Phone Tag: Laser Tag for Mobile Devices

Collin Lee  
Department of Electrical Engineering  
Stanford University  
Email: cstlee@stanford.edu

Mark Stauber  
Salisbury Robotics Lab  
Stanford University  
Email: stauber@stanford.edu

Jeff Stone  
Department of Electrical Engineering  
Stanford University  
Email: stonej@stanford.edu

Abstract—Mobile games now represent the largest sector of the game market when measured by revenue [1]. Ubiquitous smartphones with their wide range of integrated peripherals present designers with opportunities to use the devices in novel ways. Our goal was to build an Android app that turns a smartphone into the equipment to play a game similar to “laser tag”. A two-player game has been implemented as a demonstration. Android based mobile devices were selected due to the availability of advanced displays, cameras, network hardware and the accessibility of the software development environment. Algorithm selection was driven primarily by performance considerations. A fluid user experience was achieved without noticeable latency in game play or player feedback. Our work demonstrates that novel combinations of mobile device technology, computer vision, image processing, graphics, networking and myriad other capabilities can be realized while maintaining good performance in an application intended for entertainment.

Keywords—laser tag; game; marker detection; object detection; OpenCV; Android; WifiP2P

I. INTRODUCTION

Laser tag is a fun game where players attempt to tag opponent players. Playing a game of laser tag requires specialized equipment. Typically, devices emitting infrared pulses are aimed toward infrared sensitive targets worn by the players. The goal of Phone Tag is to removed these constraints and allow “laser tag” to be played anytime, anywhere. We have accomplished this by replacing the highly specialized laser tag equipment with mobile devices and software.

The basic game of laser tag can be broken down into two functional requirements:

1) System must detect a target hit.
2) System must relay information about the hit to target.

Before actual game play, each player records a feature rich section of his opponent’s clothing such as a printed shirt or a logo. We have cast the hit detection requirement into a natural feature detection problem that must be performed by the shooter. ORB (Oriented FAST, Rotation Aware BRIEF) natural feature detection can be performed on the mobile platform at practical rates for fluid game play. Feature detection is described in section II. The information about a detected hit is relayed through a networking component to other players; Wi-Fi Direct peer-to-peer (P2P) [2] was selected for its range and low latency. Networking is discussed in section III. Development and integration of the core requirements into a mobile Android [3] solution also presented challenges; this is discussed in section IV.

II. TARGET DETECTION

The goal of target detection is to determine whether a target exists in a region of interest (ROI). A stored reference image represents a valid target. Target detection provides the basic primitive on top of which a boolean “hit” query can be implemented. In our game, the ROI represents a “sight”, a small rectangular region in the center of the viewfinder image. Using the gunsight analogy in our game mechanic creates a requirement for the player to point the device accurately. In fact, the size of the sight can be adjusted to make the game easier or more difficult.

In the final version of the game, the players are free to use any reference image as a target description. In the first phase of development, however, we focus on the problem of target detection using markers like those shown in figure 1.

Fig. 1. Typical fiducial markers.

The existing image processing literature presents us with multiple options for marker detection. Our implementation of marker detection utilizes the object detection pipeline shown in figure 2. The use of object detection for marker detection is somewhat atypical; fiducial markers like those shown in figure 1 are typically detected using corner, edge, or contour detection. This contrasts with natural feature detection which is used as the basis for object detection.

Fig. 2. Detection Pipeline

The decision to use an object detection based pipeline is motivated by our desire to use any reference image as a target. Our end goal is to be able to use the existing features of a player’s clothing as a target marker. At the beginning of the game, each player stores an image of a distinguishing portion of each of their opponent’s attire. Using this same detection pipeline we can therefore provide target detection.
The target detection pipeline is implemented in native C++ using the OpenCV libraries. We created a standalone model for the image processing portion of the game implementation in standard C++, which could be prototyped rapidly on a workstation without the mobile application coding overhead. The C++ native detection code was then connected to the Android application using a JNI wrapper which will be described in section IV. The target detection implementation is based on the sample OpenCV code found in the OpenCV 2.4.9 source code [4]. Using the full-featured OpenCV library allowed us to simplify development of the algorithm since many of the core components are modules available in OpenCV.

A. Feature Detection and Descriptor Extraction

The first operation performed on both the reference image and the ROI image is feature detection. The purpose of feature detection is to extract scale and rotation tolerant features that can be used to match the reference image with the ROI image. The second step is to extract descriptors from the detected features to be used in the descriptor matching step. Originally, a SURF [5] detector/extractor was used in the pipeline. During our preliminary experiments, however, we found SURF to be far too slow, requiring on the order of seconds to complete the full pipeline on a given frame. Additionally, SURF is not available in the free OpenCV library that can be used in Android apps. Next we experimented with the ORB detector/extractor [6]. ORB proved to be 10 to 100 times faster than SURF for our pipeline.

B. Descriptor Matching

Once descriptors have been extracted from both the reference image and the ROI image, they need to be compared to find only the matching descriptors. Since the number of features is small, a brute force matching implementation is used.

C. Homography Estimation and Homography-based Descriptor Filtering

We use the assumption that our target is roughly planar in order to further filter out false-positive matches. The assumption is that descriptors that match based on their features should map geometrically given an some homographic transformation. We first estimate a homography using the findHomography module provided in OpenCV. We then use this result to throw out descriptor matches that do not match geometrically. This provides us with the final list of feature matches.

D. Detection Threshold

The final step is to report whether a target is or is not detected in the ROI. A simple thresholding mechanism is used to accomplish this. The algorithm reports a detection if the number of final matches found is greater than 10% of the number of features detected in the references image. Said another way, the algorithm reports a detection if it finds more than 10% of the references images features in the ROI image.

The 10% constant was set empirically; we experimented with a range of values and found 10% to work quite well for a wide range of realistic targets.

III. NETWORKING

The game requires a network service for player feedback and to synchronize scorekeeping among the players. Since the game presented here is 2-player only, the requirement is simply that the sum of each player’s remaining lives and the opponent’s score is equal to the starting number of lives. We researched various options for communication between the mobile devices and settled on Wi-Fi peer-to-peer networking. The following excerpt is from the Android Developers API Guide for Wi-Fi peer-to-peer [2]:

Wi-Fi peer-to-peer (P2P) allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point (Android’s Wi-Fi P2P framework complies with the Wi-Fi Alliance’s Wi-Fi Direct certification program). Using these APIs, you can discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection. This is useful for applications that share data among users, such as a multiplayer game or a photo sharing application.

The Android Wi-Fi P2P approach does not require a router or Internet connection and has the added benefit of being potentially compatible with non-Android devices. Establishing and using a connection requires the following steps:

1) Discover: Determine device addresses of peers.
2) Group Negotiation: A WiFi Direct protocol function that chooses a single group owner from among the peers.
3) Connect: Open a socket.
4) Send/Receive Packets: Utilizes UDP (User Datagram Protocol)

Wi-Fi Direct connection is intended to transfer files between devices. This functionality is not needed and is not included in our code. The code that we adapted from the WiFi Direct demo handles the network communication setup and the group negotiation...
within the scope of Android fragments, which also correspond to panels of the user interface. We added a “Poke” button to the Network Setup activity. The purpose is to test that the devices are properly configured and connected and can send and receive datagrams before game play begins.

The WiFi capability is initiated on the Network Setup screen and continues functioning under the supervision of the Play Game activity. To comply with Android application structuring requirements, game communications are handled by an Android “service” which is a separate entity from any particular activity. The Network Setup activity and the Play Game activity each bind themselves to the game communication service while they are in effect, and unbind when the player switches to the next activity.

The group owner which was selected during group negotiation is considered to be the server, and the other device is the client. If the game were expanded to three or more players, there would still be a single server, but it would serve multiple clients. For the two player game presented here, the protocol used is extremely simple. The client polls the server by sending it a string representing the number of lives that the server game has left (which is calculated from the client game’s score). The server responds with a string telling the client how many lives it has left.

These strings are wrapped in UDP datagram packets and are sent at an interval of 200 ms. The UDP/IP protocol guarantees that all packets received are correct, but it does not give any indication when packets are dropped. This lightweight protocol works well in this particular application because polling is repeated approximately five times per second, so any dropped packets will be repeated in a reasonable amount of time to give the opponent feedback. Note that from the player’s perspective, there is no distinction between server and client.

The only status information needed by this simple protocol is the device address of the opponent and a flag indicating whether the device is the owner (server). The game communication service on the server spawns a single thread on the server and two threads on the client. The server thread receives datagrams from the client game containing lives remaining information, then sends out the same for the server game. On the client, two threads are spawned, one to initiate the polls at an interval of 200 ms, and the other to receive the responses.

The lives remaining information is useful not only for scoring, but for other types of user feedback. A fully realized game would alert the user to having been hit with audible, visual and haptic indications.

With a few changes, the protocol could be extended to a multiplayer game. Care would need to be taken to avoid scoring synchronization problems among the distributed set of devices. Information from one client to another would need to be routed through the server device.

IV. ANDROID DEVELOPMENT

The project aimed to create a fast mobile gaming experience and as a result a lot of effort went into structuring the complex codebase. The final PhoneTag code was divided into 3 main categories:

1) Game User Interface
2) Image Processing (including Java interface)
3) Networking

A. Game User Interface

The Game UI for Phone Tag is contained in 2 activity classes. A StartActivity class is the default view of the app that allows the player to start a game, learn the rules, or adjust settings such as difficulty. A PTActivity contains the gameplay interaction. All original game graphics were designed using the open source vector graphics editing software Inkscape and the raster graphics editing program Gimp.

B. Image Processing

In order to create an smooth laser-tag experience without requiring an internet connection, the necessary image processing had to run on the players’ mobile devices. This required running OpenCV algorithms in native C++ as part of the Android build. To accelerate the project development, the code base was modeled after the OpenCV sample face detection project.

The image processing begins at the start of the game when players snap an image of their opponent’s shirt. The image
is passed to the constructor of a C++ feature detection object that will conduct the hit or miss detection throughout the game. The object extracts a set of features and descriptors from the opponents shirt image which are stored as a reference for all incoming image shots.

When a user “shoots” the ROI centered on the crosshairs is grabbed in PTActivity and passed via the java interface located in the DetectionBasedTracker class to the native C++ object. The C++ object applies the detection measure as describe in section II. The interface between the java and C++ code is simplified so that it returns a simple binary representing a hit or miss to shot queries.

C. Networking

The networking subsystem is composed of the following Java modules:

- WiFiDirectActivity: Network Setup UI.
- DeviceListFragment: Device List UI elements.
- DeviceDetailFragment: Device Detail UI elements.
- WiFiDirectBroadcastReceiver: Informs WiFiDirectActivity of WiFi events.
- PTCommService: Application level service bound to WiFiDirect activity during network setup and bound to PTActivity during game play.

The first four modules provide discovery, group negotiation, and a UI for network setup. PTCommService runs independently of any specific activity, but can be bound dynamically to activities that require the service. It is used by the network setup activity (WiFiDirectActivity) for the user to test the connection before starting game play. It is used by the play game activity (PTActivity) for real time networking during the game.

![WiFi Setup Screen](image)

**Fig. 6. WiFi Setup Screen (WiFiDirectActivity)**

V. CONCLUSION

The goals of the project were achieved with room to spare. The Phone Tag app is an entirely mobile, and equipment free way to play laser tag with a friend. The gameplay is fast due to the native image processing and direct peer-to-peer network. Our optimized feature recognition algorithm has proven to be accurate for a range of over 30 feet and robust to rotation, scale, and lighting changes. This distance is more than enough to allow for challenging gameplay.

Although many laser tag games involve teams of several players, the one-on-one version created here affords much of the same experience. Since we have excess bandwidth in the image pipeline, we could extend the game to three or more players at the cost of matching against multiple targets. For each device, the complexity of doing so scales only linearly. The network protocol could be extended to communicate scoring client-to-client, with the server (group owner) as an intermediary. Looking forward, these improvements will be incorporated into future versions of the game in preparation for submission to the Google Play Store.

ACKNOWLEDGMENT

We would like to thank Jean-Baptiste Boin for his mentorship. Thank you to Professor Bernd Girod for the educational and enlightening course, as well as the helpful course assistants, David Chen and Huizhong Chen, and the rest of the teaching staff. A special thanks to the rest of our classmates who’s involvement on the class forum was instructional throughout the quarter.

REFERENCES


APPENDIX A

CONTRIBUTIONS

The Phone Tag game concept emerged through group brainstorming. Our project mentor, J.B. Boin guided the group toward the natural feature detection approach. All design decisions, software architecture, networking, UI, algorithm selection and evaluation, coding and testing were group efforts. In that context, the following areas of concentration were claimed by individual group members to allow periods of uninterrupted individual effort.

Collin Lee:
- Target detection algorithm investigation, modeling and development.
- C++ native detection pipeline implementation.

Mark Stauber:
- Detection algorithm development.
- Detection API/Infrastructure, JNI (Java Native Interface).

Jeff Stone:
- Top-level Android application skeleton.
- WiFi Direct P2P service implementation and inter-device protocol design.