Detecting and Tracking Chess Pieces using a SIFT-based object tracking algorithm

Final Report

Stephen Kim and Alberto Serrano-Calva
Department of Computer Science
Department of Physics
Rochester Institute of Technology
Rochester, NY 14623
{syk4367}{axs4986}@rit.edu

Abstract

The project focuses on using a SIFT open source alternative to implement a video tracking algorithm. This has been combined with a classifier that will be trained using Turi Create with images of chess pieces to be able to track and predict the type of chess piece being displayed. The resulting software will be trained with educational snippets to teach a person how to play the game of chess.

1 Introduction

Like baseball, chess can also be described as the game of life, this is why it is an important game to teach to the younger generation. The app developed is the perfect tool to assist with this task. First, the model was trained using images of chess pieces. All images were web scraped and normalized before the chess piece is hand selected (boxed in) for the model. The model was trained using Turi Create with the default settings. One unique aspect of the library is that if a specific type of classifier is not explicitly used, the library will choose the best model for the data provided. This automatic classifier selection was used to train a basic model while we continued to explore the library.

The second part to the project involved object tracking. We used a SIFT derivative (that was open source) to find unique key features in the frames of a video and worked on a way to map a point from one frame to the next. The reason for not using SIFT is that it is a proprietary algorithm, and is therefore not supported in the latest version of OpenCV, an extensive open source library of computer vision algorithms. One potential candidate is the ORB algorithm, which is OpenCV's own SIFT implementation [1]. Although it is not the same algorithm, it shares many similarities in how it finds unique, size invariant, points within a given image. In a research paper published by some of the creators of OpenCV, they make a particular note on its advantage in efficiency compared to SIFT. This is what makes it a good candidate.

The third and last part of the project was centered around combining the last two parts and wrapping them in an iPhone application. This will make the software more accessible to younger audiences, an audience that is eager to learn the games of chess in an interactive manner. The current implementation of the app is able to track a particular chess piece over a video and identify the type

of piece and provide information on the use of the object.

2 Requirements and Technology

2.1 Code

Code was developed and tested in Python. It is an easy to use language and simplified the workflow before porting it to C++ code. OpenCV library has implementations in both languages, which allowed its use within an iOS application. Combining our C++ code that utilizes the OpenCV library with Swift, we were be able to run our previously implemented algorithms alongside our chess piece classifier.

2.2 Libraries

The following libraries will be used to complete the project:

- **OpenCV**: A library of programming functions mainly aimed at real-time computer vision. Utilized in the project to generate unique key features in an image to track objects over a video.
- **Turicreate**: An easy-to-use and flexible machine learning tool kit that contains built-in visualization capabilities. It is also efficient and scalable. It is utilized to train a model to identify chess pieces in an image.

3 Data

The majority of the data used to train the classifier will be found through a combination of Google image searches and photos taken by the authors. We will compile a sufficient amount of images of each chess piece for the purposes of cursory training. For the sake of simplicity and the scope of this project, this data will be used to train a classifier.

A detector would be much more appropriate as it would provide a location of where in the image an object exists. Knowing that information would be useful for tracking, however doing so would introduce some complexities to our problem. For example, these images would need to be normalized and perhaps changed to grayscale, this would depend on preliminary results of accuracy and efficiency. Further, data preparation would take a considerable amount of time due to the fact that each image would have to have boxes manually drawn around the chess piece. This would be done to tell the detector where the object is in a specific image when it is being trained. There may exist an easier and faster way, but for now, we are certain that this very manual technique would require a lot of time and that is out of the scope of this project.

The images collected will be randomized and split into two sets, one for training and one for testing our model. Although we have already trained a model as of this point, it only incorporates 3 chess pieces and 10 images for each piece: rook, knight, and pawn. Below are images examples of each piece taken from our compiled dataset.

4 The Model and iOS App

The model was created using the Turi Create library, specifically an auto-configured convolutional neural network. The library was chosen because it generated the coreML model required to work with swift for the iOS app. We started with a training set of 20 images, 10 images of a knight, and 10 images of a pawn. This yielded an accurate that could classify images with a confidence 99% and a 100% classification rate. One noteworthy aspect is that as the number of classes is increased, the classification rate and the classification confidence tend to decrease. This mean that as we add more chess pieces to be classified, we need to increase the number of images for each chess piece in our training set.

Figure 1: Rook

Figure 2: Pawn

Figure 3: Knight

In Figure 4 we showcase a previous iteration of our app classifying a black knight and a white knight. The app is simple but it will serve a few purposes. The classification of a chess piece and the tracking of that chess piece throughout a video or live video stream. The current implementation performs all these steps but in a segregated manner.

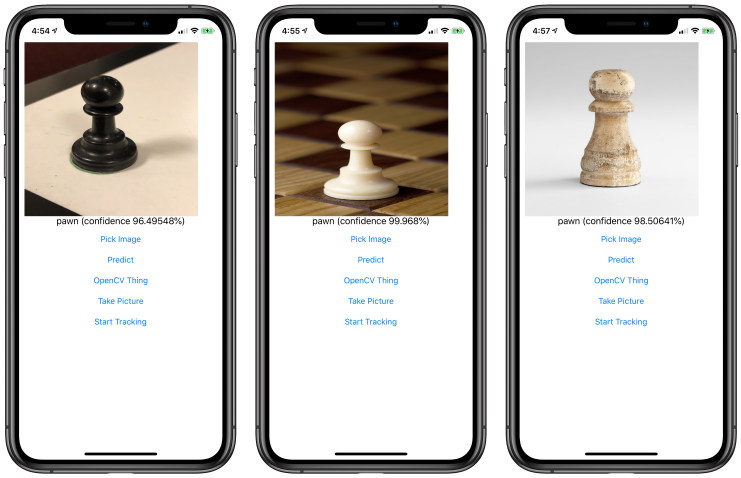

Figure 4: iOS Application showcasing pawn piece classification

One of the major hurdles encountered when integrating the object tracking algorithm into the iOS application was dealing with the intricacies of both Python and C++ as well as the context in which the OpenCV library was being used in the application. The application created was implemented in Swift, Apple's modern development language. However, Swift cannot directly communicate with C++ so an intermediary Objective C++ class had to facilitate the communication between the library and the application.

5 The Object Tracker

Creating the object tracking algorithm involved a significant portion of the research and implementation of the project. SIFT was the original candidate for the feature detection algorithm, but it was switched for its proprietary status, making it difficult to use, especially with OpenCV, which only include open source algorithms. ORB soon became the next candidate, boasting its open source status, and more efficient algorithm compared to SIFT. Two ways in which ORB is more efficient and becomes a better alternative is that the descriptors are less costly to generate, due to their binary nature [1]. This also means that it is also easier to compare two descriptors by computing the Hamming distance. This makes comparing features from frame to frame more efficient.

The tracking is algorithm compares two images, we can call them Frame_i and Frame_{i+1}. We assume that we already have a bounding box around our object that we want to track over a

video/stream. We define F_i's bounding box as $\text{Rec}_i(x_i, y_i, w, h)$ and the search box for F_{i+1} is defined as $S_{i+1}(u_{i+1}, v_{i+1}, w^{'}, h^{'})$, where

$$(u_{i+1}, v_{i+1}) = (x_i, y_i) + \vec{M}_i,$$
$$w^{'} = w + \alpha,$$
$$h^{'} = h + \alpha,$$

and $\vec{M}_i = (x_i, y_i) - (x_{i-1}, y_{i-1})$, and α is a constant value.

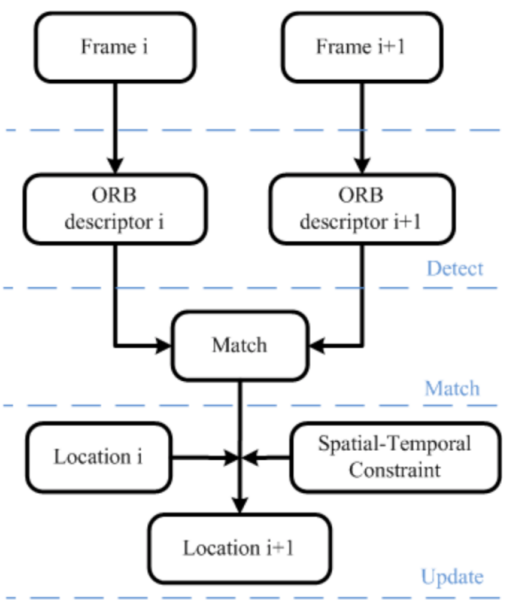

Figure 5: The framework of our tracking system includes three steps: detect, match and update [2].

ORB is ran within both bounding boxes on each frame to find all unique key features. Now, we compute the hamming distances between all descriptors to find matching key points. After all a select number of key features have been matched from one image to the next, the average distance between both images of a certain key feature is calculated. The center of the bounding box in F_i is displaced by that average displacement of key features to create a new bounding box that we define as $\text{Rec}_{i+1}(x_{i+1}, y_{i+1}, w, h)$. This process is outlined in Figure 5.

6 Testing

The data was split into two subsets: training and testing. As was mentioned before, the model was only trained to classify three chess pieces: rook, knight, and pawn. The training dataset consisted of 15 training images, and the testing had 10 images for each chess piece. The confusion matrix is presented in Table 1. One intriguing observation from the table is the gross misclassification of rooks. This indicates that the model has a hard time trying to distinguish a rook from other pieces and could require a larger training set. There are also exist limits to what the model is able to classify, take for instance Figure 6, we can identify the piece as a pawn, but the model fails to classify the horrific pawn. Other limitations may include the orientation of the piece, take for instance, an aerial view of a chess piece, most (even to humans) will look even more similar.

Figure 6: A pawn.

The object tracking algorithm was tested by running it against a video of a player moving a pawn along a chess board. Here, the tracker excels, as there exists a good contrast

$n = 10$	True	False
Pawn	9	1
Knight	10	0
Rook	3	7

Table 1: Confusion matrix generated from model.

between the object and the background. In the left image of Figure 7, the piece was picked up and displaced along the frame over the chess board while keeping the camera stationary. The path was drawn (in green) to illustrate the location of the bounding box over time. The picture on the right depicts the same tracker following a face as it bops and circles around the frame with similar accuracy. Translations along the z-axis have not been tested though, as it known that the ORB tracker does not handle scale invariance as well as SIFT does [1]. A future continuation of this project may include the classification and tracking completely integrated into the app. And Although it has been integrated into the app. there exist as separate features, as there was not enough time to fully get both working along side each other.

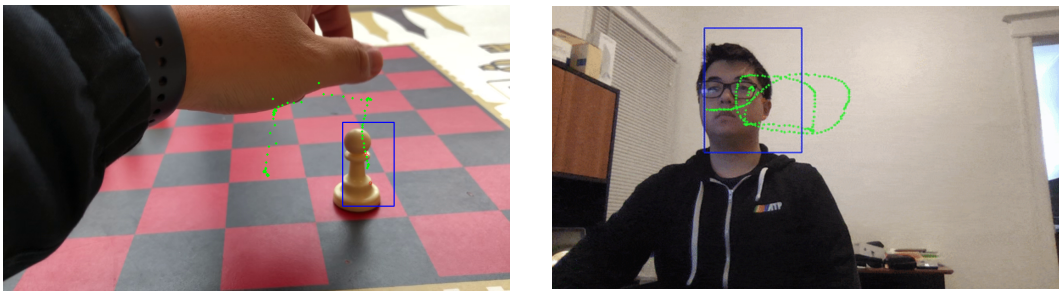

Figure 7: The path taken by the object (and bounding box) over time, drawn in green. A pawn was tracked in the left image, and Stephen's face was tracked on the right.

The iOS app was also tested by trying to access all these other previously tested features within the app, so minimal testing was required.

7 Conclusion

Although the full integration of all the features was not realized, all the features were implemented and tested within the app. Future work may involve the full integration of all the algorithms and a more robust design of the app. The current implementation did accomplish the goals of the project, and the biggest achievement was the creation of an object tracker using the SIFT alternative, ORB. Some difficulties were encountered when attempting to integrate the application into a mobile context, but in an environment where mobile applications are becoming more and more popular, it was important to explore this vector of deployment. The benefit of implementing this object tracker in OpenCV is that it could be ported to many platforms, which could include iOS *and* Android. This could prove to be convenient in possible commercial tools that create an application using this algorithm. It is, however, important to note that both iOS and Android have native support for computer vision processing (Vision Framework in iOS and Mobile Vision in Android).

Possible future work would involve expanding the set of identifiable pieces to include all chess pieces, as well as transitioning from a classifier to a detector. A detector extends on the implementation of a classifier by determining where in a given image an object is detected. This would obviate the task of prompting a user for the bounding box as the detector would already have an initial bounding box drawn.

References

[1] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. pages 2564–2571, Nov 2011.

[2] S. Wu, Y. Fan, S. Zheng, and H. Yang. Object tracking based on orb and temporal-spacial constraint. pages 597–600, Oct 2012.