SIPL Streaming App

גרסא 1.0

יוני 2019

הקדמה

מה האפליקציה מאפשרת לעשות?

1. הזרמת וידאו גולמי ממצלמה של מכשיר Android לשרת.
2. עיבוד הווידאו על ידי אלגוריתם ב-C++ או ב-Python.
3. שליחת טקסט (תוצאות העיבוד) לטלפון.

למה נועד המדריך?

המדריך יעקוב אחר הוראות ההתקנה של הפרויקט לדוגמא Darknet (שנמצא גם על עמדת דמו ייעודית במעבדה) וכן ידריך כיצד ניתן להסב אותו לשימושכם. בנוסף, קיים פרויקט מצומצם בתיקייה נפרדת ב-git שבו נמצא הפרויקט והוא משמש רק לחילוץ הפריימים מתוך Stream מהמכשיר (להציג אותם/לשמור אותם), בלי לעבד אותם, ויש לו README משלו.

הערה

מאחר ש-Python יותר איטי מ-C++, המימוש של הפונקציה המוציאה את הפריימים מהשרת עובדת בצורה הרבה פחות טובה ב-Python. המשמעות היא שבייחוד אם עובדים ב-fps גבוה הרבה מידע נאבד בדרך, ושיתכן שיהיה צריך לנסות ליצור התקשרות עם השרת מספר פעמים לפני שזה יצליח.
לכן, מדריך זה ידגים כיצד ניתן לסנכרן את האפליקציה עם פרויקט ב-C++ בלבד. עם זאת, באותו Github שבו נמצאים הקבצים הדרושים ל-C++ נמצאים גם הקבצים המיועדים ל-Python וניתן להוריד אותם ולהשתמש בהם בצורה דומה. הם לא מיועדים להרצה של הפרויקט לדוגמא Darknet, אלא רק לקריאה של הפריימים מהמכשיר בשרת.

התקנות

סביבת עבודה

אין מגבלה על מערכת ההפעלה עליה מותקן צד השרת, כלומר, כל מחשב שניתן להריץ בו את הפרויקט הרצוי יעשה את העבודה (תוך הנחה שהשתמשתם בפרויקט ב-OpenCV, אם לא אז זה הזמן להתקין).
לתיקייה הראשית של הפרויקט בה נעבוד נקרא בהמשך (sitory)**repo**, ואנו מניחים כאן שהיא מסודרת לפי תיקיות build, lib/src וקובץ Makefile.

קבצים

היכנסו ל-Github הבא:

https://github.com/ayeletc/serverSide

הורידו לתיקייה של הפרויקט את הקבצים הרלוונטיים:

- מהדף הראשי: rtmpServerTest.js => לתוך repo
- מתוך CPPprocess:
  - geturl.cpp, geturl.h => לתוך תיקיית ה-src
  - server.js => לתוך ה-repo

תיקונים והערות:
איילת, ayelet@live.com

- מתוך ה-Makefile => העתיקו ל-Makefile שלכם את כל מה שנוגע לטיפול ב-geturl.

```
main.o: src/main.cpp src/VideoProcess.h src/darknet.h src/geturl.h
        $(CC) -std=c++11 -c src/main.cpp

VideoProcess.o: src/VideoProcess.cpp src/VideoProcess.h
        $(CC) -std=c++11 -c src/VideoProcess.cpp

geturl.o: src/geturl.cpp src/geturl.h
        $(CC) -std=c++11 -c src/geturl.cpp

$(EXEC): $(OBJS) geturl.o VideoProcess.o main.o
        $(CC) $(COMMON) $(CFLAGS) $^ -o $@ $(LDFLAGS)
```

- היעזרו ב-main של darknet (שנמצא ב-serverSide/darknetFiles/main.cpp) והוסיפו לתוך ה-main.cpp של הפרויקט שלכם את החלק הדרוש לקריאת הפריימים מהשרת:

```
#include "geturl.h"
#include "VideoProcess.h"

[…]

int main(int argc, char **argv) {
        std::string videoAddress;
        getVideoAddress(&videoAddress); //get the server URL
        std::cout <<  "Video Address: " + videoAddress << endl;
    try {
        // 1: create VideoProcess Instance with the location of the video.
//      auto vp = VideoProcess(string(argv[1])); // in case you want to
use local video using its path (instead of reading it from the server)
        auto vp = VideoProcess(videoAddress);
 //       vp.play(); // just play the video if you want
        // 2: call the process method and
        // pass to it the name of function that going to handle each frame
        vp.process(process_frame);
    }catch(...){
        std::cout << "video process failed" << endl;
    }
    return 0;
}
```

אם השתמשתם ב-VideoProcess כדי לחלץ פריימים מהווידאו, ב-imread או בפונקציה אחרת אז תצטרכו לעשות את ההתאמות כך שתחלצו את הפריימים מהשרת עם VideoProcess לפי ה-URL של השרת. הפונקציה מחזירה פריימים מטיפוס **Mat**.

תיקונים והערות:
איילת, ayelet@live.com

התקנת צד שרת

<u>על Ubuntu</u>

1. פתחו טרמינל בתיקייה usr/lib והריצו:

- sudo apt install nodejs
- sudo apt install npm

הפקודה הראשונה מתקינה NodeJs והפקודה השנייה מתקינה את מנהל החבילות שלו.

- בדקו שה-NodeJs מותקן על ידי הרצת "node -v" בטרמינל (וכך גם תגלו איזו גרסא מותקנת).

2. עברו ל-repo והריצו שם את הפקודות הבאות על מנת להתקין את החבילות הדרושות לשרת:

- npm install http
- npm install tree-kill
- npm install rtmp-server
- npm install child_process
- npm install socket.io

אם הכל עובר בשלום והטרמינל לא מתלונן לאחר כל התקנה, הייתה אמורה להיפתח בתוך repo תיקייה בשם node_modules שבה נמצאות הספריות שהורדתם ועוד כמה אחרות.

3. על מנת לבדוק שההתקנה הצליחה ננסה לפתוח שרת על המחשב:
   - הריצו בתוך הטרמינל:

- node rtmpServerTest.js

   - פתחו את הדפדפן בכתובת:

- localhost: 1936

   - חכו כמה שניות ואז סגרו את הדף.
   - במצב תקין הטרמינל היה צריך לכתוב לכם:

client disconnected

   - סגרו את השרת על ידי ctrl+c.

תיקונים והערות:
איילת, ayelet@live.com

על Windows

1. הורידו את NodeJs ואת מנהל החבילות שלו npm מהאתר הרשמי בעזרת ה-Installer.

- בדקו שה-NodeJs מותקן על ידי הרצת "node -v" בטרמינל (וכך גם תגלו איזו גרסא מותקנת).

2. פתחו cmd ב-repo והריצו שם את הפקודות הבאות על מנת להתקין את החבילות הדרושות לשרת:

- npm install http
- npm install tree-kill
- npm install rtmp-server
- npm install child_process
- npm install socket.io

אם הכל עובר בשלום והטרמינל לא מתלונן לאחר כל התקנה, הייתה אמורה להיפתח בתוך repo תיקייה בשם node_modules שבה הספריות שהורדנו ועוד רבות אחרות.

3. על מנת לבדוק שההתקנה הצליחה ננסה לפתוח שרת על המחשב:

- הריצו בתוך ה-cmd:

  - node rtmpServerTest.js

- פתחו את הדפדפן בכתובת:

  - localhost: 1936

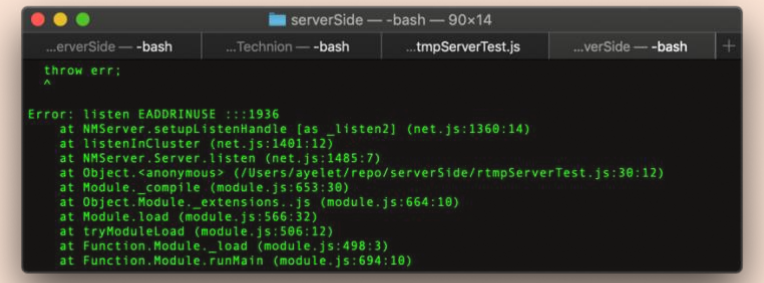
- חכו כמה שניות ואז סגרו את הדף.
- במצב תקין ה-cmd היה צריך לכתוב לכם:

  client disconnected

- סגרו את השרת על ידי ctrl+c.

תיקונים והערות:
איילת, ayelet@live.com

תקלות נפוצות

שלב	תקלה	פתרון
1	ההתקנה נכשלה (הותקנו 0 קבצים)	אם אתם עובדים על Ubutntu 16.04 נסו לכתוב את הפקודות עם "apt-get" (במקום "apt")
	הפקודה node -v לא זוהתה	נסו להריץ "node --version"
2	לא נפתחו תיקיות / הותקנו 0 קבצים	ב-Ubuntu : נסו להריץ את הפקודות עם sudo
3	חבילה ספציפית לא הותקנה	בדקו אם קיימת תיקייה בתוך node_modules בשם של החבילה שניסיתם להוריד. - אם לא, נסו להוריד שוב את החבילה (להריץ שוב את הפקודה). - אם כן, אז בדקו איזו גרסא של NodeJs התקנתם על ידי הרצה בטרמינל של : "node -v", בדקו באינטרנט מה הגרסא העדכנית של NodeJs. אם הגרסא שלכם מאוד ישנה, נסו לשדרג אותה ואז להתקין שוב את החבילה/חבילות שלא הותקנו.
	עדכון גרסא של node לא צלח	נסו לעדכן בעזרת מנהל חבילות אחר - nvm
6	ה-port הרצוי כבר נמצא בשימוש	אם הטרמינל מציג שגיאה כזאת :  המשמעות היא שה-port שעליו אנחנו מנסים לפתוח את השרת כבר תפוס. רוב הסיכויים שניסיתם להריץ את הקובץ במקביל פעמיים. נסו לסגור את הטרמינלים/cmd הפתוחים ואז להריץ מחדש. אם זה לא עובד נסו לשנות את מספר ה-port מ-1936, למשל ל-1946 (ואז תצטרכו לשנות אותו גם בקובץ הכולל של השרת server.js).

תיקונים והערות :
איילת, ayelet@live.com

הורדת האפליקציה

הפרויקט של האפליקציה נמצא ב-GitHub:

https://github.com/ayeletc/sipl-streaming-app

ניתן להוריד את כל הקבצים ולפתוח דרכם את הפרויקט ב-Android Studio, אבל בייחוד אם תרצו לעשות שינויים באפליקציה מומלץ לפתוח משתמש ב-GitHub, ולהוריד את הפרויקט בעזרת fork ו-clone ישירות ל-Android Studio. תוכלו להיעזר במדריך:

https://www.londonappdeveloper.com/how-to-clone-a-github-project-on-android-studio/

טסט של השרת עם Darknet

1. פתחו טרמינל ב-repo/build
2. הריצו node server.js
3. הדליקו את המכשיר ופתחו את האפליקציה SIPL Streaming App
4. הכניסו את ה-ip של המחשב לתוך: rtmp://xxx.xxx.xxx.xxx:1936/live/stream
   - אם לא מחליפים Router אז הכתובת הנכונה נמצאת בתור ברירת מחדל
   - אם כן מחליפים אז צריך לבדוק מה הכתובת יש כתובת פנימית וכתובת חיצונית, אז אם אחת לא עובדת צריך לנסות את השנייה.
   - אם יש בעיות עם ההתחברות לשרת כדאי לבדוק לאיזה WiFi מחובר המכשיר. אם לא מחברים Router ייעודי אז כדאי להשתמש ב-efsecure.
5. לחצו על Start Stream
6. לחצו על START
7. התוכנה אמורה לרוץ עם פלט ראשוני:

```
Server: connect live
Server: PLAY stream
```

   - אם יש בעיות והשרת כותב בטרמינל שהוא לא מקבל פריימים אז כדאי ללחוץ על STOP, לחכות כמה שניות ולנסות שוב. אם גם זה לא עובד לנסות לאתחל את השרת ושוב לחכות כמה שניות בין פתיחת ה-server (node server.js) לבין ה-START של האפליקציה במכשיר
   - מומלץ לצלם עם הטלפון אנכי ולא אופקי
   - יש סיכוי שהזיהוי יעבוד טוב רק במיתוג הראשון של הרמזור
8. בסיום השימוש כדאי להרוג את השרת עם ctrl+c בטרמינל, והפלט שיופיע:

```
Server: client disconnected
Server: kill cpp process
```

תקלות נפוצות

אם הורדתם את האפליקציה למכשיר סביר להניח שתצטרכו לשנות כל מיני הגדרות במכשיר (developer mode, permissions ועוד), אבל למזלכם יש די והותר מידע בגוגל בנושאים האלה.

תיקונים והערות:
איילת, ayelet@live.com

התאמת הכלי לאלגוריתם

השרת שבו אנו משתמש על מנת להזרים את הווידאו מהמצלמה מבוסס על RTMP – Real Time Messaging Protocol ונכתב בג'אווה סקריפט, ועל מנת להחזיר תוצאות למכשיר אנו משתמשים בספרייה Socket.io שמאפשרת התקשרות בין שרת למשתמש וזמינה בפלטפורמות רבות, ביניהן NodeJS Java ו-Python.

בשביל התאמות מינימליות לא נדרשת הבנה ברשתות ובפרוטוקולי תקשורת. באופן תיאורטי, גם אם רוצים להחזיר תמונה למכשיר יהיה אפשר לעשות את זה על ידי המרה ל-string, וגם בשביל להעביר תמונה לשרת אפשר לשלוח stream וממנו להוציא פריים על השרת עצמו. אם תרצו לעשות דברים מסובכים יותר או לקצר את זמני הריצה, כנראה שתצטרכו למצוא פרוטוקול תקשורת חלופי.

Events

הסנכרון בין השרת והאפליקציה נעשה באמצעות אירועים (Events). נסביר כאן כיצד עובד האירוע של התחברות לשרת כך שתוכלו להגדיר בצורה דומה Events נוספים.

ה-Emitter שמכניס את האירוע לפעולה הוא התחברות של משתמש לשרת. גם ניסיון לגשת ל-url של השרת דרך הדפדפן יפעיל את האירוע, אבל במקרה שלנו הכוונה היא להתחברות דרך האפליקציה על-ידי לחיצה על הכפתור שמתחיל את הצילום. ככה זה נראה בקוד:

- <u>צד שרת</u> (server.js):

השרת שמעביר את הווידאו נקרא "rtmpServer". אחד מהאירועים הראשיים שלו הוא "client" שמקבל כפרמטר את המשתמש client. למשתמש client יש אירוע משני שנקרא "connect" שלא מקבלים פרמטרים. בתוך "connect" יש לנו הדפסה ל-log ובנוסף, Emitter לאירוע של Socket.io ששמו "clientConnected" והוא מקבל פרמטרים בצורת json – הערך הוא הטקסט: "I got a client" והשם שלו הוא "data".

```
const RtmpServer = require('rtmp-server');
const rtmpServer = new RtmpServer();
[...]
const http = require('http').createServer();
const io = require('socket.io')(http);
[...]
rtmpServer.on('client', client => {

        client.on('connect', () => {
        console.log(S_LOG + 'connect', client.app);
        io.emit('clientConnected', {"data" : "I got a client"});
        });
         [...]

});
```

תיקונים והערות:
איילת, ayelet@live.com

- <u>באפליקציה:</u>

בתוך הקובץ:

sipl-streaming-app/qiscus-streaming/src/main/java/com/qiscus/streaming/ui/activity/QiscusVideoStreamActivity.java

אנחנו יוצרים טיפוס של Socket.io לקבלת התוצאות. הגדרנו לשרת אירוע עם אותו השם שהגדרנו בשרת: "clientConnected". כשהאירוע מופעל הוא נכנס לפונקציה "onClientConnected".

```
import io.socket.client.IO;
import io.socket.client.Socket;
import io.socket.emitter.Emitter;
[...]
public class QiscusVideoStreamActivity extends AppCompatActivity implements ConnectCheckerRtmp, SurfaceHolder.Callback {
        [...]
        private Socket mSocket;
        public void onCreate(Bundle savedInstanceState) {
        [...]
                String HttpSocketUrl = getHttpUrl(streamUrl);
                try {
                        mSocket = IO.socket(HttpSocketUrl);
                        mSocket.on("clientConnected", onClientConnected);
                        [...]
                } catch (URISyntaxException e) {}
        }
        [...]
```

תיקונים והערות:
איילת, ayelet@live.com

הפונקציה מוציאה הדפסה ל-log ולאחר מכן מחלצת את ה-string מתוך ה-json ששמו "data" (כפי שהגדרנו בשרת). התוכן של message הוא: "I got a client".

```
private Emitter.Listener onClientConnected = new Emitter.Listener() {
        @Override
        public void call(final Object... args) {
                Log.i("Ayelet", "inside onClientConnected");
                QiscusVideoStreamActivity.this.runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                                JSONObject messageObj = (JSONObject) args[0];
                                String message;
                                try {
                                        message = messageObj.getString("data");
                                        Log.i("Ayelet", "onClientConnected got data from the server: " + message);
                                } catch (JSONException e) {
                                        return;
                                }
                        }
                });
        }
};

[...]
}
```

לסיכום, כשמגדירים Event חייבים לדאוג לתאימות בין צד השרת לאפליקציה בשם של ה-event (clientConnected) ובשם של ה-json שמועבר (data).

<u>באלגוריתם</u>

על-מנת שהפלט של האלגוריתם יגיע לשרת, צריך לרוקן את ה-std. ב-C++ אפשר לעשות את זה באמצעות **endl** (במקום \n). אם זה לא עובד או שאתם עובדים עם שפה אחרת, חפשו בגוגל איך עושים flush.

תיקונים והערות:
איילת, ayelet@live.com

Text To Speech – TTS

בתור ברירת מחדל, האפליקציה אומרת בקול את המידע שמגיע מהשרת ב-clientConnected (וב-Events נוספים – "red light" אם לא זוהה רמזור ירוק ו-"green light" לאחר זיהוי אור ירוק). באופן דומה תוכלו להגדיר עוד Events שבהם המכשיר יגיב על-ידי שימוש בפונקציה:

```
speakWords(message);
```

שמקבלת את ההודעה מטיפוס String.

על מנת להשבית את האפשרות של הדיבור:

- פתחו את הקובץ:

sipl-streaming-app/qiscus-streaming/src/main/java/com/qiscus/streaming/ui/activity/QiscusVideoStreamActivity.java

- הגדירו את המשתנה absEnableTTS ל-false.

```
private TextToSpeech myTTS;
private int MY_DATA_CHECK_CODE = 0;
private boolean absEnableTTS = true; //enable the voice notification absolutely
private boolean enableTTS = false; //disable the voice notification in case
```

עבור פרויקט ב-Python:

אל תשכחו לכתוב את הלינק ל-Interpreter של ה-Python במקום המתאים – בקובץ "rtmpServerPythonProc.js" בקריאה ל-spawn (פעמיים בקובץ):

```
var process = spawn('/Users/ayelet/.virtualenvs/cv/bin/python', [] );
```

בנוסף, ה-sample של האפליקציה והשרת המתאימים נמצא בקובץ processRTMP.py, תוכלו לראות בקובץ הזה שבשביל שהשרת יעבוד צריך לייבא את החבילות Socket, cv2, וכיצד מתבצעת הקריאה של הפריימים לתוך המשתנה frame (גם כאן באמצעות VideoCapture של OpenCV).

תיקונים והערות:
איילת, ayelet@live.com