```cpp
#include <stdio.h>
#include <string>
#include <opencv2/opencv.hpp>

#include "cvimagewidget.h"

/*!
 * \file asn1.cpp
 *
 * Computes an algorithm for calculating the primary object of motion in a scene
 * using two input images for calculating motion. The usage is:
 * `./asn1 <Image_Path1> <Image_Path2> [<Output_Path>]`
 * Where `<Image_Path[12]>` are mandatory image paths which have motion in
 * them. `<Output_Path>` is optional and will default to "output.png".
 *
 * \date 23 Sept 2014
 * \author Kevin Brightwell
 */

using std::string;
using std::cout;
using std::endl;

/*!
 * \brief DEFAULT_OUTPUT Default path for output image
 */
const static string DEFAULT_OUTPUT = "output.png";

/*!
 * \brief showImage is a convenience function for displaying an image.
 * \param title Title for the window
 * \param src Image to display
 */
void showImage(const string &title, const cv::Mat &src) {
    cv::namedWindow(title, CV_WINDOW_AUTOSIZE );
    cv::imshow(title, src);

    cv::waitKey(0);
}

/*!
 * \brief computePyrDown computes a n-level Gaussian pyramid and returns the last
 *      level.
 * \param pyrLevels Number of levels to compute
 * \param src Input array (level 0).
 * \return (`pyrLevels - 1`)th level image
 */
cv::Mat computePyrDown(const int pyrLevels, const cv::Mat &src) {

    cv::Mat from = src.clone();
    cv::Mat to = src.clone();

    for (int i = 0; i < pyrLevels; ++i) {
        cv::pyrDown(from, to);

        if (i != pyrLevels - 1)
            to.copyTo(from);
    }

    return to;
}

int main(int argc, char** argv )
{
    if ( argc != 3 && argc != 4 )
    {
        cout << "Usage: ./asn1 <Image_Path1> <Image_Path2> [<Output_Path>]" << endl;
        return -1;
    }
```

```
    // check if optional param was specified, take if it's available.
    const string OUTPUT_PATH = (argc == 4 ? argv[3] : "output.png");

    // Load images for processing
    const cv::Mat srcImage1 = cv::imread( argv[1] );
    const cv::Mat srcImage2 = cv::imread( argv[2] );

    // verify the files were valid
    if ( srcImage1.empty() || srcImage2.empty() )
    {
        printf("No image data \n");
        return -1;
    }

    // Store as arrays for iterations later
    const cv::Mat srcs[] = { srcImage1, srcImage2 };
    cv::Mat greys[2];

    // For the two images, compute the four-level Gaussian pyramid, then
    // convert to grayscale and store in `greys`.
    for (int i = 0; i < 2; ++i) {
        cv::Mat pyr = computePyrDown(3, srcs[i]);

        greys[i] = pyr.clone();
        cv::cvtColor(pyr, greys[i], CV_RGB2GRAY);
    }

    // compute the difference of the grayscale images
    cv::Mat diff(greys[0]);
    cv::absdiff(greys[0], greys[1], diff);

    // threshold the diff image using (75, 255)
    cv::Mat thresh(diff);
    cv::threshold(diff, thresh, 75, 255, cv::THRESH_BINARY);

    // Since the image is the 4th level pyr image, resize it to normal size
    cv::Mat resized(srcImage1);
    cv::resize(thresh, resized, resized.size(), 0, 0, cv::INTER_NEAREST);

    // dialate the mask to "fill in the blanks"
    cv::Mat dilated(resized);
    cv::dilate(resized, dilated, cv::Mat(), cv::Point(-1, -1), 25);

    // convert the mask's one channel to 3 channel RGB
    cv::Mat mask(dilated);
    cv::cvtColor(dilated, mask, CV_GRAY2RGB);

    // create the output and bitwise_and() the mask to only keep the marked
    // pixels
    cv::Mat output = srcImage1.clone();
    output &= mask;

    // save and show the image output
    showImage("Output", output);

    cv::imwrite(OUTPUT_PATH, output);
    cout << "Output written to: " << OUTPUT_PATH << endl;

    return 0;
}
```