|  |  |
| --- | --- |
|  | Virtual Backdrop |
|  |  |
|  | Edward Jezisek, Nichole Dugan, Brian Schulte  CprE 575  4/25/16 |

# Project Concept

The team’s Virtual Backdrop project will eliminate the need to spend hours at the Driver’s License Station waiting for other driver’s license staff to take photos. It does this by creating a professional grade driver's license photo using a picture from a cell phone. Benefits to this project include reduced costs in Department of Transportation (DOT) staff, decreased time spent at the DOT, and a license photo you are proud to have. Virtual Backdrop will let you take advantage of this by taking your photo, removing the background and allowing you to choose your favorite photo before you even step into the Driver’s License Station. Several attempts have been made at this concept in the past, but the main drawback to many of the alternatives is that hair is often cropped out of the picture.

Virtual Backdrop will specifically be used to separate the background from pictures or videos. This will allow it to be used in applications other than providing a backdrop for driver's license photos. However, the primary purpose of this application is to allow users to use their own photos as their driver's license photo.

# Target Audience

Self-service kiosks are becoming popular in states across the US to give citizens a convenient choice in renewing or replacing their driver’s licenses. This project will focus only on kiosks being used by the state of Iowa. Currently, the following services are offered at kiosk locations: 

Figure

* Renew/Replace Iowa Driver’s License or identification card
* Update Iowa Driver’s License (e.g. removing “under 18” or “under 21”, updating address)

As a result of these select options, kiosk users must meet the following requirements:

* Possess a valid Iowa Driver’s License or ID card (e.g. not a Commercial Driver’s License)
* Be between the ages of 18 and 70
* Reside in Iowa with US citizenship
* Not be required to provide a medical or vision report during the renewal process

Although anyone between 18 and 70 can use the kiosks, the system is designed to target drivers between the ages of 18 and 30 who routinely download applications and tend to choose self-service over in person interactions. The typical kiosk user also likes to avoid time wasted at the Driver’s License Station. We are specifically targeting users with Android-based smart phones and anyone who currently uses a kiosk to replace their driver's license photo. At the current time, kiosks need to have a standing or mounted backdrop behind the photos and Virtual Backdrop could help eliminate this. (Iowa Department of Transportation)

# Need for Application

Virtual Backdrop is needed to enhance the experience of getting a driver’s license for residents of the state of Iowa. This application can further be used to populate a database for facial recognition. The pictures we create will meet the standards for a driver's license and in doing so, will meet the standards for facial recognition.

With the introduction of new technologies that require a photo with a high enough quality to be included in biometric databases used for facial recognition, photos will need to remove artifacts from their backgrounds to create consistent images of faces. Previously, colored backdrops were used to ensure noise was reduced from the photos. But this places significant limitations on the photos that can be used in the databases and the process to take said photos (i.e. cellphone cameras cannot be used). Ideally, software could be created to identify faces, remove backgrounds and produce a uniform virtual backdrop for the photo regardless of the composition of the original photo. One specific hurdle to overcome is that current algorithms may be unable to properly identify a person's hair, replacing it with the backdrop color instead of keeping it intact.

# Previous Approaches

In researching current algorithms for this process, we found three major ways to approach this problem. Each of these algorithms appeared to have their own drawback, including problems identifying hair properly, not removing the whole background, or other issues that arise from color matching on pixels.

The first method is an algorithm by the identity solution vendor MorphoTrust USA. MorphoTrust currently provides physical credentials for 80% of the jurisdictions in the United States. MorphoTrust is looking to expand their business by providing this virtual backdrop solution. In a patent filed in 2015, the algorithm for the virtual backdrop solution tries to estimate the foreground and background of the image. To determine the difference between the two, alpha matting is applied to the image. Their algorithm also attempts to estimate hair pixels, facial pixels, and other foreground pixels using a formula that utilizes a convex hull algorithm. It also uses a cross algorithm to determine where the subject’s chin is located and again where the face exists in the foreground. The algorithm also tries to determine the background by using eigenvectors and pixel intensity. It also uses neighborhood matching to try to determine similar pixels in a region. Their approach seems to be a good starting point, but as of yet the company does not appear to have a working prototype. (MorphoTrust USA)

The second algorithm the team found was a paper from Incheon National University in South Korea. The authors Jeong-In Park and Jin-Tak Choi used an algorithm that utilized a code book to try to determine smaller portions of the image to try to determine background for each smaller image, in a similar fashion to solving our Sudoku problem. Each of the smaller units is a vector, and uses vector rotation about a center of rotation for the vector. Reviewing the author's work, the background does not appear to be completely removed from the images, so the algorithms appear that they could use additional refinement. The authors also noted that their algorithms appear to be slower than they would optimally like. The paper also notes limitations of dealing with clusters of regions next to each other. Ideally, the smaller vectors would need to know something about the blocks around them in order to make better guesses about the background of the image as a whole. Splitting up the image processing into blocks would allow for the algorithm to be multi-threaded, which would give this approach an advantage over other algorithms. (Jeong-In Park)

The third algorithm we found was a paper written for Microsoft Research by authors Jian Sun, Weiwei Zhang, Xiaoou Tang, and Heung-Yeung Shum. This paper deals with removing the background image in videos. Their algorithm again seeks to determine the difference between the foreground and the background of the image. The authors note that the background subtraction is relatively easy to do with videos if the algorithm has two cameras trained on an object, but note that most people do not have two webcams connected to their computers. The algorithm uses a Gaussian mixture model to try to determine if a pixel belongs to the foreground or the background. The authors note that dealing with sudden changes in lighting is the biggest hurdle their algorithm must overcome. (Jian Sun)

Our team believes the shortcomings of these algorithms give us opportunities to improve on existing techniques. Our team will use our knowledge from this class in C++ and OpenCV to try to determine the subject by using video techniques in OpenCV. Our team also has a strong mathematical and programming background to give us a fresh perspective on this challenge.

# Previous Experience

Our team is comprised of three people: Edward Jezisek, Nichole Dugan, and Brian Schulte. Edward Jezisek has experience with mobile development and a strong interest in developing consumer based applications. He is currently a programmer for Verizon Wireless, and is incredibly interested in algorithms, data structures, overall systems architecture, and learning more. Some of his Android applications have been featured on Fox News and XDA-Developers.

Nichole Dugan is a C# developer working for the state of Iowa in the Department of Transportation. She has been working with driver’s license issuance since 2007 and has helped with the facial recognition program in the state of Iowa. She has also worked with vendors for the state of Iowa implementing self-service kiosks for the state, and recently has worked with a vendor to interface with the state of Iowa’s system of record for the mobile driver’s license project.

Brian Schulte is a software engineer for General Dynamics Mission Systems. He has experience in C++ and Java. In addition to his programming skills, Brian has a minor in Math and strong skills and experience in developing algorithms. Brian’s experience in systems engineering has also helped with managing the project, project requirements and system testing.

# Approach

Our approach to this project was two-pronged. First, we researched current algorithms already in use, and examined whether these algorithms could be modified to accomplish our goal. Second, we designed a high-level process that is streamlined, scalable and secure for users to import pictures taken with many types of cameras in many environments, manipulate them easily (e.g. clicking a button rather than manually altering), and save a picture that meets the necessary specifications into the database.

To accomplish this, we divided our project into 4 phases.

1. Phase 1 involved developing a way for users to upload images that will be used for their license. Users are able to use a camera to capture video or upload a previously saved photo. Once the picture is loaded into the application we manipulate it to conform to the Iowa DOT standards.
2. Phase 2 of the project was to create the algorithms used to alter the photographs. More specifically, during this phase algorithms were developed to find the background, determine the edge of the face without cropping out hair, and replace the background of the photo from Phase 1 with a blue background.
3. Once the algorithms were in place, Phase 3 integrated Phase 1 and Phase 2 into an Android application. Users are able to take photos using an Android application and then submit them to be used for their driver’s license.
4. The American Association of Motor Vehicle Administrators (AAMVA) and the International Civil Aviation Organization (ICAO) define several criteria which photos used for driver’s license and identification cards must meet. These requirements include rules such as no sun glasses, well-lit with minimum shadows, no hats, and many more. Figure 2 shows the photo requirements for passport photos imposed by the ICAO. During Phase 4 we implemented conformance checks on the uploaded photos for several of these requirements. This phase was a stretch goal for our team. Because of this, some ICAO requirements were not imposed. (American Association of Motor Vehicle Administrators)

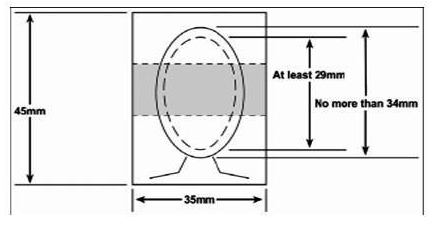


Figure 2

# Overall Requirements

AAMVA is the governing body regulating driver’s license documents. According to AAMVA standards, the background for driver’s license photos is defined as: “Background. A uniform light blue color or white background shall be used to provide a contrast to the face and hair. Note: Preference is for uniform light blue color, such as Pantone 277 (though the specific Pantone color is not a requirement – a uniform light blue color or white background is a requirement)”. (American Association of Motor Vehicle Administrators)

An example photo from a driver’s license is shown in Figure 3.



Figure 3

A typical driver’s license station with the photo backdrop is shown in Figure 4. You can see from the image that the blue backdrop is placed behind the camera in order to ensure that the photo in Figure 3 contains the required background coloring.



Figure 4

When kiosks were introduced as a solution for citizens of the state of Iowa to be able to renew or replace their driver’s licenses without visiting a driver’s license station, certain limitations were required because of the backdrop functionality. The kiosk needs to have a standing or mounted backdrop behind the photo area to allow for the photo to have the required blue backdrop. News station KCCI reported on the state’s installation of the kiosks in the state of Iowa, with their news story focusing on the kiosk in the West Des Moines library. In the video, you can see how the kiosk needs to have a particular configuration in its location. (KCCI)

In addition to the kiosks, the state of Iowa has recently introduced a pilot program to allow for mobile driver’s licenses. One of the suggested features has been allowing the user to take a photo from their device and use that as their driver’s license photo.

# Hardware Requirements

For this project a camera was needed. Photos on driver’s licenses need to be at least 72 dpi (dots per inch). Although dpi is a printer specification, the camera needs to be able to produce pictures with a ppi (pixels per inch) equivalent to the 72 dpi requirement. To produce this level of quality, the camera needs to support at least 1 megapixel (MP). Since most cameras commercially available today are over 1 MP, almost any camera should be acceptable. These pictures will be approximately 4MB in size (or smaller). For our project to be successful we need to be able to process all of the drivers in Iowa in a one month time period. As there are 2.3 million drivers we need to be able to process approximately 8 terabytes worth of pictures per month. In addition, if this is to scale to multiple states we must ensure that our solution is distributive and able to scale out to be run on multiple machines. Our initial product runs on the Ubuntu operating system, but future releases may use Red Hat for additional scalability.

# Software Requirements

Our application was created using C++ and the gcc (Ubuntu 4.8.4-2ubuntu1~14.04) 4.8.4 compiler, the OpenCV 3.1.0 library and Boost. This allowed us to use the library based functions inside of OpenCV and develop intermediary tests for our application using the Boost::Test framework. Finally, we provided several videos and pictures that were used in our application to ensure that the provided photos met necessary standards. Some of the processing was done on the user’s device in order to prevent the server from receiving and processing too much of the required load.

This required us to include parts of our code in Android. The specific version of Android that we used was 5.0.1 running on an HTC One. It may run on other phones as well, but due to the scope of the project we limited development and testing to a single Android version and device. Other applications could be developed to ensure access by as many users as possible. An Android phone was chosen for the convenience of having a device we already own that takes pictures and video, and has a connection to the Internet. We felt that many of our customers would have a similar device and that they would be able to use the product.

# Additional Tools

1. In addition to the software requirements and tools listed above, our team utilized a few other tools to assist with project management and development.
2. GitHub was used for version control. All documents, presentations, and source code were maintained by version control. This ensured efficient collaboration among team members and that all members could participate by adding their changes easily.
3. Microsoft Office was used for developing project papers and presentations. All members of the team had access to the Microsoft Office suite by virtue of being Iowa State students. Word is also an established standard for document writing in most environments.

Project deadlines and issues were tracked in Yodiz, a web-based project management tool. This allowed the team to assign specific tasks to each member of the group and ensure that deadlines were met.

# Preliminary Results

Our preliminary attempt in algorithm development involved using the MOG2 background subtraction algorithm available in OpenCV. In addition, the algorithm uses a facial detecting algorithm using Haar Cascades. These combined approaches try to determine the background of the image from the video by focusing on the non-moving aspects in the background versus the moving aspect in the foreground. The algorithm secondarily uses facial recognition and determines that everything within the bounding ellipse of the face is not in the background.

1. Preliminary results of testing with a background subtraction yielded the results shown in Figure 5. As you can see, while the algorithm did a fairly decent job it still improperly identified some of the subject’s facial features as being part of the background. It also incorrectly determined pieces of the background were in the foreground.
2. 

Figure 5

# First Approach

The first attempt to solve this problem was to use background substitution on a video. First the video recorded the background by itself and then the subject moved into frame. As the subject moved, the detected background was converted into a blue background. The relevant source code used the OpenCV built in function createBackgroundSubtractorMOG2.

This source gave us the results shown in Figure 5 above. While this algorithm was fairly good at detecting the background, you can see that the body was still identified as part of the background. The next part of this approach was to add facial recognition using Haar Cascades to try to isolate the subject and their corresponding bodies. Since the Haar Cascasde facial recognition algorithm yields an area that is contained in a square, the area to recognize as the face needed to be converted to a circle or an ellipse to match the subject’s facial features. Using facial recognition yielded the photo shown in Figure 6.



Figure 6

As you can see from the above photo, while the face had a better area of recognition, the shirt in this photo was still identified as background. The team gave consideration to using Haar Cascades to identify the shirt in the photos, but after consideration of alternate videos, such as the results in Figure 7, the team decided to try an alternate approach. The relevant code snippets are shown below.

vector<Rect\_<int> > faces;

detectFaces(frame, faces, face\_cascade\_path);

Mat faceImg = frame.clone();

Mat m=frame.clone();

if(faces.size()>0)

{

Rect face = faces[0];

faceImg.setTo(Scalar(0, 0, 0));

int width=face.width;

int height=face.height;

ellipse(faceImg, RotatedRect(Point(face.x + (width/2), face.y + (height/2)), Size(width,height), 0),

Scalar(255, 255, 255), -1, 4);

ellipse(display, RotatedRect(Point(face.x + (width/2), face.y + (height/2)), Size(width,height), 0),

Scalar(0, 0, 0), -1, 4);

m=frame&faceImg;

Mat disp=m|display;

imshow("FG Mask MOG 2", disp);

}

static void detectFaces(Mat& img, vector<Rect\_<int> >& faces, string cascade\_path)

{

CascadeClassifier face\_cascade;

face\_cascade.load(cascade\_path);

face\_cascade.detectMultiScale(img, faces, 1.15, 3, 0 | CASCADE\_SCALE\_IMAGE, Size(30, 30));

return;

}



Figure 7

# Secondary Approach

The second algorithm written by the group was designed to use a weighted average of frames to more efficiently produce the background to be subtracted from each additional frame. As with the MOG2 background subtractor used in the first approach, the background was built up at the beginning of the video. The subject then entered the frame and the background was removed. This approach allowed the background to continually evolve and adapt to changes, including lighting variations. This portion of the algorithm can be seen in lines 6 - 18 below. In order to keep the subject from being built into the background, frames were not used for the background if they contained a face (using facial recognition). To increase the accuracy and adaptability, the algorithm used a configurable number of maximum frames to be used for the background. An example of this approach can be seen in Figure 8.

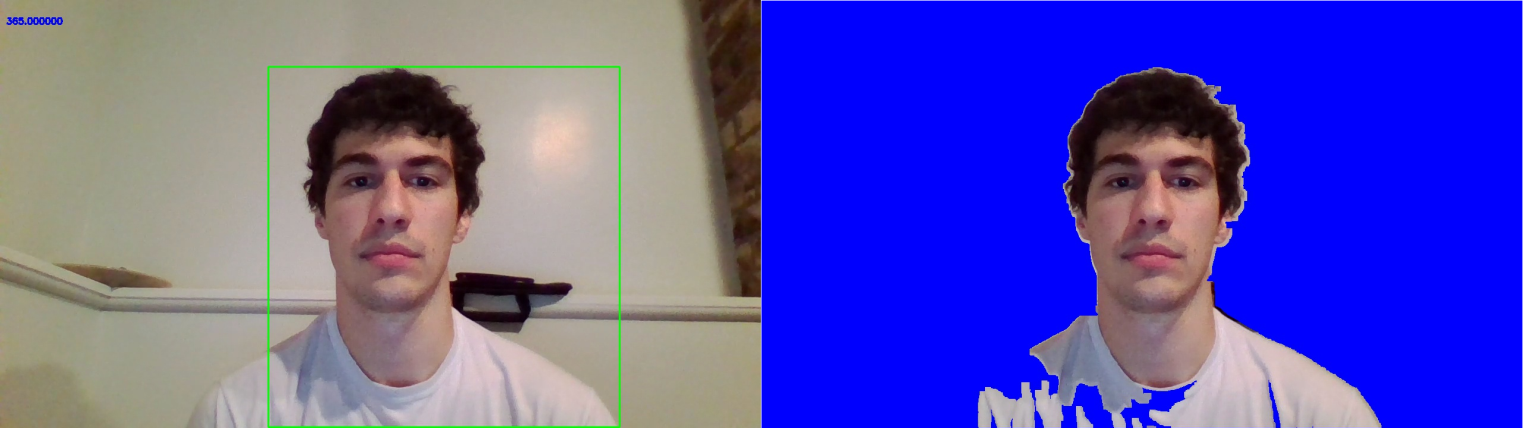


Figure 8

The results of this algorithm were fairly accurate and repeatable, as seen in Figure 9.

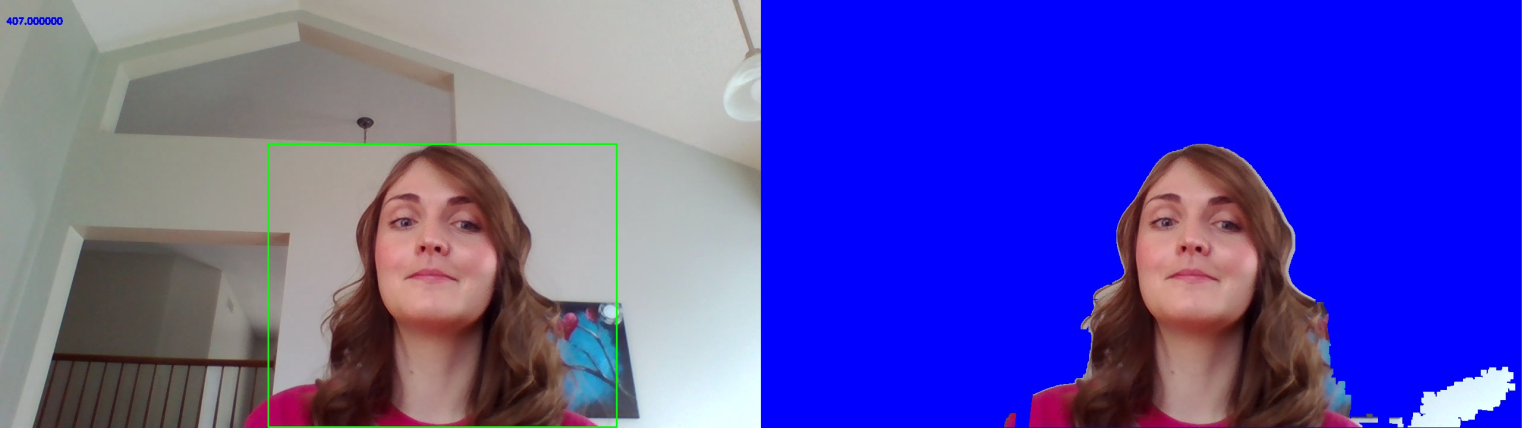


Figure 9

Although this algorithm produced good results, there were a few shortcomings. As stated above, the algorithm required time to learn the background. This is not much of a problem in a kiosk setup, but it would not work well in a mobile application. A key observation made by the team during this approach was the limitations of dilation and eroding to remove noise. Dilating and eroding techniques were used for many of the homework assignments and they seemed to work very well. Dilation was able to fill gaps as seen in Figure 8, and eroding can be used to erase noise, such as light reflections, as seen in Figure 10.



Figure 10

However, particularly when working with facial detection, eroding and dilation will remove and add pixels around individual pieces of hair. The frame in Figure 11 is the same frame as seen in Figure 10. The noise has been removed from the background and the foreground has been filled in. The downside is the border around the entire head has been smoothed out. Individual strands of hair have been removed and some hairs have been filled in to appear as one. This is similar to some of the difficulties encountered in previous attempts to solve this problem.

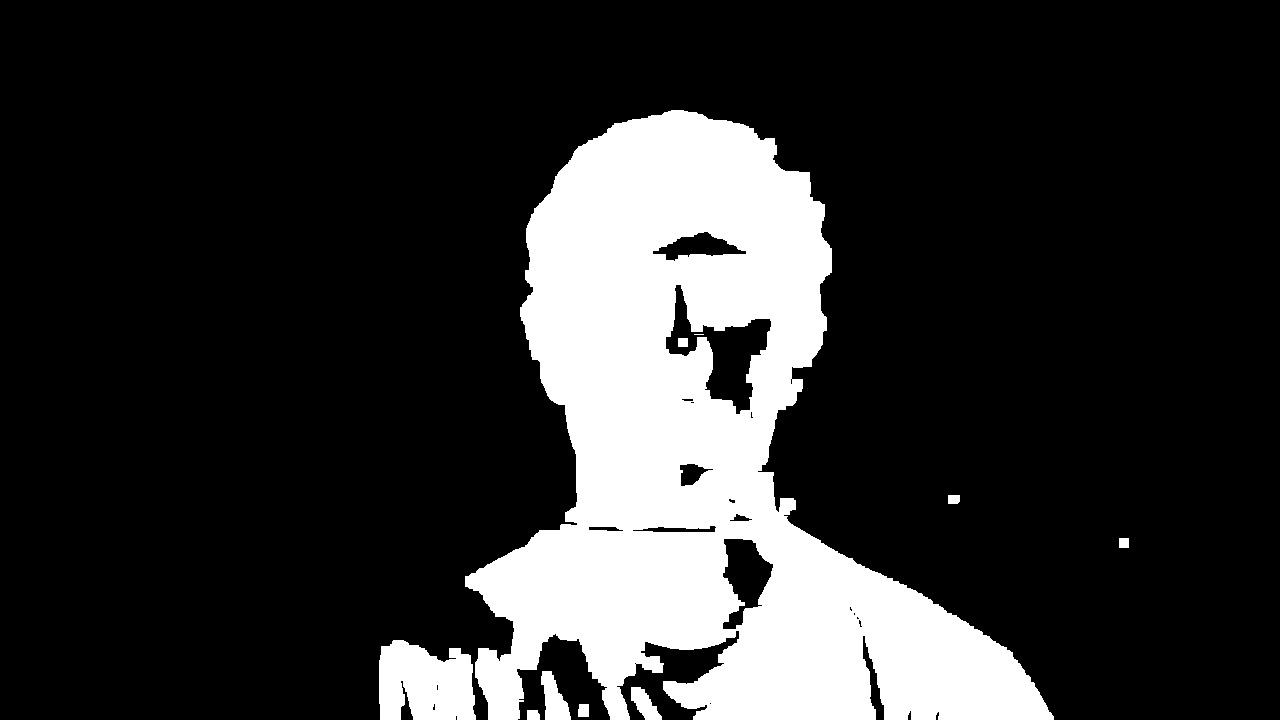


Figure 11

The team decided rather than continuing to work on this algorithm, which had its mobile limitations and experienced setbacks similar to previous failed attempts by others, an altogether different approach was needed.

The relevant code snippet for this approach is shown below.

if (backGround.empty()){

backGround = frame.clone();

faceFrame = frame.clone();

}

// Use weighted frames to maintain background

if (i >= 100)

{

k = 100;

}

else

{

k = i;

}

k = i \* 100;

alpha = ((k-1)/k);

beta = (1/k);

addWeighted(backGround, alpha, frame, beta, 0.0, backGround);

cvtColor(frame, frmGray, COLOR\_BGR2GRAY);

cvtColor(backGround, bkgdGray, COLOR\_BGR2GRAY);

se = getStructuringElement( MORPH\_RECT, Size( 1, 4 ), //remove bkgd noise

Point( -1, -1 ) );

erode (bkgdGray, bkgdGray, se, Point(-1, -1), 1, 1, 1);

dilate(bkgdGray, bkgdGray, se, Point(-1, -1), 1, 1, 1);

absdiff(bkgdGray, frmGray, diffGray);

threshold(diffGray, notDiff, 20, 255, THRESH\_BINARY);

// Only run if motion is present (ie > 15% of frame)

perOn = (double)(countNonZero(notDiff)) / (double)((notDiff.cols \* notDiff.rows));

if (perOn > 0.14)

{

erode (diffGray, diffGray, se, Point(-1, -1), 1, 1, 1);

dilate(diffGray, diffGray, se, Point(-1, -1), 3, 1, 1);

se = getStructuringElement( MORPH\_RECT, Size( 4, 1 ), //remove bkgd noise

Point( -1, -1 ) );

erode (diffGray, diffGray, se, Point(-1, -1), 1, 1, 1);

dilate(diffGray, diffGray, se, Point(-1, -1), 3, 1, 1);

//bitwise\_xor(diffGray, noiseBkgd, diffGray);

threshold(diffGray, notDiff, 20, 255, THRESH\_BINARY);

xorFrame = notDiff.clone();

// Track Person

cntrIn = xorFrame.clone();

findContours(cntrIn, contrR, hierarchy, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE, Point(0, 0));

for(int c = 0; c < contrR.size(); c++)

{

if(contrR[c].size() >= largestContr)

{

bounding\_face = boundingRect(contrR[c]);

//drawContours( frame, contrR, c, Scalar(0,255,0), 1, 8, hierarchy, 0, Point() );

//rectangle(frame, bounding\_face, Scalar(0,255,0),2, 8,0);

//polyTest = pointPolygonTest(contrR[c], ctr, false);

largestContr = contrR[c].size();

}

}

rectangle(frame, bounding\_face, Scalar(0,255,0),2, 8,0);

largestContr = 0;

/\*/ Debug \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

namedWindow("xorFrame", WINDOW\_AUTOSIZE);

imshow("xorFrame", xorFrame);

// Debug \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

# Facial Tracking

The facial tracking algorithm was the team’s first attempt to use static photos to try to remove a background. While the algorithm yielded generally good results, some of the problems identified with previous algorithms occurred with this method, such as trouble distinguishing the subject’s hair. An example of this problem is shown in Figure 12.



Figure 12

# GrabCut

The final algorithm the team decided on was the GrabCut algorithm in OpenCV. GrabCut was developed in 2004 by Microsoft Research of Cambridge, UK. (Blake) The algorithm is based on a method called graph cut (among others). GrabCut relies on two techniques. The first requires user input into determining the primary image from the background. Anything outside of this user set boundary is considered to be part of the background. Using the remaining “foreground,” GrabCut utilizes a Gaussian Mixture Model. Through the process of several iterations, GrabCut compares neighboring pixels to each other. If two pixels are vastly different, a high probability is assigned to that pixel as being the border between the foreground and the background. Some of the capability of GrabCut can be seen in Figure 13 (Blake).



Figure 13

The team began working with the GrabCut algorithm. Immediately it worked better than several previous attempts, as seen in Figure 14. However, the goal was to have minimal user input at the kiosk or mobile application. To reduce the user interaction, the facial detection developed earlier was integrated with GrabCut to select the face based on the presence of the face. Although this approach seemed to work, it took quite a while to compute: up to six minutes on larger images. This computation time was deemed unacceptable. Because of the nature of this project, it was noted that for most test images uploaded by the team, the face comprises a majority of the total image. The GrabCut algorithm was modified to select the entire picture. This approach produced good results. Additionally, the team considered having an on screen box to help the user align his or her face correctly within the picture. At this point, it was determined to be unnecessary.



Figure 14

Below is the team’s first attempt to use no user input to make the process easier for users, Figure 15.



Figure 15

As you can see from the above image, the results were nearly perfect. While this is an excellent result, the image background for this image was relatively plain. In the image shown in Figure 16, you can see the algorithm works well even with more complex backgrounds as the subject is standing in front of a pine tree.



Figure 16

The relevant sections of the GrabCut code are shown below.

void setRectInMask(Mat &mask, Rect &rect)

{

CV\_Assert( !mask.empty() );

mask.setTo( GC\_BGD );

rect.x = max(0, rect.x);

rect.y = max(0, rect.y);

rect.width = min(rect.width, mask.cols-rect.x);

rect.height = min(rect.height, mask.rows-rect.y);

(mask(rect)).setTo( Scalar(GC\_PR\_FGD) );

}

void getBinMask( const Mat& comMask, Mat& binMask )

{

if( comMask.empty() || comMask.type()!=CV\_8UC1 )

CV\_Error( Error::StsBadArg, "comMask is empty or has incorrect type (not CV\_8UC1)" );

if( binMask.empty() || binMask.rows!=comMask.rows || binMask.cols!=comMask.cols )

binMask.create( comMask.size(), CV\_8UC1 );

binMask = comMask & 1;

}

void getGrab(Mat &source, Mat &dest, Rect& rect)

{

Mat bgM, fgModel;

dest.create(source.size(), CV\_8UC1);

if(!dest.empty())

dest.setTo(Scalar::all(GC\_BGD));

rect.width=source.cols-5;

rect.x=5;

rect.height=source.rows-10;

rect.y=10;

setRectInMask(dest, rect);

cout<<"THE RECT IS: "<<rect<<endl;

grabCut(source, dest, rect, bgM, fgModel, 5);

getBinMask(dest, dest);

}

void getImages(Mat & source, Mat &dst, Rect& r)

{

Mat cropped=source.clone();

Rect rect(r);

Mat mask;

getGrab(cropped, mask, rect);

Mat face=mask.clone();

Mat facecropped=cropped(rect);

/\* rect.height/=3;

getGrab(cropped, mask, rect);

Mat hair=mask.clone();

Mat haircropped=cropped(rect);

rect.y+=2\*rect.height;

getGrab(cropped, mask, rect);

Mat shirt=mask.clone();

//cropped.copyTo(shirt, mask);

\*/

Mat totalMask=face;

cropped.copyTo(dst, totalMask);

Mat white=cropped.clone();

white.setTo(Scalar(255, 255, 255));

Mat black=cropped.clone();

black.setTo(Scalar(0, 0, 0));

white.copyTo(black, totalMask);

black=~black;

imshow("Black", black);

Mat blue=cropped.clone();

blue.setTo(Scalar(232, 209, 181));

blue=black&blue;

imshow("Blue", blue);

dst|=blue;

// Mat shirtcropped=cropped(rect);

// dst=face|shirt|hair;

}

void colorReduce(cv::Mat& image, int div=256)

{

int nl = image.rows; // number of lines

int nc = image.cols \* image.channels(); // number of elements per line

for (int j = 0; j < nl; j++)

{

// get the address of row j

uchar\* data = image.ptr<uchar>(j);

for (int i = 0; i < nc; i++)

{

// process each pixel

data[i] = data[i] / div \* div + div / 2;

}

}

}

void drawContours(Mat &source, Mat& dest)

{

Mat source\_gray;

cvtColor(source, source\_gray, CV\_BGR2GRAY);

const int max\_BINARY\_value=255;

adaptiveThreshold(source\_gray, source\_gray, max\_BINARY\_value, CV\_ADAPTIVE\_THRESH\_MEAN\_C, THRESH\_BINARY, 3, -2);

std::vector<std::vector<Point> > oaoa;

findContours(source\_gray, oaoa, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE, Point(0,0));

int i=0;

for( std::vector<std::vector<Point> >::iterator iterator=oaoa.begin(); iterator!=oaoa.end(); iterator++, i++)

{

std::vector<Point> p=\*iterator;

const Point \*points[1]= {&p[0]};

const int lineType=1;

int npt[]={(int)p.size()};

fillPoly(dest, points, npt, 1, CV\_RGB(255\*abs(sin(i\*i\*.0174533)),255\*abs(cos(i\*i\*i\*.0174533)),i), lineType);

}

}

# Alternate Approaches

The team also considered using an edge detection algorithm that was developed for one of the homework assignments. This approach would use the OpenCV function findContours. The premise of this technique would be to identify all edges and solid objects in a frame. Once all of the solid objects and edges were detected, the largest object would be assumed to be the face. Facial tracking would be performed to ensure the object was a face and everything outside the object would be treated as background. Unfortunately, this design is limited to some of the problems discussed above in background subtraction and the use of eroding and dilation. Some of the preliminary results of this method can be seen in Figure 17.



Figure 17

# Evaluation

1. To determine success of an algorithm, testing needed to be performed. Testing for each algorithm was done with a sampling of videos or pictures from each of the three members of their team. This ensured that each of the videos or pictures were taken in different lighting conditions with different facial features of the subjects. Visual inspection of the results compared to how well the background was removed was how the algorithm was determined a success or failure. A more detailed explanation of our testing process is explained below.

# Test Plan

1. The smoke test consisted of an image being processed and displaying just the face. To do this automatically we obtained an image that has been processed and its original input. We compared the output to the result and had a threshold for differences. If there are too many differences, the smoke test was deemed a failure. This ran for several small subsets of images. Furthermore, we included tests that should fail, for example pictures without anyone included. We performed these tests with random objects entering the screen, and more than one individual in the picture. These tests gave a fast turnaround regarding whether or not a feature was successfully included. If the pictures were different enough a comparison image was displayed to the developer. This allowed the developer to either accept or reject the new image.
2. Following the smoke test, a regression test was performed. This regression test consisted of the user manually checking many of the photos included to prevent inaccuracies. This test highlighted any key differences in pixels between the previous and current picture. This allowed the developer to easily notice any changes in the application, whether or not they were positive. The regression test also took in at least one new video that the user performed and added this video to the list of tests to be performed in the full regression. This video was taken with webcam to ensure that our application was able to extract an image from a new video as well as the old tests.
3. Finally, a load test was performed. This load test was needed to ensure that our server can handle the computational load of the inserted pictures. For this project we aimed to be able to upload 100 pictures per hour for the server. If this is inserted in a kiosk, the kiosk may need a small computer or system to perform the processing. Approximately one photo processed per minute should be acceptable for our application.

# Determination of Success

1. If all of the previously defined tests run successfully our application was considered a success. The previous test cases prove that our application is successful with video and pictures from a webcam or mobile phone, and that our application is able to withstand a specific amount of load. The tests demonstrate accuracy as well as speed and automatically deduce the application’s accuracy. Finally, complete success would include being included in the DOT’s kiosk based product. Our attempt should be able to fix many of the issues in current approaches.
2. Finally, if our solution's pictures meet the requirement for a driver's license in 90% of the pictures taken, our application will be a success. This will require continuous analysis after the completion of the project to ensure quality; but it will hopefully remove many of the issues apparent in the current approach. These pictures must be accessible to facial recognition and ensure that a user is the same as their previous photo.

# Test Conditions

1. Our application tests consisted of a minimum of three videos or photos in the regression test. These videos or photos will be of each team member. The reason for this was to ensure the quality of our solution. We will likely add more videos or photos as test evidence, but at this point three photos or videos will be required for the successful completion of our application.

# Test Subjects

1. As our application consisted of multiple videos or photos in the regression test, we required a separate subject for each of these pictures or videos. This ensured our application works for more than one person and was convenient as we are a group of three people. More people could be added and more videos could be taken to enhance the necessary regression test.

# Evaluation of Results

1. Our results were evaluated by comparing what we have created with a standard driver’s license photo. If our application fell short of creating acceptable photos with a blue backdrop, the problem needed to be investigated. Our results will be evaluated based on how accurately we can create a driver's license photo with a background. This background will be of varying colors. Some backgrounds will be unacceptable due to them being close in color to hair or skin. If the display does not change, it may be impossible to recognize a face correctly.
2. The initial results of our application will be compared manually as a final evaluation metric. However, in the future we will automatically compare previous successful results with the most recent result. This will allow us to potentially speed up development and create an acceptable solution to this problem.

# Improving the Application

1. After our initial goals are met, we will try to improve our application. We will ensure that our results are comparable with the previous results or manually input them as the new base image. This will allow for fast development and it will ensure the quality of our application. After our application is successful with a large majority of faces, the application will no longer need to be “improved” and will be ready for production implementation.
2. If we are able to meet the first phases of our project early, we will look into additional features. Specifically, we would like to include having the application running on multiple Android devices and OSes. Also, as we want to improve the perception a user has over their picture; we will potentially allow them to do some image editing on the final picture. This editing naturally must be compliant with the aforementioned requirements.
3. We will also maintain compliance to the ICAO standards. This will ensure that our product is usable and acceptable to the state of Iowa. If performance becomes an issue, we will measure various aspects of our application and make modifications to provide a quality project. This will provide for a solution that meets federal requirements and will ensure that our project is a success. These standards include the size of a picture needed for a driver's license and the background image that needs to be included. We hope to print at least one imaginary driver's license to prove the concept we have created. We plan to display video of us taking the picture, inputting it into a driver's license format, and displaying the resulting image.

# End Result

Since the end result of our application was to allow a licensed driver or identification card holder to have the ability to use any photo they wished for his or her photo, the team decided to mock up a driver’s license using the results from Figure 15 as an example for the end product. The resulting driver’s license image is shown in Figure 18.

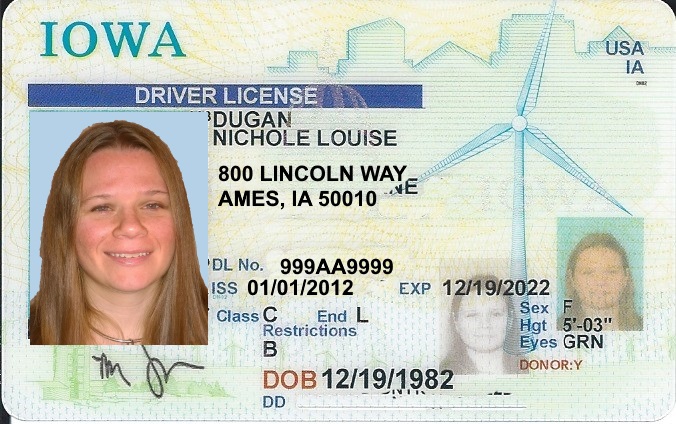


Figure 18

As it can be seen in the image, the photo would easily pass for a user’s driver’s license photo while allowing the user to supply or take a more flattering picture.

# Conclusion

The team undertook this project because we felt it had both practical applications as well as potential entertainment uses. While the project focused mostly on using the produced photo for a driver’s license image, the background could instead be replaced with something fun or humorous for use in social media or other entertainment purposes.

To complete this project, we researched previous approaches to understand deficiencies that other teams had encountered when trying to solve this problem. As noted, two major issues included difficulty recognizing hair properly and lighting problems causing incorrect identification of the background. The team also noted these problems as we worked through several algorithms to identify the best solution for our project. In the end, the GrabCut algorithm works well for the project as proposed and the results appear to be fairly good. With more time the algorithms could be further refined and a better product could be produced.

This project could have useful applications not only for the citizens of the state of Iowa, but also much more broadly with improvement and widespread adoption. We believe that our team’s success was not only a combination of our computer science and engineering backgrounds, but equally important was our research into how other teams have tried to solve this problem unsuccessfully, ultimately improving on their techniques. By using industry standard tools such as Git, Microsoft Office, and issue tracking software, we believe we were able to manage our project effectively and use our resources to their full potential. Our team enjoyed using concepts learned in class to create a real world project that could have potential benefits to real citizens.

# References

American Association of Motor Vehicle Administrators. (2016, February 10). *AAMVA DL/ID Card Design Standard.* Retrieved from AAMVA: http://www.aamva.org/WorkArea/DownloadAsset.aspx?id=4435

Iowa Department of Transportation. (2016, February 20). *Kiosks*. Retrieved February 20, 2016, from http://www.iowadot.gov/mvd/kiosks.html

Jeong-In Park, J.-T. C. (2016, February 19). *A Background Removal Algorithm using the Variable Order nxm Dimenional Vector.* Retrieved from SERCS: http://www.sersc.org/journals/IJSEIA/vol7\_no6\_2013/22.pdf

Jian Sun, W. Z.-Y. (2016, February 19). *Background Cut.* Retrieved from Microsoft: http://research.microsoft.com/en-us/um/people/jiansun/papers/BgCut\_Camera.pdf

KCCI. (2016, February 18). *List of locations released for new driver's license kiosks*. Retrieved from KCCI: http://www.kcci.com/news/new-kiosks-let-you-renew-your-iowa-drivers-license/33374300

MorphoTrust USA. (2016, February 19). *System and Method for Creating a Virtual Backdrop.* Retrieved from Free Patents Online: http://www.freepatentsonline.com/20150030213.pdf

Wikipedia. (2016, February 19). *Background subtraction*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Background\_subtraction