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 | TODO |
| Audit | TODO |
| 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. |

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 |  |

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

**Place and date:**

Authors Supervisor

Contents

[1. Opening 5](#_Toc374396349)

[1.1. Purpose 5](#_Toc374396350)

[1.2. Reading instruction 5](#_Toc374396351)

[2. 4+1 view model 6](#_Toc374396352)

[2.1. What is 4+1? 6](#_Toc374396353)

[3. Android 7](#_Toc374396354)

[3.1. Scenarios 7](#_Toc374396355)

[3.2. Process view 9](#_Toc374396356)

[3.3. Physical/Deployment view: 10](#_Toc374396357)

[3.3.1. Deployment diagram 10](#_Toc374396358)

[3.4. Development/Implementation view: 10](#_Toc374396359)

[3.4.1. Component diagram 10](#_Toc374396360)

[3.4.2. Package diagram 11](#_Toc374396361)

[*3.5.* Logical view 12](#_Toc374396362)

[3.5.1. Class diagrams 12](#_Toc374396363)

[3.5.1.a. Package com.iha.wcc 13](#_Toc374396364)

[3.5.1.b. Package com.iha.wcc.job 14](#_Toc374396365)

[3.5.1.c. Detailed explanations about methods and instance variables 15](#_Toc374396366)

[3.5.2. Sequence diagrams 16](#_Toc374396367)

[3.5.2.a. Move 16](#_Toc374396368)

[3.5.2.b. Initialize car settings 17](#_Toc374396369)

[4. Arduino 20](#_Toc374396370)

[4.1. Class diagram 20](#_Toc374396371)

[4.2. Schematics (Electronic) 21](#_Toc374396372)

[4.2.1. Arduino Yun 21](#_Toc374396373)

[4.2.2. Arduino YUN – communication between ATmega32u4 and AR 9331 22](#_Toc374396374)

[4.2.3. Motor shield 23](#_Toc374396375)

[4.2.4. Battery 24](#_Toc374396376)

[4.2.5. Voltage regulator 24](#_Toc374396377)

[4.2.6. Camera 24](#_Toc374396378)

[4.2.6.a. Connection to Arduino 24](#_Toc374396379)

[4.2.6.b. Installing drivers 25](#_Toc374396380)

[4.2.6.c. SD card configuration 25](#_Toc374396381)

[4.2.6.d. Streaming tool: MJPG-streamer 26](#_Toc374396382)

[4.2.6.e. Getting the Stream/Snapshot 26](#_Toc374396383)

[4.2.7. Overview 27](#_Toc374396384)

[5. Glossary 28](#_Toc374396385)

[6. References 28](#_Toc374396386)

Tables of figures

[Figure 1 - 4+1 view model diagram 6](#_Toc374396331)

[Figure 2 - Use case - System version 1.0 7](#_Toc374396332)

[Figure 3 - Use case - System version 2.1 8](#_Toc374396333)

[Figure 4 - Communication between Android and Arduino 9](#_Toc374396334)

[Figure 5 - Deployment diagram 10](#_Toc374396335)

[Figure 6 - Component diagram 10](#_Toc374396336)

[Figure 7 - Packages diagram 11](#_Toc374396337)

[Figure 8 - Class diagram - Overview 12](#_Toc374396338)

[Figure 9 - Class diagram - Package com.iha.wcc 13](#_Toc374396339)

[Figure 10 - Class diagram - Package com.iha.wcc.job 14](#_Toc374396340)

[Figure 11 - Sequence - Move 16](#_Toc374396341)

[Figure 12 - Sequence - Initialize car settings 17](#_Toc374396342)

[Figure 13 - Sequence - Take picture 1.2 17](#_Toc374396343)

[Figure 14 - Sequence - Take picture 2.0 18](#_Toc374396344)

[Figure 15 - Sequence - Video stream 19](#_Toc374396345)

[Figure 16 - Class diagram 20](#_Toc374396346)

[Figure 17 - Arduino YUN 21](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#_Toc374396347)

[Figure 18 - Communication between ATmega32u4 and AR 9331 22](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#_Toc374396348)

TODO

*This page is about the tasks to do, some of these can be done only after a long time so have a page for list them is useful. If you think that some tasks to do are missing, just add them.*

* Check pagination.
* Check spelling by 3 members of the team minimum.
* Check internal “TODO”. (On others pages)
* Check PDF generation. (Document well created, all figures readable, not truncated, etc.)
* Delete this page before generate PDF version or send the final release.
* Regenerate pagination.

***This document belongs to the project manager. If you want to change it, you have to tell to him and explain your changes in the document history part. Be sure to update the last version of this document.***

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)

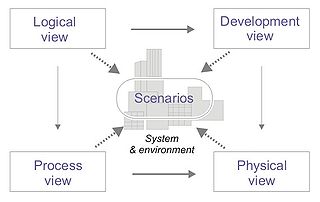


Figure 1 - 4+1 view model diagram

* ***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.

# Process view



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, the fragments 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



Figure 8 - Class diagram - Overview

### Package com.iha.wcc



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

### Package com.iha.wcc.job



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

This package is subdivided in other packages such as “*com.iha.wcc.job.carCommunication*”, “*com.iha.wcc.job.network*” and “*com.iha.wcc.job.car*”. These classes have a role well defined, a job. They could use each other but it’s not useful in our application. They are used directly by the *activities*.

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.

### Detailed explanations about methods and instance 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 | Method | initializeCarSettings | Get the car settings from the local phone settings and send them to the car. |
| 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 | \_saveNewDirection | Update the lastSens when the lastDirection is a sens. (Not a simple direction/action) |

## Sequence diagrams

### Move



Figure 11 - Sequence - Move

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 12 - Sequence - Initialize car settings

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 13 - 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 14 - 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 15 - 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

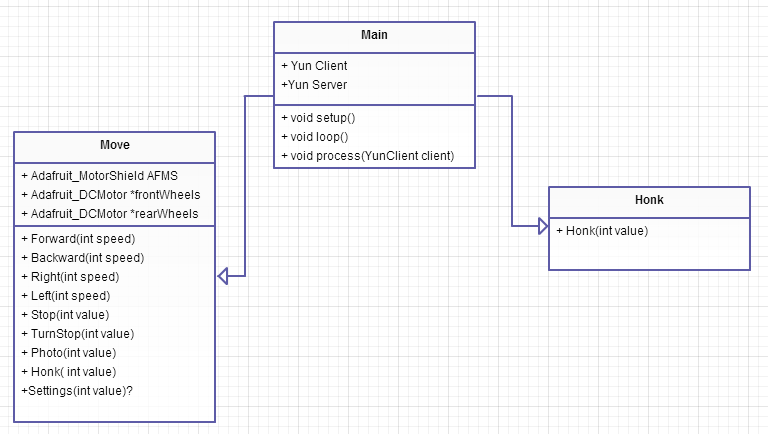


Figure 16 - Class diagram

AdaFruit library function will be add.

# Schematics (Electronic)

## Arduino Yun

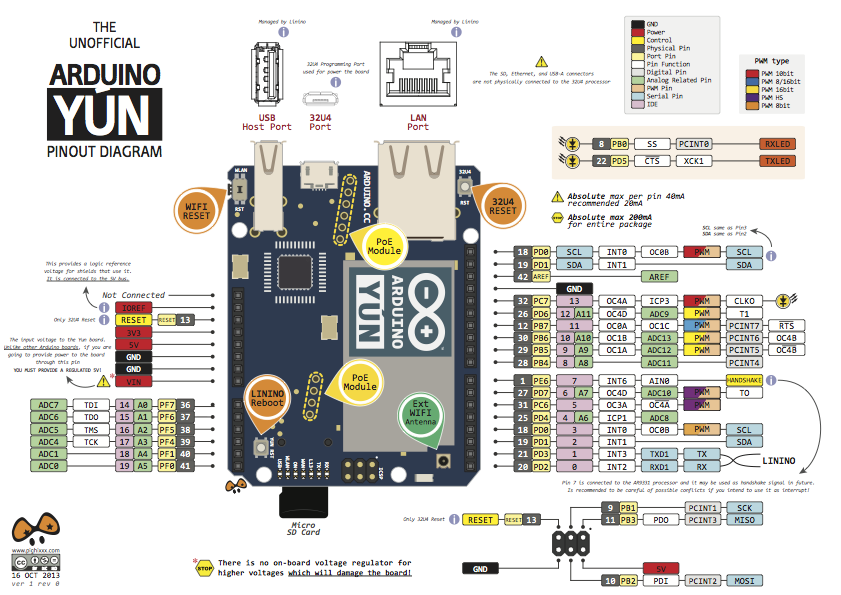
Arduino board is one of the most important parts in the car. The board functions as a data receiver and command sender.

Figure 17 - 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

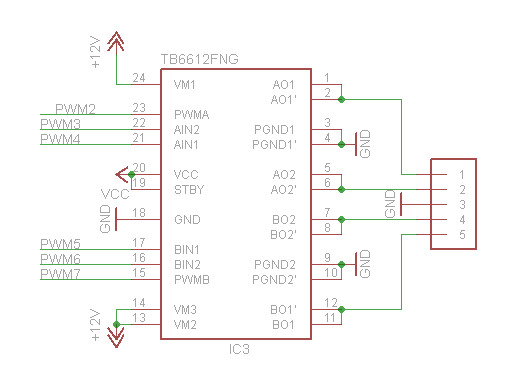
Figure 18 - Communication between ATmega32u4 and AR 9331

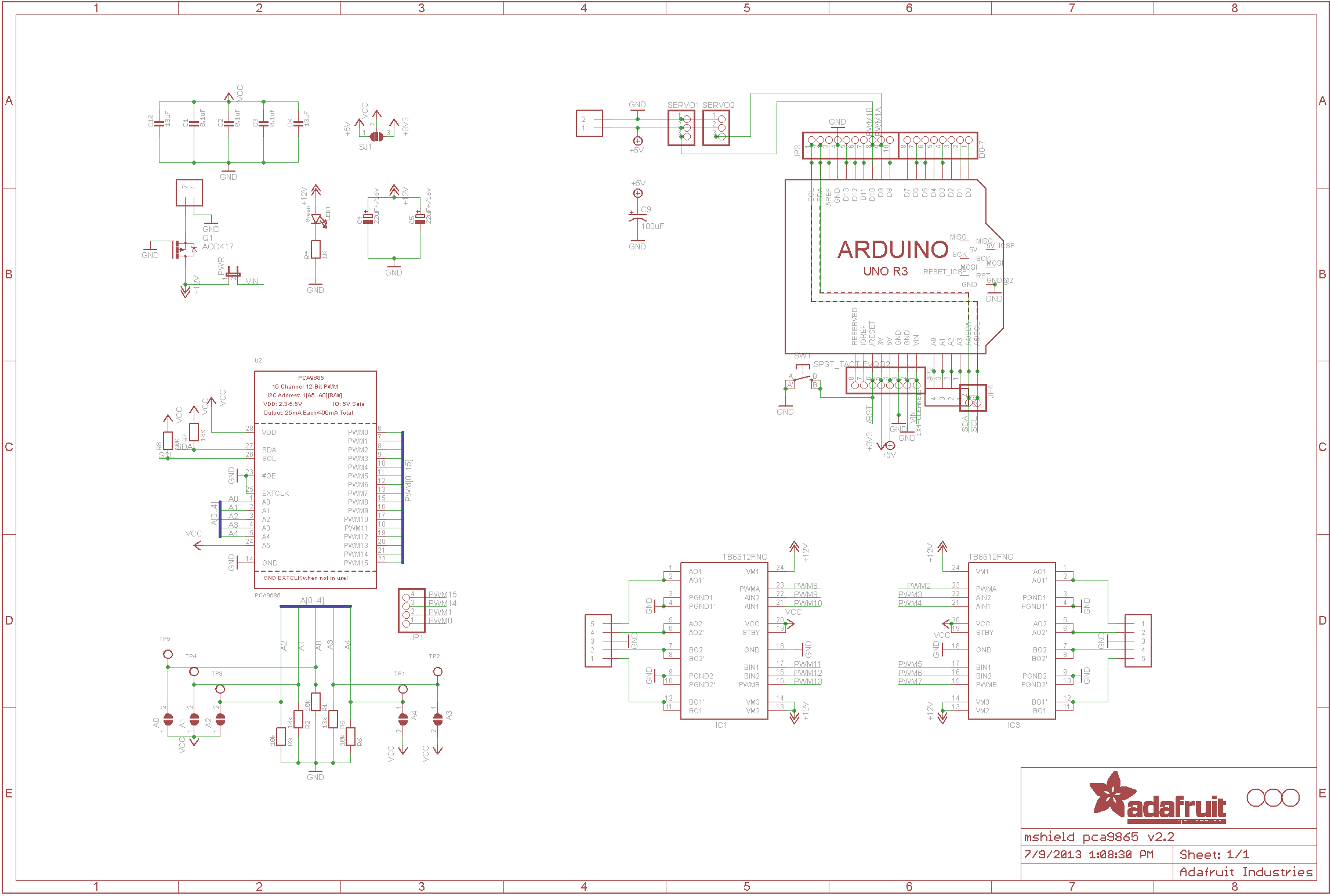
Microcontroller ATmega32 communicates with the processor AR 9331 by bridge connection. That gives a possibility to run shell scripts, communication with network and receiving information from processor to microcontroller. Port USB, host, network interface and slot for SD cards are controlled by AR 9331.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

## Motor shield

2. PCA 9685

3. TB6612FNG





1. 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.

## Voltage regulator

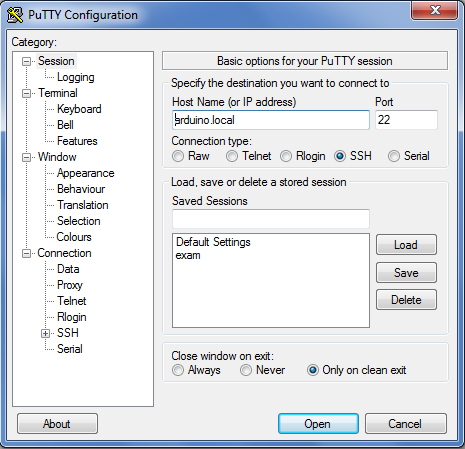
## Camera



### 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 wifi, we can connect through putty using the local address : arduino.local and the SSH port, 22.



### 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 take a single snapshot, and display it at this address.

## Overview

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.

1. References
2. Wikipedia:
   * <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>

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)