**TEL AVIV UNIVERSITY**

The Iby and Aladar Fleischman Faculty of Engineering

The Zandman-Slaner School of Graduate Studies

**Andrid Environment for SLAM SW Development**

project submitted by

Ran Attias - 027214733

February 2021

# 

# Abstract

Simultaneous Localization And Mapping (SLAM) algorithms are widely used in the open-source robot operating system (ROS) libraries, but are less available in Android operating system (OS) environments.

Implementing SLAM on Android OS requires understanding of the entire system environment starting from the hardware and the available sensors suit, throughout basic knowledge in software drivers and computation libraries, and up to application software layers and graphic user interface (GUI) implementation. That is a vast range of knowledge and expertise.

This project aims to explore optional environments and SW frameworks that can be used for SLAM development on Android systems and provide a reference guide for setting up the development environment and implementing image processing algorithms on Android OS.

# Table Of Contents

[Abstract 2](#_Toc57304957)

[Table Of Contents 3](#_Toc57304958)

[Introduction 4](#_Toc57304959)

[Hardware Selection 5](#_Toc57304960)

[Hardware comparison table 5](#_Toc57304961)

[Selected Development Hardware 6](#_Toc57304962)

[SLAM on Android Software - Web Survey 6](#_Toc57304963)

[Exploring Image Processing Environment and Frameworks 7](#_Toc57304964)

[VLC - Open source Application 8](#_Toc57304965)

[VLC Option summary 9](#_Toc57304966)

[GStreamer - Open source Framework 10](#_Toc57304967)

[Working flow 10](#_Toc57304968)

[GStreamer solution summery 11](#_Toc57304969)

[Recommendation 11](#_Toc57304970)

[Summary 12](#_Toc57304971)

[Appendix: GStreamer environment - work log 13](#_Toc57304972)

[Appendix: Android Project for Streaming RTSP 16](#_Toc57304973)

[Appendix: VLC work log 22](#_Toc57304974)

# Introduction

SLAM technology is already widely used for commercial and industrial products. It allows a device to map its environment while positioning itself in it. In robotics implementations SLAM enables the transition from automated guided vehicles to autonomous mobile robots.

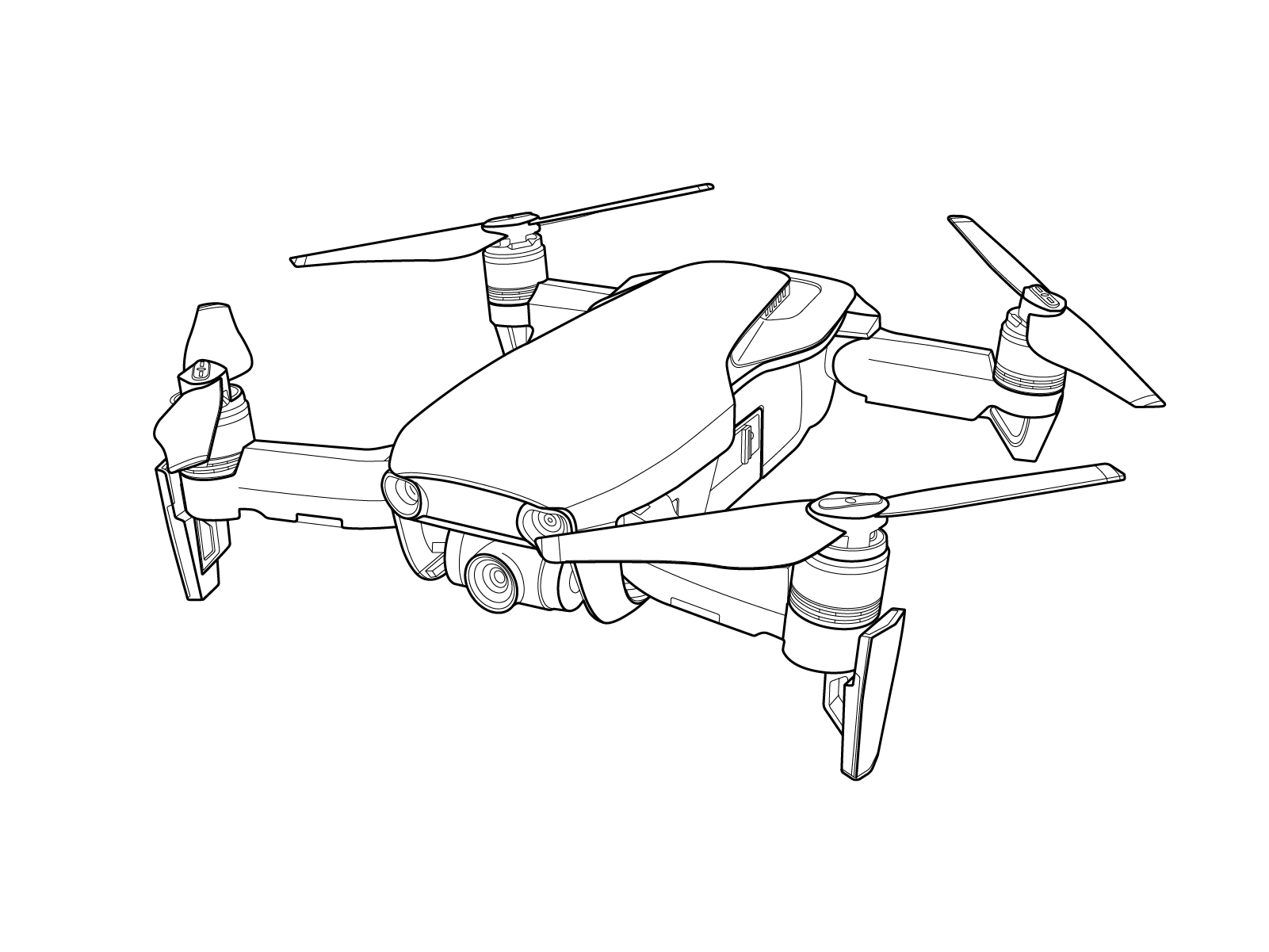
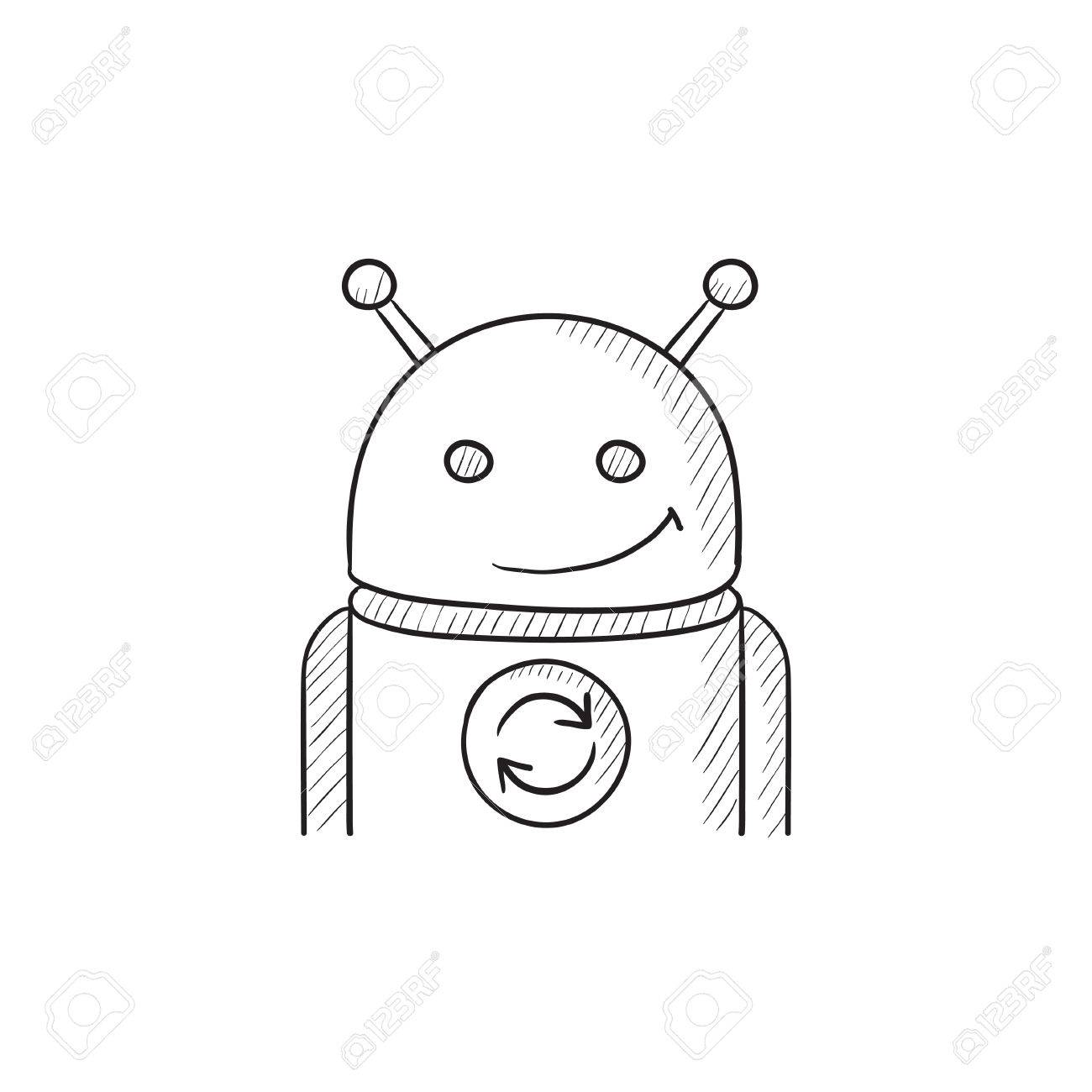
In the augmented reality (AR) products, SLAM makes it possible to recognize 3D objects and scenes, and to overlay digital interactive augmentations.

***Google Maps using SLAM for AR overlay***

In this project we are looking into different scenarios where a drone is streaming a video stream captured by it’s camera to an Android device, and the SLAM & computer vision (CV) algorithm is running in the device itself.

Android Device

Display



CV & SLAM

Algorithm

***Concept diagram of our project use-case***

# Hardware Selection

As part of our project work, we held a basic survey of the available development hardware. This section is relevant to the time of the survey.

As our project focus is running SLAM algorithms on Android OS, we realized we should look into processors used in the mobile smartphone market, and more precisely, latest computation devices introduced by Qualcomm Inc.

Qualcomm is a leading silicon provider for mobile phones market. Qualcomm’s Snapdragon device family is holding 36% of the market share in 2019.

We chose to pursue Qualcomm SDM845 SoC (System On Chip) based hardware, as it was the most advanced SoC just showed up on smartphones OEM flagship devices.

### Hardware comparison table

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Google Pixels 4** | **Snapdragon 845**  **Mobile Kit** | **Snapdragon 845  VR Kit** |
| **Cost** | $323 | $1079 | Not available |
| Display\* - $185 |
| Camera\*\* - $350 |
| **Body** | Small, case | Open, bulky | Head straps |
| **Camera** | Integrated | Plug-in, not included |  |
| **Qualification** | In production | Development | Development |
| **Sensors** | accelerometer, gyro, compass, | Not available | 6 DOF, 800Hz |
| **SW** | Android 9 | Android 10 | Android 10 |

\* Open-Q™ 845 HDK LCD (<https://shop.intrinsyc.com/collections/accessories/products/open-q-845-hdk-lcd>)

\*\* Open-Q™ 845/670 HDK Camera <https://shop.intrinsyc.com/collections/accessories/products/open-q-845-hdk-camera>)

### Selected Development Hardware

Using the latest Google Pixel device seems to be the best option at the time of this project. Pixel 5 was released on Oct. 19 (two months before writing) and the next Pixel is expected at the only end of 2021. so Pixel 4 looks to us as the most advanced system available and suits our project requirement.

# SLAM on Android Software - Web Survey

We’ve conducted a wide survey of the available open source libraries, repositories and former work done on SALM on Android.

Our search was focused on open source solutions. As we knew that most solutions were Linux based, we searched for libraries and repositories that are cross platform, meaning, they can build and run on both Linux and Android OS.

Here are some repositories we used as a reference to our work:

<https://github.com/raulmur/ORB_SLAM2>

<https://github.com/ivalab/gf_orb_slam2>

<https://github.com/UZ-SLAMLab/ORB_SLAM3>

<https://github.com/xdspacelab/openvslam>

All of the open source repositories we researched are for Linux Ubuntu build and we couldn't migrate from them to the Android environment.

The article <http://wscg.zcu.cz/wscg2019/Short/A79-full.PDF> provides an overview of the process of SLAM development on Android, but it focuses mainly on sensor calibration, and do not provide any software development details.

# Exploring Image Processing Environment and Frameworks

We will describe here our work exploring software environments and frameworks that can enable SLAM and image processing software development.

For the **communication** with the drone we explored VLC, GSTREAMER, FFMPEG and android native socket with various codecs.

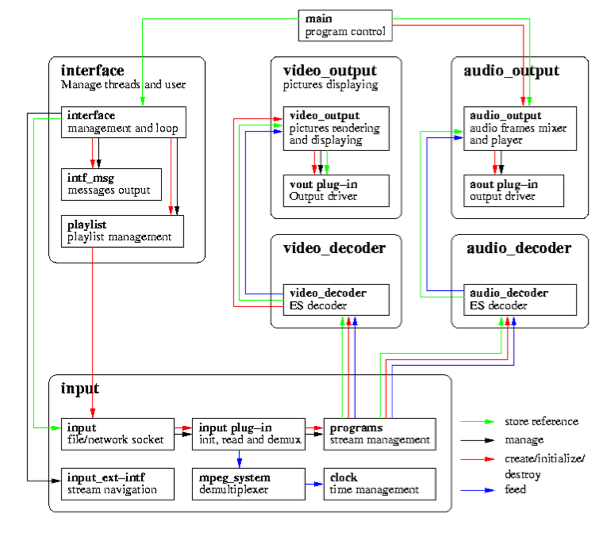
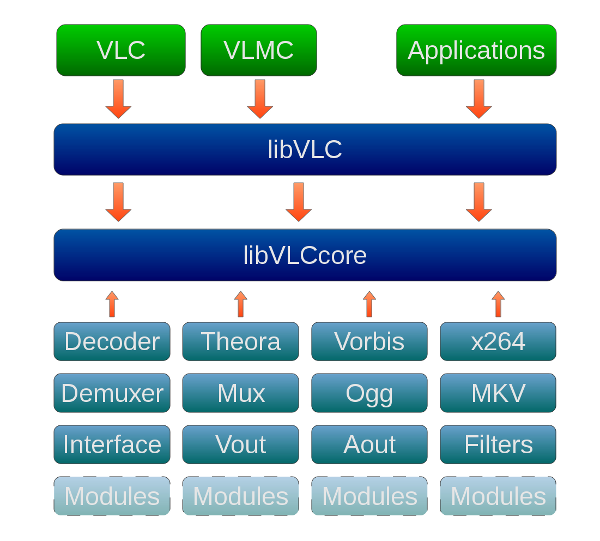
For the **frames manipulation** we explore GSTREAMER, OpenCV and JAVACV.

# VLC - Open source Application

Our initial attempt was to try using the widely used, open source VNC application known as VideoLAN Client . VLC is a cross platform streaming media player, which was started as an academic project in 1996.

VLC is a modular framework which allow entertaining new plugins and modules to support new multimedia file formats

Many of VLC codecs provide by FFmpeg project (lead us later to other solution direction in this project), and mainly use its own audio/video muxer and demuxer



***VLC SW architecture VLC Pipeline***

Sources:

* <https://en.wikipedia.org/wiki/VLC_media_player>
* VLC for android github, <https://code.videolan.org/videolan/vlc-android>
* stackoverflow, <https://stackoverflow.com/questions/36220552/vlc-video-architecture>

Our solution idea was to utilize the steaming capabilities of the application for receiving the video streamed from the drone, then, run SLAM or other video processing algorithm on the video buffer provide by the VLC, and then send it back to the application display pipeline.

## VLC Option summary

As VLC is a versatile video player with many supported features and capabilities, makes it very hard for tweak and change. In addition, its capabilities are way beyond our project needs.

The VLC application was mainly designed for streaming, and not image processing. The VLC application is not running any actual image processing algorithm and was not designed for such. Therefore, bringing OpenCV capabilities into it remove any advantage using it.

The GUI application is very complicated and hard to modify

# GStreamer - Open source Framework

In our search for a simplified framework we found the GStreamer framework to be a low level, open source library, with cross platform support for Linux, Windows and Android.

GStreamer is a multimedia framework uses pipeline structure to support a variety of media processing for both video and audio.

The GStreamer is a well-known, easy to use framework, with significant distribution. GStreamer was adopted for many video streaming products. It also seems to have a significant support community with frequent release updates.

In our learning process we went through more than 20 exercises available on GStreamer tutorial website, and many guidelines and training classes available on YouTube.

<https://gstreamer.freedesktop.org/documentation/tutorials/index.html?gi-language=c>

## Working flow

Although GStreamer is a cross platform framework, it doesn't provide any reference for Android development toolchain. The most common toolchain environment is the one running on Linux.

Therefore, the development process includes a Linux based toolchain called Celbero and CMAKE scripting. Once GStreamer software build it completed, to tool created a library folder. The library folder should be copied to the Android Development environment.

Under the Android development environment, we created a very basic Android application that runs GStreamer pipeline.

Although Linux toolchain requires good DevOp skills, we manage to go through it using GStreamer community support. The toolchain complexity is the outcome of cross platform development. Whereas compiling GStreamer for Linux is quite straightforward, the required adjustments needed for Android build creation are confusing, not well documented, and not widely used . We found very little support among GStreamer community.

Eventually we manage to operate the toolchain and we manage to:

* Modify GStreamer library C code manipulating video frame
* Build GStreamer library and use it on Android studio
* Define GStreamer media pipeline streaming video from generic URL
* Create Android application (.apk file) run the GStreamer pipeline we define
* Run and .apk file on Android smartphone

## GStreamer solution summery

The GStreamer can be a good candidate for further investigation, searching for the right way integrating OpenCV libraries into GStreamer under Android. We believe that with knowledge in GStreamer compile and build toolchain, one should be able to create the video pipeline required for our use-case.

# Summary

As described in this report, our work covers several open source software solutions from SLAM on Android, along with other system aspects, as hardware, development environment, tooling and drone communication required for accomplish this project.

Bringing SLAM into Android, using open source software is very challenging as it should bridge over a wide range of technologies.

Android software environment provide the simplified toolchain allows easy way to create applications and user interface, but, on the other hand, it use many abstractions layer to reduce design complexity and dependency on hardware. Therefor it adds a significant complication for one to run real-time, low latency, heavy load computation software.

Our work lay down the building blocks and the challenges using open source. That should allow the following projects to pick the most suitable software framework and successfully integrate the SLAM algorithm into the Android environment.

# Appendix: GStreamer environment - work log

In this section we describe our GStreamer solution workflow and step by step commands need to bring up the environment.

We start with going through the GStreamer Tutorials available in the official website (<https://gstreamer.freedesktop.org/documentation/tutorials/basic/index.html?gi-language=c>) and then view some lectures available in YouTube.

The tutorial is lacking a good explanation about the toolchain required to compile and build the GStreamer libraries. We started with MESON toolchain, but then realized it was lacking Android support, therefore we had to move to CERBERO tool. The second build environment was also challenging for it’s dependencies on external libraries.

The building learning required extensive web search for support and solutions.

The latest GStreamer toolchain release was based on UBUNTU 16.04, and we had to adjust it to match UBUNTU 18.04, and later on to UBUTU 20.04.

The general concept of how to manipulate GStreamer video buffer is described in the next link with web support chat: <http://gstreamer-devel.966125.n4.nabble.com/Simple-example-of-a-video-filter-td3252938.html>

Based on that we start by changing the code in “effectv” GStreamer library, and make sure our change is visually seen on the steam. We manage to change “agingtv.c” file to control the aging effect (the aging effect add noise usually seen on old movies)

Once we modify the code we followed the GStreamer library building instructions available at:

* <https://gitlab.freedesktop.org/gstreamer/gst-plugins-base>
* <https://gitlab.freedesktop.org/gstreamer/gst-build/blob/master/README.md>

## GStreamer for Linux

The below describe how to build GStreamer on Linux UBUNTU machine.

As mention above, that environment have a good support and most solution can be found within developer community.

Key commands:

* meson builddir
* ninja -C builddir

(builddir is working directory inside gst-build git repo)

**Meson environment trouble shouting:**

* #error "Legacy libpng versions are not supported. Please use libpng 1.4+ (1.6+ recommended). To address that issue check: <https://gitlab.freedesktop.org/gstreamer/gst-build/issues/57>

Summery: Run the command:

meson configure -Dgst-plugins-bad:iqa=disabled -Dgst-plugins-good:png=disabled build/

Yet, on UBUNTU 20.04 we did not faced that issue.

* Meson version 0.54 and above should be used. To update meson 0.53 to 0.56: sudo -H pip3 install --target=/usr/bin --upgrade meson
* We faced issue with Cairo lib build, but it didn’t effect our code.

Once GStreamer build passed successfully, we can get into the development environment using the command:

* ninja -C builddir devenv

lunched Gstreamer pipeline using next commands:

* gst-launch-1.0 uridecodebin uri=https://www.freedesktop.org/software/gstreamer-sdk/data/media/sintel\_trailer-480p.webm ! videoconvert ! **agingtv** ! videoconvert ! autovideosink
* gst-launch-1.0 videotestsrc ! videoconvert ! **agingtv** ! videoconvert ! autovideosink

## GStreamer for Android

The tutorials for GStreamer on Android provide a good background and can be found at: <https://gstreamer.freedesktop.org/documentation/tutorials/android/index.html?gi-language=c>

To create a cross platform build, GStreamer community created the CERBERO, which is a python based build system, capable of aggregate, compile and packaging open source projects, to different platforms, using cross compilers. More information can be found at: <https://github.com/GStreamer/cerbero>

As we found the CERBERO a complicate system, we recommend using also the information shared on:

* <https://gstreamer.freedesktop.org/documentation/installing/building-from-source-using-cerbero.html?gi-language=c>
* <http://gstreamer-devel.966125.n4.nabble.com/Building-GStreamer-SDK-from-latest-sources-for-ARM-Android-td4670784.html>
* <https://gitlab.freedesktop.org/gstreamer/gst-build>
* <https://gitlab.freedesktop.org/gstreamer/gst-build/blob/master/README.md>

**Key operation commands:**

* *sudo* ./cerbero-uninstalled bootstrap (building tool directory)
* ./cerbero-uninstalled -c config/cross-android-universal.cbc package gstreamer-1.0
* sudo ./cerbero-uninstalled -c config/cross-android-x86.cbc buildone gstreamer-1.0
* sudo ./cerbero-uninstalled -c config/cross-android-x86.cbc wipe
* ./cerbero-uninstalled buildone -c config/cross-android-universal.cbc package gst-plugins-good-1.0 - 1.17.0.1

**CERBERO environment trouble shouting:**

* As mentioned above, in the time of that project we had to update our Linux platform to UBUNTU 18.04 as we failed to compile libpng.
* We also had to move to meson 0.54:
* download from <https://github.com/mesonbuild/meson/releases/tag/0.54.0>
* install using: /Downloads/meson-0.54.0$ python3 -m pip install . (use . running from uncompress folder)
* One should also make sure to run CERBERO with Android NDK R21, although GStreaem tutorial advice to use NDK18
* We notice a case where build command should run with [-y] flag, to default approve download:
* *sudo ./cerbero-uninstalled -c config/cross-android-universal.cbc bootstrap* **-y**
* When running Bootstrap command (build tools) CERBERO fail to fetch next build tool. libffi, Cairo. We could not find a solution for that. Here solution direction I start look into.
* To try build libffi use
  + - *sudo ./cerbero-uninstalled buildone libffi*
    - *sudo ./cerbero-uninstalled -c libffi*
* CERBERO fail to build Cairo. We could not find a solution for that. Here solution direction I start look into.
* Need to build libpng use
  + - *sudo ./cerbero-uninstalled libpng*
* To try build libffi use
  + - *sudo ./cerbero-uninstalled buildone cairo*
* If not pass, something went wrong, wipe all using command:
* *sudo ./cerbero-uninstalled wipe*

the lib folder to copy:

* When run on phone (Pixels 4) use **arm64** folder (arm64-v8a)
* When run on emulator use 0x86
  + - * + Need to install: glib2-dev, libmount-dev, flex, bison,
        + sudo apt-get install libmount-dev
        + sudo apt-get install flex
        + sudo apt-get install bison

**CERBERO environment on windows:**

Follow precondition installation as describe in:

<https://github.com/GStreamer/cerbero>

Run command

* Run msys.batfile
* cd /home/cerbero/cerbero-master
* python ./cerbero-uninstalled -v visualstudio package gstreamer-1.0

fail to build: str, plugin-bad, openjpeg

* python cerbero-uninstalled build gst-plugins-base-1.0

# Appendix: Android Project for Streaming RTSP

This repository is for app developers in the Android Studio environment. The app receives RTSP stream and casting opencv and native custom functions on the frames.

SW Development environment:

* Google NDK
* Eclipse IDE
* Android Studio

Between Android Studio and Eclipse, we choose to proceed with the later.

# Create Android project

New Basic android project

Jave (not kotlin)

API:28

Setup the native libraries

Step 1: Download OpenCV Android Library Go to the OpenCV Android Sourceforge page and download the latest OpenCV Android library and extract it to the android folder.

Link:

<https://sourceforge.net/projects/opencvlibrary/files/opencv-android/3.4.3/opencv-3.4.3-android-sdk.zip/download>

Step 2: Import OpenCV Module

Click on File -> New -> Import Module.

Browse to the folder where you extracted the OpenCV Android library zip file contents (sdk/java).

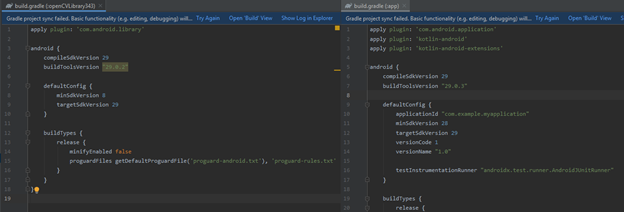
In windows: …(downloads) …\opencv-3.4.3-android-sdk\OpenCV-android-sdk\sdk\java

After that click ok, next and finish.

Step 3: Fixing Gradle Sync Errors

Browse to OpenCV library module and open its build.gradle file.

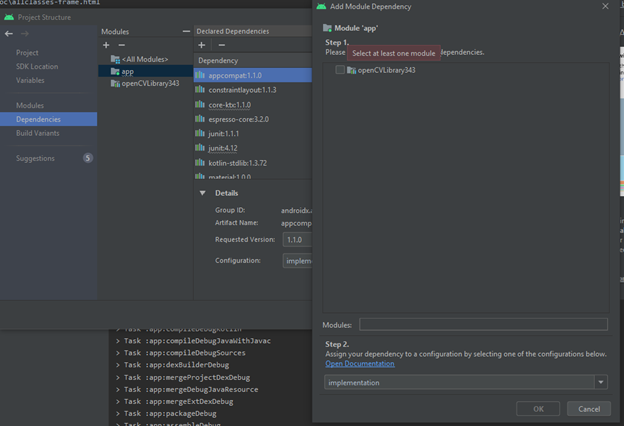
change compileSdkVersion and targetSdkVersion to the app build.gradle versions. After changing the version click on sync.



Step 4: Add the OpenCV Dependency

click on File -> Project Structure.

navigate to module, click on the Dependencies tab click on the app module You should see a green plus button on the far right of the dialog, click on it and select Module dependency. Select the OpenCV library module and click on OK.

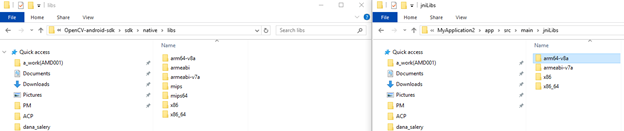


Step 5: Add Native Libraries

On your file explorer, navigate to the folder where you extracted the content of the OpenCV Open the sdk folder and then the native folder. copy the content of the libs folder (4 files: x86, x86\_64 etc.).

right click on the app, in the project view, choose new/folder/jni folder and change it's name to jniLibs.

then paste the 4 files there.

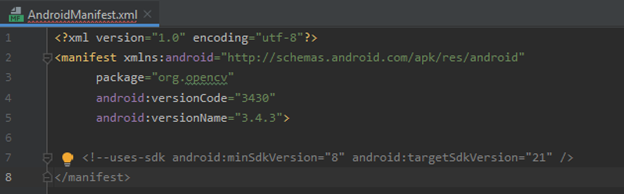


Step 6:

The minSdk version should not be declared in the android manifest file.

Remove the line or comment it:

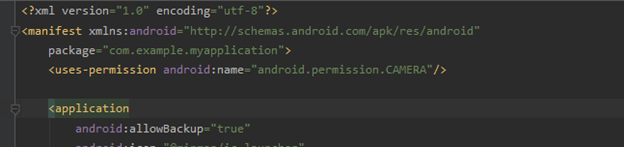
<!--uses-sdk android:minSdkVersion="8" android:targetSdkVersion="21" />



Step 7: Add Required Permissions

Add permissions in the AndroidManifest.xml file, if necessary.

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



\* To compile the Native code from the android studio ide, make use of the cmakelist as is in the repository.

Usage

One can use the ready to use function in the java scope via import, or implement a c++ native code in the native-lib scope.

Add FFMPEG-Mobile to GRADLE

Open the build.gradle (Module:app)

add the following line in the dependency list:

implementation 'com.arthenica:mobile-ffmpeg-full-gpl:4.3.1.LTS'

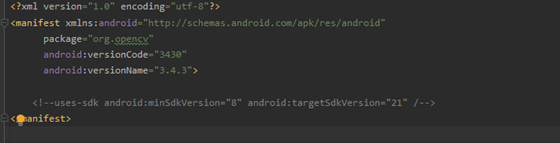
Usage

The usage of the FFMPEG in this project is only in the java scope, via FFMpeg.execute.

see application and api guide here:https://ffmpeg.org/

# comment out the next line in openCVLibrarary343…\AndroidManifest.xml file

uses-sdk android:minSdkVersion="8" android:targetSdkVersion="21"



1. Follow instruction at: <https://medium.com/@SinemKulac/opencv-4-2-0-set-up-for-c-in-android-studio-dc88b0bb34b2>

a. Build gradle (open CV)

android {

compileSdkVersion 29

2. Copy: Main activity.java

a. Keep the first line

b. new Grabber().execute("/storage/emulated/0/Movies/%03d.jpg");

c. TELO // int rc = FFmpeg.execute("-i udp://192.168.10.1:11111 -r 6 -f image2 "+params[0]);

d. int rc = FFmpeg.execute("-i rtsp://wowzaec2demo.streamlock.net/vod/mp4:BigBuckBunny\_115k.mov -f image2 "+params[0]);

3. Copy: Activity\_main.xml (GUI)

4. Copy: Android manifest.xml

a. <uses-permission android:name="android.permission.INTERNET" />

b. <uses-permission android:name="android.permission.RECORD\_AUDIO" />

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

d. <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

Options: app gradle: update from 4.0.0. to 3.63

- classpath "com.android.tools.build:gradle:3.6.3"

WITHOUT JNI

<https://stackoverflow.com/questions/20890116/javacv-android-studio-gradle-possible>

example to run native c++ : <https://github.com/bytedeco/javacv>

# Appendix: Running Gstreamer Android tutorial

Follow installation and setup at: <https://gstreamer.freedesktop.org/documentation/installing/for-android-development.html?gi-language=c>

And:

<https://stackoverflow.com/questions/45044210/gstreamer-examples-in-android-studio/46223465#46223465>

**GStreamer tutorial #5** fail to build. There is a fixed version at: <https://gitlab.com/eduardoprado/gstreamer-tutorial5>

* For that I had to update SDK according to: <https://forums.bignerdranch.com/t/failed-to-find-target-with-hash-string-android-26-in/13827/2>
  + Had to install andriond 8 (Aureo) with api 26
  + **Go to option File->Project Structure**
* Follow tutorial 5 import instruction at: <https://stackoverflow.com/questions/45044210/gstreamer-examples-in-android-studio>
* Issues I faced
  + Had to delete project .idea folder and .iml files. – some Android studio cache issues.
  + Have to upgrade to IDE version 4.4.1
  + Repository – move back to 3.4.1.
  + Use SDK 27 instead of 26

Good guide since 2020!!

<https://stackoverflow.com/questions/57855409/android-studio-error-executing-external-native-build-for-ndkbuild-when-trying-t>

Update from 2020: <https://cgit.freedesktop.org/gstreamer/gst-docs/>

Moving to GStreamer 1.18: <https://gstreamer.freedesktop.org/releases/1.18/>

* Set environment variable: GSTREAMER\_ROOT\_ANDROID – pointing to lib binary.
* I had to shut down and reopen Android studio (Windows version) in order to successfully build the environment.
* Run “Sync Project with Gradle Files “(elephant)
* Push the “Play” button



* I run based on GStreamer tutorial -3

# Appendix: GStreamer camera

sudo apt-get install gstreamer1.0-plugins-bad

sudo apt-get install gstreamer1.0-libav

gst-launch-1.0 udpsrc port=5000 ! h264parse ! avdec\_h264 ! autovideosink

# Appendix: VLC work log

According to a short web survey, there are two main open-source candidates for video grabbing:

* VLC app
* Exoplayer

We also manage to find a lightweight Java app that can grab video frames. (but not from URL streaming)

Optional directions

We identify 3 way to integrate VLC into Android Studio:

* Entire VLC - can be achieved only on Linux environments. No official support under windows (android studio)
  + We managed to generate APK
* Entire VLC on Windows Env.
  + We face an issue with the reference. (as stated above)
* Create an application with only the relevant features (LAN streaming)

Installing VLC

To install VLC we follow the instructions on: <https://wiki.videolan.org/AndroidCompile> There is also a need to install JAVA! - Follow instruction on: <https://itsfoss.com/install-java-ubuntu/>

Note:

* need to run compile.sh before opening Android studio
* need to set Variable JAVA\_HOME, ANDROID\_SDK/NDK
* To do so need to update the .bashrc file as describe on <https://www.youtube.com/watch?v=OcFhTaXj5M0>