CSE578 Computer Vision
Assignment 0 : OpenCV and Chroma Keying

Karnik Ram
2018701007

1 Video $\leftrightarrow$ Images

Code is written to extract the constituent frames of a video and write them into a specified folder. The video is opened using the `VideoCapture` module and the frames are read one-by-one using the `read` function. These frames are then written into the specified directory using the `imwrite` function.

1.1 Code

```
/*
 * Program to convert a given video sequence into
 * its constituent frames.
 *
 * Usage : ./video_to_images <path-to-video> <path-to-store-images>
 */

#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>

#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>

int main(int argc, char *argv[])
{
  if(argc != 3)
  {
    std::cout << "Invalid input format!\nCorrect usage: ./
   video_to_images <path-to-video> <path-to-store-images>\n";
    return -1;
  }

  const std::string video_path = argv[1], images_path = argv[2];

  cv::VideoCapture cap(video_path);
```

```cpp
  if(!cap.isOpened())
  {
    std::cout << "Video could not be opened!\n";
    return -1;
  }

  std::cout << "Video opened!\n";
  std::cout << "Number of frames in the video: " << cap.get(cv::
   CAP_PROP_FRAME_COUNT) << std::endl;

  cv::Mat frame;
  size_t frame_no = 0;

  while(cap.read(frame))
  {
    frame_no++;
    std::stringstream image_path;
    image_path << images_path << "/img" << std::setw(5) << std::
   setfill('0') << frame_no << ".png";
    std::cout << "Writing frame #" << frame_no << " into disk at " <<
    image_path.str() << std::endl;
    cv::imwrite(image_path.str(), frame);
  }

  std::cout << frame_no << " frame(s) written to the disk!\n";
  return 0;
}
```

Code is also written for merging a set of images from a specified folder into a single video. The folder name, desired frame rate, and desired video file name are taken as command line arguments. All the filenames in the specified folder are read using the `filesystem` module and then sorted according to name. Then the files are read in sorted order and written into a video using the `VideoWriter` module, at the specified frame rate.

1.2 Code

```cpp
/*
 * Program to merge a set of images
 * into a video sequence. Assumes all images are of same size.
 *
 * Usage: ./images_to_video <path_to_image_folder> <frame_rate> <
   video_name>
 */
```

```cpp

#include <iostream>
#include <stdlib.h>
#include <string>
#include <sstream>
#include <experimental/filesystem>
#include <vector>
#include <algorithm>

#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>

namespace fs = std::experimental::filesystem;

int main(int argc, char *argv[])
{
  if(argc != 4)
  {
    std::cout << "Invalid input!\nCorrect usage: ./images_to_video <path-to-image-folder> <frame-rate> <video-name>\n";
    return -1;
  }

  const std::string images_path = argv[1];
  size_t frame_rate = atoi(argv[2]);
  const std::string video_name = argv[3];

  std::vector<std::string> file_names;
  for(const auto & image_name : fs::directory_iterator(images_path))
  {
    file_names.push_back(image_name.path());
  }

  std::sort(file_names.begin(), file_names.end());

  cv::Mat frame;
  frame = cv::imread(file_names[0], 1);
  cv::VideoWriter video_writer(video_name + ".mp4", cv::VideoWriter::fourcc('X','2','6','4'), frame_rate, frame.size(), true);

  if(!video_writer.isOpened())
  {
    std::cout << "Could not open video file for writing!\n";
    return -1;
  }

  for(auto file_name : file_names)
  {
    frame = cv::imread(file_name, 1);
    video_writer.write(frame);
  }

  std::cout << "Finished writing video file: " << video_name << ".mp4
    into disk!" << std::endl;

  return 0;
}
```

2 Capturing Images

Code is written to capture frames from the webcam (video device id 0) and write them into a specified folder. This is done using the `VideoCapture` module and `imwrite` function. The frames are captured until the Esc key is pressed and the frames are also shown on screen using the `imshow` function.

2.1 Code

```
/*
 * Program to capture frames from webcam until Esc
 *
 * Usage: ./capture_images <path-to-save>
 */

#include <iostream>
#include <chrono>
#include <thread>
#include <sstream>
#include <iomanip>

#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>

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

```cpp
{
  if(argc != 2)
  {
    std::cout << "Invalid input!\nCorrect usage: ./capture_images <path-to-store-images>\n";
    return -1;
  }

  const std::string images_path = argv[1];

  cv::VideoCapture cap(0);
  cv::Mat frame;
  size_t frame_no = 0;

  cv::namedWindow("Feed", cv::WINDOW_AUTOSIZE);
  std::cout << "Beginning to capture frames from webcam in 3 seconds..\n";
  std::this_thread::sleep_for(std::chrono::milliseconds(3000));

  while(1)
  {
    cap.read(frame);
    frame_no++;

    cv::imshow("Feed", frame);
    char c = cv::waitKey(10);
    if(c == 27)
      break;

    std::stringstream image_path;
    image_path << images_path << "/img" << std::setw(5) << std::setfill('0') << frame_no << ".png";
    std::cout << "Writing frame into disk at " << image_path.str() << std::endl;
    cv::imwrite(image_path.str(), frame);
  }

  std::cout << "Exiting..\n";

  return 0;
}
```

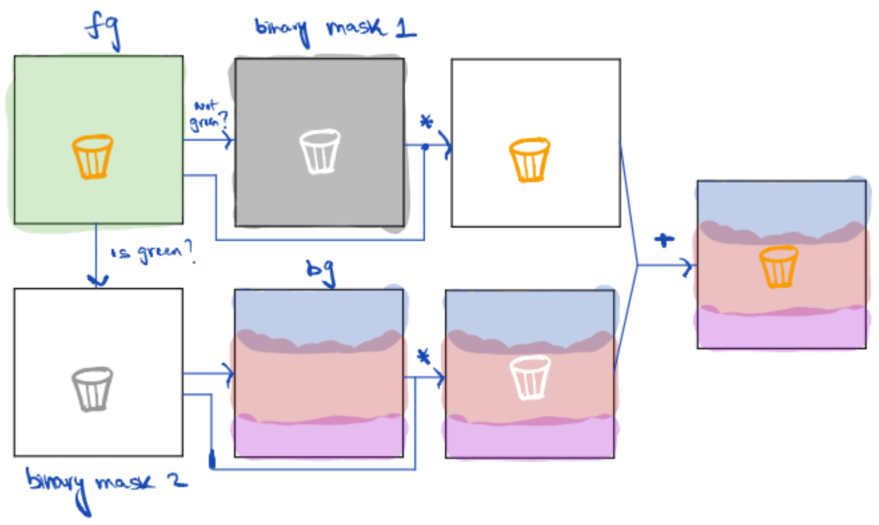


Figure 1: Rough sketch illustrating the basic chroma keying technique

3 Chroma keying

A basic Chroma Keying technique based on the constant back color assumption is implemented. Two video sequences are taken as input and a composite video sequence is created by replacing the constant green back color in the foreground video sequence with the colors from the target background sequence.

This is done by first creating two binary masks, based on whether the corresponding pixels in the foreground image are green or not, as shown in the sketch above. These masks are then applied to the foreground and background images respectively to cut out the foreground object from the foreground image and the corresponding pixels in the same position from the background image. These two masked-out images are then added togther to form the final composite image.

3.1 Code

```
/* Program to composite two video sequences together.
 * Usage: ./chroma_keying <video1-path> <video2-path>
 *
 * Video1 is assumed to be the foreground sequence with
 * a constant green background. Video2 is the target background
   sequence.
 */

#include <iostream>
#include <string>
```

```cpp

#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>

void create_composite(const cv::Mat &fg, const cv::Mat &bg, const cv::Mat &result)
{
  cv::Mat mask1(fg.size(), CV_8UC1, cv::Scalar(0));
  cv::Mat mask2(fg.size(), CV_8UC1, cv::Scalar(0));

  for(size_t i = 0; i < fg.rows; i++)
    for(size_t j = 0; j < fg.cols; j++)
    {
      cv::Vec3b intensity = fg.at<cv::Vec3b>(i,j);
      if(intensity[0] <= 100 && intensity[1] >= 100 && intensity[2] <=100)
        mask2.at<uchar>(i,j) = 255;

      else
        mask1.at<uchar>(i,j) = 255;
    }

  cv::Mat temp1, temp2;
  fg.copyTo(temp1,mask1);
  bg.copyTo(temp2,mask2);

  cv::add(temp1,temp2,result);
}

int main(int argc, char *argv[])
{
  if(argc != 3)
  {
    std::cout << "Invalid input!\nCorrect usage: ./chroma_keying <video-1-path> <video-2-path>\n";
  }

  const std::string v1_path = argv[1], v2_path = argv[2];

  cv::VideoCapture cap1(v1_path), cap2(v2_path);
  if (!cap1.isOpened())
  {
    std::cout << "Could not open video 1!\n";
    return -1;
  }

  if(!cap2.isOpened())
  {
    std::cout << "Could not open video 2!\n";
    return -1;
  }

  cv::Size v1_frame_size(cap1.get(cv::CAP_PROP_FRAME_WIDTH),
      cap1.get(cv::CAP_PROP_FRAME_HEIGHT));
  cv::Size v2_frame_size(cap2.get(cv::CAP_PROP_FRAME_WIDTH),
      cap2.get(cv::CAP_PROP_FRAME_HEIGHT));

  if(!(v1_frame_size.height == v2_frame_size.height && v1_frame_size.width == v1_frame_size.width))
  {
    std::cout << "Video sequences are not of same size!\n";
    return -1;
  }

  cv::VideoWriter video_writer("composite.mp4", cv::VideoWriter::fourcc('X','2','6','4'), 30, v1_frame_size,true);

  cv::Mat fg, bg;
  cv::Mat result_frame(v1_frame_size, CV_8UC3);

  while(cap1.read(fg) && cap2.read(bg))
  {
    create_composite(fg, bg, result_frame);
    video_writer.write(result_frame);
  }

  std::cout << "Done!\n";

  return 0;
}
```

4 Sample Images

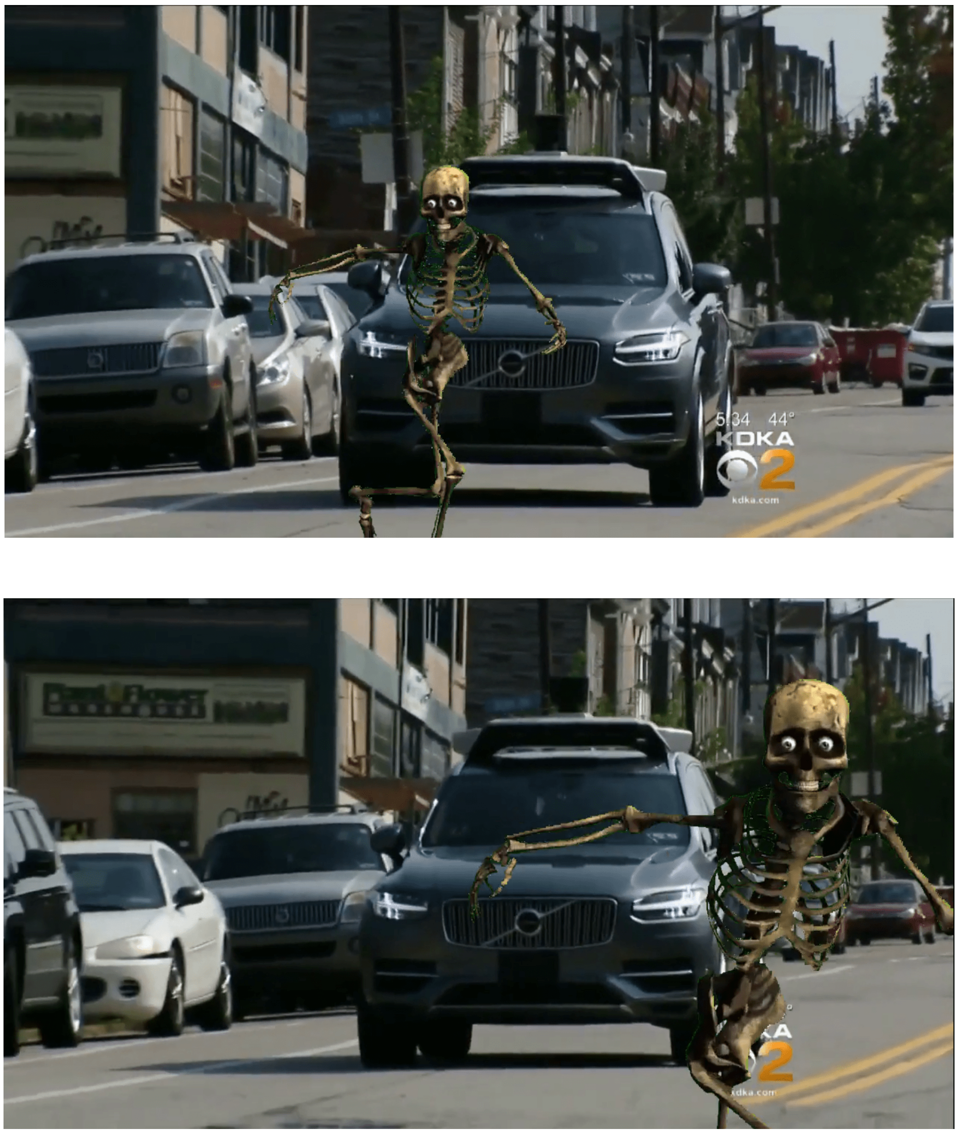


Figure 2: Composite scene involving a scared person running away from an Uber self-driving car.

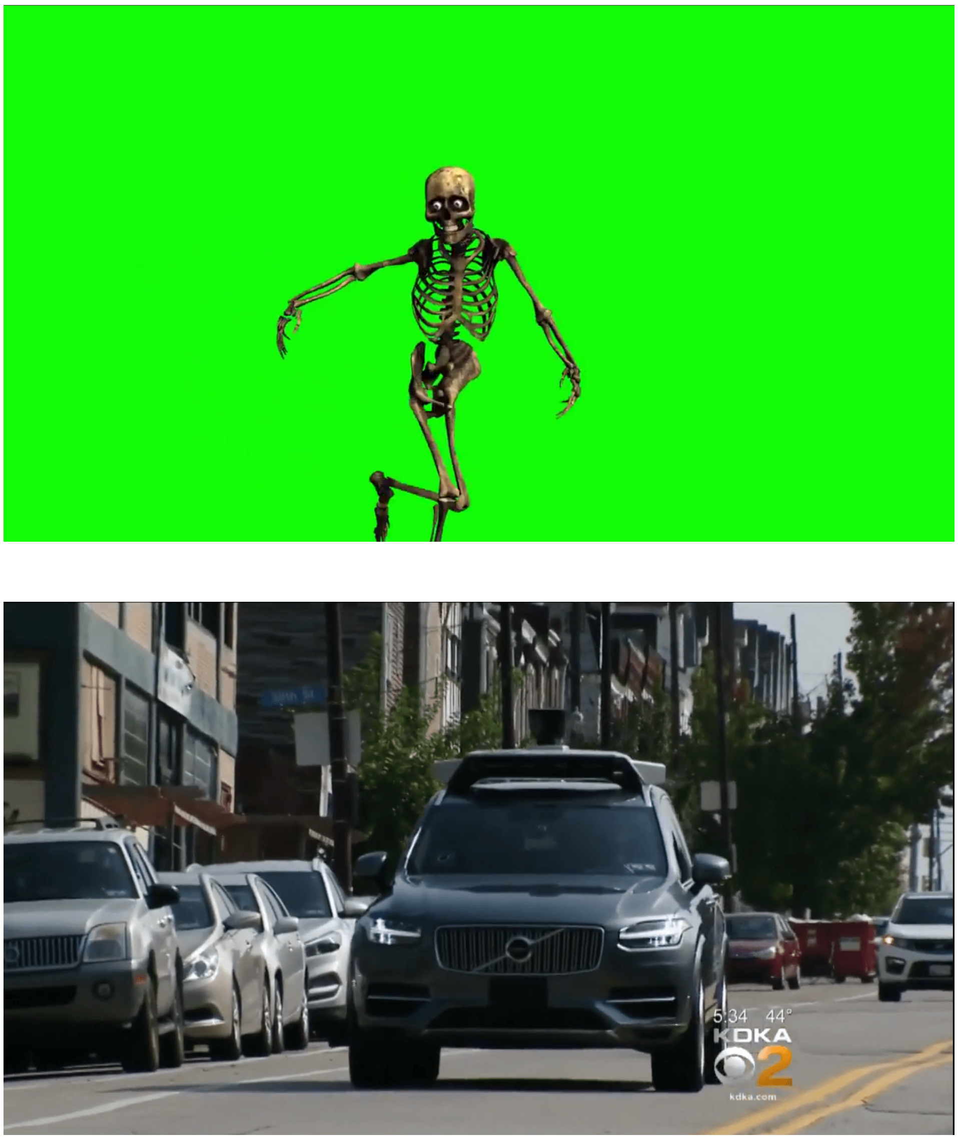


Figure 3: Frames from the input sequences.

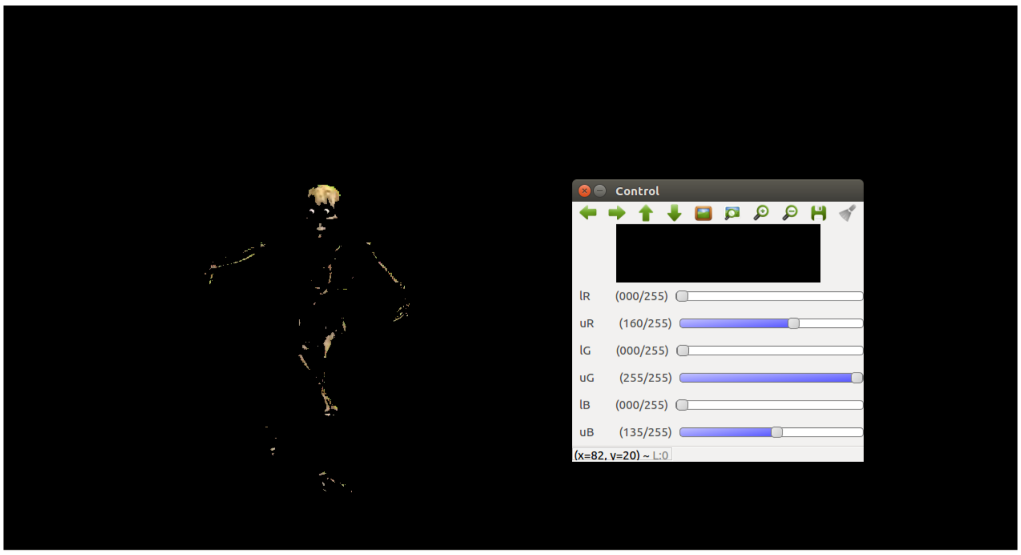
Video link

5 Challenges and Experiments

The main difficulty was in perfectly masking out the foreground object from the background since even the foreground object has some shades of green on it. I tried to finely determine the range of green values to mask out, by writing a small interactive program with trackbars to control the range of RGB values to mask out. But this did not help and some parts of the object were always being masked out as shown in Fig.4. I also tried to determine the range of values in the HSV space but that did not help much either.

Another related problem was with the edge pixels that cover parts of both the foreground object and the background. I tried applying a smoothing operation over the foreground image and it did help with some of the green edge pixels to a degree but that reduced the sharpness of the object and made the composite look a lot less realistic.

A further problem would be with translucent foreground objects, which were not present in my scene. But for such scenes, the Triangulation technique would be a much better solution.



```
lR   (000/255)
uR   (160/255)
lG   (000/255)
uG   (255/255)
lB   (000/255)
uB   (135/255)
(x=82, y=20) ~ L:0
```

Figure 4: Trying to precisely mask out all the green pixels in the image removes parts of the foreground object too.

6 Learnings

I got familiar with the basic API of OpenCV to load, read, and write images and videos, and with manipulating their pixel values. I also got familiar with CMake during the installation process. I understood the Chroma Keying problem and its difficulty, and the extra constraints we need to introduce to reasonably solve the problem. I implemented and realized the weaknesses of a technique with constant back color assumption and tried out some approaches to solving them.