándole a la variable `dir` el método `listDir()` cuyo parámetro es la función de GAmuza `gaImportFile()` que indica la carpeta en la que están almacenados. A la variable global `numImagenes` se le asigna el valor que devuelve el método `size()` de la clase `ofDirectory()`, así se obtiene el número de imágenes guardadas en el directorio antes indicado. Y mediante un `for` que se repetirá tantas veces como imágenes hayan, se le asignan a todas la clase `ofImage()`, y se cargan utilizando los métodos `loadImage` de la clase `ofImage()` y `getPath` de la clase `ofDirectory()`.

En el bloque `update()`, se le asigna a la tabla `motion[0].q` el valor que devuelve el método `getMotionQ()` y para el valor de `motion[0].factor` se establece el siguiente cálculo: la función `math.floor()` devuelve un número entero redondeando hacia abajo, si el resultado de los datos que tiene entre paréntesis fuera 0.5, devuelve 0, y con la función `ofMap()`, los valores de la cantidad de movimiento que vayan entre 0.0 y lo que se le asignó a la variable `motionScaler` se mapean entre 0 y el número de imágenes almacenadas, devolviendo así este factor el número de imágenes que van a mostrarse en función de la cantidad de movimiento.

En el bloque `draw()`, un `for` se repite tantas veces como el valor que `motion[0].factor` ha asignado a `numImagenes` en el `update()`. La línea de código `ofDrawBitmapString(tostring(motion[0].factor),20,20)`, muestra en el borde superior izquierdo de la ventana de salida cuál es ese valor.

9.1.4. Haar



La clase `gaCameraTracking()` tiene una serie de métodos para el reconocimiento de partes del cuerpo humano que vienen preconfigurados por archivos Haar cascade de OpenCV. Por defecto, tiene cargado el archivo `"haarcascade_frontalface_alt.xml"`. Si se quiere reconocer otra parte del cuerpo o utilizar otra configuración para la detección de rostros frontales hay que cargar el archivo con el método `setHaarFile(gaImportFile("ArchivoHaar.xml"))` y guardar previamente ese archivo Haar en la carpeta data.

Debe estar activo el algoritmo **compute haar finder** en el panel. Otros métodos vinculados a este tipo de tracking son: `getNumHaars()`, `getHaarX()`, `getHaarY()`, `getHaarW()`, `getHaarH()`,

`getNumHaars()`, devuelve el número de Haar detectados en cada frame.

`getHaarX(haarID)` y `getHaarY(haarID)`, devuelven las coordenadas X e X, respectivamente, del haar especificado en su parámetro.

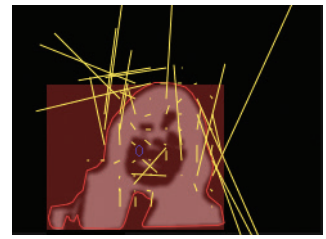
`getHaarW(haarID)` y `getHaarH(haarID)`, devuelven la anchura y altura, respectivamente, del haar especificado en su parámetro.

Ejemplo:

```
/*
 GAmuza 043                  E-9-6
 ---------------------------------
 ComputerVision - Haar
 creado por n3m3da | www.d3cod3.org
*/
                      // variables panel tracking video
camPanel = gaCameraTracking()
drawGUI = true
captureWidth = 320
captureHeight = 240
runningHaars = 0

function setup()
   //inicializa panel tracking
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(0,captureWidth,captureHeight)
end
```

```
function update()
   camPanel:update()
   runningHaars = camPanel:getNumHaars()        // nº de haar activos
end

function draw()
   gaBackground(0.0,1.0)
   ofSetColor(255,0,0)
   ofFill()
   for i=0, runningHaars do
      posX = OUTPUT_W * camPanel:getHaarX(i)  // escala X de haars a ancho pantalla
      posY = OUTPUT_H * camPanel:getHaarY(i)  // escala Y
      ancho = camPanel:getHaarW(i) * OUTPUT_W //escala ancho haars a ancho pantalla
      alto = camPanel:getHaarH(i) * OUTPUT_H  // escala alto
      ofRect(posX, posY, ancho, alto)         // dibuja rectángulos en cada haar
   end

   ofSetColor(255)
      if drawGUI then
         camPanel:draw()                 // muestra el panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI             // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end
function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

9.1.5. Optical Flow



GAmuza establece una rejilla a partir de la captura de la cámara de video, vinculada a las constantes `OPTICAL_FLOW_GRID_X`, número de columnas de la rejilla y `OPTICAL_FLOW_GRID_Y`, número de filas de la rejilla, para representar vectorialmente la dirección del movimiento que se produce en cada una de las posiciones que marca esa rejilla. Los métodos vinculados a este tipo de tracking son:

`getOpticalFlowX(index)` y `getOpticalFlowY(device,index)`, devuelven las coordenadas x e y, respectivamente, de una determinada posición en la rejilla. Su parámetro es el index de la posición en la rejilla.

`getOpticalFlowVX(index)` y `getOpticalFlowVY(device)`, devuelven la velocidad de movimiento de las coordenadas x e y, respectivamente, de una determinada posición en la rejilla del optical flow. Su parámetro es el index de la posición en la rejilla.

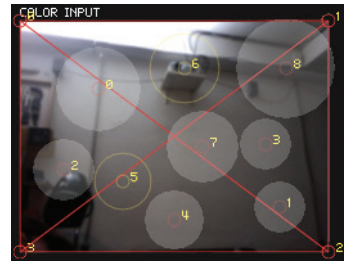
En el panel **Computer Vision** debe estar activo el algoritmo **Compute Optical Flow**

```
/*
 GAmuza 043             E-9-7
 ------------------------------
 ComputerVision - Optical Flow
 creado por n3m3da | www.d3cod3.org
*/

// variables panel tracking video
camPanel = gaCameraTracking()
camID = 0
cam = ofTexture()
drawGUI = true

captureWidth = 320
captureHeight = 240
opticalFlowCols = math.ceil(captureWidth/OPTICAL_FLOW_COLS_STEP)
opticalFlowRows = math.ceil(captureHeight/OPTICAL_FLOW_ROWS_STEP)

function setup()              //inicializa panel tracking
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(camID,captureWidth,captureHeight)

   cam:allocate(captureWidth,captureHeight,GL_RGB)
end
```

```
function update()
   cam = camPanel:getCameraTextureMod()
   camPanel:update()
end

function draw()
   gaBackground(0.0,1.0)
   ofSetColor(255)
   scaleH = OUTPUT_H                    // dibuja imagen video a pantalla completa
   scaleW = scaleH* captureWidth / captureHeight
   cam:draw(OUTPUT_W/2 - scaleW/2,0,scaleW,scaleH)
   ofSetColor(31,165,210)               // dibuja representación de optical flow
   ofSetLineWidth(3)
   ofPushMatrix()
   ofTranslate(OUTPUT_W/2 - scaleW/2, 0, 0)
      ofScale(scaleW/captureWidth,scaleH/captureHeight,1)
      for i=0, opticalFlowCols*opticalFlowRows-1 do
         x = camPanel:getOpticalFlowX(i)
         y = camPanel:getOpticalFlowY(i)
         vX = camPanel:getOpticalFlowVX(i)
         vY = camPanel:getOpticalFlowVY(i)
         ofLine(x, y, x + vX, y + vY)
      end
   ofPopMatrix()

   ofSetColor(255)
      if drawGUI then
         camPanel:draw()                // muestra el panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI             // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()                 //para ajustar opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end
function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end
function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

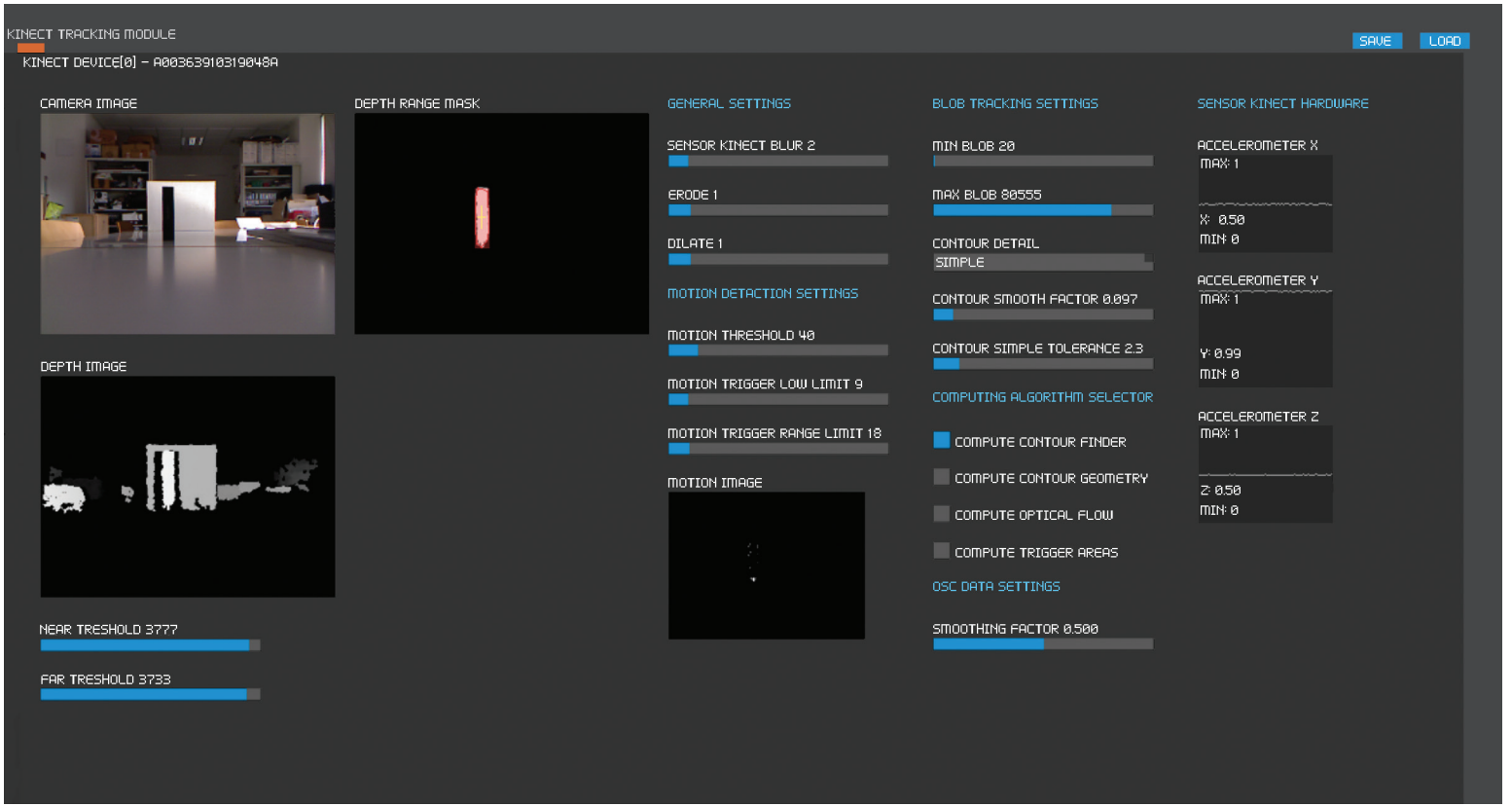
9.1.6. Trigger Areas

Desde el panel de **Camera Tracking**, en la imagen que captura la cámara se pueden activar nueve áreas circulares, ajustables en tamaño y posición para que las acciones programadas para el tracking sólo se apliquen si el punto central del blob entra en el área especificada. Clicando en el centro de un área y arrastrando el ratón se puede desplazar su posición. Clicando en cualquier otra zona del área y arrastrando el ratón, se puede ampliar o reducir su tamaño.




```
/*
 GAmuza 043              E-9-8
 ----------------------------
 ComputerVision - triggerAreas
 uso opción trigger area con panel Camera Tracking.
 creado por n3m3da | www.d3cod3.org
 */

camPanel = gaCameraTracking()
camID = 0
cam = ofTexture()
drawGUI = true
captureWidth = 320
captureHeight = 240

function setup()
   gaWave(GA_BROWN,1)      // inicializa un oscilador con ruido marrón
                           // inicializa panel tracking
   camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
   camPanel:setup(camID,captureWidth,captureHeight)
   cam:allocate(captureWidth,captureHeight,GL_RGB)
end

function update()
   cam = camPanel:getCameraTextureMod()
   camPanel:update()
            // activa el volumen del ruido marrón si el trigger area 0 está activada
    if camPanel:getTrigger(0) then
        gaWaveVolume(0,0.8)
    else
        gaWaveVolume(0,0.0)
    end
end
```

```
function draw()
   gaBackground(0.0,0.8)
                                        // dibuja la imagen capturada por la cámara
    ofSetColor(255)
   scaleH = OUTPUT_H
   scaleW = scaleH* captureWidth / captureHeight
    cam:draw(OUTPUT_W/2 - scaleW/2,0,scaleW,scaleH)

   ofSetColor(255)
      if drawGUI then
         camPanel:draw()               // muestra el panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI            // con la tecla g se muestra/oculta el panel
   end
end

function mouseDragged()          // para ajustar las opciones del panel con el ratón
   camPanel:mouseDragged(gaMouseX(),gaMouseY())
end
function mousePressed()
   camPanel:mousePressed(gaMouseX(),gaMouseY())
end
function mouseReleased()
   camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```



Panel Tracking por sensor Kinect

9.2. Panel Tracking: por sensor kinect

A continuación se describe la interface del panel de Tracking para el sensor kinect y los métodos de la clase de GAmuza `gaKinectTracking()`, deteniéndonos especialmente en las diferencias que hay respecto al tracking por cámara de video. Al igual que en el capítulo anterior, se explica la interface a partir del código que genera su carga.

```
/*
 GAmuza 0428 ejemplos        E-9-9
 ---------------------------------
 Panel Tracking kinect
 creado por n3m3da | www.d3cod3.org
 */

kinectPanel = gaKinectTracking()
drawGUI = true
kinectDevID = 0
kinectImage = ofTexture()
function setup()
   kinectPanel:setGuiSettingsFile(gaDataPath("kinectTrackingSettings.xml"))
         // parámetros setup: id,IR,videoImage,LED MODE
   kinectPanel:setup(kinectDevID,true,false,1)
end

function update()
   kinectImage = kinectPanel:getCameraTexture()
   kinectPanel:update()
end

function draw()
   gaBackground(0.1,1.0)
      // muestra  la imagen desde el dispositivo escalada a pantalla completa
   ofSetColor(255)
   scaleH = OUTPUT_H
   scaleW = scaleH* 640 / 480
   kinectImage:draw(OUTPUT_W/2 - scaleW/2,0, scaleW,scaleH)

   ofSetColor(255)
   if drawGUI then
      kinectPanel:draw()          // muestra el panel si drawGUI está activo
   end
end
```

```
function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI       // con la tecla g se muestra/oculta el panel
   end
end
                                   // para ajustar las opciones del panel con el ratón
function mouseDragged()
   kinectPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   kinectPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   kinectPanel:mouseReleased(gaMouseX(),gaMouseY())
end

function exit()
   kinectPanel:close()             // siempre hay que cerrar el dispositivo
end
```

Al igual que para el tracking con cámara de video, se inicia asociando una variable a la clase de GAmuza que gestiona, en este caso, al sensor Kinect, `gaKinectTracking()`, se establece un swich booleano para mostrar u ocultar el panel en la ventana de salida; y la variable `kinectImage` se vincula a la clase de openFrameworks `ofTexture()` para acoger la imagen que viene de la Kinect.

La interface del panel del sensor Kinect también está regulada por un archivo .xml que se guarda en la carpeta data junto al archivo del script utilizando el método `setGuiSettingsFile()` cuando se salvan los ajustes del panel, y se inicializa con el método `setup(int,bool,bool,int)`, cuyos parámetros son: el ID del sensor, el uso o no de infrarrojos, el uso o no de color en la imagen y el último parámetro `int` es Led Mode cuyos valores posibles son: 0 -> LED_OFF, 1 -> LED_GREEN, 2 -> LED_RED, 3 -> LED_YELLOW, 4 -> LED_BLINK_GREEN y 6 -> LED_BLINK_YELLOW_RED.

En el bloque del `update()`, se asigna la imagen obtenida por el sensor Kinect a la textura: `kinectImage = kinectPanel:getCameraTexture()`; y se actualiza la información del panel con el método `update`.

En el bloque del `draw()`, se puede dibujar la imagen capturada por el sensor y dibujar o no el panel en la ventana de salida. El resto de bloques de código funcionan igual que en el panel de tracking con cámara de video.

9.2.1. Descripción panel

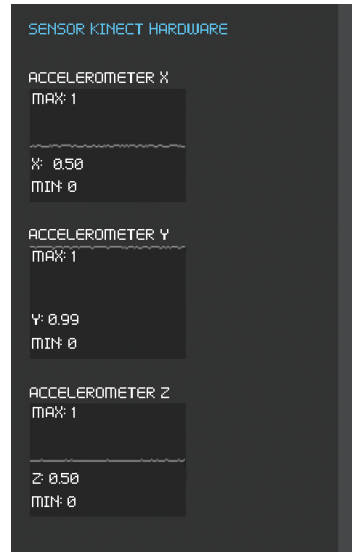
Las diferencias fundamentales entre el sensor Kinect y una cámara de video en su utilización para tracking reside en la capacidad para detectar profundidad y el mecanismo de inclinación motorizado.

DEPTH IMAGE

El sensor de profundidad combina infrarrojos con un sensor CMOS monocromo lo que le permite a Kinect ver/calcular el espacio en 3D sin que las condiciones de luz ambiental influyan; el rango de detección de profundidad va 70 cm a 6 metros y en el panel puede ajustarse con los sliders:

- NEAR TRESTHOLD: determina el plano de los elementos más próximos, como mínimo desde 70 cm del sensor
- FAR TRESHOLD: hasta qué plano de profundidad se quiere detectar, como máximo 6 m

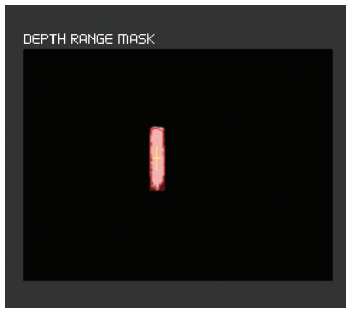
SENSOR KINECT HARDWARE



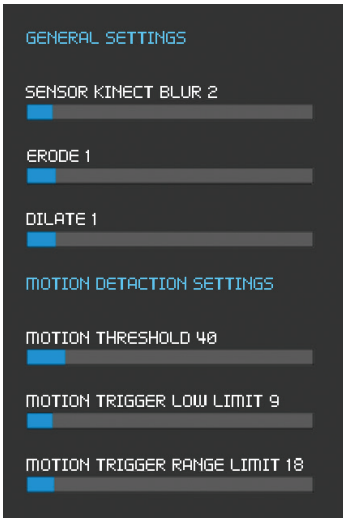
El mecanismo de inclinación es un acelerómetro para determinar la orientaciónde la Kinect. La interface del panel refleja estos datos en los 3 monitores de la columna derecha y en los scripts se pueden conocer con los métodos: `getAccelX()`, `getAccelY()` y `getAccelZ()` que devuelven un valor `float` ente `0.0` y `1.0`

Los otros componentes de la interface actúan de forma similar a los panel de tracking para cámaras de video.

DEPTH RANGE MASK: monitor donde se muestra el resultado de los algoritmos aplicados.



GENERAL SETINGS

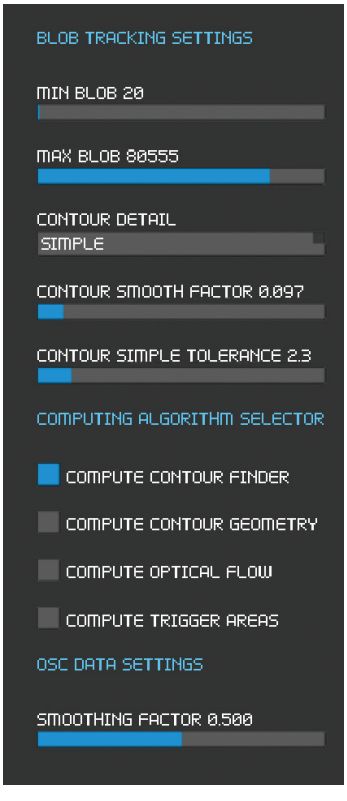


- SENSOR KINECT BLUR: grado de desenfoque de la señal de entrada
- ERODE: reducir el área de las superficies detectadas
- DILATE: ampliar el área de las superficies detectadas

MOTION DETECTION SETTINGS. Regula la detección de movimiento según niveles de profundidad.

- MOTION TRESHOLD: regula la sensibiliad del sistema para activar la detección, si es poco sensible se debe ampliar el umbral (Threshold)
- MOTION TRIGGER LOW LIMIT: nivel más bajo para activar la detección de movimiento
- MOTION TRIGGER RANGE LIMIT: limite del rango para la detección de movimiento.

BLOB TRACKING SETTINGS



- MIN BLOB: tamaño mínimo de los blobs
- MAX BLOB: tamaño máximo de los blobs
- CONTOUR DETAIL: menú con tres opciones, raw, smooth o simple
- CONTOUR SMOOTH FACTOR, si se ha seleccionado smooth, ajusta el nivel de suavizado
- CONTOUR SIMPLE TOLERANCE, si se ha seleccionado simple, ajusta el rango de vértices que articulan el contorno.

COMPUTING ALGORITHM SELECTOR

- COMPUTE CONTOUR FINDER, para procesar el contorno de los blobs detectados.
- COMPUTE CONTOUR GEOMETRY. Determina formas geométricas a partir de los contornos. Al activarlo se activa automáticamente el anterior si no lo estaba.
- COMPUTE OPTICAL FLOW. Calcula el flujo óptico del movimiento y lo describe con vectores que indican la dirección e intensidad.
- COMPUTE TRIGGER AREAS. Activa la posibilidad de utilizar 8 zonas circulares, como en el módulo Computer Vision.

9.2.2. Métodos y funciones para el sensor Kinect

Además de los métodos mencionados hasta ahora la clase `gaKinectTracking()` tiene también los siguientes:

Clase	Método	Devuelve
	`gaKinectTracking:getDepthTexture()`	`ofTexture`
	`gaKinectTracking:getCameraPixels()`	`ofPixelsRef`
Blobs		
	`gaKinectTracking:getNumBlobs()`	`int` num Blobs
	`gaKinectTracking:getBlobX(int blobID)`	`float` posX del blob
	`gaKinectTracking:getBlobY(int blobID)`	`float` posY del blob
	`gaKinectTracking:getBlobW(int blobID)`	`float` ancho del blob
	`gaKinectTracking:getBlobH(int blobID)`	`float` alto del blob
	`gaKinectTracking:getBlobAngle(int blobID)`	`float` ángulo del blob
Contorno		
	`gaKinectTracking:getBlobContourSize(int blobID)`	`int` num puntos contorno
	`gaKinectTracking:getBlobCPointX(int blobID,int pointContID)`	`float` posX del punto
	`gaKinectTracking:getBlobCPointY(int blobID,int pointContID)`	`float` posY del punto
Geometría (líneas)		
	`gaKinectTracking:getBlobGeometrySize(int blobID)`	`int` tamaño geometría
	`gaKinectTracking:getBlobGLineX1(int blobID,int lineID)`	`float` posX1 líneaGeom
	`gaKinectTracking:getBlobGLineY1(int blobID,int lineID)`	`float` posY1 líneaGeom
	`gaKinectTracking:getBlobGLineX2(int blobID,int lineID)`	`float` posX2 líneaGeom
	`gaKinectTracking:getBlobGLineY2(int blobID,int lineID)`	`float` posY2 líneaGeom
Optical Flow		
	`gaKinectTracking:getOpticalFlowX(int indexGridPosition)`	`float` posX grid
	`gaKinectTracking:getOpticalFlowY(int indexGridPosition)`	`float` posY grid
	`gaKinectTracking:getOpticalFlowVX(int indexGridPosition)`	`float` posVX grid
	`gaKinectTracking:getOpticalFlowVY(int indexGridPosition)`	`float` posVY grid
Trigger Areas		
	`gaKinectTracking:getTrigger(int triggerID)`	`bool` `true` or `false`

En el siguiente ejemplo se muestra el uso de algunos de estos métodos para dibujar el contorno de la imagen reconocida a través de los blobs.

```
/*
 GAmuza 043                    E-9-10
 ---------------------------------------
 Sensor Kinect - Kinect contorno

 creado por n3m3da | www.d3cod3.org
*/
                                                  // Kinect Variables
kinectPanel = gaKinectTracking()
drawGUI = true
kinectDevID = 0
kinectImage = ofTexture()
runningBlobs = 0

function setup()
                                // guarda archivo configuración XML e inicializa Kinect
   kinectPanel:setGuiSettingsFile(gaDataPath("kinectTrackingSettings.xml"))
   kinectPanel:setup(kinectDevID,false,true,1)
end

function update()
   kinectImage = kinectPanel:getCameraTexture()  // asigna imagen kinect a la textura
   kinectPanel:update()                          // actualiza datos del panel
   runningBlobs = kinectPanel:getNumBlobs()      // número de blobs detectados
end

function draw()
   gaBackground(0.1,1.0)
   ofSetColor(255)      // dibuja imagen capturada por kinect a pantalla completa
   scaleH = OUTPUT_H
   scaleW = ((scaleH* 640) / 480) - 240
   kinectImage:draw(OUTPUT_W/2 - scaleW/2,0, scaleW,scaleH)

   ofPushMatrix()                          //inicializa opción escalar
   ofTranslate(OUTPUT_W/2 - scaleH/2,0,0)  // traslada y escala las posiciones
   ofScale(scaleW/640,scaleH/480,1.0)
   for j=0, runningBlobs-1 do              // recorre los blobs detectados
      ofSetLineWidth(3)
      ofSetColor(255)
      ofNoFill()
      ofBeginShape()                 //inicializa el dibujo del contorno de cada blob
```

```
      for i=0, kinectPanel:getBlobContourSize(j)-1 do  // recorre los puntos contorno
         x = kinectPanel:getBlobCPointX(j,i)           // posición X de cada punto
         y = kinectPanel:getBlobCPointY(j,i)           // posición Y de cada punto
         ofVertex(x,y)                                 // dibuja los contornos

      end
      ofEndShape(false)                                //finaliza sin cerrar las formas
   end
   ofPopMatrix()                                       // finaliza opción escalar

   ofSetColor(255)
      if drawGUI then
         kinectPanel:draw()       // muestra el panel si drawGUI está activo
      end
end

function keyReleased()
   if gaKey() == string.byte('g') then
      drawGUI = not drawGUI       // con la tecla g se muestra/oculta el panel
   end
end
                                  // para ajustar las opciones del panel con el ratón
function mouseDragged()
   kinectPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
   kinectPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
   kinectPanel:mouseReleased(gaMouseX(),gaMouseY())
end

function exit()
   kinectPanel:close()            // siempre hay que cerrar el dispositivo
end
```