ישנם 2 סרברים. סרבר אחד שירוץ על המחשב (נקרא סרבר לוקאלי) של השרת וסרבר שירוץ על הצוללת.
מטרתו של הסרבר שירוץ על המחשב של השרת היא לאפשר הרצת פילטרים על סרטונים ותמונות.
מטרתו של הסרבר שירוץ על הצוללת היא לאפשר הרצת אלגוריתמים בזמן אמת.

שני הסרברים מציעים את אותם הפיצ'רים.

מדריך למתחזק

סרבר לוקאלי

בסרבר הלוקאלי ישנן חמש מחלקות:

Log: בכל פעולה שהקליינט מבצע, הסרבר רושם לקובץ "log.txt" וגם ל-stdout מה מצבו. למשל, כאשר נשלח לסרבר קובץ כלשהו, הוא ידפיס את כמות ה-bytes שקיבל, את שם הקובץ שקיבל והאם הפעולה שהקליינט רצה לעשות עם הקובץ התבצעה בהצלחה.

VideoStream: מחלקה זו לוקחת תמונות מסרטון כלשהו (או תמונה אחת שנשלחה ע"י הקליינט), מריצה עליהם את הפילטרים שהקליינט רוצה (באמצעות מחלקה FilterRun), שומרת את התמונות לקובץ וידאו כלשהו אם הקליינט ביקש, ואז שולחת אותם אל הקליינט. במחלקה ישנם 2 ת'רדים שפועלים במקביל:

- _clientThread
- _streamThread

כאשר הראשון מחכה לקליינט שיתחבר אליו, והשני שולח תמונות לקליינט שהתחבר אליו. כרגע, המימוש שלי לא מאפשר שליחת תמונות ליותר מקליינט אחד.

כל הפונקציות במחלקה הזו (ובכל המחלקות האחרות) מתועדות חזק. אין סיבה שלא תבינו מה הולך שם אחרי קריאת הקוד וקריאת התיעוד.

FilterHandler: מחלקה זו מטפלת ביצירת ותחזוקת הפילטרים. ישנם 3 סוגי פילטרים: Built-in, SO, Created filters.

Built-in הם פילטרים המקומפלים במערכת. SO הם פילטרים אשר נשלחו ע"י הקליינט (הקליינט שלח קובץ something.so וגם something.config) ו-Created הם פילטרים אשר נוצרו ע"י הקליינט מהפילטרים הקיימים במערכת - לדוגמא, אם בסרבר ישנם 10 פילטרים, והקליינט רוצה ליצור משלושה פילטר חדש, אז הסרבר ישמור את השמות של הפילטרים ולא ניצור אובייקט חדש.

ביצירה של האובייקט FilterHandler, המערכת טוענת את כל הפילטרים למערכת ושומרת אותם בשלושה וקטורים מתאימים:

```
map<std::string, BaseAlgorithm*> _filtersInMachine;

map<std::string, BaseAlgorithm*> _SOFilters;

map<std::string, CreatedFilter*> _createdFilters;
```

כאשר המשתמש שולח קובץ .so, המערכת שומרת את הקובץ במקום המתאים (כולל את הקובץ config המתאים), טוענת מחדש את כל הפילטרים .so ומוסיפה אותם לוקטור

```
map<std::string, BaseAlgorithm*> _SOFilters;
```

אותו הדבר כאשר המשתמש יוצר Created filter. המערכת יוצרת פילטר חדש וטוענת מחדש את כל ה-created filters לוקטור המתאים.

הכוונה בטוענת מחדש זה – מחיקת האובייקטים שקיימים ויצירת חדשים.

טעינת ה-SO filters מתבצעת ע"י הפונקציה:

```
void loadSOFilters();
```

טעינת ה-Created filters מתבצעת ע"י הפונקציה:

```
bool loadCreatedFilters();
```

FilterRun: מחלקה האחראית על הרצת הפילטרים.

למערכת ישנם 2 סוגי הרצות של פילטרים:

<u>Chained</u>: מריצים את הפילטרים בסדר שהתקבלו מהקליינט כאשר הפלט של הפילטר הראשון נכנס כקלט של הפילטר השני, הפלט של השני נכנס כקלט של השלישי, וכך הלאה...

<u>Unordered</u>: מריצים את הפילטרים בסדר כלשהו. אין קשר בין פילטר אחד לשני.

המחלקה שומרת את הפילטרים שהקליינט מעוניין להריץ בוקטור:

```
std::vector<std::string> _frontCameraFilters;
```

המחלקה שומרת את סוג ההרצה במשתנה:

```
bool _useUnorderedListFront;
```

כאשר הוא TRUE, משתמש ב-Unordered, אחרת משתמשים ב-Chained.

בכדי לשנות את סוג ההרצה, או את רשימת הפילטר שהקליינט רוצה להריץ, פונקציה changeList ב-main.cpp מקבלת מהקליינט את רשימת הפילטרים וסוג ההרצה, ואז שולחת את רשימת הפילטרים לאחת מהפונקציות, לפי סוג ההרצה (ב-FilterRun):

```
void useUnorderedFilterList(const std::vector<std::string>&);

void useChainFilterList(const std::vector<std::string>&);
```

המחלקה VideoStream לוקחת תמונה מהמצלמה (או סרטון, או תמונה שנשלחה אליה) ועבור כל תמונה שהיא לוקחת, היא שולחת אותה לפונקציה (ב-FilterRun):

```
map<std::string,cv::Mat*> run(cv::Mat*);
```

אשר מריצה את התמונה על הפילטרים שהקליינט רוצה ולפי סוג ההרצה שהוא רוצה ומחזירה מבנה נתונים אשר מכיל את התמונה הכי פילטור של כל הפילטרים.

CreatedFilter: מחלקה השומרת מידע על כל Created filter. השם שלו, וקטור של הפילטרים שלו ווקטור שמות הפילטרים (כנראה שהיה עדיף אם במקום 2 וקטורים הייתי משתמש ב-map, תעשו את זה אם יש לכם כח).

Main: בכל בקשה של הקליינט מהסרבר, הקליינט שולח 3 bytes המייצגים קוד של פעולה כלשהי. לפי קבלת הקוד, הסרבר יודע איזה פעולה הקליינט רוצה והוא קורא לפונקציה המתאימה.

פונקציות ב-main.cpp:

<u>אם אתה רוצה לשנות קובץ config:</u>

```
void changeConfig()
```

הפונקציה מקבלת מהקליינט את הקובץ, עושה overwrite על הקובץ הנוכחי וקוראת לפונקציה **updateConfigs** ב-FilterHandler אשר טוענת מחדש את כל הפילטרים. למה אנחנו רוצים לטעון מחדש? מפני שאפשר לשנות את הקונפיגורציות של פילטר רק באמצעות יצירת האובייקט מחדש עם הקובץ config שלו המעודכן.

<u>אם אתה רוצה לשנות את רשימת הפילטרים שיפלטרו את התמונה:</u>

```
void changeList(bool unordered)
```

המיין קורא לפונקציה הזאת עם ערך True כאשר הקליינט רוצה להריץ ב-unordered ו-false אם הקליינט רוצה להריץ את הפילרים ב-chained. הפונקציה הזאת מקבלת רשימת פילטרים מהקליינט, מפסיקה את שידור הוידיאו (אם קיים) ואז קוראת לאחת מהפונקציה ב-FilterRun (לפי סוג ההרצה) המעדכנות את הרשימה:

```
void useUnorderedFilterList(const std::vector<std::string>&);

void useChainFilterList(const std::vector<std::string>&);
```

<u>אם אתה רוצה למחוק פילטר מהמערכת:</u>

```
void deleteFilter()
```

הפונקציה מקבלת מהקליינט את השם של הפילטר. דבר ראשון בודקים האם הפילטר בשימוש. אם כן, **אי אפשר** למחוק אותו. אם הוא לא בשימוש בודקים אם הפילטר שייך ל-SO או ל-Created: אם כן נמחוק אותו. אחרת, הפילטר שייך ל-Built-in או שהוא לא קיים במערכת ונשלח שגיאה לקליינט (אם הוא שייך ל-Built-in לא נוכל למחוק אותו, מפני שהאובייקט מקומפל עם הסרבר!).

<u>אם אתה רוצה להוסיף פילטר SO למערכת:</u>

```
void addFilter()
```

הפונקציה מקבלת 2 קבצים מהקליינט: something.config, something.so. היא שומרת אותם במקום הדרוש, מפסיקה את שידור הוידאו (אם קיים) ואז טוענת את כל הפילטרים מחדש. טעינת כל הפילטרים מחדש היא בזבוז זמן. אם יש לכם זמן תעשו שזה פשוט יטען רק את הפילטר החדש. בכל מקרה, אני מניח שלא יתבצעו הרבה פעולות כאלה, לכן זה לא כל כך משנה.

אם אתה רוצה ליצור Created פילטר:

```
void createFilter()
```

הפונקציה מקבלת את השם של הפילטר החדש שאתה רוצה, ואז יוצרת אותו מרשימת הפילטרים הקיימים במערכת. שים לב! אם אתה רוצה ליצור פילטר חדש מפילטרים כלשהם, עליך קודם כל לעדכן את הרשימה של הפילטרים במערכת ורק אז ליצור את הפילטר. עוד הערה: אפשר ליצור created פילטר רק מהרצה של chained. אם תרצה ליצור created כאשר סוג ההרצה הוא unordered, המערכת פשוט תבחר פילטר כלשהו שיהיה הפילטר החדש.

אם אתה רוצה לשמור לוידאו חלק תמונות שמפולטרות ע"י פילטר כלשהו:

```
void record(bool start)
```

הערך start אומר האם להתחיל לשמור וידאו או לסיים שמירת וידאו נוכחית. הפונקציה מקבלת רשימת פילטרים שהקליינט רוצה לשמור כוידאו ושולחת את הרשימה הזו אל המחלקה VideoStream.

אם אתה רוצה לקבל סטטיסטיקות של הדיסק קשיח שעל הצוללת:

```
void sendHDDStats()
```

אין הרבה מה להסביר. פשוט שולחת String שמייצג את הסטטיסטיקות של הדיסק קשיח. תריצו בטרמינל במחשב שלכם את הפקודה "df -h" והפלט הזה זה מה שהפונקציה שולחת לקליינט.

סרבר שעל הצוללת

כל הפונקציות והמחלקות שהוזכרו למעלה שייכות גם לסרבר שעל הצוללת. ההבדל בין הסרבר הלוקאלי לסרבר של הצוללת הוא שבכל פעולה בסרבר שעל הצוללת יש לבחור גם מצלמה עליה הפעולה תתבצע. לדוגמא, לכל מצלמה יש וקטור משלה המייצג את הפילטרים שרצים עליה וגם משתנה בוליאני משלה המייצג את סוג ההרצה של כל מצלמה. אם נרצה לשנות את הפילטרים של מצלמה כלשהי, הקליינט יצטרך לציין בכל בקשה שלו על איזה מצלמה הוא פועל. מרבית הפונקציות מקבלות משתנה בוליאני המייצג את המצלמה שאנחנו פועלים עליה.

יש לסרבר שעל הצוללת 2 מחלקות נוספות:

- CamerasControl
- FrontCamera

אשר נכתבו ע"י יובל. אני לא בקיא במחלקות האלה. הן אחראיות על המצלמות.

קבצי .so

אם יש לכם קוד של פילטר חדש שאתם מעוניינים לשלוח לסרבר, עליכם לקמפל את המחלקה הזו לקובץ .so. איך זה מתבצע?

שימו בתיקייה כלשהי את כל הקבצים הדרושים לקימפול (כל הקבצים אשר יש להם "include"). שימו לב שאם אתם דורשים את המחלקה a ומהחלקה a דורשת את המחלקה b, עליכם גם להוסיף את b לתיקייה. אחרי הוספת כל הקבצים לתיקייה יש לקמפל עם פקודה:

```
g++ -fPIC -shared Utils.cpp ParamUtils.cpp BaseAlgorithm.cpp TorpedoAlgo.cpp -o a.so
```

כאשר Utils.cpp ParamUtils.cpp BaseAlgorithm.cpp TorpedoAlgo.cpp הן המחלקות שאני צריך לדוגמא, ו-a יהיה השם של קובץ הפלט.

הערה חשובה!!!: בכדי שהמערכת תוכל לטעון את הקובץ הזה באופן דינאמי (בזמן ריצה ללא קימפול), עליכם להוסיף לקובץ cpp של הפילטר שלכם (ולא לקבצים שהוא עושה להם include) 2 פונקציות:

```
extern "C" BaseAlgorithm* maker(){
        return new TorpedoAlgo;
}
extern "C" int offset() {
        return 1;
}
```

<u>הפונקציה הראשונה</u>: שימו לב שהיא מחזירה **אובייקט** TorpedoAlgo. תשאירו את החתימה דומה, אך את השם של האובייקט תשנו לשם של המחלקה שלכם.

<u>הפונקציה השניה</u>: אם הפילטר צריך offset תשאירו אותה כמו שהיא. אחרת, תשלחו 0 במקום 1.

אם אתם מעוניינים להיכנס לעומק של הפונקציות האלה וטעינה דינאמית יש לכם פה אחלה מקור:

http://www.linuxjournal.com/article/3687

מדריך למשתמש

<u>דרישות</u>:

- Boost 1.55.0 (זו הגירסא בה אני השתמשתי)
- OpenCV בגירסא הכי חדשה אם אפשר (השתמשתי בגירסא 2.4.8).
- eclipse בכדי להריץ את הסרברים

אחרי יצירת הפרוייקט ב-eclipse, הכנסו לאפשרויות של הפרוייקט, לחצו על "C/C++ Build" ואז על "Settings".

באפשרות "GCC C++ Compiler" לחצו על "Includes" והוסיפו את ה-path של OPENCV. אצלי זה:

"/usr/local/include/opencv"

באפשרות"GCC C++ Linker" לחצו על "Libraries" והוסיפו בקוביה הראשונה את:

opencv_core

boost_filesystem

dl

boost_system

boost_thread

opencv_imgproc

opencv_highgui

ובקוביה השניה את המיקום של הספריות שלכם. אצלי זה:

"/usr/local/lib"

תקמפלו ותבדקו שעובד. אם לא עובד יש לכם כנראה בעיה במיקום של הסיפריות או בהתקנה של הסיפריות. השתמשו ב-Google.

הסרברים כרגע עובדים רק בלינוקס.

אחרי שהרצתם את הסרבר אתם יכולים לפעול עליו רק דרך הגוי.

Future Work

ממליץ לכם להוסיף אפשרות להתחבר ל-VideoStream עם כמה קליינטים. יהיה מגניב לראות את הפלט בכמה מחשבים.

לא הספקנו להריץ את הסרבר כאשר הצוללת במים. מקווה שלא יהיו יותר מדי בעיות עם זה. אני ממליץ לכם להוסיף עוד טיפול בשגיאות (נפילת אינטרנט, בקשות לא טובות מהקליינט וכו'...). הסרבר לדעתי חשוף ליותר מדי שגיאות.

כרגע, אם יצרנו Created filter בשם fil, אז אי אפשר ליצור Created filter חדש המשתמש ב-fil. תנסו לאפשר זאת.

בתכלס, הסרבר עושה כל מה שדרוש ממנו, לכן כל עוד לא ישנו את הדרישות אין לכם הרבה שינויים לעשות.

מוזמנים לפנות אלי באימייל:

elirankoren@gmail.com

אם יש לכם שאלות כלשהם לגבי הסרבר.

עבודה נעימה.