Object Tracking Report

Group 2 VT 2013

Version 0.1

Status

Reviewed	Group 2	2013–03–19
Approved		

2013–03–19

PROJECT IDENTITY

Computer Vision, VT 2013, group 2
Department of Electrical Engineering (ISY), Linköping University

Participants

Name	Responsibilities	Phone	E-mail
Gustav Häger	Background modelling		gusha124@student.liu.se
Alexander Sjöholm	Kalman prediction, Evaluation of results		alesj050@student.liu.se
Martin Svensson	Foreground segmentation, Documentation	070–289 01 49	marsv106@student.liu.se
Mattias Tiger	Object identification		matti166@student.liu.se

E-mail list to the group: marsv106@student.liu.se

Project supervisor: Fahad Khan, Linköping University, fahad.khan@liu.se
Course leader: Per-Erik Forssén, 013–28 56 54, per-erik.forssen@liu.se

Contents

List of Figures

Document history

Version	Date	Changes	Sign	Reviewed
0.1	2013–03–19	Initial draft	MS	

1 Introduction

This document is the final documentation of mini-project performed as a part od the course Computer Vision (TSBB15) at Linköping university. The project goal was to create an application that can, in real-time, detect motion in an image sequence from e.g. a surveillance camera. The application was implemented in C++ using the OpenCV API.

The purpose of this document is to provide a thorough description of how the application was created and the problems encountered during the implementation work.

2 System description

The system implementation is divided into four modules, Background modeling, Foreground segmentation, Object identification and Kalman prediction.

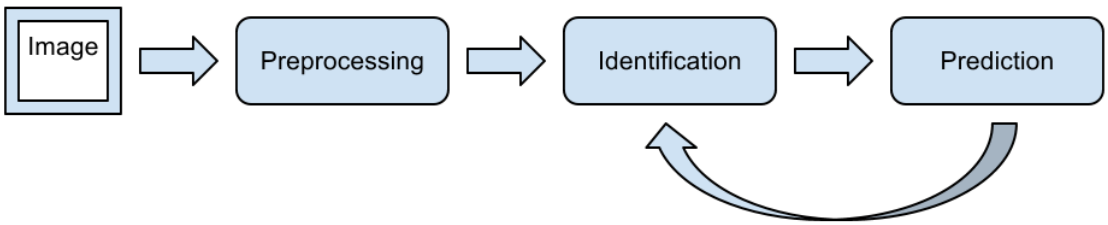

Figure 2.1: *Data flow between modules.*

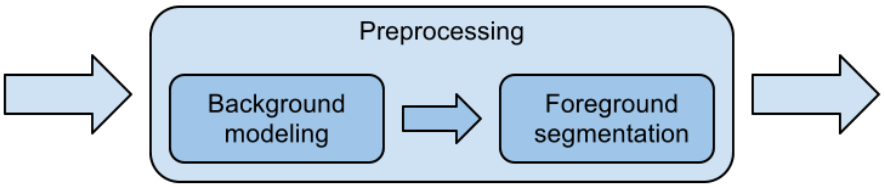

Figure 2.2: *Data flow within preprocessing block.*

2.1 Main Program

The main program is composed of two parts. The first one is initiation part, where the modules are declared, parameters specified, and movie clip loaded. The second one is the main loop, where the actual program is executed.

2.1.1 Main loop

When the program is started and the initiation is done the program is set to run until the end of the specified movie sequence. For each frame in the sequence, the Background model is updated and a probability map matrix containing information about what pixels are part of the background is created. After the probability map has been created, the foreground processing module is called upon to perform some noise removal and to detect all the interesting regions in the probability map (the regions that are likely to be part of the foreground).

Once all the interesting regions are labeled, the identification module takes all the created objects in the current frame and associates them with the proper ID. This is done by comparison with the objects in the previous frame.

To help in the labeling process we use a Kalman predictor, that predicts the position of the each object in the next frame. The Prediction module is called upon after the identification is done.

Once all the processing in the module is finished the objects in the current frame are drawn in the current frame as bounding boxes with velocity vectors, position in pixel coordinates and ID numbers.

See code in appendix F.

Figure 2.3: *Some flowchart of main program.*

2.2 Data structures

In order to keep a good structure on the program and a logical separation of functionalities, several data structures have been created.

Figure 2.4: *UML diagram of the data structures.*

2.2.1 ProbabilityMap

DErpa derpa derpa derpa. DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa. DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa.DErpa derpa derpa derpa. DErpa derpa derpa derpa. See code in appendix B.2. [1]

2.2.2 Object

The *Object* class represents a moving object in the scene. The information stored about the objects is ID, position, velocity, and bounding box. This information is what is processed in the object identification and prediction models.

2.2.3 Frame

The *Frame* class contains the current image as well as the probability map. Both if these are stored as *cv::Mat*. The image is used for creation of the probability map as well as drawing the bounding boxes. From the probability map The foreground processing module finds and creates Objects that are stored in a vector. To draw the objects and their bounding boxes in the image, the function *DrawObject* is called with the appropriate color. 2.4.

2.2.4 FrameList

The *FrameList* class manages data sources (video) and provides frames sequentially as well as a history of previous frames. It contains methods for displaying the current frame and various interesting/useful information.

2.3 Background modeling

How go i BGModel?? How go i BGModel??

2.3.1 Expectation maximization

How Do i EM? See code in appendix B.

Figure 2.5: *Background modeling figure.*

2.3.2 Moar info

Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, Moar info, This can be seen in figure 2.5.

2.4 Foreground segmentation

The Foreground segmentation is performed by the foreground processor module. The purpose of the foreground processor is to, from the probability map, decide which regions are interesting and create an object for each interesting region, and add these to the object vector of the frame.

The foreground processing is done by the foreground processor and there are two different algorithms implemented at the moment, one for real-time processing and the other for off-line processing. Which one of these that is to be used is specified by the algorithm enumerator during the initiation of the foreground processor object.

The foreground processor is initiated by picking one algorithm and setting appropriate parameter values using the *init* command. After being initiated, the foreground processor is called using the *segmentForeground* command, which takes the current frame as an input. For more info, see appendix C.

2.4.1 Morphological cleanup

The first step in the foreground segmentation process is to threshold the probability map at some appropriate value and create a binary image where everything but the foreground objects are set to zero. By performing some simple morphological erode/dilate iterations at least some of the noise is removed, after which, in case the fast algorithm is selected, the object detection algorithm is used. If off-line processing is chosen a distance transform of each detected region is a calculated, and objects that are not "thick" enough are assumed to be garbage, and is ignored.

2.4.2 Object detection

To detect the regions the OpenCV command *findContours* is used. This will extract all contours in the image. For each detected contour a bounding rectangle of the type *cv::Rect* is created and, from this rectangle an *object* is created. All of the detected objects are then put into the Frame's object vector, and the foreground processing is complete.

2.5 Object identification

The purpose of *Object identification* is to assign id's to objects discovered in the current frame based on what was known previous frames and using predictions from the Kalman filter. The aim is to assign a previously detected objects id to the most probable current object, or to none if the previous objects was moving out of the frame. New objects must be given new id's. Occlusion should be handled and objects that disappears by entering something or becoming stationary should be marked as lost but kept.

2.5.1 Possible situations

2.5.2 Algorithms

See code in appendix D.

Figure 2.6: *Object Identification flowchart.*

2.6 Kalman prediction

How do i Kalman?? How do i Kalman??

2.6.1 Some stuff

How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? How Do i some stuff?? See code in appendix E.

Figure 2.7: *Kalman Prediction figure.*

2.6.2 Moar stuff

How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? How Do i moar stuff?? This can be seen in figure 2.7.

3 System Evaluation

How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system?? How do i Evaluate system??

3.1 Evaluation setup

How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? How do i setup evaluation? See code in appendix F.

Figure 3.1: *System evaluation figure.*

3.2 Results and stuff

Why no result?? No result because: 3.1.

References

[1] M. Sonka, V. Hlavac, R. Boyle, *Image Processing, Analysis, and Machine Vision.* Toronto: Thompson Learning, cop. 2008, 3rd ed., ISBN 0495244384.

A Source code for data structures

A.1 Frame class

```
/*
*  frame.h
*/
#ifndef FRAME_H
#define FRAME_H

#include "Object.h"
#include "BackgroundModelling/ProbabilityMap.h"
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <string>
#include <map>

using namespace cv;

class ProbabilityMap;

class Frame
{
public:
   Frame() {}  //Default constructor, used by Tiger for testing..
   Frame(Mat image, Mat probMap);

   //Draws the bounding boxes and velocity vectors of the objects in object list.
   void drawObjects(cv::Scalar color);
   void drawObjects(std::vector<Object> & objects, Scalar color);

   //Data
   Mat image;          //Should be 3-channel color
   Mat probMap;       //Should be single channel grayscale
   std::vector<Object> objects;
   std::map<std::string, double> profileData;   // Time in seconds
   ProbabilityMap * bgModel;

   //For debugging/showoff
   void showObjects();
   void showImageRaw(std::string windowID);
   void showImageProbMap(std::string windowID);
};


#endif
```

```cpp
/*
*  frame.cpp
*/

#include "Frame.h"

using namespace cv;
using namespace std;

Frame::Frame(cv::Mat image,cv::Mat probMap) : image(image),probMap(probMap)  {}

void Frame::drawObjects(Scalar color)
{
   for (std::vector<Object>::iterator it = objects.begin(); it != objects.end(); ++it)
   {
      rectangle(image, Point(it->boundingBox.x, it->boundingBox.y),
                    Point(it->boundingBox.x + it->width, it->boundingBox.y + it->height),
                    color, 1, 8);
      line(image, Point(it->x, it->y), Point(it->x + (int)(it->dx), it->y + (int)(it->dy)), color, 2 ,8);

      std::string objectText = "("+std::to_string(it->x)+","+std::to_string(it->y)+") id:"+std::to_string(it-
      int fontFace = CV_FONT_HERSHEY_COMPLEX;
      double fontScale = 0.3;
      int thickness = 0.3;
      putText(image, objectText, Point(it->boundingBox.x-it->width, it->boundingBox.y+it->height+10), fontFac
   }
}

void Frame::drawObjects(std::vector<Object> & objects, Scalar color)
{
   for (std::vector<Object>::iterator it = objects.begin(); it != objects.end(); ++it)
   {
      rectangle(image, Point(it->boundingBox.x, it->boundingBox.y),
                    Point(it->boundingBox.x + it->width, it->boundingBox.y + it->height),
                    color, 1, 8);
      line(image, Point(it->x, it->y), Point(it->x + (int)(it->dx), it->y + (int)(it->dy)), color, 2 ,8);

      std::string objectText = "("+std::to_string(it->x)+","+std::to_string(it->y)+") id:"+std::to_string(it-
      int fontFace = CV_FONT_HERSHEY_COMPLEX;
      double fontScale = 0.3;
      int thickness = 0.3;
      putText(image, objectText, Point(it->boundingBox.x-it->width, it->boundingBox.y-10), fontFace, fontScal
   }
}


////////////////////////////////////////////// Showimage ////////////////////////////////////////////////

void Frame::showObjects()
{
   for (std::vector<Object>::iterator it = objects.begin(); it != objects.end(); ++it)
   {
      it->info();
   }
}

void Frame::showImageRaw(string windowID)
{
   namedWindow( windowID, CV_WINDOW_AUTOSIZE );    // Create a window for display.
    imshow( windowID, image );                   // Show our image inside it.
}

void Frame::showImageProbMap(string windowID)
{
   namedWindow( windowID, CV_WINDOW_AUTOSIZE );        // Create a window for display.
    imshow( windowID, probMap );                // Show our image inside it.
}
```

A.2 FrameList class

```cpp
#ifndef __FRAMELIST_H_
#define __FRAMELIST_H_

```

```
4  #include <opencv2/core/core.hpp>
5  #include <list>
6  #include <vector>
7  #include <ctime>
8  #include "Frame.h"
9
10 using namespace cv;
11 using namespace std;
12
13 class FrameList
14 {
15 public:
16    FrameList(int framesToKeep);
17
18    void open(std::string path);
19    Frame & getLatestFrame();
20    list<Frame> & getFrames();
21    bool isSourceEmpty();
22    void queryNextFrame();
23
24    int getFrameAmount();
25    int getFrameRate();
26    int getCurrentFrameNumber();
27
28    //Profiling
29    void setTime(std::string name, double time);
30
31    //Visualisation
32    void display(std::string windowID);
33    void displayBackground(std::string windowID);
34    void displayForeground(std::string windowID);
35
36    void displayInfo(std::string windowID);
37
38    vector<Frame> toVector();
39
40 private:
41    list<Frame> oldFrames;
42    CvCapture * source;
43    int frameAmount, frameRate;
44    int maxFrames;
45    int currentFrameNumber;
46
47    void appendFrame(IplImage *frameImage);
48
49    //Debug
50    Mat probMap;
51    Mat infoDisplayMatrix;
52 };
53
54 #endif
```

```
#include "FrameList.h"

FrameList::FrameList(int framesToKeep)
{
   maxFrames = framesToKeep;
   probMap = imread("starsCorner.tif", CV_8UC1);
   infoDisplayMatrix = Mat(480,720, CV_8UC3);
}

void FrameList::open(std::string path)
{
   source = cvCaptureFromFile(path.c_str());
   frameAmount = cvGetCaptureProperty(source,CV_CAP_PROP_FRAME_COUNT);
   frameRate = cvGetCaptureProperty(source,CV_CAP_PROP_FPS);
   currentFrameNumber = 0;

   // Load the first frame from source
   queryNextFrame();
}

Frame & FrameList::getLatestFrame()
{
   return oldFrames.front();
}

list<Frame> & FrameList::getFrames()
{
   return oldFrames;
}

bool FrameList::isSourceEmpty()
{
   return getCurrentFrameNumber() >= frameAmount;
}

void FrameList::queryNextFrame()
{
   if(getFrameAmount() <= getCurrentFrameNumber())
      return;

   appendFrame(cvQueryFrame(source));
   currentFrameNumber++;
}

void FrameList::appendFrame(IplImage *frameImage)
{
   if(getCurrentFrameNumber() >= maxFrames)
   {
      oldFrames.pop_back();
   }
   oldFrames.push_front(Frame(Mat(frameImage), probMap));
}

int FrameList::getFrameAmount()
{
   return frameAmount;
}

int FrameList::getCurrentFrameNumber()
{
   return currentFrameNumber;
}

int FrameList::getFrameRate()
{
   return frameRate;
}

vector<Frame> FrameList::toVector()
{
   return vector<Frame>(oldFrames.begin(),oldFrames.end());
}

void FrameList::setTime(std::string name, double time)
{
```

```
76     getLatestFrame().profileData[name] = time;
77 }
78 
79 void FrameList::display(std::string windowID)
80 {
81     int fontFace = CV_FONT_HERSHEY_COMPLEX;
82     double fontScale = 0.5;
83     int thickness = 1;
84 
85     std::string text = "[Frame "+std::to_string(getCurrentFrameNumber())+"("+std::to_string(get
86     getLatestFrame().drawObjects(cv::Scalar(250, 0, 0, 255));
87     putText(getLatestFrame().image, text, Point(5, 15), fontFace, fontScale, Scalar::all(0), th
88     imshow( windowID.c_str(), getLatestFrame().image );
89 }
90 
91 void FrameList::displayBackground(std::string windowID)
92 {
93     imshow( windowID.c_str(), getLatestFrame().probMap );
94 }
95 
96 void FrameList::displayForeground(std::string windowID)
97 {
98     //<TODO>
99 }
100 
101 #define PUTTEXT(x,y,text) putText(infoDisplayMatrix, text, Point(x, y), fontFace, fontScale, S
102 
103 void FrameList::displayInfo(std::string windowID)
104 {
105     Frame * frame = &getLatestFrame();
106     if(oldFrames.size() > 2)
107         frame = &(*(++oldFrames.begin()));
108 
109     int fontFace = CV_FONT_HERSHEY_COMPLEX;
110     double fontScale = 1;
111     int thickness = 1;
112 
113     infoDisplayMatrix = Scalar::all(200);
114     string text;
115     int baseline;
116 
117     PUTTEXT(5,25,"Profiling:");
118     int l = 60;
119     for(std::map<std::string, double>::iterator i = frame->profileData.begin(); i != frame->pro
120     {
121         text = "    "+i->first+":";
122         while(getTextSize(text, fontFace, fontScale, thickness, &baseline).width < 400)
123             text += " ";
124         text += std::to_string(i->second);
125         PUTTEXT(5,l,text);
126         l += 35;
127     }
128 
129     imshow( windowID.c_str(), infoDisplayMatrix);
130 }
```

A.3 Object class

```
1  // object.h, Contains class definition of object
2  
3  
4  #include <iostream>
5  #include <opencv2/core/core.hpp>
6  
7  #ifndef OBJECT_H
8  #define OBJECT_H
9  
10 class Object
11 {
12 public:
13     Object() {x = 0, y = 0, dx = 0, dy = 0, width = 0, height = 0, id = 0;}
14     Object(int x, int y, float dx, float dy, int width = 0, int height = 0, int id = 0);
15     Object(cv::Rect boundingBox, float dx = 0, float dy = 0, int id = 0);
```

```
16
17      int id, x, y, width, height;
18      float dx, dy;
19
20      cv::Rect boundingBox;
21
22      friend std::ostream & operator<< (std::ostream & o, Object & obj);
23      void info();
24  };
25  #endif
```

```cpp
// object.ccp

//#include "stdafx.h"

#include "Object.h"
#include <iostream>

using namespace std;

Object::Object(int x, int y, float dx, float dy, int width, int height, int id)
: x(x), y(y), dx(dx), dy(dy), width(width), height(height), id(id)
{
    boundingBox.x = x-width/2;
    boundingBox.y = y-height/2;
    boundingBox.width = width;
    boundingBox.height = height;
}

Object::Object(cv::Rect boundingBox, float dx, float dy, int id)
: boundingBox(boundingBox), dx(dx), dy(dy), id(id)
{
    x = boundingBox.x + boundingBox.width/2;
    y = boundingBox.y + boundingBox.height/2;
    width = boundingBox.width;
    height = boundingBox.height;
}

std::ostream & operator<< (std::ostream & o, Object & object)
{
    o << "ID: " << object.id << "\n";
    o << "Position: (" << object.x << ", " << object.y << ")" << "\n";
    o << "Dimenson: (" << object.width << ", " << object.height << ")" << "\n";
    o << "Velocity: (" << object.dx << ", " << object.dy << ")" << "\n";
    return o;
}

void Object::info()
{
    cout << "ID: " << id << endl;
    cout << "Position: (" << x << ", " << y << ")" << endl;
    cout << "Dimenson: (" << width << ", " << height << ")" << endl;
    cout << "Velocity: (" << dx << ", " << dy << ")" << endl;
}
```

B Source code for Background modeling

B.1 Background Model

```
#ifndef _BACKGROUND_MODEL_H_
#define _BACKGROUND_MODEL_H_

#include "../Frame.h"
#include "ProbabilityMap.h"
#include <iostream>
using namespace std;

namespace BackgroundModelling
{
   class BackgroundModel
   {
   public:
      void update(std::list<Frame> &frames);

   };
}

#endif
```

```
#include "BackgroundModel.h"

string getImgType(int imgTypeInt)
{
    int numImgTypes = 35; // 7 base types, with five channel options each (none or C1, ..., C4)

    int enum_ints[] =       {CV_8U,  CV_8UC1,  CV_8UC2,  CV_8UC3,  CV_8UC4,
                             CV_8S,  CV_8SC1,  CV_8SC2,  CV_8SC3,  CV_8SC4,
                             CV_16U, CV_16UC1, CV_16UC2, CV_16UC3, CV_16UC4,
                             CV_16S, CV_16SC1, CV_16SC2, CV_16SC3, CV_16SC4,
                             CV_32S, CV_32SC1, CV_32SC2, CV_32SC3, CV_32SC4,
                             CV_32F, CV_32FC1, CV_32FC2, CV_32FC3, CV_32FC4,
                             CV_64F, CV_64FC1, CV_64FC2, CV_64FC3, CV_64FC4};

    string enum_strings[] = {"CV_8U",  "CV_8UC1",  "CV_8UC2",  "CV_8UC3",  "CV_8UC4",
                             "CV_8S",  "CV_8SC1",  "CV_8SC2",  "CV_8SC3",  "CV_8SC4",
                             "CV_16U", "CV_16UC1", "CV_16UC2", "CV_16UC3", "CV_16UC4",
                             "CV_16S", "CV_16SC1", "CV_16SC2", "CV_16SC3", "CV_16SC4",
                             "CV_32S", "CV_32SC1", "CV_32SC2", "CV_32SC3", "CV_32SC4",
                             "CV_32F"  "CV_32FC1", "CV_32FC2", "CV_32FC3", "CV_32FC4",
                             "CV_64F", "CV_64FC1", "CV_64FC2", "CV_64FC3", "CV_64FC4"};

    for(int i=0; i<numImgTypes; i++)
    {
        if(imgTypeInt == enum_ints[i]) return enum_strings[i];
    }
    return "unknown image type";
}

namespace BackgroundModelling
{
   void BackgroundModel::update(std::list<Frame> &frames)
   {
      Frame * currFrame = &frames.front();
      Frame * prevFrame;

      if(frames.size() > 2)
         prevFrame = &(*(++frames.begin()));
      else
         prevFrame = NULL;

      //and create the acual background model
      ProbabilityMap *backgroundModel = new ProbabilityMap(prevFrame,currFrame);
      currFrame->probMap = backgroundModel->pImage;
      currFrame->bgModel = backgroundModel;

      cout << "first few positions are:";
      cout << currFrame->probMap.at<float>(1,0) << ", ";
      cout << currFrame->probMap.at<float>(2,0) << ", ";
      cout << currFrame->probMap.at<float>(3,0) << endl;
```

```
51
52          cout << "first few w is: ";
53          cout << currFrame->bgModel->biggestW[0] << ", ";
54          cout << currFrame->bgModel->biggestW[4] << ", ";
55          cout << currFrame->bgModel->biggestW[20] << ", ";
56          cout << endl;
57
58          cout << "the first gauss has sigma:";
59          cout << (int)currFrame->bgModel->distributions[0].sigma[0] << ", ";
60          cout << (int)currFrame->bgModel->distributions[1].sigma[0] << ", ";
61          cout << (int)currFrame->bgModel->distributions[2].sigma[0] << ", ";
62          cout << endl << endl << endl;
63
64      }
65      // Additional function-/methodimplementations here
66  }
```

B.2 Probability Map

```
#ifndef __PROBABILITYMAP__
#define __PROBABILITYMAP__

#include <opencv2/core/core.hpp>
#include <math.h>
#include <iostream>

#include "../Frame.h"

class Frame;

using namespace cv;
using namespace std;

struct gauss3D{
   uchar sigma[3];
   uchar mean[3];
   float w;
};

class ProbabilityMap{
public:
   Mat pImage;
   float *biggestW;

   ProbabilityMap(Frame *prevFrame, Frame *currFrame);
   void setB(int rows, int cols);

   static const int numGauss = 3; //how many distributions will be used
   const float lambda; //treshold for belonging to a distribution
   const float initSigma; //inital sigma for new distribution
   const float alpha;
   int maxRow, maxCol, maxIndex;

   gauss3D *distributions;
   void updateDistributions(Mat image);
   bool betterMatch(gauss3D bestDist, gauss3D distK);
   void initDistribution(gauss3D &g, uchar sigma, Vec3b mean);
   float sigmaSize(gauss3D g);
   float sumW(int row, int col, int maxRow);
   int ci(int row,int col, int k); //dumb index computation
   int ci(int row, int col);
   uchar distanceToGauss(gauss3D g, Vec<unsigned char, 3> p);
};

#endif
```

```
#include "ProbabilityMap.h"

ProbabilityMap::ProbabilityMap(Frame *prevFrame, Frame *currFrame)
   : lambda(2.5), initSigma(3), alpha(0.1){

   int numPixels = currFrame->image.rows * currFrame->image.cols;
   biggestW = new float[numPixels];
   Mat image = currFrame->image;
   maxRow = image.rows;
   maxCol = image.cols;
   maxIndex = maxCol * maxRow;

   if(prevFrame == NULL){
      int cRow = 0,cCol = 0;
      distributions = new gauss3D[numGauss*numPixels];
      //for all pixels
      for(int row=0; row < image.rows; row++){
         //for all columns
         for(int col=0; col < image.cols*3; col++){
            //for each channel in the image
            for(int c=0; c < 3; c++){
               //set all gausess
               for(int k=0; k < numGauss; k++){
                  distributions[ci(row,col,k)].mean[c] = image.at<Vec3b>(row,col)[c];
                  distributions[ci(row,col,k)].sigma[c] = initSigma;
                  distributions[ci(row,col,k)].w = 1.0/numGauss;

               }
            }
         }
      }
   }else{
      distributions = prevFrame->bgModel->distributions;
      updateDistributions(image);
   }
   setB(image.rows,image.cols);
}

void ProbabilityMap::updateDistributions(Mat image){
   int bestMatch;
   float distance;

   //update all distributions for all pixels
   for(int row=0; row < image.rows; row++){
      for(int col=0; col < image.cols; col++){

         //measure distance to each gaussian for this pixel
         distance = 0;
         bestMatch = -1;
         gauss3D bestDist;
         for(int k=0; k < numGauss; k++){

            distance = distanceToGauss(distributions[ci(row,col,k)],image.at<Vec3b>(row,col));

            if(distance < lambda){//se if we belong to this distribution
               if(bestMatch == -1){
                  bestDist = distributions[ci(row,col,k)];
               }else if(betterMatch(bestDist,distributions[ci(row,col,k)])){
                  bestDist = distributions[ci(row,col,k)];
                  bestMatch = k;
               }
            }
         }

         if(bestMatch == -1){//create a new distribution if we did not match at all
            bestMatch = numGauss;//this should be the worst match instead
            initDistribution(distributions[row*image.rows+col+bestMatch],
                                    initSigma,image.at<Vec3b>(row,col));
         }else{
            float roh;
            float w = distributions[row*image.rows+col+bestMatch].w;
            w = (1 - alpha)*w + alpha;
            distributions[ci(row,col,bestMatch)].w = w;
            roh = alpha/w;
            for(int c=0; c < 3; c++){
```

```
76                distributions[ci(row,col,bestMatch)].mean[c] = (1 - roh)*distributions[ci(row,col,bestMatch)].
77                distributions[ci(row,col,bestMatch)].sigma[c] = (1 - roh)*distributions[ci(row,col,bestMatch)]
78             }
79          }
80          //n got  r  fuckat h r.
81          //kolla mer imorgon bitti
82          float wSum = sumW(row,col,image.rows);
83          for(int k=0; k < numGauss; k++){
84             float w = distributions[ci(row,col,k)].w;
85             distributions[ci(row,col,k)].w = w/wSum;
86             float wSig = distributions[ci(row,col,k)].w / sigmaSize(distributions[row*image.rows+col+k]);
87             if(biggestW[ci(row,col)] < wSig){
88                biggestW[ci(row,col)] = wSig;
89             }
90          }
91       }
92    }
93 }
94
95 uchar ProbabilityMap::distanceToGauss(gauss3D g, Vec<unsigned char, 3> p){
96    uchar distance = 0;
97    //for each channel
98    for(int c=0; c < 3; c++){
99       distance = distance + pow((p[c]-g.mean[c])/g.sigma[c],2);
100   }
101   return distance;
102 }
103
104 void ProbabilityMap::setB(int rows, int cols){
105    Mat p(rows,cols,DataType<float>::type);
106
107    for(int row=0; row < rows; row++){
108       for(int col=0; col < cols; col++){
109          p.at<float>(row,col) = biggestW[row*rows+col];
110       }
111    }
112    pImage = p;
113
114 }
115
116 float ProbabilityMap::sumW(int row, int col,int maxRow){
117    float acc = 0;
118    for(int k=0; k < numGauss; k++){
119       acc = acc + distributions[row*maxRow+col+k].w;
120    }
121    return acc;
122 }
123
124 void ProbabilityMap::initDistribution(gauss3D &g, uchar sigma, Vec3b mean){
125    for(int i=0; i < 3; i++){
126       g.sigma[i] = sigma;
127       g.mean[i] = mean[i];
128    }
129    g.w = 1/numGauss;
130 }
131
132 float ProbabilityMap::sigmaSize(gauss3D g){
133    float acc = 0;
134    for(int i=0; i < 3; i++){
135       acc = acc + sqrt(g.sigma[i]);
136    }
137    return acc;
138 }
139
140 bool ProbabilityMap::betterMatch(gauss3D bestMatch, gauss3D thisone){
141    return thisone.w/sigmaSize(thisone) > bestMatch.w/sigmaSize(bestMatch);
142 }
143
144 int ProbabilityMap::ci(int row, int col, int k){
145    return ci(row, col) + k;
146 }
147
148 int ProbabilityMap::ci(int row, int col){
149    return row * maxRow + col;
150 }
```

C Source code for Foreground segmentation

```
#ifndef _FOREGROUND_PROCESSOR_H_
#define _FOREGROUND_PROCESSOR_H_

#include "../Frame.h"
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>

////////////////// Module /////////////////
//////////////////////////////////////////
namespace ForegroundProcessing
{
   enum Algorithm
   {
      FAST = 0,
      SLOW
   };

   class ForegroundProcessor
   {
   public:
      ForegroundProcessor(){ algorithm = FAST; threshval = 50; iterations = 3; minDist = 20; }

      void segmentForeground(Frame & frame);

      // Fast algorithm (>30ms)
      // Specify threshold value and number of iterations
      void segmentForegroundFast(Frame & frame, int threshval, int iterations);

      // Higher performance algorithm (Hopefully) (~650ms)
      // Specify threshold value and minimum consour thickness
      void segmentForegroundSlow(Frame & frame, int threshval, double minDist);

      void init(Algorithm algorithm, int threshval, double iterationsORmindist);

   private:
      //Finds objects in a binary image and puts them in the list.
      void getObjects(Frame & frame);
      //Same as above but also performs a cleanup using the distance transform.
      void getObjectsDistMap(Frame & frame, double minDist);

      //Image Processing of probabilitymap.
      void threshMap(cv::Mat probMap, int threshval);
      void openingBinMap(cv::Mat probMap, int iterations = 1);
      void closingBinMap(cv::Mat probMap, int iterations = 1);
      void erodeBinMap(cv::Mat probMap, int iterations = 1);
      void dilateBinMap(cv::Mat probMap, int iterations = 1);

      // Settings
      Algorithm algorithm;
      int threshval, iterations, minDist;

   };
}


#endif
```

```
// ForecroundProcessor.cpp

#include "ForegroundProcessor.h"
//#include "stdafx.h"

namespace ForegroundProcessing
{

   ////////////////// Foreground Segmentation ////////////////////
   void ForegroundProcessor::segmentForeground(Frame & frame)
   {
      switch(algorithm)
      {
      case 0:
         segmentForegroundFast(frame, threshval, iterations);
         break;
      case 1:
         segmentForegroundSlow(frame, threshval, minDist);
         break;
      }
   }

   void ForegroundProcessor::segmentForegroundFast(Frame & frame, int threshval, int iterations)
   {
      threshMap(frame.probMap, threshval); //Threshold at threshval

      // Erode followed by 3 iterations of dilate (3x3 kernel)
      openingBinMap(frame.probMap, iterations);

      getObjects(frame);
      return;
   }

   void ForegroundProcessor::segmentForegroundSlow(Frame & frame, int threshval, double minDist)
   {
      threshMap(frame.probMap, threshval); //Threshold at threshval

      getObjectsDistMap(frame, minDist);

      return;
   }



   void ForegroundProcessor::init(Algorithm algorithm, int threshval, double iterationsORmindist)
   {
      this->algorithm = algorithm;
      this->threshval = threshval;
      this->iterations = int(iterationsORmindist);
      this->minDist = iterationsORmindist;
   }

   ////////////////// Private Functions ////////////////////
   /////////////////////////////////////////////////////////

   ////////////////// Object Detection ////////////////////
   void ForegroundProcessor::getObjects(Frame & frame)
   {
      vector<vector<Point>> contours;
      findContours( frame.probMap.clone(), contours, CV_RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

      for(unsigned int i = 0; i < contours.size(); i++)
      {
         //Create an object for every countour using the boundingRect command
         frame.objects.push_back(Object(boundingRect(contours[i])));
      }
   }

   void ForegroundProcessor::getObjectsDistMap(Frame & frame, double minDist)
   {
      vector<vector<Point>> contours;
      findContours( frame.probMap.clone(), contours, CV_RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

      Rect objRect;
      double dist = 0;
```

```
76          double minSize = 20;
77          for(unsigned int i = 0; i < contours.size(); i++)
78          {
79             objRect = boundingRect(contours[i]);
80             vector<Point> contour = contours[i];
81
82             //Measure distance to the contour of all pixels within the bounding box.
83             for( int j = objRect.x; j < objRect.x + objRect.width; j++)
84                { for( int k = objRect.y; k < objRect.y + objRect.height; k++)
85                   {
86                      if (pointPolygonTest(contour, Point(j, k), false) == 1) //If object is inside
87                      {
88                         dist = max(dist, pointPolygonTest(contour, Point(j, k), true)); // Calcul
89                      }
90                   }
91                }
92             if (dist > minDist) //Create object only if distance is great enough.
93             {
94                frame.objects.push_back(Object(objRect));
95             }
96             dist = 0;
97          }
98       }
99
100      ////////////////// Image Processing //////////////////////
101      void ForegroundProcessor::threshMap(Mat probMap, int threshval)
102      {
103         int maxVal = 255;
104         threshold(probMap, probMap, threshval, maxVal, THRESH_BINARY);
105      }
106
107      void ForegroundProcessor::openingBinMap(Mat probMap, int iterations)
108      {
109         cv::Mat kernel;
110         kernel = getStructuringElement( MORPH_RECT, Size(3, 3));
111         dilate(probMap, probMap, kernel, cv::Point(-1,-1), iterations);
112         erode(probMap, probMap, kernel, cv::Point(-1,-1), iterations);
113      }
114
115      void ForegroundProcessor::closingBinMap(Mat probMap, int iterations)
116      {
117         cv::Mat kernel;
118         kernel = getStructuringElement( MORPH_RECT, Size(3, 3));
119         dilate(probMap, probMap, kernel, cv::Point(-1,-1), iterations);
120         erode(probMap, probMap, kernel, cv::Point(-1,-1), iterations);
121      }
122
123      void ForegroundProcessor::erodeBinMap(Mat probMap, int iterations)
124      {
125         cv::Mat kernel;
126         kernel = getStructuringElement( MORPH_RECT, Size(3, 3));
127         erode(probMap, probMap, kernel, cv::Point(-1,-1), iterations);
128      }
129
130      void ForegroundProcessor::dilateBinMap(Mat probMap, int iterations)
131      {
132         cv::Mat kernel;
133         kernel = getStructuringElement( MORPH_RECT, Size(3, 3));
134         dilate(probMap, probMap, kernel, cv::Point(-1,-1), iterations);
135      }
136
137   }
```

D Source code for Object identification

```
#ifndef _IDENTIFICATION_H_
#define _IDENTIFICATION_H_

#include "../Frame.h"
#include <list>
#include <cmath>
#include <queue>

namespace Identification
{

   class ProbabilityContainer;
   enum Algorithm
   {
      Naive = 0,
      Test,
      Experimental
   };


   //////////     Module     ///////////
   ////////////////////////////////////

   class Identifier
   {
   public:
      Identifier() {uniqueIDPool = 1; algorithm = &Identifier::algorithm_naive;}
      void identify(std::list<Frame> & frames);

      void init(Algorithm algorithmName);

   private:
      Algorithm algorithmName;
      void (Identifier::*algorithm)(std::list<Frame> & frames);

      int uniqueIDPool;
      int newID() {return uniqueIDPool++;}

      std::vector<std::list<ProbabilityContainer> > mostProbable;
      std::list<int> undecidedObjects;
      std::vector<bool> isDecided;

      void algorithm_naive(std::list<Frame> & frames);
      void algorithm2(std::list<Frame> & frames);
      void algorithm3(std::list<Frame> & frames);
   };


   //////////  Internal structures  ///////////
   ////////////////////////////////////////////

   class ProbabilityContainer
   {
   public:
      int index;
      int probableId;
      float error;
      ProbabilityContainer(int index, int probableId, float error) : index(index),probableId(probableId),erro
      bool operator<(const ProbabilityContainer & pc) { return error < pc.error; }
   };

   struct Condition
   {
      int indexValue;
      Condition(int indexValue) : indexValue(indexValue) {}
      bool operator() (const ProbabilityContainer& pc) { return pc.index == indexValue; }
   };


   ////////// TEST GENERATION ///////////
   //////////////////////////////////////

   void generate_testdata(std::list<Frame> & frameList, std::string test = "simple1");
```

```
74      float randf();
75  }
76
77  #endif
```

```
#include "Identification.h"


namespace Identification
{

   //////////     Module     //////////
   ////////////////////////////////////

   void Identifier::init(Algorithm algorithmName)
   {
      switch(algorithmName)
      {
      case Naive:
         algorithm = &Identifier::algorithm_naive;
         break;
      case Test:
         algorithm = &Identifier::algorithm2;
         break;
      case Experimental:
         algorithm = &Identifier::algorithm3;
         break;
      }
   }

   void Identifier::identify(std::list<Frame> & frames)
   {
      if(frames.size() == 1)  // First time no objects are identified
      {
         for(int i = 0; i < frames.front().objects.size(); i++)
         {
            frames.front().objects[i].id = newID();
         }
      }
      else
      {
         (this->*algorithm)(frames);
      }
   }

   void Identifier::algorithm_naive(std::list<Frame> & frames)
   {
      Frame * current = &frames.front();
      Frame * previous = &(*(++frames.begin()));

      // Find the previous object which is probably the current object
      float distanceError, error;
      int pIndex, prevpIndex;
      std::list<ProbabilityContainer> mostProbable;

      isDecided.clear();
      for(std::vector<Object>::iterator p = previous->objects.begin(); p != previous->objects.end(); p++)
         isDecided.push_back(false);

      for(std::vector<Object>::iterator c = current->objects.begin(); c != current->objects.end(); c++)
      {
         mostProbable.push_back(ProbabilityContainer(-1, -1, 1000000));
         pIndex = 0;
         prevpIndex = -1;
         for(std::vector<Object>::iterator p = previous->objects.begin(); p != previous->objects.end(); p++)
         {
            /*
             * distanceError = (x-x0-vx0)^2 + (y-y0-vy0)^2
             */
            distanceError = std::pow(c->x - p->x - p->dx, 2) + std::pow(c->y - p->y - p->dy, 2);

            error = distanceError;
            if(!isDecided[pIndex] && mostProbable.back().error > error && error < 5000)
            {
               mostProbable.back().error = error;
               mostProbable.back().index = pIndex;
               if(prevpIndex >= 0)
                  isDecided[prevpIndex] = false;
               isDecided[pIndex] = true;
               prevpIndex = pIndex;
```

```
76              }
77              pIndex++;
78           }
79        }
80
81        pIndex = 0;
82        for(std::list<ProbabilityContainer>::iterator p = mostProbable.begin(); p != mostProbabl
83        {
84           if(p->index >= 0)
85              current->objects[pIndex].id = previous->objects[p->index].id;
86           else
87              current->objects[pIndex].id = newID();
88           pIndex++;
89        }
90
91     }
92
93     void Identifier::algorithm2(std::list<Frame> & frames)
94     {
95
96        Frame * current = &frames.front();
97        Frame * previous = &(*(++frames.begin()));
98
99        mostProbable.clear();
100       undecidedObjects.clear();
101       float distanceError, error;
102
103       for(int i = 0; i < current->objects.size(); i++)
104       {
105          undecidedObjects.push_back(i);
106          mostProbable.push_back(std::list<ProbabilityContainer>());
107
108          for(int j = 0; j < previous->objects.size(); j++)
109          {
110             /*
111              * distanceError = (x-x0-vx0)^2 + (y-y0-vy0)^2
112              */
113             distanceError = std::pow(current->objects[i].x - previous->objects[j].x - previous
114             error = distanceError;
115
116             mostProbable[i].push_back(ProbabilityContainer(j,previous->objects[j].id,error));
117          }
118          mostProbable[i].sort();
119       }
120
121       //std::cout << "\nFind most probable previous object:\n";
122       std::list<int>::iterator bestMatch;
123       int matchingPrevious;
124       float min;
125       for(int candidate = 0; candidate < std::min(previous->objects.size(), current->objects.s
126       {
127          min = 1000000;
128          for(std::list<int>::iterator i = undecidedObjects.begin(); i != undecidedObjects.end(
129          {
130             if(mostProbable[*i].front().error < min)
131             {
132                min = mostProbable[*i].front().error;
133                bestMatch = i;
134             }
135          }
136
137          //A most probable candidate found!
138          matchingPrevious = mostProbable[*bestMatch].front().index;
139          current->objects[*bestMatch].id = previous->objects[matchingPrevious].id;
140
141          //std::cout << "\tObject " << mostProbable[*bestMatch].front().probableId << " found
142
143          for(std::list<int>::iterator i = undecidedObjects.begin(); i != undecidedObjects.end(
144          {
145             std::list<ProbabilityContainer>::iterator it = mostProbable[*i].begin();
146             while(it != mostProbable[*i].end())
147             {
148                if(it->index == matchingPrevious)
149                   mostProbable[*i].erase(it++);
150                else
151                   it++;
```

```
152             }
153             mostProbable[*i].sort();
154          }
155          undecidedObjects.erase(bestMatch);
156       }
157
158
159       //Take care of the new objects detected (still undecided):
160       for(std::list<int>::iterator i = undecidedObjects.begin(); i != undecidedObjects.end(); i++)
161       {
162          current->objects[*i].id = newID();
163       }
164
165    }
166
167    void Identifier::algorithm3(std::list<Frame> & frames)
168    {
169       /*
170       // Select the objects in the previous frame that are candidates to the unidentified in the current fram
171       std::vector<Object *> candidates;
172       candidates.clear();
173       for(std::vector<Object>::iterator c = previous->objects.begin(); c != previous->objects.end(); c++)
174       {
175          if(!probablyOutsideOfImage(*c))
176             candidates.push_back(&(*c));
177       }
178       */
179    }
180
181
182
183    ////////// TEST GENERATION ///////////
184    ///////////////////////////////////////
185
186    const int cTEST_FRAME_WIDTH = 480;
187    const int cTEST_FRAME_HEIGHT = 360;
188
189    #define NEW_FRAME() frameList.push_back(Frame(cv::Mat(cTEST_FRAME_HEIGHT, cTEST_FRAME_WIDTH, CV_8UC3), cv:
190    #define INSERT_OBJECT(x,y,dx,dy) frameList.back().objects.push_back(Object(x, y, dx, dy, 20, 60));
191    //#define INSERT_OBJECT(x,y) frameList.back().objects.push_back(Object(x, y, 0, 0, 20, 60));
192
193    void generate_testdata(std::list<Frame> & frameList, std::string test)
194    {
195       if(test == "simple1")
196       {
197          int stepLength = 10;
198          float var = 10;   // Variance
199          for(int i = 0; i < cTEST_FRAME_WIDTH; i+=stepLength)
200          {
201             NEW_FRAME();
202             INSERT_OBJECT(i, 200+var*randf(), stepLength+var*randf(), var*randf());
203             INSERT_OBJECT(i, 100+10*randf(), stepLength+var*randf(), var*randf());
204          }
205       }
206       else if(test == "complex1")
207       {
208          float var = 20;   // Variance
209          for(int t = 0; t < cTEST_FRAME_WIDTH; t++)
210          {
211             NEW_FRAME();
212             INSERT_OBJECT(t*10, 200+var*randf(), 10+var*randf(), var*randf());
213             INSERT_OBJECT(20+t*10+10*randf(), 20+t*10+10*randf(), 10+var*randf(), var*randf());
214          }
215       }
216
217    }
218
219    float randf()
220    {
221       return (float(rand()) - float(RAND_MAX)/2.0)/RAND_MAX;
222    }
223
224 };
```

E Source code for Kalman prediction

```
#ifndef _KALMAN_H_
#define _KALMAN_H_

#include "../Frame.h"

namespace Prediction
{
   class Kalman
   {
   public:
      Kalman(float x0 = 0, float y0 = 0);

      void predict(Frame & frame);

      cv::Mat A, C, Q, P, xHat;
      float R;
   };
   // Additional function-/datastructuredeclarations here
}

#endif
```

```
#include "Kalman.h"

namespace Prediction
{
   Kalman::Kalman(float _x0, float _y0)
   {
      // Default values for matrices
      float AData[] = { 1, 0, 1, 0,
                        0, 1, 0, 1,
                        0, 0, 1, 0,
                        0, 0, 0, 1 };
      A = cv::Mat(4, 4, CV_32FC1, AData).clone();

      float CData[] = { 1, 0, 0, 0,
                        0, 1, 0, 0};
      C = cv::Mat(2, 4, CV_32FC1, CData).clone();

      float PData[] = { 1, 0, 0, 0,
                        0, 1, 0, 0,
                        0, 0, 1, 0,
                        0, 0, 0, 1 };
      P = cv::Mat(4, 4, CV_32FC1, PData).clone();

      float Qvalue = 0.1f;
      float QData[] = { Qvalue, 0, 0, 0,
                        0, Qvalue, 0, 0,
                        0, 0, Qvalue, 0,
                        0, 0, 0, Qvalue };
      Q = cv::Mat(4, 4, CV_32FC1, QData).clone();

      R = 0.1f;

      float x0Data[] = {_x0, _y0, 0, 0};
      xHat = cv::Mat(4, 1, CV_32FC1, x0Data).clone();
   }

   void Kalman::predict(Frame & frame)
   {

   }
   // Additional function-/methodimplementations here
}
```

F Source code for Main program

```
#include "Modules/FrameList.h"
#include "Modules/BackgroundModelling/BackgroundModel.h"
#include "Modules/ForegroundProcessing/ForegroundProcessor.h"
#include "Modules/ObjectIdentification/Identification.h"
#include "Modules/Prediction/Kalman.h"

#include "Modules/Profiler.h"

#define PROFILER_INIT() ProfilerClock c;
#define PROFILER_RESET() c.reset();
#define PROFILE(string) frameList.setTime(string, c.getTime()); c.lap();
#define PROFILE_TOTALTIME() frameList.setTime("Total Time", c.getTotalTime());
#define PROFILE_TOTALFPS() frameList.setTime("Total FPS", c.getTotalFPS());

int main()
{
   // Profiler init
   PROFILER_INIT();

   // Frame container
   FrameList frameList(10000);   // Keep a history of up to 100 frames (might be used by some

   // Modules
   BackgroundModelling::BackgroundModel backgroundModel;
   ForegroundProcessing::ForegroundProcessor foregroundProcessor;
   Identification::Identifier identifier;
   Prediction::Kalman kalmanFilter;

   // Init
   foregroundProcessor.init(ForegroundProcessing::FAST, 50, 3);

   // Load frame source
   frameList.open("camera1.mov");

   // Create windows
   namedWindow("Info",CV_WINDOW_AUTOSIZE);
   namedWindow("Background",CV_WINDOW_AUTOSIZE);
   namedWindow("Foreground",CV_WINDOW_AUTOSIZE);
   namedWindow("Tracking",CV_WINDOW_AUTOSIZE);

   // Track objects through all frames
   while(!frameList.isSourceEmpty())
   {
      // Reset profiler
      PROFILER_RESET();

      // Do the nessecary processing
      backgroundModel.update(frameList.getFrames());                PROFILE("BackgroundModel"
      foregroundProcessor.segmentForeground(frameList.getLatestFrame());   PROFILE("Foreground
      identifier.identify(frameList.getFrames());                   PROFILE("Identification")
      kalmanFilter.predict(frameList.getLatestFrame());             PROFILE("Kalman Predictio


      // Display result
      frameList.display("Tracking");
      frameList.displayBackground("Background");
      frameList.displayForeground("Foreground");

      // Give the GUI time to render
      waitKey(1);                                                   PROFILE("Display");

      // Optional pause between each frame
      //waitKey(0);

      // Read next frame from source
      frameList.queryNextFrame();                                           PROFILE("QueryNextFrame");
                                                                 PROFILE_TOTALTIME();
                                                                 PROFILE("Total FPS");
                                                                 PROFILE_TOTALFPS();


      // Display info
      frameList.displayInfo("Info");
   }
```

```
74
75      waitKey(0);
76
77      return 0;
78  }
```