Comp Photography
Final Project

Dylan Fiedler

Spring 2018

dtfiedler@gatech.edu

FlightFollow

A mobile app that can track the path of your golf ball based on its color and shape.

The Goal of Your Project

Original project scope:

The goal of this project was to create a mobile app that can track the path of your golf ball based on its color and shape.

What motivated you to do this project?

As an avid golf fan I have been amazed with how quickly technology has aided in the advancement of Golf. New technologies like 3D mapping software, golf ball tracking lasers, and so on have helped improve the game for players and viewers alike. My hope was to create a “cheap” and relatively simple replication of the common ball tracking software that allows recreational golfers like myself to learn more about their swing and flight paths of their balls.

Scope Changes

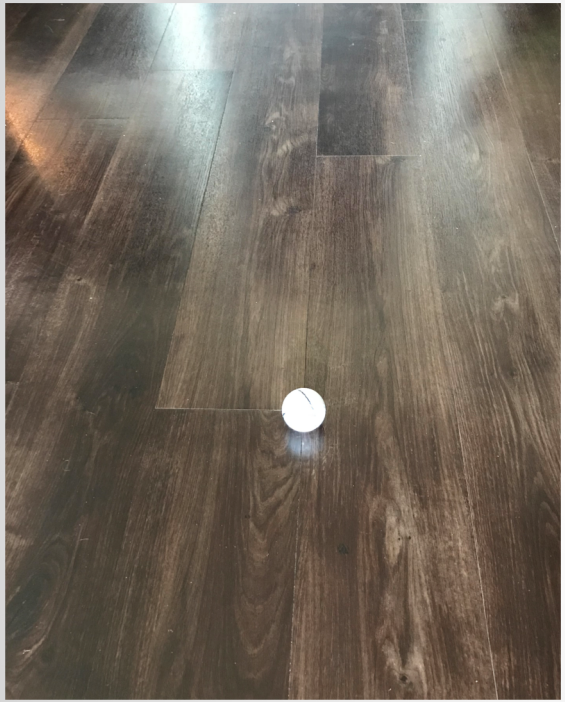
Did you run into issues that required you to change project scope from your proposal?

Yep, I ran into a mirage of issues/challenges that made me have to reduce my scope.

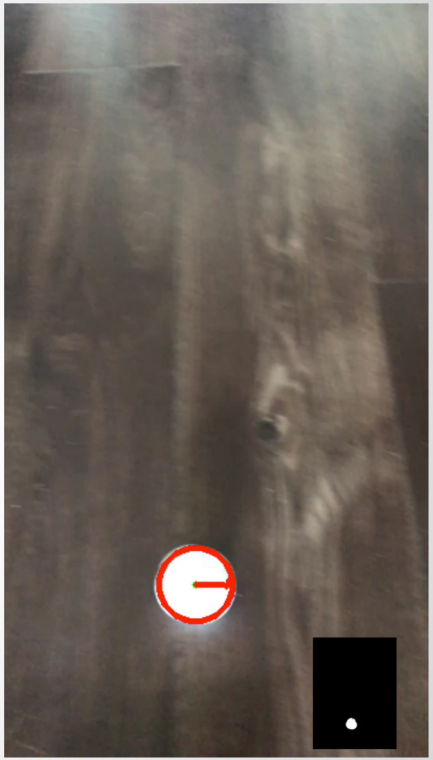
Give a detailed explanation of what changed

Originally, my goal was to start tracking the ball based on color and then dive into more advance techniques such as Kalman filtering and scene awareness. My whole goal was to then package this functionality and embed it into an iOS app. This took the most amount of effort as I have never worked with native iOS or Objective-C before, causing me to reduce the scope of the ball tracking implementation. I also wasn't able to implement what I had done in python, so I then had to convert it to C++ which slowed my overall progress. Overall, my scope narrowed to being able to track a ball's movement within an iOS app and overlay that path to the user real-time.

Showcase



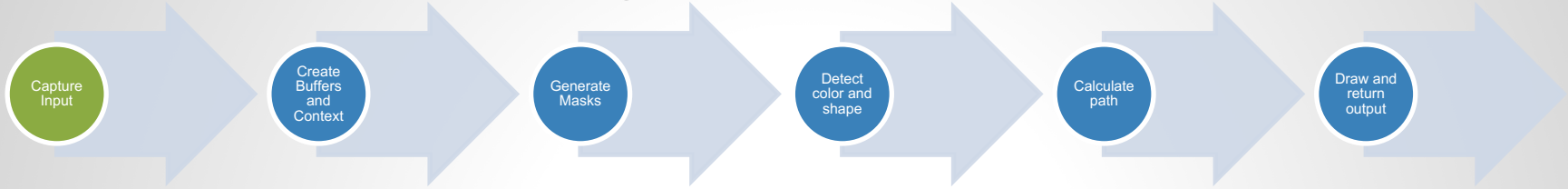
Input



Output:

[View video here]

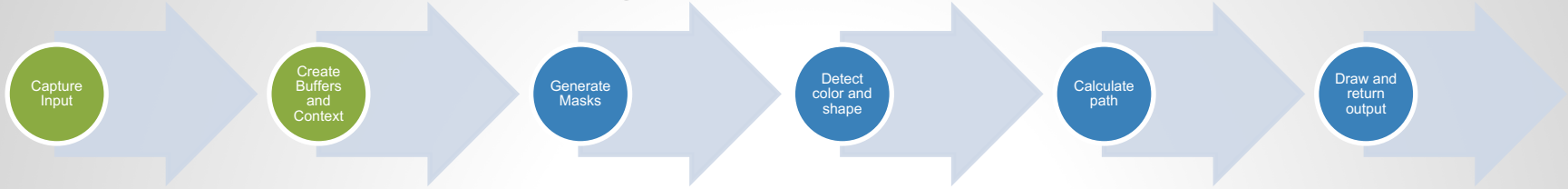
Project Pipeline



The first step is to tell the app what color golf ball you are using, and then request access to the iPhone camera to capture the input. I was unable to implement a more robust color detection feature, so instead it is hardcoded to detect WHITE circles only.

```
func setupCamera(){
    //create an AVSession to capture camera output
    self.session = AVCaptureSession()
    self.session.sessionPreset = AVCaptureSession.Preset.vga640x480
    self.device = AVCaptureDevice.default(.builtInWideAngleCamera, for: AVMediaType.video, position: .back)
    if (self.device == nil) {
        print("ERROR: Device not available")
        return
    }
    do {
        //access front facing camera input
        let input = try AVCaptureDeviceInput(device: self.device)
        self.session.addInput(input)
    } catch {
        print("ERROR: NO DEVICE INPUT FOUND")
        return
    }
```

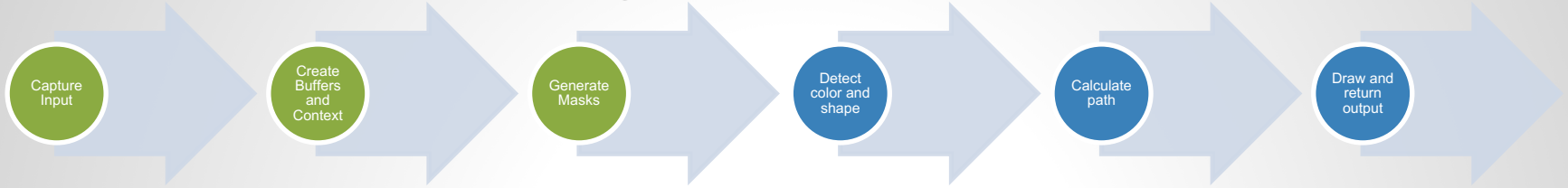
Project Pipeline




```
func captureOutput(_ captureOutput: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) {

    // Convert a captured image buffer to UIImage.
    guard let buffer: CVPixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else {
        print("could not get a pixel buffer")
        return
    }
    let capturedImage: UIImage
    do {
        CVPixelBufferLockBaseAddress(buffer, CVPixelBufferLockFlags.readOnly)
        defer {
            CVPixelBufferUnlockBaseAddress(buffer, CVPixelBufferLockFlags.readOnly)
        }
        let address = CVPixelBufferGetBaseAddressOfPlane(buffer, 0)
        let bytes = CVPixelBufferGetBytesPerRow(buffer)
        let width = CVPixelBufferGetWidth(buffer)
        let height = CVPixelBufferGetHeight(buffer)
        let color = CGColorSpaceCreateDeviceRGB()
        let bits = 8
        let info = CGBitmapInfo.byteOrder32Little.rawValue | CGImageAlphaInfo.premultipliedFirst.rawValue
        guard let context = CGContext(data: address, width: width, height: height, bitsPerComponent: bits, bytesPerRow: bytes, space: color, bitmapInfo: info) else {
            print("ERROR: FAILED TO CREATE CGContext")
            return
        }
        guard let image = context.makeImage() else {
            print("ERROR: FAILED TO CREATE CGImage")
            return
        }
        //USE THE IMAGE WITH OUR BUFFER
        capturedImage = UIImage(cgImage: image, scale: 1.0, orientation: UIImageOrientation.right)
    }
```

Next we need to create the necessary buffers and context to allow us to draw the ball and path on the image itself. Once we have that setup, we pass our live video (the ImageView) to the C++ ball tracking implementation that will return the output with the drawn path. From there, I rely heavily on the OpenCV framework adapted for C++ and Objective-C[5].

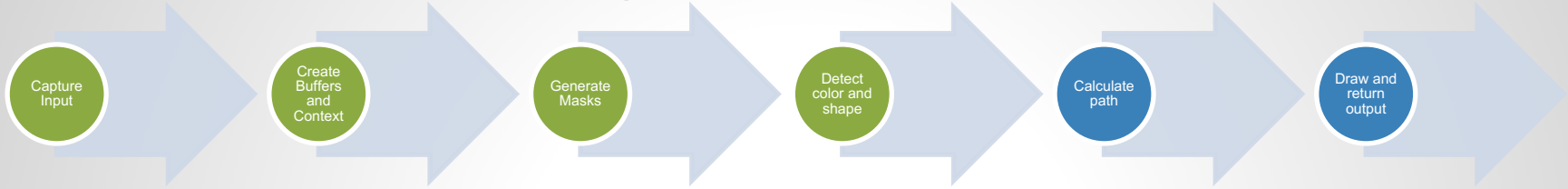
Project Pipeline



Now that we have the raw input we need to convert the raw image to something OpenCV can manipulate, in this case a MatImage. I relied on some help from Hiroki Ishiura to convert the input images to Mat so I could then use OpenCV functions on them [2]. Once I had that, I use Gaussian Blur to generate a mask of the live input, and highlight objects including our golf ball that have colors within the predetermined range. You'll notice that I first convert the original image to hsv, blur it, and then use both erosion and dilation. This is a common technique to reduce the noise of the original input and makes it a little easier to detect the edges of the ball (as described in the next slide) [1].

```
//returns hte mask of the current frames to show as subview
+ (nonnull UIImage *)getMaskedImage:(nonnull UIImage *)image {
    //deteremine the hue of the color ball you want to track
    cv::Mat originalImage;
    cv::Mat hsvImage;
    cv::Mat threshImage;

    //convert image to mat
    UIImageToMat(image, originalImage);

    /// Convert Original Image to HSV Threshold Image
    cv::cvtColor(originalImage, hsvImage, CV_BGR2HSV);
    //find ball in color range proivded, right now hard coded to be white
    cv::inRange(hsvImage, cv::Scalar(0, 0, 240), cv::Scalar(255, 15, 255), threshImage);
    //Blur
    cv::GaussianBlur(threshImage, threshImage, cv::Size(3, 3), 0);
    // Dilate
    cv::dilate(threshImage, threshImage, 0);
    // Erode
    cv::erode(threshImage, threshImage, 0);
    //return our masked image
    UIImage *maskedImage = MatToUIImage(threshImage);
    return RestoreUIImageOrientation(maskedImage, image);
}
```

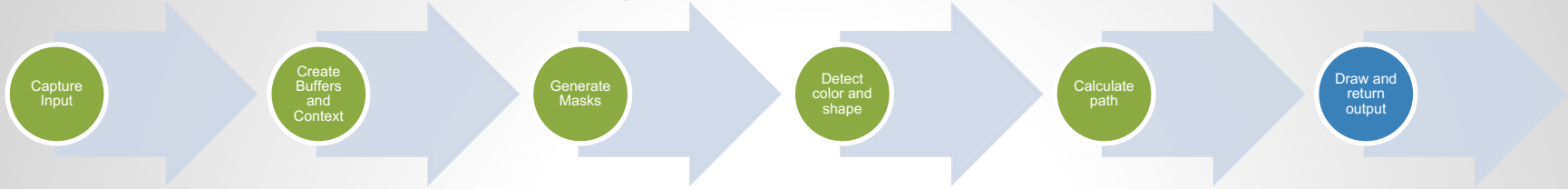
Project Pipeline




```
try {
    // fill circles vector with all circles in processed image
    cv::HoughCircles(threshImg, v3fCircles, CV_HOUGH_GRADIENT, 1,
                     threshImg.rows / 4, //distance between circles
                     100, // Upper threshold for the internal Canny edge detector
                     50, //Threshold for center detection
                     10, //minimum radius
                     800); //maximum radius
    std::cout << v3fCircles.size();
} catch (...) {
    std::cout << "Unable to find circles";
}
```

Once I have a properly generated mask, I can use the OpenCV Hough Circle Transform function that uses the Hough Gradient to detect edges and identify ones that are of circles. I only want to detect one circle (the ball) so when we produce our output we need to make sure the size of v3fCircles (coordinates of the circle's detecting) is not larger then one.

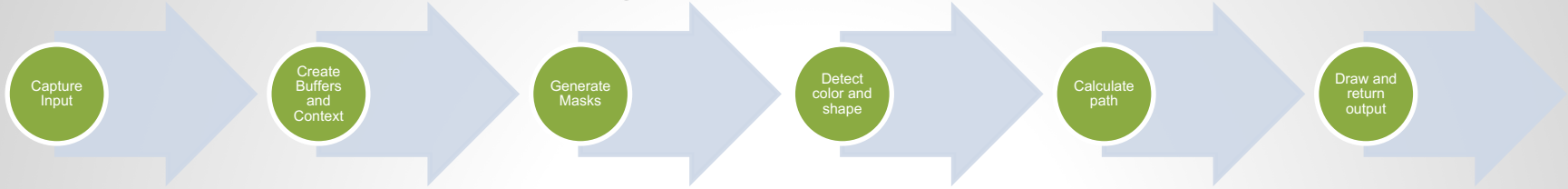
Project Pipeline

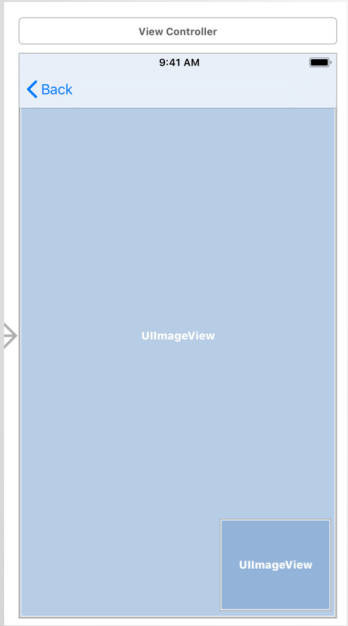


Once the circle of the ball is detected, I draw the outline and center using the coordinates provided by the Hough Circle Transform. I also add the coordinates to an array that tracks the actual path of the ball (up to its 10 most recent locations) and displays it as well. Because I only want to track one ball (i.e. one circle) we limit the output to only draw on the first circle identified in the Hough Transform.

```
int i = 0;
cv::Point center;
//if not already tracking a circle
if (v3fCircles.size() > 0){
    int x_position = v3fCircles[i][0];
    int y_position = v3fCircles[i][1];
    int radius = v3fCircles[i][2];
    // get the center point of the ball
    center = cv::Point((int)x_position, (int)y_position);

    // locate the center and draw a green circle
    cv::circle(imgOriginal, center,3, cv::Scalar(0, 255, 0), CV_FILLED);

    // draw red circle around object detected
    cv::circle(imgOriginal,cv::Point((int)x_position, (int)y_position), (int)radius,cv::Scalar(0, 0, 255),3);

    //add point to to path
    path.insert(path.begin(), center);
    if (path.size() > 10){
        path.erase(path.end() - 1);
    }
    for (int z = 1; z < path.size(); z++){
        //draw red line
        cv::line(imgOriginal, path.at(z-1), path.at(z),cv::Scalar(0, 0, 255),3);
    }
}
```

Project Pipeline

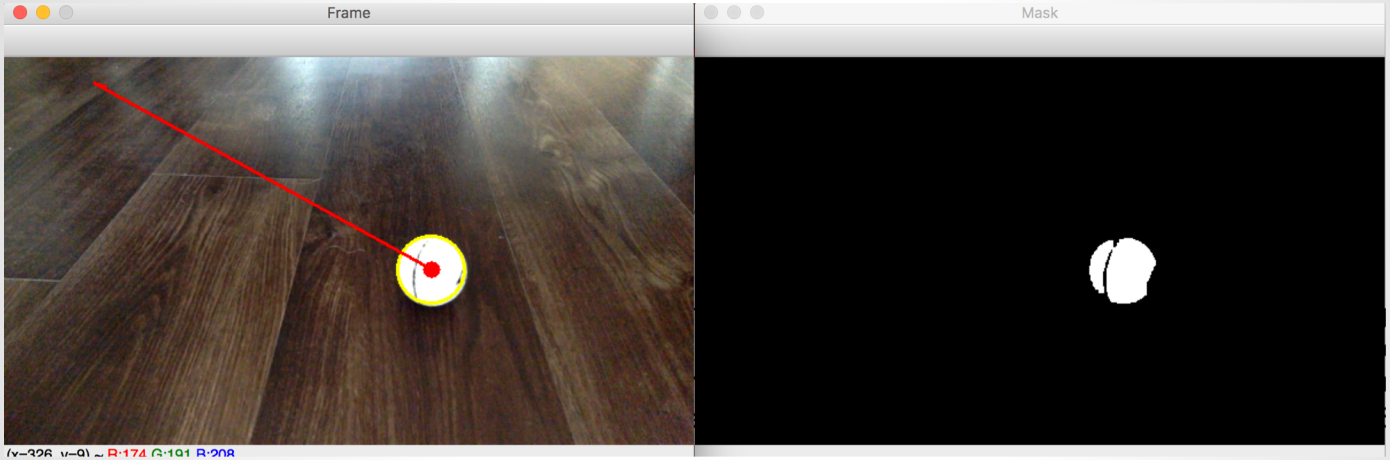




Now I need to display the output back to the user and return the newly drawn image to the ImageView presented from the iPhone. The new ImageView has the drawings of both the circle detecting by the Hough Circle Transform as well as the path of the 10 last locations of the ball. To help determine what is being picked up by the mask, I added a small ImageView of the mask image as well (storyboard view on the left). The final ouput and mask are what the user sees on their phone.

```
let outputImage = OpenCV.trackBall(withColor: capturedImage, "WHITE")
let maskedImage = OpenCV.getMaskedImage(capturedImage)
// display the result on the output, as well as the imageview mask
DispatchQueue.main.async(execute: {
    self.imageView.image = outputImage
    self.maskedImage.image = maskedImage;
})
```

Result Set 2: Computer/Python Implementation

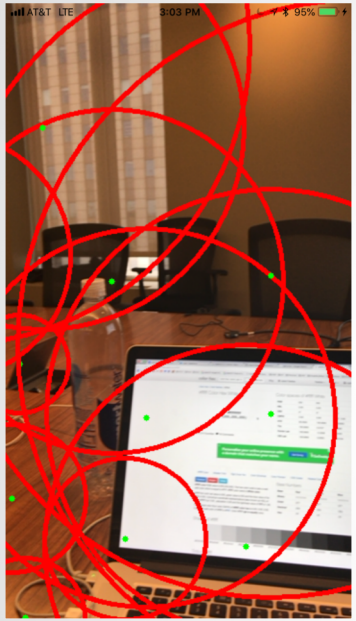


Ball rolling across the floor.

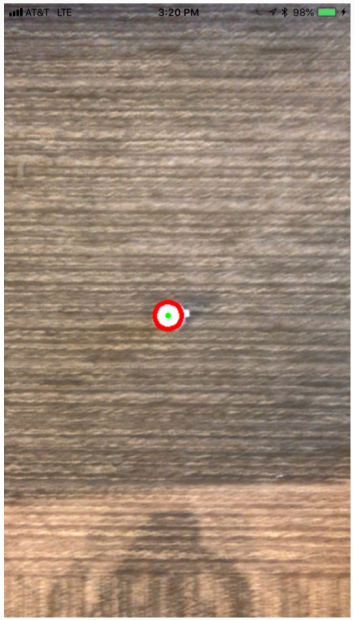
Mask identifying white ball

The original ball tracking implementation was written in Python and used the camera from a computer and inspired by Adrian Rosenback[3]. The goal of my project was to extend that implementation and make it accessible within the iOS app that could interact with the camera directly. This process took quite a bit of time and made the implementation not as clean as I originally had. I relied on some help from Hiroki Ishiura OpenCV to create an iOS implementation and then as able to rewrite the program in Swift, Objective-C and C++.

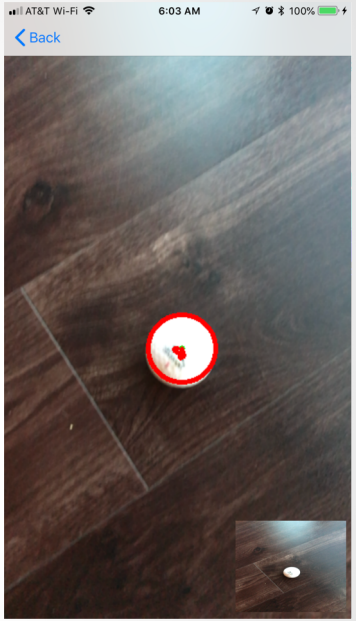
Result Set 2: iOS/Swift Adaptation



Early implementation



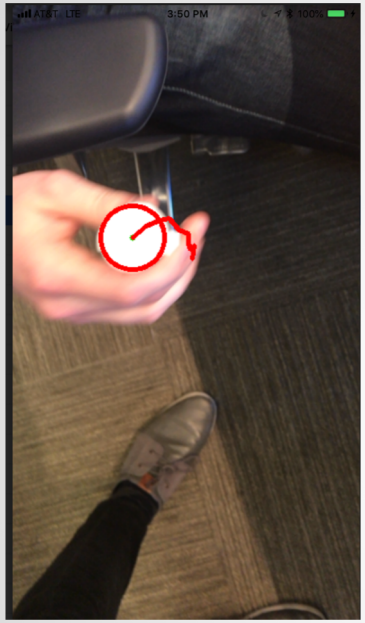
Removing other objects



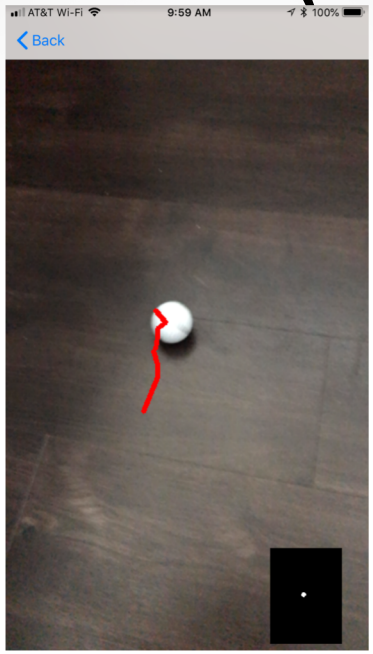
Adding mask preview

This result set shows the adaptation of the python implementation into an iOS app. The input images are now being provided by the iPhone camera as opposed to the computer. As you can see on the left, my earlier versions did not work as expected. It took me some time to determine how to single out the color of the ball separate from other objects in the frame. Images on the left so better captures of detecting the ball, particularly when there is less noise and fewer objects in the video display.

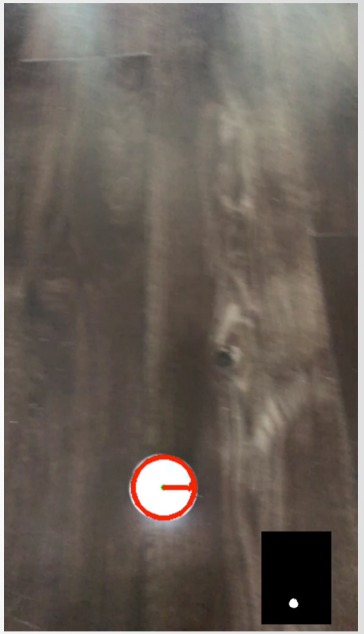
Result Set 3: Putt(ing) it to use



Because I was relying on the shape and size of the ball for tracking, the program could be spoofed by other white circular objects



Slow putts on dark flows were easy to follow



It works..sort of!
View video here

While the original goal was to track the flight of the ball with full swings, I was unable to do much more than track the path of the ball with a putter. The resulting image set and videos available on Google Drive and show me putting the ball with the output and path being recorded from my phone. It wasn't perfect and there is a lot of refinement and modification I want to do as discussed in the next few slides, but overall I was pleased.

Project Development

The first step of my development was to try and implement the ball tracking functionality on my computer using the libraries and frameworks we have used throughout class, including OpenCV and Numpy. That didn't take too long as I was familiar with Python and had a good idea some good resources to start with [4]. The next step was to take that implementation and find how to bundle it into an iOS app. I have some mobile development experience, but have never had to adapt a python script into a mobile app, which proved to be quite difficult. To do so, I needed to use the iOS OpenCV framework, and then modify my python implementation into C++ (woof).

Also, before I could do any of that I needed to get more familiar with Apple Swift. Since I was relying on the input source to come from the phone itself, and as a live feed rather than prerecorded video, I need to determine how to leverage the camera on the iPhone to act as my input source and then overlay the path of the ball on top of that feed.

As for the actual implementation, I would have liked to make it more robust. As you can see from the result sets, it only works when the ball is in a setting with not many other objects, and the movement is relatively slow. Some of the images and videos also show it not being able to detect the ball for a few seconds, or if the movement was too rapid. I found that while using the iPhone camera was the whole point of the project, I was limited by not relying on additionall input sources to help track the location and movement.

Project Development

If I were to do this project again, I would have looked deeper into using techniques like Kalman filtering and something like scene properties to better determine the position of the ball relatively to the golfer or person. My implementation failed to find the ball when it was either more than a few feet away and/or their were several other objects of similar shape and color in the images. I also would have looked into to using other input sources, like my drone and other cameras, that could capture more detail and make it easy for me to test (hard to putt while holding a phone!).

Lastly, I found that my development ended up being more hindered by figuring out how to bundle it as a mobile app as opposed to implementing more advanced computation. I should have looked to work with a partner who was more familiar with mobile development so we could spend more time on the actual computational implementation.

Overall, I learned a lot from this project (and class) and am hoping to continue to build out this app into the summer. My ultimate goal will be able to continue to adapt it and make it more functional and even put it in the app store for recreational golfers to use. I also am much more appreciative of the incredible tracking software and cameras the PGA use for almost every tournament now, I'm at the point where I can't watch golf without it!

Additional Details

Code and Output: All my code and outputs can be found in the Google Drive Folder available here.

Note: If you want to try and run the app yourself, you will need an iPhone and Xcode installed. To do so, open the project in Xcode (available here) and run it on an iOS device (it doesn't work on simulator).

Useful links to run:

http://techapple.net/2015/01/4-waysmethods-install-ipa-file-app-iphone-ipad-ipod-online-offline-methods/

http://www.mechdome.com/update/2016/10/13/testing-android-apps-ios-simulator.html

Resources

[1] Eroding and Dilating - https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[2] Hough Circle Transform - https://docs.opencv.org/3.3.1/d4/d70/tutorial_hough_circle.html

[3] Ball Tracking - https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

[4] Hiroki Ishiura - https://github.com/ura14h/OpenCVSample/blob/master/OpenCVSample_iOS/OpenCVSample_iOS/OpenCV.mm

[5] ObjC with Swift - https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/MixandMatch.html

[6] Object Tracking and Color Separation - http://opencv-cpp.blogspot.com/2016/10/object-detection-and-tracking-color-separation.html?_sm_au_=i0HjjLF16wqjfRQj

[7] Tracking algorithms - https://www.matec-conferences.org/articles/matecconf/pdf/2016/39/matecconf_cscc2016_04031.pdf

Credits or Thanks

Credit to Adrian Rosebrock and Hiroki Ishiura for their inspiration as well as my coworkers for helping capture some photos.

Also, thanks to the professors and TAs for all the work done in this class!