Wi-Fi Controlled Car

|  |  |
| --- | --- |
| Document title | Design |
| Document ID | WCC-Design |
| Authors | Ambroise Dhenain  Szymon Klepacz  Anatolii Shakhov  Alvaro Garcia  Pierre Le Texier |
| Supervisor | Torben Gregersen |
| No. pages | 31 |
| Date | 02-12-2013 |

Design

This document contains the design of the application Wi-Fi Controlled Car.

Document history

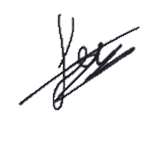
|  |  |  |  |
| --- | --- | --- | --- |
| Date | Audit | Author | Description |
| 02-12-2013 | 1.0 | AD | Moved design from project report to Design document. |
| 06-12-13 | 1.1 | AD | Corrections after Torben’s metting.  Diagrams updated in physical and deployment views.  Class diagrams updated. |
| 08-12-13 | 1.2 | AS | Added battery specs |
| 08-12-13 | 1.3 | SzK | Changed motor shield schematic and description. Added YUN schematic and described communication. |
| 08-12-13 | 1.4 | AS | Minor Spelling check |
| 09-12-13 | 1.5 | AD | Merge changes. Add diagrams. |
| 09-12-13 | 1.6 | AD | Correction after meeting with Torben. |
| 09-12-13 | 1.7 | PLT | Add camera documentation. |
| 11-12-13 | 1.8 | SzK | Added voltage regulator and schematic. |
| 11-12-13 | 2.0 | AD | Final version. Correction after meeting with Torben. Change all class diagrams, check figure titles, information about camera, use case 2.1. |

Approval

|  |  |
| --- | --- |
| Author(s) | Ambroise Dhenain – 201301255 [AD]  Szymon Klepacz – 201301204 [SzK]  Anatolii Shakhov – 201301534 [AS]  Alvaro Garcia – 201301220 [AG]  Pierre Le Texier – 201301944 [PLT] |
| Project name | Wi-Fi controlled car |
| Document ID | WCC-Design |
| No. pages | 31 |

**By signing this document both parties accept, that this is the requirements for the development of the desired system.**

**Place and date: 13-12-2013**

Authors Supervisor

Contents

[1. Opening 4](#_Toc374701344)

[1.1. Purpose 4](#_Toc374701345)

[1.2. Reading instruction 4](#_Toc374701346)

[2. 4+1 view model 5](#_Toc374701347)

[2.1. What is 4+1? 5](#_Toc374701348)

[3. Android 6](#_Toc374701349)

[3.1. Scenarios 6](#_Toc374701350)

[3.2. Process view 8](#_Toc374701351)

[3.3. Physical/Deployment view: 9](#_Toc374701352)

[3.3.1. Deployment diagram 9](#_Toc374701353)

[3.4. Development/Implementation view: 9](#_Toc374701354)

[3.4.1. Component diagram 9](#_Toc374701355)

[3.4.2. Package diagram 10](#_Toc374701356)

[*3.5.* Logical view 11](#_Toc374701357)

[3.5.1. Class diagrams 11](#_Toc374701358)

[3.5.1.a. Package com.iha.wcc 12](#_Toc374701359)

[3.5.1.b. Package com.iha.wcc.job.car 13](#_Toc374701360)

[3.5.1.c. Package com.iha.wcc.job.camera 14](#_Toc374701361)

[3.5.1.a. Package com.iha.wcc.job.ssh 15](#_Toc374701362)

[3.5.1.b. Detailed explanations about constants, methods and variables 16](#_Toc374701363)

[3.5.2. Sequence diagrams 18](#_Toc374701364)

[3.5.2.a. Move 18](#_Toc374701365)

[3.5.2.b. Initialize car settings 19](#_Toc374701366)

[4. Arduino 22](#_Toc374701367)

[4.1. Class diagram 22](#_Toc374701368)

[4.2. Schematics (Electronic) 23](#_Toc374701369)

[4.2.1. Arduino Yun 23](#_Toc374701370)

[4.2.2. Arduino YUN – communication between ATmega32u4 and AR 9331 24](#_Toc374701371)

[4.2.3. Motor shield 25](#_Toc374701372)

[4.2.4. Battery 26](#_Toc374701373)

[4.2.5. Voltage regulator 26](#_Toc374701374)

[4.2.6. Camera 27](#_Toc374701375)

[4.2.6.a. Connection to Arduino 27](#_Toc374701376)

[4.2.6.b. Installing drivers 28](#_Toc374701377)

[4.2.6.c. SD card configuration 28](#_Toc374701378)

[4.2.6.d. Streaming tool: MJPG-streamer 28](#_Toc374701379)

[4.2.6.e. Getting the Stream/Snapshot 29](#_Toc374701380)

[4.2.6.f. Camera automatically started from Android application 29](#_Toc374701381)

[5. Glossary 30](#_Toc374701382)

[6. References 30](#_Toc374701383)

Tables of figures

[Figure 1 - 4+1 view model diagram 5](#_Toc374701317)

[Figure 2 - Use case - System version 1.0 6](#_Toc374701318)

[Figure 3 - Use case - System version 2.1 7](#_Toc374701319)

[Figure 4 - Communication between Android and Arduino 8](#_Toc374701320)

[Figure 5 - Deployment diagram 9](#_Toc374701321)

[Figure 6 - Component diagram 9](#_Toc374701322)

[Figure 7 - Packages diagram 10](#_Toc374701323)

[Figure 8 - Class diagram – Overview 11](#_Toc374701324)

[Figure 9 - Class diagram - Package com.iha.wcc 12](#_Toc374701325)

[Figure 10 - Class diagram - Package com.iha.wcc.job.car 13](#_Toc374701326)

[Figure 11 – Class diagram - Package com.iha.wcc.job.camera 14](#_Toc374701327)

[Figure 12 – Class diagram - Package com.iha.wcc.job.ssh 15](#_Toc374701328)

[Figure 13 - Sequence – Move (1.0) 18](#_Toc374701329)

[Figure 14 - Sequence - Initialize car settings (1.1) 19](#_Toc374701330)

[Figure 15 - Sequence - Take picture 1.2 19](#_Toc374701331)

[Figure 16 - Sequence - Take picture 2.0 20](#_Toc374701332)

[Figure 17 - Sequence - Video stream 21](#_Toc374701333)

[Figure 18 – Arduino class diagram 22](#_Toc374701334)

[Figure 19 - Arduino YUN 23](file:///G:\CESI\PEE\Aarhus\Cours\ETCCCP\02%20-%20Project%20Working%20part%20of%20the%20ETCCCP%20course\Project\Project%20management\3%20-%20WCC%20-%20Design.docx#_Toc374701335)

[Figure 20 - Communication between ATmega32u4 and AR 9331 24](file:///G:\CESI\PEE\Aarhus\Cours\ETCCCP\02%20-%20Project%20Working%20part%20of%20the%20ETCCCP%20course\Project\Project%20management\3%20-%20WCC%20-%20Design.docx#_Toc374701336)

[Figure 21 - TB6612FNG 25](file:///G:\CESI\PEE\Aarhus\Cours\ETCCCP\02%20-%20Project%20Working%20part%20of%20the%20ETCCCP%20course\Project\Project%20management\3%20-%20WCC%20-%20Design.docx#_Toc374701337)

[Figure 22 - PCA 9685 25](file:///G:\CESI\PEE\Aarhus\Cours\ETCCCP\02%20-%20Project%20Working%20part%20of%20the%20ETCCCP%20course\Project\Project%20management\3%20-%20WCC%20-%20Design.docx#_Toc374701338)

[Figure 23 - Arduino YUN 25](#_Toc374701339)

[Figure 24 - Voltage regulator 26](#_Toc374701340)

[Figure 25 - Voltage regulator schematic 26](file:///G:\CESI\PEE\Aarhus\Cours\ETCCCP\02%20-%20Project%20Working%20part%20of%20the%20ETCCCP%20course\Project\Project%20management\3%20-%20WCC%20-%20Design.docx#_Toc374701341)

[Figure 26 - Communication between the camera, Arduino YUN and the Android application 27](#_Toc374701342)

[Figure 27 - Use PuTTY to access to the embedded Linino 27](#_Toc374701343)

1. Opening

# Purpose

This document is about the design of the application “Wi-Fi controlled car”.

The design is based on the “**Requirement specification**” document, to understand better the way we designed it, please consult the document before read the “Design” document.

We will explain everything about how we designed the Android application, the Car application, the communication between these applications and the Car design (hardware).

We will use the 4+1 view model to show and explain our design.

# Reading instruction

Chapter 2: Explains the **4+1** view model architecture.

Chapter 3: Describes the design of the Android application.

Chapter 4: Describe the design of the Arduino application.

Chapter 5: Glossary which contains explanations about some words and abbreviations used in this document.

Chapter 6: Contains the references we used in this document.

1. 4+1 view model

# What is 4+1?

4+1 is a view model designed by Philippe Kruchten to “describe the architecture of software-intensive systems, based on the use of multiple, concurrent views”. These views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers. [[1]](#footnote-1)

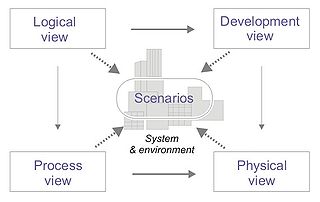


Figure 1 - 4+1 view model diagram

* ***Logical view***: The logical view is concerned with the functionality that the system provides to end-users. UML diagrams are used to represent the logical view that includes [*Class* diagram](https://en.wikipedia.org/wiki/Class_diagram), [*Communication* diagram](https://en.wikipedia.org/wiki/Communication_diagram)s and [*Sequence* diagram](https://en.wikipedia.org/wiki/Sequence_diagram)s.
* ***Development view***: The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is **also known as the implementation view**. It uses the UML [*Component* diagram](https://en.wikipedia.org/wiki/Component_diagram) to describe system components. UML diagrams used to represent the development view include the [*Package* diagram](https://en.wikipedia.org/wiki/Package_diagram).
* ***Process view***: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behaviour of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability. UML diagrams to represent process view include the *Activity* diagram.
* ***Physical view***: The physical view depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is **also known as the deployment view**. UML diagrams are used to represent physical view include the [*Deployment* diagram](https://en.wikipedia.org/wiki/Deployment_diagram).
* ***Scenarios***: The description of the architecture is illustrated using a small set of [*Use cases*](https://en.wikipedia.org/wiki/Use_case), or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is **also known as use case view**.

1. Android

We will present our design using a top-down approach, starting by the overviews of the system and explaining details only at the end, using detailed sequence diagrams.

# Scenarios

To describe the system functionalities, we use the UML Use case diagrams. There is one diagram per version. The **Use cases** diagrams show the actors that can interact with the system and show also which kind of action they can use/do.

We will only explain the main versions of the scenarios. [[2]](#footnote-2)



Figure 2 - Use case - System version 1.0

This is the version 1.0 of the system, which can basically only move, the details about all the possible movements, it’s the minimal version we wanted. This version was reached the 28th November.



Figure 3 - Use case - System version 2.1

We didn’t reach completely this version, we didn’t use a proximity sensor, so the use cases “Detect obstacle” and “Alarm obstacle” are not done. This diagram is what we planned to do. Other use cases are done and work properly.

We didn’t reach the 1.2 version, we reached the 2.0 directly instead, we preferred make it works with a real video stream than use a fake video using refreshed photos.

# Process view

We decided to use a Sequence diagram to explain as exactly as possible how the communication works. With this sequence diagram and the **Deployment** and **Component** diagrams in the *next* page, you should understand really precisely how the communication works between Android and Arduino.



Figure 4 - Communication between Android and Arduino

This diagram shows how the communication works between the Android application and the Arduino program.  
When the Arduino is powered, it enters in an infinite loop, waiting for an available client. Once this client is connected then it enters in another infinite loop, waiting for commands from the client.  
Once a command is received, it processes it.

We use the TCP communication protocol but we don’t need a response from the server, so the server doesn’t send any response to the client. However, it’s possible to do it if we need it in the future.

This part should also display the Activity diagrams we have done, but they all are in the “*2 - WCC - Requirements specification*” document too. Please consult this document to see them.

# Physical/Deployment view:

## Deployment diagram



Figure 5 - Deployment diagram

This diagram shows the different components we use, which classes or libraries depending on the hardware. It’s a high abstraction level diagram.

# Development/Implementation view:

## Component diagram



Figure 6 - Component diagram

This diagram shows how the components interact with each other. We have only two different components, the **Android Phone** and the **Car**. They communicate by Wi-Fi using a predefined port on the Car. The car contains sub-components such as a motor shield, a camera and so on.

## Package diagram



Figure 7 - Packages diagram

This diagram represents every package in the application. There are only few packages because the application doesn’t use a lot of classes. We choose to separate the activities and the jobs. A job is a class which has basically a “job” and must fill this job and only this one.

The class diagram in the next page will represent every class, grouped by package.

# Logical view

## Class diagrams



**com.iha.wcc.job.ssh**

**com.iha.wcc.job.camera**

**com.iha.wcc**

**com.iha.wcc.job.car**

Figure 8 - Class diagram – Overview

The details about this diagram will be explained in the following pages grouped per package.

### Package com.iha.wcc



Figure 9 - Class diagram - Package com.iha.wcc

This package contains all the Activities (or children) of the entire application. Each Activity is linked to a XML file, the XML view.

### Package com.iha.wcc.job.car



Figure 10 - Class diagram - Package com.iha.wcc.job.car

This package contains all classes in relationship with the car, such as the embedded camera and the Linino embedded Linux OS. The Car class is basically a controller for the car, it’s this class that will control the speed of the car, the direction and so on. Everything is controlled from the application in order to limit the size of the Arduino application and control everything from a high level abstraction and POO language such as Java instead of Arduino language.

### Package com.iha.wcc.job.camera



Figure 11 – Class diagram - Package com.iha.wcc.job.camera

This package contains classes that control the camera, including photo capture and video stream. Basically, everything is managed from the **MjpegView**, it’s a view specialized to deal with **Mjpeg** video stream format, because Android doesn’t support *natively* this format. The use pattern is the same, there is always a \*Task class to call to start the action such as **TakePictureTask** and **MjpegVideoStreamTask**. We won’t use other classes directly, we’ll use only the task to execute an action.

### Package com.iha.wcc.job.ssh



Figure 12 – Class diagram - Package com.iha.wcc.job.ssh

Because we need to execute a SSH command to start the streaming from the camera (actually, once the camera is powered by the YUN via the USB wire, it starts. But the camera doesn’t start the streaming automatically). We decided to do it from android because we found nothing to do it from Arduino (could be better, faster and automatically executed independently on the phone application) and because we can use the settings to configure the options of the streaming such as the FPS, the width and height of the video. But we could also do it from Arduino.

The CarActivity will call the SshTask on start to automatically start the streaming on the camera. Once again, the Activity deals only with \*Task classes.

We use an external library: **Jsch**, to use SSH from Android, *version 0.1.50*.

### Detailed explanations about constants, methods and variables

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Package | Class | Type | Name | Details |
| com.iha.wcc | CarActivity | Variable | queriesQueueSocket | Array of strings that contains all messages to send to the server using sockets. |
| com.iha.wcc | CarActivity | Variable | stopProcessingSocket | Atomic boolean shared and accessible between different threads, used to be sure the connection is available. |
| com.iha.wcc | CarActivity | Variable | socket | Socket connected to the Arduino. |
| com.iha.wcc | CarActivity | Variable | socketThread | Thread which manage socket stream. |
| com.iha.wcc | CarActivity | Variable | networkRunnable | Runnable running in another thread, responsible to the communication with the car. |
| com.iha.wcc | CarActivity | Variable | cameraContent | Contains the Mjpeg view which contains all other components and the video stream. |
| com.iha.wcc | CarActivity | Method | initializeCarSettings | Get the car settings from the local phone settings and send them to the car. |
| com.iha.wcc | CarActivity | Method | startStreaming | Start the video streaming. |
| com.iha.wcc | CarActivity | Method | refreshStreamingView | Refresh the entire view with the special Mjpeg view to stream the video. |
| com.iha.wcc | CarActivity | Method | goForward | Send a request to the car to go forward. |
| com.iha.wcc | CarActivity | Method | goBackward | Send a request to the car to go backward. |
| com.iha.wcc | CarActivity | Method | goLeft | Send a request to the car to go to the left. |
| com.iha.wcc | CarActivity | Method | goRight | Send a request to the car to go to the right. |
| com.iha.wcc | CarActivity | Method | stopTurn | Send a request to the car to stop turn. (Front engine) |
| com.iha.wcc | CarActivity | Method | doStop | Send a request to the car to stop all motors. |
| com.iha.wcc | CarActivity | Method | doPhoto | Send a request to the car to take a photo to store on the SD card. |
| com.iha.wcc | CarActivity | Method | doHonk | Send a request to the car to generate a a sound from the car (honk). |
| com.iha.wcc | CarActivity | Method | send | Send a message using the socket connection to the Arduino. |
| com.iha.wcc | CarActivity | Method | updateViewDirection | Update the displayed direction on the view. |
| com.iha.wcc | CarActivity | Method | updateViewSpeed | Update the displayed speed on the view. |
|  |  |  |  |  |
| com.iha.wcc.job.car | Car | Variable | speed | Current speed of the car. |
| com.iha.wcc.job.car | Car | Variable | lastDirection | Last direction used by the car. Stopped by default. |
| com.iha.wcc.job.car | Car | Variable | lastSens | Last sens where the car was going. Useful to avoid change of sens after a LEFT/RIGHT action. |
| com.iha.wcc.job.car | Car | Method | setSettings | Change the settings of the car. Check each setting before set to protect the motor engine. |
| com.iha.wcc.job.car | Car | Method | calculateSpeed | Calculate the new speed. |
| com.iha.wcc.job.car | Car | Method | \_accelerate | Increase the speed depending on the sens of the car. |
| com.iha.wcc.job.car | Car | Method | \_decelerate | Decrease the speed depending on the sens of the car. Can also change the sens of the car. |
| com.iha.wcc.job.car | Car | Method | \_turn | Update the speed when turning depending on the sens of the car. |
| com.iha.wcc.job.car | Car | Method | \_stop | Stop the car. |
| com.iha.wcc.job.car | Car | Method | \_saveNewDirection | Update the lastDirection for the next action. |
| com.iha.wcc.job.car | Car | Method | \_saveNewSens | Update the lastSens when the lastDirection is a sens. (Not a simple direction/action) |
|  |  |  |  |  |
| com.iha.wcc.job.car | Camera | Constant | DEFAULT\_CAMERA\_STREAMING\_URL | Linino default camera streaming address. |
| com.iha.wcc.job.car | Camera | Constant | DEFAULT\_CAMERA\_PICTURE\_URL | Linino default camera picture address. |
| com.iha.wcc.job.car | Camera | Constant | DEFAULT\_CAMERA\_WIDTH | Default width used for the camera stream. |
| com.iha.wcc.job.car | Camera | Constant | DEFAULT\_CAMERA\_HEIGHT | Default height used for the camera stream. |
| com.iha.wcc.job.car | Camera | Constant | DEFAULT\_CAMERA\_FPS | Default FPS used for the camera stream. |
| com.iha.wcc.job.car | Camera | Method | getCommand | Return the command to execute to start the camera video stream using values in SharedPreferences. |
|  |  |  |  |  |
| com.iha.wcc.job.car | Linino | Constant | DEFAULT\_NETWORK\_IP | Linino default IP address for its hotspot. |
| com.iha.wcc.job.car | Linino | Constant | DEFAULT\_NETWORK\_PORT | Linino default port number for its hotspot. |
| com.iha.wcc.job.car | Linino | Constant | DEFAULT\_SSH\_USER | Linino default user for SSH. |
| com.iha.wcc.job.car | Linino | Constant | DEFAULT\_SSH\_PASSWORD | Linino default user password for SSH. |
|  |  |  |  |  |
| com.iha.wcc.job.camera | TakePictureTask | Method | execute | Take a picture from the camera and store it. |
|  |  |  |  |  |
| com.iha.wcc.job.camera | MjpegVideoStreamTask | Method | execute | Load the stream from an URL and set it to a MjpegView. |
|  |  |  |  |  |
| com.iha.wcc.job.ssh | SshTask | Method | execute | Run a SSH command using Jsch library to start the camera stream video. |

This table contains all constants, instance variables and method that are the most important in the Android application.

## Sequence diagrams

### Move



Figure 13 - Sequence – Move (1.0)

All the actions such as go forward, backward, left, right use the same pattern, this sequence diagram is true for all these actions. It’s basically just adding the action to a queue after calculated the speed to use. Once the query is in the queue, the queue manager will automatically use the socket to send the query as soon as possible. This system is really efficient, the response time is less than one second. (~*100ms*)

The actions such as honk and take photo won’t calculate the speed, it’s just add the action to the same queue.

### Initialize car settings



Figure 14 - Sequence - Initialize car settings (1.1)

This sequence diagram is the same for both Activity diagrams “Initialize car” and “Change settings” because it’s the same process, the process is just called at a different time.



Figure 15 - Sequence - Take picture 1.2

In this version, the Android application use the pictures from the camera as a video stream, there is no real video stream, but it looks like for the client.



Figure 16 - Sequence - Take picture 2.0

In this version, the process is the same than previously but we don’t display the picture, we save it on the Android phone. The process is not running in a loop anymore, but only when the client really want to take a picture.



Figure 17 - Sequence - Video stream

In 2.0, we use a real video stream using MJPEG video format, we use a class in the Android program to manage this kind of stream, because it’s not supported natively by Android.

2. Arduino

# Class diagram



Figure 18 – Arduino class diagram

The Arduino program doesn’t use really classes. But to understand and represents in an easier way we decided to use a Class diagram.  
The class Arduino is the sketch loaded on the Arduino YUN board, it includes many other libraries such as **Adafruit\_DCMotor**, **YunClient**, **YunServer**, **Bridge** and more.  
**YunServer** is basically the listening server waiting for a client. Once it gets a client it instantiates a **YunClient**. This client is basically the connected client, the client can use sockets or HTTP requests (and more) protocols to communicate with the YUN.

We developed only the “class” Arduino here, other are provided from libraries. The **setup**() method is actually the main method, automatically called once the YUN is powered. It initializes all instances variables and run the **loop**() method. This method will wait until a client is available then call the **process**() method to process the commands. We decided to use *protected* methods such as doForward(), etc. to represents functions that doesn’t really exists, they are processed into the **process**() method depending on the received command. We decide to use *private* methods to represent real functions. We used a doStop() function because it’s called several times and not only processed on the **process**() method.

# Schematics (Electronic)

## Arduino Yun

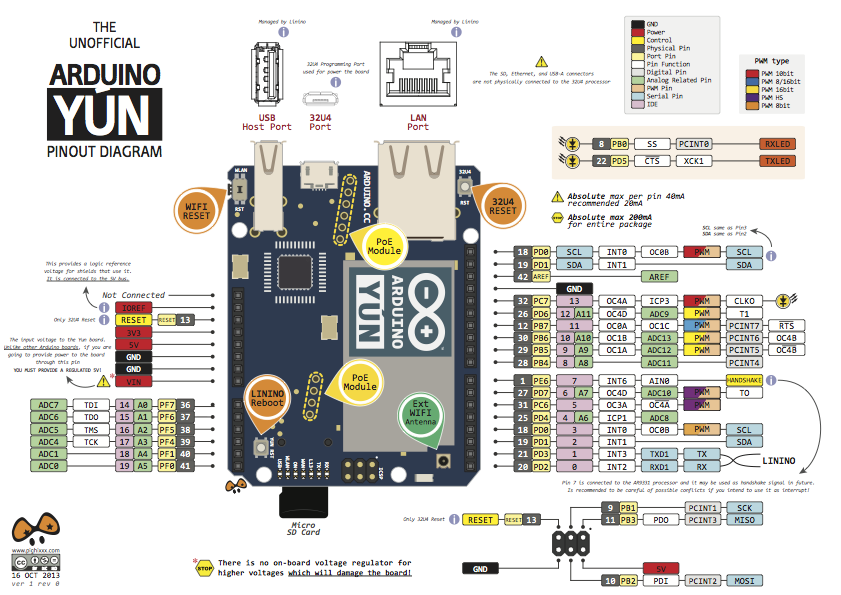
Arduino board is one of the most important parts in the car. The board functions as a data receiver and command sender.

Figure 19 - Arduino YUN

To provide the connection between the phone and the board we used a built-in Wi-Fi chip (grey metal cover with logo on it).

To provide the physical connection between the board and two motors we used a special motor shield. The shield is an important part to connect the board with an engine of the car.

As it can be observed from the schematics, Arduino has a variety of ports for a number of connections:

* Standard Ethernet (LAN) Port
* USB Host Port
* Micro USB Port
* Micro SD Card Slot
* Pins for adding shield and connecting components

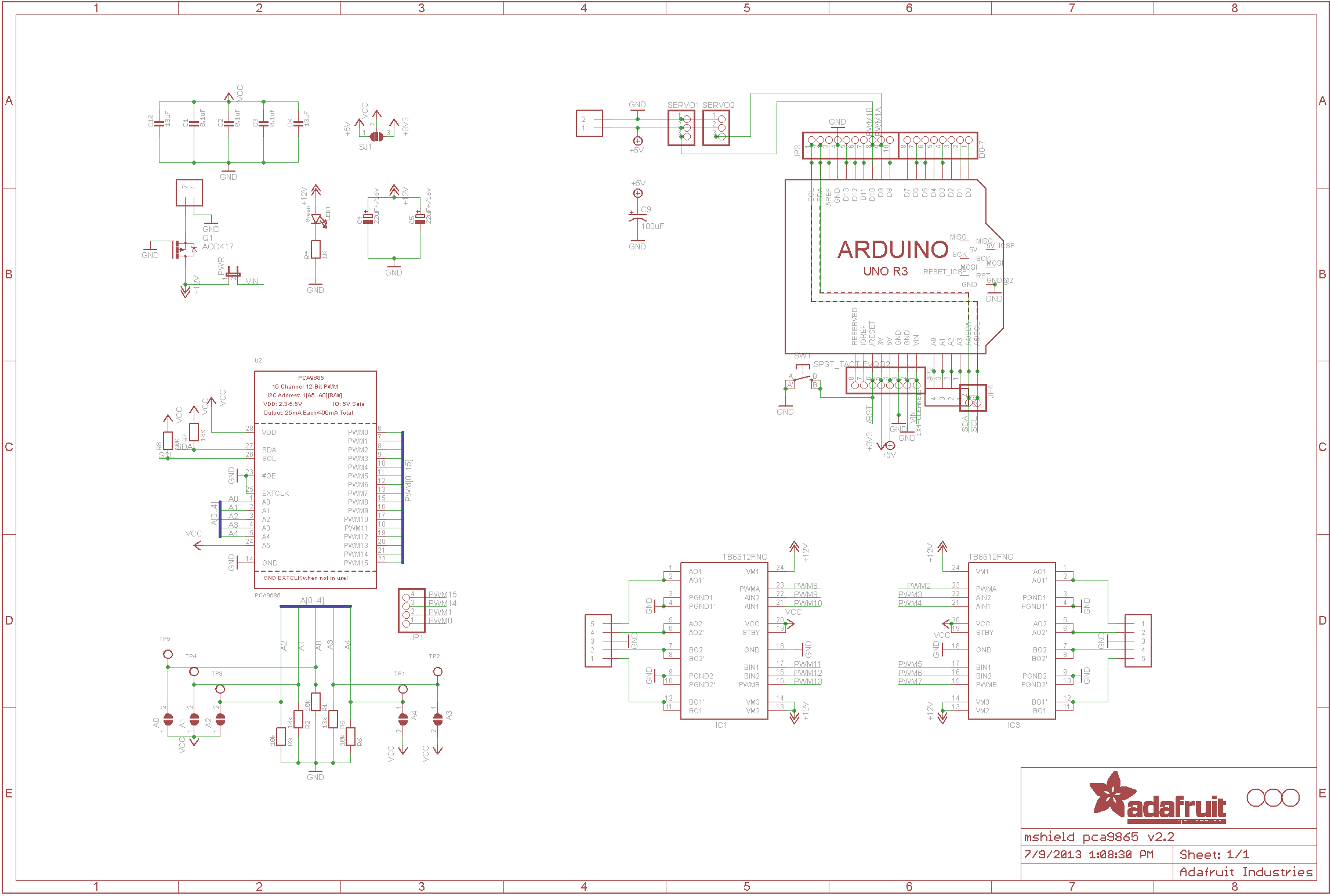
Arduino has three reset buttons on-board (Wi-Fi reset, OS reset, Processor reset).

## http://arduino.cc/en/uploads/Main/BridgeInShort.pngArduino YUN – communication between ATmega32u4 and AR 9331

Figure 20 - Communication between ATmega32u4 and AR 9331

Microcontroller ATmega32 communicates with the processor AR 9331 by bridge connection. That gives a possibility to run shell scripts, communication with network and receiving information from processor to microcontroller. Port USB, host, network interface and slot for SD cards are controlled by AR 9331. But we have access from ATmega using bridge.

## Motor shield



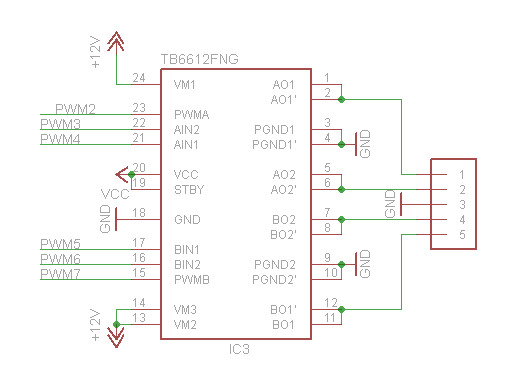


Figure 21 - TB6612FNG

Figure 22 - PCA 9685

Figure 23 - Arduino YUN

The schematic above provides information about the communication between Arduino and shield. Shield on board has PCA 9685. It is an I2C-bus controller 16-channel (12 PWM). PWM allows us to set from 0% to 100% power. It’s also possible to set frequency (40Hz-1000Hz) but it is the same on all of the pins. To simplify, Arduino communicate with PCA 9685 by I2C protocol and this bus controller controls two TB6612FNG dual motor drivers. One motor driver allows us to control max 2 DC motors. (So we can control all PWN on Arduino Board. Minimum connection to shield are pins SDA an SCL- and that’s all)

## Battery

We used a 9.7V standard battery for this project. It is a Ni-MH type of battery which has less capacity than a 1 time battery. They lose some energy even If not in use. This battery has only one great disadvantage: when it is fully charged – it has a higher voltage. Full 9V battery can have even 11,2V. Because battery life is not very important for our project, we decided to use them for test. They were available at University. For the contest instead of Ni-MH we would use Li-Po.

## G:\CESI\PEE\Aarhus\Cours\ETCCCP\02 - Project Working part of the ETCCCP course\Project\Documentation\Pictures\DSC_2243.jpgVoltage regulator

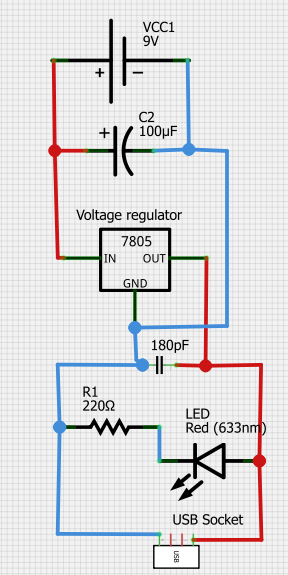
Figure 24 - Voltage regulator

Figure 25 - Voltage regulator schematic

We use TS7805 substitute of LM7805. Popular and cheap component. Output voltage 5.00±0.25 V. Input voltage between 7 - 25 V. We need it to supply an Arduino board (otherwise the first tests were done with Arduino connected through wire). Output voltage 5V is in USB. To connect it to Arduino we use wire with plug USB (Standard A)-USB (B-plug).

## Camera



Figure 26 - Communication between the camera, Arduino YUN and the Android application

This diagram shows us how works the communication between the camera and the YUN and between the YUN and the Android application.

### Connection to Arduino

We used SSH to configure Arduino to be able to recognize the webcam and start streaming. We used the software *Putty.* Putty is a SSH and Telnet client developed for windows platform.

After connecting the computer to the Arduino’s Wi-Fi, we can connect through putty using the local address: arduino.local and the SSH port, 22.

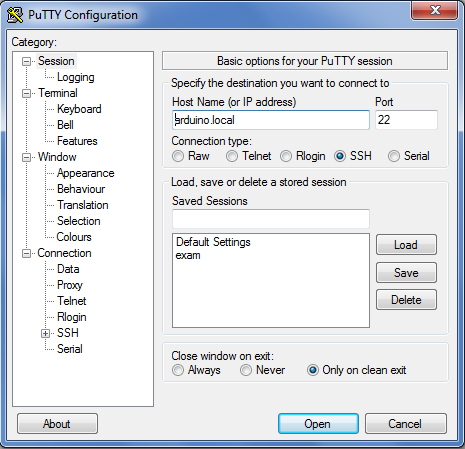


Figure 27 - Use PuTTY to access to the embedded Linino

### Installing drivers

The first thing to do is to install the driver for the camera, our camera is supported by the driver UVC. We installed it using the command:

opkg install kmod-video-uvc

Then, after plugging the camera into the Arduino board, we tried to see if the camera was detected and the drivers correctly installed, using this command:

dmesg

### SD card configuration

In order to have a stream we need a temporary memory to stock the stream, so we can access it using a webpage.

We have inserted then the SD card and appeared here:

/dev/sda1

We create a new folder for mounting the SD card:

mkdir /mnt/sda1

mount /dev/sda1 /mnt/sda1

### Streaming tool: MJPG-streamer

MJPEG-streamer allowed us to stream a video or take snapshot from a webcam and put it on a local server. We first downloaded the software:

wget <http://www.custommobileapps.com.au/downloads/mjpg-streamer.ipk>

And we installed it:

opkg install mjpg-streamer.ipk

Here you can see the command line which allows us to start streaming:

mjpg\_streamer -i "input\_uvc.so -d /dev/video0 -r 320x240" -o "output\_http.so -p 8080 -w /mnt/share"

* **-d /dev/video0** : The device/webcam
* **-r 320x240** : The streaming resolution
* **-p 8080** : Output port for the local server
* -**w /mnt/share** : Destination of output

### Getting the Stream/Snapshot

To access the stream or a single snapshot, we just have to reach on the phone those address:

[**http://arduino.local:8080/?action=stream**](http://arduino.local:8080/?action=stream)

We got a MJPG encoded stream, only the web browser Mozilla Firefox has been able to decode it, but with the right library we have been able to access it on the phone.

[**http://arduino.local:8080/?action=**](http://arduino.local:8080/?action=stream)**snapshot**

This one takes a single snapshot, and displays it at this address.

### Camera automatically started from Android application

Because we don’t want to use PuTTy or another soft to start the camera, we developed a class inside the Android application to automatically start the video stream on the camera once the application started.

We used an external library to connect to the Linino and execute commands via SSH. The library use the SSH-2 protocol.

1. Glossary

**WCC**: Wi-Fi controlled car, the system as a whole.

**Linino**: Linux OS embedded on the Arduino board.

**Arduino YUN**: Arduino is the microcontroller used to control the car. We chose the YUN model.

**SSH**: Secure Shell is a secured communication protocol to secure a communication on an unsecured network. All the communication is encrypted.

1. References
2. <https://en.wikipedia.org/wiki/4%2B1_architectural_view_model>
3. <http://arduino.cc/en/Main/ArduinoBoardYun?from=Main.ArduinoYUN>
4. <http://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/>
5. <http://www.adafruit.com/datasheets/PCA9685.pdf>
6. <http://www.jcraft.com/jsch/>

1. **Source** : Wikipedia [↑](#footnote-ref-1)
2. *The other use case diagrams are not in this document but in the “2 - WCC - Requirements specification”.* [↑](#footnote-ref-2)