```
/*
 * File:   analyzeFrame.h
 * Author: mcaprio2
 *
 * Created on July 11, 2014, 4:20 PM
 */
#include <stdlib.h>
#include <stdio.h>
//#include <highgui.h>
#include <opencv2/opencv.hpp>
//#include </usr/local/include/opencv2/opencv.hpp>
//#include "opencv2/nonfree/nonfree.hpp"
#include <vector>
#include <cmath>
#include <pthread.h>
//#include<random>

#include "LineDesc.h"
#include "PotentialBug.h"
#include "tracker.h"

using namespace cv;
using namespace std;

#ifndef ANALYZEFRAME_H
#define ANALYZEFRAME_H

class analyzeFrame {
public:

   Mat first_frame_gray, next_frame_gray, subtracted_frame, diff_frame, img3, diff3ch_frame;
   Mat img, img2, imgMapped, img2Mapped;
   Mat fore, back;
   unsigned int frameCount;
   vector<KeyPoint> currentBlobs;
   BackgroundSubtractorMOG2 bg;
   //BackgroundSubtractorGMG bg;


   struct thread_data {
      int thread_id;
      vector<KeyPoint> *pt_keypointsBlob;
      Mat *subtracted_frame;
   };


   analyzeFrame();
   analyzeFrame(Size ims);
```

```
analyzeFrame(const analyzeFrame& orig);
virtual ~analyzeFrame();
void initializeMats(Size);

bool StartInternalThread() {
   return (pthread_create(&this->thread, NULL, InternalThreadEntryFunc, this) == 0);
}

/** Will not return until the internal thread has exited. */
void WaitForInternalThreadToExit() {
   (void) pthread_join(thread, NULL);
}

void *analyzeFrameMT();

void setParameters() {
   no_erode = N_ERODE;
   no_erode2 = N_ERODE2;
   no_dilate = N_DILATE;
   threshold = FILTER_THRESHOLD; //was 20  *
   threshold3ch = FILTER_THRESHOLD3CH; //was 6000 *
}

void setImg2(Mat img2) {
   this->img2.release();
   this->img2 = img2.clone();
}

Mat getImg2Mapped() const {
   return img2Mapped;
}

void setImg(Mat img) {
   this->img = img;
}

Mat getImg() const {
   return img;
}

void setFramecount(unsigned int framecount) {
   this->frameCount = framecount;
}

unsigned int getFramecount() const {
   return frameCount;
}
```

```
    void SetCurrentBlobs(vector<KeyPoint> currentBlobs) {
        this->currentBlobs = currentBlobs;
    }

    vector<KeyPoint> GetCurrentBlobs() const {
        return currentBlobs;
    }

    void setSubtracted_frame(Mat subtracted_frame) {
        this->subtracted_frame = subtracted_frame;
    }

    Mat getSubtracted_frame() const {
        return subtracted_frame;
    }

    void setMapy(Mat mapy) {
      this->mapy = mapy;
    }

    Mat getMapy() const {
      return mapy;
    }

    void setMapx(Mat mapx) {
      this->mapx = mapx;
    }

    Mat getMapx() const {
      return mapx;
    }
    void setBack(Mat back);
    Mat getBack() const;
    void setFore(Mat fore);
    Mat getFore() const;
private:
    int no_erode, no_erode2;
    int no_dilate;
    uchar threshold; //was 20  *
    int threshold3ch; //was 6000 *


    pthread_t thread;
    Mat mapx, mapy;
    //Mat img2Mapped;

    static void * InternalThreadEntryFunc(void * This) {
        ((analyzeFrame *) This)->analyzeFrameMT();
```

```
      return NULL;
   }


};

#endif   /* ANALYZEFRAME_H */

#ifdef NOPE

class MyThreadClass {
public:

   MyThreadClass() {
      /* empty */
   }

   virtual ~MyThreadClass() {
      /* empty */
   }

   /** Returns true if the thread was successfully started, false if there was an error starting the thread
*/
   bool StartInternalThread() {
      return (pthread_create(&_thread, NULL, InternalThreadEntryFunc, this) == 0);
   }

   /** Will not return until the internal thread has exited. */
   void WaitForInternalThreadToExit() {
      (void) pthread_join(_thread, NULL);
   }

protected:
   /** Implement this method in your subclass with the code you want your thread to run. */
   virtual void InternalThreadEntry() = 0;

private:

   static void * InternalThreadEntryFunc(void * This) {
      ((MyThreadClass *) This)->InternalThreadEntry();
      return NULL;
   }

   pthread_t _thread;
};
#endif
```