PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

MODELO HEURÍSTICO PARA LA DETERMINACIÓN DE LA MOTILIDAD EN CÉLULAS ESPERMÁTICAS MEDIANTE EL ANÁLISIS AUTOMÁTICO DE TRACKING EN VIDEO

ANEXOS

Anexo 1: Simulador

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

body {
  width: 800px;
  height: 400px;
  background: #000;
  border-left: 1px solid #900;
  border-right: 1px solid #900;
  border-top: 1px solid #900;
  border-bottom: 1px solid #900;

}
ellipse {
  fill: #fff;
}

path {
  fill: none;
  stroke: #fff;
  stroke-linecap: round;
}

.mid {
  stroke-width: 4px;
}

.tail {
  stroke-width: 2px;
}
</style>

var width = 800,
height = 400;

var espermasLineales  = 10,
```

```
espermasSinusoidales = 10,
espermasCirculares = 10,
longitudCola = 12,
time =0,
degrees = 360/(2*Math.PI),
frames =0,
framesTotales=1500;

var spermatozoaLineal = d3.range(espermasLineales).map(function() {
var y = Math.random() * height,
x = Math.random() * width;

return {
vx: Math.random()*2 -1 ,
amplitud : Math.random()*20 + 20,
vy: Math.random()*2 -1 ,
x0: x,
y0: y,
difx :1,
dify :1,
path: d3.range(longitudCola).map(function() { return [x, y]; }),
trajectorie: d3.range(framesTotales).map(function(){return [x,y];}),
trajectorieFin: 0,
position : [x,y],
count: 0
};
});

var spermatozoaSeno = d3.range(espermasSinusoidales).map(function() {
var y = Math.random() * height,
x = Math.random() * width;

return {
vx: Math.random()*2 -1 ,
amplitud : Math.random()*20 + 20,
vy: Math.random()*2 -1 ,
x0: x,
y0: y,
difx :1,
dify :1,
path: d3.range(longitudCola).map(function() { return [x, y]; }),
```

```
trajectorie: d3.range(framesTotales).map(function(){return [x,y];}),
trajectorieFin: 0,
position : [x,y],
count: 0
};
});

var spermatozoaCir = d3.range(espermasCirculares).map(function() {
var y = Math.random() * height,
x = Math.random() * width;

return {
vx: Math.random()*2 -1 ,
amplitud : Math.random()*20 + 20,
vy: Math.random()*2 -1 ,
x0: x,
y0: y,
difx :1,
dify :1,
path: d3.range(longitudCola).map(function() { return [x, y]; }),
trajectorie: d3.range(framesTotales).map(function(){return [x,y];}),
trajectorieFin: 0,
position : [x,y],
count: 0
};
});

var svg = d3.select("body").append("svg")
.attr("width", width)
.attr("height", height);

var union = spermatozoaLineal.concat(spermatozoaSeno,spermatozoaCir);
var g = svg.selectAll("g").data(union).enter().append("g");

var head = g.append("ellipse").attr("rx", 6.5).attr("ry", 4);

g.append("path").datum(function(d) { return d.path.slice(0, 3); })
.attr("class", "mid");

g.append("path").datum(function(d) { return d.path; }).attr("class",
"tail");
```

```
var tail = g.selectAll("path");

d3.timer(function() {
      time = time +2;
      for (var i = -1; ++i < spermatozoaLineal.length;) {
            var spermatozoon = spermatozoaLineal[i],
            path = spermatozoon.path,
            dx = spermatozoon.vx,
            dy = spermatozoon.vy,
            posAX = spermatozoon.position[0],
            posAY =spermatozoon.position[1];

            spermatozoon.dx=dx;
            spermatozoon.dy=dy;

            //lineal
            x= spermatozoon.position[0]=spermatozoon.x0 + dx*time,
            y = spermatozoon.position[1]=spermatozoon.y0 + dy*time;

            spermatozoon.difx = spermatozoon.position[0]-posAX;
            spermatozoon.dify = spermatozoon.position[1]-posAY;

            //off the walls.
            if (x < 0 || x > width) {
                  spermatozoon.vx = 0;
                  spermatozoon.x0=width+10;
                  spermatozoon.vy=0;
                  spermatozoon.y0=height+10;
                  if                  (spermatozoon.trajectorieFin==0){
                        spermatozoon.trajectorieFin=frames;
                  }
            }

            if (y < 0 || y > height) {
                  spermatozoon.vx = 0;
                  spermatozoon.x0=width+10;
                  spermatozoon.vy=0;
                  spermatozoon.y0=height+10;
                  if (spermatozoon.trajectorieFin==0){
                        spermatozoon.trajectorieFin=frames;
```

```
            }
        }

        path[0][0] = x;
        path[0][1] = y;

        if (spermatozoon.trajectorieFin==0){
            spermatozoon.trajectorie[frames][0] =x;
            spermatozoon.trajectorie[frames][1] =y;
        }

        speed = Math.sqrt(dx * dx + dy * dy),
        count = speed * 10,
        k1 = -5;




        for (var j = 0; ++j < longitudCola;) {
            var vx = x - path[j][0],
            vy = y - path[j][1],
            k2 = Math.sin(((spermatozoon.count+=10)+ j*3)/300)/
            speed;

            path[j][0] = (x += dx / speed * k1) - dy * k2;
            path[j][1] = (y += dy / speed * k1) + dx * k2;
            speed = Math.sqrt((dx = vx) * dx + (dy = vy) * dy);
        }
}

for (var i = -1; ++i < spermatozoaSeno.length;) {
        var spermatozoon = spermatozoaSeno[i],
        path = spermatozoon.path,
        dx = spermatozoon.vx,
        dy = spermatozoon.vy,
        posAX = spermatozoon.position[0],
        posAY =spermatozoon.position[1];
        //sinusoideal
        x= spermatozoon.position[0] = spermatozoon.x0 + (dx*time),
        y= spermatozoon.position[1] = spermatozoon.y0 +
```

```
Math.sin(x/15)*spermatozoon.amplitud;

spermatozoon.difx = spermatozoon.position[0]-posAX;
spermatozoon.dify = spermatozoon.position[1]-posAY;

//off the walls.
if (x < 0 || x > width) {
      spermatozoon.vx = 0;
      spermatozoon.x0=width+10;
      spermatozoon.vy=0;
      spermatozoon.y0=height+10;
      if (spermatozoon.trajectorieFin==0){
            spermatozoon.trajectorieFin=frames;
      }
}
if (y < 0 || y > height) {
spermatozoon.vx = 0;
spermatozoon.x0=width+10;
spermatozoon.vy=0;
spermatozoon.y0=height+10;
if (spermatozoon.trajectorieFin==0){
      spermatozoon.trajectorieFin=frames;}
}

path[0][0] = x;
path[0][1] = y;
if (spermatozoon.trajectorieFin==0){
spermatozoon.trajectorie[frames][0] =x;
spermatozoon.trajectorie[frames][1] =y;
}

speed = Math.sqrt(dx * dx + dy * dy),
count = speed * 10,
k1 = -5;

for (var j = 0; ++j < longitudCola;) {
      var vx = x - path[j][0],
      vy = y - path[j][1],
      k2 = Math.sin(((spermatozoon.count+=10)+j*3)/300)/
      speed;
      path[j][0] = (x += dx / speed * k1) - dy * k2;
```

```
                path[j][1] = (y += dy / speed * k1) + dx * k2;
                speed = Math.sqrt((dx = vx) * dx + (dy = vy) * dy);
        }
}
for (var i = -1; ++i < spermatozoaCir.length;) {
        var spermatozoon = spermatozoaCir[i],
        path = spermatozoon.path,
        dx = spermatozoon.vx,
        dy = spermatozoon.vy,
        posAX = spermatozoon.position[0],
        posAY =spermatozoon.position[1];

        //circular
        x= spermatozoon.position[0] = spermatozoon.x0 +
        Math.cos(dx*time/15)*spermatozoon.amplitud,
        y = spermatozoon.position[1] = spermatozoon.y0 +
        Math.sin(dy*time/15)*spermatozoon.amplitud;
        spermatozoon.difx = spermatozoon.position[0]-posAX;
        spermatozoon.dify = spermatozoon.position[1]-posAY;

        //off the walls.
        if (x < 0 || x > width) {
                spermatozoon.vx = 0;
                spermatozoon.x0=width+10;
                spermatozoon.vy=0;
                spermatozoon.y0=height+10;
                if (spermatozoon.trajectorieFin==0){
                        spermatozoon.trajectorieFin=frames;
                }
        }
        if (y < 0 || y > height) {
        spermatozoon.vx = 0;
        spermatozoon.x0=width+10;
        spermatozoon.vy=0;
        spermatozoon.y0=height+10;
        if (spermatozoon.trajectorieFin==0){
                spermatozoon.trajectorieFin=frames;}
        }

        if (spermatozoon.trajectorieFin==0){
                spermatozoon.trajectorie[frames][0] =x;
```

```
                        spermatozoon.trajectorie[frames][1] =y;
                }
                path[0][0] = x;
                path[0][1] = y;
                speed = Math.sqrt(dx * dx + dy * dy),
                count = speed * 10,
                k1 = -5;
                for (var j = 0; ++j < longitudCola;) {
                        var vx = x - path[j][0],
                        vy = y - path[j][1],
                        k2 = Math.sin(((spermatozoon.count+=10)+j*3)/300)/
                        speed;
                        path[j][0] = (x += dx / speed * k1) - dy * k2;
                        path[j][1] = (y += dy / speed * k1) + dx * k2;
                        speed = Math.sqrt((dx = vx) * dx + (dy = vy) * dy);
                }
        }
head.attr("transform", headTransform);
tail.attr("d", tailPath);
frames = frames +1;

if(frames==1500){

        var  A  =  [["Sperm  Number","VSL","VCL","LIN","Classification"]];
        // initialize array of rows with header row as 1st item

        for (var x=0;x<union.length;x++){
                var sperm = union[x];
                var trayectoria = sperm.trajectorie;
                var VSL;
                var VCL;
                var LIN;
                var clase;
                var iniciox;
                var inicioy;
                var finx;
                var finy;
                var globlalDistance=0;

                if ( sperm.trajectorieFin!=0){
                        //El esperma a salido de la pantalla
```

```
                iniciox=trayectoria[0][0];
                inicioy=trayectoria[0][1];
                finx=trayectoria[sperm.trajectorieFin -1][0];
                finy=trayectoria[sperm.trajectorieFin -1][1];
                for(var y=0;y<sperm.trajectorieFin -1;y++){
                      var localDist = lineDistance(
                      trayectoria[y][0],trayectoria[y][1],
                      trayectoria[y+1][0],trayectoria[y+1][1]);
                globlalDistance = globlalDistance + localDist;
                }
          VCL = globlalDistance/sperm.trajectorieFin;
          VSL= lineDistance(iniciox,inicioy,finx,finy)/
          sperm.trajectorieFin;
          }
          else{
                //El esperma no ha salido de la pantalla
                iniciox=trayectoria[0][0];
                inicioy=trayectoria[0][1];
                finx=trayectoria[frames-1][0];
                finy=trayectoria[frames-1][1];

                for(var y=0;y< frames-1 ;y++){
                      var localDist = lineDistance(trayectoria[y][0],
                      trayectoria[y][1], trayectoria[y+1][0],
                      trayectoria[y+1][1]);
                      globlalDistance = globlalDistance + localDist;
                }
                VCL = globlalDistance/frames;
                VSL = lineDistance(iniciox,inicioy,finx,finy)/frames;
          }
          LIN = VSL/VCL;

          clase = clasificador(VSL*30*0.7,LIN);
          A.push([x,VSL,VCL,LIN,clase])
    }


    var csvRows = [];

    for(var i=0,l=A.length; i<l; ++i){
          csvRows.push(A[i].join(','));   // unquoted CSV row
```

```
        }
        var csvString = csvRows.join("\r\n");
        var a = document.createElement('a');
        a.innerHTML = "Click here";
        a.href     = 'data:application/csv;charset=utf-8,' +
        encodeURIComponent (csvString);
        a.target   = '_blank';
        a.download = 'Muestra_30o_25s.csv';
        document.body.appendChild(a);
        }
});

function clasificador(vsl,lin){
        if (vsl>=23.0 && lin>=0.58) return "1";
        if ( vsl>10.0 && vsl<23.0 && lin >=0.58) return "2";
        if ( vsl >10.0 && lin <0.58) return "3";
        if (vsl <= 10.0) return "4";
        return "No hay clasificacion";
}

function headTransform(d) {
        return   "translate("   +   d.position[0]+","+d.position[1]   +")
        rotate(" + Math.atan2(d.dify, d.difx) * degrees + ")";
}

function tailPath(d) {
        return "M" + d.join("L");
}

function lineDistance( iniciox,inicioy,finx,finy){
        var xs = 0;
        var ys = 0;
        xs = iniciox - finx;
        xs = xs * xs;
        ys = inicioy -finy;
        ys = ys * ys;
        return Math.sqrt( xs + ys );
}
</script>
```

Anexo 2:

Pre-procesamiento y Detección de células espermáticas

```
Mat img_proc(Mat img,int a){
	Mat gray;
	cvtColor( img, gray, CV_BGR2GRAY );
	Mat thresh;
	threshold(gray,thresh,127,255,0);
	Mat kernel;
	kernel = Mat::ones(3,3,CV_32F);
	erode(thresh,thresh,kernel,Point(-1,-1),a);
	return thresh;
}

int main(int argc, _TCHAR* argv[])
{

VideoCapture cap(file_entrada);

Mat old_frame;

vector< vector < Point > > contours;

cap.read(old_frame);

Mat old_gray = img_proc(old_frame,2);

//We must find contours in the image processed

findContours(old_gray, contours,CV_RETR_LIST,CV_CHAIN_APPROX_SIMPLE,Point());

Mat mask1;

//copy the image and initialize to zeros (black)

old_frame.copyTo(mask1);

mask1 = Scalar(0,0,0);
```

```
//Draw the contours finded and show how many were they;

for (int j=0;j<contours.size();j++){
	for (int i=0;i<contours.at(j).size();i++){

		Point a = contours.at(j).at(i);
		Point b = contours.at(j).at( (i+1)%contours.at(j).size());
		int x0=a.x;
		int y0 = a.y;
		int x1 = b.x;
		int y1 = b.y;
		line(mask1,a,b,Scalar(255,255,255),1);

	}
}

mask1=Scalar(255,255,255);

Moments moment;

vector<Point2f> prevPoint;


for (int i=0;i<contours.size();i++){

	moment= moments(contours.at(i));
	if (moment.m00==0) continue;
	double cx = moment.m10/moment.m00;
	double cy = moment.m01/moment.m00;
	prevPoint.push_back(Point2f(cx,cy));
	circle(mask1,Point(cx,cy),5,Scalar(120,120,120));

}

} //main
```

Anexo 3:

Modelo Heurístico y Optical Flow

```
vector < vector<Point2f> >addTrajectories(vector <vector<Point2f> > trajectories,
vector<Point2f> prevPoint,vector<Point2f> nextPoint){
	if (trajectories.empty()){
		for (int i=0;i<prevPoint.size();i++){
			vector <Point2f> newTrajectories;
			Point2f a = prevPoint.at(i);
			Point2f b = nextPoint.at(i);
			newTrajectories.push_back(Point2f(a));
			newTrajectories.push_back(Point2f(b));
			trajectories.push_back(newTrajectories);
		}
	}
	else {
		for(int i=0;i<trajectories.size();i++){
			trajectories.at(i).push_back(Point2f(nextPoint.at(i)));
		}
	}

	return trajectories;
}

float euclidDist(Point2f a, Point2f b){

	return sqrt((b.x - a.x)*(b.x - a.x) + (b.y - a.y)*(b.y - a.y));

}
float angle(Point2f a, Point2f b,Point2f c, Point2f d){

	//vector AB

	Point2f vectorAB = Point2f(b.x-a.x,b.y-a.y);
	Point2f vectorCD = Point2f(d.x-c.x,d.y-c.y);

	float denominator = sqrt( (vectorAB.x*vectorAB.x) +
	(vectorAB.y*vectorAB.y)	)	*sqrt((vectorCD.x*vectorCD.x)	+
	(vectorCD.y*vectorCD.y));
```

```
        float numerator = (vectorAB.x*vectorCD.x) + (vectorAB.y*vectorCD.y);
        float epsilon = 0.001f;
        if ( abs(denominator) < epsilon ) return 0;
        else {
               return acos(numerator/denominator);
        }
}

float heuristicFunction(Point2f pointA, Point2f pointB, Point2f pointC){

        float distValue = euclidDist(pointB,pointC);

        float angleValue = angle(pointA,pointB,pointB,pointC);

        return distValue+angleValue;

}



vector < vector<Point2f> >  addTrajectoriesHeuristic(vector <vector<Point2f> >
trajectories, vector<Point2f> prevPoint,vector<Point2f> nextPoint){

        int sizeA = prevPoint.size();
        int sizeB = nextPoint.size();

        if (trajectories.empty()){
               for (int i=0;i<prevPoint.size();i++){
                //Find the point in nextPoint that suits the Heuristic Function.
                      int index=0;
                      float heuristicGlobalValue=FLT_MAX;
                      float heuristicLocalValue;
                      for (int j=0;j<nextPoint.size();j++){
                             heuristicLocalValue= heuristicFunction(
                             prevPoint.at(i), prevPoint.at(i), nextPoint.at(j));

                             if (heuristicLocalValue < heuristicGlobalValue) {
                                    heuristicGlobalValue = heuristicLocalValue;
                                    index=j;
```

```
                        }
                }
        //Join point (prev to next selected) to form trajectory and delete
        //the point selected
        vector <Point2f> newTrajectories;
        Point2f a = prevPoint.at(i);
        Point2f b = nextPoint.at(index);
        newTrajectories.push_back(Point2f(a));
        newTrajectories.push_back(Point2f(b));
        trajectories.push_back(newTrajectories);
        //delete the point selected only if prevPoints size <= nextPoint
        //size (A<=B)
        if (sizeA<=sizeB)
        if (!nextPoint.empty()) nextPoint.erase(nextPoint.begin()+index);

        }
}
else {
        for(int i=0;i<trajectories.size();i++){
                vector<Point2f>::iterator it = trajectories.at(i).end();
                it--;
                Point2f B = Point2f(*it);
                it--;
                Point2f A = Point2f(*it);
                //Vector AB gives the direction
                //B is the last element in the trajectory, there is a match
                //in prevPoint, find it

                vector<Point2f>::iterator itk;
                itk = find (prevPoint.begin(), prevPoint.end(),B);
                if (itk!=prevPoint.end()){
                        //found it, apply heuristic
                        int index=-1;
                                float heuristicGlobalValue=FLT_MAX;
                                float heuristicLocalValue;
                                for (int j=0;j<nextPoint.size();j++){
                                        heuristicLocalValue=
                                        heuristicFunction(A,B, nextPoint.at(j));
                                if (heuristicLocalValue<heuristicGlobalValue)
                                {
                                heuristicGlobalValue = heuristicLocalValue;
                                index=j;
```

```
                        }
                    }
                if (heuristicGlobalValue>10.0) {
                    //the point selected is too far away, probably the
                    //trayectory is already finish.
                        index =-1;
                    }

                    //Join point (prev to next selected) to form
                    //trajectory and delete the point selected
                if (index!=-1)
                trajectories.at(i).push_back( nextPoint.at(index));

                if (sizeA<=sizeB){
                    if (index!=-1 && !nextPoint.empty())
                        nextPoint.erase(nextPoint.begin()+index);
                    }
                }
                else {
                    //not in prevPoint
                    continue;
                }
            }
        }

        return trajectories;
}

pair< vector <double>,vector<Point2f> > motilityValues( vector<Point2f>
trajectorie){
        //0: VCL
        //1: VSL
        //2: VAP
        //3: LIN
        pair< vector <double>,vector<Point2f> > rpt;
        vector<double> values;
        int time=1;
        double totalSpeed=0;
        vector<Point2f> truePath;

        for (int i=1;i<trajectorie.size();i++){
            double speed;
```

```
        Point2f a = trajectorie.at(i-1);
        Point2f b = trajectorie.at(i);
        speed = euclidDist(a,b);
        if (speed>0.001){
        time++;
        totalSpeed+=speed;
               if(truePath.empty()){
                      truePath.push_back(Point2f(a));
                      truePath.push_back(Point2f(b));
               }
               else {
                      truePath.erase(truePath.begin()+(truePath.size()-1));
                      truePath.push_back(Point2f(a));
                      truePath.push_back(Point2f(b));

               }
        }
}



//this are in Pixel/frame
double vcl = totalSpeed/time;
values.push_back(vcl);

Point2f start = trajectorie.at(0);
Point2f end = trajectorie.at((int)trajectorie.size()-1);
float linearDistance= euclidDist(start,end);
double vsl = linearDistance/time;
values.push_back(vsl);

vector<Point2f> vapPath;
float x=0;
float y=0;
if                                                    (!truePath.empty())
vapPath.push_back(Point2f(truePath.at(0).x,truePath.at(0).y));

for (int i=0;i<truePath.size();i++){

       if ( (i+1)%6==0 ){
       vapPath.push_back(Point2f(truePath.at(i).x,truePath.at(i).y));
       }
```

```
        }

        if (!truePath.empty()) vapPath.push_back(Point2f(truePath.at(
                (int)truePath.size()-1 ).x,truePath.at((int)truePath.size()-1).y));

        double totalSpeedVap=0;

        for (int i=1;i<vapPath.size();i++){
                double speed;
                Point2f a = vapPath.at(i-1);
                Point2f b = vapPath.at(i);
                speed = euclidDist(b,a);
                totalSpeedVap+=speed;
        }

        double vap =totalSpeedVap/time;

        values.push_back(vap);

        double lin = vsl/vcl;
        values.push_back(lin);

        rpt.first=vector<double>(values);
        rpt.second=vector<Point2f>(vapPath);
        return rpt;
}

string Clasificador(double vsl,double lin){

        if (vsl>=23.0 && lin>=0.58){
                return "1";
        }
        if ( vsl>10.0 && vsl<23.0 && lin >=0.58) return "2";
        if ( vsl >10.0 && lin <0.58) return "3";
        if (lin < 0) return "nan";
        if (vsl <= 10.0) return "4";
}



int main(int argc, _TCHAR* argv[])
```

```
{
/*PREPROCCESS ALGORITHM*/

vector<Point2f> prevPointHeuristic(prevPoint);

Mat mask;
Mat traj;
Mat trajHeuristic;
//in order to copy the size
old_frame.copyTo(mask);
old_frame.copyTo(traj);
old_frame.copyTo(trajHeuristic);

//create white image
mask = Scalar(0,0,0);
traj = Scalar(0,0,0);
trajHeuristic = Scalar(0,0,0);

Mat frame;
Mat frame_gray;
Mat imageFlow;
vector< vector<Point2f> > trajectories;
vector< vector<Point2f> > trajectoriesHeuristic;

while(cap.read(frame)){
	vector<Point2f> nextPoint;
	vector<Point2f> nextPointHeuristic;
	vector< vector < Point > > contoursHeuristic;
	Moments momentHeuristic;
	vector<uchar> status;
	vector<float> error;

	frame_gray= img_proc(frame,2);
	Mat copy_old;
	old_gray.copyTo(copy_old);
	Mat copy_frame;
	frame_gray.copyTo(copy_frame);

	Mat copy_old_heuristic;
	old_gray.copyTo(copy_old_heuristic);
	Mat copy_frame_heuristic;
	frame_gray.copyTo(copy_frame_heuristic);
```

```
calcOpticalFlowPyrLK(copy_old,copy_frame,prevPoint,nextPoint,status,error);

findContours(copy_old_heuristic,contoursHeuristic,CV_RETR_LIST,
CV_CHAIN_APPROX_SIMPLE,Point());

for (int i=0;i<contoursHeuristic.size();i++){
        momentHeuristic= moments(contoursHeuristic.at(i));
        if (momentHeuristic.m00==0) continue;
        double cx =  momentHeuristic.m10/momentHeuristic.m00;
        double cy = momentHeuristic.m01/momentHeuristic.m00;
        nextPointHeuristic.push_back(Point2f(cx,cy));
        circle(mask1,Point(cx,cy),5,Scalar(120,120,120));
}
for (int i=0;i<prevPoint.size();i++){
        Point2f a = prevPoint.at(i);
        Point2f b = nextPoint.at(i);
        line(mask,a,b,Scalar(255,255,255),1);
}

trajectoriesHeuristic  =    addTrajectoriesHeuristic(trajectoriesHeuristic,
prevPointHeuristic, vector<Point2f>(nextPointHeuristic));

trajectories = addTrajectories(trajectories,prevPoint,nextPoint);
cv::imshow("OpticalFlow",mask);
waitKey(50);
old_gray=frame_gray;
prevPoint=vector<Point2f>(nextPoint);
prevPointHeuristic=vector<Point2f>(nextPointHeuristic);
}


pair< vector <double>,vector<Point2f> > valores;

//output .csv file
ofstream outdata;
outdata.open(file_salida, ios::app);

//Optical Flow
outdata << "OPTICAL FLOW,"<<trajectories.size()<<endl;

for (int i=0;i<trajectories.size();i++){
```

```
            for (int j=0;j<(int)trajectories.at(i).size()-1;j++){
            line(traj,trajectories.at(i).at(j),trajectories.at(i).at(j+1),
            Scalar(120,120,120),2);
            }
            cv::imshow("OpticalFlow",traj);
            valores = motilityValues(trajectories.at(i));
            // in px/frame
            //convertion factors
            // um/px =  0.7um/1px
            // frame/s = 30fr/s;


            double VSL,LIN;
            //apply convertion factors
            VSL = valores.first.at(1)*30*0.7;
            LIN = valores.first.at(3);
            string grado = Clasificador(VSL,LIN);
            outdata << grado << endl;
      }
      // Heuristic
      outdata << "Heuristic," <<trajectoriesHeuristic.size()<<endl;
      for (int i=0;i<trajectoriesHeuristic.size();i++){
            for (int j=0;j<(int)trajectoriesHeuristic.at(i).size()-1;j++){
                  line(trajHeuristic,trajectoriesHeuristic.at(i).at(j),
                  trajectoriesHeuristic.at(i).at(j+1),Scalar(120,120,120),2);
            }
            cv::imshow("Heuristic",trajHeuristic);
            valores = motilityValues(trajectoriesHeuristic.at(i));
            double VSL,LIN;
            //apply convertion factors
            VSL = valores.first.at(1)*30*0.7;
            LIN = valores.first.at(3);
            string grado = Clasificador(VSL,LIN);
            outdata << grado << endl;
      }
return 0;
}
```