# UbiComp09 project: aMir

**Group B**

Amirhosein Azarbakht
Efraín Calderon
Karl Löfholm
Alexander Neumann

# Introduction

In context of the course "Ubiquitous Computing" during the autumn term 2009 we developed a prototype of an augmented mirror called "aMir".

The project took five weeks. During that time we have developed a concept, tried out different hardware solutions and evaluated the system itself. In addition a user interface was designed and implemented. In the next chapter we want to picture the concept of aMir and give a brief introduction to the used hardware components. In the third chapter the realization approach is described. Afterwards, we present the result of our work. This includes the product aMir, but also includes the evaluation of approaches we had to evaluate and reject because of several issues. In the last chapter, we want to give a summary and suggest possible further work that can be done within the project context.

## Concept

The concept of having a mirror, which will serve as an informative medium, and assist people to be punctual, while relieving the stress of it, was chosen after a number of rounds of brainstorming and then receiving target user feedback. The process contributed to the evolutionary transformation of the raw idea into a fully-fledged product that is customized to the real needs of the individuals.

This product is for everyone with a lot on their minds when preparing to go out through the door. It is perhaps best suited for people who live in bigger cities with well-developed public transportation system. In the city of Gothenburg, for example, the majority of the people use the trams and buses as their primary choice of transport. They, therefore, need to look up the schedule every time they want to go somewhere. This usually takes time, as well as energy, and causes a pause, but by integrating a schedule display into a mirror saves them a lot of trouble. The displaying of the weather is used to give the person a sense of how warm they should dress.

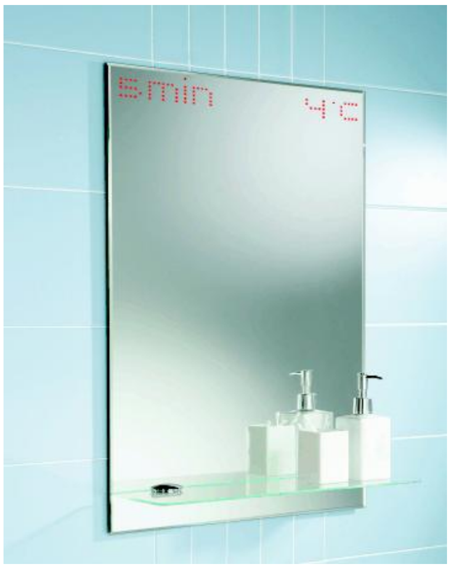


**Figure 1 - First concept sketch of aMir**

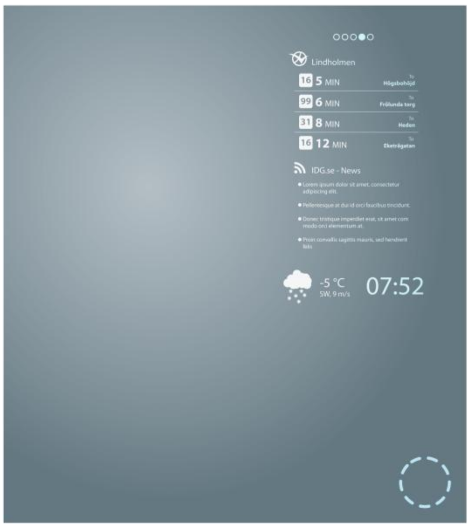


**Figure 2 - Second concept sketch of aMir**

## Approach

Following we want to explain the project approach. We planned to start this project with having an introduction phase, which we used to define different key aspects of this product further. What materials to use, in what context should the mirror be used, what kind of input source is the most efficient and etc. After this we went through a design phase and started to design different kinds of solutions and after evaluating them, chose one design to realize further. Throughout this realization phase, we continuously evaluated the prototype and according to the evaluations, made small changes to the design.

We decided to split up into smaller groups to divide responsibilities. One group focused on all hardware issues including finding suitable material and evaluating it. The other group focused on the user interaction and the software design and implementation. The advantage of this approach is quite obvious. The limited time amount made it impossible for every group member to get into every field of work. In addition it was possible to make more precise assumptions about the actual state and the work that had to be done.

Amir and Efraín were in the hardware group. The task was to research and evaluate possible hardware solutions and testing material concerning its usefulness for the project. The software group consisted of Alex and Karl, to develop the user interface and deciding the information that should be shown was their main task.

However, the groups were just a rough orientation. More than just once we assisted each other with every kind of task disregarding the associated group.

**Table 1. Work Packages. The work was divided to 10 work packages.**

| **Name** | **Category** | **Description** | **Responsible** | **Duration** |
|---|---|---|---|---|
| WP1 | Prototyping | Getting to know the used technology related work and thinking about solutions | everyone | 2 weeks |
| WP2 | Hardware | Finding and testing a solution for a user detection functionality | Efraín | 3 weeks |
| WP3 | Hardware | Research and designing a hardware architecture of the mirror | Amir | 2 weeks |
| WP4 | Hardware | Getting special hardware like mirror and case | Karl, Alex | 2 weeks |
| WP5 | Hardware | Designing and implementing the control of the display | Efrain, Amir | 3 weeks |
| WP6 | Software | User Interface design | Karl | 1 week |
| WP7 | Software | Implementing the user interface | Alex | |
| WP8 | Documentation | Creating a website for the project | Karl | 2 weeks |
| WP9 | Documentation | Creating exhibition material and planning the presentation | everyone | 1 week |
| WP10 | Documentation | writing the report | everyone | 3 weeks |

**Table 2. Estimated schedule. This table shows how the work packages were spreaded across the time that was available.**

| **WW47** | **WW48** | **WW49** | **WW50** | **WW51** |
|---|---|---|---|---|
| WP1 | | | | |
| | WP2 | WP2 | WP2 | |
| WP3 | WP3 | | | |
| | | WP4 | WP4 | |
| | WP5 | WP5 | WP5 | |
| | WP6 | | | |
| | WP7 | WP7 | WP7 | |
| | | | WP8 | WP8 |
| | | | WP9 | WP9 |
| | | | | WP10 |

## The Design Process

The primary idea was an augmented mirror with Västtrafik information. The first idea to implement it was using an LED dot matrix, configured to show a number - indicating the minutes left to next bus departure. Then we thought it would be more beneficial if it could show weather information – current temperature of the city– as well. Having searched for similar previous projects, we found that it would be somehow banal and not-so-new; so, we came up with the motive of making something different, instead of just repeating what that has already been done.

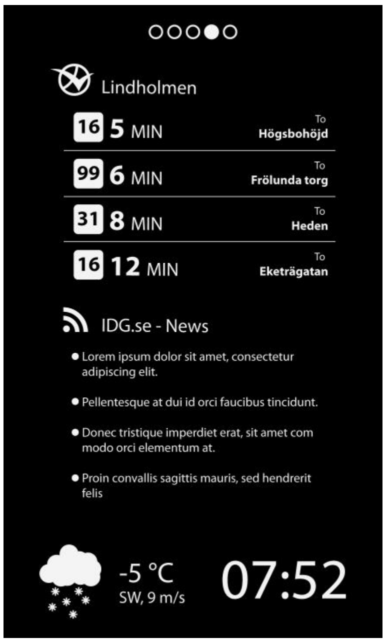
Obviously, we did not want to reinvent the wheel. Consequently, we went for end-user preferences and did a survey to see what other features they would like to have on the mirror, and the result was, on top of all, the view of the back of their head, and secondly, weather information, so that they would exactly know how cold it feels like outside. At some point, there was a suggestion that –instead of just showing the temperature in the form of a number– it would be more conventional to have a view of how people are dressed-up outside, i.e. a live cam view of a populated area in downtown, e.g. Brunnsparken in Gothenburg.

For the back view feature, we opted to use an LCD display instead of the old stylish LED dot matrix. Then, the mirror had to be a good see-through mirror, a.k.a. spy glass or two-way mirror. We tested using a web-cam to capture the back view and showing it on the LCD display, and also, using a projector, -in front or from back- but, neither resulted in acceptable resolution. The idea of using a projector instead of LED/LCD was adopted to take advantage of the next generation cell phones, which, presumably, will have projection feature. That was an option, regarding near future expected technology possibility. Though, it proved to be unpromising. Unfortunately it is quite expensive to get a mirror or window glass that has the ability to show a projector image. We did some research and found an interesting solution developed by G+B pronova GmbH called HoloPro[i]. They developed mirrors, window glass and special foils that can be attached to ordinary mirrors or windows to show a projector image. Concerning the possibility of mobile phones having a built-in projector this solution seems quite promising. Unfortunately the creation process is still expensive. A 30" mirror would cost about 2900€ what is not covered by our budget.

At some point, we thought of a solution to address the problem of mirrors getting foggy/misty in a steamy bathroom environment. A common solution is to use something like the voltage-resistant wires on back window of cars; which gets warmed up and warms up the glass, when electrical power passes through them. It has long been in use in car industry, hence, would not have contributed to the modern looks of our prototype. Moreover, we managed to find a demister pad as an alternative to voltage-resistant wires.

Having re-thought of the whole concept once again, and having developed it, we could draw the conclusion that, now that we will be using an LCD as the display, we would go with the set of features comprised traffic information, weather information, date and time, and news headlines. The resulting prototype was splendid, and was kept as the eventual pleasing prototype.

The design of the graphical user interface was inspired by “Dashboard software Apple Widgets”. These are lightweight single-purpose applications, which are used to provide the user with a particular function such as weather information, ”post-it notes” and a calculator. The concept of aMir is very similar to the idea of using widgets on your computer desktop, supporting the functionality you often need. We just brought in to another context, the bathroom mirror. The Apple widgets have a very clean look and are often designed to be very user friendly and



- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Pellentesque at dui id orci faucibus tincidunt.
- Donec tristique imperdiet erat, sit amet com modo orci elementum at.
- Proin convallis sagittis mauris, sed hendrerit felis

**Figure 3 - Graphical user interface**

the usage of the widgets is most often very clear to the user. This was also something that we wanted to bring in to the graphical interface of aMir. One constraint, though, was that we only could use black and white colors and text and icons had to be really big in order to be visual through the two-way mirror. The decision to display the information in the up right corner was also based on testing were the information displayed would interfere with the reflection of the mirror was minimal.

To interact with the mirror we also designed a framework of physical interactions on how the user could interact with aMir. Specifications about the interactions are the following:

| **Function** | **Action** | **Requirement** |
|---|---|---|
| Activate screen | Hand in front of sensor (**1sec**) | Screen is not activated |
| Toggle between different setups | Hand in front of sensor (**0-2sec**) and then remove the hand | Screen is activated and different setups are defined |
| Deactivate screen – manually | Hand in front of sensor (**2sec**) | Screen is activated |
| Deactivate screen – automatically (time) | None | Screen is activated and no actions have been registered for some time (**15min**) |
| Deactivate screen – automatically (light) | Turn room dark (**1min?**) | Screen is activated |

## Implementation

To show the information an application written in Java 1.6 [ii]is used. This application uses the API from Västtraffic[iii] to get real time information about specific bus stops. In addition the weather.com API[iv] is used to get weather information. The RSS feed is implemented with a RSS library provided by Sun. All this APIs deliver XML documents via HTTP GET. Those XML files were parsed with the JDOM library[v]. The GUI was created with the GUI builder of the Netbeans[vi] project.

The prototypes and pretests we did were a huge help while building the actual prototype. The actual mirror consists of a mirror cabinet from IKEA called SALTSKÄR. We removed the mirror and included a double-side mirror from *Olles Glasmästeri*[vii] that was cut by them into the right shape. We used a zelo m8 display from X4-tech and mounted it behind the mirror. The controls were soldered to an Arduino with a custom made circuit to control them via serial commands. For the internet connection we used a Lenovo Thinkpad T61 that run the software and provided the GUI image as well.

**Figure 4 Development Stages**

## The following issues were tested by us:

### Visibility

When using the VGA output the GUI is easy to see even if the surrounding light is not optimal. Unfortunately we faced some problems with the display that made it necessary to use the SVIDEO output instead. The lower resolution causes a major reduction of the video quality. The RSS feed is hard to read and the average reading distance is way shorter as well.

### Robustness (Software)

The system runs very stable but depends highly on a fast Internet connection and a low answering time from the web services. Otherwise it will get stuck sometimes which reduces usage experience and make the shown data less reliable. A blocking http request blocks the serial connection as well what makes the mirror freeze for a couple of seconds. These problems were reduces by setting request time outs by hand.

### Usability

We got a positive feedback concerning the GUI. Some people would like to see the names of the person's profile that is shown. The way of interaction was graded very good as well. Unfortunately the LDR-system is not working properly for changing surrounding light. The capacity changes are often very small and it takes some calibration if the light is changing. Especially while the exhibition it took us a lot of time to recalibrate the system. Under static circumstances the reaction was very low and the interaction could be done fluently.

### Design

The approach of making the technology calm and invisible (ergo ubiquitous) seems to be a success. The display is almost invisible from the front side. While turned off it was completely invisible. Tester of the mirror mentioned that the size of the display was good and positioning was done very well as well. No tester felt annoyed by it or discovered the display without asking.

## Summary

Our intention is to provide a hidden interface, in the sense of the user not seeing the button that he/she is interacting with, or not even feeling that he is having interaction; the user's eyes will be the only door to the perception of being actively connected to the mirror. The approach is to interact with the mirror with gestures since gestures are more natural and intuitive; actually, we always use gestures to interact with other people, to show them what we want them to do, or probably just to express how are. This work can be seen as a digital artifact that will give you the information that you need in your daily life, such information is the bus scheduled, weather and some news headlines, nothing else to avoid an overhead of the user.

## Future work

One possible improvement may be taking advantage of a flat über-thin display instead of a thick hefty LCD display behind the mirror. The razor-thin display has already been developed, and hopefully, will be available widely in near future.
Next generations of the aMir might probably go wireless, and the use of cable will be minimized, which will result in an easier installation and require significantly less time and expertise.
Basic configurations in the next versions of aMir can be done in an easier way. Smart phones can be an option, though, we are investigating alternative ways to possibly bypass the use of any high-complexity device in connection to the design, and make it self-contained, and in accordance to its concept.

## Related works

**A project by Fluid Interfaces Group at MIT, called Augmented Mirror:**

http://fluid.media.mit.edu/projects.php?action=details&id=17

**A project at the University of Tokyo, called i-Mirror, based on the mirror metaphor:**

http://themeaddicts.com/pages/mirror.html

**A commercial product, called Magic Mirror:**

http://www.vrsj.org/ic-at/papers/02113.pdf

## References

i HoloPro, http://www.holopro.de/en/index/

ii Java 1.6, https://jdk6.dev.java.net/

iii Västtraffic API, http://labs.vasttrafik.se/

iv weather.com API, http://www.weather.com/services/xmloap.html

v JDOM 1.1.1, http://www.jdom.org/

vi Netbeans, http://netbeans.org/

vii Olles Glasmästeri, http://www.ollesglas.se/

# Appendix 1.

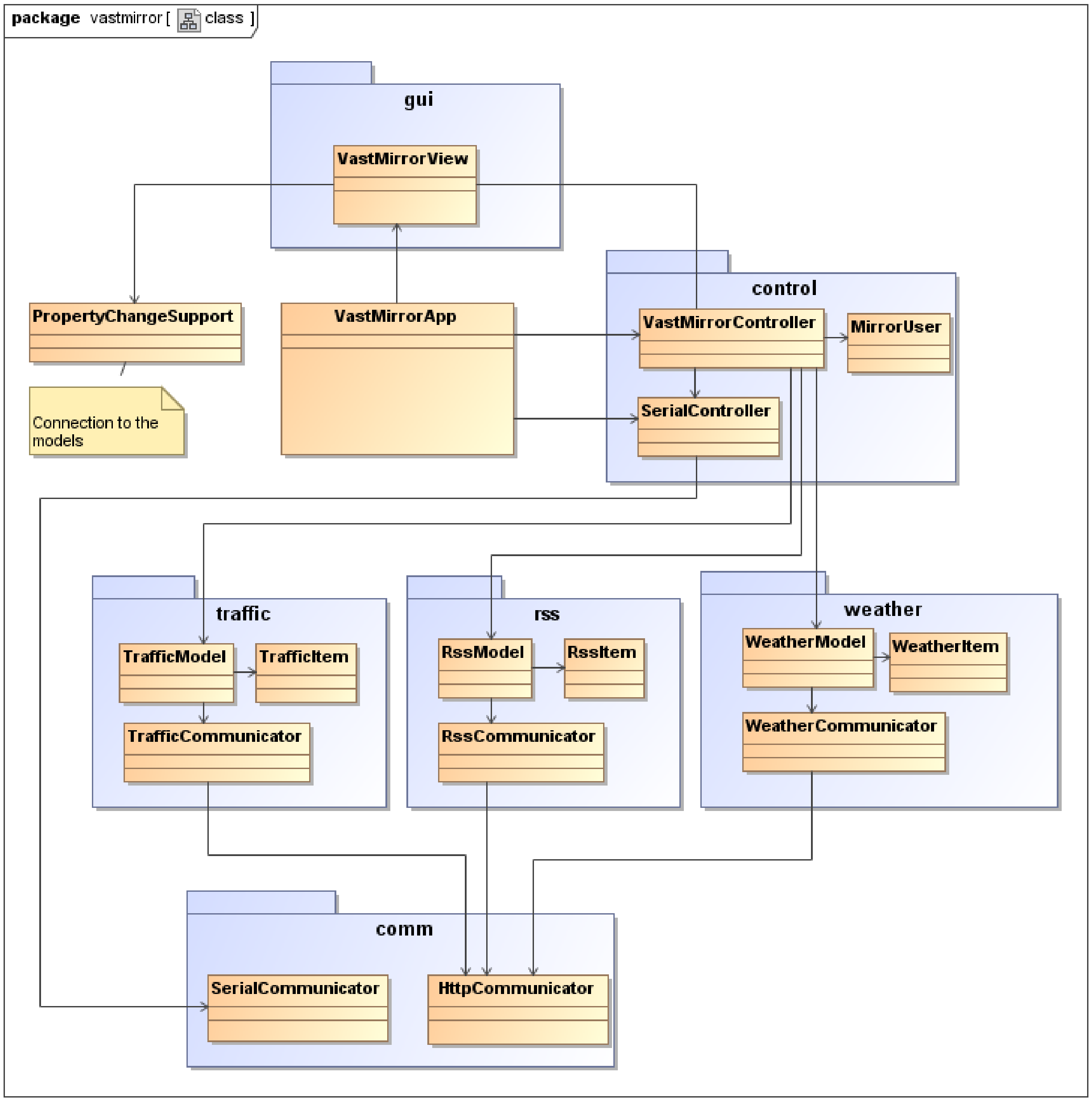


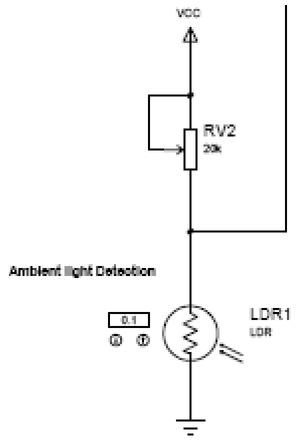
Figure 1. Domain model of the aMir prototype

# Appendix 2.

## LDR as a Button

LDRs or Light Dependent Resistors are very useful especially in light/dark sensor circuits. Normally the resistance of an LDR is very high, sometimes as high as 1000 000 ohms, but when they are illuminated with light, resistance drops dramatically.

This is a basic circuit that shows how an LDR works:



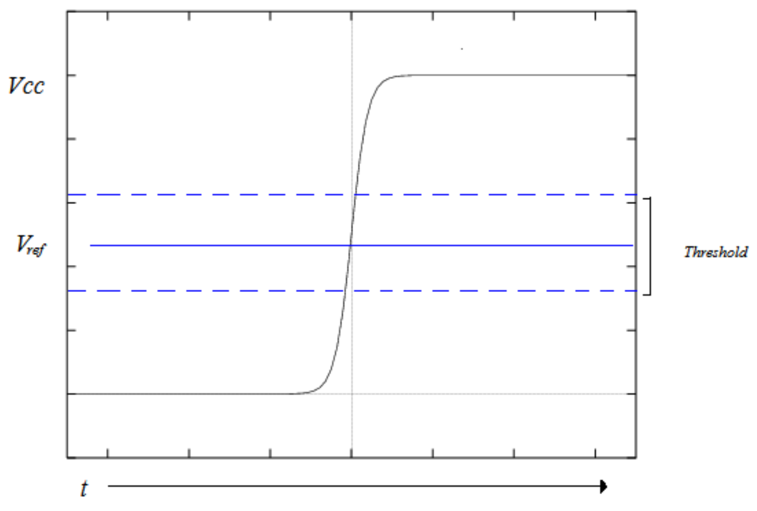
Depending on the amount of light perceived by LDR1, the voltage VLDR1 will decrease. This voltage may be expressed as

$$V_{LDR1} = Vcc \frac{R_{V2}}{R_{LDR1} + R_{V2}}$$

This means that when the LDR is exposed to light, there will be a very small voltage that would be interpreted as LOW logical level; by the other hand, when it is the darkness the LDR1 will have a higher voltage, which would be implemented as HIGH logical level.

This looks complete easy, but there is a problem: not always is possible to reach complete darkness or a full brightness. Well, there is an easy way to go through this:

1. Set the potentiometer such that the voltage in LDR1 changes enough from maximum brightness to minimum brightness. Expose the LDR to a lamp (trying to recreate the worst case scenario) and thereafter cover it.
2. Take a middle point between both voltages as the limit between Activated/Deactivated.
3. Set the size of the threshold. From this threshold will depend how sensitive is the LDR button.



Since the light conditions change constantly, aMir will calibrate itself according to the current light condition during the start up. The threshold was fix to 20 units (5 V / 1024) since it is enough to not detect movement beyond a distance of 3cm from the mirror surface.

According to the testing these were de results:

| Threshold size (units) | Max distance (approx.) | Observations |
|---|---|---|
| 0 | 25 | Sometimes Persons passing by were detected |
| 10 | 10 | Some shadows activate the LDR button |
| 20 | 3 | Sometimes some movements were not delectated |
| 30 | 1 | Almost you have to touch the mirror |
| 40 | 1 | Almost you have to touch the mirror |
| 60 | 0 | You have to touch the mirror |

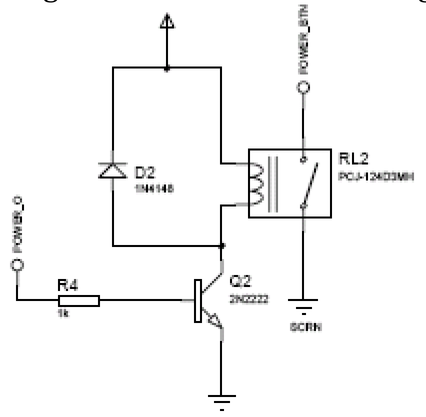
In order to manage this special kind of switches, an Arduino library was created; it was called AnalogSwitch (See Apendix).

## Hacking the LCD Screen

To show the video behind the mirror we used an LCD Screen (add more info). To be able to control the LCD from the Arduino we had to:

1. Make a circuit to activate the screen button. We used relays since it is less intrusive and still allow us to have control using the original buttons.
2. Create software able to generate the proper pulses to activate the screen. For this purpose we create the library Pulse (see Appendix)

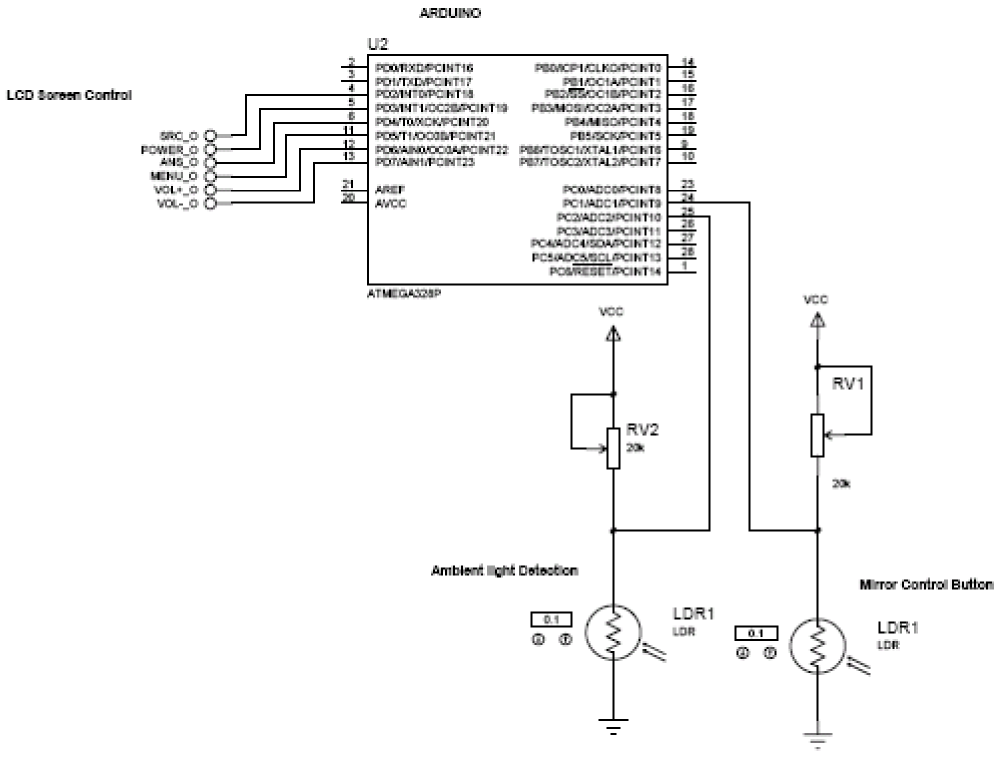
The circuit used to activate a single button is as the following:

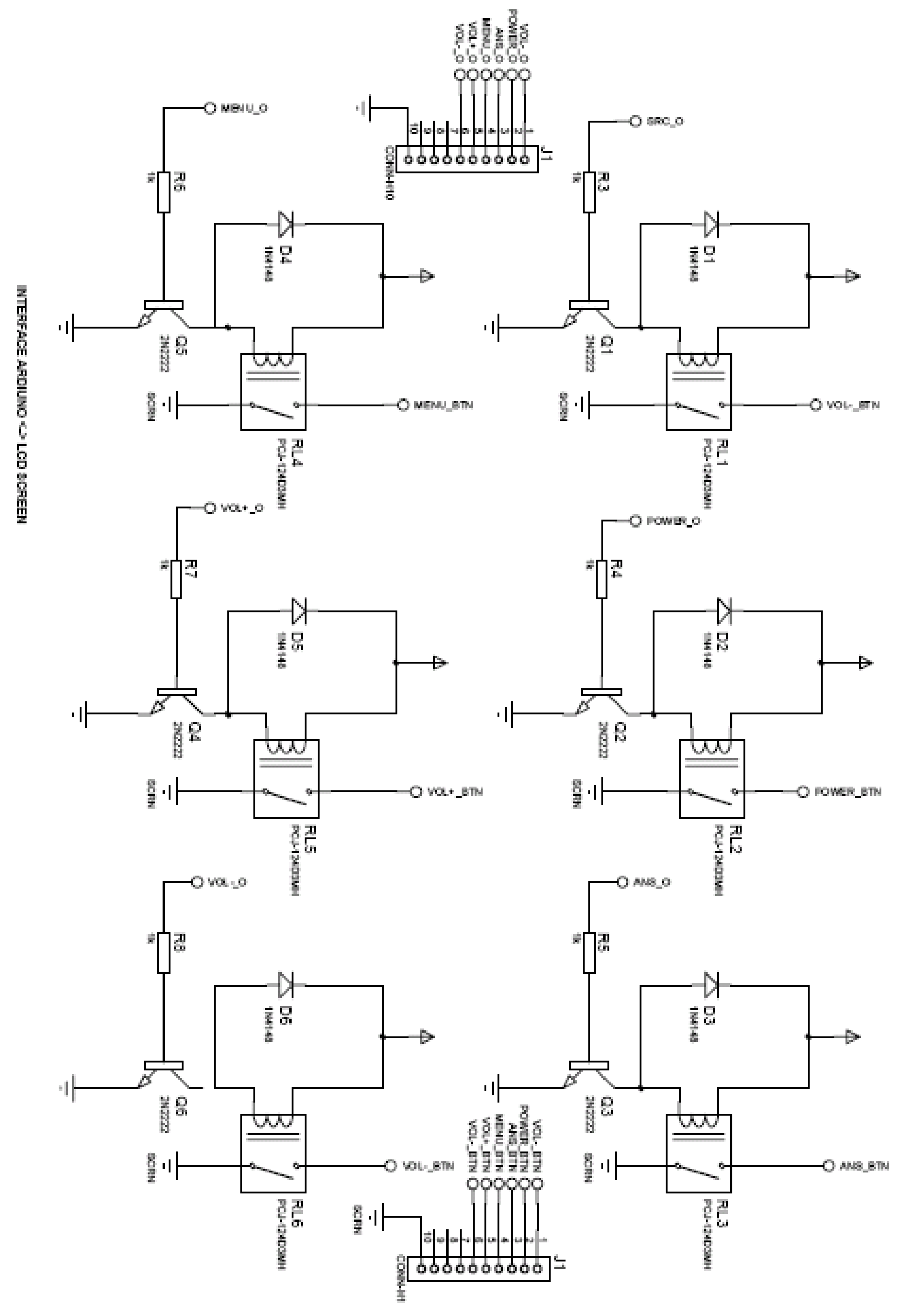


These are the buttons that were uses:

| Screen Button | uC port | Functionality |
|---|---|---|
| Source + | 3 | Change the video source |
| Power | 4 | ON: press it 3 sec.<br>OFF: press once |
| Menu | 5 | Special functions |
| ANS | 6 | Toggles between: volume, brightness & contrast |
| Vol+ | 7 | Volume up<br>Adjust settings |
| Vol- | 8 | Volume down<br>Adjust settings |

The full schematic is shown in the following images:


ARDUINO
U2
LCD Screen Control
SRC_O
POWER_O
ANS_O
MENU_O
VOL+_O
VOL-_O
PD0/RXD/PCINT16
PD1/TXD/PCINT17
PD2/INT0/PCINT18
PD3/INT1/OC2B/PCINT19
PD4/T0/XCK/PCINT20
PD5/T1/OC0B/PCINT21
PD6/AIN0/OC0A/PCINT22
PD7/AIN1/PCINT23
AREF
AVCC
PB0/ICP1/CLKO/PCINT0
PB1/OC1A/PCINT1
PB2/SS/OC1B/PCINT2
PB3/MOSI/OC2A/PCINT3
PB4/MISO/PCINT4
PB5/SCK/PCINT5
PB6/TOSC1/XTAL1/PCINT6
PB7/TOSC2/XTAL2/PCINT7
PC0/ADC0/PCINT8
PC1/ADC1/PCINT9
PC2/ADC2/PCINT10
PC3/ADC3/PCINT11
PC4/ADC4/SDA/PCINT12
PC5/ADC5/SCL/PCINT13
PC6/RESET/PCINT14
ATMEGA328P
VCC
RV2
20k
VCC
RV1
20k
Ambient light Detection
0.1
LDR1
LDR
Mirror Control Button
0.1
LDR1
LDR


INTERFACE ARDUINO <-> LCD SCREEN
J1
CONN-H10
VOL-_O
POWER_O
ANS_O
MENU_O
VOL+_O
VOL-_O
SRC_O
R3
R4
R5
R6
R7
R8
1k
D1
D2
D3
D4
D5
D6
1N4148
Q1
Q2
Q3
Q4
Q5
Q6
2N2222
RL1
RL2
RL3
RL4
RL5
RL6
PCJ-124D3MH
SCRN
VOL-_BTN
POWER_BTN
ANS_BTN
MENU_BTN
VOL+_BTN
MENU_BTN
VOL-_BTN

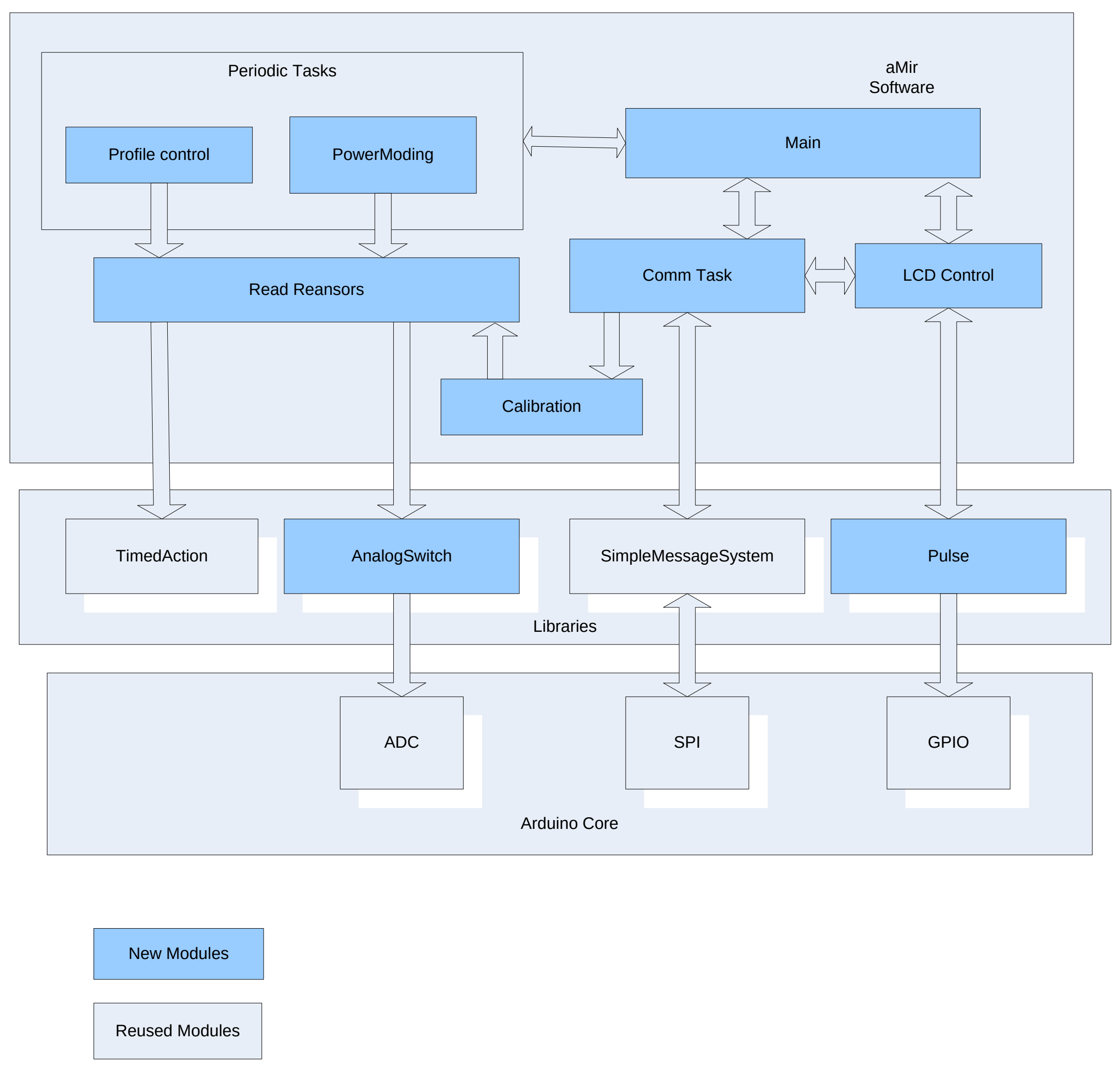
# Appendix 3.

## Arduino Software

The software in compounded by the following modules:

Mirror – Main module
LcdControl – Controls the LCD
Power Moding – Handles the behavior of aMir
ProfileControl – Handles the events
Communication – Interface PC-Arduino

aMir software
(Arduino)

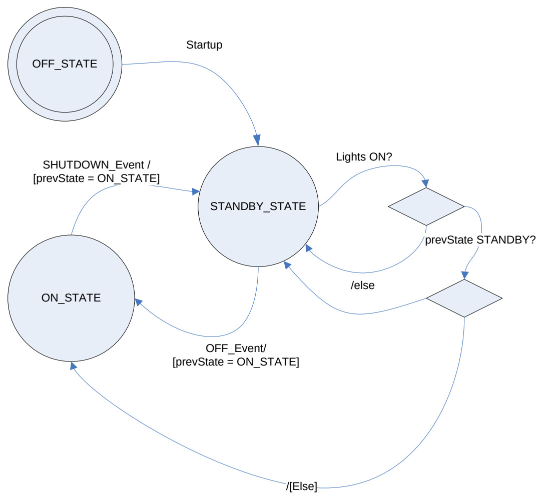


## Power Moding

As any other artifact, aMir has to handle different behaviors in many different scenarios. For aMir we have 3 main states: Off, Standby & On. The next table show how aMir will act upon its different states:

| Current | Ambient Light | LDR | Screen Status | |
|---|---|---|---|---|

| Sate | Detection | button | | |
|---|---|---|---|---|
| Off | Available | Disable | Off | Only occurs during the start up |
| Standby | Available | Limited | Off | Only to turn on the screen |
| On | Available | Available | On | Screen On only when it’s light in the environment |



## Communication PC - Arduino

SimpleMessageSystem[i] is a library for Arduino 0004 and up. It facilitates communication with terminals or message based programs like Pure Data or Max/Msp.
All serial input and output is interpreted as ASCII messages.
- Send, receive and parse lists of characters and integers to and from the Arduino Board.

A ***message*** is a series of words, made from ASCII characters, separated by spaces and terminated by a carriage return (and an optional line feed). Messages are built using the following structure:

word1 (space) word2 (space) word3 (EOM)

EOM = End of message: @ (0x44)

### Arduino code example 1:

// Arduino code

```
    if (messageBuild()) { // Checks to see if the message is complete
      firstChar = messageGetChar()) { // Gets the first word as a character
      if (firstChar = 'r') { // Checking for the character 'r'
        secondChar = messageGetChar() // Gets the next word as a character
        if (firstChar = 'd') // The next character has to be 'd' to continue
          messageSendChar('d'); // Echo what is being read
          for (char i=2;i<14;i++) {
            messageSendInt(digitalRead(i)); // Read pins 2 to 13
          }
          messageEnd(); // Terminate the message being sent
        }
      }
    }
// Arduino code end
```

If the preceding Arduino+SimpleMessageSystem code receives the message:

```
r d CR (CR stands for a carriage return)
```

it will return the value of all the digital pins in a message taking the following structure:

```
d pin2 pin3 pin4 pin5 pin6 pin7 pin8 pin9 pin10 pin11 pin12 pin13 CR
```

## aMir Lingo

A series of messages was generated to properly synchronize the PC application that will show the image in the screen.

### Messages PC -> Arduino (>>)

| Command | Arg 1 | Arg 2 | | Action |
|---|---|---|---|---|
| 'b'<br><br>Activates a button to control the LCD | ‘0’ | | | Activate Source Button |
| | ‘1’ | | | Activate Power Button |
| | ‘2’ | | | Activate Menu Button |
| | ‘3’ | | | Activate ANS Button |

| | | | | |
|---|---|---|---|---|
| | ‘4’ | 1-255: times that the button will be activated | | Activate Vol+ Button |
| | ‘5’ | 1-255: times that the button will be activated | | Activate Vol- Button |
| 'c' Calibration | | | | Calibrate the LDR sensors |
| ‘l’<br>Run the LCD setup function | | | | Calls a function to adjust the display |
| ‘s’<br><br>Status confirmation | ‘0’ | | | Start up |
| | ‘1’ | | | Stand-by mode |
| | ‘2’ | | | System is running |

Messages Arduino -> PC (<<)

| Command | Arg 1 | Arg 2 | | Action |
|---|---|---|---|---|
| 'S'<br>Send the current status (this requires confirmation) | ‘0’ | | | Start up |
| | ‘1’ | | | Stand-by mode |
| | ‘2’ | | | System is running |
| ‘N’<br>Request next profile | | | | Request the next available profile |

## Examples:

1. Assuming that the LCD is off, turn it off and select the next A/V source.

```
>>  b 1@   // Power button
>>  b 0@   // source button
```

2. Assuming that the LCD is on, set the brightness to the minimum.

```
>>  b 3@    // ANS button (Volume)
>>  b 3@    // ANS button (Brightness)
>>  b 5 25@ // Press Vol- 25 times (maybe that is enough)
```

3. Assuming that the system is starting up, the arduino shall inform to the pc about its status.

```
<<  s 0@    // Start up
>>  S 0@    // Confirmation
```

## Source Code:

### *Mirror.pde*

```
#include <TimedAction.h>
#include "ioConfig.h"
#include "mirror.h"
#include "lcdControl.h"
#include "communication.h"

// void systemRunningLED();

// set pin numbers:
 const int ledPin =  13;      // the number of the LED pin

 // Variables will change:
 int ledState = LOW;             // ledState used to set the LED
 long previousMillis = 0;        // will store last time LED was updated

 // the follow variables is a long because the time, measured in miliseconds,
 // will quickly become a bigger number than can be stored in an int.
 long interval = 200;           // interval at which to blink (milliseconds)

TimedAction readSensorsAction = TimedAction(25, readSensors);
TimedAction lcdPollingAction = TimedAction(150, lcdPolling);

 void setup()
 {
   setupComm();
   calibrateLdrButton();
   setupLcdControl();
 }


 void loop()
 {
   readSensorsAction.check();
   lcdPollingAction.check();
   PowerModingTask();
   //Testing
   commTask();
```

```
  delay(10);
}


void systemRunningLED()
{
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
}

void readSensors()
{
   ButtonsTask();
   systemRunningLED();
}

// This function is called peridocalli to detect if
// any button was pressed
void lcdPolling()
{
    lcdTask();
}
```

***Mirror.h***

```
#include "WProgram.h"

// Set it to true to see debugging information
#define _DEBUG_MODE_IS false
```

***ioConfig.h***

```
#ifndef IOCONFIG_H
  #define IOCONFIG_H

  // analog
  #define ldrButtonPort 1 // buttton for profile switchinf
  #define ldrSensorPort       2 // detect if there is light or not

  // digital
  #define srcControlPin      2 // settings
  #define powerControlPin    3
  #define menuControlPin     4 // settings
  #define ansControlPin      5
  #define volPlusControlPin  6
  #define volMinusControlPin 7 // settings
```

```
#endif
```

***Communication.pde***

```
#include "lcdControl.h"
#include <SimpleMessageSystem.h>

void commTask()
{

  if (messageBuild() > 0)
  { // Checks to see if the message is complete and erases any previous messages
    switch (messageGetChar())
    { // Gets the first word as a character
    case 'b': // Activate the a button from the LCD display
      activateButton(); // Call the readpins function
      break; // Break from the switch
    case 'c': // calibrate
      autoCalibrate();
      break;
    case 'l': // calibrate
      setupLcdControl();
      break;
    case 's': // calibrate
      notificationOfStatus(messageGetInt());
      break;
    }

  }

}

//Setup the communication
void setupComm()
{
  Serial.begin(9600);
}

// Decodes a message to activate the proper button
void activateButton() {
  int data;
  switch (messageGetInt()) { // Gets the next word as a integer

    case POWER_BTN :
       pressOnOffButton();
    break;  // Break from the switch
    case MENU_BTN :
       pressMenuButton();
    break;
    case ANS_BTN :
       pressAnsButton();
    break;
    case SRC_BTN :
       pressSrcButton();
  break;
  case VOL_PLUS_BTN :
    data = messageGetInt();
    pressVolPlusButton(data);
  break;
  case VOL_MINUS_BTN :
    data = messageGetInt();
    pressVolMinusButton(data);
  break;
  default :
    //nothing
  break;
 }
}

// this function executes the required calibration
void autoCalibrate() {
  calibrateLdrButton();
 }
```

***Communication.h***

```
#include "WProgram.h"

void commTask();
void activateButton();
```

lcdControl.h

```
#include <Pulse.h>
#include "lcdControl.h"
#include "ioConfig.h"

#define pulseWidth      50 //ms

// Set the GPIO used to control the LCD
Pulse powerPulseControl(powerControlPin, 3000);
Pulse menuPulseControl(menuControlPin, pulseWidth);
Pulse ansPulseControl(ansControlPin, pulseWidth);
Pulse srcPulseControl(srcControlPin, pulseWidth);
Pulse volPlusPulseControl(volPlusControlPin, pulseWidth);
Pulse volMinusPulseControl(volMinusControlPin, pulseWidth);

void setupLcdControl()
{
  // during the set-up turn on the screen
  pressOnOffButton();
   // Its called twice to set the screen to S-Video
  pressSrcBtutton();
  pressSrcBtutton();
}

void lcdTask()
{
  powerPulseControl.polling();
  menuPulseControl.polling();
  ansPulseControl.polling();
  srcPulseControl.polling();
  volPlusPulseControl.polling();
  volMinusPulseControl.polling();
}

void pressOnOffButton()
{
 powerPulseControl.trigger();
}
void pressMenuButton()
{
 menuPulseControl.trigger();
}

void pressAnsButton()
{
 ansPulseControl.trigger();
}
void pressSrcButton()
{
 srcPulseControl.trigger();
}

void pressVolPlusButton(int num)
{
 volPlusPulseControl.trigger(num);
}
void pressVolMinusButton(int num)
{
 volMinusPulseControl.trigger(num);
}
```

***lcdControl.h***

```
#ifndef  LCD_CONTROL_H
  #define  LCD_CONTROL_H
#include "WProgram.h"

//enumerate the buttons
enum lcdButtons
{
  SRC_BTN,
  POWER_BTN,
  MENU_BTN,
  ANS_BTN,
  VOL_PLUS_BTN,
  VOL_MINUS_BTN,
  MAX_NUM_BTN,
};

void setupLcdControl();
void pressOnOffButton();
void pressInputButton();
void pressVolMinusButton(int num);

#endif
```

***powerModing.pde***

```
#include "AnalogSwitch.h"
#include "powerModing.h"

#define OFF_DELAY   10000

static int currentPowerState = OFF_STATE;
static int newPowerState = STANDBY_STATE;
static int prevPowerState = OFF_STATE;
static unsigned long startTime;
static unsigned long curTime; // this is global variable
unsigned int lightSensorLevel;
boolean goToPrevStateFlag = false;
boolean changeFlag = true;

TimedAction refreshPowerStateAction = TimedAction(750, refreshPowerStatePolling);
AnalogSwitch lightSensor(ldrSensorPort, lightSensorLevel, 0);

 void PowerModingTask()
 {
  switch(currentPowerState)
  {
    case OFF_STATE:
      //Call initialization or any other setup
      offState();
    break;
    case STANDBY_STATE:
      //Call initialization
      standbyState();
    break;
    case ON_STATE:
      //Call
      standbyState();
    break;
  }
  //Update the the current state
  if(newPowerState != currentPowerState)
  {
    currentPowerState = newPowerState;
    //SendNotificationOfStatus(currentPowerState);
  }
  refreshPowerStateAction.check();
 }

 void offState()
 {
  currentPowerState = STANDBY_STATE;
  delay(500);
 }
```

```
void standbyState()
{
   ExecuteDuringStandby();
}

void onState()
{
   ExecuteDuringOn();
}

void ExecuteDuringStandby()
{
  checkLightLevel();
}

void ExecuteDuringOn()
{
  //Check the ambient light level
  checkLightLevel();

}

int CurrentPowerState()
{
   return currentPowerState;
}

void notificationOfStatus(int data)
{
}

 void refreshPowerStatePolling()
{
  SendNotificationOfStatus(newPowerState);
}

void SendNotificationOfStatus(int data)
{
    messageSendChar('S');
    messageSendInt(data);
    messageEnd();
}

void checkLightLevel()
{
    if(lightSensor.Read() && changeFlag)
    {
        #if _DEBUG_MODE_IS
        Serial.println("there is NO light");
        #endif
        prevPowerState = currentPowerState;
        newPowerState = STANDBY_STATE; //Turns off the screen
        goToPrevStateFlag = true;
        changeFlag = false;
    }else
    if(goToPrevStateFlag && !lightSensor.Read())
    {
      #if _DEBUG_MODE_IS
      Serial.println("there is  light");
      #endif
      newPowerState = prevPowerState;
      goToPrevStateFlag = false;
      changeFlag = true;
      SendNotificationOfStatus(newPowerState);
    }
 }
```

***powerModing.h***

```
#ifndef  POWER_MODING_H
  #define  POWER_MODING_H
#include "WProgram.h"
```

***enum PowerState***

```
{
  OFF_STATE,
  STANDBY_STATE,
  ON_STATE
};

#endif
```

***profileControl.pde***

```
#include "AnalogSwitch.h"
#include "ioConfig.h"

#define LDR_WAIT_TIME 500 //milliseconds
#define LDR_NEXT_TIME 300 //milliseconds
#define TIME_FOR_INACTIVITY  15000
#define TRESHOLD_WIDTH  10

unsigned int ldrTimeStamp;
unsigned int ldrElapsedTime;
unsigned int lastActionTimeStamp;
unsigned int levelLdrButton;
AnalogSwitch selectBtn(ldrButtonPort, levelLdrButton, TRESHOLD_WIDTH);
boolean selectBtnActive = false;

void ButtonsTask(void)
{
 if(selectBtnActive)
 {
   if(selectBtn.Read())
   {
     //nothing
   }
```

```
    else //Button released
    {
      #if _DEBUG_MODE_IS
      Serial.println("end");
      #endif
      selectBtnActive = false;
      ldrElapsedTime = millis() - ldrTimeStamp;
      performAction();
    }
  }
  else
  {
    if(selectBtn.Read())
    {
      ldrTimeStamp = millis();
      selectBtnActive = true;
      #if _DEBUG_MODE_IS
      Serial.println("Start");
      #endif
    }
  }
  // Go to stand by for inactivity
  if(CurrentPowerState() == ON_STATE)
   {
     if((lastActionTimeStamp > (millis() + TIME_FOR_INACTIVITY)))
     {
       ScreenOff();
     }
   }
}


void performAction(void)
{
  long int currentTime = millis();
  //Change profile
   #if _DEBUG_MODE_IS
   Serial.println("Action");
   #endif
   if((CurrentPowerState() == STANDBY_STATE))
   {
     if(ldrElapsedTime > 1500)
     {
       //Deactivate the screen
       ScreenOn();
     }
   }
   else if(CurrentPowerState() == ON_STATE)
   {
     if((ldrElapsedTime > 50) && (ldrElapsedTime < (1500)))
     {
       ReqNextProfile();
     }
     else if(ldrElapsedTime >  1500)
     {
       ScreenOff();
     }
   }
   #if _DEBUG_MODE_IS
   Serial.println(ldrElapsedTime);
   #endif
   lastActionTimeStamp = millis();
}

void ScreenOff(void)
{
   #if _DEBUG_MODE_IS
   Serial.println("turn Off screen");
   #endif
   newPowerState = STANDBY_STATE;
}

void ScreenOn(void)
{
   #if _DEBUG_MODE_IS
   Serial.println("turn On screen");
   #endif
   newPowerState = ON_STATE;

}

void ReqNextProfile(void)
{
     messageSendChar('N');
     messageEnd();
    #if _DEBUG_MODE_IS
    Serial.println("Next profile");
    #endif

}

void calibrateLdrButton(void)
{
   long int ldrButtonValueSum = 0;
   long int ldrSensorValueSum = 0;
   //long int ldrSensorValueSum = 0;
   //long int ldrSensorValueSum = 0;
   long int timmer = millis() + 3000;
   int i = 0;

   // calibrate during the first seconds
   while (millis() < timmer) {
    ldrButtonValueSum += analogRead(ldrButtonPort);
    ldrSensorValueSum += analogRead(ldrSensorPort);
    i++;
    delay(100);
   }
   // set the values
   selectBtn.SetTriggerLevel((ldrButtonValueSum / i) + 2*TRESHOLD_WIDTH);
   lightSensor.SetTriggerLevel(ldrSensorValueSum / i *2 );
```

```
    #if _DEBUG_MODE_IS
    Serial.print("LdrButton is set to: ");
    Serial.println(ldrButtonValueSum / i);
    //Serial.Println("Ldr is set to: %d", levelLdr);
    #endif
    selectBtnActive = false;
}
```

***Pulse.cpp***

```
#include "WProgram.h"
#include "Pulse.h"

/*
 * two-wire constructor.
 * Sets which wires should control the motor.
 */
Pulse::Pulse(int pin, unsigned long ms)
{
  this->init_time = millis();      // time stamp in ms since the pin will be active
  this->inverted = false;          // this is an inverted pulse (LOW active)
  this->active = false;            // current state of the pin
  this->width_ms = ms;             // amount of time that the pin will be active
  this->n_pulses = 0;              // number of pulses left
  this->wait = false;              // wait flag
  this->wait_time = 10;            // wait this time to start the next pulse

  // Arduino pin used:
  this->pulse_pin = pin;

  // setup the pins on the microcontroller:
  pinMode(this->pulse_pin, OUTPUT);
  digitalWrite(pin, LOW);

}

/*
 * two-wire constructor.
 * Sets which wires should control the motor.
 */
Pulse::Pulse(int pin, unsigned long ms, boolean inverted)
{
  this->init_time = millis();      // time stamp in ms since the pin will be active
  this->inverted = inverted;       // this is an inverted pulse (LOW active)
  this->active = false;            // current state of the pin
  this->width_ms = ms;             // amount of time that the pin will be active
  this->n_pulses = 0;              // number of pulses left
  this->wait = false;              // wait flag
  this->wait_time = 10;            // wait this time to start the next pulse

  // Arduino pins for the motor control connection:
  this->pulse_pin = pin;

  // setup the pins on the microcontroller:
```

```
 pinMode(this->pulse_pin, OUTPUT);
 digitalWrite(pin, HIGH);

}

// Allow to set the pulse width
void Pulse::setWidth(unsigned long ms)
{
 this->width_ms = ms;
}

/*
 Generates a single pulse
 */
void Pulse::trigger(void)
{
 if (!(this->active)) {
    this->init_time = millis();
        this->active = true;
        startPulse();
        this->n_pulses = 1;
  }
}

/*
 Generates a train pulse.
 n  Number of pulses
*/
void Pulse::trigger(int n)
{
 if (!(this->active)) {
    startPulse();
    this->active = true;
        this->n_pulses = n;
  }
}

/* Ininitates the pulse generation*/
void Pulse::startPulse()
{

  this->init_time = millis();
  if (this->inverted) {
        // get the timeStamp of when you stepped:
    digitalWrite(this->pulse_pin, LOW);
  }
  else{
        digitalWrite(this->pulse_pin, HIGH);
  }
  Serial.print("pressed pin ");
  Serial.println(this->pulse_pin);
}

/*
 This functions MUST be called periodically
 */
void Pulse::polling(void)
{

  if (this->active) {
		// does it have to wait to triggger the next pulse or not?
		if(!this->wait)
		{
		  // move only if the appropriate delay has passed:
		  if (millis() - this->init_time >= this->width_ms){
				// get the timeStamp of when you stepped:
				digitalWrite(this->pulse_pin, !(digitalRead(this->pulse_pin)));
				this->wait = true;
					this->n_pulses --;
			      }
		}
		else { //if it does have to wait
		  if (millis() - this->init_time >= this->wait_time){
				// get the timeStamp of when you stepped:
				startPulse();
				this->wait = false;
			      }
		}
		if(this->n_pulses == 0)
		{
			this->active = false;
		}
  }
}

void Pulse::toggle(void)
{
  this->inverted = !(this->inverted);
}
/*
  version() returns the version of the library:
*/
int Pulse::version(void)
{
  return 1;
}
```

Pulse.h

```
#ifndef Pulse_h
#define Pulse_h

// library interface description
class Pulse {
  public:
    // constructors:
    Pulse(int pin, unsigned long ms);
    Pulse(int pin, unsigned long ms, boolean inverted);
```

```
   // toggle method:
            void toggle(void);
   void polling(void);
            void setWidth(unsigned long ms);
            void trigger(void);
            void trigger(int n);

   int version(void);

  private:
            void startPulse(void);

   boolean inverted;       // high pulse or low pulse
   boolean active;         // the pulse is active
   boolean wait;           // the pulse is in wait mode
   unsigned long init_time;    // delay between steps, in ms, based on speed
   unsigned long width_ms;     // ms that the puls will last
   unsigned long wait_time;    // delay between steps, in ms, based on speed
   int n_pulses;
   int pulse_pin;
};

#endif
```

***AnalogButton.cpp***

```
#include "WProgram.h"
#include "AnalogSwitch.h"

/*
 * two-wire constructor.
 * Sets which wires should control the motor.
 */
AnalogSwitch::AnalogSwitch(int aPin, int triggerLevel, int threshold)
{
  // Arduino pin used:
  this->aPin = aPin;

  this->triggerLevel = triggerLevel;
  this->threshold = threshold;
  setActive(false);
  this->inverted = false;

}

boolean AnalogSwitch::Read(void)
{
   int data = analogRead(this->aPin);
   if(data > (this->triggerLevel + this->threshold))
   {
     setActive(true);
   }
   else if(data < (this->triggerLevel - this->threshold))
   {
     setActive(false);
   }
   #if 1
   if(this->aPin ==0) {
   Serial.print("[");
   Serial.print(this->triggerLevel - this->threshold);
   Serial.print(", ");
   Serial.print(this->triggerLevel + this->threshold );
   Serial.println("] ");
   Serial.println(data);}
   #endif
   return this->state;
}

void AnalogSwitch::InvertedMode(boolean inverted)
{
   this->inverted = state;
}

void AnalogSwitch::setActive(boolean state)
{
   if(this->inverted)
   {
     this->state = ~ state;

   }
   else
   {
     this->state = state;
   }
}

void AnalogSwitch::SetTriggerLevel(int level)
{
   this->triggerLevel = level;
   setActive(false);
}
```

***AnlaogSwitch.h***

```
#ifndef AnalogSwitch_h
#define AnalogSwitch_h

// library interface description
class AnalogSwitch {
 public:
  // constructors:
  AnalogSwitch(int aPin, int triggerLevel, int threshold);
  // methods
        boolean Read(void);
        void InvertedMode(boolean inverted);
        void SetTriggerLevel(int level);
 private:
  int aPin;              // analog pin number
  int triggerLevel;       // current level
  int threshold;          // threshold size(+/-)
        boolean inverted;       // used to handle the switch as active in low
        boolean state;          // activated/deactivated flag
        void setActive(boolean state); // activate/deactivate
};

#endif
```

[i] http://tof.danslchamp.org/
Contact: tof [at] danslchamp [dot] org