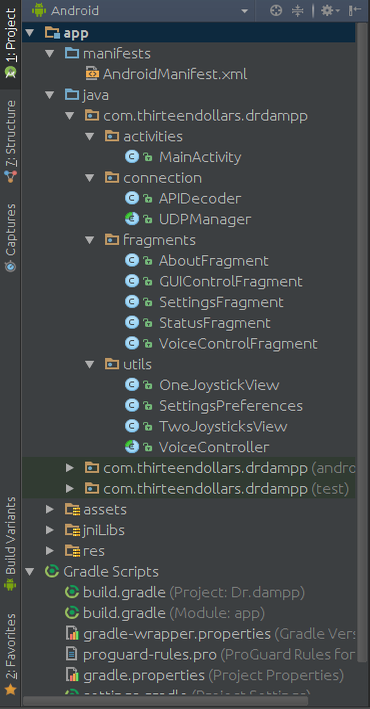
Źródła:

[1] <https://developer.android.com/studio/index.html>

[2] https://docs.oracle.com/javase/8/

[3] http://cmusphinx.sourceforge.net/

Aplikacja mobilna do sterowania pojazdem została przewidziana tylko na platformę Android. Projekt stworzono wykorzystując zintegrowane środowisko programistyczne od firmy Google - Android Studio 2.1.1 [1] oraz obiektowo zorientowany język programowania Java SE8 [2].

W celu lepszego zorganizowania kodu aplikacji, poszczególne klasy umieszczono w odpowiednio nazwanych paczkach, co stworzyło przejrzystą strukturę projektu, znacznie ułatwiając zrozumienie zamysłu aplikacji.

Jak widać na powyższym zrzucie ekranu, wyróżnia się paczki:

-activities

-connection

-fragments

-utils

To w nich umieszczony jest cały kod aplikacji.

Widoczne są również paczki:

**paczka .assets -** to wszystkie dodatki niezbędne dla biblioteki CMU Sphinx [3]

**paczka .jniLibs -** to zewnętrzna biblioteka CMU Sphinx [3]

**paczka .res -** to wszystkie zasoby w postaci grafiki, czy plików XML w których zdefiniowano układy każdego z ekranów jak i menu

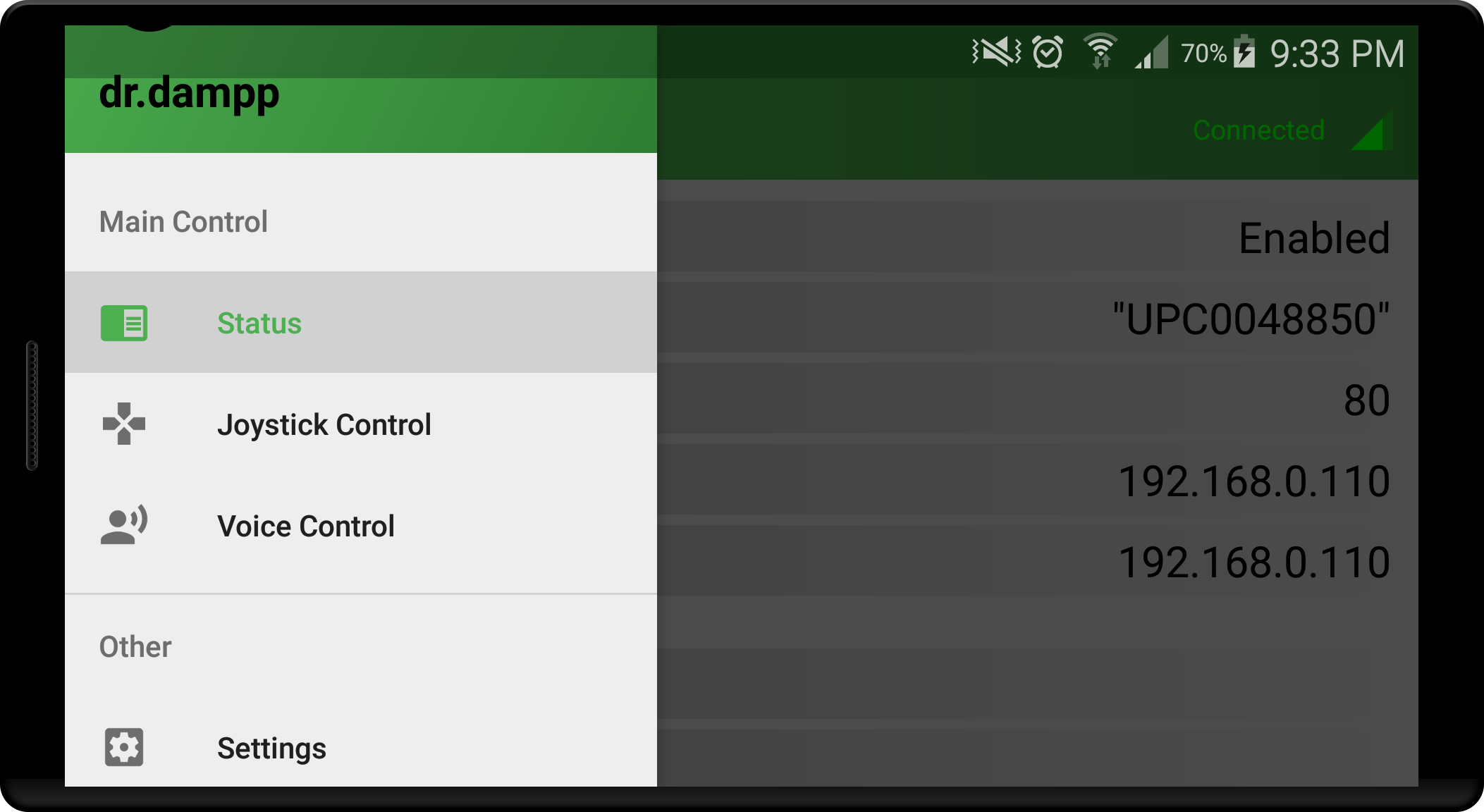
Ich nie będziemy opisywać.

W dalszej części zajmiemy się opisem funkcjonalności poszczególnych klas zawartych w każdej z tych paczek.

**paczka .activities:**

Zawiera tylko jedną klasę - MainActivity.java, która to jest główną klasą aplikacji, odpowiedzialną za inicjalizację wszystkich usług oraz obsługę wszystkich widoków aplikacji - fragmentów.

Klasa aby być Activity dziedziczy po AppCompatActivity, oraz implementuje OnNavigationItemSelectedListener w celu możliwości obsługi menu kontekstowego w formie “wysuwanej szufladki” po lewej stronie aplikacji.



Na zdjęciu widoczna “szufladka” - NavigationDrawer

Na początku kodu nadpisujemy metody klasy AppCompatActivity, dzięki czemu będziemy mogli czuwać nad stanem poszczególnych funkcjonalności aplikacji w przypadku jej uruchomienia,zatrzymania czy też ponownego uruchomienia.

Metoda onCreate() oraz onResume() wywołuję się na początku cyklu życia Activity, przy czym metoda onResume() wywołuję się również w przypadku powrócenia do aplikacji po wcześniejszym jej zminimalizowaniu. W metodach tych ustawiamy nasz główny widok oraz inicjalizujemy wszystkie pola.

W przypadku opuszczenia aplikacji wywoływana funkcja onDestroy(), wstrzymujemy działanie obiektu klasy UDPManager, w celu zaprzestania korzystania z połączenia Wi-Fi, które mogło by przyczynić się do nadmiernego zużycia energii i zasobów procesora oraz zablokowało by możliwość korzystania z tej funkcjonalności innym aplikacjom.

Funkcja initUDP() tworzy obiekt klasy UDPManager implementując jednocześnie metody onResponseSkipped() oraz onDataReceived(), niezbędne w komunikacji dwustronnej oraz przy szacowaniu jakości połączenia

Funkcja initViews() odpowiedzialna jest za inicjalizację wszystkich zmiennych dla widoków aplikacji.

Funkcja onBackPressed() została nadpisana w celu zmiany zachowania przycisku wstecz w przypadku gdy NavigationDrawer jest otwarty. Naciśnięciu przycisku wstecz podczas gdy szufladka jest otwarta powoduje jej schowanie, zamiast całkowitego wyjścia z aplikacji jak by to miało miejsce w przypadku wersji oryginalnej funkcji.

Funkcja onNavigationItemSelected() jest skutkiem implementacji klasy OnNavigationItemSelectedListener i odpowiada za obsługę menu kontekstowego.

Po kliknięciu na element z menu, aktualny widok jest podmieniany na inny za pomocą wcześniej zdefiniowanej już funkcji replaceFragment().

Funkcja replaceFragment() służy do podmiany obecnego widoku na inny.

Funkcje na samym końcu klasy, zaczynające się od przyrostka”get-” to tzw. “gettery”, które to udostępniają prywatne pola tej klasy dla innych obcych obiektów aplikacji. Taktyka ta jest znana dobrze w programowaniu obiektowym jako mechanizm enkapsulacji.

Funkcja setConnectionQuality() ustawia odpowiednią ikonę jakości połączenia na górnym pasku aplikacji - Toolbar

public class MainActivity extends AppCompatActivity

implements NavigationView.OnNavigationItemSelectedListener{

private UDPManager mUDPManager;

private SettingsPreferences mSettings;

private DrawerLayout mDrawer;

private Toolbar mToolbar;

private ImageView mToolbarConnectionIcon;

private TextView mToolbarMainText;

private TextView mToolbarRightText;

private int mSignalStrength; // 0-100%

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

mSettings = new SettingsPreferences(this);

mSignalStrength=0;

initViews();

replaceFragment(R.id.*content*,new StatusFragment(),StatusFragment.*FRAGMENT\_TAG*,null);

}

@Override

protected void onResume() {

super.onResume();

if( mUDPManager==null ) {

initUDP();

}

}

@Override

protected void onDestroy() {

super.onDestroy();

if( mUDPManager!=null) {

mUDPManager.cancel();

mUDPManager=null;

}

}

private void initUDP() {

mToolbarRightText.setText(R.string.*toolbar\_connecting\_mssg*);

try {

mUDPManager = new UDPManager(mSettings.getServerAddress(), mSettings.getServerPort()){

@Override

public void onResponseSkipped(int skippedResponses) {

if( skippedResponses >1 ){

setConnectionQuality(0);

}

Log.*e*(getClass().getName(),"RESPONSE SKIPPED:"+skippedResponses);

}

@Override

public void onDataReceived(byte[] data) {

WifiManager manager = (WifiManager) getSystemService(Context.*WIFI\_SERVICE*);

mSignalStrength = WifiManager.*calculateSignalLevel*( manager.getConnectionInfo().getRssi(),101 );

setConnectionQuality((int) Math.*round*((mSignalStrength) / 25.));

}

};

} catch (UnknownHostException exception) {

Log.*e*(getClass().getName(), exception.toString());

} catch (SocketException exception) {

Log.*e*(getClass().getName(), exception.toString());

}

}

private void initViews() {

//init variables

mToolbar=(Toolbar) findViewById(R.id.*toolbar*);

mDrawer=(DrawerLayout) findViewById(R.id.*drawer\_layout*);

mToolbarConnectionIcon=(ImageView) findViewById(R.id.*toolbar\_connection\_icon*);

mToolbarMainText=(TextView)findViewById(R.id.*toolbar\_maintext*);

mToolbarRightText=(TextView)findViewById(R.id.*toolbar\_righttext*);

//init views

setSupportActionBar(mToolbar);

ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(

this, mDrawer, mToolbar, R.string.*navigation\_drawer\_open*, R.string.*navigation\_drawer\_close*);

mDrawer.setDrawerListener(toggle);

toggle.syncState();

NavigationView navigationView = (NavigationView) findViewById(R.id.*nav\_view*);

navigationView.setNavigationItemSelectedListener(this);

}

@Override

public void onBackPressed() {

if (mDrawer.isDrawerOpen(GravityCompat.*START*)) {

mDrawer.closeDrawer(GravityCompat.*START*);

} else {

super.onBackPressed();

}

}

@Override

public boolean onNavigationItemSelected(MenuItem item) {

// Handle navigation view item clicks here.

int id = item.getItemId();

if (id == R.id.*nav\_status*) {

mToolbarMainText.setText(R.string.*toolbar\_status*);

replaceFragment(R.id.*content*,new StatusFragment(),StatusFragment.*FRAGMENT\_TAG*,null);

} else if (id == R.id.*nav\_joystickcontrol*) {

mToolbarMainText.setText(R.string.*toolbar\_joystick*);

replaceFragment(R.id.*content*,new GUIControlFragment(),GUIControlFragment.*FRAGMENT\_TAG*,null);

} else if (id == R.id.*nav\_voicecontrol*) {

mToolbarMainText.setText(R.string.*toolbar\_voice\_control*);

replaceFragment(R.id.*content*,new VoiceControlFragment(),VoiceControlFragment.*FRAGMENT\_TAG*,null);

} else if (id == R.id.*nav\_settings*) {

mToolbarMainText.setText(R.string.*toolbar\_settings*);

replaceFragment(R.id.*content*,new SettingsFragment(),SettingsFragment.*FRAGMENT\_TAG*,null);

} else if (id == R.id.*nav\_about*) {

mToolbarMainText.setText(R.string.*toolbar\_about*);

replaceFragment(R.id.*content*,new AboutFragment(),AboutFragment.*FRAGMENT\_TAG*,null);

}

mDrawer.closeDrawer(GravityCompat.*START*);

return true;

}

public void replaceFragment(int contaitnerId, Fragment fragment,String fragmentTag,String backStackTag) {

if (backStackTag == null) {

getSupportFragmentManager()

.beginTransaction()

.replace(contaitnerId, fragment, fragmentTag)

.disallowAddToBackStack()

.commit();

} else{

if (getSupportFragmentManager().getBackStackEntryCount()>4){

getSupportFragmentManager().popBackStack(

getSupportFragmentManager().getBackStackEntryAt(0).getId(),

FragmentManager.*POP\_BACK\_STACK\_INCLUSIVE* );

}

getSupportFragmentManager()

.beginTransaction()

.replace(contaitnerId, fragment, fragmentTag)

.addToBackStack(backStackTag)

.commit();

}

}

public Toolbar getToolbar(){

return mToolbar;

}

public UDPManager getConnection(){

return mUDPManager;

}

public SettingsPreferences getSettings(){

return mSettings;

}

public int getSignalStrength() {

return mSignalStrength;

}

private void setConnectionQuality(int signal) {

switch (signal) {

case 4:

mToolbarRightText.setText(R.string.*toolbar\_connected\_mssg*);

mToolbarRightText.setTextColor(Color.*GREEN*);

mToolbarConnectionIcon.setImageResource(R.drawable.*ok\_connection*);

break;

case 3:

mToolbarRightText.setText(R.string.*toolbar\_connected\_mssg*);

mToolbarRightText.setTextColor(Color.*GREEN*);

mToolbarConnectionIcon.setImageResource(R.drawable.*one\_s\_connection*);

break;

case 2:

mToolbarRightText.setText(R.string.*toolbar\_connected\_mssg*);

mToolbarRightText.setTextColor(Color.*GREEN*);

mToolbarConnectionIcon.setImageResource(R.drawable.*two\_s\_connection*);

break;

case 1:

mToolbarRightText.setText(R.string.*toolbar\_connected\_mssg*);

mToolbarRightText.setTextColor(Color.*GREEN*);

mToolbarConnectionIcon.setImageResource(R.drawable.*three\_s\_connection*);

break;

default:

mToolbarRightText.setText(R.string.*toolbar\_disconnected\_mssg*);

mToolbarRightText.setTextColor(ContextCompat.*getColor*(getApplicationContext(),R.color.*darkRed*));

mToolbarConnectionIcon.setImageResource(R.drawable.*no\_connection*);

}

}

**paczka .connection:**

Zawiera dwie klasy, odpowiedzialne za komunikację telefonu z pojazdem.

Są to APIDecoder.java oraz UDPManager.java.

-APIDECODER

Klasa APIDecoder zawiera metody formujące odpowiednie 4 bajtowe paczki zgodnie z specyfikacją API i zwracające je jako tablica bajtów. Klasa ta zawiera również statyczna metodę arePacketsEqual(), która usprawnia porównywanie dwóch paczek do siebie.

public class APIDecoder {

public static final int *MAX\_PACKET\_LENGTH* =4;

private final static byte *TEST\_CONNECTION\_ID*=127;

private final byte SET\_MOTORS\_ID=0;

private final byte SET\_SWITCH\_ID=1;

private final byte SET\_ALL\_SWITCHES\_ID=1;

private final byte LEFT\_ID=2;

private final byte RIGHT\_ID=2;

public byte[] setMotors(int leftLevel, int rightLevel ){

byte[] pack = new byte[*MAX\_PACKET\_LENGTH*];

pack[0]=SET\_MOTORS\_ID;

pack[1]=(byte)leftLevel;

pack[2]=(byte)rightLevel;

return pack;

}

public byte[] setSwitch(int switchIndex, boolean turnOn ){

byte[] pack = new byte[*MAX\_PACKET\_LENGTH*];

pack[0]=SET\_SWITCH\_ID;

pack[1]=(byte)switchIndex;

pack[2]= (byte)( turnOn ? 1:0 );

return pack;

}

public byte[] setAllSwitches( boolean turnOn ){

byte[] pack = new byte[*MAX\_PACKET\_LENGTH*];

pack[0]=SET\_ALL\_SWITCHES\_ID;

pack[1]=-1;

pack[2]= (byte)( turnOn ? 1:0 );

return pack;

}

public byte[] turnLeft( int angle ){

byte[] pack = new byte[*MAX\_PACKET\_LENGTH*];

pack[0]=LEFT\_ID;

pack[1]=0;

pack[2]= (byte) angle;

return pack;

}

public byte[] turnRight( int angle ){

byte[] pack = new byte[*MAX\_PACKET\_LENGTH*];

pack[0]=RIGHT\_ID;

pack[1]=1;

pack[2]= (byte) angle;

return pack;

}

public static byte[] testConnection(){

byte[] pack = new byte[*MAX\_PACKET\_LENGTH*];

pack[0]=*TEST\_CONNECTION\_ID*;

pack[1]=*TEST\_CONNECTION\_ID*;

pack[2]= *TEST\_CONNECTION\_ID*;

pack[3]=*TEST\_CONNECTION\_ID*;

return pack;

}

public static boolean arePacketsEqual(byte[] pack1,byte[] pack2){

if(pack1==null || pack2==null){

return false;

}

if(pack1[0]==pack2[0] &&

pack1[1]==pack2[1] &&

pack1[2]==pack2[2] &&

pack1[3]==pack2[3] ){

return true;

}

else return false;

}

}

-UDPMANAGER

Klasa abstrakcyjna UDPManager odpowiada za ciągłą komunikację za pomocą protokołu UDP. Posiada ona kolejkę do której za pomocą funkcji send() dodawane są kolejne paczki do wysłania. Funkcja send() wyposażona jest również w mechanizm uniemożliwiający dodanie do kolejki dwóch identycznych paczek jedna za drugą - co znacznie poprawia wydajność aplikacji w przypadku gdy zmiana w zachowaniu pojazdu nie jest wymagana,a pojazd mimo to wciąż się porusza. W klasie głównej zagnieżdżono również klasę ConnectionThread która to dziedzicząc po klasie AsyncTask umożliwia nam wykonywanie czynności związanych z wysyłaniem i odbieraniem pakietów jako osobnego wątku - nie blokując przy tym głównego wątku aplikacji. Obiekt klasy ConnectionThread pobiera przygotowaną wczesniej paczkę z kolejki ( o ile nie jest pusta) i wysyła ją bezpośrednio do pojazdu.

Jeśli kolejka jest pusta, proces co jakiś czas wysyła paczki testowe, w celu sprawdzenia jakości połączenia. W momencie wysłania paczki, przełącza się na odbieranie i czeka przez określony czas na odpowiedź z pojazdu czy paczka dotarła. W przypadku gdy komenda została wysłana z powodzeniem, obiekt klasy ConnectionThread uruchamia się od nowa, algorytm się powtarza. Jeśli natomiast odpowiedź z pojazdu nie nadejdzie, użytkownik jest natychmiast o tym informowany za pomocą ikonki utraty połączenia.Klasa główna posiada abstrakcyjne metody onResponseSkipped() i onDataReceived(), które implementowane w klasie MainActivity dają możliwość odpowiedniej reakcji w przypadku otrzymania wiadomości z pojazdu jak i w przypadku braku odpowiedzi od pojazdu.

public abstract class UDPManager {

private final int RESPONSE\_TIMEOUT\_MS = 200;

private InetAddress mServerAddr;

private DatagramSocket mSocket;

private int mPortNumber;

private Queue<byte[]> mRequestsQueue;

//AsyncTask variables

private ConnectionThread mConnection;

private byte[] data;

private static byte[] *lastData*;

private int skippedResponses = 0;

private boolean wasPacketSent = false;

private int timeoutCounter=0;

public UDPManager(String serverAddress, int portNumber) throws UnknownHostException, SocketException {

mServerAddr = InetAddress.*getByName*(serverAddress);

mSocket = new DatagramSocket(portNumber);

mPortNumber = portNumber;

mRequestsQueue = new ConcurrentLinkedQueue<>();

startListening();

send( APIDecoder.*testConnection*() );

}

private void startListening() {

mConnection = new ConnectionThread();

mConnection.execute();

}

public void cancel(){

if(mConnection!=null){

mConnection.cancel(false);

}

}

public void setNewSettings(String serverIpAddress, int serverPortNumber) throws UnknownHostException {

mConnection.cancel(true);

mPortNumber = serverPortNumber;

mServerAddr = InetAddress.*getByName*(serverIpAddress);

startListening();

//test conncection

send( APIDecoder.*testConnection*() );

}

public void send(byte[] dataToSend) {

if( !APIDecoder.*arePacketsEqual*( dataToSend,APIDecoder.*testConnection*() ) && APIDecoder.*arePacketsEqual*(dataToSend,*lastData*) ){

return;

}

else {

*lastData* = dataToSend;

mRequestsQueue.add(dataToSend);

}

}

private class ConnectionThread extends AsyncTask< Void, Void, SocketTimeoutException> {

@Override

protected SocketTimeoutException doInBackground(Void... param) {

DatagramPacket packet;

if( !mRequestsQueue.isEmpty() ){

// send data

timeoutCounter=0;

try {

data = mRequestsQueue.poll();

packet = new DatagramPacket(data, data.length, mServerAddr, mPortNumber);

mSocket.send(packet);

if(!wasPacketSent) {

skippedResponses = 0;

wasPacketSent=true;

}

Log.*d*(getClass().getName(), "Sent data: " + data[0] + " " + data[1] + " " + data[2] + " " + data[3]);

} catch (IOException exception) {

Log.*e*(getClass().getName(), exception.toString());

}

}

// just receive data

timeoutCounter++;

try {

data = new byte[APIDecoder.*MAX\_PACKET\_LENGTH*];

packet = new DatagramPacket(data, data.length);

mSocket.setSoTimeout(RESPONSE\_TIMEOUT\_MS);

mSocket.receive(packet);

wasPacketSent=false;

Log.*d*(getClass().getName(), "Received data: " + data[0] + " " + data[1] + " " + data[2] + " " + data[3]);

} catch (SocketTimeoutException exception) {

if(wasPacketSent){

skippedResponses++;

return exception;

}

else {

data=null;

return null;

}

} catch (SocketException exception) {

Log.*e*(getClass().getName(), exception.toString());

} catch (IOException exception) {

Log.*e*(getClass().getName(), exception.toString());

}

return null;

}

@Override

protected void onPostExecute(SocketTimeoutException exception) {

super.onPostExecute(exception);

if(exception==null){

if(data!=null) {

onDataReceived(data);

}

else if( timeoutCounter>=10 ){

timeoutCounter=0;

send( APIDecoder.*testConnection*() );

}

}

else{

if(skippedResponses>20){

wasPacketSent=false;

}

else {

onResponseSkipped(skippedResponses);

}

}

if( !isCancelled() ) {

startListening();

}

}

}

public abstract void onResponseSkipped(int skippedResponses);

public abstract void onDataReceived(byte[] data);

}

**paczka .fragments:**

Zawiera 5 klas:

-AboutFragment

-GUIControlFragment

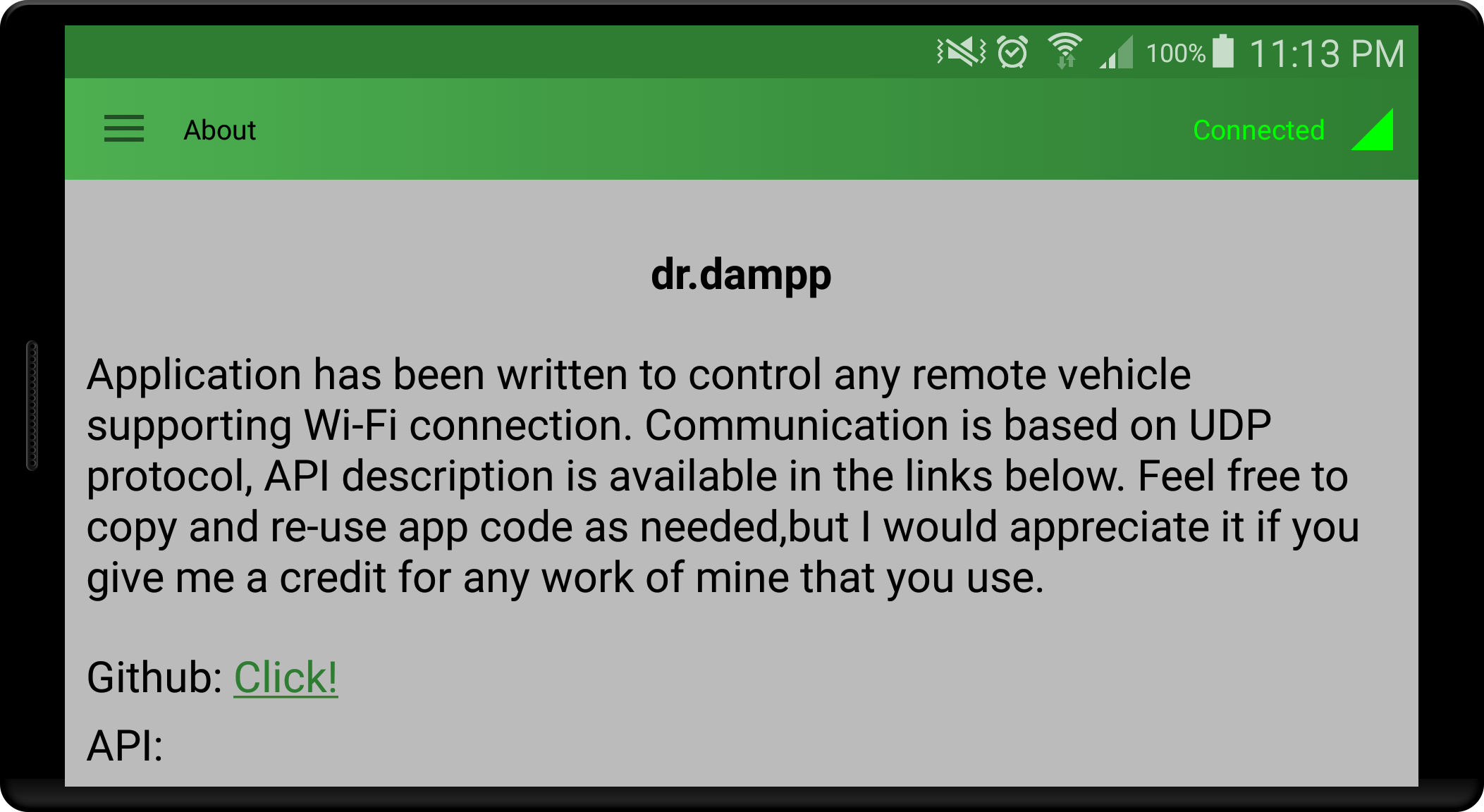
-SettingsFragment

-StatusFragment

-VoiceControlFragment

Każda z powyższych jest odpowiedzialna za utworzenie odpowiedniego widoku oraz jego obsługę. Klasy te dziedziczą po klasie Fragment oraz w niektórych przypadkach implementują stosowne “Listenery” odpowiedzialne za wyłapywanie naciśnięć na przyciski.

-AboutFragment



public class AboutFragment extends Fragment {

public static final String *FRAGMENT\_TAG*="AboutFragmentTag";

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.*about\_layout*,container,false);

TextView githubLink = (TextView) view.findViewById(R.id.*about\_github\_link*);

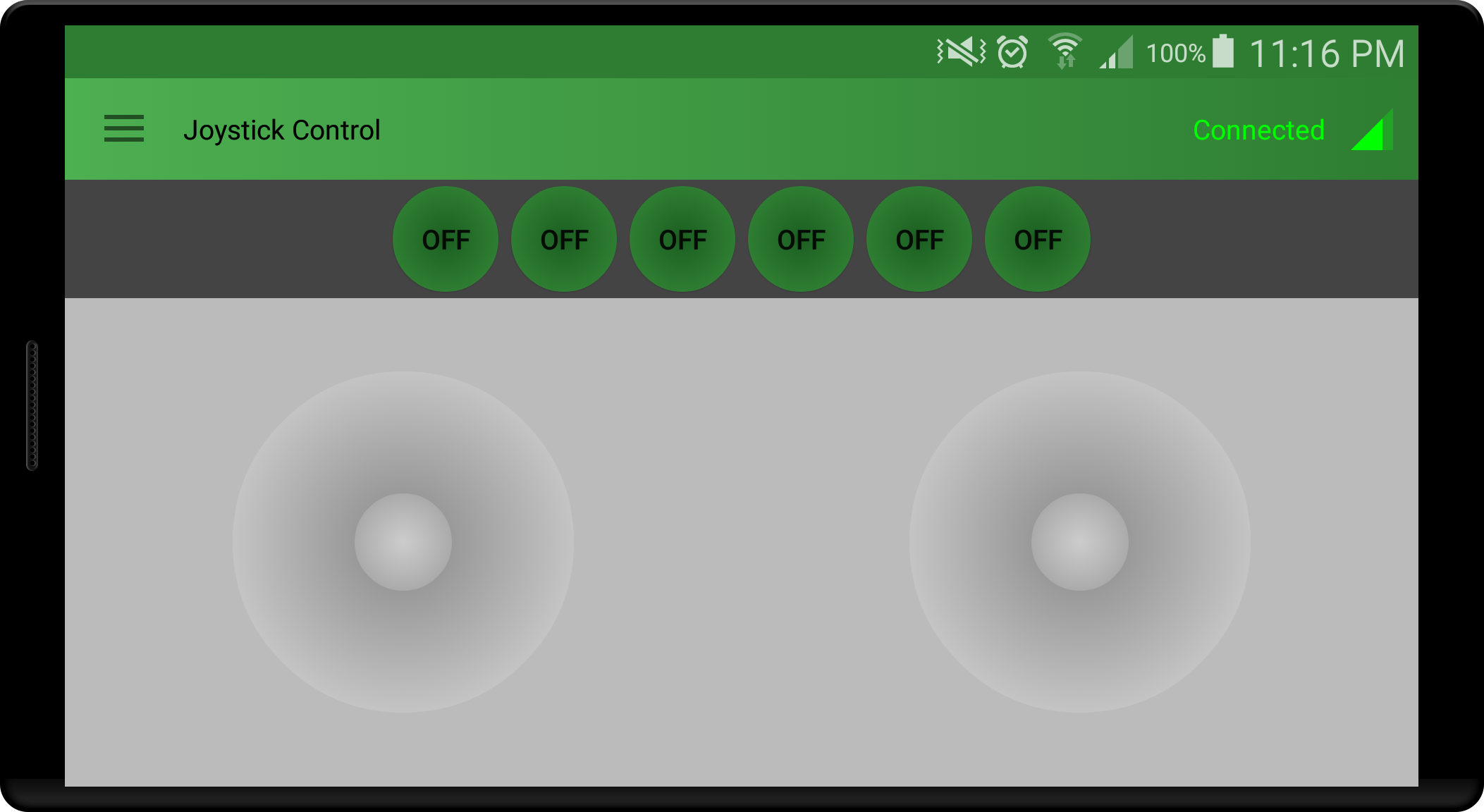
githubLink.setMovementMethod(LinkMovementMethod.*getInstance*());

return view;

}

}

-GUIControlFragment



public class GUIControlFragment extends Fragment implements View.OnClickListener {

public final static String *FRAGMENT\_TAG*="GUIControlFragment";

private UDPManager mConnection;

private APIDecoder mApi;

private ToggleButton mButton0;

private ToggleButton mButton1;

private ToggleButton mButton2;

private ToggleButton mButton3;

private ToggleButton mButton4;

private ToggleButton mButton5;

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

SettingsPreferences settings= new SettingsPreferences( getActivity().getApplicationContext() );

View view= initAppropriateJoystickFromSettings( settings,inflater,container );

initButtonsPanel(view);

return view;

}

@Override

public void onResume() {

super.onResume();

mApi=new APIDecoder();

mConnection= ((MainActivity)getActivity()).getConnection();

}

private void initButtonsPanel(View view) {

mButton0=(ToggleButton)view.findViewById(R.id.*control\_button\_0*);

mButton1=(ToggleButton)view.findViewById(R.id.*control\_button\_1*);

mButton2=(ToggleButton)view.findViewById(R.id.*control\_button\_2*);

mButton3=(ToggleButton)view.findViewById(R.id.*control\_button\_3*);

mButton4=(ToggleButton)view.findViewById(R.id.*control\_button\_4*);

mButton5=(ToggleButton)view.findViewById(R.id.*control\_button\_5*);

mButton0.setOnClickListener(this);

mButton1.setOnClickListener(this);

mButton2.setOnClickListener(this);

mButton3.setOnClickListener(this);

mButton4.setOnClickListener(this);

mButton5.setOnClickListener(this);

}

private View initAppropriateJoystickFromSettings(SettingsPreferences settings,LayoutInflater inflater,ViewGroup container) {

View view=null;

switch ( settings.getJoystickMode() ){

case SettingsPreferences.*ONE\_JOYSTICK\_MODE*:

view = inflater.inflate(R.layout.*gui\_one\_control\_layout*, container,false);

OneJoystickView joystick = (OneJoystickView)view.findViewById(R.id.*gui\_joystick*);

joystick.setOnJoystickMoveListener(new OneJoystickView.OnJoystickMoveListener() {

@Override

public void onValueChanged(int speed, int turn) {

notifyJoystickChange(speed,turn);

}

}, OneJoystickView.*DEFAULT\_LOOP\_INTERVAL*);

break;

case SettingsPreferences.*TWO\_JOYSTICKS\_MODE*:

view = inflater.inflate(R.layout.*gui\_two\_controls\_layout*,container,false);

TwoJoysticksView joysticks = (TwoJoysticksView)view.findViewById(R.id.*gui\_joystick*);

joysticks.setOnJoystickMoveListener(new TwoJoysticksView.OnJoystickMoveListener() {

@Override

public void onValueChanged(int speed, int turn) {

notifyJoystickChange(speed,turn);

}

}, OneJoystickView.*DEFAULT\_LOOP\_INTERVAL*);

break;

}

return view;

}

private void notifyJoystickChange(int speed, int turn) {

int leftLevel;

int rightLevel;

if(speed<0) turn= -turn;

if(speed\*turn>=0){

leftLevel= speed;

rightLevel= speed-(turn/2);

}

else {

leftLevel= speed+(turn/2);

rightLevel= speed;

}

mConnection.send( mApi.setMotors(leftLevel,rightLevel) );

}

@Override

public void onClick(View v) {

byte[] dataToSend=null;

switch ( v.getId() ){

case R.id.*control\_button\_0*: dataToSend=mApi.setSwitch( 0, mButton0.isChecked() ); break;

case R.id.*control\_button\_1*: dataToSend=mApi.setSwitch( 1, mButton1.isChecked() ); break;

case R.id.*control\_button\_2*: dataToSend=mApi.setSwitch( 2, mButton2.isChecked() ); break;

case R.id.*control\_button\_3*: dataToSend=mApi.setSwitch( 3, mButton3.isChecked() ); break;

case R.id.*control\_button\_4*: dataToSend=mApi.setSwitch( 4, mButton4.isChecked() ); break;

case R.id.*control\_button\_5*: dataToSend=mApi.setSwitch( 5, mButton5.isChecked() ); break;

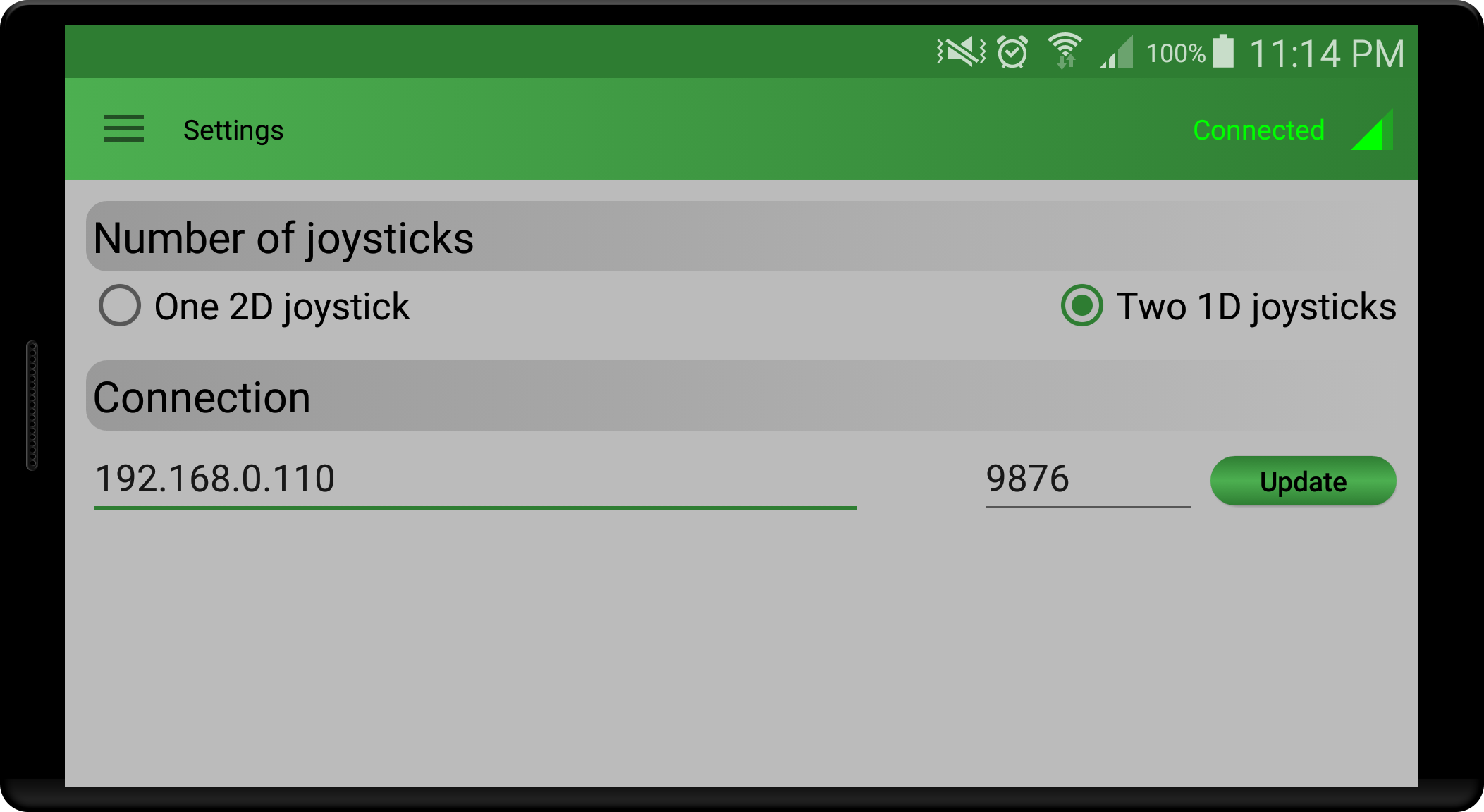
}

mConnection.send(dataToSend);

}

}

-SettingsFragment



public class SettingsFragment extends Fragment implements RadioGroup.OnCheckedChangeListener {

public final static String *FRAGMENT\_TAG*="SettingsFragment";

private RadioGroup mJoystickSettings;

private SettingsPreferences mSettings;

private EditText mIpEditText;

private EditText mPortEditText;

private Button mUpdateConnectionButton;

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

mSettings=new SettingsPreferences( getActivity() );

View view = inflater.inflate(R.layout.*settings\_layout*,container,false);

initViews(view);

return view;

}

private void initViews(View view) {

mJoystickSettings=(RadioGroup)view.findViewById(R.id.*settings\_joystick*);

switch ( mSettings.getJoystickMode() ){

case SettingsPreferences.*ONE\_JOYSTICK\_MODE*: mJoystickSettings.check(R.id.*one\_joystick*); break;

case SettingsPreferences.*TWO\_JOYSTICKS\_MODE*: mJoystickSettings.check(R.id.*two\_joysticks*); break;

}

mJoystickSettings.setOnCheckedChangeListener(this);

mIpEditText= (EditText) view.findViewById(R.id.*settings\_ip\_address*);

mIpEditText.setText( mSettings.getServerAddress() );

mPortEditText= (EditText) view.findViewById(R.id.*settings\_port*);

mPortEditText.setText( mSettings.getServerPort()+"" );

mUpdateConnectionButton= (Button) view.findViewById(R.id.*settings\_save\_conn\_settings*);

mUpdateConnectionButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

updateConnection();

}

});

}

private void updateConnection() {

String ipAdress = mIpEditText.getText().toString();

int portNum = Integer.*parseInt*( mPortEditText.getText().toString() );

// check data correctness

int dotNum=0;

for( char letter : ipAdress.toCharArray() ){

if( letter=='.' ) {

dotNum++;

}

}

if( dotNum!=3 ) {

Toast.*makeText*(getActivity(), "Your ip address is incorrect (ex. 192.168.1.1)",Toast.*LENGTH\_LONG*).show();

return;

}

if( portNum<0 || portNum>65535 ) {

Toast.*makeText*(getActivity(), "Your port number is incorrect (min.0,max.65535)",Toast.*LENGTH\_LONG*).show();

return;

}

//try to change UDPManager settings

UDPManager udp= ((MainActivity)getActivity()).getConnection();

try {

udp.setNewSettings(ipAdress,portNum);

} catch (UnknownHostException e) {

Toast.*makeText*(getActivity(), "Your ip address is incorrect (ex. 192.168.1.1)",Toast.*LENGTH\_LONG*).show();

Log.*e*(getClass().getName(),e.toString() );

return;

}

// if everything is ok save to SharedPreferences

mSettings.setServerAddress(ipAdress);

mSettings.setServerPort(portNum);

//and test connection

udp.send(APIDecoder.*testConnection*() );

Toast.*makeText*(getActivity(), "Updated",Toast.*LENGTH\_SHORT*).show();

}

@Override

public void onCheckedChanged(RadioGroup group, int checkedId) {

if(group.equals(mJoystickSettings)){

switch (checkedId){

case R.id.*one\_joystick*: mSettings.setJoystickMode(SettingsPreferences.*ONE\_JOYSTICK\_MODE*); break;

case R.id.*two\_joysticks*: mSettings.setJoystickMode(SettingsPreferences.*TWO\_JOYSTICKS\_MODE*); break;

}

}

}

}

-StatusFragment



public class StatusFragment extends Fragment{

public static final String *FRAGMENT\_TAG*= "StatusFragmentTag";

private TextView mWifiStatus;

private TextView mSSID;

private TextView mSignal;

private TextView mPhoneIP;

private TextView mServerIP;

private TextView mBattery;

private TextView mTemperature;

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

View view =inflater.inflate(R.layout.*status\_layout*,container,false);

mWifiStatus=(TextView)view.findViewById(R.id.*status\_wifi\_text*);

mSSID=(TextView)view.findViewById(R.id.*status\_ssid\_text*);

mSignal=(TextView)view.findViewById(R.id.*status\_signal\_text*);

mPhoneIP=(TextView)view.findViewById(R.id.*status\_phoneip\_text*);

mServerIP=(TextView)view.findViewById(R.id.*status\_serverip\_text*);

mBattery=(TextView)view.findViewById(R.id.*status\_battery\_text*);

mTemperature=(TextView)view.findViewById(R.id.*status\_temperature\_text*);

return view;

}

@Override

public void onResume() {

super.onResume();

fillViewsWithData();

}

@SuppressWarnings("deprecation")

private void fillViewsWithData() {

WifiManager manager = (WifiManager) getActivity().getSystemService(Context.*WIFI\_SERVICE*);

if( manager.getWifiState()==WifiManager.*WIFI\_STATE\_ENABLED*){

mWifiStatus.setText(R.string.*wifi\_enabled*);

}

else{

mWifiStatus.setText(R.string.*wifi\_disabled*);

}

mSSID.setText( manager.getConnectionInfo().getSSID() );

mSignal.setText( ((MainActivity)getActivity()).getSignalStrength()+"");

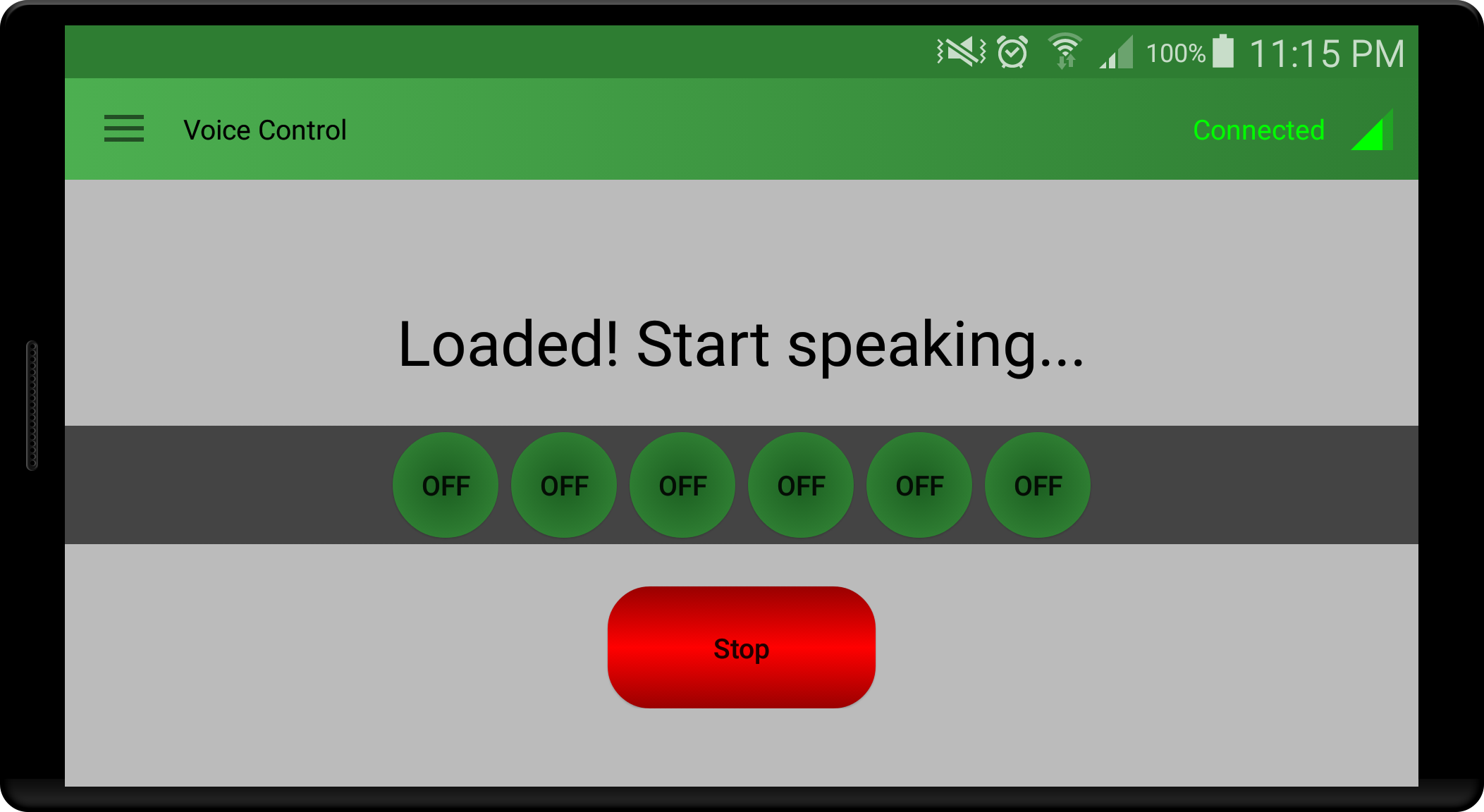
mPhoneIP.setText( Formatter.*formatIpAddress*( manager.getConnectionInfo().getIpAddress() ) );

mServerIP.setText( ((MainActivity)getActivity()).getSettings().getServerAddress() );

}

}

-VoiceControlFragment



public class VoiceControlFragment extends Fragment implements View.OnClickListener {

public final static String *FRAGMENT\_TAG*="VoiceControlFragmentTag";

private final int MOTOR\_SPEED\_CONST\_VALUE = 50; // 0-100%

private final int MOTOR\_TURN\_ANGLE\_CONST\_VALUE = 90; // 0-100%

private UDPManager mUDPManager;

private APIDecoder mApi;

private VoiceController mVoiceController;

private TextView mHeardText;

private Button mStopButton;

private ToggleButton mButton0;

private ToggleButton mButton1;

private ToggleButton mButton2;

private ToggleButton mButton3;

private ToggleButton mButton4;

private ToggleButton mButton5;

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.*voicecontrol\_layout*,container,false);

initViews(view);

return view;

}

@Override

public void onResume() {

super.onResume();

mUDPManager=((MainActivity)getActivity()).getConnection();

mApi= new APIDecoder();

mVoiceController=initVoiceController();

mVoiceController.execute();

}

@Override

public void onPause() {

super.onPause();

if( mVoiceController!=null ){

mVoiceController.cancel(true);

mVoiceController=null;

}

}

@Override

public void onClick(View v) {

byte[] dataToSend=null;

switch ( v.getId() ){

case R.id.*voicecontrol\_stop\_button*:

dataToSend=mApi.setMotors(0,0);

mHeardText.setText(R.string.*command\_stop*);

break;

case R.id.*control\_button\_0*: dataToSend=mApi.setSwitch( 0, mButton0.isChecked() ); break;

case R.id.*control\_button\_1*: dataToSend=mApi.setSwitch( 1, mButton1.isChecked() ); break;

case R.id.*control\_button\_2*: dataToSend=mApi.setSwitch( 2, mButton2.isChecked() ); break;

case R.id.*control\_button\_3*: dataToSend=mApi.setSwitch( 3, mButton3.isChecked() ); break;

case R.id.*control\_button\_4*: dataToSend=mApi.setSwitch( 4, mButton4.isChecked() ); break;

case R.id.*control\_button\_5*: dataToSend=mApi.setSwitch( 5, mButton5.isChecked() ); break;

}

mUDPManager.send(dataToSend);

}

private void initViews(View view){

//init

mHeardText = (TextView) view.findViewById(R.id.*voicecontrol\_heard\_text*);

mStopButton = (Button) view.findViewById(R.id.*voicecontrol\_stop\_button*);

mButton0=(ToggleButton)view.findViewById(R.id.*control\_button\_0*);

mButton1=(ToggleButton)view.findViewById(R.id.*control\_button\_1*);

mButton2=(ToggleButton)view.findViewById(R.id.*control\_button\_2*);

mButton3=(ToggleButton)view.findViewById(R.id.*control\_button\_3*);

mButton4=(ToggleButton)view.findViewById(R.id.*control\_button\_4*);

mButton5=(ToggleButton)view.findViewById(R.id.*control\_button\_5*);

// set OnClickListeners

mStopButton.setOnClickListener(this);

mButton0.setOnClickListener(this);

mButton1.setOnClickListener(this);

mButton2.setOnClickListener(this);

mButton3.setOnClickListener(this);

mButton4.setOnClickListener(this);

mButton5.setOnClickListener(this);

}

private VoiceController initVoiceController() {

mHeardText.setText(R.string.*voicecontrol\_initializing*);

if(mVoiceController!=null){

mVoiceController.cancel(true);

}

VoiceController controller = new VoiceController( getActivity().getApplicationContext() ) {

@Override

public void initSuccess() {

mHeardText.setText(R.string.*voicecontroller\_init\_success*);

}

@Override

public void initFailed(Exception result) {

mHeardText.setText(R.string.*voicecontrol\_failed*);

Log.*e*(getClass().getName(),result.toString());

}

@Override

public void onWordSaid(String text) {

parseCommand(text);

}

@Override

public void onError(Exception e) {

mHeardText.setText(R.string.*voicecontrol\_error*);

Log.*e*(getClass().getName(),e.toString());

}

};

return controller;

}

private void parseCommand(String text) {

switch (text){

case VoiceController.*GO\_AHEAD*:

mUDPManager.send( mApi.setMotors(MOTOR\_SPEED\_CONST\_VALUE,MOTOR\_SPEED\_CONST\_VALUE));

mHeardText.setText(R.string.*command\_goahead*);

break;

case VoiceController.*GO\_BACK*:

mUDPManager.send( mApi.setMotors(-MOTOR\_SPEED\_CONST\_VALUE,-MOTOR\_SPEED\_CONST\_VALUE));

mHeardText.setText(R.string.*command\_back*);

break;

case VoiceController.*TURN\_LEFT*:

mUDPManager.send( mApi.turnLeft(MOTOR\_TURN\_ANGLE\_CONST\_VALUE));

mHeardText.setText(R.string.*command\_left*);

break;

case VoiceController.*TURN\_RIGHT*:

mUDPManager.send( mApi.turnRight(MOTOR\_TURN\_ANGLE\_CONST\_VALUE));

mHeardText.setText(R.string.*command\_right*);

break;

default:

mUDPManager.send( mApi.setMotors(0,0) );

mHeardText.setText(R.string.*command\_stop*);

}

}

}

**paczka .utils:**

Zawiera 4 klasy, odpowiedzialne za wszystkie dodatkowe funkcjonalności aplikacji takie jak widoki joysticków, zapamiętywanie ustawień użytkownika oraz sterowanie głosem.komunikację telefonu z pojazdem. Klasy widoków joysticków to :

-OneJoystickView

-TwoJoysticksView

Klasa odpowiedzialna za zapisywanie ustawień w trwałej pamięci telefonu to klasa SettingsPreferences.

Sterowanie głosowe zaimplementowane jest w klasie VoiceController i opiera się ono na zewnętrznej bibliotece CMU Sphinx[3].

-OneJoystickView

public class OneJoystickView extends View implements Runnable {

public final static long *DEFAULT\_LOOP\_INTERVAL* = 300; // ms

protected OnJoystickMoveListener onJoystickMoveListener;

protected Thread thread = new Thread(this);

protected long loopInterval = *DEFAULT\_LOOP\_INTERVAL*;

private int xPosition = 0; // Touch x position

private int yPosition = 0; // Touch y position

private float centerX = 0; // Center view x position

private float centerY = 0; // Center view y position

private Paint mainCircle;

private Paint button;

private int joystickRadius;

private int buttonRadius;

public OneJoystickView(Context context, AttributeSet attrs) {

super(context, attrs);

initJoystickViews();

}

public OneJoystickView(Context context, AttributeSet attrs, int defaultStyle) {

super(context, attrs, defaultStyle);

initJoystickViews();

}

protected void initJoystickViews() {

mainCircle = new Paint(Paint.*ANTI\_ALIAS\_FLAG*);

mainCircle.setStyle(Paint.Style.*FILL\_AND\_STROKE*);

button = new Paint(Paint.*ANTI\_ALIAS\_FLAG*);

button.setStyle(Paint.Style.*FILL\_AND\_STROKE*);

}

@Override

protected void onSizeChanged(int xNew, int yNew, int xOld, int yOld) {

super.onSizeChanged(xNew, yNew, xOld, yOld);

// before measure, get the center of views

xPosition = getWidth() / 2;

yPosition = getWidth() / 2;

int d = Math.*min*(xNew, yNew);

buttonRadius = (int) (d / 2 \* 0.20);

joystickRadius = (int) (d / 2 \* 0.75);

}

@Override

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

// setting the measured values to resize the view to a certain width and

// height

int d = Math.*min*(measure(widthMeasureSpec), measure(heightMeasureSpec));

setMeasuredDimension(d, d);

}

protected int measure(int measureSpec) {

int result = 0;

// Decode the measurement specifications.

int specMode = MeasureSpec.*getMode*(measureSpec);

int specSize = MeasureSpec.*getSize*(measureSpec);

if (specMode == MeasureSpec.*UNSPECIFIED*) {

// Return a default size of 200 if no bounds are specified.

result = 200;

} else {

// As you want to fill the available space

// always return the full available bounds.

result = specSize;

}

return result;

}

@Override

protected void onDraw(Canvas canvas) {

centerX = (getWidth()) / 2;

centerY = (getHeight()) / 2;

// painting the main circle

mainCircle.setShader(new RadialGradient(xPosition, yPosition,joystickRadius\*2,Color.*GRAY*, Color.*WHITE*, Shader.TileMode.*CLAMP*));

canvas.drawCircle(centerX, centerY, joystickRadius,mainCircle);

// painting the move button

button.setShader(new RadialGradient(xPosition, yPosition,buttonRadius\*2, getResources().getColor(R.color.*joystick\_button\_grey*), Color.*GRAY*, Shader.TileMode.*CLAMP*));

canvas.drawCircle(xPosition, yPosition, buttonRadius, button);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

invalidate();

xPosition = (int) event.getX();

yPosition = (int) event.getY();

double abs = Math.*sqrt*((xPosition - centerX) \* (xPosition - centerX)

+ (yPosition - centerY) \* (yPosition - centerY));

if (abs > joystickRadius) {

xPosition = (int) ((xPosition - centerX) \* joystickRadius / abs + centerX);

yPosition = (int) ((yPosition - centerY) \* joystickRadius / abs + centerY);

}

if (event.getAction() == MotionEvent.*ACTION\_UP*) {

xPosition = (int) centerX;

yPosition = (int) centerY;

thread.interrupt();

if (onJoystickMoveListener != null)

sendCurrentControllerPosition();

}

if (onJoystickMoveListener != null && event.getAction() == MotionEvent.*ACTION\_DOWN*) {

if (thread != null && thread.isAlive()) {

thread.interrupt();

}

thread = new Thread(this);

thread.start();

if (onJoystickMoveListener != null)

sendCurrentControllerPosition();

}

return true;

}

protected void sendCurrentControllerPosition() {

int speedValue = (int) ( ( (yPosition-centerY)/joystickRadius ) \*100);

int turnValue = (int) ( ( (xPosition-centerX)/joystickRadius ) \*100);

onJoystickMoveListener.onValueChanged( -speedValue,turnValue );

}

public void setOnJoystickMoveListener(OnJoystickMoveListener listener,long repeatInterval) {

this.onJoystickMoveListener = listener;

this.loopInterval = repeatInterval;

}

public interface OnJoystickMoveListener {

void onValueChanged(int speed, int turn);

}

@Override

public void run() {

while (!Thread.*interrupted*()) {

post(new Runnable() {

public void run() {

if (onJoystickMoveListener != null){

sendCurrentControllerPosition();

}

}

});

try {

Thread.*sleep*(loopInterval);

} catch (InterruptedException e) {

break;

}

}

}

}

-TwoJoysticksView

public class TwoJoysticksView extends View implements Runnable {

public final static long *DEFAULT\_LOOP\_INTERVAL* = 300; //ms

private OnJoystickMoveListener onJoystickMoveListener;

private Thread thread = new Thread(this);

private long loopInterval = *DEFAULT\_LOOP\_INTERVAL*;

// Speed joystick

private float ySpeedPosition = 0;

private float speedCenterX = 0;

private float speedCenterY = 0;

private Paint speedMainCircle;

private Paint speedMainButton;

// Turn joystick

private float xTurnPosition = 0;

private float turnCenterX = 0;

private float turnCenterY = 0;

private Paint turnMainCircle;

private Paint turnMainButton;

// Common settings

private int joysticksRadius;

private int buttonsRadius;

public TwoJoysticksView(Context context, AttributeSet attrs) {

super(context, attrs);

initJoystickView();

}

public TwoJoysticksView(Context context, AttributeSet attrs, int defaultStyle) {

super(context, attrs, defaultStyle);

initJoystickView();

}

protected void initJoystickView() {

// speed joystick

speedMainCircle = new Paint(Paint.*ANTI\_ALIAS\_FLAG*);

speedMainCircle.setStyle(Paint.Style.*FILL\_AND\_STROKE*);

speedMainButton = new Paint(Paint.*ANTI\_ALIAS\_FLAG*);

speedMainButton.setStyle(Paint.Style.*FILL\_AND\_STROKE*);

// turn joystick

turnMainCircle = new Paint(Paint.*ANTI\_ALIAS\_FLAG*);

turnMainCircle.setStyle(Paint.Style.*FILL\_AND\_STROKE*);

turnMainButton = new Paint(Paint.*ANTI\_ALIAS\_FLAG*);

turnMainButton.setStyle(Paint.Style.*FILL\_AND\_STROKE*);

}

@Override

protected void onSizeChanged(int xNew, int yNew, int xOld, int yOld) {

super.onSizeChanged(xNew, yNew, xOld, yOld);

// before measure, get the center of views

ySpeedPosition = getHeight() / 2;

xTurnPosition = 3\*((getWidth()) / 4 );

int d = Math.*min*(xNew, yNew);

buttonsRadius = (int) (d / 2 \* 0.20);

joysticksRadius = (int) (d / 2 \* 0.70);

}

@Override

protected void onDraw(Canvas canvas) {

// set size

speedCenterX = (getWidth()) / 4;

speedCenterY = (getHeight()) / 2;

turnCenterX = 3\*((getWidth()) / 4 );

turnCenterY = (getHeight()) / 2;

// paint main circles

speedMainCircle.setShader(new RadialGradient(speedCenterX, ySpeedPosition,joysticksRadius\*2,Color.*GRAY*, Color.*WHITE*, Shader.TileMode.*CLAMP*));

canvas.drawCircle(speedCenterX, speedCenterY, joysticksRadius, speedMainCircle);

turnMainCircle.setShader(new RadialGradient(xTurnPosition, turnCenterY,joysticksRadius\*2,Color.*GRAY*, Color.*WHITE*, Shader.TileMode.*CLAMP*));

canvas.drawCircle(turnCenterX, turnCenterY, joysticksRadius, turnMainCircle);

// paint move buttons

speedMainButton.setShader(new RadialGradient(speedCenterX, ySpeedPosition,buttonsRadius\*2, getResources().getColor(R.color.*joystick\_button\_grey*), Color.*GRAY*, Shader.TileMode.*CLAMP*));

canvas.drawCircle(speedCenterX, ySpeedPosition, buttonsRadius, speedMainButton);

turnMainButton.setShader(new RadialGradient(xTurnPosition, turnCenterY,buttonsRadius\*2, getResources().getColor(R.color.*joystick\_button\_grey*), Color.*GRAY*, Shader.TileMode.*CLAMP*));

canvas.drawCircle(xTurnPosition, turnCenterY, buttonsRadius, turnMainButton);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

invalidate();

// if stop touching

if (event.getAction() == MotionEvent.*ACTION\_UP*) {

ySpeedPosition = speedCenterY;

xTurnPosition = turnCenterX;

thread.interrupt();

if (onJoystickMoveListener != null) {

onJoystickMoveListener.onValueChanged(0, 0);

}

return true;

}

// if touched first time

if (onJoystickMoveListener != null && event.getAction() == MotionEvent.*ACTION\_DOWN*) {

if (thread != null && thread.isAlive()) {

thread.interrupt();

}

//if touched in joystick range

if( ( Math.*abs*( turnCenterX - event.getX() ) <= joysticksRadius &&

Math.*abs*( turnCenterY - event.getY() ) <= joysticksRadius ) ||

(Math.*abs*( speedCenterX - event.getX() ) <= joysticksRadius &&

Math.*abs*( speedCenterY - event.getY() ) <= joysticksRadius ) ) {

thread = new Thread(this);

thread.start();

}

if (onJoystickMoveListener != null ){

sendCurrentControllerPosition();

}

return true;

}

// if only one finger

if( event.getPointerCount()<=1 ){

// if finger is on speed joystick

if( Math.*abs*( speedCenterX - event.getX() ) <= joysticksRadius &&

Math.*abs*( speedCenterY - event.getY() ) <= joysticksRadius ) {

ySpeedPosition= event.getY();

xTurnPosition=turnCenterX;

}

// if finger is on turn joystick

else if( Math.*abs*( turnCenterX - event.getX() ) <= joysticksRadius &&

Math.*abs*( turnCenterY - event.getY() ) <= joysticksRadius ) {

xTurnPosition= event.getX();

ySpeedPosition= speedCenterY;

}

else{

xTurnPosition= turnCenterX;

ySpeedPosition= speedCenterY;

}

}

// if more than one finger

else{

for(int index=0; index<event.getPointerCount(); index++ ){

// if finger is on speed joystick, set speed value

if( Math.*abs*( speedCenterX - event.getX(index) ) <= joysticksRadius &&

Math.*abs*( speedCenterY - event.getY(index) ) <= joysticksRadius ) {

ySpeedPosition= event.getY(index);

}

// else if finger is on turn joystick, set turn value;

else if( Math.*abs*( turnCenterX - event.getX(index) ) <= joysticksRadius &&

Math.*abs*( turnCenterY - event.getY(index) ) <= joysticksRadius ) {

xTurnPosition= event.getX(index);

}

// if not in range,reset

else{

xTurnPosition= turnCenterX;

ySpeedPosition= speedCenterY;

}

}

}

return true;

}

private void sendCurrentControllerPosition() {

int speedValue = (int) ( ( (ySpeedPosition-speedCenterY)/joysticksRadius ) \*100);

int turnValue = (int) ( ( (xTurnPosition-turnCenterX)/joysticksRadius ) \*100);

onJoystickMoveListener.onValueChanged( -speedValue,turnValue );

}

public void setOnJoystickMoveListener(OnJoystickMoveListener listener,long repeatInterval) {

this.onJoystickMoveListener = listener;

this.loopInterval = repeatInterval;

}

public interface OnJoystickMoveListener {

void onValueChanged(int speed, int turn);

}

@Override

public void run() {

while (!Thread.*interrupted*()) {

post(new Runnable() {

public void run() {

if (onJoystickMoveListener != null){

sendCurrentControllerPosition();

}

}

});

try {

Thread.*sleep*(loopInterval);

} catch (InterruptedException e) {

break;

}

}

}

}

-SettingsPreferences

public class SettingsPreferences {

private final String PREFERENCES\_TAG = "SharedPreferencesTag";

//GUI controller settings

private final String JOYSTICK\_SETTINGS = "GUIControllerSettings";

public static final int *TWO\_JOYSTICKS\_MODE* = 1;

public static final int *ONE\_JOYSTICK\_MODE* = 2;

//Connection settings

private final String IP\_ADDRESS\_SETTINGS = "IpAddressSettings";

private final String PORT\_NUMBER\_SETTINGS = "PortNumberSettings";

private final String DEFAULT\_SERVER\_IP\_ADDRESS = "192.168.0.110";

private final int DEFAULT\_SERVER\_PORT = 9876;

private SharedPreferences mPreferences;

private SharedPreferences.Editor mEditor;

public SettingsPreferences(Context context) {

mPreferences= context.getSharedPreferences(PREFERENCES\_TAG,Context.*MODE\_PRIVATE*);

mEditor = mPreferences.edit();

}

public int getJoystickMode(){

return mPreferences.getInt(JOYSTICK\_SETTINGS,*TWO\_JOYSTICKS\_MODE*);

}

public void setJoystickMode(int MODE){

mEditor.putInt(JOYSTICK\_SETTINGS,MODE).commit();

}

public int getServerPort(){

return mPreferences.getInt(PORT\_NUMBER\_SETTINGS,DEFAULT\_SERVER\_PORT);

}

public void setServerPort(int port){

mEditor.putInt(PORT\_NUMBER\_SETTINGS,port).commit();

}

public String getServerAddress(){

return mPreferences.getString(IP\_ADDRESS\_SETTINGS,DEFAULT\_SERVER\_IP\_ADDRESS);

}

public void setServerAddress(String ip){

mEditor.putString(IP\_ADDRESS\_SETTINGS,ip).commit();

}

}

-VoiceController

public abstract class VoiceController extends AsyncTask<Void,Void,Exception> implements RecognitionListener {

public static final String *GO\_AHEAD*="goahead";

public static final String *GO\_BACK*="goback";

public static final String *TURN\_LEFT*="left";

public static final String *TURN\_RIGHT*="right";

public static final String *STOP*="stop";

private Context mContext;

private SpeechRecognizer recognizer;

public VoiceController(Context context) {

mContext=context;

}

public abstract void initSuccess();

public abstract void initFailed(Exception result);

public abstract void onWordSaid(String text);

@Override

protected void onCancelled() {

super.onCancelled();

}

@Override

protected Exception doInBackground(Void... params) {

try {

Assets assets = new Assets(mContext);

File assetDir = assets.syncAssets();

setupRecognizer(assetDir);

} catch (IOException e) {

return e;

}

return null;

}

@Override

protected void onPostExecute(Exception result) {

if (result != null) {

initFailed(result);

} else {

initSuccess();

startListeningAgain();

}

}

@Override

public void onPartialResult(Hypothesis hypothesis) {

if (hypothesis == null) {

return;

}

String text = hypothesis.getHypstr();

int space = text.indexOf(' ');

text = text.substring(0, space);

onWordSaid(text);

}

@Override

public void onResult(Hypothesis hypothesis) {

}

@Override

public void onBeginningOfSpeech() {

}

@Override

public void onEndOfSpeech() {

startListeningAgain();

}

private void startListeningAgain() {

recognizer.stop();

if( !isCancelled() ) {

recognizer.startListening("menu");

}

}

private void setupRecognizer(File assetsDir) throws IOException {

recognizer = *defaultSetup*()

.setAcousticModel(new File(assetsDir, "en-us-ptm"))

.setDictionary(new File(assetsDir, "8966.dict"))

// Use context-independent phonetic search, context-dependent is too slow for mobile

.setBoolean("-allphone\_ci", true)

.getRecognizer();

recognizer.addListener(this);

File menuGrammar = new File(assetsDir, "menu.gram");

recognizer.addKeywordSearch("menu", menuGrammar);

}

@Override

public void onTimeout() {

startListeningAgain();

}

}