TRACES

Processing Code

```
/*
 --------------------------------------------------------------------
 TRACES
 --------------------------------------------------------------------

 IR camera tracking combined with on floor projection.
 An overhead camera tracks people and their movement in a specified area
 the detected movement is projected back onto the space as traces left behind.

 Unlike the traces we leave behind online the projected traces can be removed
 by a special broom.

 The set-up:
 * IR-sensitive surveillance camera with integrated IR-LEDs
 * Projector
 * a broom
 * 4 bright white LEDs to be used as tracking markers at the edges of the broom
 * a push button to switch between tracing and cleaning mode

 The surveillance camera as well as the projector is mounted on the ceiling.
 The tracking LEDs are attached to the corners of the broom so that the light of the
 LEDs can be seen from above and tracked accordingly. The push button is attached to
 the broom stick to enable switching between the two modes.
 Push button and LEDs are set up in Arduino and connected to Processing via the serial
 port.

 Please see Arduino code for further information on the Arduino circuit.

 Created November 10 2009
 By Katja Knecht & Yuan Qi

 Appendant Processing files:
 Class_Circle
 Class_Trace

 Appendant Arduino files:
 Trace Arduino

 References:
 * TRACKING
 JMyron Library: CameraAsMouse by JTNIMOY

 * COLLISION DETECTION
 boolean insidePolygon() by aaron steed
http://processing.org/discourse/yabb_beta/YaBB.cgi?board=Programs;action=display;num=118917882
6;start=4#4

 boolean lineIntersectsCircle() and Class_Circle by fjen
 http://processing.org/discourse/yabb2/YaBB.pl?board=Programs;action=display;num=1189178826

 * PARTICLE SYSTEM
 Rotate by Jared "BlueThen" C.
 www.bluethen.com

 * PUSHBUTTON
 pa_PushButton by Melvin Ochsmann for Malmoe University
 http://webzone.k3.mah.se/projects/arduino-
workshop/projects/arduino_meets_processing/instructions/pushbutton.html
 */

// --------------------------------------------------------------------
// --------------------------------------------------------------------
```

```
import processing.serial.*;
import cc.arduino.*;
import JMyron.*;


// -------- VARIABLES ------------------------------------------

Serial myPort;  // The serial port
JMyron m;//a camera object

Point[] polygon;
ArrayList polyTrack = new ArrayList(); // ArrayList to store polygon points
ArrayList trackTraces = new ArrayList(); // Arraylist to store traces

boolean mode = false; // mode: false - tracing, true - removing trace

int videoW = 320; // width of input camera image
int videoH = 240; // height of input camera image

int projW = 160; // width of projection in camera image
int projH = 120; // height of projection in camera image

int offsetX = 20; // offset of projection in regard to origin
int offsetY = 40; // of camera image - for tracking people
int offX = 50; // offset of projection in regard to origin
int offY = 84; // of camera image - for tracking broom

int value=0; // variable to store serial input

//variables to maintain movement of traces
float objx = 400;
float objy = 300;
float objdestx = 400;
float objdesty = 300;


// -------- SET-UP ---------------------------------------------

void setup()
{
  size(800,600); // output screen size

  m = new JMyron();//make a new instance of the object
  m.start(videoW,videoH);//start a capture at 320x240
  m.trackColor(255,255,255,256*3-100);//track white
  m.update(); //update camera view
  m.adaptivity(2); // // sets the speed at which the retina image adapts
  m.adapt();// immediately take a snapshot of the background for differencing

  // List all the available serial ports to determine arduino serial port
  // println(Serial.list());
  myPort = new Serial(this, Serial.list()[0], 9600); // default serial port is [0]

  println("Myron " + m.version());
  noStroke();
}



// -------- DRAW -----------------------------------------------

void draw()
{
  m.trackColor(255,255,255,256*3-100);//track white
  m.update(); //update camera view

  background(0); // set background to black
```

```
// SERIAL CONNECTION
// listen to serial port and trigger serial event
while(myPort.available() > 0)
{
  value = myPort.read(); // read serial input and store in value
  if(value=='H')
  {
    mode=true; // switch to cleaning mode
    println("Value read from serial port is 'H' ");
  }
  else
  {
    // if serial event is 'L' mode is false
    mode=false; // switch to tracing mode
    println("Value read from serial port is 'L' ");
  }
}

// TRACING MODE
if(mode == false)
{
  int[][] centers = m.globCenters();//get the center points of blobs

  float avX=0; // variables for calculating center of movement
  float avY=0;

  for(int i=0;i<centers.length;i++)
  {
    float cx = centers[i][0];
    float cy = centers[i][1];

    if(cx!=0 && cy!=0) // if detected blob center is not in origin (JMyron always detects a
                       // blob there by default)
    {
      if(cx > offsetX && cx <= (offsetX+projW))
      {
        if(cy > offsetY && cy <= (offsetY+projH))
        {
          float centerX = map(cx-offsetX, 0, projW, 0, 800) + random(-2,2); // adjust x- and
          float centerY = map(cy-offsetY, 0, projH, 0, 600) + random(-2,2); // y-position to
                                                                            // output format

          Trace t = new Trace(random(3,8), centerX, centerY, color(random(60,150)));
           // create and initialize new Trace
          trackTraces.add(t); // store Trace object in ArrayList

          avX += centerX;
          avY += centerY;
        }
      }
    }
  }

  if(centers.length-1>0)
  {
    avX/=centers.length-1; // calculate centerpoint of movement
    avY/=centers.length-1;
  }

  if(!(avX==0&&avY==0)&&centers.length-1>0)
  {
    objdestx = avX;
    objdesty = avY;
  }

  objx += (objdestx-objx)/10.0f; // calculate speed and direction for
  objy += (objdesty-objy)/10.0f; // center of movement
```

```
    /*fill(30,100,0); // draw center of movement (for debugging purposes)
     ellipseMode(CENTER);
     ellipse(objx,objy,30,30);*/

    if (centers.length==0)
    {
      leaveTrace(trackTraces, 0, 0); // draw Traces with no center of movement
    }
    else
    {
      leaveTrace(trackTraces, objx, objy); // draw Traces
    }
  }

  // CLEANING MODE
  else if (mode == true)
  {
    m.sensitivity(20.0);
    int[][] polyCenters = m.globCenters(); //get the center points of blobs

    float avX=0; // variables for calculating center of movement
    float avY=0;

    if (polyCenters.length>1)
    {
      for( int i = 0; i<polyCenters.length; i++ )
      {
        float px = polyCenters[i][0];
        float py = polyCenters[i][1];

        if (px != 0.0 && py != 0.0)
        {
          if(px > offX && px <= (offX+projW))
          {
            if(py > offY && py <= (offY+projH))
            {
              float polyCenterX = map(px-offX, 0, projW, 0, 800); // adjust x- and y- position
              float polyCenterY = map(py-offY, 0, projH, 0, 600); // to output format
              Point p = new Point(int(polyCenterX),int(polyCenterY));
              polyTrack.add(p); // store new Point in polygon array

              avX += polyCenterX;
              avY += polyCenterY;
            }
          }
        }
      }

      int count = polyTrack.size();
      polygon = new Point[count];

      for(int k = 0; k < count; k++)
      {
        polygon[k] = (Point) polyTrack.get(k);
      }

      avX/=polyCenters.length-1; // calculate centerpoint of movement
      avY/=polyCenters.length-1;

      objdestx = avX;
      objdesty = avY;

      if (polyCenters.length == 0)
      {
        removeTrace(trackTraces,0,0);
      }
      else
      {
```

```
        removeTrace(trackTraces, objx, objy); // draw Traces
      }

      beginShape(POLYGON); // draw polygon

      // Collision detection
      for (int j=(trackTraces.size()-1); j>=0; j--)
      {
        Trace t = (Trace) trackTraces.get(j);
        boolean inside = insidePolygon(t.getX(),t.getY(), polygon); // determines whether
                                                                    // Trace object is inside polygon
        boolean intersect=false;

        for(int i = 0;i<polygon.length; i++)
        {
          intersect = inside || lineIntersectsCircle( polygon[i],
polygon[(i+1)%polygon.length], t.getCircle());
          //fill(255,0,0);
          //vertex( polygon[i].x, polygon[i].y ); // draw polygon on screen (for debugging
                                                  // purposes)
        }
        if(inside == true || intersect == true)
        {
          trackTraces.remove(j); // remove Trace object if it is inside or intersecting
                                 // polygon
        }
      }
      endShape(CLOSE);
      for(int p=0; p<polyTrack.size(); p++)
      {
        polyTrack.remove(p); // empty polygon ArrayList
      }
    }
    drawTrace(trackTraces); // draw traces
  }
}


// -------- FUNCTIONS ---------------------------------------------

// Drawing
void drawTrace(ArrayList blobsar) // draws Traces on screen
{
  for( int i=0; i<blobsar.size()-1; i++ )
  {
    Trace blobs = (Trace) blobsar.get(i);
    blobs.draw();
  }
}

void leaveTrace(ArrayList blobsar, float xp, float yp) // draws Traces on screen
{
  for( int i=0; i<blobsar.size()-1; i++ )
  {
    Trace blobs = (Trace) blobsar.get(i);
    blobs.update(xp,yp);
  }
}


void removeTrace(ArrayList blobsar, float xp, float yp) // draws Traces on screen
{
  for( int i=0; i<blobsar.size()-1; i++ )
  {
    Trace blobs = (Trace) blobsar.get(i);
    blobs.update2(xp,yp);
  }
}
```

```
// Collision detection
boolean insidePolygon ( float x, float y, Point [] p) // calculates if a Trace is within the
                                                      // polygon
{
  int i, j, c = 0;
  for (i = 0, j = p.length-1; i < p.length; j = i++)
  {
    if ((((p[i].y <= y) && (y < p[j].y)) ||
      ((p[j].y <= y) && (y < p[i].y))) &&
      (x < (p[j].x - p[i].x) * (y - p[i].y) / (p[j].y - p[i].y) + p[i].x))
      c = (c+1)%2;
  }
  return c==1;
}

boolean lineIntersectsCircle ( Point p1, Point p2, Circle c ) // calculates if a Trace
                                                              // intersects the polygon outline
{
  float ldx = p2.x-p1.x;
  float ldy = p2.y-p1.y;
  float lk = ldy / ldx;
  float ld = p1.y - p1.x * lk;

  float cd2 = c.y - c.x * (-1.0/lk); // parallel line through circle center

  float xx = (ld - cd2) / ((-1.0/lk)-lk);
  float yy = xx * lk + ld;

  if ( !(xx >= min( p1.x, p2.x ) & xx <= max( p1.x, p2.x ) & yy >= min( p1.y, p2.y ) & yy <=
max( p1.y, p2.y )) ) return false;

  float rr = sqrt((c.x-xx)*(c.x-xx)+(c.y-yy)*(c.y-yy));

  return c.rad >= rr;
}


// Settings
void mousePressed()
{
  m.settings();//click the window to get the settings
}

// Key events
void keyPressed()
{
  if ( key==' ' ) // clean screen
  {
    for(int i=0; i<trackTraces.size()-1; i++)
    {
      trackTraces.remove(i);
    }
  }
  if ( key=='m' ) // switch modes manually
  {
    if (mode == false)
    {
      mode = true;
    }
    else mode = false;
  }
}
```

```
public void stop()
{
  m.stop();//stop the object
  super.stop();
}


// -------- CLASS CIRCLE -----------------------------------------

class Circle
{
    float x, y, rad, dia;

    Circle ( float _x, float _y, float _rad )
    {
        x = _x;
        y = _y;
        rad = _rad;
        dia = 2.0 * _rad;
    }

    /*void drawCircle ()
    {
        fill( Colors.white );
        stroke( Colors.red );
        ellipse( x, y, dia, dia );
    }*/
}


// -------- CLASS TRACE -----------------------------------------

class Trace // class definition
{
  float radius; // variable for radius
  float xPos; // position in X
  float yPos; // position in Y
  float velX=0; // velocity in X
  float velY=0; // velocity in Y
  float threshold; // brightness threshold for fading
  color col; // current color
  int age = 0; // age of trace


  Trace(float r, float xp, float yp, color c)
  {
    radius = r; // transfer local to global
    xPos = xp ; // variables
    yPos = yp;
    col = c;
    threshold = brightness(c)/5.0;
  }

  float getX() // returns x-Position of Trace
  {
    return xPos;
  }

  float getY() // returns y-Position of Trace
  {
    return yPos;
  }

  Circle getCircle() // returns Circle of Trace
  {
    Circle c = new Circle(xPos,yPos,radius);
    return c;
  }
```

```
void draw() // draw Trace
{
  ellipseMode(CENTER);
  float curCol = brightness(col);

  if(age%5 == 0 && curCol >= threshold) // every 5th loop Trace gets darker
  {
    col = color(int(curCol)-1);
  }

  fill(col);
  ellipse(xPos,yPos,radius,radius); // draw ellipse
  age++;
}

void update(float mx, float my) // update coordinates and redraw particles
{
  /* Variables are used to keep track of the x and y coordinates of the movement. */
  int rx = int(mx);
  int ry = int(my);

  /* This if statement determines whether the movement is within the boundaries, then
     proceeds if so. */
  if ((rx > 0) && (rx < width) && (ry > 0) && (ry < height))
  {
    /* The radius variable stores the distance between the movement and the particle. */
    float rad = distance(xPos,yPos,rx,ry);
    /* Proceed if the particle is within 150 pixels of the movement. */
    if (rad < 150)
    {
      /* atan2 is used to find the angle between the movement and the particle. */
      float angle = atan2(yPos-ry,xPos-rx);

      velX -= (150 - rad) * 0.0001 * cos(angle + (0.7 + 0.0005 * (150 - rad)));
      velY -= (150 - rad) * 0.0001 * sin(angle + (0.7 + 0.0005 * (150 - rad)));

    }
  }

  /* x and y are increased by our velocities. This completes our formula c + r * cos(a) or
     sin(a), with vx/vy being the r * cos(a) or sin(a) */
  xPos += velX;
  yPos += velY;

  /* The velocities are decreased by 3% to simulate friction. */
  velX *= 0.97;
  velY *= 0.97;

  /* We check if the particles are beyond our borders, and add/subtract to the velocities
     accordingly. */

  if (xPos > width) velX -= 1;
  else if (xPos < 0) velX += 1;
  else if (yPos > height) velY -= 1;
  else if (yPos < 0) velY += 1;
  else
  {
    draw();
  }
}

void update2(float mx, float my) // update coordinates and redraw particles
{
  /* Variables are used to keep track of the x and y coordinates of the movement. */
  int rx = int(mx);
  int ry = int(my);
```

```
    /* This if statement determines whether the movement is within the boundaries, then
       proceeds if so. */
    if ((rx > 0) && (rx < width) && (ry > 0) && (ry < height))
    {
      /* The radius variable stores the distance between the movement and the particle. */
      float rad = distance(xPos,yPos,rx,ry);
      /* Proceed if the particle is within 300 pixels of the movement. */
      if (rad < 300)
      {
        /* atan2 is used to find the angle between the movement and the particle. */
        float angle = atan2(yPos-ry,xPos-rx);

        velX -= 0.001 * cos(angle + (0.7 + 0.0005 * (150 - rad)));
        velY -= 0.001 * sin(angle + (0.7 + 0.0005 * (150 - rad)));
      }
    }

    /* x and y are increased by our velocities. This completes our formula c + r * cos(a) or
       sin(a), with vx/vy being the r * cos(a) or sin(a) */
    xPos += velX;
    yPos += velY;

    /* The velocities are decreased by 3% to simulate friction. */
    velX *= 0.97;
    velY *= 0.97;



    /* We check if the particles are beyond our borders, and add/subtract to the velocities
       accordingly.     */


    if (xPos > width) velX -= 1;
    else if (xPos < 0) velX += 1;
    else if (yPos > height) velY -= 1;
    else if (yPos < 0) velY += 1;
    else
    {
      draw();
    }
  }
}

// -------- FUNCTION ----------------------------------------------
/* The distance formula. Used for finding the distance between two points (x1,y1,x2,y2) */
float distance(float x,float y,float cx,float cy)
{
  return sqrt(sq(cx - x) + sq(cy - y));
}
```

TRACES

Arduino Code

```
/*
 ---------------------------------------------------------------
 TRACES
 ---------------------------------------------------------------

 IR camera tracking combined with on floor projection.
 An overhead camera tracks people and their movement in a specified area
 the detected movement is projected back onto the space as traces left behind.

 Unlike the traces we leave behind online the projected traces can be removed
 by a special broom.

 The set-up:
 * IR-sensitive surveillance camera with integrated IR-LEDs
 * Projector
 * a broom
 * 4 bright white LEDs to be used as tracking markers at the edges of the broom
 * a push button to switch between tracing and cleaning mode

 The Arduino circuit:
 * the 4 LEDs are connected in parallel to Ground and Pin 13
 * the push button pins are connected to 5V and Ground, the signal to Arduino Pin 2

 The surveillance camera as well as the projector is mounted on the ceiling.
 The tracking LEDs are attached to the corners of the broom so that the light of the
 LEDs can be seen from above and tracked accordingly. The push button is attached to
 the broom stick to enable switching between the two modes.
 Push button and LEDs are set up in Arduino and connected to Processing via the serial
 port.

 Please see trace_processing for further information on the Processing part of the project.

 Created November 10 2009
 By Katja Knecht & Yuan Qi

 References:
 Arduino Tutorial: Switch
 http://www.arduino.cc/en/Tutorial/Switch

 Arduino meets processing and pa_PushButton by Melvin Ochsmann for Malmoe University
 http://webzone.k3.mah.se/projects/arduino-
workshop/projects/arduino_meets_processing/instructions/pushbutton.html

 */
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

// -------- VARIABLES ------------------------------------------------------

int inPin = 2;         // the number of the input pin
int outPin = 13;       // the number of the output pin

int state = HIGH;      // the current state of the output pin
int reading;           // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pin

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers
```

```
// --------- SET-UP ------------------------------------------------------------

void setup()
{
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}


// --------- LOOP --------------------------------------------------------------

void loop()
{
  reading = digitalRead(inPin);

  // if the input just went from LOW and HIGH and we've waited long enough
  // to ignore any noise on the circuit, toggle the output pin and remember
  // the time
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    if (state == HIGH){
      state = LOW;
      // print the results to the serial monitor:
      Serial.write('L');
    }
    else {
      state = HIGH;
      // print the results to the serial monitor:
      Serial.write('H');
    }

    time = millis();
  }

  digitalWrite(outPin, state);

  previous = reading;
}
```