mobile tracking car

System Architecture & Design, SW

Multidisciplinary Project for the Applied App Development courses (ITIPRJ)

Supervisor: MSc. Torben Gregersen

Group: AAD1

Editor: Berger Andreas, Pfister Michael, François Grzybowski, Constantin Nivet, Noel Peña, Jorge Mañanas Fernández

December 9, 2014

Contents

Document History

Version	Date	Initials	Description
1.0	25.11.2014	**JM**	start of the document
1.1	28.11.2014	**JM**	finished system overview and architectural view
1.2	29.11.2014	**NP**	logical view
1.3	29.11.2014	**NP**	finished process view
1.4	02.12.2014	**NP**	finished document for skimming
1.5	09.12.2014	**BA** & **PM**	finished document for hand-in

Table 1: Document History

Approval

Author(s):	Berger Andreas - [**BA**]
	Pfister Michael - [**PM**]
	Jorge Mañanas Fernández - [**JM**]
	Constantin Nivet - [**CN**]
	Noel Peña - [**NP**]
	François Grzybowski - [**FG**]
Project Name:	mobile tracking car
Document ID:	software-interface-design.pdf

Table 2: Approval

1 Information

This document is based on the requirement-specification.pdf document.

2 Introduction

2.1 Purpose

The purpose of this document is to describe and explain the software implementation of the Multidisciplinary Project that consists in mobile tracking car. This document and the source code will provide information in order to further develop the system or implement the existing system.

2.2 Document Structure and Reading instructions

In this part we will explain every different section of the whole document by using the View Technic, with each view representing the system in some way.

- "The System Overview" here we describe the system.

- "System Interfaces" describes the Interfaces in the system.

- "Architectural View" describes the different levels of architecture and helped by interfaces shows the decisions applied to the overall design.

- "Logical View" here we show every use cases we have needed in some class.

- "Module View" contains a description of some important modules in the project.

- "Process View" shows the classes and threads from the system.

- "Deployment View" description of the different ways of the implementation and the possible different target.

- "Implementation View" shows how to build and run de target.

3 System Overview

The mobile tracking car (MTC) is to be designed to do recon missions in areas where humans can't stay.

3.1 System Context

The context diagram describes how the system interacts with the build in components and also with the actors.

The target of this diagram is to show the different main actors and the components.

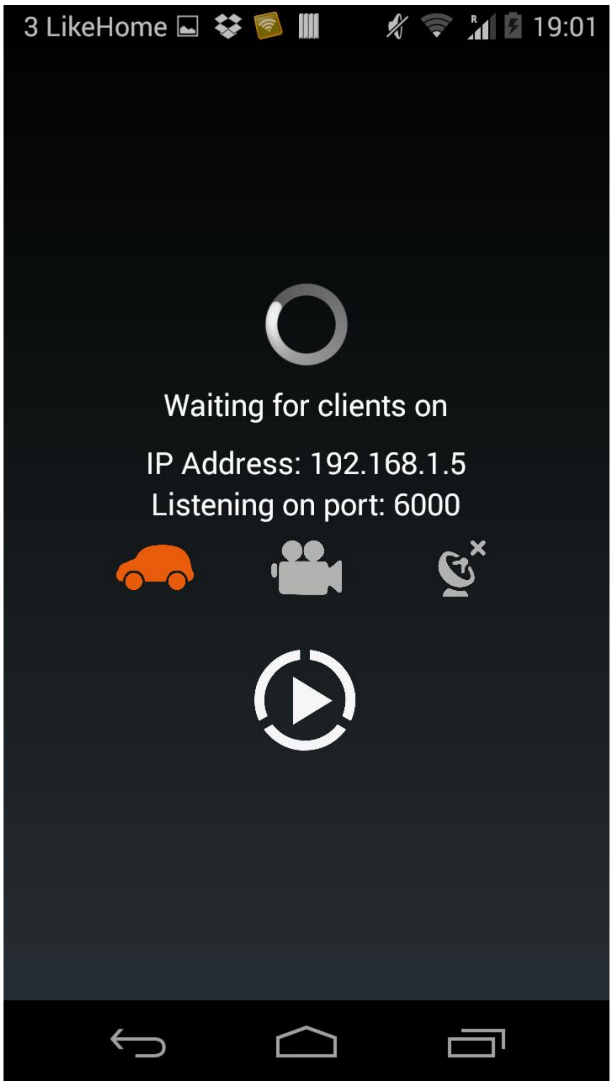
In the Requirement Specification, in the 2.1.2 Context Diagram point , we show this diagram in a picture.

3.2 System Introduction

The car will consist out of a toy car, a microcontroller and a camera. The camera of a smart-phone will be used, because a smart-phone provides a lot of additional features like for example GPS. The car will be controlled by a second smart-phone. On the main smart-phone, which is responsible for the controls, the video should be shown. Therefore a convenient user interface which provides the video on the one hand, and the control buttons on the other hand has to be developed. For the data transfer wireless LAN will be used. All three components, the two smart-phones and the micro-controller with a wireless LAN module, will connect to a router. As an addition the system should provide image recognition to detect barriers in the form of simple shapes.

4 System Interfaces

Our project consists in android application that uses a TCP connection to send data to the microcontroller via WIFI. This application shows a main menu where the user can see if the connection is stablished or not through simple visual emoticons. How we can see in the image, If the light is orange the connection is done.



Waiting for clients on

IP Address: 192.168.1.5
Listening on port: 6000

Figure 1: System Interface

When the connection is done the application throws the main menu screen. Once the user is in the main menu can to use three different buttons to interact with the application. If the user choose the “Button Control” button will have access to steer the car mode through two system control.

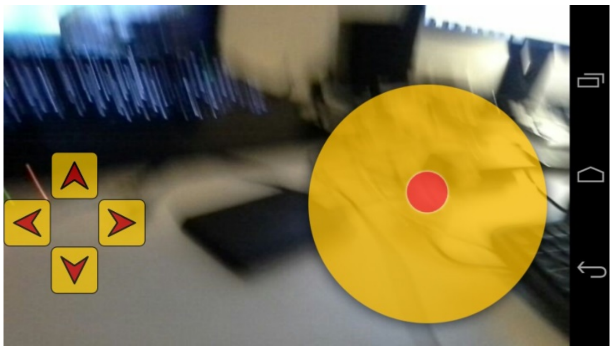


Figure 2: System Interface II

The last picture shows the different modes to steer the car. The user can choose between driving the car with the typical direction that appears in the left part of the screen or choose the joystick option that the user can find in the right part of the screen. In additon, in the steering mode the user can see the images that the car shows in the backgroung of the screen. Another interface option is steer the car through a map with a GPS tracking mode.

5 Architectural View

In this point we will show the primary structure and architecture that the team has used in the project. Each architectural design is detailed below. In the overview the applications that have been defined will be presented. Each architectural decision for our three applications is described under one general package for each application.

5.1 Overview

The target to divide architecturally the project in three applications is to save up time and effort the member of the group because our work is divided in three different teams. The system has been divided into three applications, ”Emulator (Microcontroller)”, “Main-App (Graphical Interface)” and ”Video-stream”.

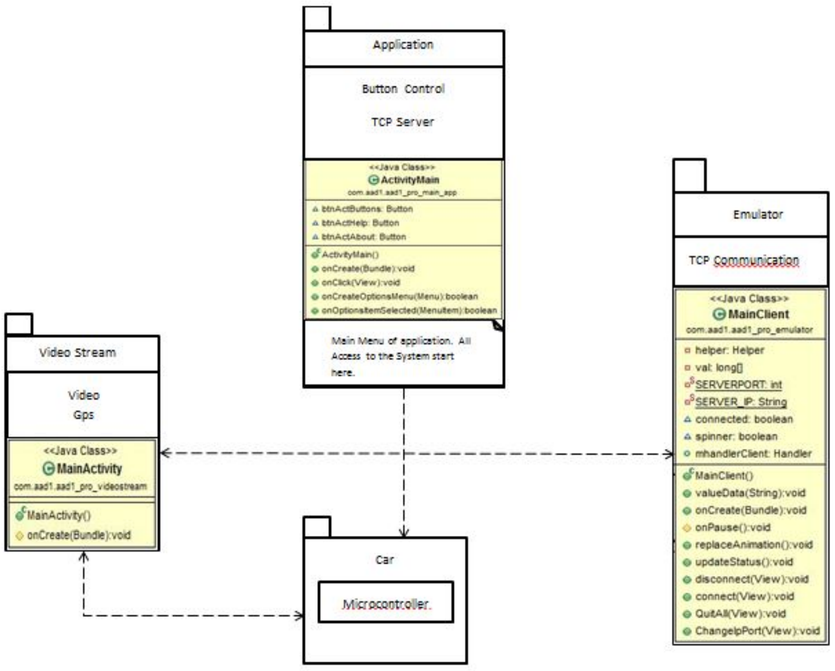


Figure 3: Architectural View

5.2 Architectural significant design packages

5.2.1 UI

The UI is built around the Hardware, the Interface and the Server, which were explained before. The Hardware depends on Car mainly, which is composed of one microcontroller and another device to use the video streaming system. The

Interface is based on two main parts, Control and video. On the one hand the control is responsible of steering the car through its different buttons and on the other hand the Video working shows the images in the background. Finally, the server makes possible all the different communications thanks to TCP connection.

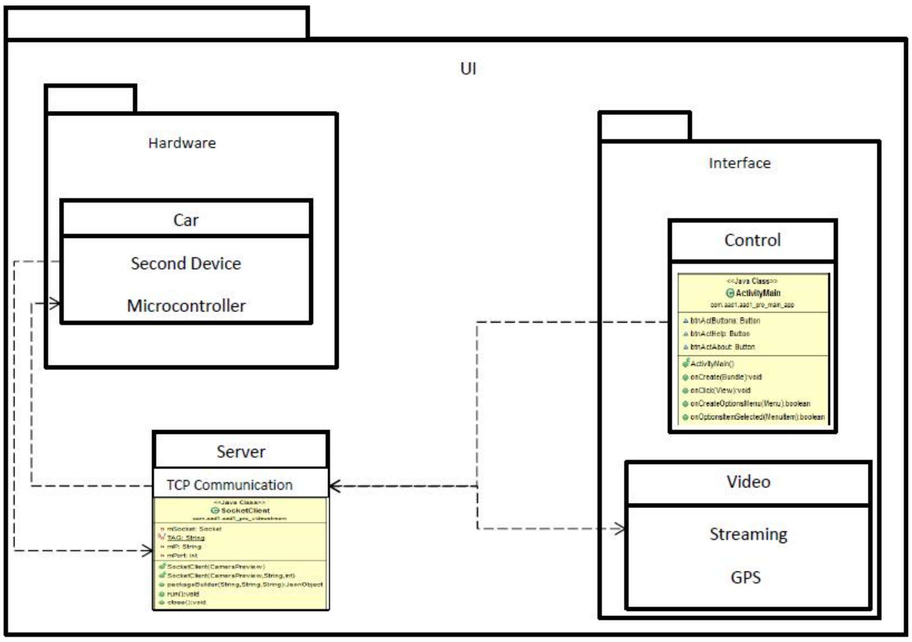


Figure 4: Architectural View Overview

6 Logical View

In this section we will show all the use case we have in order to describe the functionality provided to the user

6.1 Use Cases - Secuence Diagram

Every use case shows the important aspects if each case. Here we explain how our project is going to work. In our project we had 4 use cases, we could reach the first 2 that are steering the car and streaming the images that the second phone capture and send. The other 2 use case couldn't have been finished, these two are the gps tracking and

6.1.1 Use case 1 : "Steer the car"

Description:

This use case describes the main interaction between the user and the car. The user can adjust the speed and control the driving direction of the car. Pre-conditions: The car and the main smart-phone have to be connected to the wireless LAN network. Post conditions: The car is moving and avoiding barriers.

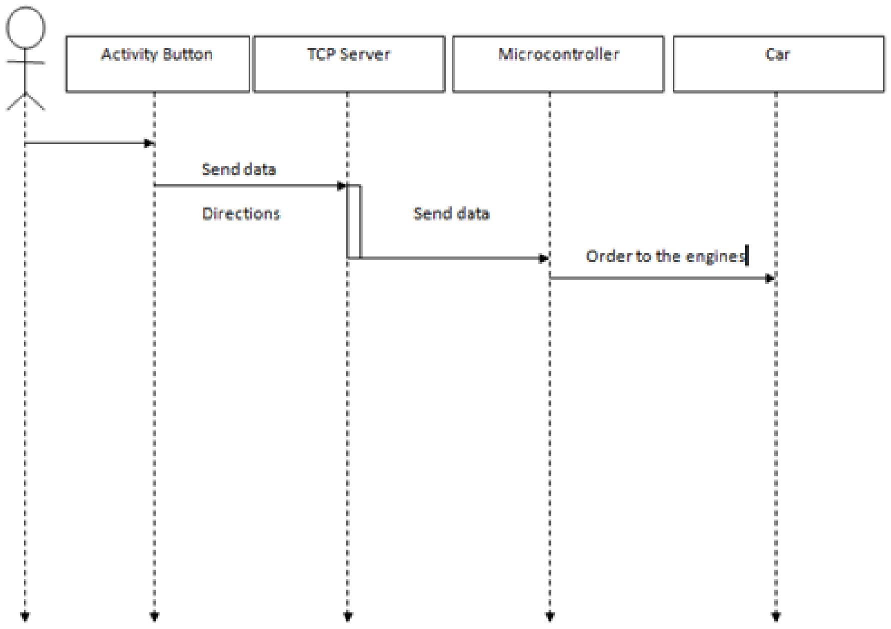


Figure 5: Use Case 1: Steer the car

6.1.2 Use case 2: "Video Streeming"

Description:

This use case describes the video transmission from the embedded smart-phone to the main smart-phone. The camera on the embedded smart-phone captures the the video and sends a stream to the main smart-phone. The video will then be shown on the main smart-phones user interface. Pre conditions: The embedded and the main smart-phone have to be connected to the wireless LAN network. Post conditions: The video is shown on the main smart-phone and provides the user with a live stream of the current environment of the mobile tracking car.

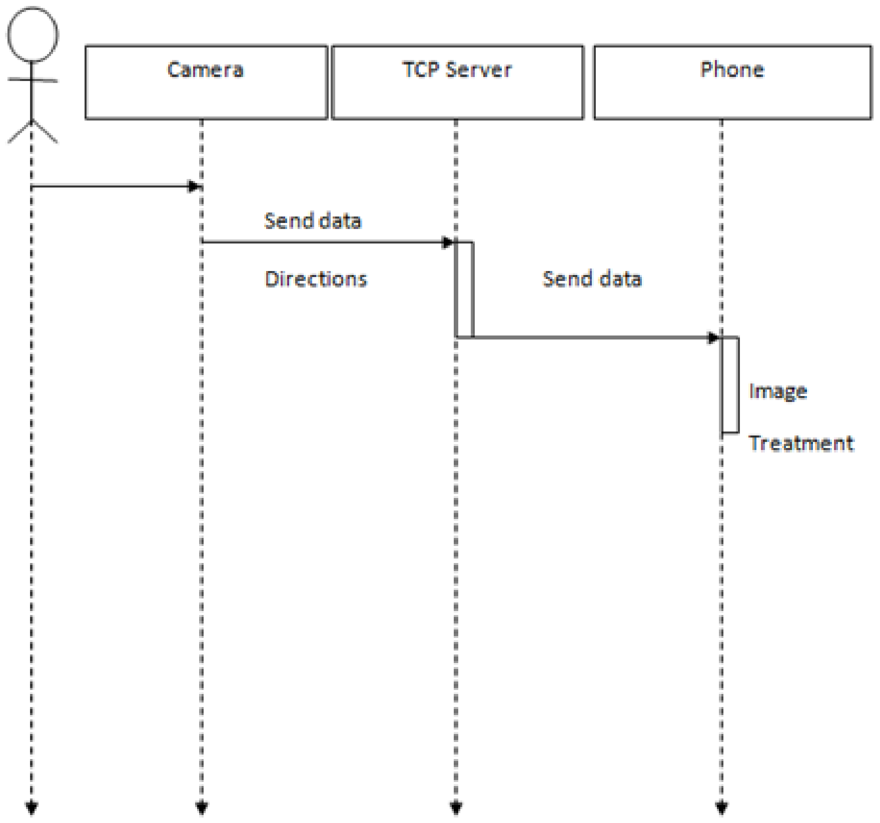


Figure 6: Use Case 1: Video Streaming

6.1.3 Use case 3 : ”GPS Tracking”

Description:

This use case describes interaction with the GPS - satellites. A Google-Maps integration in the main smart-phone will display the user the cars current position.

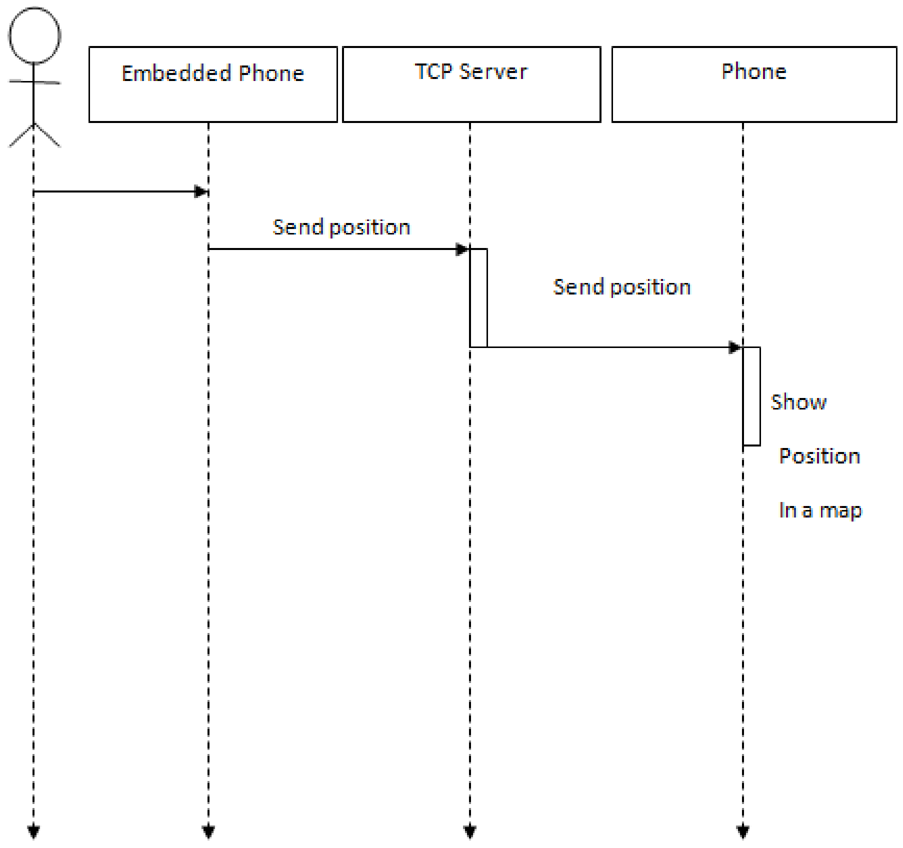


Figure 7: Use case 3 : ”GPS Tracking”

For this case we had two options, the first one consists in the embedded phone has a tracker class in it. It gets the position from the satellite, send this position to the monitoring device, which gets this value and put a dot on the map, and the section option is put the gps on the embedded phone, and send the screenshots (image format) through the socket already existing and in the main phone only images we received from the server like the video stream.

AARHUS UNIVERSITY

6.1.4 Use case 4 : ”Image recognition”

Description:

The embedded smart-phone is recording his environment. An application running on this phone processes the current video and reacts on them. If there are shapes which are predefined in the program, an alert should triggered to the user.

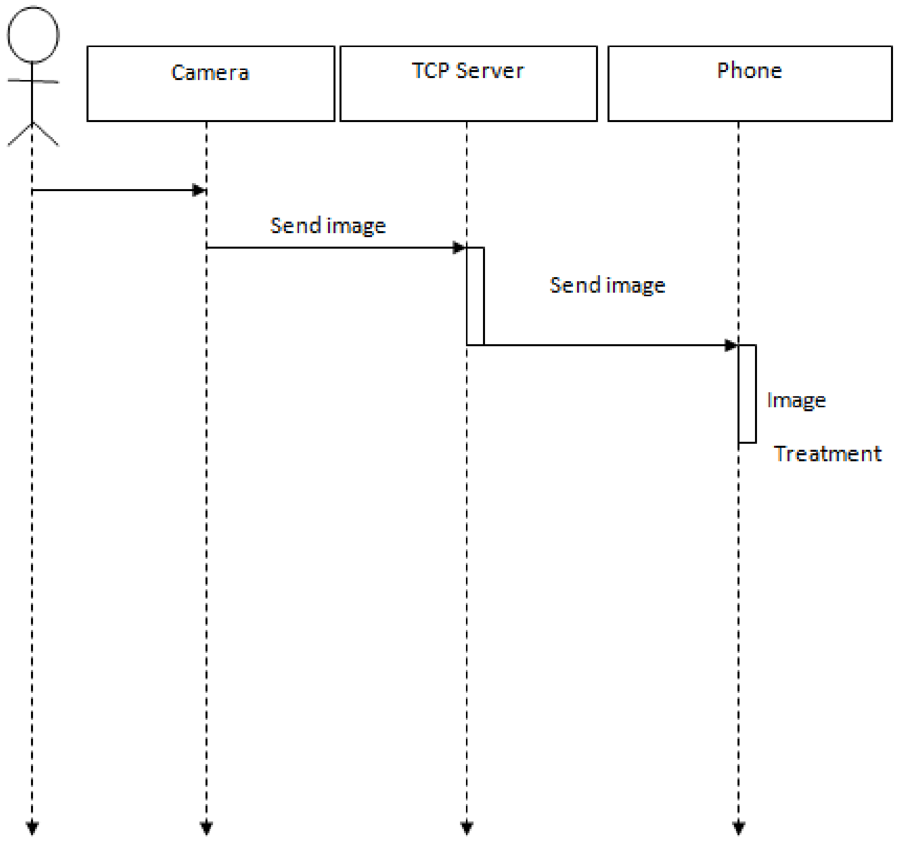


Figure 8: Use case 4 : ”Image recognition”

6.2 Static Class Diagram

In our project we have 4 different class diagrams, one for each of the applications we have done and an other one for the server. The steering app, the video streaming and the emulator we design before using the microcontroller.

6.2.1 Graphical interface

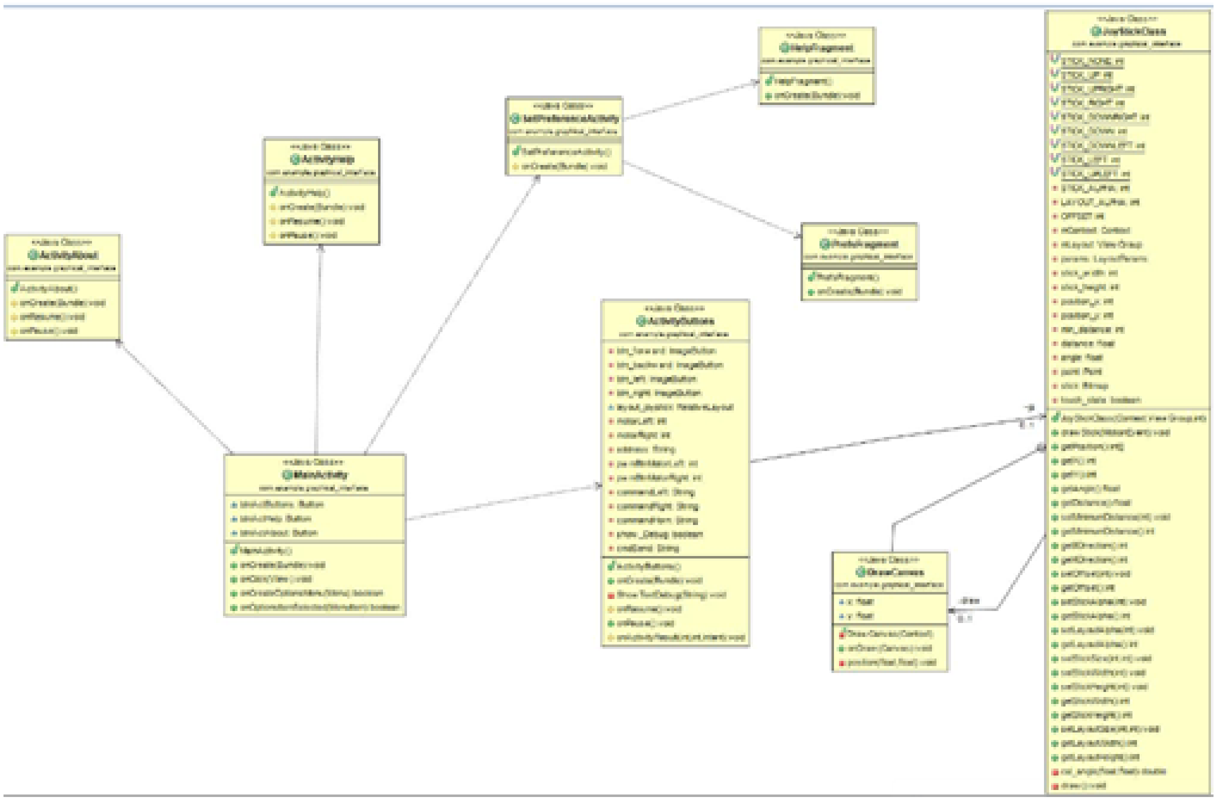


Figure 9: GUI Static Class Diagram

The Main Menu consists in three buttons which throw new secondary activities as well as different fragments of the application. Moreover, our application counts on an action overflow where we can access to the settings page. In this page, we can find all the information about the camera working, details of the buttons and also driving modes.

6.2.2 Video Streaming

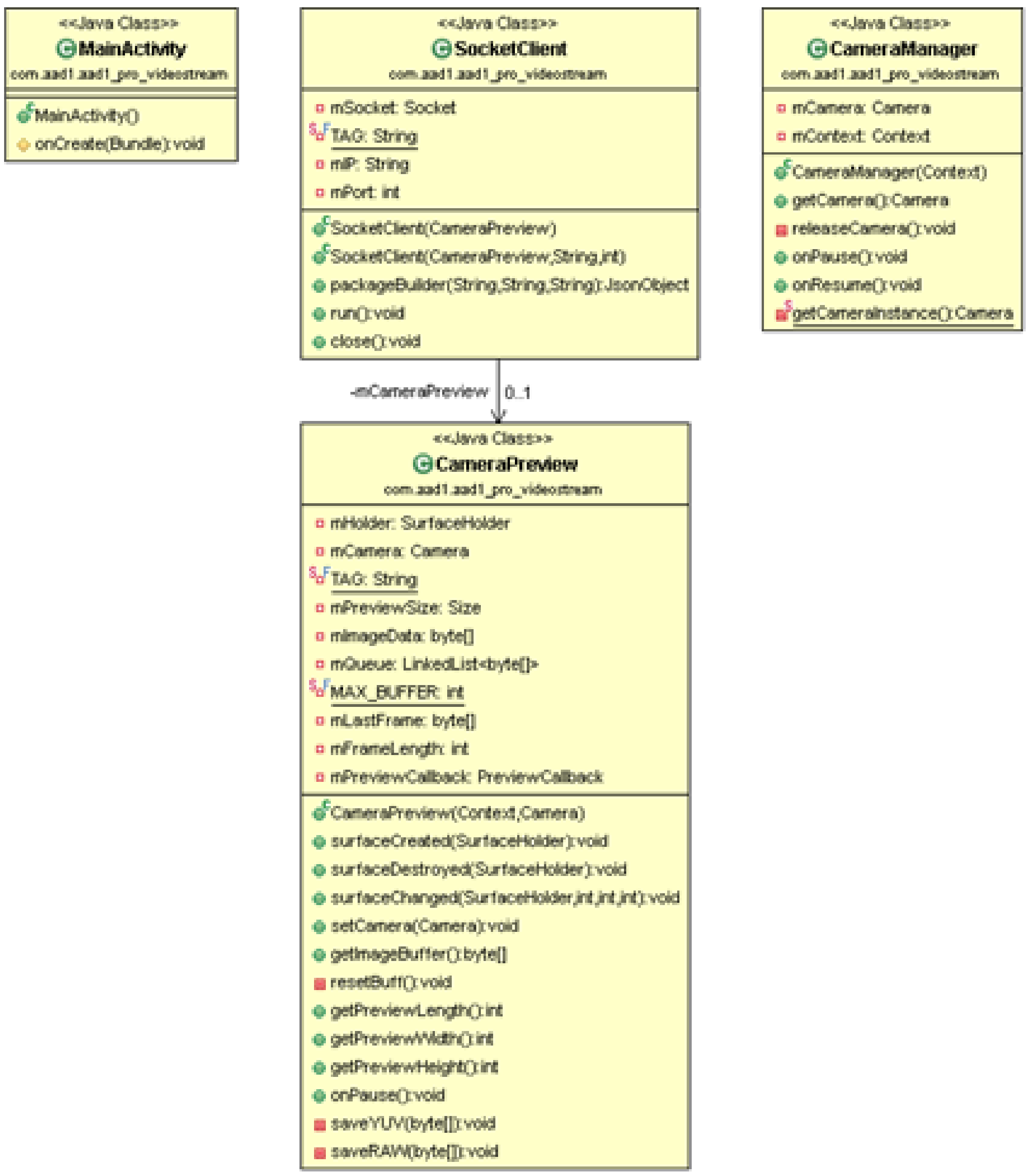


Figure 10: Video Streaming Static Class Diagram

We did the CameraPreview class. This class must extends a SurfaceView and implement SurfaceHolder.Callback. First we called a Camera and a Holder in this class and we settled up the holder so it can holds the surface view. This is the class that initializes the Camera and helps to display the new frame that the on-the-car phone sees.

6.2.3 Emulator

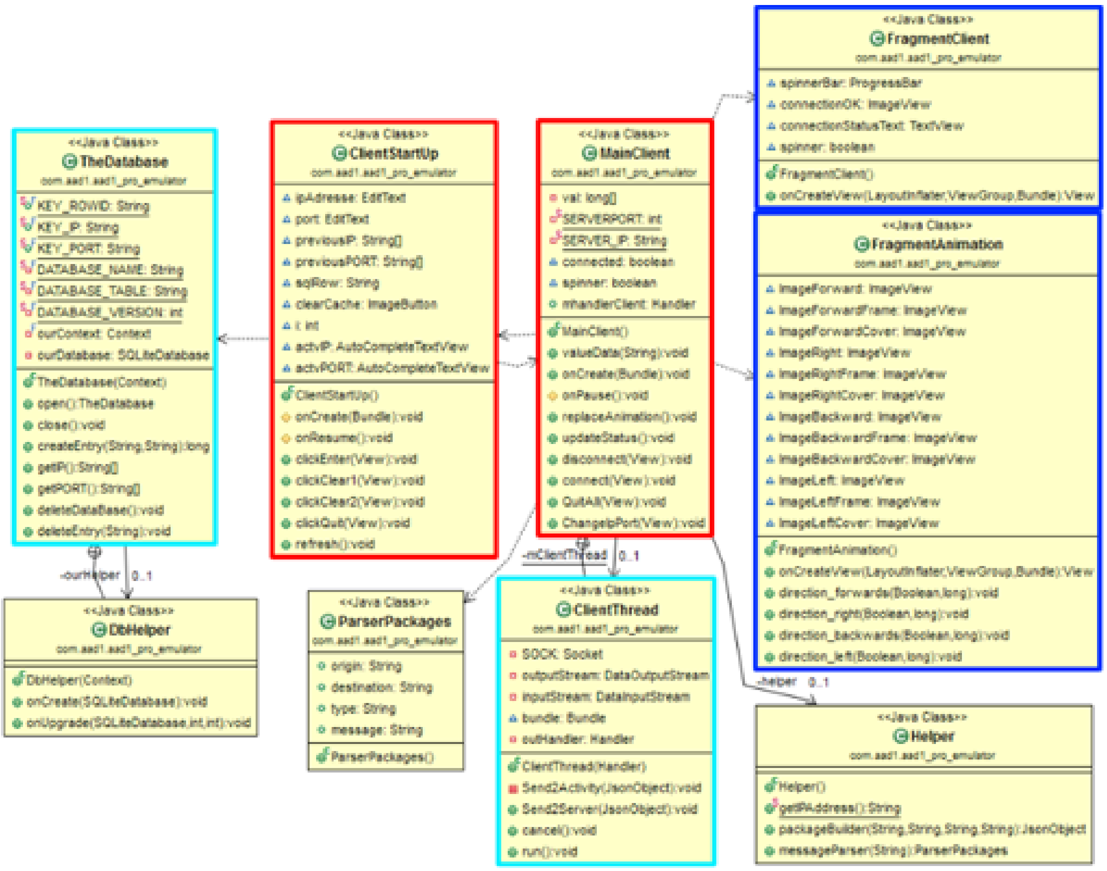


Figure 11: Emulator Static Class Diagram

The communication between the emulator and the car will be done by wireless LAN.

First of all we check if there is an entry in the list with the same IP . If the IP exists, the "deleteEntry" method is called. This method deletes the old Entry and adds a new Entry with the IP address and the new Port. If the IP address does not exist the "createEntry" method is called, which adds a new Entry to the Database. The "deleteDataBase" method, deletes the whole database with all the entries. The "refresh" method is responsible of updating the list. It calls the "TheDatabase" class and asks for the IP address and Port String Array. The two String Arrays are created in the "getIP" and "getPORT" methods. When the IP address has been inserted, it is saved in the Database with the port. The Animation has been implemented with the "setTranslation" Method. It moves the bar depending on the incoming intensity of the according direction.

The Client gets the IP and the Port via an Intent from the previous activity. Then a Thread is created with an added Message Handler to communicate with the Activity. Then a Socket is generated. The two objects "outputStream" and "inputStream" are responsible for the data transfer between the server and the client. While there is a Input-stream the buffer reads the input. A ”JSON-Object” is created, which translates the Stream into the JSON format.

The ”Send2Activity” Method is used to send a Message to the Activity. The ”Send2Server” Method sends a Message to the Server. More precisely the client sends the Server the online/offline state. To get the movement-data into the Animation-Fragment the message handler calls the ”valueData” object. There the input data is converted and stored in a string array. The ”replaceAnimation” Object finally bundles the string array and gives it to the animation fragment.

6.2.4 TCP Server

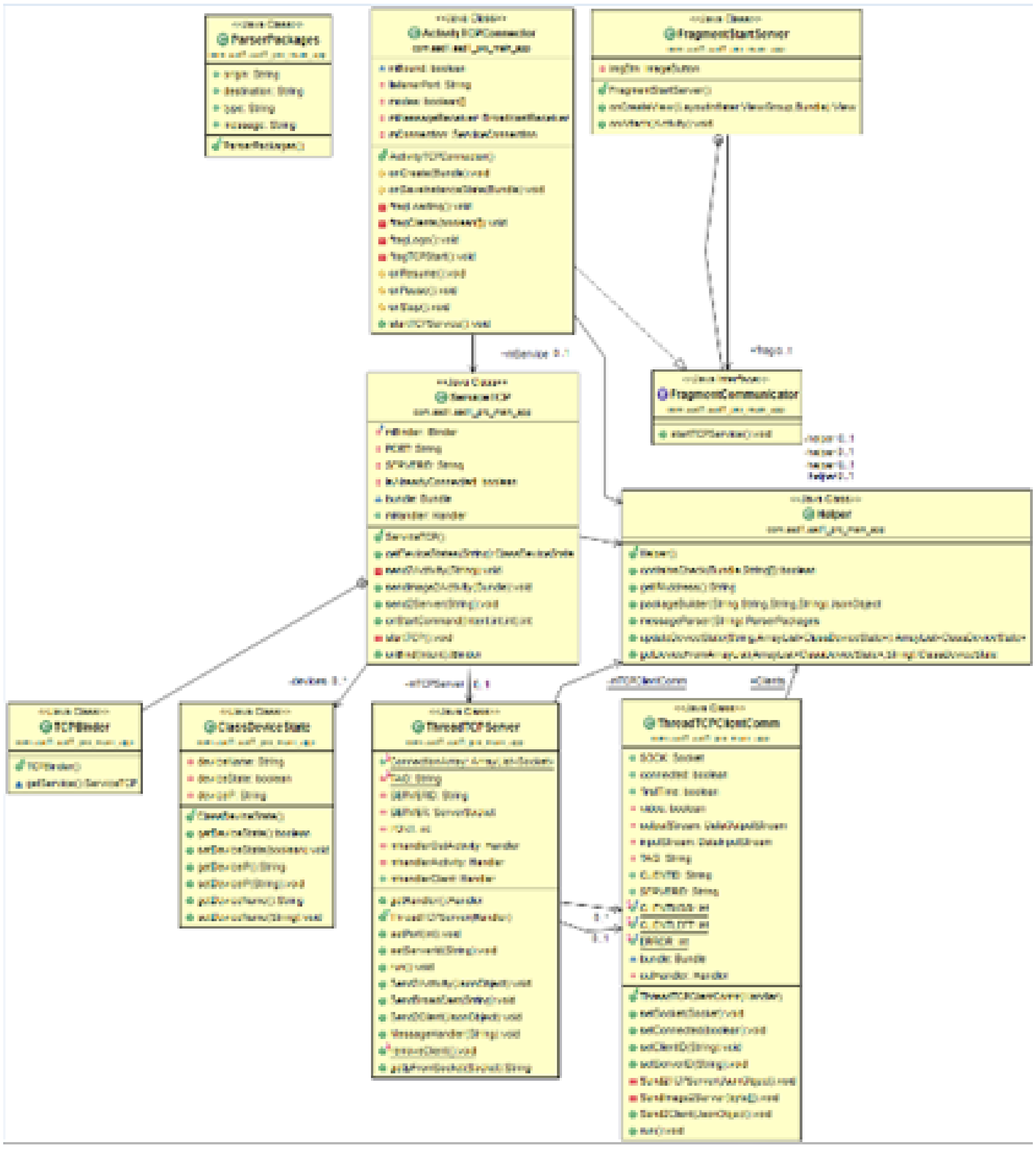


Figure 12: TCP Static Class Diagram

We start the TCP Server in an own Service. The Service get started at the beginning. When the Service is started once, it is possible to bind and unbind to the Service. The Service provides also the current state of the devices. Hence it is possible to gather anywhere in the application the current connection states. When TCP Service is started by the Activity, a new TCP Main Thread get invoked with a Message Handler with is located in the Service to communicate. The TCP Main Thread creates a Server Socket with a specific port. After a successful creation the Server is waiting for clients. If a client attempt connect to the Server a new Socket is created by the Server. Now it is possible to create a new Thread for this client. Every Socket and corresponding Thread is stored in arrays. This is necessary to send single messages or broadcasts to the clients. The communication of every Client Thread to the TCP Server Thread is also implemented with a Message Handler.

When the client sends a message to the Server, the TCP Client Thread send this message to the TCP Main Thread. The Server decides according to the destination attribute either route the message to the Server Activity or to another client. For the video stream it is necessary to handle the incoming stream different to the normal message traffic with JSON. The transmission of a compressed bitmap is based on byte arrays. The frames are send one by one from the embedded-camera application to the main application

The communication between the Service, TCP Server Thread and the TCP Client Threads is based on Message Handlers. A Local Broadcast Manager is used to send intents between the Activity and the Service.

7 Module View

This section describes some of the most important classes of our project. Under each module we show a short description about them.

7.1 ActivityMain - ActivityButtons

This part contains the access menu of the application. The class called "ActivityMain" is the responsible of create the activities and give value to the buttons. From here can link every activity with the next classes and fragments. Another view to look at is the class called "ActivityButtons". With this class the user can send move orders that after will be interpreted for the microcontroller and be used for move the car.

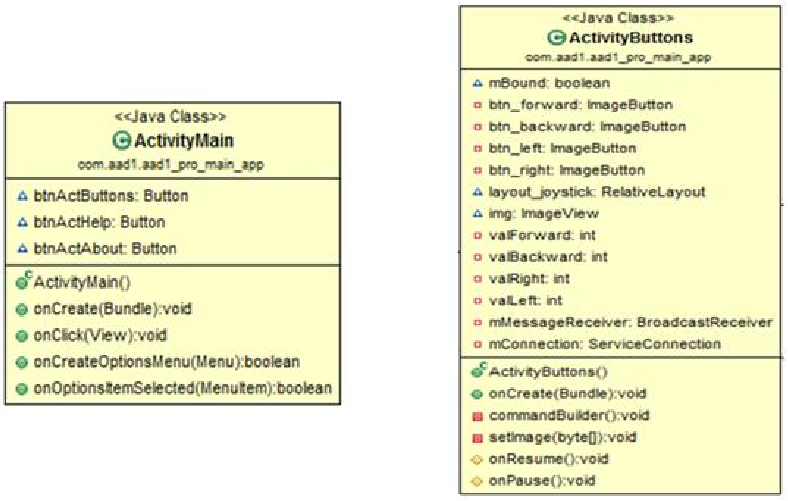
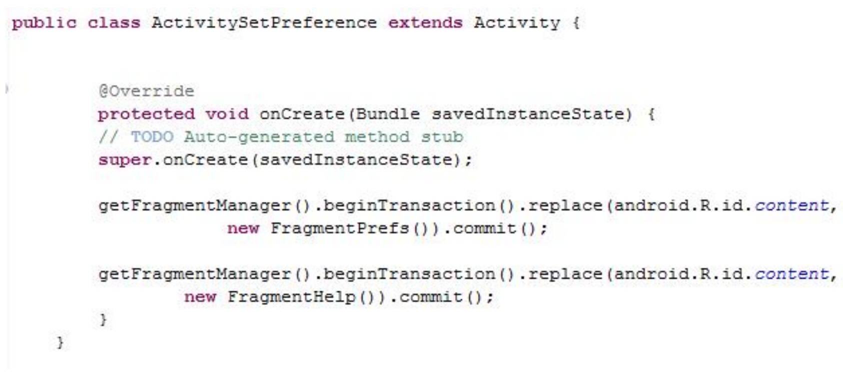


Figure 13: ActivityMain - ActivityButtons



```
public class ActivitySetPreference extends Activity {


        @Override
        protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);

        getFragmentManager().beginTransaction().replace(android.R.id.content,
                    new FragmentPrefs()).commit();

        getFragmentManager().beginTransaction().replace(android.R.id.content,
                new FragmentHelp()).commit();
        }
    }
```

Figure 14: ActivitySetPreference

The last class called "ActivitySetPreference" is essential to communicate the "ActivityMain" class with the "FragmentPref" class responsible of show the preferences of the application.

7.2 ClassJoystick

This class is fundamental for the user to interact with the move system. "ClassJoyStick" contents the sentence that allows send potency and addressing to the car.

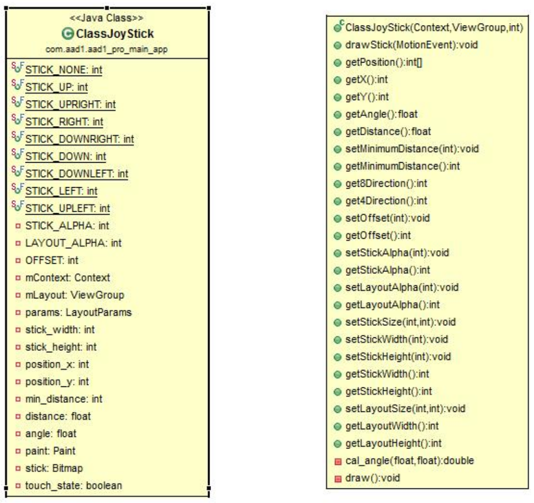
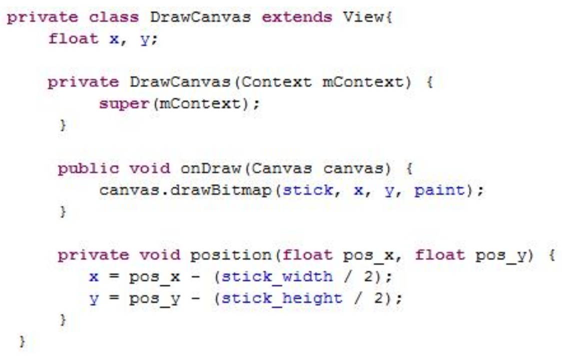


Figure 15: ClassJoystick

Finally, the class called "DrawCanvas" works making possible the limit in the Joystick.



```
private class DrawCanvas extends View{
    float x, y;

    private DrawCanvas(Context mContext) {
         super(mContext);
     }

     public void onDraw(Canvas canvas) {
         canvas.drawBitmap(stick, x, y, paint);
     }

     private void position(float pos_x, float pos_y) {
        x = pos_x - (stick_width / 2);
        y = pos_y - (stick_height / 2);
     }
 }
```

Figure 16: Private class DrawCanvas

8 Process View

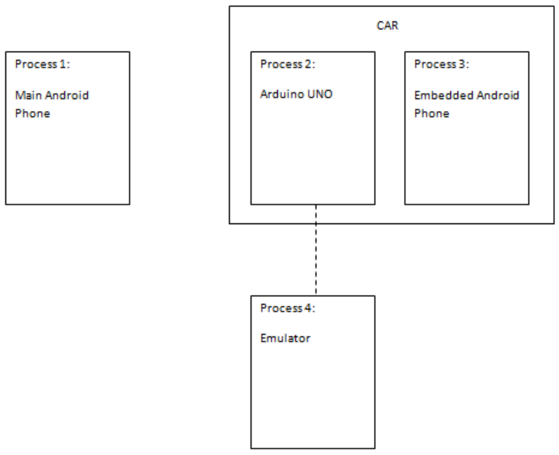


Figure 17: Process View

As you can see in the image, in our project we have 3 main processes, and then an optional 4th that is the emulator that can be replaced by the Arduino. The first one is the main android phone; with it we are going to be able to drive the car as well as watching what the camera in the car shows and establish a TCP connection as a server. The second one is the Arduino Uno microcontroller, which is in charge of steering the car with the signals it receives from the main app via wifi, this process can be replaced by the process 4 which is the emulator, in which instead of having a car and a microcontroller we have another android phone that simulates the car and the speed. And finally, the 3rd process is the secondary android phone, which is the responsible of taking pictures and sending them to the server. Once we have seen all the process we are going to see how they interact with each other, in this case the 4th project is not going to be explained due to we don't use it in our final prototype, it's just a process we created when we didn't have the Arduino program. Therefore, the first process we need is the main android phone because it sets the TCP connection, after having that, this process wait to the client, here is where we have to activate the microcontroller and the video streaming, once the connection is done we use the main app to steer the car, so then the server sends data to the microcontroller in order to steer the car and the only thing that the second android phone does is sending images from the camera every 40 ms or the time you wish.

9 Deployment View

9.1 Deployment Description

In this part we show the final deployment view about our project with the different connection networks.

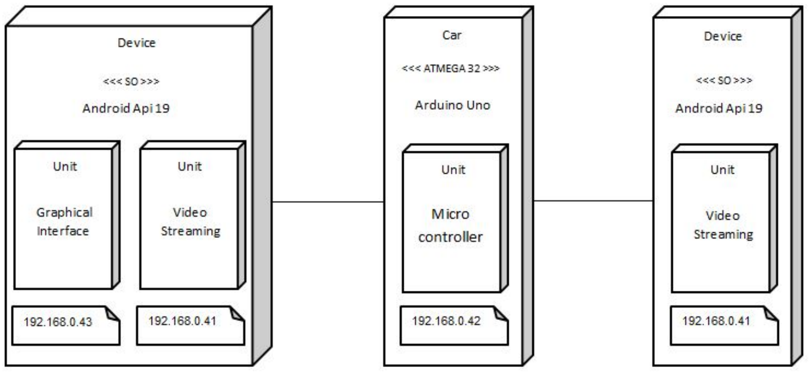


Figure 18: Deployment View

Our Deployment Descriptor is built with one Device connected with WiFi connection to a car, in turn which is connected to a second device responsible of send the video streaming starting from WiFi too.

9.1.1 Device

This part is composed of two units called ”Graphical Interface” and ”Video-Streaming” which used two different connection directions to communicate with the second part of this project. The car.

9.1.2 Car

The main part of the car is the microcontroller unit related with the second device. We use ”Arduino Uno” with a wireless LAN connection to connect both.

9.1.3 Second Device

Its unit is related with the video streaming. It’s connected with LAN protocol to the Car.

List of Figures

List of Tables