**SOURCE CODING**

**AR.CPP**

#include <opencv2/opencv.hpp>

#include <string>

#include <iostream>

#include <vector>

#include <stdio.h>

#include <time.h>

using namespace cv;

using namespace std;

CvFont font;

CvSeq\* face;

IplImage\* hsv;

Point2f p[4],q[4];

IplImage\* img=NULL;

int found, scale = 1;

cv::Point2f v[4],b[4];

unsigned int j=0, i=0;

int line\_type = CV\_AA; // change it to 8 to see non-antialiased graphics

static CvMemStorage\* storage = 0;

CvPoint pt1, pt2,pt3,pt4,pt5,pt6,pt7;

const int boardWidth=5, boardHeight=4;

Mat overlay, perspMat, pic,frame, templ;

static CvHaarClassifierCascade\* cascade = 0;

const Size boardSize = Size(boardWidth,boardHeight);

const char \*cascade\_name="haarcascade\_frontalface\_alt.xml"; //semi-plam

void merge(cv::Mat &in1, cv::Mat &in2,cv::Mat &out)

{

int nLines = in1.rows;

int nc = in1.cols \* in1.channels();

if(in1.isContinuous())

{

nc = nc\*nLines;

nLines = 1;

}

for(int j=0;j<nLines;j++)

{

uchar\* dataIN1 = in1.ptr<uchar>(j); //fajne :)

uchar\* dataIN2 = in2.ptr<uchar>(j);

uchar\* dataOUT = out.ptr<uchar>(j);

for(int i=0;i<nc; i++)

{

if(dataIN2[i] == 0)

{

dataOUT[i] = dataIN1[i];

}

else

{

dataOUT[i] = dataIN2[i];

}

}

}

}

int main()

{

int b=1,x=0,y=0,c=0,c1=0,a=0,w=0,N=0,L=0;

// CvCapture \*capture = cvCaptureFromCAM(0);

CvCapture \*capture = cvCreateFileCapture("rtsp://10.0.0.90:8086/");

cvSetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_WIDTH, 320);

cvSetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_HEIGHT, 240);

cvSetCaptureProperty(capture, CV\_CAP\_PROP\_FPS, 10);

/\* CvSize size = cvSize(

(int)cvGetCaptureProperty( capture,

CV\_CAP\_PROP\_FRAME\_WIDTH),

(int)cvGetCaptureProperty( capture,

CV\_CAP\_PROP\_FRAME\_HEIGHT)

);\*/

// CvVideoWriter \*writer = cvCreateVideoWriter("out.jpg",CV\_FOURCC('M','J','P','G'),10,size);

vector<Point> corners;

Point2f p[4],q[4];

overlay = Mat::zeros(319,239,CV\_8UC3);

storage = cvCreateMemStorage(0);

cvClearMemStorage( storage );

cascade = (CvHaarClassifierCascade\*)cvLoad( cascade\_name, 0, 0, 0 );

pic = cv::imread("N1.png");

flip(pic,pic,0);

flip(pic,pic,1);

while(1)

{

c=c1=0;

img = cvQueryFrame( capture );

if ( !img )

{

printf("Connect Camera.....\n");

break;

}

if ( L == 3 )

{

for( y=0; y<10; y++ )

{

L=0;

img = cvQueryFrame( capture );

cvShowImage( "Frame", img );

cvWaitKey(33);

}

}

Mat src(img);

if(!img)break;

face = cvHaarDetectObjects( img, cascade, storage,1.1, 2, CV\_HAAR\_DO\_CANNY\_PRUNING,cvSize(40, 40));

for( i = 0; i < (face ? face->total : 0); i++ )

{

CvRect\* r = (CvRect\*)cvGetSeqElem( face, i );

found =10;

pt1.x = r->x\*scale+50 ;

pt2.x = (r->x+r->width)\*scale+50;

pt1.y = r->y\*scale ;

pt2.y = (r->y+r->height)\*scale ;

//cvRectangle( img, pt1, pt2, CV\_RGB(0,0,0), 3, 8, 0 );

}

pt7.x=20;pt7.y=10;

cvInitFont( &font, CV\_FONT\_HERSHEY\_COMPLEX, 0.3,0.3 );

cvPutText( img, "Next", pt7, &font, CV\_RGB(255,255,0));

cvRectangle( img, cvPoint(0,0),cvPoint(65,20), CV\_RGB(0,255,0), 2, 8, 0 );

pt7.x=280;pt7.y=14;

cvInitFont( &font, CV\_FONT\_HERSHEY\_COMPLEX, 0.4,0.4 );

cvPutText( img, "Prev", pt7, &font, CV\_RGB(255,255,0));

cvRectangle( img, cvPoint(270,0),cvPoint(320,20), CV\_RGB(0,255,0), 2, 8, 0 );

pt7.x=96;pt7.y=14;

cvInitFont( &font, CV\_FONT\_HERSHEY\_COMPLEX, 0.4,0.4 );

cvPutText( img, "Augment Reality", pt7, &font, CV\_RGB(255,255,0));

pt3.x=65;pt3.y=20;pt4.x=270;pt4.y=20;//hori

cvLine( img, pt3, pt4, CV\_RGB(0,255,255), 2, 8, 0 );//bottom-hor

Mat hsv;

cvtColor(src, hsv, CV\_BGR2HSV);

Mat bw;

inRange(hsv, Scalar(0, 50, 170, 0), Scalar(10, 180, 256, 0), bw);//red

vector<vector<Point> > contours;

findContours(bw.clone(), contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE);

Mat dst = Mat::zeros(src.size(), src.type());

Mat dst1 = Mat::zeros(src.size(), src.type());

drawContours(dst, contours, -1, Scalar::all(255), CV\_FILLED);

unsigned int v1=0,v2=0,v3=0,v4=0,v5=0,v6=0,v7=0,v8=0;

if ( found > 5 )

{

for( y = 8; y < 13; y++ )

{

for( x = 8; x < 55; x++ )

{

a=saturate\_cast<uchar>(( dst.at<Vec3b>(y,x)[0] ));

if ( (a > 230 ) )

{

c =c+1;

c1 =0;

}

}

}

for( y = 8; y < 13; y++ )

{

for( x = 275; x < 317; x++ )

{

w=saturate\_cast<uchar>(( dst.at<Vec3b>(y,x)[0] ));

if ( (w > 230 ) )

{

c1 = c1 + 1;

c = 0;

}

}

}

prev:

if( ( c > 7 ) && ( b == 0 ) )

{

pic = cv::imread("N2.png");

b=b+1;c=0;L=3;

goto Next;

}

if( ( c > 7 ) && ( b == 1 ) )

{

pic = cv::imread("N3.png");

flip(pic,pic,0);

flip(pic,pic,1);

if (N==9){N=0;L=3;c=0;goto Next;}

b=b+1;c=0;L=3;

goto Next;

}

if( ( c > 7 ) && ( b == 2 ) )

{

pic = cv::imread("N4.png");

flip(pic,pic,0);

flip(pic,pic,1);

if (N==9){N=0;L=3;b=b-1;goto Next;}

c=0;L=3;

goto Next;

}

if( ( c1 > 7 ) )

{

b=b-1;

c1=0;

//printf("Prev.............\n");

c=10;

N=9;

goto prev;

}

Next:

found=0;

frame=img;

///printf("argument...........\n");

q[0].x= 0;

q[0].y= 0;

q[1].x= 150;

q[1].y= 0;

q[2].x= 150;

q[2].y= 180;

q[3].x= 0; //center

q[3].y= 180;

v1=pt1.y,v2=pt1.x,v3=pt2.y,v4=pt1.x,v5=pt2.y,v6=pt2.x,v7=pt1.y,v8=pt2.x;

for(i=0;i<corners.size();i++)//find the size of contour

{

cout << "# of corners points: " << corners.size() << endl ;

}

p[0].x= v1;// pt1.x;

p[0].y= v2;//pt1.y;

p[1].x= v3;//pt2.x;

p[1].y= v4;//pt1.y;

p[2].x= v5;//pt2.x;

p[2].y= v6;//pt2.y;

p[3].x= v7;//pt1.x;

p[3].y= v8;//pt2.y;

perspMat = getPerspectiveTransform(q,p);

warpPerspective(pic,overlay,perspMat,Size(frame.cols,frame.rows));

merge(frame,overlay,frame);

}

//cvWriteFrame(writer, img);

cvSaveImage("out.jpg", img);

cvWaitKey( 33);

}

return 0;

}

**Options Activity**

import net.majorkernelpanic.spydroid.api.CustomHttpServer;

import net.majorkernelpanic.spydroid.api.CustomRtspServer;

import net.majorkernelpanic.streaming.SessionBuilder;

import net.majorkernelpanic.streaming.gl.SurfaceView;

import net.majorkernelpanic.streaming.rtsp.RtspServer;

import android.app.AlertDialog;

import android.app.Notification;

import android.app.NotificationManager;

import android.app.PendingIntent;

import android.content.ComponentName;

import android.content.Context;

import android.content.DialogInterface;

import android.content.Intent;

import android.content.ServiceConnection;

import android.content.pm.ActivityInfo;

import android.os.Bundle;

import android.os.IBinder;

import android.os.PowerManager;

import android.support.v4.app.Fragment;

import android.support.v4.app.FragmentActivity;

import android.support.v4.app.FragmentManager;

import android.support.v4.app.FragmentPagerAdapter;

import android.support.v4.app.NotificationCompat;

import android.support.v4.view.MenuItemCompat;

import android.support.v4.view.ViewPager;

import android.util.Log;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.SurfaceHolder;

import android.widget.LinearLayout;

import android.widget.Toast;

/\*\*

\* Spydroid basically launches an RTSP server and an HTTP server,

\* clients can then connect to them and start/stop audio/video streams on the phone.

\*/

public class SpydroidActivity extends FragmentActivity {

static final public String TAG = "SpydroidActivity";

public final int HANDSET = 0x01;

public final int TABLET = 0x02;

// We assume that the device is a phone

public int device = HANDSET;

private ViewPager mViewPager;

private PowerManager.WakeLock mWakeLock;

private SectionsPagerAdapter mAdapter;

private SurfaceView mSurfaceView;

private SpydroidApplication mApplication;

private CustomHttpServer mHttpServer;

private RtspServer mRtspServer;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

mApplication = (SpydroidApplication) getApplication();

setContentView(R.layout.spydroid);

if (findViewById(R.id.handset\_pager) != null) {

// Handset detected !

mAdapter = new SectionsPagerAdapter(getSupportFragmentManager());

mViewPager = (ViewPager) findViewById(R.id.handset\_pager);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);

mSurfaceView = (SurfaceView)findViewById(R.id.handset\_camera\_view);

SessionBuilder.getInstance().setSurfaceView(mSurfaceView);

SessionBuilder.getInstance().setPreviewOrientation(90);

} else {

// Tablet detected !

device = TABLET;

mAdapter = new SectionsPagerAdapter(getSupportFragmentManager());

mViewPager = (ViewPager) findViewById(R.id.tablet\_pager);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE);

SessionBuilder.getInstance().setPreviewOrientation(0);

}

mViewPager.setAdapter(mAdapter);

// Remove the ads if this is the donate version of the app.

if (mApplication.DONATE\_VERSION) {

((LinearLayout)findViewById(R.id.adcontainer)).removeAllViews();

}

// Prevents the phone from going to sleep mode

PowerManager pm = (PowerManager) getSystemService(Context.POWER\_SERVICE);

mWakeLock = pm.newWakeLock(PowerManager.FULL\_WAKE\_LOCK, "net.majorkernelpanic.spydroid.wakelock");

// Starts the service of the HTTP server

//this.startService(new Intent(this,CustomHttpServer.class));

// Starts the service of the RTSP server

this.startService(new Intent(this,CustomRtspServer.class));

}

public void onStart() {

super.onStart();

// Lock screen

mWakeLock.acquire();

// Did the user disabled the notification ?

if (mApplication.notificationEnabled) {

Intent notificationIntent = new Intent(this, SpydroidActivity.class);

PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, PendingIntent.FLAG\_CANCEL\_CURRENT);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this);

Notification notification = builder.setContentIntent(pendingIntent)

.setWhen(System.currentTimeMillis())

.setTicker(getText(R.string.notification\_title))

.setSmallIcon(R.drawable.icon)

.setContentTitle(getText(R.string.notification\_title))

.setContentText(getText(R.string.notification\_content)).build();

notification.flags |= Notification.FLAG\_ONGOING\_EVENT;

((NotificationManager)getSystemService(Context.NOTIFICATION\_SERVICE)).notify(0,notification);

} else {

removeNotification();

}

bindService(new Intent(this,CustomHttpServer.class), mHttpServiceConnection, Context.BIND\_AUTO\_CREATE);

bindService(new Intent(this,CustomRtspServer.class), mRtspServiceConnection, Context.BIND\_AUTO\_CREATE);

}

@Override

public void onStop() {

super.onStop();

// A WakeLock should only be released when isHeld() is true !

if (mWakeLock.isHeld()) mWakeLock.release();

if (mHttpServer != null) mHttpServer.removeCallbackListener(mHttpCallbackListener);

unbindService(mHttpServiceConnection);

if (mRtspServer != null) mRtspServer.removeCallbackListener(mRtspCallbackListener);

unbindService(mRtspServiceConnection);

}

@Override

public void onResume() {

super.onResume();

mApplication.applicationForeground = true;

}

@Override

public void onPause() {

super.onPause();

mApplication.applicationForeground = false;

}

@Override

public void onDestroy() {

Log.d(TAG,"SpydroidActivity destroyed");

super.onDestroy();

}

@Override

public void onBackPressed() {

Intent setIntent = new Intent(Intent.ACTION\_MAIN);

setIntent.addCategory(Intent.CATEGORY\_HOME);

setIntent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

startActivity(setIntent);

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.menu, menu);

MenuItemCompat.setShowAsAction(menu.findItem(R.id.quit), 1);

MenuItemCompat.setShowAsAction(menu.findItem(R.id.options), 1);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

Intent intent;

switch (item.getItemId()) {

case R.id.options:

// Starts QualityListActivity where user can change the streaming quality

intent = new Intent(this.getBaseContext(),OptionsActivity.class);

startActivityForResult(intent, 0);

return true;

case R.id.quit:

quitSpydroid();

return true;

default:

return super.onOptionsItemSelected(item);

}

}

private void quitSpydroid() {

// Removes notification

if (mApplication.notificationEnabled) removeNotification();

// Kills HTTP server

this.stopService(new Intent(this,CustomHttpServer.class));

// Kills RTSP server

this.stopService(new Intent(this,CustomRtspServer.class));

// Returns to home menu

finish();

}

private ServiceConnection mRtspServiceConnection = new ServiceConnection() {

@Override

public void onServiceConnected(ComponentName name, IBinder service) {

mRtspServer = (CustomRtspServer) ((RtspServer.LocalBinder)service).getService();

mRtspServer.addCallbackListener(mRtspCallbackListener);

mRtspServer.start();

}

@Override

public void onServiceDisconnected(ComponentName name) {}

};

private RtspServer.CallbackListener mRtspCallbackListener = new RtspServer.CallbackListener() {

@Override

public void onError(RtspServer server, Exception e, int error) {

// We alert the user that the port is already used by another app.

if (error == RtspServer.ERROR\_BIND\_FAILED) {

new AlertDialog.Builder(SpydroidActivity.this)

.setTitle(R.string.port\_used)

.setMessage(getString(R.string.bind\_failed, "RTSP"))

.setPositiveButton("OK", new DialogInterface.OnClickListener() {

public void onClick(final DialogInterface dialog, final int id) {

startActivityForResult(new Intent(SpydroidActivity.this, OptionsActivity.class),0);

}

})

.show();

}

}

@Override

public void onMessage(RtspServer server, int message) {

if (message==RtspServer.MESSAGE\_STREAMING\_STARTED) {

if (mAdapter != null && mAdapter.getHandsetFragment() != null)

mAdapter.getHandsetFragment().update();

} else if (message==RtspServer.MESSAGE\_STREAMING\_STOPPED) {

if (mAdapter != null && mAdapter.getHandsetFragment() != null)

mAdapter.getHandsetFragment().update();

}

}

};

private ServiceConnection mHttpServiceConnection = new ServiceConnection() {

@Override

public void onServiceConnected(ComponentName name, IBinder service) {

mHttpServer = (CustomHttpServer) ((TinyHttpServer.LocalBinder)service).getService();

mHttpServer.addCallbackListener(mHttpCallbackListener);

mHttpServer.start();

}

@Override

public void onServiceDisconnected(ComponentName name) {}

};

private TinyHttpServer.CallbackListener mHttpCallbackListener = new TinyHttpServer.CallbackListener() {

@Override

public void onError(TinyHttpServer server, Exception e, int error) {

// We alert the user that the port is already used by another app.

if (error == TinyHttpServer.ERROR\_HTTP\_BIND\_FAILED ||

error == TinyHttpServer.ERROR\_HTTPS\_BIND\_FAILED) {

String str = error==TinyHttpServer.ERROR\_HTTP\_BIND\_FAILED?"HTTP":"HTTPS";

new AlertDialog.Builder(SpydroidActivity.this)

.setTitle(R.string.port\_used)

.setMessage(getString(R.string.bind\_failed, str))

.setPositiveButton("OK", new DialogInterface.OnClickListener() {

public void onClick(final DialogInterface dialog, final int id) {

startActivityForResult(new Intent(SpydroidActivity.this, OptionsActivity.class),0);

}

})

.show();

}

}

@Override

public void onMessage(TinyHttpServer server, int message) {

if (message==CustomHttpServer.MESSAGE\_STREAMING\_STARTED) {

if (mAdapter != null && mAdapter.getHandsetFragment() != null)

mAdapter.getHandsetFragment().update();

if (mAdapter != null && mAdapter.getPreviewFragment() != null)

mAdapter.getPreviewFragment().update();

} else if (message==CustomHttpServer.MESSAGE\_STREAMING\_STOPPED) {

if (mAdapter != null && mAdapter.getHandsetFragment() != null)

mAdapter.getHandsetFragment().update();

if (mAdapter != null && mAdapter.getPreviewFragment() != null)

mAdapter.getPreviewFragment().update();

}

}

};

private void removeNotification() {

((NotificationManager)getSystemService(Context.NOTIFICATION\_SERVICE)).cancel(0);

}

public void log(String s) {

Toast.makeText(getApplicationContext(), s, Toast.LENGTH\_SHORT).show();

}

class SectionsPagerAdapter extends FragmentPagerAdapter {

public SectionsPagerAdapter(FragmentManager fm) {

super(fm);

}

@Override

public Fragment getItem(int i) {

if (device == HANDSET) {

switch (i) {

case 0: return new HandsetFragment();

case 1: return new PreviewFragment();

case 2: return new WebviewFragment();

case 3: return new AboutFragment();

}

} else {

switch (i) {

case 0: return new TabletFragment();

case 1: return new AboutFragment();

}

}

return null;

}

@Override

public int getCount() {

return device==HANDSET ? 4 : 2;

}

public HandsetFragment getHandsetFragment() {

if (device == HANDSET) {

return (HandsetFragment) getSupportFragmentManager().findFragmentByTag("android:switcher:"+R.id.handset\_pager+":0");

} else {

return (HandsetFragment) getSupportFragmentManager().findFragmentById(R.id.handset);

}

}

public PreviewFragment getPreviewFragment() {

if (device == HANDSET) {

return (PreviewFragment) getSupportFragmentManager().findFragmentByTag("android:switcher:"+R.id.handset\_pager+":1");

} else {

return (PreviewFragment) getSupportFragmentManager().findFragmentById(R.id.preview);

}

}

@Override

public CharSequence getPageTitle(int position) {

if (device == HANDSET) {

switch (position) {

case 0: return getString(R.string.page0);

case 1: return getString(R.string.page1);

case 2: return getString(R.string.page2);

case 3: return getString(R.string.page3);

}

} else {

switch (position) {

case 0: return getString(R.string.page0);

case 1: return getString(R.string.page2);

}

}

return null;

}

}

}

**Spydroid**

import static org.acra.ReportField.DEVICE\_FEATURES;

import static org.acra.ReportField.LOGCAT;

import static org.acra.ReportField.PHONE\_MODEL;

import static org.acra.ReportField.PRODUCT;

import static org.acra.ReportField.SHARED\_PREFERENCES;

import static org.acra.ReportField.STACK\_TRACE;

import static org.acra.ReportField.USER\_APP\_START\_DATE;

import static org.acra.ReportField.USER\_CRASH\_DATE;

import net.majorkernelpanic.streaming.SessionBuilder;

import net.majorkernelpanic.streaming.video.VideoQuality;

import org.acra.annotation.ReportsCrashes;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.content.IntentFilter;

import android.content.SharedPreferences;

import android.content.SharedPreferences.OnSharedPreferenceChangeListener;

import android.preference.PreferenceManager;

@ReportsCrashes(formKey = "dGhWbUlacEV6X0hlS2xqcmhyYzNrWlE6MQ", customReportContent = { APP\_VERSION\_NAME, PHONE\_MODEL, BRAND, PRODUCT, ANDROID\_VERSION, STACK\_TRACE, USER\_APP\_START\_DATE, USER\_CRASH\_DATE, LOGCAT, DEVICE\_FEATURES, SHARED\_PREFERENCES })

public class SpydroidApplication extends android.app.Application {

public final static String TAG = "ARApplication";

/\*\* Default quality of video streams. \*/

public VideoQuality videoQuality = new VideoQuality(320,240,20,500000);

/\*\* By default AMR is the audio encoder. \*/

public int audioEncoder = SessionBuilder.AUDIO\_AAC;

/\*\* By default H.263 is the video encoder. \*/

public int videoEncoder = SessionBuilder.VIDEO\_H264;

/\*\* Set this flag to true to disable the ads. \*/

public final boolean DONATE\_VERSION = false;

/\*\* If the notification is enabled in the status bar of the phone. \*/

public boolean notificationEnabled = true;

/\*\* The HttpServer will use those variables to send reports about the state of the app to the web interface. \*/

public boolean applicationForeground = true;

public Exception lastCaughtException = null;

/\*\* Contains an approximation of the battery level. \*/

public int batteryLevel = 0;

private static SpydroidApplication sApplication;

@Override

public void onCreate() {

// The following line triggers the initialization of ACRA

// Please do not uncomment this line unless you change the form id or I will receive your crash reports !

//ACRA.init(this);

sApplication = this;

super.onCreate();

SharedPreferences settings = PreferenceManager.getDefaultSharedPreferences(this);

notificationEnabled = settings.getBoolean("notification\_enabled", true);

// On android 3.\* AAC ADTS is not supported so we set the default encoder to AMR-NB, on android 4.\* AAC is the default encoder

audioEncoder = (Integer.parseInt(android.os.Build.VERSION.SDK)<14) ? SessionBuilder.AUDIO\_AMRNB : SessionBuilder.AUDIO\_AAC;

audioEncoder = Integer.parseInt(settings.getString("audio\_encoder", String.valueOf(audioEncoder)));

videoEncoder = Integer.parseInt(settings.getString("video\_encoder", String.valueOf(videoEncoder)));

// Read video quality settings from the preferences

videoQuality = new VideoQuality(

settings.getInt("video\_resX", videoQuality.resX),

settings.getInt("video\_resY", videoQuality.resY),

Integer.parseInt(settings.getString("video\_framerate", String.valueOf(videoQuality.framerate))),

Integer.parseInt(settings.getString("video\_bitrate", String.valueOf(videoQuality.bitrate/1000)))\*1000);

SessionBuilder.getInstance()

.setContext(getApplicationContext())

.setAudioEncoder(!settings.getBoolean("stream\_audio", true)?0:audioEncoder)

.setVideoEncoder(!settings.getBoolean("stream\_video", false)?0:videoEncoder)

.setVideoQuality(videoQuality);

// Listens to changes of preferences

settings.registerOnSharedPreferenceChangeListener(mOnSharedPreferenceChangeListener);

registerReceiver(mBatteryInfoReceiver, new IntentFilter(Intent.ACTION\_BATTERY\_CHANGED));

}

public static SpydroidApplication getInstance() {

return sApplication;

}

private OnSharedPreferenceChangeListener mOnSharedPreferenceChangeListener = new OnSharedPreferenceChangeListener() {

@Override

public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {

if (key.equals("video\_resX") || key.equals("video\_resY")) {

videoQuality.resX = sharedPreferences.getInt("video\_resX", 0);

videoQuality.resY = sharedPreferences.getInt("video\_resY", 0);

}

else if (key.equals("video\_framerate")) {

videoQuality.framerate = Integer.parseInt(sharedPreferences.getString("video\_framerate", "0"));

}

else if (key.equals("video\_bitrate")) {

videoQuality.bitrate = Integer.parseInt(sharedPreferences.getString("video\_bitrate", "0"))\*1000;

}

else if (key.equals("audio\_encoder") || key.equals("stream\_audio")) {

audioEncoder = Integer.parseInt(sharedPreferences.getString("audio\_encoder", String.valueOf(audioEncoder)));

SessionBuilder.getInstance().setAudioEncoder( audioEncoder );

if (!sharedPreferences.getBoolean("stream\_audio", false))

SessionBuilder.getInstance().setAudioEncoder(0);

}

else if (key.equals("stream\_video") || key.equals("video\_encoder")) {

videoEncoder = Integer.parseInt(sharedPreferences.getString("video\_encoder", String.valueOf(videoEncoder)));

SessionBuilder.getInstance().setVideoEncoder( videoEncoder );

if (!sharedPreferences.getBoolean("stream\_video", true))

SessionBuilder.getInstance().setVideoEncoder(0);

}

else if (key.equals("notification\_enabled")) {

notificationEnabled = sharedPreferences.getBoolean("notification\_enabled", true);

}

}

};

private BroadcastReceiver mBatteryInfoReceiver = new BroadcastReceiver() {

@Override

public void onReceive(Context context, Intent intent) {

batteryLevel = intent.getIntExtra("level", 0);

}

};

}

**Media stream**

import net.majorkernelpanic.streaming.rtp.AbstractPacketizer;

import net.majorkernelpanic.streaming.video.VideoStream;

import android.annotation.SuppressLint;

import android.media.MediaCodec;

import android.media.MediaRecorder;

import android.net.LocalServerSocket;

import android.net.LocalSocket;

import android.net.LocalSocketAddress;

import android.util.Log;

/\*\*

\* A MediaRecorder that streams what it records using a packetizer from the rtp package.

\* You can't use this class directly !

\*/

public abstract class MediaStream implements Stream {

protected static final String TAG = "MediaStream";

/\*\* Raw audio/video will be encoded using the MediaRecorder API. \*/

public static final byte MODE\_MEDIARECORDER\_API = 0x01;

/\*\* Raw audio/video will be encoded using the MediaCodec API with buffers. \*/

public static final byte MODE\_MEDIACODEC\_API = 0x02;

/\*\* Raw audio/video will be encoded using the MediaCode API with a surface. \*/

public static final byte MODE\_MEDIACODEC\_API\_2 = 0x05;

/\*\* Prefix that will be used for all shared preferences saved by libstreaming \*/

protected static final String PREF\_PREFIX = "libstreaming-";

/\*\* The packetizer that will read the output of the camera and send RTP packets over the networkd. \*/

protected AbstractPacketizer mPacketizer = null;

protected static byte sSuggestedMode = MODE\_MEDIARECORDER\_API;

protected byte mMode, mRequestedMode;

protected boolean mStreaming = false, mConfigured = false;

protected int mRtpPort = 0, mRtcpPort = 0;

protected InetAddress mDestination;

protected LocalSocket mReceiver, mSender = null;

private LocalServerSocket mLss = null;

private int mSocketId, mTTL = 64;

protected MediaRecorder mMediaRecorder;

protected MediaCodec mMediaCodec;

static {

// We determine wether or not the MediaCodec API should be used

try {

Class.forName("android.media.MediaCodec");

// Will be set to MODE\_MEDIACODEC\_API at some point...

sSuggestedMode = MODE\_MEDIACODEC\_API;

Log.i(TAG,"Phone supports the MediaCoded API");

} catch (ClassNotFoundException e) {

sSuggestedMode = MODE\_MEDIARECORDER\_API;

Log.i(TAG,"Phone does not support the MediaCodec API");

}

}

public MediaStream() {

mRequestedMode = sSuggestedMode;

mMode = sSuggestedMode;

}

/\*\*

\* Sets the destination ip address of the stream.

\* @param dest The destination address of the stream

\*/

public void setDestinationAddress(InetAddress dest) {

mDestination = dest;

}

/\*\*

\* Sets the destination ports of the stream.

\* If an odd number is supplied for the destination port then the next

\* lower even number will be used for RTP and it will be used for RTCP.

\* If an even number is supplied, it will be used for RTP and the next odd

\* number will be used for RTCP.

\* @param dport The destination port

\*/

public void setDestinationPorts(int dport) {

if (dport % 2 == 1) {

mRtpPort = dport-1;

mRtcpPort = dport;

} else {

mRtpPort = dport;

mRtcpPort = dport+1;

}

}

/\*\*

\* Sets the destination ports of the stream.

\* @param rtpPort Destination port that will be used for RTP

\* @param rtcpPort Destination port that will be used for RTCP

\*/

public void setDestinationPorts(int rtpPort, int rtcpPort) {

mRtpPort = rtpPort;

mRtcpPort = rtcpPort;

}

/\*\*

\* Sets the Time To Live of packets sent over the network.

\* @param ttl The time to live

\* @throws IOException

\*/

public void setTimeToLive(int ttl) throws IOException {

mTTL = ttl;

}

/\*\*

\* Returns a pair of destination ports, the first one is the

\* one used for RTP and the second one is used for RTCP.

\*\*/

public int[] getDestinationPorts() {

return new int[] {

mRtpPort,

mRtcpPort

};

}

/\*\*

\* Returns a pair of source ports, the first one is the

\* one used for RTP and the second one is used for RTCP.

\*\*/

public int[] getLocalPorts() {

return new int[] {

this.mPacketizer.getRtpSocket().getLocalPort(),

this.mPacketizer.getRtcpSocket().getLocalPort()

};

}

/\*\*

\* Sets the streaming method that will be used.

\*

\* If the mode is set to {@link #MODE\_MEDIARECORDER\_API}, raw audio/video will be encoded

\* using the MediaRecorder API. <br />

\*

\* If the mode is set to {@link #MODE\_MEDIACODEC\_API} or to {@link #MODE\_MEDIACODEC\_API\_2},

\* audio/video will be encoded with using the MediaCodec. <br />

\*

\* The {@link #MODE\_MEDIACODEC\_API\_2} mode only concerns {@link VideoStream}, it makes

\* use of the createInputSurface() method of the MediaCodec API (Android 4.3 is needed there). <br />

\*

\* @param mode Can be {@link #MODE\_MEDIARECORDER\_API}, {@link #MODE\_MEDIACODEC\_API} or {@link #MODE\_MEDIACODEC\_API\_2}

\*/

public void setStreamingMethod(byte mode) {

mRequestedMode = mode;

}

/\*\*

\* Returns the packetizer associated with the {@link MediaStream}.

\* @return The packetizer

\*/

public AbstractPacketizer getPacketizer() {

return mPacketizer;

}

/\*\*

\* Returns an approximation of the bit rate consumed by the stream in bit per seconde.

\*/

public long getBitrate() {

return !mStreaming ? 0 : mPacketizer.getRtpSocket().getBitrate();

}

/\*\*

\* Indicates if the {@link MediaStream} is streaming.

\* @return A boolean indicating if the {@link MediaStream} is streaming

\*/

public boolean isStreaming() {

return mStreaming;

}

/\*\*

\* Configures the stream with the settings supplied with

\* {@link VideoStream#setVideoQuality(net.majorkernelpanic.streaming.video.VideoQuality)}

\* for a {@link VideoStream} and {@link AudioStream#setAudioQuality(net.majorkernelpanic.streaming.audio.AudioQuality)}

\* for a {@link AudioStream}.

\*/

public synchronized void configure() throws IllegalStateException, IOException {

if (mStreaming) throw new IllegalStateException("Can't be called while streaming.");

mMode = mRequestedMode;

mConfigured = true;

}

/\*\* Starts the stream. \*/

public synchronized void start() throws IllegalStateException, IOException {

if (mDestination==null)

throw new IllegalStateException("No destination ip address set for the stream !");

if (mRtpPort<=0 || mRtcpPort<=0)

throw new IllegalStateException("No destination ports set for the stream !");

mPacketizer.setTimeToLive(mTTL);

if (mMode != MODE\_MEDIARECORDER\_API) {

encodeWithMediaCodec();

} else {

encodeWithMediaRecorder();

}

}

/\*\* Stops the stream. \*/

@SuppressLint("NewApi")

public synchronized void stop() {

if (mStreaming) {

try {

if (mMode==MODE\_MEDIARECORDER\_API) {

mMediaRecorder.stop();

mMediaRecorder.release();

mMediaRecorder = null;

closeSockets();

mPacketizer.stop();

} else {

mPacketizer.stop();

mMediaCodec.stop();

mMediaCodec.release();

mMediaCodec = null;

}

} catch (Exception e) {

e.printStackTrace();

}

mStreaming = false;

}

}

protected abstract void encodeWithMediaRecorder() throws IOException;

protected abstract void encodeWithMediaCodec() throws IOException;

/\*\*

\* Returns a description of the stream using SDP.

\* This method can only be called after {@link Stream#configure()}.

\* @throws IllegalStateException Thrown when {@link Stream#configure()} wa not called.

\*/

public abstract String getSessionDescription();

/\*\*

\* Returns the SSRC of the underlying {@link net.majorkernelpanic.streaming.rtp.RtpSocket}.

\* @return the SSRC of the stream

\*/

public int getSSRC() {

return getPacketizer().getSSRC();

}

protected void createSockets() throws IOException {

final String LOCAL\_ADDR = "net.majorkernelpanic.streaming-";

for (int i=0;i<10;i++) {

try {

mSocketId = new Random().nextInt();

mLss = new LocalServerSocket(LOCAL\_ADDR+mSocketId);

break;

} catch (IOException e1) {}

}

mReceiver = new LocalSocket();

mReceiver.connect( new LocalSocketAddress(LOCAL\_ADDR+mSocketId));

mReceiver.setReceiveBufferSize(500000);

mReceiver.setSoTimeout(3000);

mSender = mLss.accept();

mSender.setSendBufferSize(500000);

}

protected void closeSockets() {

try {

mReceiver.close();

} catch (Exception e) {

e.printStackTrace();

}

try {

mSender.close();

} catch (Exception e) {

e.printStackTrace();

}

try {

mLss.close();

} catch (Exception e) {

e.printStackTrace();

}

mLss = null;

mSender = null;

mReceiver = null;

}

}

**Androidmanifest.xml**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="net.majorkernelpanic.spydroid"

android:versionCode="1000"

android:versionName="10" >

<uses-sdk

android:minSdkVersion="14"

android:targetSdkVersion="19" />

<uses-feature android:name="android.hardware.camera" />

<uses-feature android:name="android.hardware.camera.autofocus" />

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

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

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

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

<uses-permission android:name="android.permission.WAKE\_LOCK" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_WIFI\_MULTICAST\_STATE" />

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

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

<application

android:name="net.majorkernelpanic.spydroid.SpydroidApplication"

android:allowBackup="true"

android:icon="@drawable/icon"

android:label="@string/app\_name" >

<activity

android:name="net.majorkernelpanic.spydroid.ui.SpydroidActivity"

android:label="@string/app\_name"

android:launchMode="singleInstance" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<!-- <activity -->

<!-- android:name="com.google.ads.AdActivity" -->

<!-- android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|uiMode|screenSize|smallestScreenSize" /> -->

<activity

android:name="net.majorkernelpanic.spydroid.ui.OptionsActivity"

android:label="@string/options\_activity\_title" >

</activity>

<service android:name="net.majorkernelpanic.spydroid.api.CustomHttpServer"/>

<service android:name="net.majorkernelpanic.spydroid.api.CustomRtspServer"/>

</application>

</manifest>