

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

ĐỀ TÀI:

**TÌM HIỂU HỆ ĐIỀU HÀNH ANDROID
XÂY DỰNG ỨNG DỤNG DỊCH TỰ ĐỘNG VĂN BẢN**

Chuyên ngành

: KHOA HỌC MÁY TÍNH

TP. Hồ Chí Minh, ngày 20 tháng 05 năm 2015

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

ĐỀ TÀI:

**TÌM HIỂU HỆ ĐIỀU HÀNH ANDROID
XÂY DỰNG ỨNG DỤNG DỊCH TỰ ĐỘNG VĂN BẢN**

Giảng viên hướng dẫn : **ThS. TÔN LONG PHƯỚC**

Sinh viên thực hiện :

NGUYỄN QUỐC CHÍ 11093111

PHAN VĂN HƯNG 11246131

Lớp : **DHTH7A**

Chuyên ngành : **KHOA HỌC MÁY TÍNH**

TP. Hồ Chí Minh, ngày 20 tháng 05 năm 2015

LỜI CẢM ƠN

Để hoàn thành bài báo cáo này, trước hết chúng em xin chân thành cảm ơn ban giám hiệu Trường Đại Học Công Nghiệp TP.HCM đã tạo điều kiện cho chúng em học tập và nghiên cứu, chúng em xin cảm ơn các thầy giáo, cô giáo thuộc Khoa Công Nghệ Thông Tin Trường Đại Học Công Nghiệp TP.HCM - những người đã dạy dỗ, trang bị cho chúng em những kiến thức chuyên môn và giúp cho chúng em hiểu rõ hơn các lĩnh vực đã nghiên cứu để hoàn thành bài báo cáo này.

Chúng em xin bày tỏ lòng biết ơn sâu sắc nhất tới thầy giáo ThS. Tôn Long Phước, người đã hướng dẫn, chỉ bảo tận tình để chúng em hoàn thành đề tài này.

Xin cảm ơn bạn bè và gia đình đã động viên, đóng góp ý kiến, trao đổi, động viên trong suốt quá trình học tập và nghiên cứu, giúp chúng em hoàn thành tốt đề tài này và đúng thời hạn.

Chúng em xin chân thành cảm ơn!

TP. Hồ Chí Minh, ngày 20 tháng 05 năm 2015

Nhóm sinh viên thực hiện

Nguyễn Quốc Chí – Phan Văn Hưng

NHẬN XÉT CỦA GIẢNG VIÊN

LỜI MỞ ĐẦU

Mạng điện thoại di động xuất hiện tại Việt Nam từ đầu những năm 1990 và theo thời gian số lượng các thuê bao cũng như các nhà cung cấp dịch vụ di động tại Việt Nam ngày càng tăng. Do nhu cầu trao đổi thông tin và sử dụng sản phẩm công nghệ cao ngày càng nhiều nên các nhà cung cấp phải luôn cải thiện, nâng cao những sản phẩm của mình bằng cách tích hợp nhiều tính năng, nâng cao cấu hình, đưa ra các sản phẩm chất lượng tốt hơn, kiểu dáng mẫu mã đẹp và phong phú hơn. Chính vì những lẽ đó nên việc xây dựng ứng dụng cho điện thoại di động đang là một ngành công nghiệp mới đầy tiềm năng hứa hẹn nhiều sự phát triển vượt bậc.

Cùng với sự phát triển của thị trường điện thoại di động là sự phát triển mạnh mẽ của xu hướng lập trình trên các thiết bị di động. Ứng dụng trên điện thoại di động hiện nay rất đa dạng và phong phú nên các hệ điều hành di động cũng phát triển mạnh mẽ và đang thay đổi từng ngày. Các hệ điều hành **Symbian OS, iOS, Windows Phone, Web based Mobile Application...** đã rất phát triển trên thị trường truyền thông di động.

Trong vài năm trở lại đây, hệ điều hành **Android** ra đời kế thừa những ưu việt của các hệ điều hành đi trước và kết hợp nhiều công nghệ tiên tiến nhất, được nhà phát triển công nghệ nổi tiếng **Google** phát triển. **Android** đã nhanh chóng là đối thủ cạnh tranh mạnh mẽ với các hệ điều hành trước đó và đang là hệ điều hành di động được nhiều người ưa chuộng nhất.

Ngày nay, với sự phát triển của công nghệ nhận dạng ký tự quang học (**Optical Character Recognition**) hay còn được gọi tắt là **OCR**. Đây là một công nghệ giúp chuyển đổi hình ảnh dạng văn bản của chữ viết tay hoặc đánh máy thành các ký tự đã được mã hóa trong máy tính. Vì thế, cần một ứng dụng trên thiết bị di động để rút trích văn bản từ hình ảnh chụp nhằm thuận tiện phục vụ người dùng. Chính vì lẽ đó, nhóm chúng em chọn đề tài “**Tìm hiểu hệ điều hành Android – Xây dựng ứng dụng dịch tự động văn bản**” với mục đích nghiên cứu, tìm hiểu nền tảng **Android** và xây dựng ứng dụng nhận dạng văn bản từ hình ảnh và hỗ trợ dịch sang ngôn ngữ khác để đáp ứng cho những yêu cầu trên.

PHẠM VI THỰC HIỆN ĐỒ ÁN

Trong quá trình tìm hiểu để thực hiện đồ án, chúng em đã phân chia đồ án thành ba công đoạn chính:

1. Tìm hiểu hệ điều hành Android
 - Tổng quan về hệ điều hành Android, các phiên bản, kiến trúc và thiết kế
 - Cách xây dựng một ứng dụng Android bằng công cụ *Android Studio*
2. Tìm hiểu công cụ nhận dạng ký tự quang học (*OCR*)
 - Giới thiệu chung về nhận dạng ký tự quang học, các phương pháp nhận dạng
 - Tìm hiểu bộ công cụ *Tesseract*
3. Xây dựng ứng dụng Android dịch tự động văn bản
 - Tích hợp công cụ *Tesseract OCR* lên ứng dụng android hỗ trợ chuyển đổi hình ảnh văn bản sang văn bản
 - Xử lý ảnh đơn giản bằng các thư viện sẵn có của *Android* và công cụ *OpenCV* (*Open Source Computer Vision*)
 - Sử dụng các thư viện hỗ trợ như: *GSon*, *JTAR*
 - Sử dụng dịch vụ *Google Translate* để hỗ trợ dịch tự động văn bản

Chúng em đã cố gắng tìm hiểu công cụ *Tesseract* hỗ trợ việc chuyển đổi hình ảnh văn bản sang văn bản, xử lý hình ảnh trước khi đưa vào rút trích văn bản bằng công cụ *OpenCV* cùng với việc sử dụng dịch vụ *Google Translate* thông qua các hàm *API* để hỗ trợ dịch tự động văn bản.

MỤC LỤC

LỜI CẢM ƠN	i
LỜI MỞ ĐẦU	iii
PHẠM VI THỰC HIỆN ĐỒ ÁN	iv
GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH ANDROID	1
1. <i>Tổng quan</i>	1
2. <i>Các phiên bản Android</i>	2
3. <i>Kiến trúc và thiết kế</i>	3
4. <i>Máy ảo Dalvik</i>	5
5. <i>Android Software Development Kit (SDK)</i>	6
6. <i>Lập trình ứng dụng Android</i>	8
6.1. Tạo ứng dụng đầu tiên	8
6.2. Các thành phần chính của một ứng dụng Android	14
6.3. <i>Một số thành phần cơ bản khác của Android</i>	18
NHẬN DẠNG KÝ TỰ QUANG HỌC	24
1. <i>Giới thiệu chung</i>	24
1.1. <i>Sơ lược về nhận dạng ký tự quang học – OCR</i>	24
1.2. <i>Các phương pháp áp dụng OCR</i>	25
1.3. <i>So sánh các thư viện công cụ nhận dạng ký tự quang học</i>	26
1.4. <i>Kết luận</i>	27
2. <i>Giới thiệu về bộ nhận dạng ký tự quang học Tesseract</i>	28
2.1. <i>Lịch sử</i>	28
2.2. <i>Kiến trúc hoạt động</i>	29

2.3. Xác định dòng và từ.....	30
2.4. Một số thử nghiệm	32
2.5. Kết luận	35
XÂY DỰNG ỨNG DỤNG ANDROID	36
1. Mục tiêu.....	36
2. Cài đặt các thư viện hỗ trợ.....	37
2.1. Thư viện Tesseract.....	37
2.2. Thư viện OpenCV	41
2.3. Thư viện GSON và JTAR.....	47
3. Xây dựng ứng dụng	49
4. Cài đặt và sử dụng ứng dụng	56
5. Kết quả đạt được	59
KẾT LUẬN.....	60
TÀI LIỆU THAM KHẢO	61
PHỤ LỤC	62
1. Native development kit (NDK)	62
1.1. Giới thiệu chung	62
1.2. Các hỗ trợ của NDK.....	62
1.3. Sử dụng NDK.....	62
1.4. Nội dung của bộ NDK	63
2. Thư viện xử lý hình ảnh OpenCV (Open Source Computer Vision).....	64
2.1. Giới thiệu Computer Vision.....	64
2.2. Lịch sử phát triển.....	64

2.3. Cấu trúc của OpenCV	65
--------------------------------	----

GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH ANDROID

1. Tổng quan

Android là một hệ điều hành mở dựa trên nền tảng *Linux* dùng cho các thiết bị di động bao gồm điện thoại thông minh, máy tính bảng, máy tính xách tay. Được phát triển ban đầu tại công ty liên hợp *Android* sau đó công ty này được *Google* mua lại vào năm 2005 và biến Android thành một hệ điều hành mở trên các thiết bị di động.

Android chính thức ra mắt vào ngày 5/11/2007 cùng với sự ra đời của liên minh thiết bị cầm tay mở *OHA (Open Handset Alliance)*. Liên minh *OHA* là một tổ chức bao gồm khoảng hơn 78 công ty viễn thông, di động và phần cứng như *Google*, *Sony Ericsson*, *Samsung*, *Nvidia*, *Qualcomm*,... Mục tiêu của tổ chức này là phát triển các chuẩn mở chung cho thiết bị di động trong tương lai và Android là sản phẩm chủ lực của hãng. Mã nguồn của Android là mã nguồn mở và được công bố dưới dạng giấy phép *Apache*.



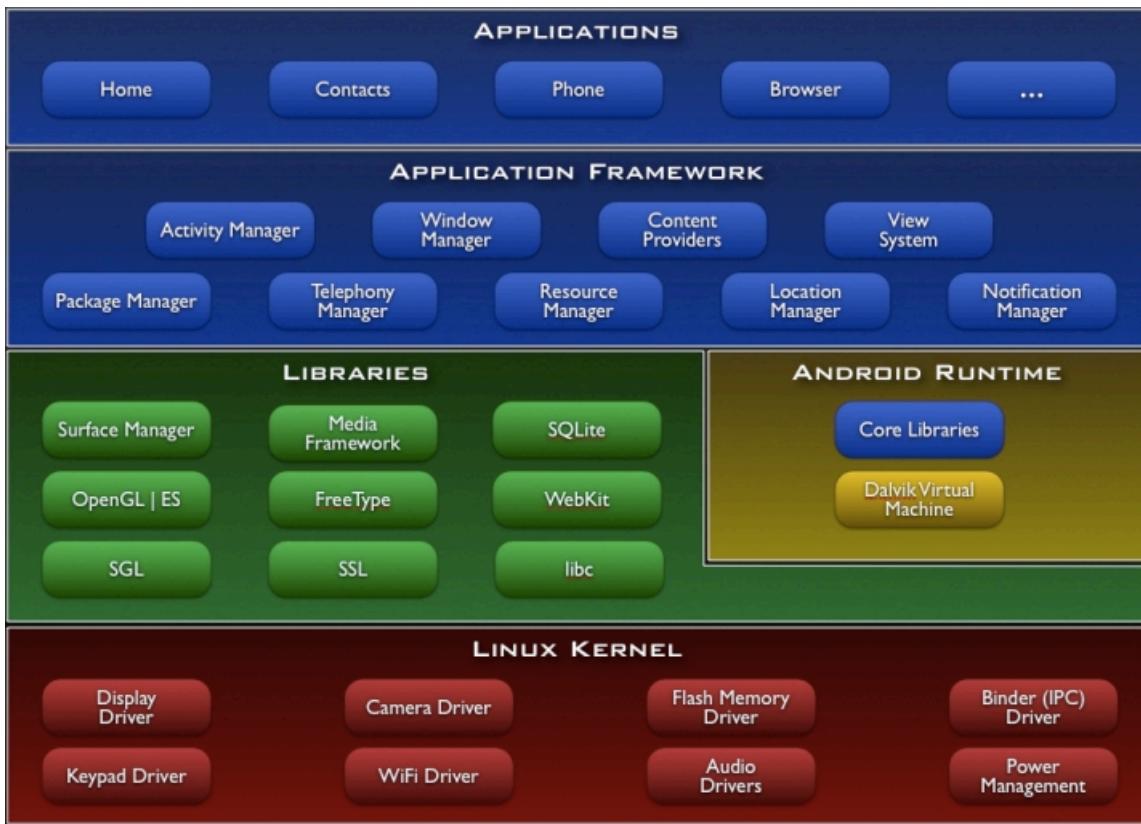
Hình 1: Các điện thoại dùng hệ điều hành Android

2. Các phiên bản Android

Từ lúc ra đời vào năm 2007 đến nay, Android đã tung ra nhiều phiên bản khác nhau với những nâng cấp và cải tiến theo từng phiên bản. Sau đây là danh sách các phiên bản Android đã ra đời:

- Android 1.0 phát hành tháng 09/2008
- Android 1.1 phát hành tháng 02/2009
- Android 1.5 (**Cupcake**) phát hành tháng 04/2009
- Android 1.6 (**Donut**) phát hành tháng 09/2009
- Android 2.0 (**Eclair**) phát hành tháng 10/2009
- Android 2.2 (**Froyo**) phát hành tháng 05/2010
- Android 2.3 (**Gingerbread**) phát hành tháng 12/2010
- Android 3.0 (**Honeycomb**) phát hành tháng 02/2011
- Android 4.0 (**Ice Cream Sandwich**) phát hành cuối năm 2011
- Android 4.1 (**Jelly Bean**) phát hành năm 2012
- Android 4.4 (**KitKat**) phát hành tháng 10/2013
- Android 5.0 (**Lollipop**) phát hành tháng 06/2014
- Android 5.1 (**Lollipop**) phát hành tháng 03/2015 và hiện đang là phiên bản mới nhất hiện nay.

3. Kiến trúc và thiết kế



Hình 2: Kiến trúc tổng thể của hệ điều hành Android

Nhìn vào kiến trúc thì hệ điều hành Android được chia thành các tầng bao gồm: **Applications**, **Application Framework**, **Libraries**, **Android Runtime**, **Linux Kernel**. Trong đó 2 tầng **Applications** và **Application Framework** được viết bằng ngôn ngữ **Java**. Còn các tầng từ **Libraries** đến **Linux Kernel** được viết bằng ngôn ngữ **C/C++** hay còn gọi là mã gốc - **Native Code**.

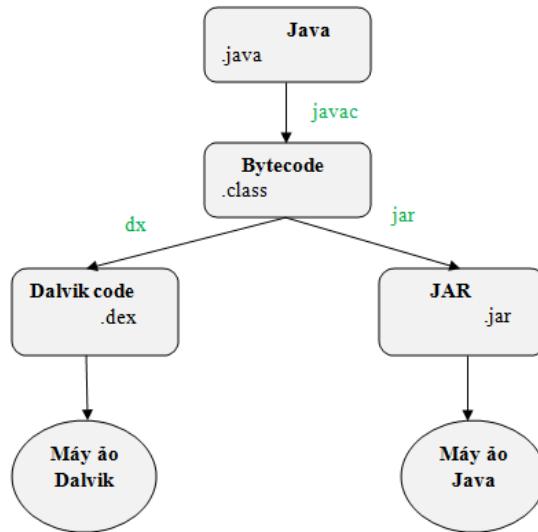
- **Tầng Applications**: đây là tầng cao nhất trong hệ điều hành Android. Tầng này bao gồm các ứng dụng được viết và cài đặt sẵn như: lịch, trình duyệt web, danh bạ, Camera,... Các ứng dụng tại tầng này đều được viết bằng ngôn ngữ **Java**.
- **Tầng Application Framework**: bên dưới tất cả các ứng dụng là một tập hợp các dịch vụ và hệ thống cho phép các nhà phát triển phần mềm có thể gọi các hàm hỗ trợ sẵn qua giao diện lập trình ứng dụng **API (Application Programming Interface)**.

- Tập hợp các màn hình (**Views**): dùng để xây dựng nên giao diện chương trình như các nút bấm, danh sách, hộp thoại, text box, các sự kiện...
 - Bộ cung cấp nội dung (**Content Provider**): cung cấp khả năng truy xuất và chia sẻ dữ liệu giữa các ứng dụng.
 - Quản lý tài nguyên (**Resource Manager**): quản lý các loại tập tin không phải là mã nguồn. Cung cấp khả năng truy cập đến các tài nguyên khác trong ứng dụng như các chuỗi, tập tin đồ họa, các tập tin định dạng giao diện (**Layout**).
 - Quản lý thông báo (**Notification Manager**): quản lý và hiển thị các thông báo ở thanh trạng thái (**Status Bar**).
 - Quản lý hoạt động (**Activity manager**): quản lý vòng đời và chu trình hoạt động của các ứng dụng.
- **Tầng Libraries:** Android có hệ thống các thư viện **C/C++** được sử dụng nhiều trong các thành phần khác nhau của hệ điều hành. Một số các thư viện **C/C++** chính trong Android:
 - **System C Library:** một **BSD (Berkeley Software Distribution)** được thừa kế từ các thư viện chuẩn **C** và được tinh chỉnh cho các thiết bị sử dụng trên nền **Linux**.
 - **Media Library:** thư viện hỗ trợ cho việc ghi âm, thực thi các định dạng nhạc, phim và hiển thị các ảnh bao gồm các định dạng sau: **MPEG4, H.264, MP3, AAC, ARM, JPG, PNG, ...**
 - **Surface Manager:** quản lý truy cập vào hệ thống hiển thị.
 - **Live Webcore:** công cụ trình duyệt web.
 - **SGL:** các hàm cơ bản về đồ họa 2 chiều.
 - **3D Library:** thư viện đồ họa 3 chiều.
 - **Freetype:** biểu diễn các font và vectơ bitmap.
 - **SQLite:** hệ cơ sở dữ liệu.

- **Tầng Android Runtime:** bao gồm một tập các thư viện lõi **Java** và máy ảo Dalvik. Máy ảo Dalvik thực thi các tập tin định dạng **dex**. Mỗi ứng dụng được chạy trên một tiến trình riêng của máy ảo Dalvik. Trên cùng một thiết bị có thể chạy nhiều máy ảo Dalvik khác nhau một cách hiệu quả.
- **Tầng Linux Kernel:** đây là tầng thấp nhất trong hệ điều hành Android, được xây dựng trên nhân của **Linux 2.6** chứa các trình quản lý thiết bị như keypad, wifi, âm thanh, quản lý điện năng... và các dịch vụ của hệ thống như: an ninh, quản lý bộ nhớ, quản lý tiến trình, kết nối mạng,... Tầng này đóng vai trò là tầng trung gian liên lạc giữa phần cứng và ngăn xếp phần mềm ở các tầng trên.

4. Máy ảo Dalvik

Dalvik là máy ảo để thực hiện các ứng dụng phần lớn viết bằng **Java** trên Android dưới định dạng là tập tin (**.dex**). Về cơ bản có thể nhận thấy máy ảo Dalvik có phần giống với máy ảo **Java** trên Desktop, tuy nhiên có phần khác là khi ta viết các ứng dụng trên **Java** thì mã nguồn sẽ được chuyển thành mã **bytecode**. Tại đây, một công cụ có sẵn trên Android là “**dx**” sẽ chuyển dạng mã **bytecode** này thành dạng tập tin **.dex** (viết tắt là **Dalvik Executable**) và được thực thi trên máy ảo Dalvik để chạy các ứng dụng Android.



Hình 3: Cơ chế hoạt động của máy ảo Dalvik và Java

5. *Android Software Development Kit (SDK)*

Bộ phát triển ứng dụng cho Android hay còn gọi là ***Android SDK*** được cung cấp cho các nhà phát triển phần mềm có thể lập trình, gỡ lỗi và kiểm thử ứng dụng được phát triển trên Android. Bộ ***SDK*** bao gồm:

- Thư viện lập trình Android (***Android API***): đây là phần cốt lõi của bộ phát triển Android, từ các thư viện lập trình ***Android API***, ***Google*** đã xây dựng nên các ứng dụng có sẵn.
- Công cụ phát triển: cung cấp sẵn cho các nhà phát triển các công cụ để lập trình, biên dịch, sửa lỗi mã nguồn trong ứng dụng.
- Tài liệu: đây là phần hướng dẫn sử dụng các thư viện, lớp/hàm có sẵn trong môi trường lập trình Android. Ngoài ra còn giải thích về cơ chế hoạt động của các ứng dụng trong Android.
- Ứng dụng mẫu: bộ ***Android SDK*** còn cung cấp các đoạn mã nguồn của chương trình có sẵn.

- Trình giả lập Android: để cung cấp sự thuận tiện cho người phát triển, bộ *Android SDK* đã cung cấp cho người dùng sẵn trình giả lập Android mô phỏng môi trường làm việc y như trên thiết bị thật để thuận tiện cho người phát triển có thể chạy hoặc sửa lỗi các ứng dụng trên thiết bị giả lập này mà không cần phải có thiết bị thật.



Hình 4: Màn hình trình giả lập điện thoại Android

6. Lập trình ứng dụng Android

Trong phần lập trình ứng dụng trên hệ điều hành Android, chúng em sẽ trình bày sơ lược về các thành phần cũng như cách xây dựng các thành phần của ứng dụng Android mà chúng em đã tìm hiểu trong quá trình thực hiện đồ án.

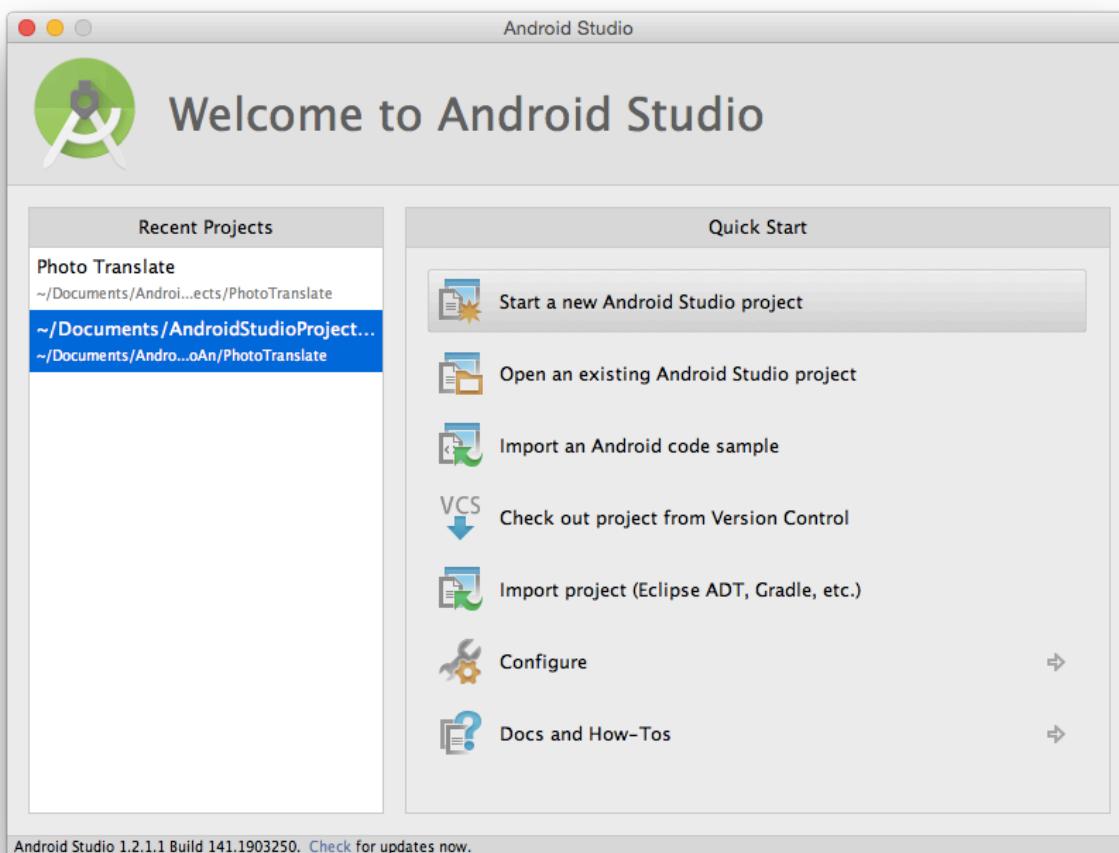
Công cụ lập trình: *Android Studio* phiên bản **1.2.1.1** dành cho *Mac*.

Môi trường cài đặt thử nghiệm: điện thoại *Sony Xperia Z1* sử dụng hệ điều hành *Android 5.0.2 Lollipop*.

6.1. Tạo ứng dụng đầu tiên

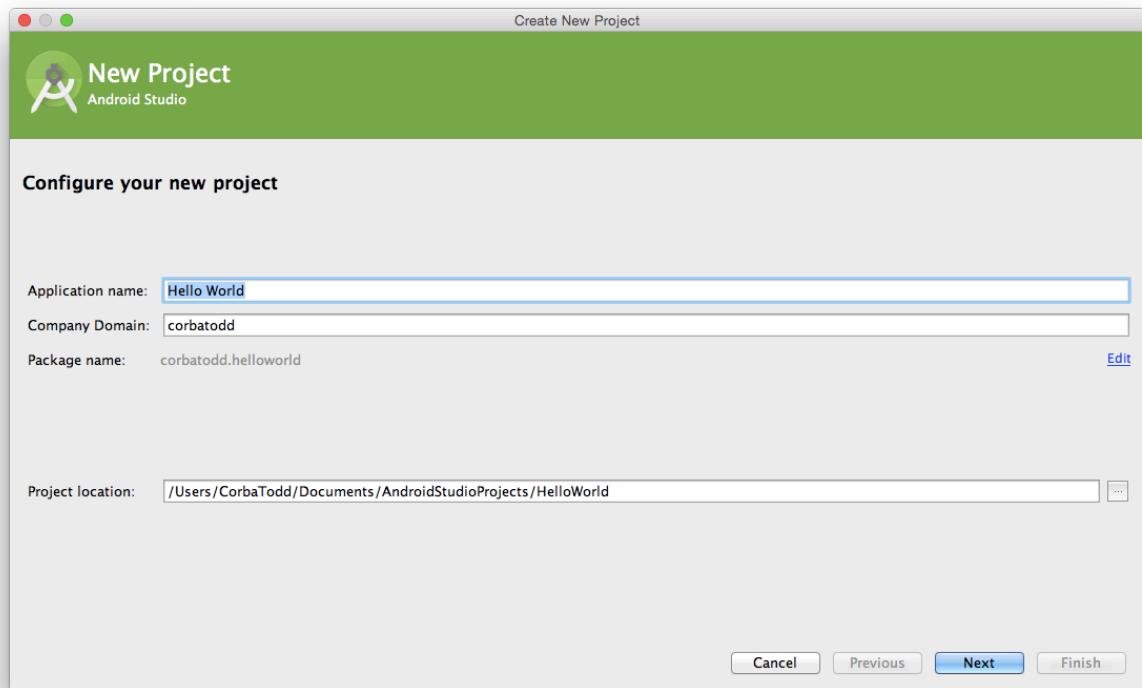
B1: mở ứng dụng *Android Studio*

B2: chọn *Start a new Android Studio project*



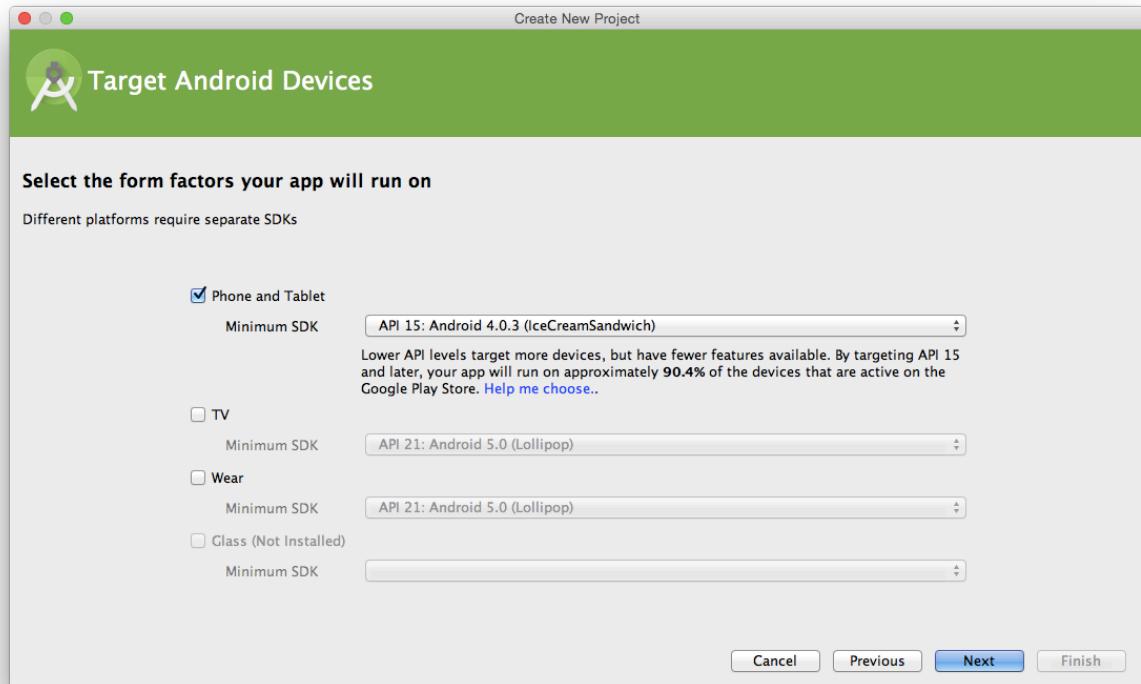
Hình 5: Mở ứng dụng Android Studio

B3: đặt tên ứng dụng tại ***Application name*** và chọn đường dẫn lưu tại ***Project location*** sau đó chọn **Next**



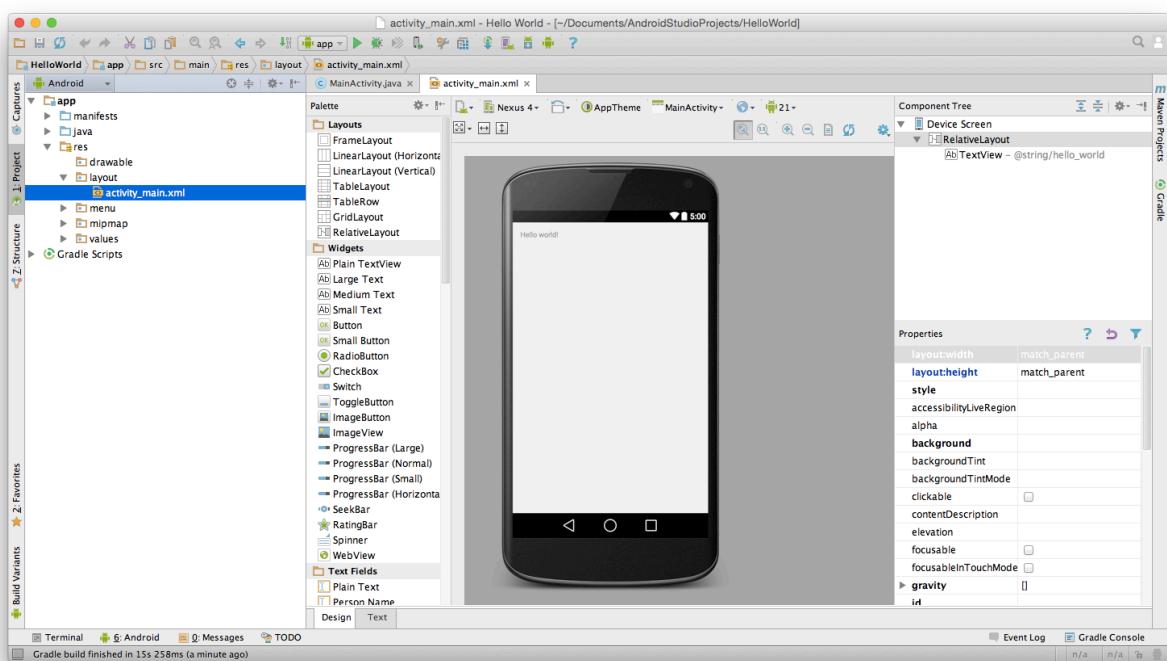
Hình 6: Đặt tên và lưu Project

B4: chọn thiết bị và phiên bản *Android* thấp nhất mà ứng dụng hỗ trợ sau đó chọn **Next**



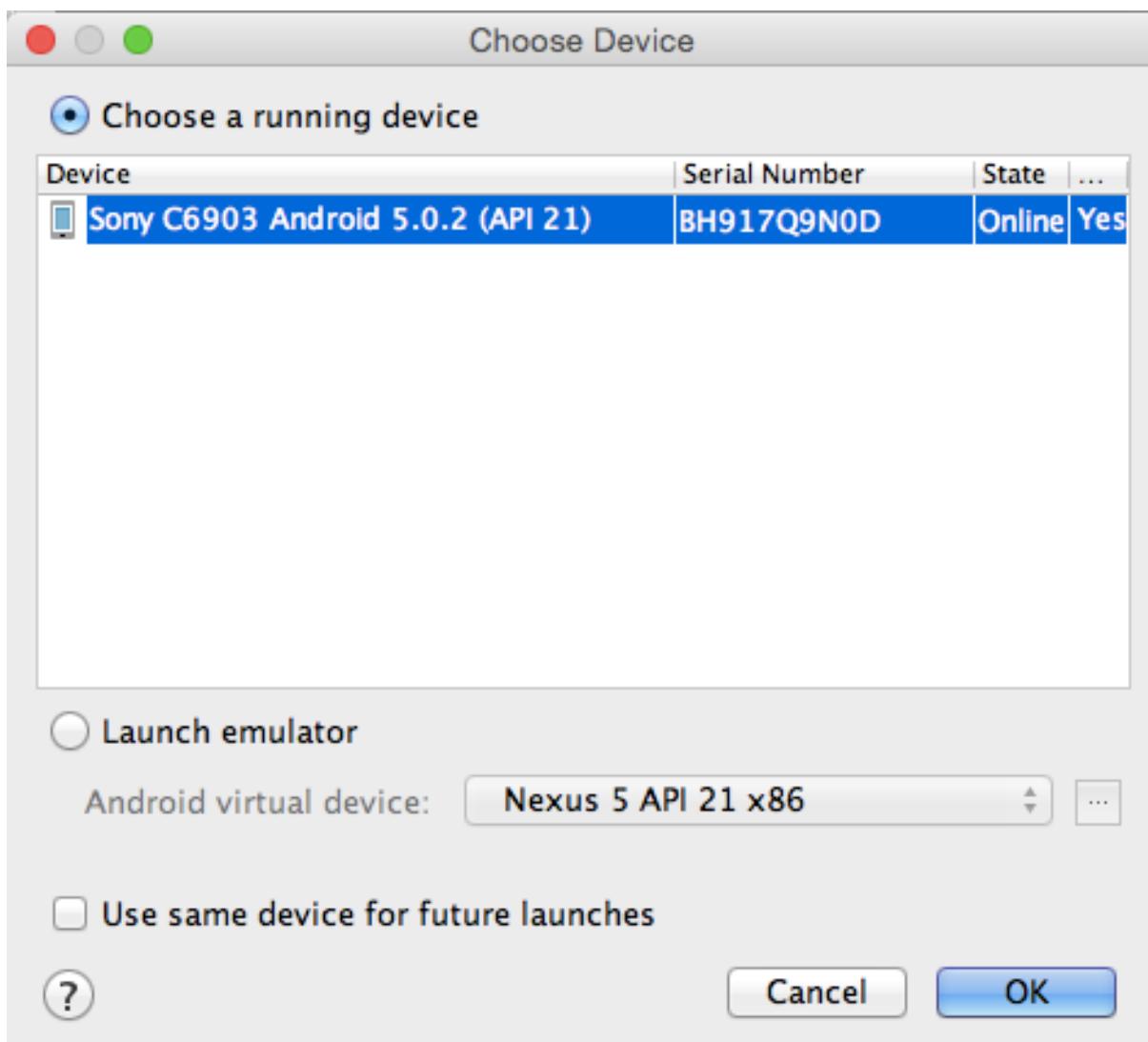
Hình 7: Chọn phiên bản Android tối thiểu hỗ trợ

B5: chọn **Blank Activity** và chọn **Next** sau đó chọn **Finish**
Android Studio sẽ bắt đầu quá trình khởi tạo project, sau khi hoàn tất ta được giao diện ứng dụng **Hello World**.



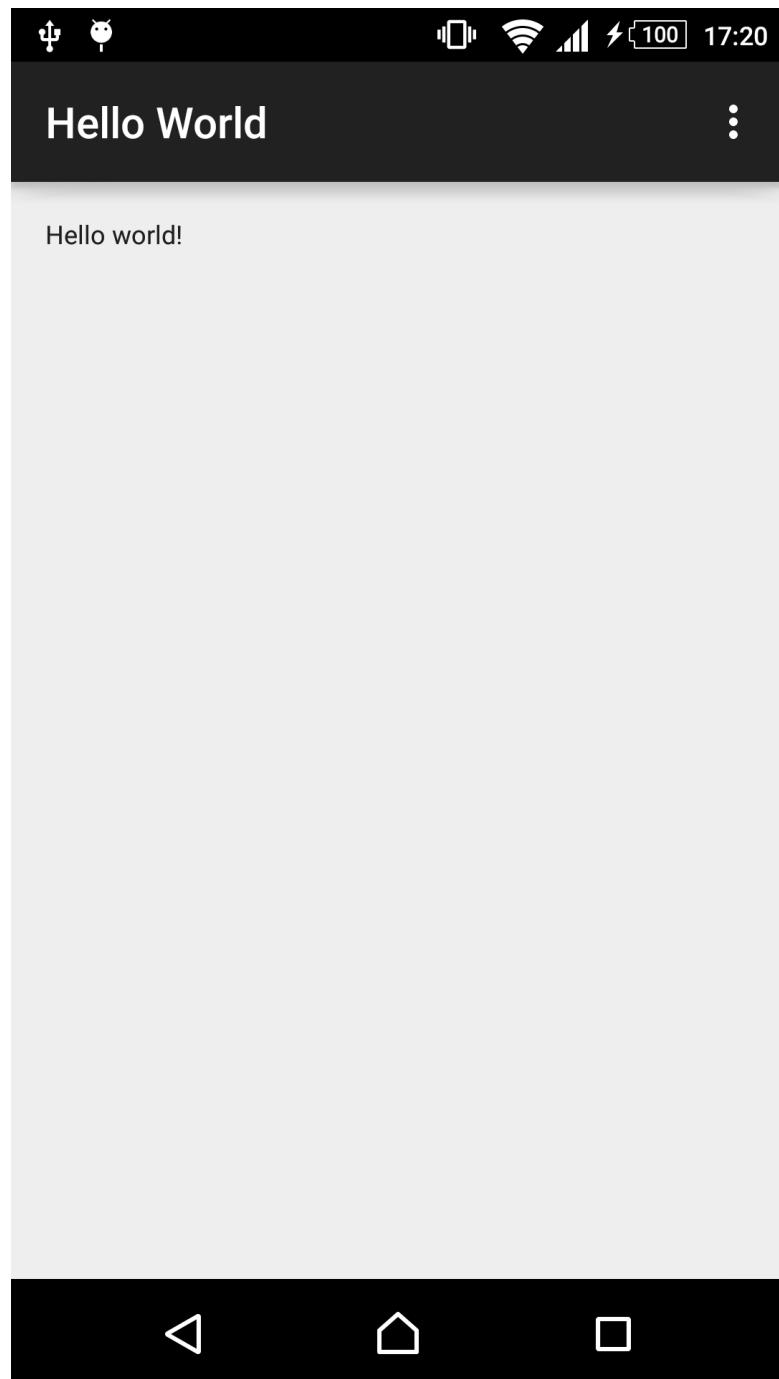
Hình 8: Khởi tạo Project

Thực thi project bằng cách chọn **Run app** trên thanh công cụ, chọn thiết bị cài đặt và **OK**. Project sẽ được cài đặt vào thiết bị và thực thi. Chọn **Launch emulator** nếu muốn cài đặt ứng dụng lên máy ảo.



Hình 9: Chọn thiết bị build ứng dụng

Giao diện sau khi build project lên thiết bị



Hình 10: Giao diện ứng dụng Hello World sau khi thực thi trên thiết bị

6.2. Các thành phần chính của một ứng dụng Android

Việc hiểu được các thành phần chính (**component**) tạo nên một ứng dụng Android là rất cần thiết cho việc lập trình. Các thành phần này được chia làm sáu loại bao gồm:

- **Activity**: hiểu một cách đơn giản thì **Activity** là nền của một ứng dụng. Khi khởi động một ứng dụng Android nào đó thì bao giờ cũng có một main **Activity** được gọi, hiển thị màn hình giao diện của ứng dụng cho phép người dùng tương tác.
- **Service**: thành phần chạy ẩn trong Android. **Service** sử dụng để cập nhật dữ liệu, đưa ra các cảnh báo (**Notification**) và không bao giờ hiển thị cho người dùng thấy.
- **Content Provider**: kho dữ liệu chia sẻ. **Content Provider** được sử dụng để quản lý và chia sẻ dữ liệu giữa các ứng dụng.
- **Intent**: nền tảng để truyền tải các thông báo. **Intent** được sử dụng để gửi các thông báo đi nhằm khởi tạo một **Activity** hay **Service** để thực hiện công việc ta mong muốn.

Ví dụ: khi mở một trang web, ta gửi một **Intent** đi để tạo một **Activity** mới hiển thị trang web đó.

- **Broadcast Receiver**: thành phần thu nhận các **Intent** bên ngoài gửi tới.
Ví dụ: ta viết một chương trình thay thế cho phần gọi điện mặc định của Android, khi đó ta cần một **Broadcast** để nhận biết các **Intent** là các cuộc gọi tới.
- **Notification**: đưa ra các cảnh báo mà không làm cho các **Activity** phải ngừng hoạt động.

Activity, **Service**, **Broadcast Receiver** và **Content Provider** mới là những thành phần chính cấu thành nên ứng dụng Android, bắt buộc phải khai báo trong tập tin **AndroidManifest.xml**.

Nội dung của một tập tin *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="corbatodd.helloworld" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />

                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Ví dụ để khai báo thêm một *NewActivity*, ta thêm đoạn code sau vào trong cặp thẻ *<application></application>*

```
<activity android:name=".NewActivity"/>
```

Để khai báo ứng dụng được phép sử dụng Internet, Camera hoặc có thể ghi file vào bộ nhớ máy ta thêm đoạn code sau vào trong cặp thẻ *<manifest></manifest>*

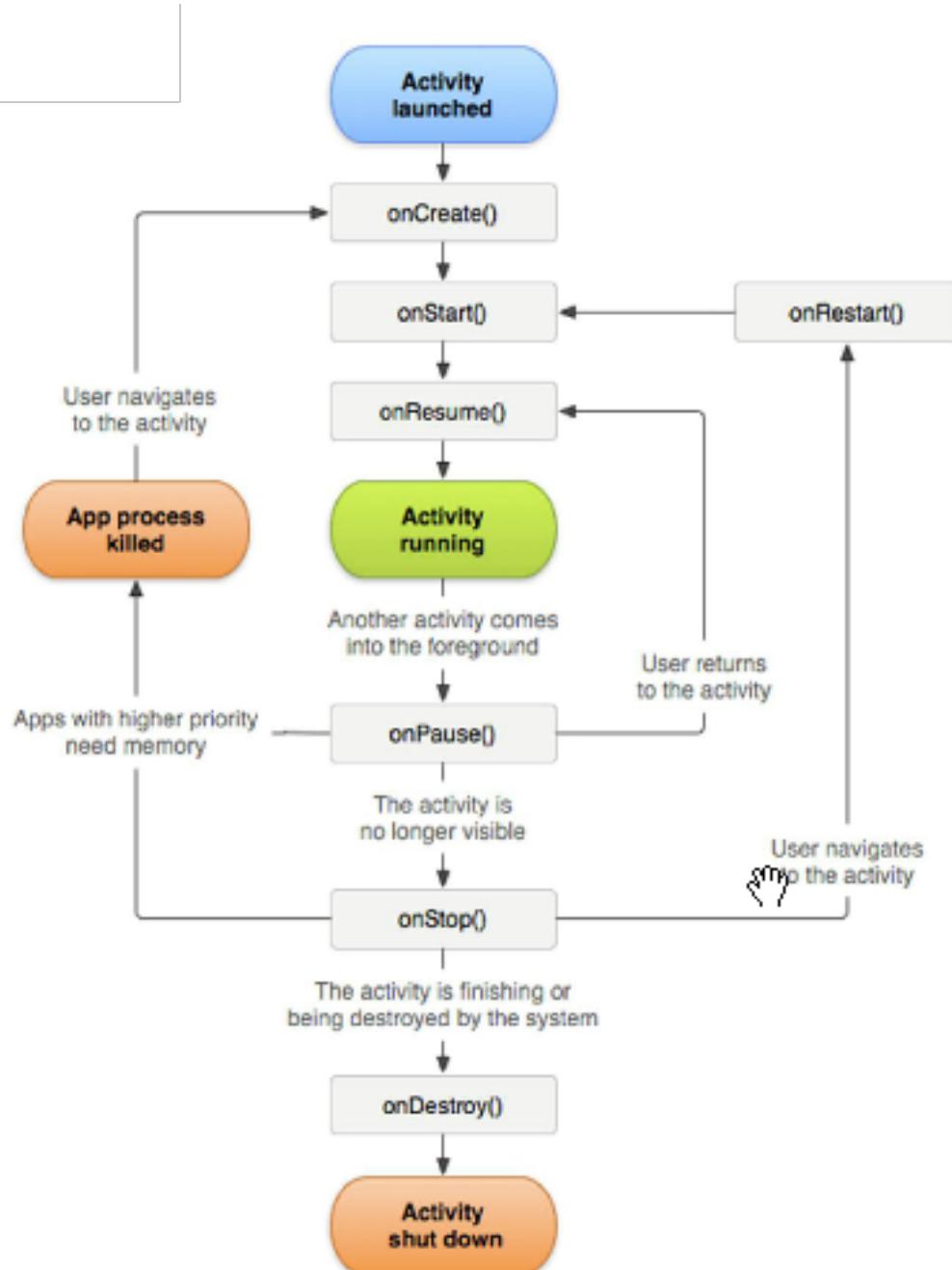
```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Hoặc để cấp phép cho ứng dụng sử dụng các đặc tính của máy như flash hay chức năng autofocus ta thêm đoạn code sau vào cùp thẻ `<manifest></manifest>`

```
<uses-feature  
    android:name="android.hardware.camera.autofocus"/>  
<uses-feature android:name="android.hardware.camera.flash"  
    android:required="false"/>  
<uses-feature android:name="android.hardware.camera"/>  
<uses-feature  
    android:name="android.hardware.camera.landscape"/>  
<uses-feature android:name="android.hardware.camera2"  
    android:required="true"/>
```

Android có cơ chế quản lý các process theo chế độ ưu tiên. Các process có priority thấp sẽ bị Android giải phóng mà không hề cảnh báo nhằm đảm bảo tài nguyên.

- **Foreground process:** là process của ứng dụng hiện thời đang được người dùng tương tác.
- **Visible process:** là process của ứng dụng mà *Activity* đang hiển thị đối với người dùng (*onPaused()* của *Activity* được gọi).
- **Service process:** là *Service* đang thực thi.
- **Background process:** là process của ứng dụng mà các *Activity* của nó không hiển thị với người dùng (*onStoped()* của *Activity* được gọi).
- **Empty process:** process không có bất cứ một thành phần nào active. Theo chế độ ưu tiên thì khi cần tài nguyên, Android sẽ tự động kill process, trước tiên là các empty process.



Hình 11: Vòng đời của một Activity

Activity bao gồm 4 trạng thái:

- **Active (running):** *Activity* đang hiển thị trên màn hình (*foreground*).
- **Paused:** *Activity* vẫn hiển thị (*visible*) nhưng không thể tương tác (*lost focus*).

Ví dụ: một *Activity* mới xuất hiện hiển thị giao diện đè lên *Activity* cũ nhưng giao diện này nhỏ hơn giao diện *Activity* cũ, do đó ta vẫn thấy được một phần giao diện của *Activity* cũ nhưng lại không thể tương tác với nó.

- **Stop:** *Activity* bị thay thế hoàn toàn bởi *Activity* mới sẽ tiến đến trạng thái stop.

- **Killed:** khi hệ thống bị thiếu bộ nhớ, nó sẽ giải phóng các tiến trình theo nguyên tắc ưu tiên. Các *Activity* ở trạng thái *stop* hoặc *paused* cũng có thể bị giải phóng và khi nó được hiển thị lại thì các *Activity* này phải khởi động lại hoàn toàn và phục hồi lại trạng thái trước đó.

6.3. Một số thành phần cơ bản khác của Android

Để thiết kế thật tốt giao diện trong Android, ta phải nắm chắc việc sử dụng các loại Layout cơ bản của nó. Các Layout cơ bản của Android bao gồm: *FrameLayout*, *LinearLayout*, *TableLayout*, *RelativeLayout*, *AbsoluteLayout*.

Việc nắm vững các loại Layout trên là rất cần thiết và để có thể thiết kế được các giao diện phức tạp ta cần kết hợp đồng thời nhiều loại Layout với nhau vì mỗi loại Layout đều có chức năng riêng của nó. Trước tiên ta cần tìm hiểu cách tạo một Layout mới và cách kết nối chúng vào *Activity*.

Khi tạo mới một *Android project*, Layout mặc định của nó là *activity_main.xml* nằm trong thư mục *layout* của *res*. Đó là Layout được chỉ định chạy đầu tiên của chương trình.

Một Layout đơn giản sau khi tạo project

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Và class *MainActivity* sẽ liên kết với layout này thông qua đoạn code
`setContentView(R.layout.activity_main)`

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Để tạo một *TextView* hiển thị một đoạn text “**Hello World**” ta thêm đoạn code sau vào file Layout.

```
<TextView
    android:id="@+id/tvHello"
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Để tạo một ***ImageView*** để hiển thị hình ảnh hoặc một ***Button OK*** ta cũng thực hiện tương tự.

```
<ImageView
    android:id="@+id/imvImage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<Button
    android:id="@+id/btOK"
    android:text="OK"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Sau khi hoàn tất giao diện mong muốn, mở class ***MainActivity*** đã được liên kết đến Layout và thực hiện việc khởi tạo ở hàm ***onCreate()***

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    TextView tvHello = (TextView) findViewById(R.id.tvHello);
    ImageView imvImage = (ImageView)
        findViewById(R.id.imvImage);
    Button btOK = (Button) findViewById(R.id.btOK);
}
```

Sau khi đã thực hiện khởi tạo ra có thể các chức năng mong muốn trên các thành phần này thông qua tên của nó.

Phần quan trọng thứ hai cần tìm hiểu là cách sử dụng Camera trong ứng dụng. Hầu hết các loại điện thoại thông minh hiện nay đều được tích hợp phần cứng Camera trong thiết bị. Và Camera trở thành phần không thể thiếu trong các hệ điều hành cho di động. Android không phải là một ngoại lệ, và trong thư viện các hàm ***API*** được hỗ trợ sẵn trong ***Android SDK*** thì Android đã cung cấp cho ta lớp tiện ích để có thể truy xuất và điều khiển Camera trên thiết bị có hỗ trợ. Để có thể thực hiện được điều này, ta sử dụng lớp ***Camera*** cùng với lớp ***SurfaceView()***.

Lớp ***Camera*** cung cấp các phương thức để có thể thay đổi các thông số thiết lập trên Camera, xem trước ảnh và đặc biệt là ghi nhận hình ảnh từ ống kính Camera của

điện thoại. Trước khi có thể sử dụng được Camera trên thiết bị ta cần phải thiết lập quyền để sử dụng các phần cứng trong thiết bị và trường hợp này là Camera, các quyền được thiết lập sẽ được đặt trong tập tin *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.CAMERA" />
```

Sau đó ta tạo ra lớp *SurfaceView* để hiển thị hình ảnh trực tiếp qua ống kính Camera điện thoại. Lớp này nắm giữ phần hiển thị hình ảnh và chịu trách nhiệm vẽ lại trên diện tích của màn hình. Ta sử dụng lớp interface *SurfaceHolder* để truy cập phần bì mặt bên dưới lớp *SurfaceView*. Ta thực thi lại các hàm trong interface *SurfaceHolder.Callback* và thêm các hàm callback này trong *SurfaceHolder*. Các phương thức được thực thi lại trong interface *SurfaceHolder.Callback* bao gồm ba phương thức sau:

```
public void surfaceChanged(SurfaceHolder arg0, int arg1, int
arg2, int arg3);
public void surfaceCreated(SurfaceHolder arg0);
public void surfaceDestroyed(SurfaceHolder arg0);
```

Hàm *surfaceCreated* được gọi sau khi mà một Surface đã được tạo ra, hàm *surfaceChanged* được gọi sau khi có bất kỳ sự thay đổi nào xảy ra trên Surface và hàm *surfaceDestroyed* được gọi khi Surface bị hủy.

Sau khi đã tạo ra một Surface để thể hiện hình ảnh trên màn hình, ta bắt đầu sử dụng lớp *Camera*, gọi phương thức *Camera.open()* để mở ống kính máy ảnh trên điện thoại, sau đó ta thiết lập xem ảnh trực tiếp trên bì mặt thông qua hàm *setPreviewDisplay()*. Các hàm này được gọi lần đầu trong hàm callback *surfaceCreated()* khi khởi tạo các thông số ban đầu cho Camera.

```
public void surfaceCreated(SurfaceHolder arg0) {
    Camera mCamera = Camera.open();
    mCamera.setPreviewDisplay(mySurface_holder);
}
```

Sau đó ta sẽ thiết lập các thông số tùy chỉnh trong Camera thông qua đối tượng *Camera.parameters* của lớp *Camera*. Gọi hàm *camera.getParameters()* để lấy các thông số được thiết lập hiện tại trong máy ảnh điện thoại. Ta có thể thiết lập lại các thông số trong Camera bằng gọi các phương thức có dạng *set...()* trong đối tượng *Parameters*

và hoàn tất việc thay đổi giá trị các thông số dùng phương thức *setParameters*. Ta cần thiết lập lại các thông số của Camera khi hàm callback *surfaceChanged()* được gọi. Sau khi thiết lập lại thông số cho Camera, gọi phương thức *startPreview()* để bắt đầu chế độ xem trước hình ảnh trực tiếp qua ống kính của Camera điện thoại.

```
public void surfaceChanged(SurfaceHolder arg0, int arg1, int
arg2, int arg3) {
    Camera.Parameters params = mCamera.getParameters();

    // Thực hiện việc thay đổi các thông số ở đây

    // hoàn tất việc thay đổi các thông số
    mCamera.setParameters(params);
    mCamera.startPreview();
}
```

Sau khi kết thúc quá trình sử dụng Camera ta gọi phương thức *camera.release()* và dừng phát khung ảnh xem trước *stopPreview()*. Các phương thức này được sử dụng khi *surfaceDestroyed()* được gọi:

```
public void surfaceDestroyed(SurfaceHolder arg0) {
    // TODO Auto-generated method stub
    mCamera.release();
    mCamera.stopPreview();
}
```

Trong lớp *Camera* có hỗ trợ các lớp interface callback đến nhiều sự kiện khác nhau trong ứng dụng. Sau đây là các interface hỗ trợ việc gửi thông báo đến các sự kiện trong lớp *Camera*:

```
Camera.AutoFocusCallback
Camera.ErrorCallback
Camera.PictureCallback
Camera.PreviewCallback
Camera.ShutterCallback
```

Interface ***Camera.AutoFocusCallback*** được sử dụng để gửi thông báo khi quá trình lấy nét tự động (***auto focus***) hoàn tất. Tính năng lấy nét tự động thường được sử dụng trong Camera để tăng chất lượng ảnh và khiến ảnh chụp rõ vật thể hơn. Tuy nhiên không phải thiết bị nào có Camera cũng hỗ trợ tính năng này. Trong interface này, hàm ***onAutoFocus()*** là hàm thuần ảo và ta cần thực thi hàm này trong chương trình. Hàm này sẽ được gọi khi quá trình lấy nét tự động hoàn tất. Và để bắt đầu cho Camera thực hiện việc lấy nét tự động ta sử dụng phương thức ***camera.autoFocus()***.

Interface ***Camera.ErrorCallback*** để báo hiệu khi có lỗi xảy ra. Phương thức chính là ***onError()*** sẽ được gọi khi có xảy ra lỗi trong việc thao tác với Camera.

Interface ***Camera.PictureCallback*** được sử dụng để cung cấp dữ liệu ảnh sau khi ảnh đã được chụp. Phương thức chính ***onPictureTaken()*** được gọi khi dữ liệu đã sẵn sàng. Định dạng của ảnh phụ thuộc vào định dạng ảnh của Camera và có thể được thiết lập thông qua đối tượng ***Camera.Parameters***.

Interface ***Camera.PreviewCallback*** được sử dụng khi cung cấp dữ liệu của khung hình xem trước. Phương thức chính là ***onPreviewFrame()*** được sử dụng khi khung duyệt trước ảnh đã có dữ liệu. Định dạng của dữ liệu cũng phụ thuộc vào định dạng hiện tại của Camera.

Cuối cùng là interface ***Camera.ShutterCallback*** để thông báo khi ảnh đã được chụp xong từ Camera điện thoại. Phương thức chính là ***onShutter()***.

Trên đây là các tìm hiểu cần thiết về việc lập trình trên hệ điều hành Android nhằm xây dựng ứng dụng Dịch tự động văn bản phục vụ cho bài toán đặt ra, những chức năng không được sử dụng trong ứng dụng chúng em xin phép không đề cập đến nhằm tiết kiệm thời gian cũng như giúp bài báo cáo ngắn gọn hơn.

NHẬN DẠNG KÝ TỰ QUANG HỌC

1. Giới thiệu chung

1.1. Sơ lược về nhận dạng ký tự quang học – OCR

Nhận dạng ký tự quang học (tên tiếng anh là **Optical Character Recognition – OCR**) là một quá trình thực hiện việc chuyển đổi từ dạng hình ảnh của chữ viết in hoặc các ký hiệu sang các dạng văn bản tài liệu hoặc thông tin có thể chỉnh sửa trên máy tính. Đầu vào của quá trình này là tập tin hình ảnh và đầu ra sẽ là các tập tin văn bản chứa nội dung là các chữ viết có trong hình ảnh đó. Nhận dạng ký tự quang học được hình thành từ các lĩnh vực nghiên cứu nhận dạng mẫu, trí tuệ nhân tạo và thị giác máy tính. Ngày nay kỹ thuật nhận dạng ký tự quang học đã được sử dụng rộng rãi và ứng dụng nhiều trong thực tế song song với việc nghiên cứu về lý thuyết để cải tiến kết quả nhận dạng.

Thông thường, các hệ thống nhận dạng ký tự quang học được sử dụng dưới dạng các phần mềm trong máy tính hoặc tích hợp trong máy in, máy quét để thực hiện việc nhận dạng ký tự. Ví dụ thường thấy nhất là quét các hình ảnh văn bản thành các văn bản tài liệu lưu trên máy tính.

**Radio Listeners in Panic,
Taking War Drama as Fact** Hình Ánh



Radio Listeners in Panic, Taking War Drama as Fact Kết Quả

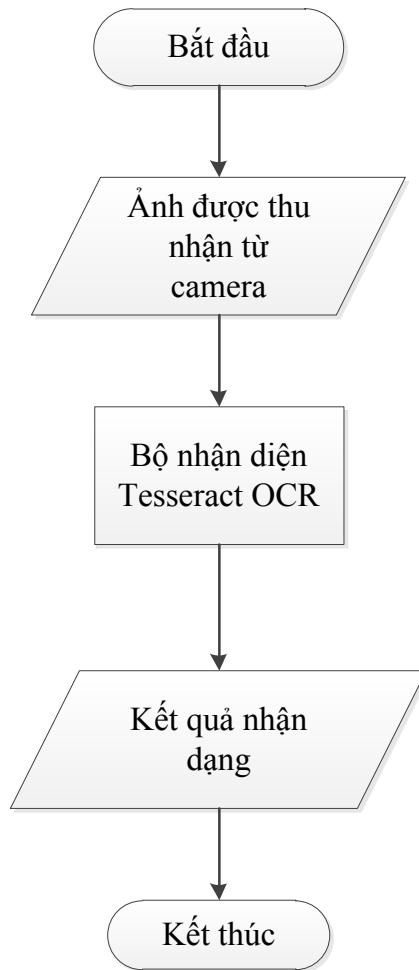
Hình 12: Quá trình thực hiện OCR

1.2. Các phương pháp áp dụng OCR

Bài báo cáo này tập trung vào việc sử dụng kỹ thuật nhận dạng ký tự quang học áp dụng trên điện thoại Android để thực hiện việc chuyển hình ảnh sang văn bản qua camera của điện thoại. Sau đây là các phương pháp có thể áp dụng kỹ thuật nhận dạng ký tự quang học:

- **Nghiên cứu và tự xây dựng một bộ nhận dạng ký tự quang học:** đây là cách khó khăn khi thực hiện vì hiện nay trên thế giới đã có nhiều hướng nghiên cứu về lĩnh vực này và cho ra đời nhiều phương pháp nhận dạng ký tự quang học. Tự viết lại bộ nhận dạng ký tự quang học sẽ tốn khá nhiều thời gian mà hiệu quả sẽ không được cao. Bài báo cáo này chủ yếu tập trung vào việc sử dụng nhận dạng ký tự quang học để thực hiện chuyển hình ảnh sang văn bản nên cách này sẽ không khả thi và bẩn thân việc nghiên cứu các kỹ thuật nhận dạng ký tự quang học đã là một đề tài lớn nên chúng em sẽ không chọn phương pháp này để thực hiện nhận dạng ký tự quang học.
- **Sử dụng các bộ nhận dạng ký tự quang học trực tiếp trên web thông qua môi trường mạng:** điện thoại sẽ gửi hình ảnh lên máy chủ web để máy chủ sẽ trực tiếp xử lý áp dụng các thuật toán nhận dạng ký tự quang học được cài đặt sẵn để xử lý, phân tích bức ảnh và gửi trả kết quả đã được nhận dạng về cho điện thoại. Cách này có ưu điểm là dễ thực hiện và độ chính xác có thể cao tuy nhiên khi sử dụng chương trình đòi hỏi người sử dụng phải cài đặt mạng điện thoại hoặc wifi trong máy để kết nối mạng internet cho việc truyền và nhận dữ liệu từ máy chủ trên web. Chưa kể đến việc xử lý và chờ kết quả từ máy chủ trên mạng sẽ khá lâu gây bất tiện cho người sử dụng chương trình. Nếu điện thoại không có kết nối mạng thì người dùng sẽ không thể sử dụng được tính năng nhận diện hình ảnh.
- **Sử dụng các bộ thư viện nhận dạng ký tự quang học có sẵn:** so với hai cách được nêu ra ở trên thì cách này có ưu điểm là thực hiện không quá khó khăn, chỉ cần cài đặt và biên dịch để chạy trên môi trường Android vừa khắc phục được nhược điểm là ứng dụng không phụ thuộc vào môi trường mạng, có thể chạy độc lập, tiết

kiệm nhiều thời gian vì việc xử lý trên khối nhận dạng được thực hiện hoàn toàn trực tiếp trên điện thoại.



Hình 13: Sơ đồ khối nhận diện ký tự quang học trong chương trình

1.3. So sánh các thư viện công cụ nhận dạng ký tự quang học

Hiện nay trên thế giới đã có khá nhiều bộ thư viện nhận dạng ký tự quang học với độ chính xác khá cao. Sử dụng một trong các thư viện đó sẽ giúp chúng ta tiết kiệm khá nhiều công sức. Sau đây là một số bộ và phần mềm nhận dạng ký tự quang học miễn phí được sử dụng rộng rãi hiện nay:

- **Tesseract OCR¹**: là bộ nhận dạng ký tự quang học thương mại ban đầu được phát triển tại công ty **HP (Hewlett-Packard)** trong khoảng 1985 – 1995 và được giải thưởng top 3 phần mềm nhận dạng ký tự quang học chính xác nhất trong hội nghị thường niên của **UNLV (University of Nevada-Las Vegas)**. Sau đó bộ nhận dạng này được chuyển thành mã nguồn mở trên **Google** và tiếp tục được phát triển cho đến ngày nay với sự đóng góp của nhiều lập trình viên chuyên nghiệp. Trưởng bộ phận của dự án hiện nay là **Ray Smith**.
- **GOCR²**: Là một chương trình nhận dạng ký tự quang học được phát triển dưới dạng giấy phép công cộng **GNU** và được bắt đầu bởi **Joerg Schulenberg** vào năm 2000.
- **FreeOCR³**: Được xem là một trong các phần mềm nhận dạng ký tự quang học chính xác nhất vì sử dụng bộ engine **Tesseract** của **HP**. Ngoài ra, **FreeOCR** còn cung cấp dịch vụ nhận dạng ký tự quang học trực tuyến trên web.
- **JavaOCR⁴**: Là phần mềm nhận dạng ký tự quang học được viết hoàn toàn bằng thư viện **Java** cho việc xử lý ảnh và nhận dạng ký tự. Ưu điểm của chương trình này là chiếm ít tài nguyên bộ nhớ, dễ thực hiện trên các môi trường di động hạn chế về bộ nhớ và chỉ sử dụng được ngôn ngữ **Java**.

1.4. Kết luận

Trong các thư viện nhận dạng ký tự quang học trên thì bộ nhận dạng **Tesseract OCR** nổi trội nhất với các ưu điểm sau:

- Có lịch sử phát triển lâu dài và mang độ chính xác cao ngay từ khi mới ra mắt.
- Khả năng mở rộng và tùy biến cao đồng thời được **Google** tài trợ và đồng đảo các nhà phát triển tham gia đóng góp cho **Tesseract**.

¹ <https://code.google.com/p/tesseract-ocr/>

² <http://jocr.sourceforge.net/>

³ <http://www.free-ocr.com/>

⁴ <http://sourceforge.net/projects/javaocr/>

- Phiên bản được cập nhật thường xuyên, hỗ trợ ngày càng nhiều ngôn ngữ, có khả năng huấn luyện trên các ngôn ngữ mới và nhiều loại font chữ khác nhau.
- Một số phần mềm **OCR** hiện nay đều sử dụng bộ nhận dạng này cho việc nhận dạng ký tự nên **Tesseract** đã trở nên phổ biến hơn, đồng thời khả năng hỗ trợ trên nhiều môi trường, nền tảng khác nhau từ máy tính cho đến các thiết bị di động.

Chính vì các ưu điểm nêu trên mà trong bài báo cáo này nhóm chúng em sẽ sử dụng **Tesseract** để thực hiện quá trình nhận dạng ký tự trong chương trình.

2. Giới thiệu về bộ nhận dạng ký tự quang học **Tesseract**

2.1. Lịch sử

Tesseract[3] là một phần mềm mã nguồn mở và ban đầu nó được nghiên cứu và phát triển tại hãng **Hewlett Packet (HP)** trong khoảng từ năm 1984 đến 1994. Vào năm 1995, **Tesseract** nằm trong nhóm ba bộ nhận dạng **OCR** đứng đầu về độ chính xác khi tham gia trong hội nghị thường niên của tổ chức **UNLV**.

Lúc mới khởi động thì **Tesseract** là một dự án nghiên cứu tiền sỹ tại phòng thí nghiệm **HP** ở **Bristol** và đã được tích hợp vào trong các dòng máy quét dạng phẳng của hãng dưới dạng các **add-on** phần cứng hoặc phần mềm. Nhưng thực tế dự án này đã thất bại ngay từ trong trứng nước vì nó chỉ làm việc hiệu quả trên các tài liệu in có chất lượng tốt.

Sau đó, dự án này cùng với sự cộng tác của bộ phận máy quét **HP** ở bang **Colorado** đã đạt được một bước tiến quan trọng về độ chuẩn xác khi nhận dạng và vượt lên nhiều bộ nhận dạng **OCR** thời đó nhưng dự án đã không thể trở thành sản phẩm hoàn chỉnh vì độ công kẽm và phức tạp. Sau đó, dự án được đưa về phòng thí nghiệm của **HP** để nghiên cứu về cách thức nén và tối ưu mã nguồn. Dự án tập trung cải thiện hiệu năng làm việc của **Tesseract** dựa trên độ chính xác đã có. Dự án này được hoàn tất vào cuối năm 1994 và sau đó vào năm 1995 bộ **Tesseract** được gửi đi tham dự hội nghị **UNLV** thường niên về độ chính xác của **OCR**, vượt trội hơn hẳn so với các phần mềm **OCR** lúc bấy giờ. Tuy nhiên, **Tesseract** đã không thể trở thành một sản phẩm thương mại hoàn chỉnh được và vào năm 2005, **HP** đã chuyển **Tesseract** sang mã nguồn mở và được hãng

Google tài trợ. Tesseract cho đến nay vẫn được nhiều nhà phát triển cộng tác và tiếp tục hoàn thiện. Phiên bản mới nhất hiện nay của bộ nhận dạng ký tự quang học **Tesseract** là phiên bản 3.0.3.

Vì **Tesseract** hiện nay là bộ thư viện mã nguồn mở hoàn toàn miễn phí nên trên thế giới đã có nhiều phần mềm nhận dạng ký tự quang học ra đời dựa trên bộ **Tesseract** ban đầu như: *VietOCR*⁵ cho nhận dạng tiếng Việt, *Tessenet2*⁶ bộ nhận diện **Tesseract** trên nền .Net của Microsoft, giao diện Java (*Java GUI front end*) cho **Tesseract**...

2.2. Kiến trúc hoạt động

Đầu tiên, bộ nhận diện **Tesseract** sẽ nhận đầu vào là ảnh màu hoặc ảnh mức xám. Ảnh này sẽ được chuyển đổi thành bộ phận phân tích ngưỡng thích ứng[4] (*adaptive thresholding*) để cho ra ảnh nhị phân. Vì trước kia HP cũng đã phát triển bộ phận phân tích bộ cục trang nên **Tesseract** không cần phải có thành phần đó và được thừa hưởng từ **HP**. Vì thế mà **Tesseract** nhận đầu vào là một ảnh nhị phân với các vùng đa giác tùy chọn đã được xác định.

Ban đầu, **Tesseract** được thiết kế làm việc trên ảnh nhị phân sau đó chương trình được cải tiến để có thể nhận dạng cả ảnh màu và ảnh mức xám. Chính vì thế mà cần bộ phận phân tích ngưỡng thích ứng để chuyển đổi ảnh màu, ảnh mức xám sang ảnh nhị phân.

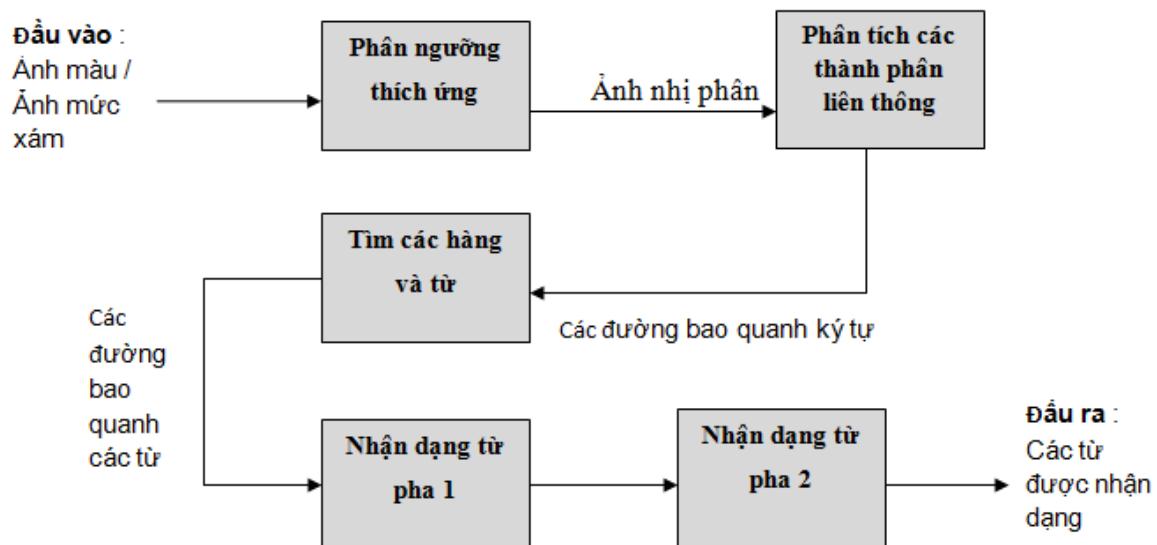
Sau đó quá trình nhận dạng sẽ được thực hiện tuần tự theo từng bước.

- Bước đầu tiên là phân tích các thành phần liên thông. Kết quả của bước này sẽ là tạo ra các đường bao quanh các ký tự.
- Bước thứ hai là tìm hàng và tìm từ, kết quả của bước này cũng giống như bước trên sẽ tạo ra các vùng bao quanh các hàng chữ và ký tự chứa trong vùng văn bản.
- Bước tiếp theo sẽ là nhận dạng từ. Công đoạn nhận dạng từ sẽ được xử lý qua hai giai đoạn. Giai đoạn đầu (pha 1) sẽ là nhận dạng các từ theo lượt. Các từ thỏa yêu

⁵ http://vietocr.sourceforge.net/usage_vi.html

⁶ <http://www.pixel-technology.com/freeware/tessnet2/>

cầu trong giai đoạn này sẽ được chuyển sang bộ phân loại thích ứng (*adaptive classifier*) để làm dữ liệu huấn luyện. Chính nhờ đó mà bộ phân loại thích ứng sẽ có khả năng nhận diện được chính xác hơn ở phần sau của trang. Sau khi bộ phân loại thích ứng đã học được các thông tin có ích từ giai đoạn đầu khi nhận dạng phần trên của trang thì giai đoạn thứ hai (pha 2) của việc nhận dạng sẽ được thực hiện. Giai đoạn này sẽ quét hết toàn bộ trang, các từ không được nhận diện chính xác ở giai đoạn đầu sẽ được nhận diện lại lần nữa. Cuối cùng bộ nhận diện sẽ tổng hợp lại các thông tin ở trên và cho ra kết quả nhận diện hoàn chỉnh.



Hình 14: Kiến trúc tổng thể của Tesseract

2.3. Xác định dòng và từ

- **Xác định dòng**

Mục đích của bước này là nhận dạng các dòng của các hình ảnh bị nghiêng, giúp giảm sự mất thông tin khi nhận dạng ảnh nghiêng. Các bộ phận quan trọng của quá trình này là lọc dãy màu (còn được gọi là *blobs*) và xây dựng dòng. Bước này cũng giúp loại bỏ các văn bản có *drop-cap*.

- ***Thiết lập dòng cơ sở***

Khi dòng văn bản được tìm thấy, các dòng cơ sở được thiết lập chính xác hơn bằng cách sử dụng một đường có tên là ***spline*** toàn phương (là dòng mà được kết hợp từ nhiều đoạn). Nó giúp ***Tesseract*** xử lý các trang có đường cơ sở là đường cong.

Các dòng cơ sở được thiết lập bằng cách phân vùng các ***blobs*** thành các nhóm có thể thay thế thích hợp liên tục trong đường cơ sở thẳng ban đầu. Một ***spline*** toàn phương được thiết lập cho phân vùng dày đặc nhất (giả định là đường cơ sở) của một hình có phương ít nhất. ***Spline*** có lợi thế là tính toán ổn định, nhược điểm là sự gián đoạn có thể xảy ra khi nhiều phân đoạn ***spline*** được yêu cầu.

Volume 69, pages 872–879.

Hình 15: Đường cơ sở dạng cong

- ***Cắt nhỏ từ***

Tesseract sẽ xác định xem có các ký tự dính với nhau trong một từ hay không. Nếu có nó sẽ cắt nhỏ các ký tự ra thành các ký tự riêng lẻ.



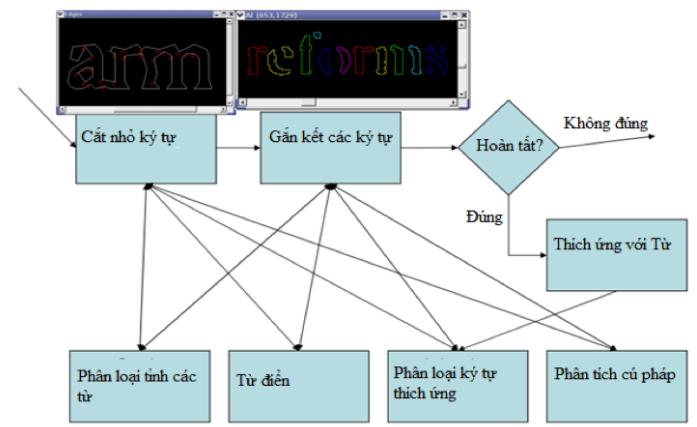
Hình 16: Cắt các ký tự bị dính

- ***Nhận dạng khoảng cách giữa chữ hoặc số***

Xác định khoảng cách giữa các số hoặc giữa các chữ là một vấn đề khá phức tạp. ***Tesseract*** giải quyết những vấn đề này bằng cách đo khoảng cách trong một phạm vi hạn chế theo chiều dọc giữa dòng cơ sở và dòng trung bình.

- **Nhận dạng từ**

Quá trình nhận dạng một từ là quá trình phân tích một từ được chia ra thành các ký tự như thế nào.



Hình 17: Quá trình nhận dạng từ

Khi kết quả xuất ra một từ mà nó không thỏa mãn nhu cầu thì **Tesseract** cố gắng cải thiện kết quả này bằng cách cắt nhỏ các từ có nghĩa không tốt nhất. Nếu việc cắt nhỏ không làm tăng chất lượng từ thì nó sẽ phục hồi lại từ trước đó.

2.4. Một số thử nghiệm

Một số kết quả đã được tiến hành thử nghiệm trên ba loại hình ảnh: Hình chụp từ chữ viết tay (18), hình chụp từ chữ đánh máy (19) và hình từ tập tin **pdf** (20).

Hình chữ viết tay

JUDAS
PRIEST
775758
HOLA
DIEGO
12312
367945

Hình 18: Hình chứa chữ viết tay

Kết quả

JUDA\$

PRIEST

775758

HOLA

DIEGO

12312

387945

Tỉ lệ sai: 1/33 chiếm 3,03%.

Hình chữ đánh máy

ESTA67

ES767

UNA4567

PRUEBA5887

Hình 19: Hình chúa ký tự đánh máy

Kết quả

STA67

ES767

UNA4567

PRUEBA5887

Tỉ lệ sai: 1/28 chiếm 3,57%

Hình ảnh tập tin.pdf

PREFACE

This book is now in its fifth edition. Each edition has corresponded to a different phase in the way computer networks were used. When the first edition appeared in 1980, networks were an academic curiosity. When the second edition appeared in 1988, networks were used by universities and large businesses. When the third edition appeared in 1996, computer networks, especially the Internet, had become a daily reality for millions of people. By the fourth edition, in 2003, wireless networks and mobile computers had become commonplace for accessing the Web and the Internet. Now, in the fifth edition, networks are about content distribution (especially videos using CDNs and peer-to-peer networks) and mobile phones are small computers on the Internet.

Hình 20: Hình dạng pdf

Kết quả

PREFACE

This book is now In "5 mm edllmn Eden edmon has cormsponded In a d|f—teaenr phase rn me way campllnt networks were used When the firs! edman ap peared in man. networks weae an academic cum: ly When me second edmorr appeared In 1933. networks were used by unlverslues and large businesses When lhe nrerrd ednmn appeared in 1995, compuler networks. especially lhe Inrer-rrer, had become a duly reamy rar mrnmna cl penplc By lhe rrnrnnr edllmn. in 2003. wu':— less nclwmks and mohllc compumeus had become commonplace for accessing the Web and me unerrrer. Now, In [he mun edllkm, networks are about content u1bullan(espeda.I|y videos using cum and pecuopccr networks) and mobile phones are small mmpulers on the xnerner.

Tỉ lệ sai trên 50% so với văn bản gốc. Văn bản càng dài độ chính xác càng giảm dần.

2.5. Kết luận

Công cụ **OCR** với mã nguồn mở **Tesseract** - dùng để nhận dạng kí tự trên một tập tin hình và chuyển kí tự thành văn bản. Bên cạnh những ưu điểm vượt trội của mình, **Tesseract** cũng có một số những hạn chế như nhầm lẫn giữa chữ hoa và chữ thường, nhầm lẫn giữa các kí tự có hình dáng tương tự, đúng từ nhưng sai trong ngữ cảnh.

XÂY DỰNG ỨNG DỤNG ANDROID

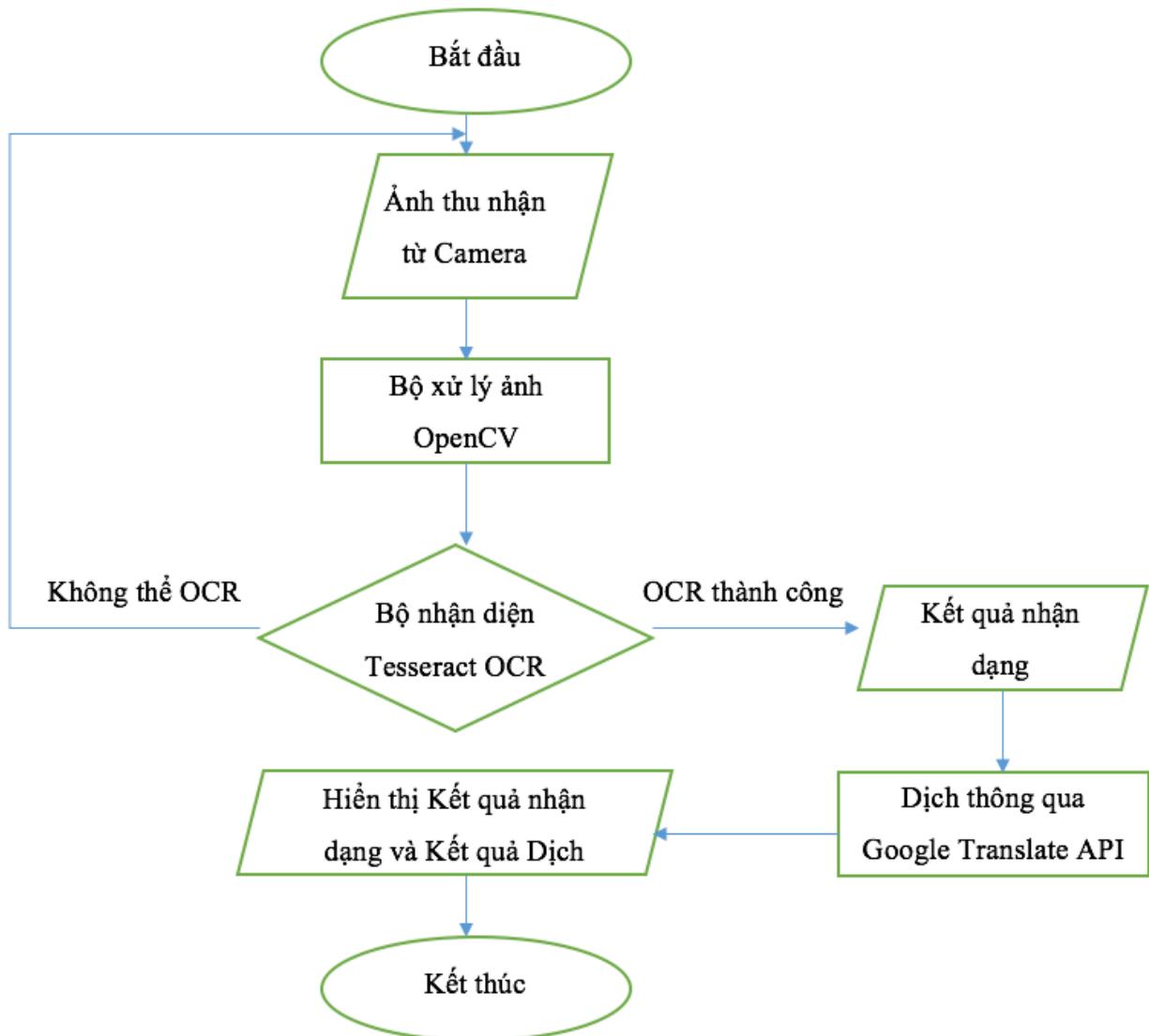
1. Mục tiêu

Xây dựng ứng dụng Android sử dụng thư viện **Tesseract** hỗ trợ chuyển đổi hình ảnh văn bản sang văn bản.

Xử lý hình ảnh bằng công cụ OpenCV (**Open Source Computer Vision**).

Dịch văn bản bằng dịch vụ **Google Translate** của Google thông qua các hàm **API**.

Sơ đồ khái niệm về hoạt động của ứng dụng



Hình 21: Sơ đồ khái niệm về hoạt động của ứng dụng

2. Cài đặt các thư viện hỗ trợ

2.1. Thư viện Tesseract

Thư viện mã nguồn của bộ **Tesseract** được viết bằng **C/C++** chuẩn nên để có thể sử dụng trên hệ điều hành **Android**, chúng ta cần biên dịch nó bằng cách sử dụng công cụ **Native Development Kit(NDK)**.

Tải bản fork của thư viện **Tesseract**⁷.

Sau khi tải về và giải nén ta được ba thư mục nhưng ta chỉ sử dụng thư mục **tess-two**, dùng **NDK** để build thư viện theo các dòng lệnh sau bằng **Terminal**:

```
cd tess-two
ndk-build
android update project --path .
ant release
```

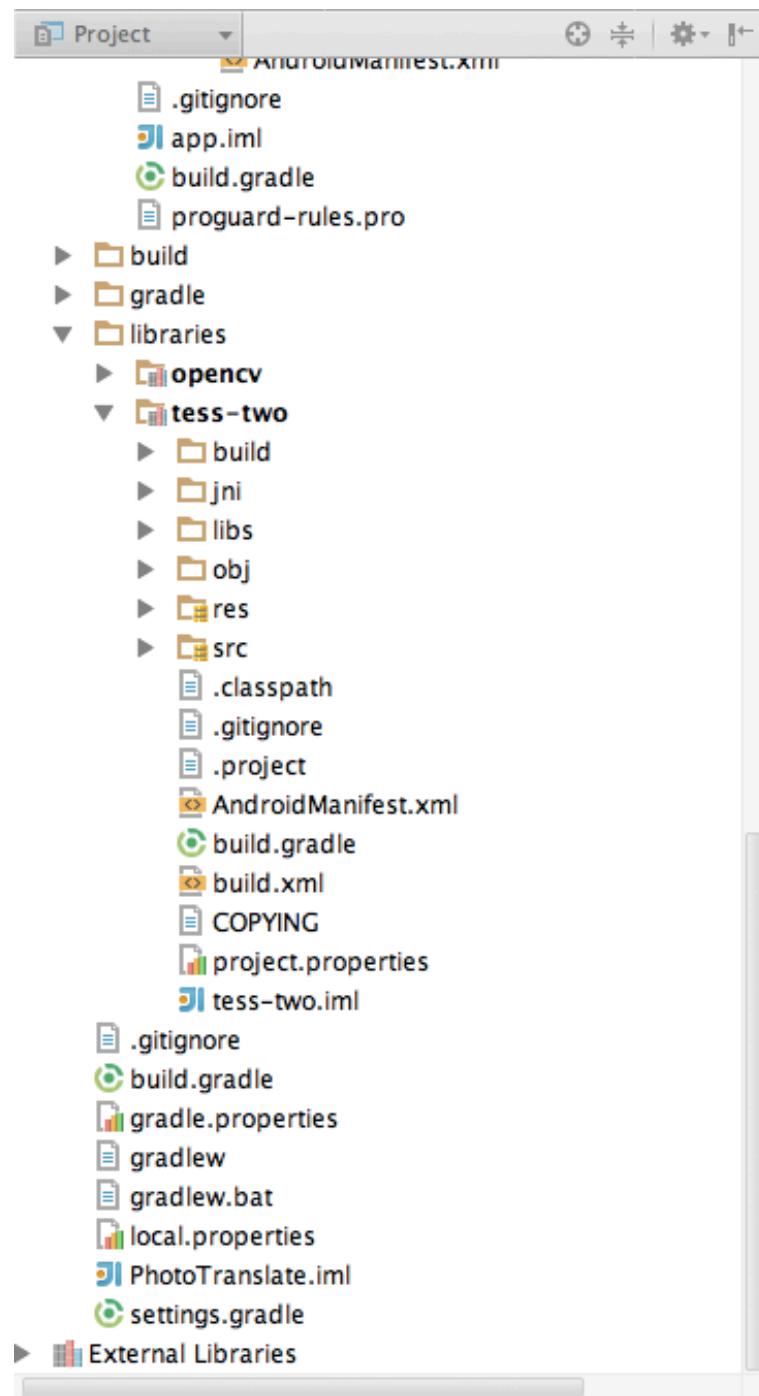
```
'Compile thumb : lept <= webpiostub.c
'Compile thumb : lept <= writefile.c
'Compile thumb : lept <= zlibmem.c
'Compile thumb : lept <= zlibmemstub.c
'Compile thumb : lept <= open_memstream.c
'Compile thumb : lept <= fopencookie.c
'Compile thumb : lept <= fmemopen.c
'Compile++ thumb : lept <= box.cpp
'Compile++ thumb : lept <= pix.cpp
'Compile++ thumb : lept <= pixa.cpp
'Compile++ thumb : lept <= utilities.cpp
'Compile++ thumb : lept <= readfile.cpp
'Compile++ thumb : lept <= writefile.cpp
```

Hình 22: quá trình build Tesseract bằng NDK

Quá trình build diễn ra khá lâu khoảng hơn 30 phút.

⁷ <https://github.com/rmtheis/tess-two>

Sau khi quá trình build hoàn tất, mở chương trình **IDE** (ở đây chúng em sử dụng **Android Studio**), chọn **project** và tạo thư mục tên **libraries** sau đó copy toàn bộ thư mục **tess-two** đã build ở trên vào.

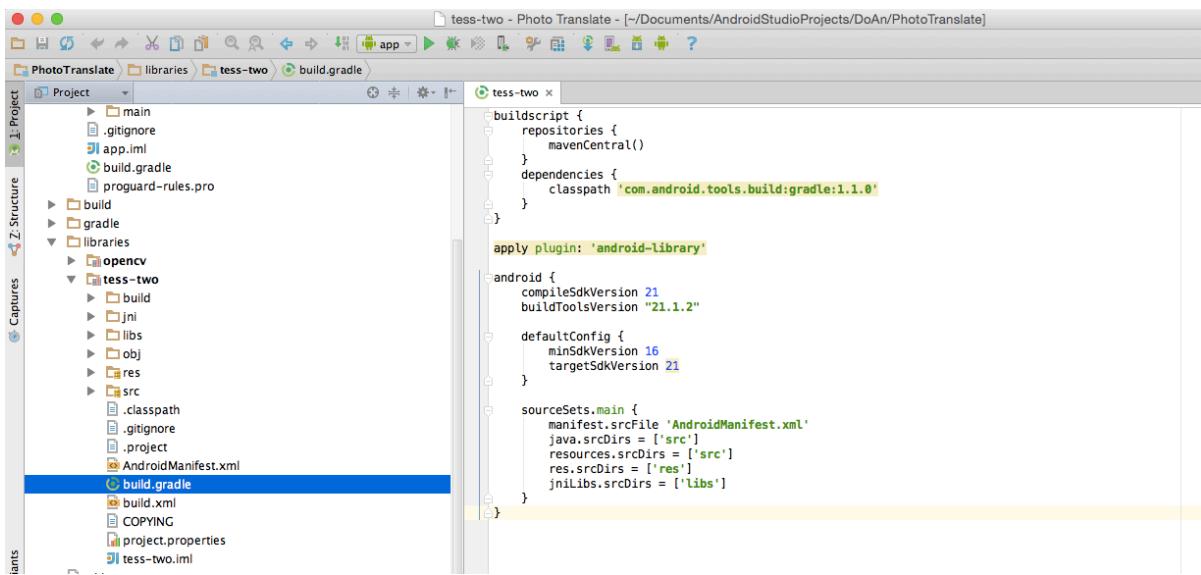


Hình 23: Import thư viện Tesseract

Tạo file ***build.gradle*** nằm trong thư mục ***tess-two*** với nội dung như sau:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.1.0'
    }
}
apply plugin: 'android-library'
android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"
    defaultConfig {
        minSdkVersion 16
        targetSdkVersion 21
    }

    sourceSets.main {
        manifest.srcFile 'AndroidManifest.xml'
        java.srcDirs = ['src']
        resources.srcDirs = ['src']
        res.srcDirs = ['res']
        jniLibs.srcDirs = ['libs']
    }
}
```



Hình 24: Nội dung file build.gradle

Sau khi hoàn tất, để gọi đến các hàm có sẵn trong thư viện **Tesseract** ta chú ý đến lớp **TessBaseAPI** trong thư mục **src** của project. Về bản chất, lớp này đóng vai trò là lớp thực thi lại các hàm trên **Tesseract**, các hàm trong lớp này được cài đặt bằng **Java** và trong thân các hàm này gọi lại các hàm **C/C++** bên dưới các tập tin thư viện.

Trong lớp **TessBaseAPI** ta lưu ý đến các hàm chính sau:

```
public boolean init(String datapath, String language)
public void setImage(Bitmap bmp)
public String getUTF8Text()
```

Hàm **init()** là hàm dùng để khởi tạo các dữ liệu ban đầu cho **Tesseract** nhận vào hai tham số: tham số **datapath** là chuỗi đường dẫn chỉ đến tập tin dữ liệu của bộ **Tesseract OCR** chứa trong thẻ nhớ **SD Card** của điện thoại, tham số thứ hai **language** chỉ định ngôn ngữ sẽ được nhận dạng.

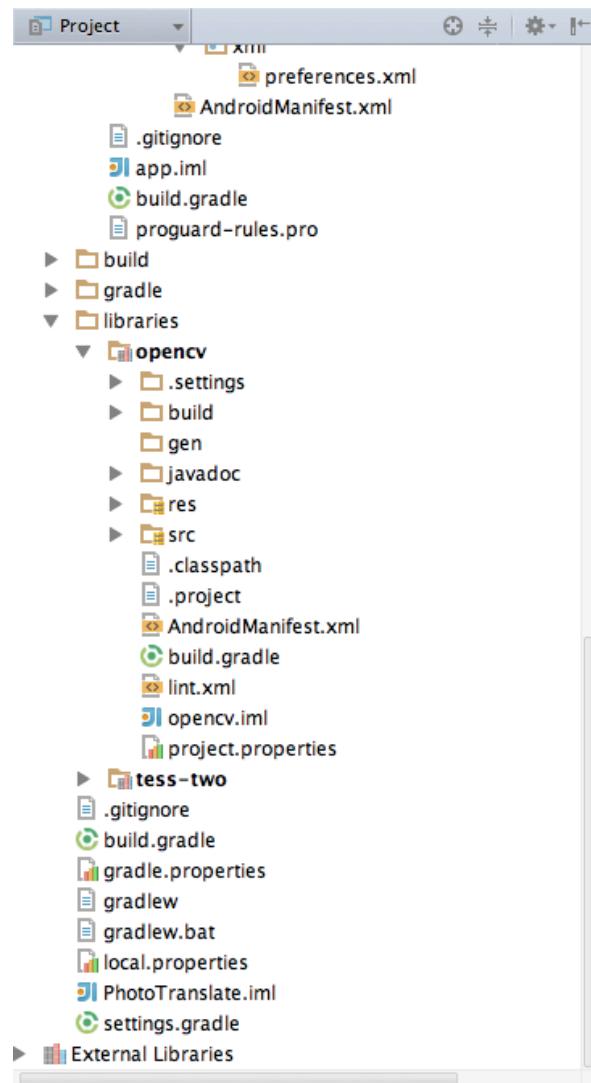
Hàm **SetImage()** để nhận vào hình ảnh dạng **bitmap**, hình ảnh này sẽ được nhận dạng sau đó thông qua hàm **getUTF8** và kết quả trả về của hàm này là một chuỗi kí tự có trong hình đó.

2.2. Thư viện *OpenCV*

OpenCV là công cụ hỗ trợ xử lý hình ảnh miễn phí được viết bằng *C/C++* hỗ trợ đa nền tảng trong đó có Android.

Tải thư viện *OpenCV*⁸ dành cho Android.

Sau khi tải về và giải nén, copy toàn bộ thư mục *opencv* trong thư mục *sdk* vào thư mục *libraries* của project.



Hình 25: Import thư viện *OpenCV*

⁸ <http://opencv.org/downloads.html>

Tạo file ***build.gradle*** nằm trong thư mục ***opencv*** với nội dung sau:

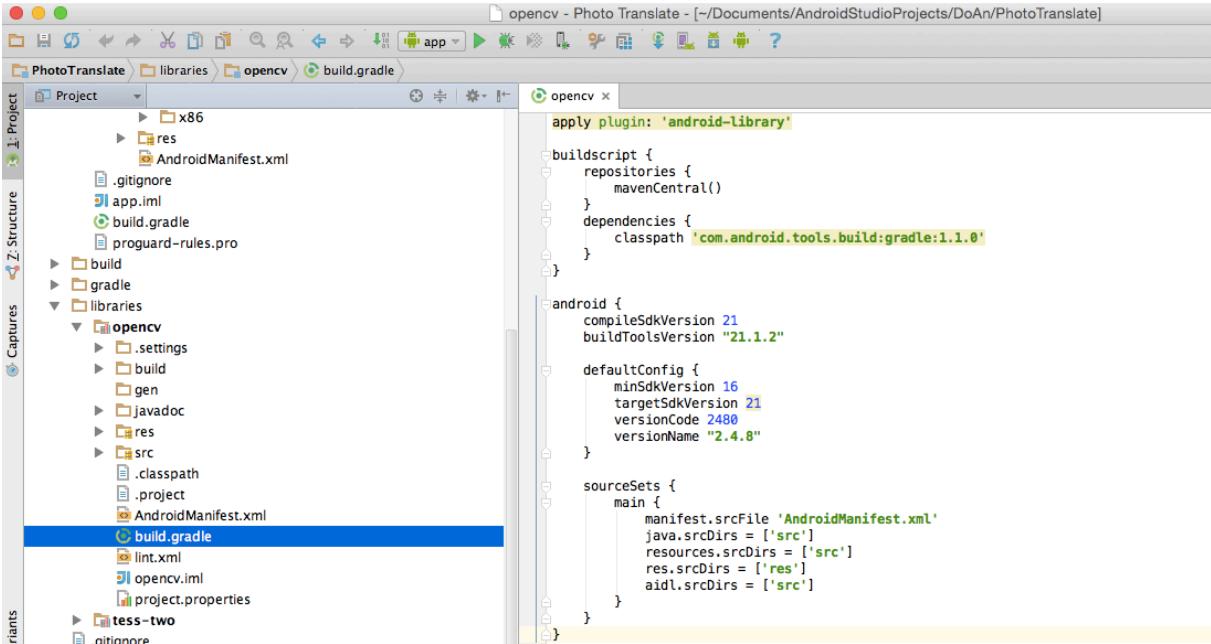
```
apply plugin: 'android-library'

buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.1.0'
    }
}

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"

    defaultConfig {
        minSdkVersion 16
        targetSdkVersion 21
        versionCode 2480
        versionName "2.4.8"
    }

    sourceSets {
        main {
            manifest.srcFile 'AndroidManifest.xml'
            java.srcDirs = ['src']
            resources.srcDirs = ['src']
            res.srcDirs = ['res']
            aidl.srcDirs = ['src']
        }
    }
}
```



```

apply plugin: 'android-library'

buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.1.0'
    }
}

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"

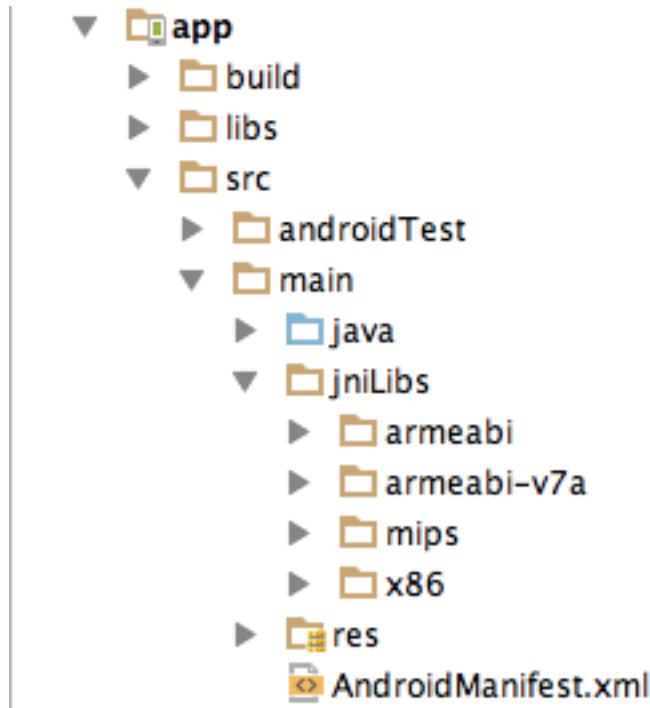
    defaultConfig {
        minSdkVersion 16
        targetSdkVersion 21
        versionCode 2480
        versionName "2.4.8"
    }

    sourceSets {
        main {
            manifest.srcFile 'AndroidManifest.xml'
            java.srcDirs = ['src']
            resources.srcDirs = ['src']
            res.srcDirs = ['res']
            aidl.srcDirs = ['src']
        }
    }
}

```

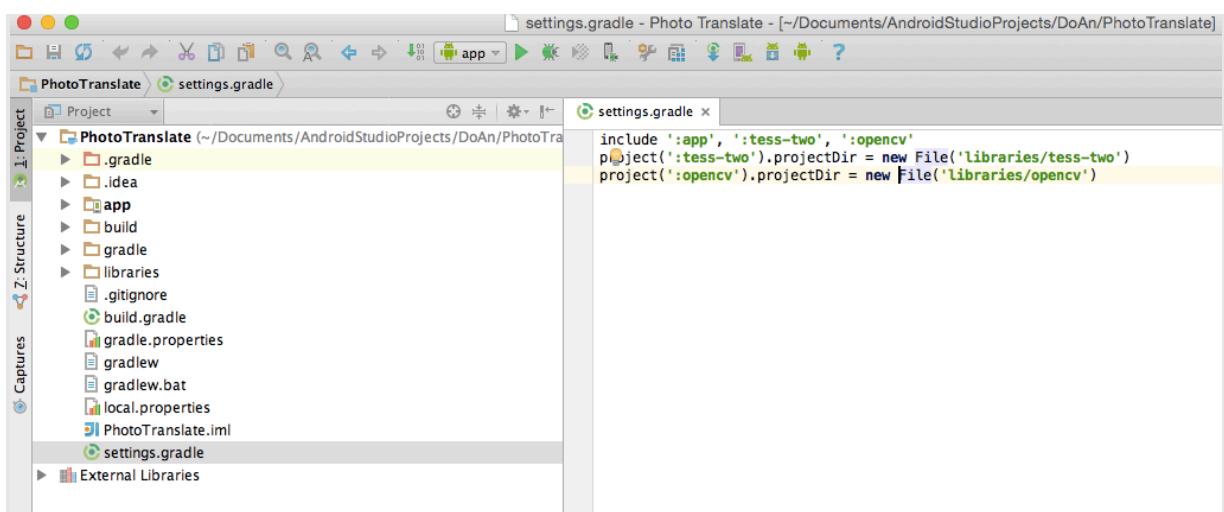
Hình 26: Nội dung file build.gradle

Tiếp tục copy thư mục **libs** nằm trong thư mục **native** vào thư mục **main** của project và đổi tên thành **jnilibs**.

**Hình 27: Thư mục jnilibs**

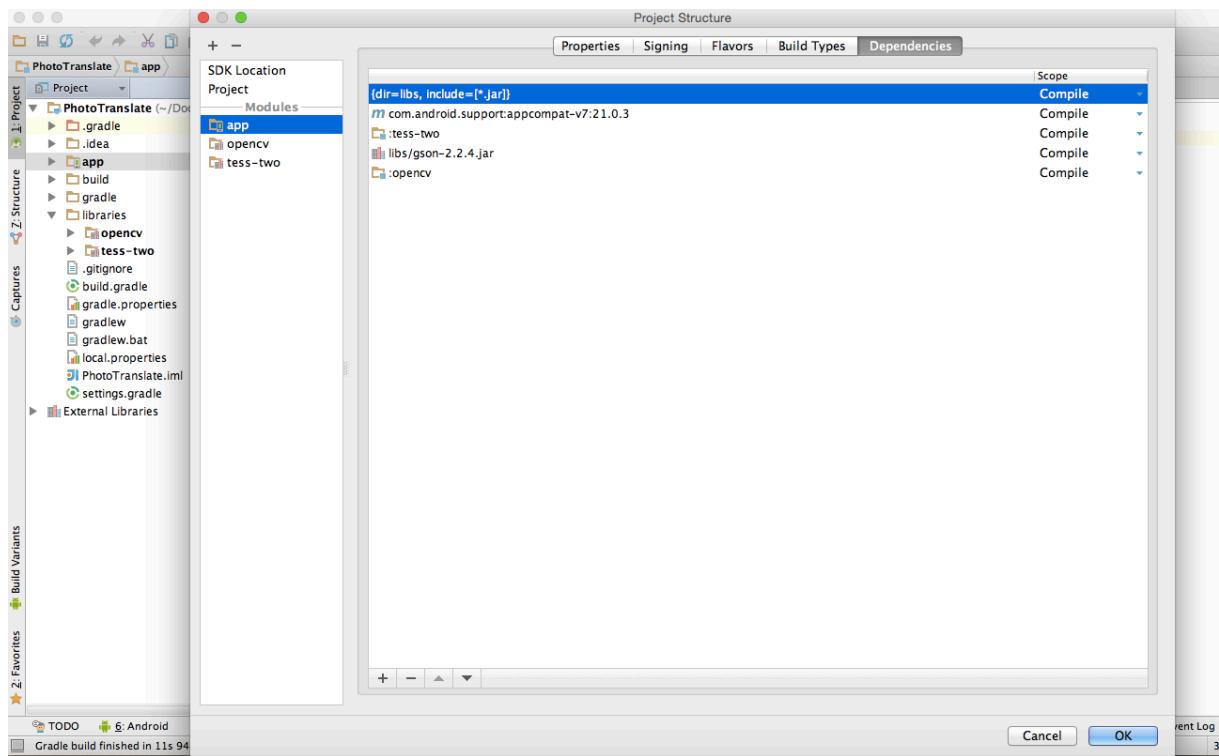
Sau khi hoàn tất, mở file *settings.gradle* và dán đoạn code sau vào:

```
include ':app', ':tess-two', ':opencv'  
project(':tess-two').projectDir = new File('libraries/tess-  
two')  
project(':opencv').projectDir = new File('libraries/opencv')
```



Hình 28: Nội dung file settings.gradle

Cuối cùng click phải vào thư mục *app* chọn ***Open Module Settings***, chọn thẻ ***Dependencies***. Click vào dấu + và chọn ***Module Dependency***, lần lượt chọn *tess-two* và *opencv* sau đó click ***OK***.



Hình 29: Add thư viện vào project

Sau khi hoàn tất tất cả các bước, hai thư mục *tess-two* và *opencv* sẽ được tô đậm lên tức là đã thành công.

OpenCV hỗ trợ xử lý nhiều loại hình ảnh cũng như rất nhiều chức năng nhưng trong yêu cầu của ứng dụng chúng em chỉ dùng đến class ***Imgproc***, trong class này chứa các hàm hỗ trợ việc ***Blur***, chuyển sang ảnh xám hay nhị phân hóa ảnh,...

Việc sử dụng *OpenCV* cũng khá đơn giản bằng cách khởi tạo ***OpenCV***

```
OpenCVLoader.initDebug()
```

Và một số hàm xử lý ảnh đơn giản mà chúng em áp dụng trong ứng dụng

```
mrgba = new Mat();
//convert bitmap to ARGB_8888
bitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);
Utils.bitmapToMat(bitmap, mrgba);
//set grayscale
Imgproc.cvtColor(mrgba, mrgba, Imgproc.COLOR_BGR2GRAY);
Size size = new Size(3,3);
//blur
Imgproc.GaussianBlur(mrgba, mrgba, size, 0);
Imgproc.threshold(mrgba, mrgba, 0, 255, Imgproc.THRESH_OTSU);
Imgproc.medianBlur(mrgba, mrgba, 3);
//Adaptive Threshold
Imgproc.threshold(mrgba, mrgba, 0, 255, Imgproc.THRESH_OTSU);
Imgproc.adaptiveThreshold(mrgba, mrgba, 255,
Imgproc.ADAPTIVE_THRESH_MEAN_C, Imgproc.THRESH_BINARY_INV, 61,
15);
Utils.matToBitmap(mrgba, bitmap, false);
```

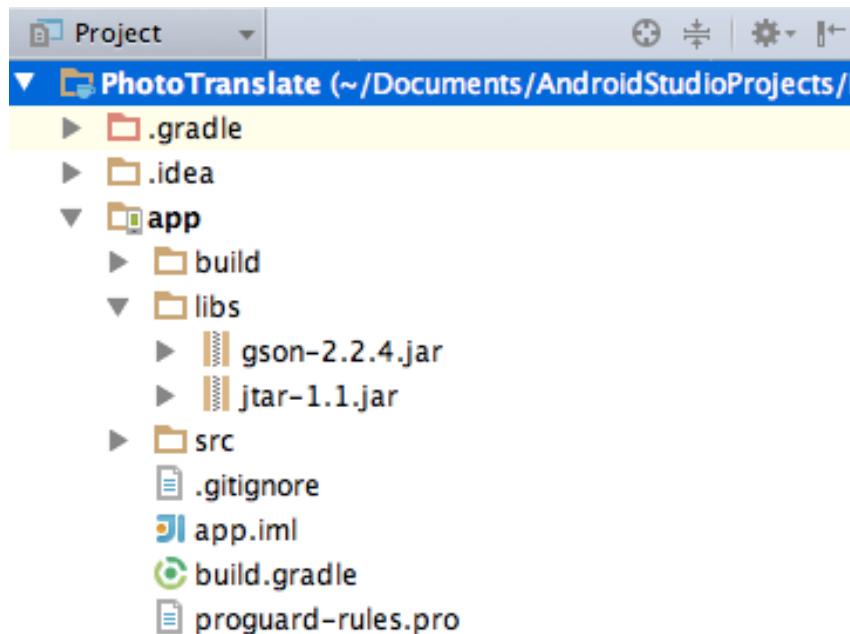
Xử lý hình ảnh trước khi đưa vào chuyển đổi văn bản hình ảnh sang văn bản bằng công cụ **Tesseract OCR** nhằm nâng cao chất lượng hình ảnh đầu vào để góp phần nâng cao kết quả nhận được, loại bỏ một phần các chi tiết thừa trong ảnh và giúp ảnh sáng hơn.

2.3. Thư viện **JSON** và **JTAR**

Tải thư viện **JSON**⁹ và **JTAR**¹⁰

Giải nén và copy file **.jar** vào thư mục **libs** trong project, click phải chọn **Use as library** sau đó click **Sync Now**.

Sau khi hoàn tất việc **Import** các gói thư viện vào **project** ta được **project** như sau:



Hình 30: Import thư viện **JSON** và **JTAR**

⁹ <https://code.google.com/p/google-gson/>

¹⁰ <https://code.google.com/p/jtar/>

Thư viện **JSON** hỗ trợ cho việc sử dụng dịch vụ **Google Translate** bằng việc hỗ trợ giao thức **JSON** để gửi dữ liệu đến **Google Translate** và nhận kết quả dịch.

```
JsonParser parser = new JsonParser();

JsonElement element = parser.parse(result.toString());

if (element.isJsonObject()){
    JsonObject obj = element.getAsJsonObject();
    if (obj.get("error") == null){
        String translatedText =
obj.get("data").getAsJsonObject().get("translations").getAsJsonArray().get(0).getAsJsonObject().get("translatedText").getAsString();
        return translatedText;
    }
}
```

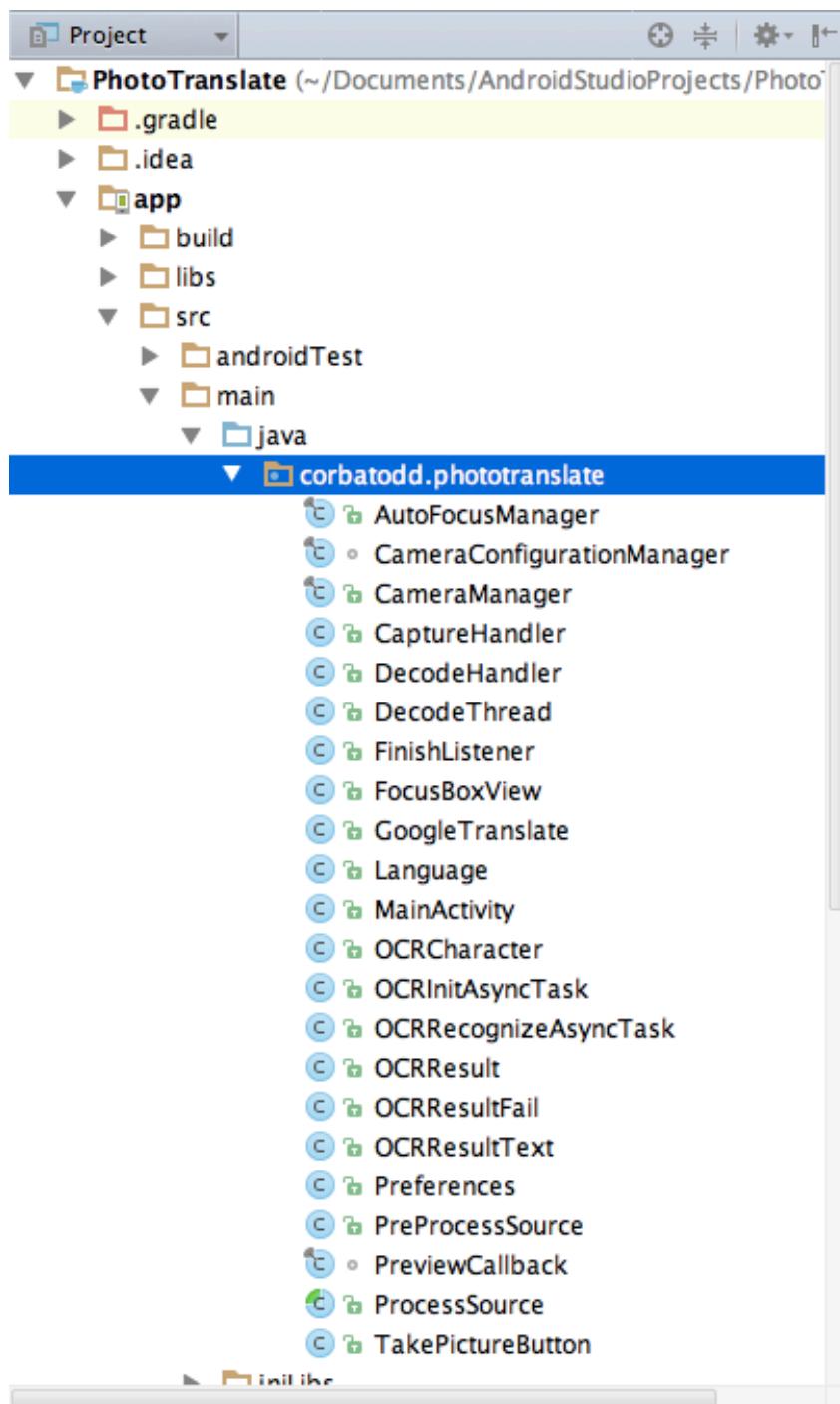
Thư viện **JTAR** hỗ trợ việc giải nén file **tar** phục vụ việc giải nén các gói ngôn ngữ mà **Tesseract** hiện đang hỗ trợ.

```
/*
Untar a tar file into the given directory
@param tarFile The tar file to be untarred
@param destinationDir The directory to untar into
*/
private void untar(File tarFile, File destinationDir)
```

Việc sử dụng các gói thư viện hỗ trợ giúp cho việc thực thi ứng dụng nhanh hơn cũng như giảm thiểu các sai sót trong quá trình lập trình.

3. Xây dựng ứng dụng

Ứng dụng gồm các *class* đảm nhiệm các vai trò khác nhau và hỗ trợ lẫn nhau trong chương trình.



Hình 31: Các class chính

Các chức năng khác nhau được phân ra các class khác nhau nhằm đơn giản hóa các đoạn code giúp dễ dàng hiệu chỉnh cũng như thêm vào.

Ngoài các chức năng về giao diện hiển thị đơn giản trên thì giao diện Camera rất quan trọng vì giao diện Camera là giao diện chính của ứng dụng, nó được thiết kế khá đặc biệt gồm có hai lớp, lớp thứ nhất hiển thị giao diện Camera, lớp thứ hai hiển thị các **Control** (ở đây là **Button Take Picture** và Khung giới hạn vùng chụp (**CropBox**)). Có hai vấn đề xảy ra cần giải quyết với hai lớp này: hiển thị và nhận sự kiện tương tác.

Đối với vấn đề hiển thị cho hai lớp nằm chồng lên nhau, thì trong Android cho phép chúng ta thiết kế layout như vậy nhờ vào **FrameLayout**. Như vậy các control được chia thành hai nhóm để hiển thị trên hai lớp này. Nhóm một là: **SurfaceView** dùng hiển thị camera. Nhóm hai gồm có khung giới hạn vùng chụp và **Button** chụp ảnh.

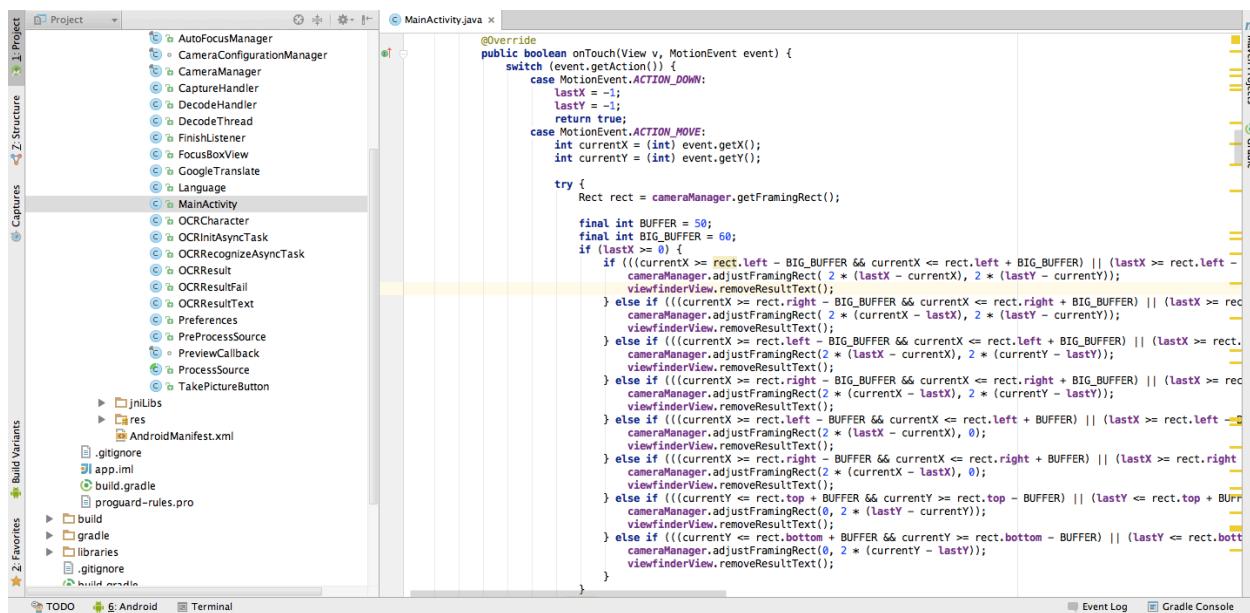
Đối với vấn đề nhận sự kiện tương tác, do ứng dụng chúng ta cần vẽ một khung hình giới hạn vùng chụp. Khung chữ nhật trên màn hình chính dùng để giới hạn vùng chụp có thể thay đổi kích thước cho phù hợp với kích thước chữ thực tế để đảm bảo độ chính xác. Trong ứng dụng lớp **RectView** được tạo ra dùng để quản lý công việc này. Để vẽ hình chữ nhật trên màn hình chúng em sử dụng phương thức **onDraw**, lớp **Paint** và **Canvas** để vẽ. Để bắt sự kiện tùy chỉnh kích cỡ cho khung giới hạn chúng em sử dụng phương thức **onTouch**.

Phương thức **onDraw** được sử dụng trong class **FocusBoxView** để vẽ khung

```
public void onDraw(Canvas canvas) {  
    paint.setColor(frameColor);  
    canvas.drawRect(frame.left, frame.top, frame.right + 1, frame.top  
    + 2, paint);  
}
```

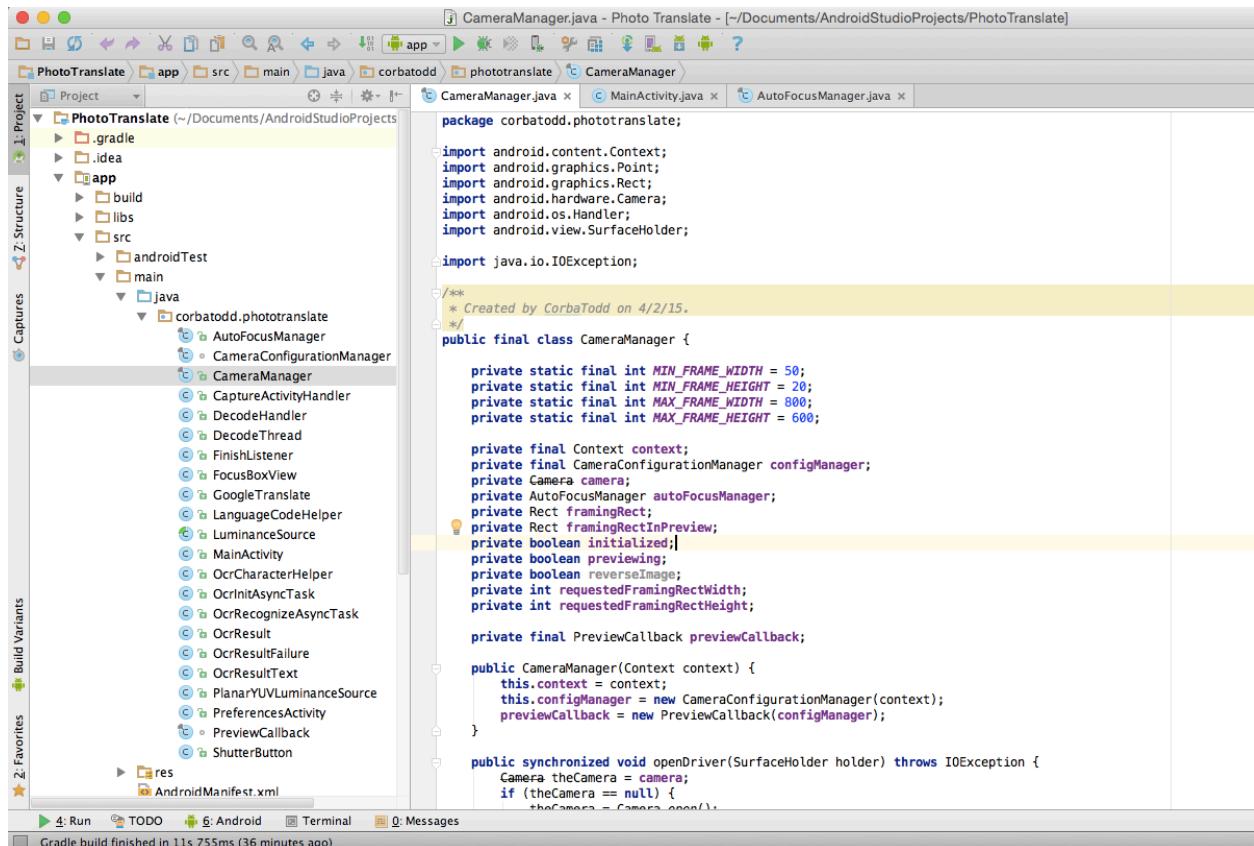
Phương thức ***onTouch*** được sử dụng trong class ***MainActivity*** để bắt sự kiện thay đổi kích thước khung.

```
public boolean onTouch(View v, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            lastX = -1;
            lastY = -1;
            return true;
        case MotionEvent.ACTION_MOVE:
            int currentX = (int) event.getX();
            int currentY = (int) event.getY();
    }
}
```



Hình 32: Phương thức ***onTouch()*** được sử dụng trong class ***MainActivity***

Class ***CameraManager*** dùng để hiển thị Camera lên màn hình chính của ứng dụng. Dựa giao diện Camera lên lớp ***SurfaceView*** để có thể thực hiện một số chức năng khác trên cùng một giao diện như ***Crop Box***, ***Take Picture*** ...



The screenshot shows the Android Studio interface with the project 'PhotoTranslate' open. The 'Project' tool window on the left shows the file structure. The 'Java' section under 'src/main/java/corbatodd.phototranslate' contains several classes, with 'CameraManager.java' selected and highlighted. The code editor on the right displays the implementation of the CameraManager class. The code includes imports for Context, Point, Rect, Camera, Handler, SurfaceHolder, and IOException. It defines static final variables for frame dimensions and creates a Context, CameraConfigurationManager, Camera, AutoFocusManager, and Rect objects. The constructor initializes these and sets up a PreviewCallback. The openDriver method opens the camera if it's null. The code is annotated with a note: /* * Created by CorbaTodd on 4/2/15. */.

```

package corbatodd.phototranslate;

import android.content.Context;
import android.graphics.Point;
import android.graphics.Rect;
import android.hardware.Camera;
import android.os.Handler;
import android.view.SurfaceHolder;

import java.io.IOException;

/**
 * Created by CorbaTodd on 4/2/15.
 */
public final class CameraManager {

    private static final int MIN_FRAME_WIDTH = 50;
    private static final int MIN_FRAME_HEIGHT = 20;
    private static final int MAX_FRAME_WIDTH = 800;
    private static final int MAX_FRAME_HEIGHT = 600;

    private final Context context;
    private final CameraConfigurationManager configManager;
    private Camera camera;
    private AutoFocusManager autoFocusManager;
    private Rect framingRect;
    private Rect framingRectInPreview;
    private boolean initialized;
    private boolean previewing;
    private boolean reverseImage;
    private int requestedFramingRectWidth;
    private int requestedFramingRectHeight;

    private final PreviewCallback previewCallback;

    public CameraManager(Context context) {
        this.context = context;
        this.configManager = new CameraConfigurationManager(context);
        previewCallback = new PreviewCallback(configManager);
    }

    public synchronized void openDriver(SurfaceHolder holder) throws IOException {
        Camera theCamera = camera;
        if (theCamera == null) {
            theCamera = Camera.open();
        }
    }
}

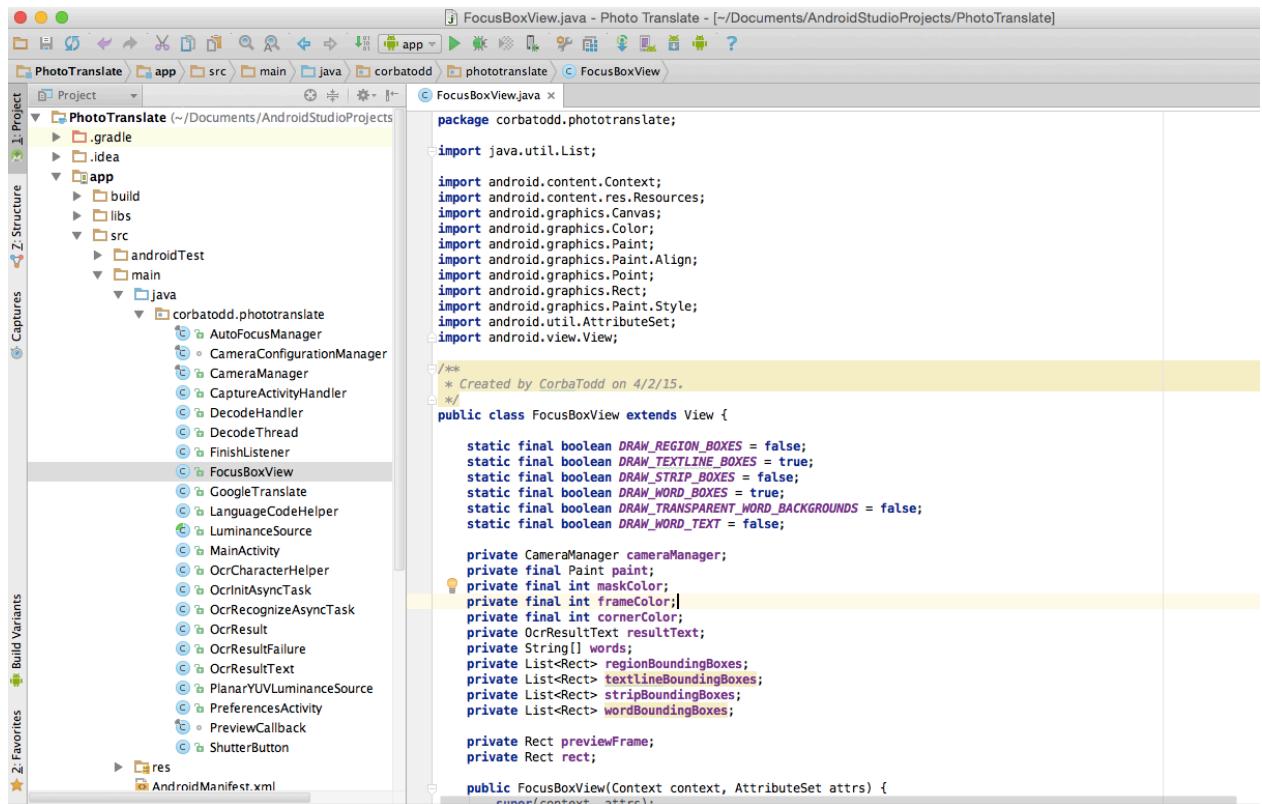
```

Hình 33: Class *CameraManager*

Xây dựng chức năng tự động lấy nét trong class ***AutoFocusManager***

FOCUS_MODES_CALLING_AF.add(Camera.Parameters.FOCUS_MODE_AUTO);

Class ***FocusBoxView*** dùng để hiển thị một khung *Crop* có thể tùy chỉnh kích thước để chọn khu vực ảnh cần chụp, giúp giảm thiểu các vật thể dư thừa, tập trung vào đoạn hình ảnh cần rút trích văn bản để nâng cao khả năng nhận diện văn bản cũng như hiệu quả xử lý.

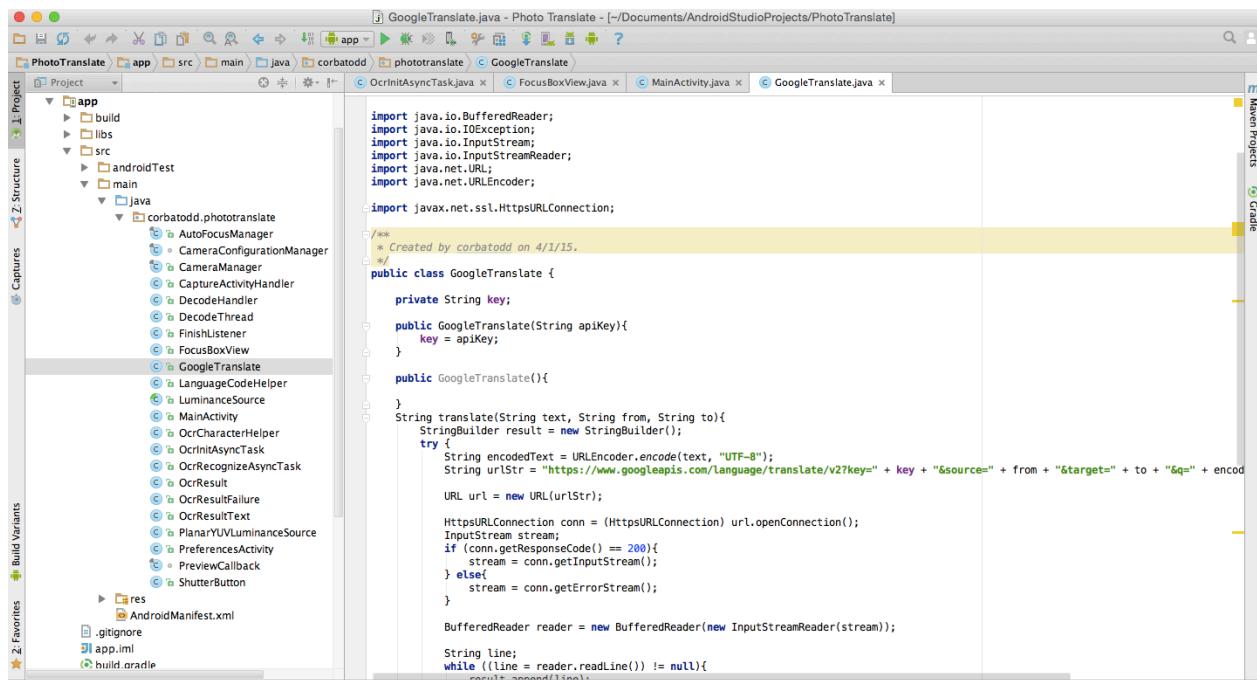


Hình 34: Class *FocusBoxView*

Sử dụng lớp *Paint* để vẽ khung *Crop*

```
paint.setColor(frameColor);
canvas.drawRect(frame.left, frame.top, frame.right + 1, frame.top + 2, paint);
canvas.drawRect(frame.left, frame.top + 2, frame.left + 2, frame.bottom - 1, paint);
canvas.drawRect(frame.right - 1, frame.top, frame.right + 1, frame.bottom - 1, paint);
canvas.drawRect(frame.left, frame.bottom - 1, frame.right + 1, frame.bottom + 1, paint);
```

Class ***GoogleTranslate*** cung cấp các phương thức để sử dụng dịch vụ ***Google Translate*** của ***Google*** với sự hỗ trợ của thư viện ***GSon***.



Hình 35: Class ***GoogleTranslate***

Để có thể sử dụng được dịch vụ ***Google Translate*** trong project ta cần phải đăng ký ***API*** của ***Google*** tại địa chỉ https://cloud.google.com/translate/v2/getting_started

Tạo ***project*** sau đó tạo ***key*** để dùng trong ứng dụng.

Sử dụng đoạn code sau để gửi và lấy dữ liệu:

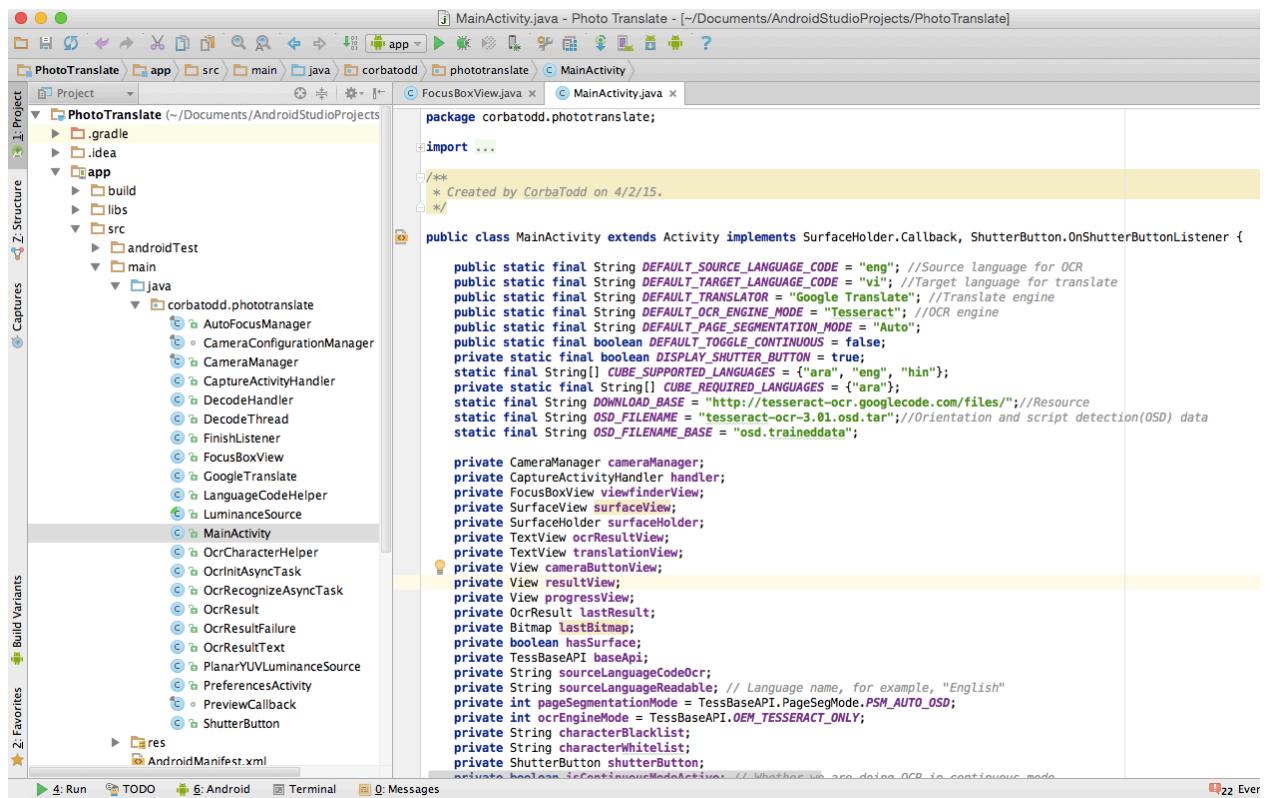
```

String urlStr =
"https://www.googleapis.com/language/translate/v2?key=" + key +
"&source=" + from + "&target=" + to + "&q=" + encodedText;

```

Dịch vụ ***Google Translate*** là dịch vụ có phí nên để có thể sử dụng dịch vụ này ta cần phải có tài khoản ***Visa***. Dịch vụ miễn phí dịch 2 triệu kí tự/ngày. Nên để có thể sử dụng tốt dịch vụ ta cần kích hoạt ***Billing*** cho ứng dụng.

Cuối cùng là class **MainActivity** điều khiển toàn bộ ứng dụng bao gồm khởi động Camera, xử lý hình ảnh, chuyển đổi hình ảnh văn bản sang văn bản bằng công cụ **Tesseract OCR** và dịch văn bản sau đó hiển thị cho người dùng thấy.



The screenshot shows the Android Studio interface with the project 'PhotoTranslate' open. The 'MainActivity.java' file is selected in the editor. The code implements the SurfaceHolder.Callback and ShutterButton.OnShutterButtonListener interfaces. It defines constants for source and target languages, the translate engine, and page segmentation mode. It also initializes various components like CameraManager, CaptureActivityHandler, and ShutterButton. The code includes comments indicating it was created by CorbaTodd on 4/2/15.

```

package corbatodd.phototranslate;

import ...

/**
 * Created by CorbaTodd on 4/2/15.
 */

public class MainActivity extends Activity implements SurfaceHolder.Callback, ShutterButton.OnShutterButtonListener {

    public static final String DEFAULT_SOURCE_LANGUAGE_CODE = "eng"; //Source language for OCR
    public static final String DEFAULT_TARGET_LANGUAGE_CODE = "vi"; //Target language for translate
    public static final String DEFAULT_TRANSLATOR = "Google Translate"; //Translate engine
    public static final String DEFAULT_OCR_ENGINE_MODE = "Tesseract"; //OCR engine
    public static final String DEFAULT_PAGE_SEGMENTATION_MODE = "Auto";
    public static final boolean DEFAULT_TOGGLE_CONTINUOUS = false;
    private static final boolean DISPLAY_SHUTTER_BUTTON = true;
    static final String[] CUBE_SUPPORTED_LANGUAGES = {"ara", "eng", "hin"};
    private static final String[] CUBE_REQUIRED_LANGUAGES = {"ara"};
    static final String DOWNLOAD_BASE = "http://tesseract-ocr.googlecode.com/files/"; //Resource
    static final String OSD_FILENAME = "tesseract-ocr-3.01.osd.tar"; //Orientation and script detection(OSD) data
    static final String OSD_FILENAME_BASE = "osd.traineddata";

    private CameraManager cameraManager;
    private CaptureActivityHandler handler;
    private FocusBoxView viewfinderView;
    private SurfaceView surfaceView;
    private SurfaceHolder surfaceHolder;
    private TextView ocrResultView;
    private TextView translationView;
    private View cameraButtonView;
    private View resultView;
    private View progressView;
    private OcrResult lastResult;
    private Bitmap lastBitmap;
    private boolean hasSurface;
    private TessBaseAPI baseApi;
    private String sourceLanguageCodeOcr;
    private String sourceLanguageReadable; // Language name, for example, "English"
    private int pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_AUTO_OSD;
    private int ocrEngineMode = TessBaseAPI.OEM_TESSERACT_ONLY;
    private String characterBlacklist;
    private String characterWhitelist;
    private ShutterButton shutterButton;

    private boolean continuousModeEnabled; // Whether we are doing OCR in continuous mode
}

```

Hình 36: Class MainActivity

Thành phần quan trọng nhất của ứng dụng là thư viện **Tesseract** hỗ trợ việc chuyển đổi hình ảnh văn bản sang văn bản.

Đoạn code đơn giản dùng để sử dụng công cụ **Tesseract**:

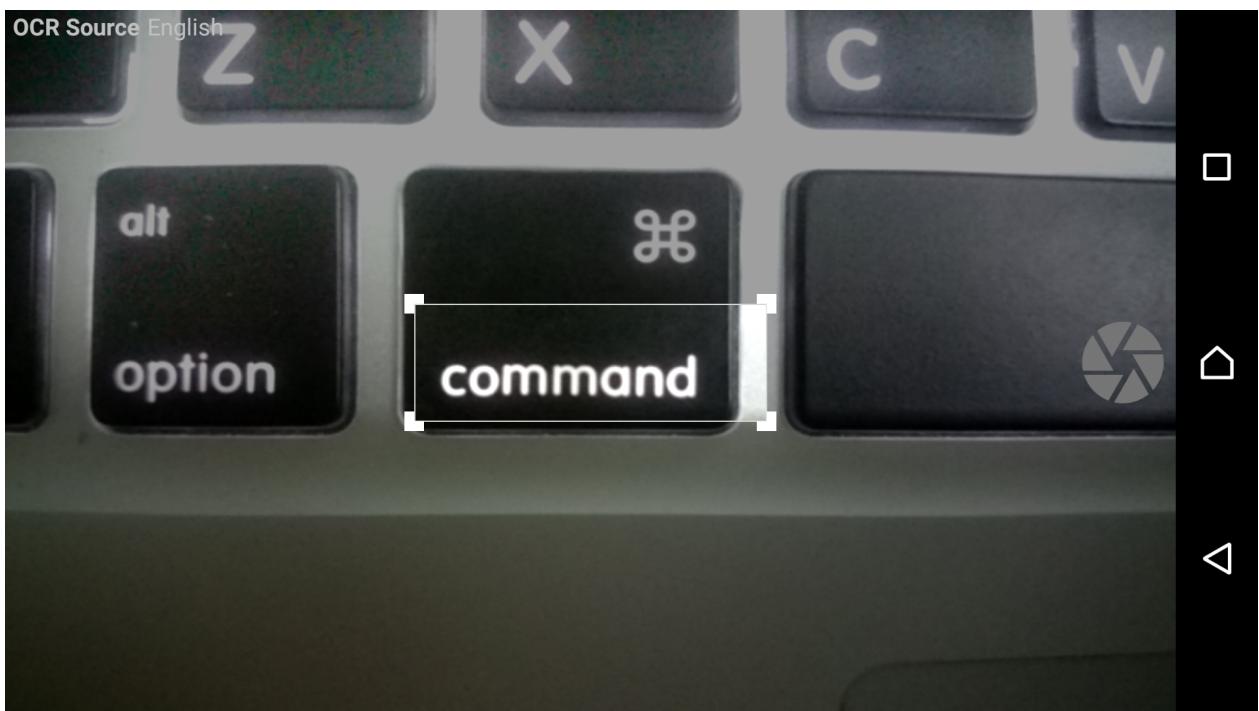
```
public void doOCR() {  
  
    bitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);  
  
    TessBaseAPI baseAPI = new TessBaseAPI();  
    baseAPI.setDebug(true);  
    baseAPI.init(dataPath, lang);  
  
    baseAPI.setImage(bitmap);  
  
    tessOCR.setBitmap(bitmap);  
  
    bitmap = tessOCR.getAnnotatedBitmap();  
  
    result = baseAPI.getUTF8Text();  
    baseAPI.end();  
  
    tivCroppedBitmap.setImageBitmap(bitmap);  
  
    edtResult.setText(result);  
}
```

4. Cài đặt và sử dụng ứng dụng

Thiết bị cài đặt ứng dụng: điện thoại **Sony Xperia Z1** sử dụng hệ điều hành Android 5.0.2 Lollipop với Camera có hỗ trợ chức năng Auto Focus.

Ở lần đầu thực thi ứng dụng, ứng dụng sẽ tự động kiểm tra dữ liệu bộ huấn luyện ngôn ngữ tiếng Anh phục vụ cho quá trình chuyển đổi hình ảnh văn bản sang văn bản, nếu trong bộ nhớ máy chưa có dữ liệu huấn luyện ngôn ngữ thì ứng dụng sẽ tự động tải về và cài đặt gói dữ liệu đó. Hiện tại ứng dụng chỉ hỗ trợ chuyển đổi hình ảnh văn bản tiếng Anh nên ứng dụng sẽ tải dữ liệu của bộ huấn luyện ngôn ngữ tiếng Anh. Tính năng này yêu cầu điện thoại phải có kết nối Internet(Wifi, 3G).

Sau khi quá trình tải và cài đặt hoàn tất, ứng dụng sẽ có giao diện hiển thị như sau



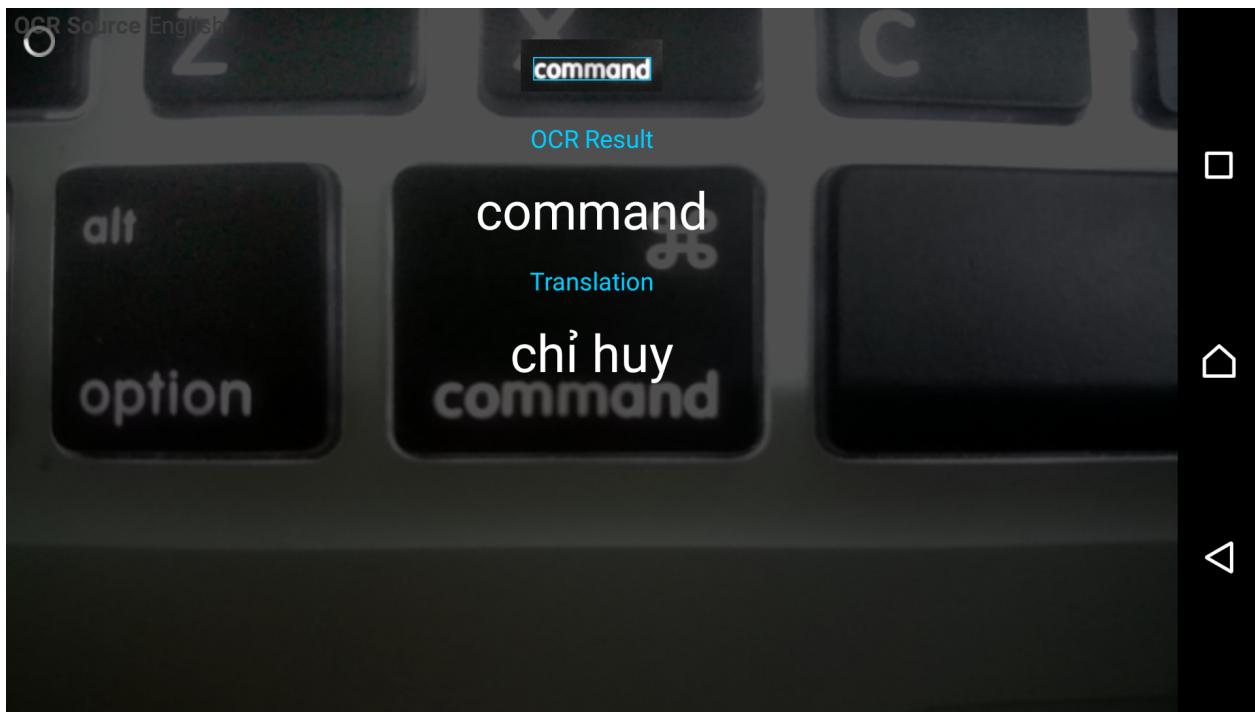
Hình 37: Giao diện thực thi ứng dụng

Chức năng Auto Focus tự động kích hoạt ở mỗi khoảng thời gian xác định. Người dùng sử dụng Khung giới hạn vùng chụp để điều chỉnh vùng hình ảnh cần rút trích văn bản. Khung giới hạn càng chính xác và focus càng sắc nét thì hiệu quả rút trích càng cao.

Các thành phần chính của giao diện thực thi ứng dụng:

- **SurfaceView** dùng để hiển thị giao diện **Camera**.
- **FocusBoxView** dùng để hiển thị khung **CropBox** ở giữa màn hình và có thể tùy chỉnh kích thước cho phù hợp với phần text cần lấy.
- Button **Take Picture** dùng để chụp ảnh.
- TextView **OCR Source** dùng để hiển thị ngôn ngữ có thể nhận dạng.

Sau khi giới hạn được hình ảnh chứa đoạn text cần lấy, nhấn nút **Take Picture** để ứng dụng thực hiện xử lý hình ảnh và chuyển đổi hình ảnh văn bản sang văn bản, sau đó dùng **Google Translate** dịch và nhận kết quả (việc dịch thông qua **Google Translate** cần có internet để thực hiện). Đồng thời các thành phần như khung Crop và Button chụp ảnh sẽ ẩn đi để hiển thị kết quả. Nếu trong quá trình Crop và Focus không chính xác hoặc không rõ nét dẫn đến việc ứng dụng không nhận diện được kí tự, ứng dụng sẽ hiển thị thông báo “**Cannot recognize text! Try again!**” và người sử dụng phải thực hiện lại việc focus chính xác vào đoạn text cần lấy.



Hình 38: hình ảnh sau khi văn bản được rút trích và nhận kết quả dịch

Các thành phần chính của giao diện hiển thị kết quả:

- **ImageView** dùng để hiển thị ảnh đã được chụp đã qua xử lý hình ảnh, đồng thời hiển thị khung **Bounding Box** bao quanh chữ đã được nhận diện.
- **TextView** dùng để hiển thị kết quả nhận diện kí tự và kết quả dịch.

Nếu không hài lòng với kết quả đạt được có thể sử dụng nút Back để quay lại màn hình giao diện chính và tiếp tục thực hiện trình chụp ảnh lại.

5. Kết quả đạt được

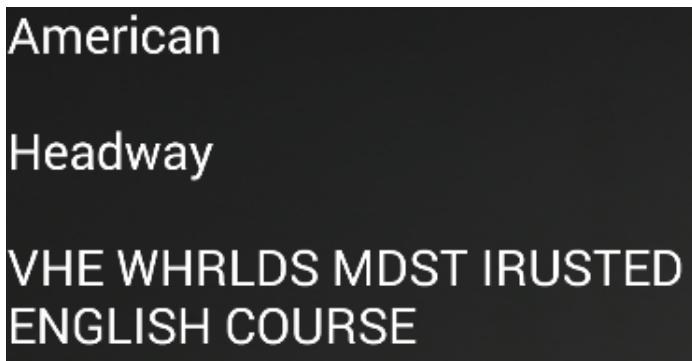
Sau khi khởi chạy chương trình và sử dụng camera để chụp một đoạn Text trong điều kiện camera tốt, chất lượng hình ảnh và chữ tốt thì kết quả tương đối khả quan.

Một số kết quả thử nghiệm mà nhóm đạt được:

Hình ảnh đưa vào



Kết quả nhận được



Kết quả cho thấy chất lượng camera và hình ảnh quyết định khá nhiều trong việc chuyển đổi hình ảnh văn bản sang văn bản. Việc tiền xử lý hình ảnh nhằm loại bỏ chi tiết thừa, giúp ảnh sáng hơn cũng ảnh hưởng rất lớn đến kết quả thu được.

Hiện ứng dụng chỉ có thể nhận dạng được các đoạn text bằng tiếng Anh và hỗ trợ dịch sang tiếng Việt. Trong thời gian tới nhóm sẽ thực hiện training thêm một số ngôn ngữ mà **Tesseract** hỗ trợ cũng như dịch sang các thứ tiếng khác mà **Google Translate** cho phép.

Do bước đầu thực hiện nên còn nhiều sai sót và hạn chế nên chất lượng ứng dụng chưa được tốt vì chưa nâng cao chất lượng hình ảnh thu được, vấn đề này làm ảnh hưởng rất lớn đến kết quả nhận dạng.

KẾT LUẬN

Sau quá trình học tập, tìm hiểu và thực hiện đồ án chúng em đã tìm hiểu được tổng quát về hệ điều hành Android, lịch sử hình thành, kiến trúc, thiết kế đồng thời tìm hiểu được cách để lập trình một ứng dụng trên hệ điều hành này. Hiểu được các thành phần cơ bản cấu thành nên một ứng dụng chạy trên hệ điều hành Android và thông qua đó chúng em đã hoàn thành được đồ án cũng như xây dựng được ứng dụng **Photo Translate** với chức năng chuyển đổi hình ảnh văn bản sang văn bản và hỗ trợ dịch với dịch vụ Google.

Để hoàn thành ứng dụng này không thể không kể đến tầm quan trọng của công cụ **Tesseract** hỗ trợ việc chuyển đổi hình ảnh văn bản sang văn bản, đây thành phần chính làm nên ứng dụng.

Đồng thời với sự hỗ trợ từ thư viện **OpenCV** cung cấp các hàm xử lý hình ảnh như làm mờ, mức xám hay nhị phân hóa hình ảnh mức nguồng động đã góp một phần không nhỏ vào việc nâng cao chất lượng hình ảnh đầu vào, góp phần cải thiện kết quả đạt được sau khi đưa vào chuyển đổi bằng công cụ **Tesseract**.

Một số vấn đề đã hiện thực:

- Xây dựng được ứng dụng **Android** với các chức năng như chụp ảnh, tự động lấy nét,...
- Sử dụng công cụ **Tesseract** hỗ trợ chuyển đổi hình ảnh văn bản sang văn bản
- Xử lý ảnh sử dụng công cụ **OpenCV**
- Dịch văn bản sử dụng dịch vụ **Google Translate** của **Google**

Các vấn đề chưa hiện thực:

- Chọn và crop hình ảnh từ thư viện ảnh của máy
- Nâng cao chất lượng hình ảnh đầu vào
- Chưa hỗ trợ nhiều ngôn ngữ

Trong thời gian tới, chúng em sẽ cố gắng phát triển tốt hơn nữa những gì đã hiện thực được và hoàn thành những vấn đề chưa hiện thực được để đáp ứng yêu sử dụng. Bổ sung thêm những tính năng hữu ích và thiết kế giao diện trực quan hơn nhằm mang lại kết quả cao cũng như sự tiện lợi mà ứng dụng mang lại cho người sử dụng.

TÀI LIỆU THAM KHẢO

Tài liệu viết:

- [1] **Bùi Tân Lộc, Cao Thái Phương Thành**, Nghiên cứu và xây dựng ứng dụng từ điển trên điện thoại di động, Luận văn cử nhân tin học, Đại học Khoa học Tự nhiên TP.HCM, 2004.
- [2] **Marko Gargenta**, Learning Android, O'REILLY, 2011.
- [3] **Ray Smith**, An overview of Tesseract OCR engine, IEEE Computer Society, 2007.
- [4] **Ray Smith, Daria Antonova, Dar Shyang-Lee**, Adapting the Tesseract Open Source OCR Engine for Multilingual OCR, ACM, 2009.

Website:

- [1] <http://developer.android.com/index.html>
- [2] <http://tesseract-ocr.googlecode.com/>
- [3] <http://www.opencv.org/>
- [4] <https://code.google.com/p/google-gson/>
- [5] <https://code.google.com/p/jtar/>

PHỤ LỤC

1. Native development kit (NDK)

1.1. Giới thiệu chung

Khi viết một ứng dụng Android ở tầng trên bằng ngôn ngữ **Java** mà ta có nhu cầu gọi lại các hàm hoặc thư viện ở tầng bên dưới (thường là các đoạn mã ở tầng dưới được viết bằng **C/C++**). Để **Java** có thể hiểu và truy xuất được các đoạn mã **C/C++** thì ta cần một giao diện chung giữa hai ngôn ngữ. Giao diện chung đó được gọi là **Java Native Interface – JNI**.

Native Development Kit – NDK là một bộ công cụ đi kèm với **Android SDK** giúp cho các nhà phát triển có thể viết hoặc nhúng các đoạn mã nguồn bằng **C/C++** bên trong chương trình. Các ứng dụng Android hoạt động trên máy ảo Dalvik. Chính nhờ **NDK** mà các ứng dụng có thể gọi được các đoạn mã gốc – **Native Code** được sử dụng trong chương trình.

1.2. Các hỗ trợ của NDK

Bộ công cụ **NDK** cung cấp các hỗ trợ sau:

- Một tập hợp các công cụ và tập tin để phát sinh ra các thư viện mã từ **C/C++**.
- Cách thức nhúng các đoạn mã phát sinh từ **C/C++** vào trong tập tin đóng gói ứng dụng (**.apk**) chạy được trên các thiết bị Android.
- Cung cấp một tập các header và thư viện sẽ được hỗ trợ ở tất cả các phiên bản **Android** từ 1.5 trở đi. Từ phiên bản 2.3 có hỗ trợ thêm viết **Native Activity**.
- Các tài liệu, mã nguồn mẫu và hướng dẫn.

1.3. Sử dụng NDK

Không phải lúc nào sử dụng **NDK** cũng có lợi cho chương trình. Vì sử dụng mã gốc (**Native code**) trong chương trình không làm tăng hiệu năng thực thi mà chỉ làm tăng thêm sự phức tạp cho ứng dụng. Chỉ sử dụng mã gốc trong trường hợp cần thiết để làm giảm sự phức tạp cho chương trình.

Android Framework cung cấp 2 cách để sử dụng **Native Code** trong chương trình:

- Viết ứng dụng sử dụng **Android Framework** và sử dụng **JNI** để truy cập các hàm **API** được cung cấp trong bộ công cụ **Android NDK**. Ưu điểm của kỹ thuật này là chúng ta có thể tận dụng các lợi ích của **Android Framework** mà vẫn sử dụng được mã gốc khi cần thiết.
- Viết một **Native Activity** để hiện thực cài đặt chu trình của ứng dụng bằng mã gốc. Bộ công cụ **Android SDK** sẽ cung cấp lớp **NativeActivity** là lớp tiện ích để hiện thực cài đặt vòng đời của ứng dụng thông qua các hàm (**OnCreate**, **OnPause**...).

1.4. Nội dung của bộ NDK

Bao gồm các công cụ và thư mục sau:

Công cụ phát triển: Bao gồm tập hợp các công cụ phát triển (trình biên dịch, trình liên kết – linker) để phát sinh ra mã nhị phân cho bộ vi xử lý **ARM** chạy trên các nền tảng **Linux**, **OSX** và **Windows** (sử dụng kèm với công cụ **Cygwin**). Các công cụ phát triển này còn cung cấp một tập hợp các hệ thống header dùng cho các hàm **API** gốc ổn định và được đảm bảo là sẽ hỗ trợ trong tất cả các phiên bản sau này của nền tảng **Android**:

- **Libc** (thư viện **C**) **header**.
- **Libm** (thư viện toán học) **header**.
- Giao diện **JNI header**.
- **Libz** (nén và giải nén) **header**.
- **Liblog** (dùng cho việc ghi log trên **Android**) **header**.
- **OpenGL ES 1.1** và **OpenGL ES 2.0** (thư viện đồ họa ba chiều) **header**.
- **Libjnigraphics** (truy cập vùng nhớ đệm trên các pixel) **header**.
- Các header hỗ trợ cho **C++**.
- **OpenSL ES** (thư viện âm thanh gốc).
- Các **API** hỗ trợ ứng dụng gốc trên **Android**.

Ngoài ra **NDK** còn cung cấp cho chúng ta một hệ thống biên dịch mã nguồn hiệu quả mà không cần phải có sự điều khiển chi tiết các công cụ/nền tảng/vi xử lý. Người dùng sẽ chỉ phải tạo ra các tập tin nhỏ để chỉ thị cho việc biên dịch mã nguồn sẽ được dùng trong chương trình. **NDK** sẽ dựa vào các tập tin biên dịch này để biên dịch mã nguồn để tạo ra thư viện liên kết động và đặt trực tiếp thư viện này trong dự án.

Bộ tài liệu: Bộ **NDK** còn chứa tập hợp nhiều tài liệu để mô tả các tính năng của **NDK**, cách thức sử dụng, viết tập tin biên dịch mã nguồn, cách thức tạo thư viện liên kết động... Người dùng có thể tham khảo thêm trong thư mục **<ndk>/docs**.

Các ứng dụng mẫu: Cung cấp các ứng dụng được viết sẵn cho người dùng tham khảo.

2. Thư viện xử lý hình ảnh OpenCV (*Open Source Computer Vision*)

2.1. Giới thiệu Computer Vision

OpenCV là một thư viện tập hợp các hàm lập trình chủ yếu nhằm vào công nghệ thời gian thực của thị giác máy tính, được phát triển bởi Intel và hiện tại được hỗ trợ bởi **Willow Garage** và **Itseez**. OpenCV là thư viện mã nguồn mở miễn phí theo giấy phép **BSD** nhằm giúp cho người dùng có thể dễ dàng thay đổi và sử dụng.

Hiện nay, OpenCV có hơn 2500 thuật toán tối ưu hóa trong đó bao gồm một tập hợp tất cả các thuật toán về thị giác máy tính và máy học. Các thuật toán này có thể được sử dụng để phát hiện và nhận diện khuôn mặt, nhận diện đối tượng, phân loại các hành động của con người trong video, theo dõi đối tượng chuyển động, trích xuất các mô hình 3D của đối tượng, tìm hình ảnh tương tự từ một cơ sở dữ liệu hình ảnh, loại bỏ mắt đỏ từ hình ảnh chụp sử dụng đèn flash, theo dõi chuyển động của mắt,... OpenCV hiện có hơn 47 nghìn người dùng trên toàn thế giới. Thư viện này được sử dụng rộng rãi trong các nhóm nghiên cứu của các công ty lớn như: **Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota**,... và các cơ quan chính phủ.

2.2. Lịch sử phát triển

Chính thức ra mắt năm 1999, dự án OpenCV ban đầu là một sáng kiến nghiên cứu của **Intel** để cải tiến **CPU** và các ứng dụng chuyên sâu, một phần của một loạt các dự án

bao gồm cả theo dõi các tia sáng theo thời gian thực và màn hình hiển thị 3D. Người đóng góp chính cho dự án tối ưu hóa bao gồm một số chuyên gia trong Intel Nga và nhóm các chuyên gia phòng nghiên cứu hiệu suất của Intel. Trong những ngày đầu của OpenCV, các mục tiêu của dự án đã được mô tả như OpenCV ban đầu chỉ hỗ trợ viết bằng ngôn ngữ **C**, nhưng hiện tại đã hỗ trợ thư viện và phát triển trên các nền tảng ngôn ngữ khác như **C++, Java**. Đồng thời nó cũng hỗ trợ đa môi trường hệ điều hành: **Windows, Linux, Android, MacOS,...**

Phiên bản alpha đầu tiên của OpenCV được phát hành ra công chúng tại hội nghị **IEEE** về **Computer Vision** và **Pattern Recognition** vào năm 2000. Phiên bản **1.0** đầu tiên được phát hành vào năm 2006. Vào giữa năm 2008, OpenCV được sự ủng hộ của công ty từ **Willow Garage**. Tháng 10/2009, OpenCV 2.0 được phát hành với nhiều cải tiến vượt bậc bao gồm những thay đổi lớn về thay đổi ngôn ngữ lập trình, về giao diện, sử dụng dễ dàng hơn, một số chức năng mới và khai thác tốt hơn về hiệu suất (đặc biệt là trên các hệ thống đa lõi).

Trong tháng 8/2012, OpenCV được hỗ trợ và tiếp quản bởi một tổ chức phi lợi nhuận và website **OpenCV.org** được xây dựng ngay sau đó.

2.3. Cấu trúc của OpenCV

Cấu trúc của OpenCV gồm bốn thành phần chính:

- **CxCore**: chứa các cấu trúc cơ bản như điểm, đường, dãy, mặt, ma trận,... và các thao tác cấp thấp liên quan.
- **MLL (Machine Learning Library)** là thư viện Machine Learning, thư viện này bao gồm rất nhiều lớp thống kê và công cụ xử lý.
- **CV (Computer Vision)**: chức hầu hết các thao tác liên quan đến việc xử lý ảnh ở cấp thấp như lọc ảnh, trích biên, phân vùng, tìm Contour, biến đổi Fourier,...
- **HighGUI**: các thao tác lên những file ảnh và file Video như đọc ảnh, hiển thị ảnh, chuyển đổi định dạng.