The Android APK • SEGA Gaming on your ODROID • Linux Gaming

# ODROID Magazine

Year One
Issue #9
Sep 2014

## BUILD YOUR OWN WALL-E

THE LOVABLE PIXAR ROBOT COMES TO LIFE WITH AN ODROID-U3

- BASH BASICS
- FREEDOMOTIC
- WEATHER FORECAST
- IO-NODE U3 CLUSTER
- ODROID-SHOW

## 3D PRINT AN ODROID-POWERED GARDENING SYSTEM


PLUS:

## A FUTURISTIC PORTABLE DIY LAPTOP

# What we stand for.

We strive to symbolize the edge technology,
future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with
developers around the world.

For that, you can always count on having the quality
and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish
everything you can dream of.

HARDKERNEL

---

Pollin ELECTRONIC

**We are now shipping the ODROID U3 devices to EU countries! Come and visit our online store to shop!**

**Address: Max-Pollin-Straße 1**
**85104 Pförring Germany**

**Telephone & Fax**
**phone : +49 (0) 8403 / 920-920**
**email : service@pollin.de**

**Our ODROID products can be found at**
http://bit.ly/1tXPXwe

# EDITORIAL

With the introduction of the ODROID-W and ODROID Weather Board, there have been several projects posted recently on the ODROID forums involving home automation, ambient lighting, and cool robotics. This month, we feature several of those projects, including predicting whether to go fishing this weekend, building a custom laptop case, taking care of your garden remotely, and building a faithful reproduction of everyone's favorite robot, Wall-E!

Hardkernel will be demonstrating the new XU3 at ARM Techcon on October 1st - 3rd, 2014 in San Jose, California. Stop by the booth if you'd like to chat with some of the Hardkernel and ODROID Magazine team members. Tickets for the exposition floor are currently $59 at www.armtechcon.com.

The recently released octa-core XU3 already has several modern operating systems available, including both Android and Ubuntu. The ARCHLinux group has already posted instructions for compiling ARCHLinux for ARM (ALARM) for the XU3 at http://bit.ly/1tS2xNs. Hardkernel offers Android 4.4 for download at http://bit.ly/1qMA60q, ODROID forum user @voodik published CyanogenMod 11 at http://bit.ly/1qMA60q, and Ubuntu 14.04 is available at http://bit.ly/1s06GZW.

If you haven't ordered one already, the XU3 is the fastest computer that Hardkernel has ever made, since it's able to use all eight cores at the same time, improving upon the original XU's design of cluster switching between the high-efficiency A7 and performance A15 cores. It's also fully compatible with USB 3.0 and eMMC 5.0, and offers the latest Mali T-628 MP6 GPU with OpenGLES 3.0 and OpenCL 1.1. It's available from the Hardkernel store at http://bit.ly/YGEnc2.

ODROID Magazine, published monthly at http://magazine.odroid.com, is your source for all things ODROIDian.
Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815
Makers of the ODROID family of quad-core development boards and the world's first ARM big.LITTLE architecture based single board computer.
Join the ODROID community with members from over 135 countries, at http://forum.odroid.com, and explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

HARDKERNEL

# ODROID Magazine

### Rob Roy, Chief Editor

I'm a computer programmer living and working in San Francisco, CA, designing and building web applications for local clients on my network cluster of ODROIDs. My primary languages are jQuery, Angular JS and HTML5/CSS3. I also develop pre-built operating systems, custom kernels and optimized applications for the ODROID platform based on Hardkernel's official releases, for which I have won several Monthly Forum Awards. I use my ODROIDs for a variety of purposes, including media center, web server, application development, workstation, and gaming console. You can check out my 100GB collection of ODROID software, prebuilt kernels and OS images at `http://bit.ly/1fsaXQs`.

### Bo Lechnowsky, Editor

I am President of Respectech, Inc., a technology consultancy in Ukiah, CA, USA that I founded in 2001. From my background in electronics and computer programming, I manage a team of technologists, plus develop custom solutions for companies ranging from small businesses to worldwide corporations. ODROIDs are one of the weapons in my arsenal for tackling these projects. My favorite development languages are Rebol and Red, both of which run fabulously on ARM-based systems like the ODROID-U3. Regarding hobbies, if you need some, I'd be happy to give you some of mine as I have too many. That would help me to have more time to spend with my wonderful wife of 23 years and my four beautiful children.

### Bruno Doiche, Art Editor

Secured his computing necromantic skills after bringing a fiber optics switch back to life, getting his Macintosh back from death, getting a PS3 back from death, getting his fiancee T400 back from death (that was a old style dd data transplant), and managing how to handle the cold innards of his steady job data center.

### Nicole Scott, Art Editor

I'm a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media directing, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from Analytics and Adwords to video editing and DVD authoring. I own an ODROID-U3 which I use to run a sandbox web server, live in the California Bay Area, and enjoy hiking, camping and playing music. Visit my web page at http://www.nicolecscott.com.

### Manuel Adamuz, Spanish Editor

I am 31 years old and live in Seville, Spain, and was born in Granada. I am married to a wonderful woman and have a child. A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. I am passionate about computer science, especially microcomputers such as the ODROID and Raspberry Pi. I love experimenting with these computers. My wife says I'm crazy because I just think of ODROIDs! My other great hobby is mountain biking, and I occasionally participate in semi-professional competitions.

# INDEX

# PLAYING SEGA GAMES IN HD 1080P

## A BLAST FROM THE PAST

**By Jeremy "Cartridge" Kenney**

An ODROID is all you need to get your favorite Sega Games running again! You will optionally need a Sega Genesis controller adapter to output your controller to USB so that the ODROID can recognize it and, of course, be able to plug it in to the USB port. You can then grab a copy of DGEN-SDL for the ODROID, compiled by myself at HardKernel Forums, at `http://bit.ly/1pgoyO8` to get started playing Sega games.

The unique features of DGEN make it a wonderful application to use, and includes a nice range of options to customize your emulation. DGEN can also make use of GameGenie codes, so if you still own a GameGenie manual or even have written codes in the last pages of your game manual under the "Notes" pages, you can use them with DGEN.

Fullscreen modes of NTSC and PAL works flawlessly for both North America and Europe, as well as Japan NTSC, in order to output video on any TV you may have. OpenGL is implemented, but is not compiled in, because ODROID computers make use of OpenGL-ES which does not easily translate to OpenGL since the ES version is a subset of OpenGL.

There are options for Screenshotting and video recording, however the record option does not output in AVI or MPG, but instead outputs video in proprietary format. This means you won't be able to view the recorded video with a media player, only through DGEN. It works nicely and has a built in menu to fast-forward, reverse and even pause the playback.

16 bit sound is used to stay true to Sega's original 16 Bit sound and graphics, but you can change the sampling from 8000hz to 48000hz to give the 16 bit sound a better quality. There are also options to change the original color output from 8, 15, 16, 24 and 32 bpp modes giving you a good range of bits to choose from. Support for compressed and zipped files, including rar, is enabled.

Scale2X and HQX is supported for desktop resolutions only, and is compiled in for your convenience because they work flawlessly on CRT resolutions. Finally, the M68K debugger is implemented for developers to debug their games and apps running on the 68K chip.

I've included a text file in the original post thread to associate button mapping to customize your controller mapping. DGEN should already detect the controller and auto-map the buttons, but for unsupported controllers, refer to the Text File Button Mapping file. Although it's incomplete, we are rounding up every controller that we can get in order to determine if they will be auto-mapped correctly. Most 2003 controllers, gamepads and joysticks are mapped perfectly without additional configuration, including the previously mentioned Sega Genesis Controller.

To install DGEN, unzip the package and type the following in Terminal, replacing the .deb file with the package name:

```
$ sudo dpkg -i NameOfYourDebian-PackageHere.deb
```

If you have a controller, it's best to plug

**Sega Genesis, the system that gave us Sonic!**


Sega game up and running

it in before running the app. To launch DGEN, type the following, replacing the filename with the selected ROM:

```
dgen -f romname.bin
```

## Options

Using the option -f permits fullscreen, and -G outputs a selected resolution, for example:

```
dgen -G 1279x719
```

This will result in a windowed desktop resolution of 1024x768. HQX and Scale2x may be enabled by pressing F6 and F5 for "Crappy TV" filters, giving the ability to have a low-resolution video output like in the old days!

-R will help you run the game of the region of your choice. Possible region codes are "E" for Europe, "U" for USA/Canada and "J" for Japan:

```
$ dgen -f -R E romname.bin
```

-D will play back your recorded video:

```
$ dgen -f -d demoname romname.bin
```

Now your ODROID is ready to play Sega Genesis games on your HDTV, old CRT TV or any other kind of monitor. If you don't have an old TV to play with and your HDTV is too high definition for your liking, enable the CrapTV filters to make it look authentic. Have fun playing Sega with your ODROID!

# LINUX GAMING

## PLAYING NATIVE GAMES ON THE ODROID - PART I

by Tobias Schaaf

In the last few articles, I presented an overview of the different emulators available on the GameStation Turbo image, which supports thousands of games by emulating different console systems such as SNES or PS1. However, because GameStation Turbo uses Linux as its underlying operating system, I'd like to now take a closer look into some of the games that exists for Linux that run natively, without an emulator. Note that all of the pictures in this article were taken with an ODROID.


### DOOM 3 – Horror SciFi First Person Shooter (18+)

For those who haven't noticed yet, there is an awesome first person shooter available for the ODROID. @AreaScout ported DOOM 3 to the ODROID platform, and was able to get it running natively with OpenGL ES. In Doom 3, you play a soldier who recently arrived at the ars base, which is a huge research facility. Many scientists work on different projects at the base, including teleportation. By using the teleportation technique, the scientists discover a new dimension, but something goes wrong! Demons are appearing and either killing humans or transforming them into monsters.

Hell breaks loose, and you have to fight your way throughout the levels of Doom 3. The game comes with a lot of scripted events that are supposed to scare and frighten the player with shocking effects. Together with the dark atmosphere of the game, this is really going to freak you out every now and then!

@AreaScout implemented a new shader for the game to fix an issue with the gamma and brightness. With this setting, you can lighten up the surrounding of the game, which makes it easier to find objects in the game and see corners where an enemy might lurk into the dark, ready to scare and attack you. Doom 3 is definitely a must have for all horror and first person shooters.

### Homeworld - Space Real Time Strategy

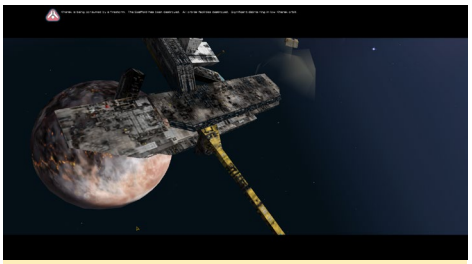
Another game that can be launched natively on the ODROID is an awesome real time strategy (RTS) game called Homeworld. In Homeworld, you are part of a group of people that

**Resource collector and the Mothership (Homeworld)**

discovered plans of an old ship while on an expedition, along with a Hyperspace Core which allows you to travel very fast through space. They discover the origin of their own species on a world called HIIGARA (home), and they decide to build a vast space ship called the "mothership" to travel to that world.

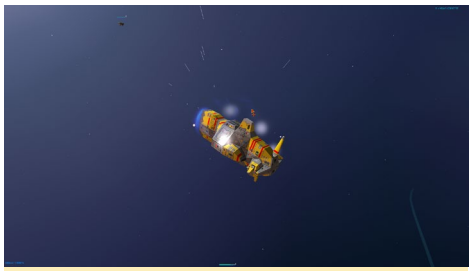
However, soon after the mothership is finished and goes on its first test hyperspace jump, something goes wrong. The ship you are supposed to meet was destroyed by raiders, and when you come back to your planet, you find it devastated along with the space station. So, you have to build up a fleet to defend yourself, revenge your people and find your way to HIIGARA. Soon you meet new allies, enemies and other species.

Homeworld is a very nice game, with beautiful graphics, and a deep soundtrack including the well known Adagio for Strings from Samuel Barber, and featuring the famous Vienna Boys Choir. The game has a lot of details in the different spaceship models and effects, backgrounds and planets.



**Destroyed planet and shipyard (Homeworld)**

**Starting to build up a fleet to defend yourself (Homeworld)**



**Detailed view of an enemy ship under fire (Homeworld)**

Homeworld's story is very thrilling since you have to research new technology and ships, plan your moves ahead, household with the rare resources you have, and protect your fleet while smashing through the enemy lines. If you like RTS games and like the space genre as well, this is definitely a must have and will keep you busy for hours and hours.


## EDuke 32 – First person shooter

**"Come get some!"** -- Duke Nukem

Duke Nukem is a very famous character in gaming history. He's the tough guy, that goes and saves the world, and all the girls, from an evil alien invasion. He will always be remembered for quotes like "What are you waiting for.... Christmas?" With the newly ported EDuke32, Duke Nukem comes back to life, and this time in HD quality and with real 3D polygon models. When the game first came out on PC, I played the DOS version. I still wonder how I was able to play the game without a mouse, resorting to looking up and down by pressing keys on the keyboard while moving left and right, all while trying not to get shot by an enemy.

Originally, the game offered a 3D world with 2D sprites as characters and objects, which was very impressive back then. It still looks quite nice with trilinear filtering.

Interestingly, mirror algorithms were not easy to implement in the age of DOS, so the programmers created an identical world and doubled all the characters and movements. So when you looked into a mirror you did not actually see a reflection, but another Duke Nukem behind a window, mirroring your movements!

**Duke Nukem in front of a mirror "Damn... I'm looking good!"**

The screenshot shows how the virtual world was filled with sprites back then, since 3D cards were not very common, and everything was done by the CPU. Luckily, those days are over, and we now have a new version of Duke Nukem in real 3D and with high resolution textures. If you haven't played Duke Nukem 3D before, the remake is well worth it, offering a lot of action, nice gameplay and a little bit of erotica here and there.

Don't forget the all famous quotes from Duke Nukem himself which made him the most badass gaming character of all time. But remember, the game was made for mature audience. Now, there's only one thing left to do: "Hail to the king, baby!"

# BUILDING AN ALL-IN-ONE DIY LAPTOP

## TAKE YOUR U3 ANYWHERE

**by Daniele S.**

**The finished laptop case, with speakers installed at the bottom of the clear plexiglass front, a handy I/O shield for quick hardware prototyping, and a beautiful 10.1" HD monitor**

Building my own ODROID laptop all started when my ancient single-core laptop broke. I searched the Internet for a replacement and discovered the fantastic ODROID-U3, which is powerful enough to be used in place of a normal computer. I decided to build a laptop from an ODROID-U3 and make a custom case out of plexiglass.

For the screen, I purchased a nice 10.1-inch monitor, as well as a couple of accessories like the I/O shield and wifi, which is all I needed to start building my "all-in-one" laptop! However, I had a lot of peripherals without a container to accommodate them. Although I had seen several PCs made of wood, I opted to use plexiglass, which is fairly easy to work with.

First, I used a pen and paper to outline the project. My design was not complicated, so I measured and cut it directly on the plexiglass. The characteristic of the transparent plexiglass helped me understand the measurements. For example, to take measurements of the holes on the ODROID, I simply put the card on the sheet of plexiglass. Once inverted, I marked off where to drill the four points.

For the plexiglass, I recommend using at least a 5mm thickness in order to strengthen the floor and structure. The top panels are OK to use 4mm. Owning DIY machinery is still very expensive (unless I borrow from a neighbor), so I used a lot of tools that are easily found in the home or garage.

* **Hand jig saw**
* **Screwdriver**
* **Hair dryer**
* **Nail file**
* **Measuring tape**
* **Marker**

For those not familiar with plexiglass, it has a tendency to fuse together at higher temperatures. The first time I made a cut on plexiglass, I went very fast and I could see that behind the blade, the plexiglass material melted and cooled back together. By the time I had finished cutting, it had repaired itself as if it had not been cut!

I learned to keep the speed of the saw blade very low, and apply a continuous on/off so as to slowly move the blade. For straight cuts, I laid a straight edge to the desired length and inserted a nice wide blade to help with staying straight. A small blade helps with creating circles for cables, connectors and switches.

An important thing to keep handy is a nice glass of water - not for drinking, but for the blade of the saw. Every 2-3 cm of cutting, I removed the blade and let it cool in the glass of water, which also lubricated it. When drilling, keep everything nice and wet with the water, and put a wooden piece below the plexiglass. The wood keeps the plexiglass from breaking when the drill pierces the bottom side.

I then spent some time curving the edges of the case, using a hair dryer to blow hot air over the plexiglass. After a while, it became soft to where I could apply pressure (it takes very little). On one side of the panel, when it began bending, I moved the hair dryer and put it back over slowly. It takes some finesse to do this: If the plexiglass is too cold, it can crack -- or melt if it's too hot.

Once the piece was bent to the proper position, I used a cool fan to lock the plexiglass into its shape. It also helped to use wood panels anchored with clamps so as to obtain a line to follow while bending. I recommend practicing on scraps of plexiglass. Once you get the hang of it, it's very fun to build a custom laptop case this way!

To affix the panels to each other, I used some self-tapping screws salvaged from an old PC and pre-drilled the screw holes. Once it was finished and fit together nicely, I disassembled everything and moved on to painting. I applied a coating to the inner part of the back piece of the panel, giving a nice mirror effect. By writing on the plexiglass panel with tape, you can remove it once the paint is dry so that the writing is transparent, giving a nice lighting effect at night.

Since I installed the board with the chip facing down, I put two pieces of plastic near the two LEDs to reflect the light. It isn't the most practical solution in the world but it certainly is simple and functional. Finally, I equipped it with two speakers retrieved from my old laptop for playing music.

Although the laptop seems all nice and finished in the pictures, the project is not yet complete, as I still need to install buttons on the front for switching power on and off in order to have a single external 12V power supply. I am also planning to add a LI-ION battery pack, so that it becomes a truly portable laptop.

I worked on the project for a long time, and opted for function over aesthetics. However, nothing should stop you from doing something more beautiful. Now you can pick up your household tools and make a nice custom case for your own ODROID!

# BASH BASICS

## SHEBANGS AND SHEBANGS

**by Tynan Overstreet**

Bash is a very useful tool for automating administrative tasks on your Odroid. A script can be something quick and dirty just to get a job done; or it can run each time your computer boots up.

### Building a bash script

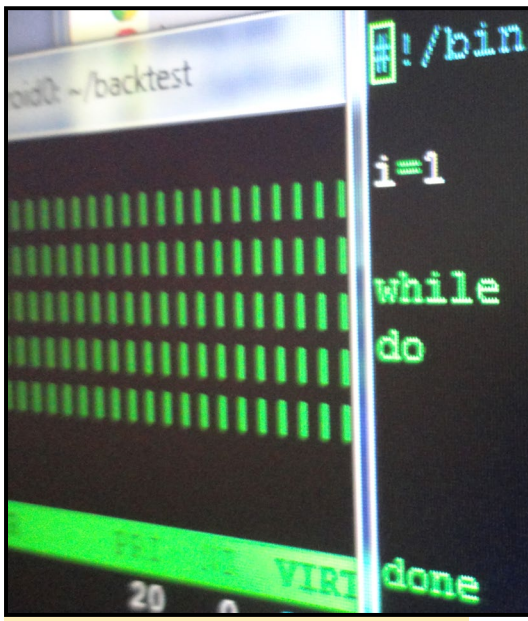
Every bash script should start with:

```
#!/bin/bash
```

This lets your system know which interpreter to use. Interestingly, the "#!" symbol is known as a "shebang". I suggest using the word as much as possible, since it is under-utilized apart from the chorus of old Ricky Martin songs. In this article, you will learn how to use bash to:

**Do something every time your Odroid boots**
**Do math and concatenate strings**
**Accept arguments**
**Run a loop**
**Launch a bunch of child processes with different arguments**

### Make a robot cat




**Basic bash skills are the starting point for learning how to be a Linux guru**

For our first example, let's make a script concatenate two strings.

**1. Create a file called meow.sh**

```
#!/bin/bash

uno="me" # i'm a comment… uno
references a string
dos="ow"  # you probably get
where I'm going with this
echo "${uno}${dos}" # variables
are referenced ${…}
```

**2. After you make the file, type the following at a command line prompt:**

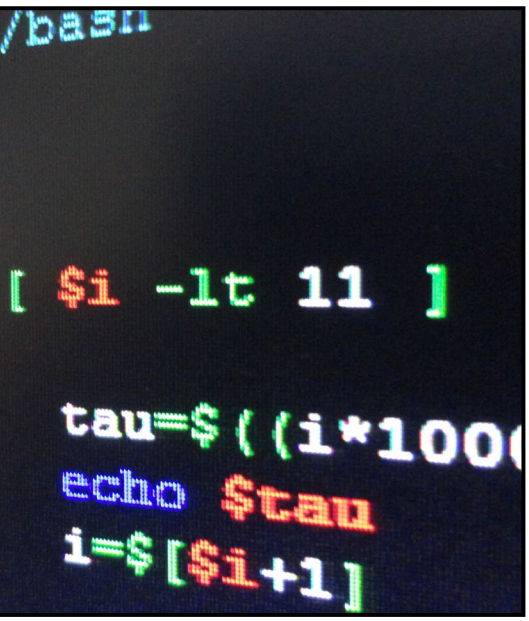
```
$ sudo chmod +x meow.sh
```

**3. Now you can run the script by typing:**

```
$ ./meow.sh
```

It should print "meow". Bash is now a robot cat!; Now, let's modify our script so we can change what our cat can say. Maybe he will say something uplifting, maybe something hurtful, but the point is the power is yours. Change meow.sh to:

```
#!/bin/bash
uno=${1:-"me"} #use first param-
```




```
eter supplied, else use default
provided
dos=${2:-"ow"}  #use second param
supplied, else use default pro-
vided
echo "${uno}${dos}" #nothing new
here…
```

Now you can type `./meow.sh` and get our default cat behavior from the previous example, OR you can type `./meow.sh <whatever_you_want>` and your robot cat will oblige by echoing it back. For example, suppose your cat sees another attractive robot cat and wants to get its attention. Type `./meow.sh meeeeeee owww` and start a conversation!

## Enable automation

Suppose, however, that you actually want to do something useful with bash, such as writing a script to execute a few commands every time your ODROID boots up. I use the following script, for example, to start a TCP server on each U3+ in my cluster to respond to client work requests:

**1. Create a file named on_boot.sh:**

```
#!/bin/bash
sleep 15   # pause execution for
15 seconds
cd /home/of/your/file    # change
places
$ twistd -y WorkNode.tac -l logs/
node.log   # put your command here
```

**2. Now edit `/etc/rc.local` and add the following lines before the exit 0:**

```
cd /the/folder/where/on_boot/
lives/
./on_boot.sh
```

Also, don't forget to make your script executable with `sudo chmod +x on_boot.sh`.

## Do some math

Bash can also do useful things like execute for loops and do math. As an example, create a file called math_loop.sh:

```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8 9 10
do
      tau=$((i*1000))   # aka tau
is i * 1000
      echo tau
done
```

After giving it the same file permissions as above, the script will print 1000, 2000, …, 10000 on your console. Alternatively, you could refactor the above script using a while loop instead. As you might expect, comparison operators look a little different in bash:

```
#!/bin/bash
i=1
while [ $i -lt 11 ]   # -lt means
less than
do
        tau=$((i*1000))
        echo $tau
        i=$[$i+1]   # increments
i by 1
done
```

Both programs perform the same behavior, with a different way of getting there. Let's put it all together now and create a script to execute multiple scripts in a loop. I can use this script whenever I need to manually submit a series of work requests to my cluster:

```
#!/bin/bash
symbol=$1
date=$2
for i in 1 2 3 4 5 6 7 8 9 10
do
        ip=$((i-1))
        tau=$((i*1000))
        path="${symbol}_${date}"
        python WorkClient.py -t
$tau -f $path -i 192.1.1.$ip
done
```

This script takes two arguments, the futures symbol name and date in YYYYMMDD form, and uses them to submit jobs using WorkClient.py (a TCP client). Notice how the bash script creates the tau, path, and ip parameters that are passed to WorkClient.py. Bash is a powerful tool that every ODROID admin should have at their disposal.

## Summary

In this article, I presented some very basic ways that we can use bash, including simple math operations, string concatenation, and loops. Passing arguments to your scripts really expands the number of functions you can perform. If you are anything like me, you want to automate as much of your workflow as possible, and bash can help relieve you of many tedious tasks in need of automation, as well as provide you with a quick and powerful tool for interacting with your systems.

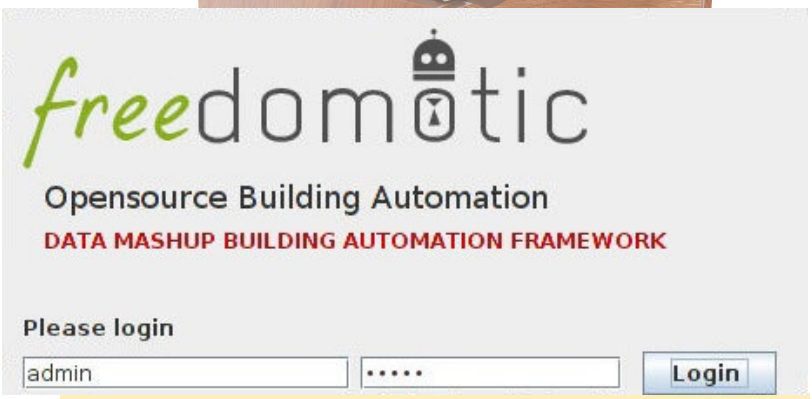
## Sample Code

To download the example code used in this article, please visit www.odroidcluster.com, and direct any questions, errata, or pen-pal requests to odroidcluster(at)gmail.com.

# INSTALLING FREEDOMOTIC

## A BUILDING AUTOMATION FRAMEWORK

by Venkat Bommakanti




The Freedomotic open source build automator login screen

Are you interested in Home Automation? The ODROID systems make for great automation controllers and managers! They can be used to control your house remotely, or set tasks to be automatically performed by the home each day. This article describes the use of the U3 as the heart of a building, home, or office automation system using the Freedomotic software platform.

## Requirements:

**Any ODROID board, with an appropriate power adapter.**
**A bootable 8+ GB MicroSD card or eMMC module containing the latest U3 Lubuntu image available from the Hardkernel website.**
**A network where the device has access to the internet and the ODROID forums.**
**Optional SSH access to the U3 via utilities like PuTTY (MS Windows 7+) or Terminal (Mac, linux to perform the steps from a remote host computer.**

## Install Apache maven

Apache maven is a software project management and comprehension tool. The freedomotic software platform uses this infrastructure. it can be installed using the command:

```
$ sudo apt-get install maven
```

## Fetch freedomotic source code

Because no relevant pre-built ARM-based ubuntu packages exist, you will have to build the framework from its source-code, right on the U3. This article does not address a cross-compilation option.

Create a sub-directory to receive the sources and change to it, using the commands:

```
$ mkdir freedomotic-src
$ cd freedomotic-src/
```

Fetch the source code from the relevant git repository, to this location, using the command:

```
$ git clone https://github.com/
freedomotic/freedomotic.git
```

## Build freedomotic using maven

A new sub-directory holding the entire source-tree is automatically created. Navigate to that directory, then type the following to launch the build process:

```
$ cd freedomotic/
$ mvn clean install
```

After the build is completed, Maven may be launched using the mvn command.

## Setup the example data

Create a copy of the example data using the command:

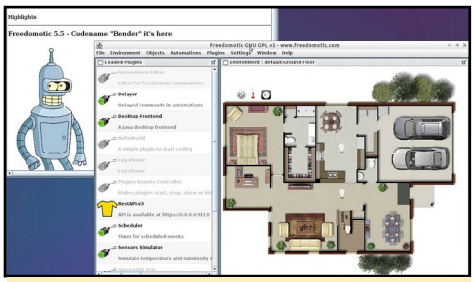
```
$ cp -r data-example/ \
framework/freedomotic-core/data
```

## Run freedomotic

Launch the platform using the command:

```
$ java -jar framework/\
freedomotic-core /target/\
freedomotic-core/freedomotic.jar
```

You will then be presented with a login dialog window.



The Freedomotic welcome screen features Bender from Futurama!

Use id/pwd of admin/admin to log-in. Now the automation world is yours! Make sure to research the automation topic well and take all the appropriate precautions while automating and making public, any aspect of your life, including but not limited to, home automation and universal web access. The information here is only for educational and fun purposes.

## Plugins

Freedomotic works on the extensible principle of plugins. You can use free open source plugins or build your own. The GIT repository holds the complete SDK that has all the code you need to build and test your own plugins. After compiling for the first time, open the freedomotic-core project with your favourite IDE.

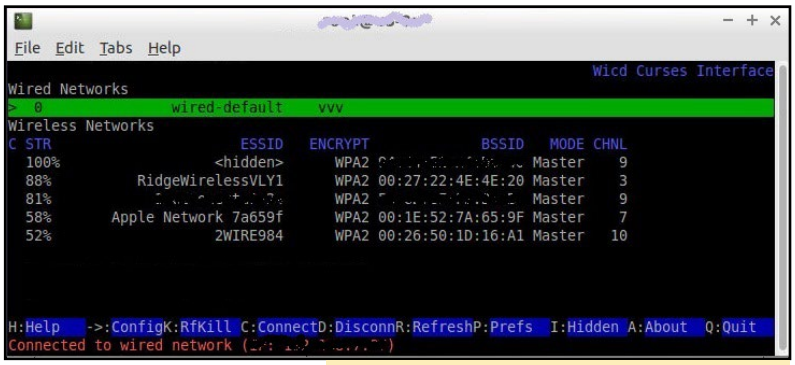
To develop your own plugin you can start from the hello-world example project included in the plugins/devices/hello-world directory. Open it in your IDE, make some changes and compile it. It will be automatically installed into the freedomotic runtime freedomotic-core project. Just restart freedomotic-core to try out your latest changes.

For additional information or questions, please visit the original information sources at http://bit.ly/1qqjyun, http://bit.ly/1nL16ZI, and http://bit.ly/1Cdwdai.

# INSTALLING WICD

## A NETWORK CONNECTION MANAGER

by Venkat Bommakanti




WICD allows the ODROID to easily connect to any wired or wireless network

If you wish to use a network connection manager to manage both wired and wifi interfaces, the lightweight wicd utility is an option. It is an alternative to the gnome based NetworkManager tool. This article describes the installation of wicd on the ODROID U3.

## Requirements

1. An ODROID U3 board, with an appropriate power adapter.
2. A MicroSD card (with an SDCard reader/writer) containing the latest U3 specific lubuntu desktop desktop image, or an 8+ GB eMMC card.
3. A network where the device has access to the internet and the ODROID forums.
4. SSH access to the U3 via SSH utilities like PuTTY (MS Windows 7+) or Terminal (Mac, linux) from the remote desktop.

## Install wicd and needed infrastructure

Run the following command to install all the needed components:

```
$ sudo  apt-get install \
wicd-curses wicd
```

## Start the wicd service and application

Start the application and required service, using the command:

```
$ sudo service wicd
start
 * Starting Network connection
manager wicd
 [ OK ]
$ sudo wicd-curses
```

## Verify installation

An user-interface should be presented listing various networks, like the one shown at the article main image:

Please refer to the man-pages or online-help for usage details. For additional information or questions, visit the original information sources at http://bit.ly/1powWRH and http://bit.ly/1vTU7Df.

# 3DPONICS

## AN OPEN SOURCE ODROID-POWERED GARDENING SYSTEM

*by Lucy Morrissey*

Are you a city dweller lacking space for a garden, a fresh food lover tired of paying a fortune for vegetables at the grocery store, or simply too busy to manually take care of vegetables? Now, you can grow your own food at home using a next-generation hydroponics gardening system called 3Dponics, available for free at http://www.3Dponics.com. Best of all, you can print most of the parts at home!

For the past two years, 3Dprintler, an Ottawa-based technology lab, has been developing a 3D-printable hydroponics system, and is offering it to the public as a completely open source project. Because of its affordable price and powerful processor, 3Dponics recently chose the ODROID-U3 as the recommended hardware for connecting the garden sensors with the Internet, allowing the garden to be operated and monitored remotely.

### Getting started

**1. Download the open source files from the 3Dponics website or the 3Dprintler account on http://www.thingiverse.com.**
**2. 3D print the files yourself, or use a 3D printing service to create the system components.**
**2. Collect the parts that are not 3D printable from your home or hardware store.**
**4. Set up the system, using the detailed instructions and easy video tutorial available from the 3Dponics website.**

### Software and hardware

Various types of software, including SolidWorks, SketchUp and AutoCAD, were used to design the 3Dponics components and prepare them for the printer. Although 3Dponics needed the software to create the files, you don't, since you can simply access the prepared files online. However, you are welcome to alter the files and share your changes with other users.

When the original files were designed and prepared, the parts were 3D-printed using a Makerbot Fifth Generation Replicator and Formlabs Form 1+. The entire system can be printed in just five hours at a low resolution setting.

**With 3Dponics, vegetables can be grown in almost any small space**

If you don't have access to a 3D printer, you can still build the 3Dponics system by contacting the 3D-printing location service at http://www.3dhubs.com. They will connect you to someone nearby who has a 3D printer and can print the files for you. With this service, you don't need to pay high shipping costs, and can have the components ready quickly. The most important parts are the drip nozzle, conduit and silencer.

### Printable system parts

**Drip Nozzles for Plastic Bottles**
**Conduit with a Hole for Aquarium Air Bubbler**
**Conduit for Aquarium Air Bubbler**
**Silencer to Reduce Noise Levels**
**Outer Bottle Clip**
**Inner Bottle Clip**

A 3Dponics garden setup using inexpensive household items

**Bottle Height Adjuster**
**Pump Connector for Multiple Systems**
**Modular Support Rod for Tubing**
**Bottle Sleeve Root Protector**
**Sprinkler Head for Foliage Enhancement**
**Sprinkler for Root Enhancement**

## Non-printable system parts

**3-4 empty plastic bottles (1L or 2L recommended)**
**Hagen Marina 200 Quiet Aquarium Air Pump (or equivalent)**
**10 ft of aquarium air bubbler tubing**
**20 zip ties**

## Autonomous system

Thanks to the ODROID-U3, the 3Dponics system is Internet-enabled. After experimenting with several different units, the ODROID-U3 was found to work best for powering the 3Dponics system because:

**1. The ODROID-U3 costs less ($65) than other comparable boards such as the Intel Nuc i3 ($300), and its microSD card, USB HD cameras and data sensors are affordable,**
**2. The operating system (OS) is free (Android or Linux),**
**3. You can 3D print your own custom ODROID-U3 case at http://bit.ly/1qmCfAv (thanks to Thingiverse user miguif).**

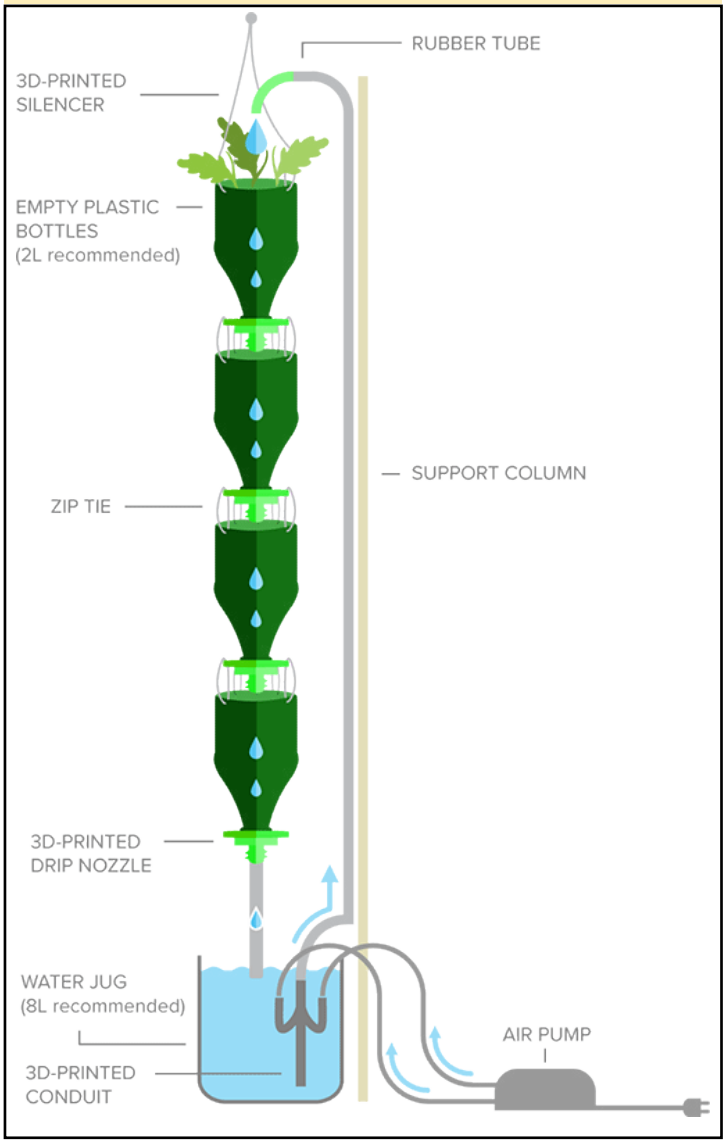
## ODROID-U3 app

An app called the 3Dponics Farm App is under active developed, which users can install on their own ODROID-U3 device. The app takes data from sensors on the 3Dponics system and communicates with the 3Dponics servers, which can then forward the data to a smartphone.

Users simply open the ODROID app on their phone (Android or iOS) and connect with the ODROID-U3 server to monitor and control their own 3Dponics system. They can check the temperature and moisture levels; turn the system on or off, set up a timer (for example, schedule operation when rates are lowest), watch live video feed, sync the system with the sunrise and sunset, and connect it to solar panels as a renewable source of energy.

Because the system's footprint is already small, only 4.5 watts of electricity is required to operate a 3Dponics system. By adding solar cells and a battery, the air bubbler and ODROID-U3 can be fully powered by sunlight. 3Dponics is the most economical way to start growing your own vegetables right now!

**A typical design for the 3Dponics garden watering system**



# WALL-E

## BUILDING YOUR OWN ROBOT AT HOME PART I

*by Vincenzo Siriaani*


**The basic remote-control Wall-E Pixar model is available for purchase from Amazon, ready for an ODROID-U3 brain!**

I had a dream since childhood that I wanted a robot to live in my house. Recently, since home robotics has become affordable, I began programming Arduino-based robots, and created some to walk around and avoid obstacles, but they were basic robots without any personality. About 6 months ago, I began studying Python, because Java and C were too difficult for me.

I bought an ODROID-U3 with a Linux eMMC module, but I had never used Linux before. In fact, I could not even install the basic OpenCV computer vision library. I eventually downloaded an OS image from the ODROID Robotics forums called Linaro 12.11 Robotics Edition (ROS), and I used some example Python code to teach my robot to detect faces. After I got OpenCV working, I connected an Arduino to the U3 using a USB cable and sent some strings.

The inspiration for my Wall-E robot began when i saw the movie "Wall-E", then bought a used Wall-E U-Command, made by Pixar (http://amzn.to/1lBYyC2). In my research, I found the site of DJ Sures (http://bit.ly/1pfKxEQ) and I realized that it was possible to make the same robot without a Windows PC, using an ODROID-U3 instead!

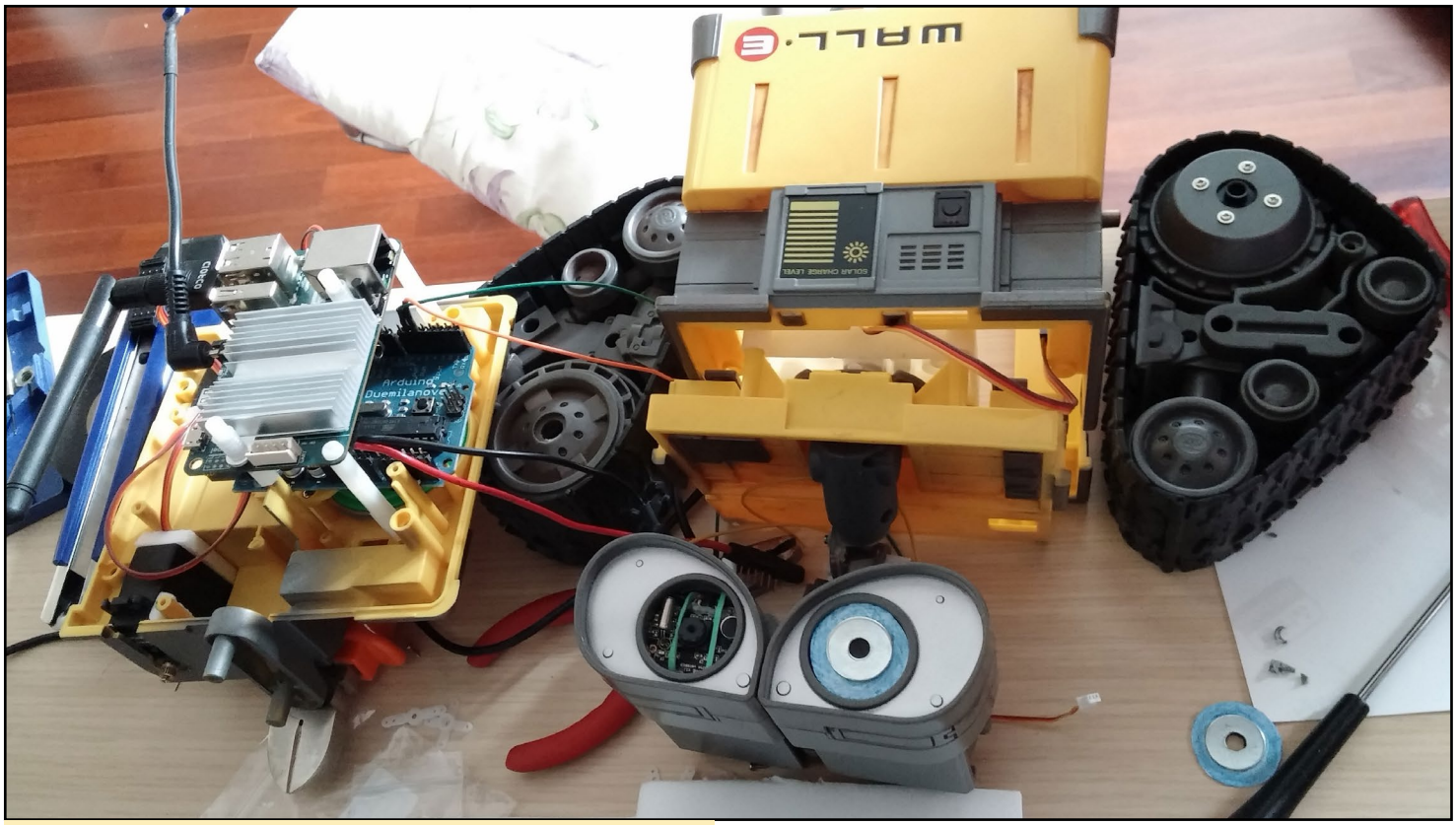
## Materials

1. Arduino 2009
2. ODROID U3
3. Webcam from Hardkernel
4. eMMC with Ubuntu pre-installed
5. MicroSD card class 10 (for Linaro)
6. 2 servos with metal gear for the "caterpillar"- Turnigy Digital high torque bearing servo 26.0g/3.5kg/.12sec
7. 2 servos for head pan and tilt - HK15168 Coreless Analog Micro Servo 8g / 1.2kg / 0.12s
8. 2 servos ultramicro for the eyes - HK-282A Single-Screw, Ultra-Micro Servo 2g / 0.2kg / 0.08sec
9. 2 servos for the arms (the same as the pan and tilt servos) - HK15168 Coreless Analog Micro Servo 8g / 1.2kg / 0.12s
10. A voltage regulator of 5 volt (max 5 Ah) to feed the circuits - TURNIGY 3A UBEC w/ Noise Reduction
11. Lipo battery - Turnigy nano-tech A-SPEC 2200mah 3S 65~130C Lipo Pack

I hacked the Wall-E robot that I bought and inserted the head servos and webcam, as shown in the accompanying pictures. Then, I started developing the software controllers. For now, I wrote the code in Python 2.7, without any GUI. Part of the code is for face detection and tracking, part is for sending the video to another computer, and another part is to receive the commands.

**Wall-E rescued a friend and decided to revive him with an ODROID**

```
# Wall-E Main Controller Python Script

import threading
import time
import cv2
import timeit
import socket
import serial
import numpy
# these are all the libraries
global command
global face_positionx
global face_positiony
#queue
face_positionx = 0
face_positiony = 0
command = ""

class tasks(threading.Thread):
    def __init__(self, threadID, name, counter, functions):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
        self.functions = functions
    def run(self):
        print "Starting " + self.name
        self.functions()
        print "Exiting " + self.name
# this is a class that performs the server that receives the command, i don't know why, but
# without this class the thread doesn't works

def server():
    global command
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind(("localhost",5001))
    server_socket.listen(5)
    client_socket, address = server_socket.accept()
    while 1:
        client_command = client_socket.recv(1024)
        client_socket.send("From " + repr(address) + " Recived " + repr(client_command))
                command = client_command
        if client_command == "q":
            time.sleep(5)
            client_socket.close()
            server_socket.close()
            break
    print "Uscito server"

# this is a server that receives the command for the Wall-E, you can change the "localhost"
# with a ip number and it works out of the same machine.  to send the command
# out of the home network you must to redirect the port in the settings of your router

def cattura_immagine():
    time.sleep(1)
    global command
    global stringData
```

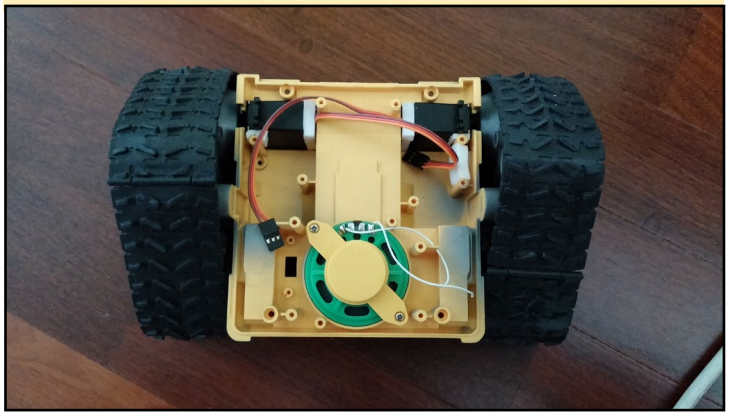



**Wall-E's parts are disassembled in order to test the eye motors**

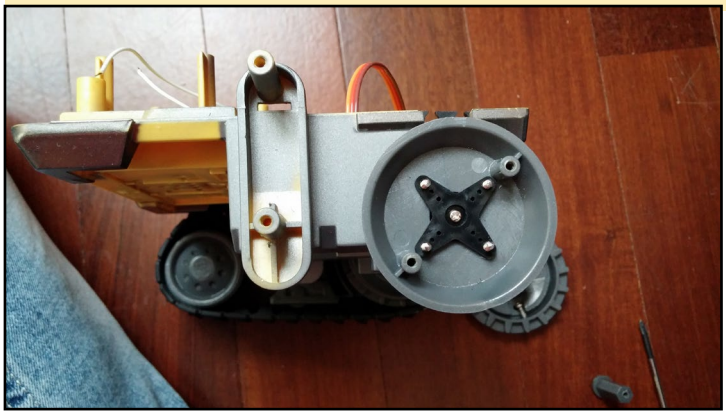
```
    face_positionx = 0
    face_positiony = 0
    command_arduino = 0
    TCP_IP = "192.168.1.107"
    TCP_PORT = 5002
    print "before socket.socket"
    sock = socket.socket(socket.AF_INET, socket.SOCK_
STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_RE-
USEADDR, 1)
    print "after socket.socket"
    sock.bind((TCP_IP, TCP_PORT))
    sock.listen(5)
    client_socket_video, address = sock.accept()
    # this part of code start the variables and video
serve, you can change the IP
    # number as the server for commands

    print "ARDU"
    try:
        arduino = serial.Serial('/dev/tty-
USB0',115200)
        arduinoconnesso = 1
        print "ARDUINO CONNESSO"

    except:
        arduinoconnesso = 0
        print "ARDUINO NON CONNESSO"
    # this piece try to connect the ODROID U3 to the
arduino, you can change the port name and
    # the speed

    SCALA = 2
    TRAINSET = "/home/linaro/opencv-2.4.6.1/data/lbp-
cascades/lbpcascade_frontalface.xml"
    classifier = cv2.CascadeClassifier(TRAINSET)
    # this set the cascade for the search of faces,
lbp is faster, but you can change

    webcam = cv2.VideoCapture(0)
    time.sleep(3)
    if webcam.isOpened():
        print "Video aperto"
        ret, frame = webcam.read()
        if ret:
            print "ret True"
            contatore = 0
            # this start the webcam capture, you can
change the number of the webcam port

            while(1):
                contatore = contatore + 1
                ret, frame = webcam.read()
                t = timeit.default_timer()
                height = frame.shape[0]
                width = frame.shape[1]
```

```
                cv2.putText(frame, "Larghezza " +
repr(width) +
                            "Altezza " + repr(height),
(50,10),
                             cv2.FONT_HERSHEY_SIMPLEX,
0.6, (255,255,255))
                minisize = (frame.shape[1]/
SCALA,frame.shape[0]/SCALA)
                miniframe = cv2.resize(frame, mini-
size)
                gray = cv2.cvtColor(miniframe, cv2.
COLOR_BGR2GRAY)
                gray = cv2.equalizeHist(gray)

                # all this part of code starts to
read frames for ever, gets the time for FPS,
                # shrinks the frame, turns the frame
first to grey scala, then makes Histogram
                # Equalization, to improves the con-
trast in the image, to speed up the process
                # of face detection

                if command == "4":
                    command_arduino = "4"

                if command == "5":
                    command_arduino = "3"

                if command == "r":
                    command_arduino = "2"

                if command == "t":
                    command_arduino = "1"
                if command_arduino != 0:
                    if arduinoconnesso == 1:
                        arduino.write (repr(command_
arduino))
                    command_arduino = 0
                    command = ""

            # this is for the manual command of pan
and tilt of Wall-E head

                if command == "f":
                    faces = classifier.
detectMultiScale(gray)
                    for f in faces:
                        x, y, w, h = [ v*SCALA for v
in f ]
                        cv2.rectangle(frame, (x,y),
(x+w,y+h), (0,0,255))
                        cv2.rectangle(frame, (x+w/2-
1,y+h/2-1), (x+w/2+1,y+h/2+1), (0,0,255))
                        cv2.putText(frame, "X =
"+repr(x+w/2)+" Y = " + repr(y+h/2), (5, 25),
                            cv2.FONT_HERSHEY_SIMPLEX,
0.6, (255,255,255))
                        face_positionx = repr(x+w/2)
                        face_positiony = repr(y+h/2)
                        cv2.putText(frame, "Volti n.
" + repr(len(faces)), (x-50,y-10),
                            cv2.FONT_HERSHEY_SIMPLEX,
0.6, (255,255,255))

# this performs the real face detection,
                    # (http://bytefish.de/blog/
opencv/object_detection/)

                    if face_positionx != 0 or face_
positiony != 0:
                        if int(face_positionx) < 220:
                            command_arduino = "4"
                        if int(face_positionx) > 390:
                            command_arduino = "3"
                        if command_arduino != 0:
                            if arduinoconnesso == 1:
```

**A front view of Wall-E's tractor wheels and motor**



**A side view of Wall-E's tractor wheels with the treads removed**



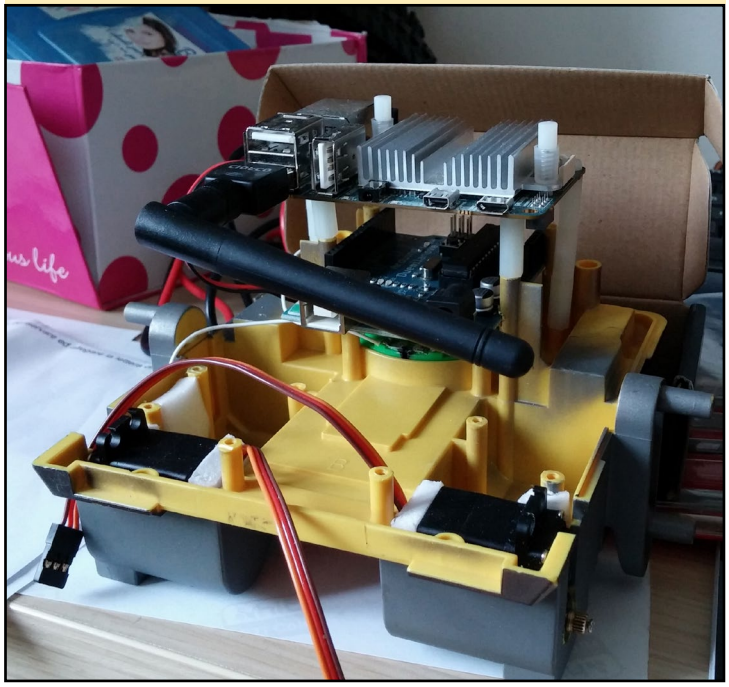
**Consider a waterproof case for the U3, since Wall-E loves to swim!**

```
                                arduino.write
(repr(command_arduino))
                            command_arduino = 0

                        if int(face_positiony) < 160:
                            command_arduino = "1"
                        if int(face_positiony) > 320:
                            command_arduino = "2"
                        if command_arduino != 0:
                            if arduinoconnesso == 1:
                                arduino.write
(repr(command_arduino))
                            command_arduino = 0
                    face_positionx = 0
                    face_positiony = 0
                dt = timeit.default_timer() - t
                cv2.putText(frame, "FPS =  " +
repr(1/dt), (5, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(255,255,255))

          # this part sends the command to Wall-E
head to track the faces

                encode_param = [int(cv2.IMWRITE_JPEG_
QUALITY),90]
                result, imgencode = cv2.imencode('.
jpg', gray, encode_param)
                data = numpy.array(imgencode)
                stringData = data.tostring()
                if contatore == 3:
                    client_socket_video.
send(str(len(stringData)).ljust(16));
                    client_socket_video.
send(stringData);
                    contatore = 0
          # this send the stream video to the client
video, it send strings of text!!
                cv2.imshow('frame',frame)
                if command == "q":
                    webcam.release()
                    cv2.destroyAllWindows()
                    break
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    command = "q"
                    break

            sock.close()
            time.sleep(2)
            if arduinoconnesso == 1:
                arduino.close()
            webcam.release()
            time.sleep(2)
            cv2.destroyAllWindows()
            print "Uscito Opencv"

            time.sleep(2)
               # this closes all when you push "q"

# this starts everything
print "Comincio"
thread2 = tasks(2, "server", 2, server)
thread2.start()
cattura_immagine()
```

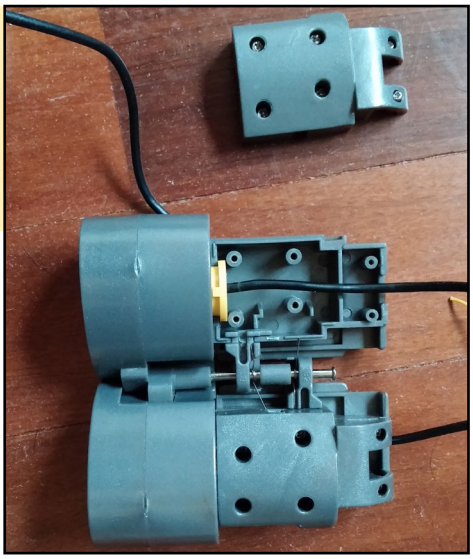
The script above is the main software, but there are 2 other programs needed. After the main module is started, the central command service and the video service are launched:

```
import socket                 # Import socket module

s = socket.socket()           # Create a socket object
host = "localhost"            # Get local machine name
port = 5001                   # Reserve a port for
your command service

s.connect((host, port))
print "Connect to " + host
print "Use f for detection, 4, 5, r, t for manual
command of the head, q to quit "
while (True):
    command = raw_input ("Command? ")
    s.send(command)
    print s.recv(1024)
    if command == "q":
        break
s.close                       # Close the socket when
"q"

video client:

import socket
import cv2
import numpy

def recvall(sock, count):
    buf = b''
    while count:
        newbuf = sock.recv(count)
        if not newbuf: return None
```

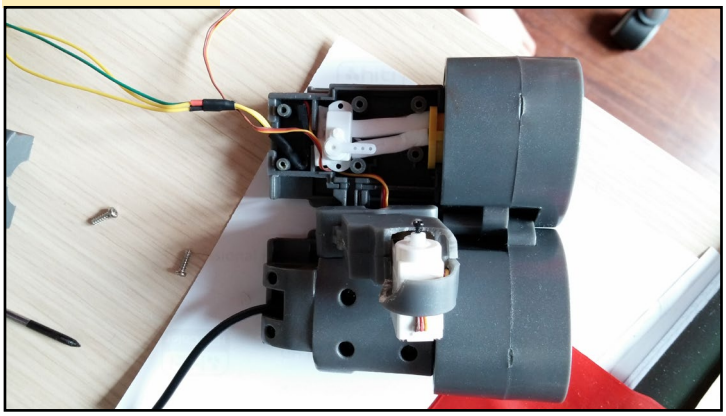
**An ODROID-U3 mounted to the Wall-E base, with wireless dongle**



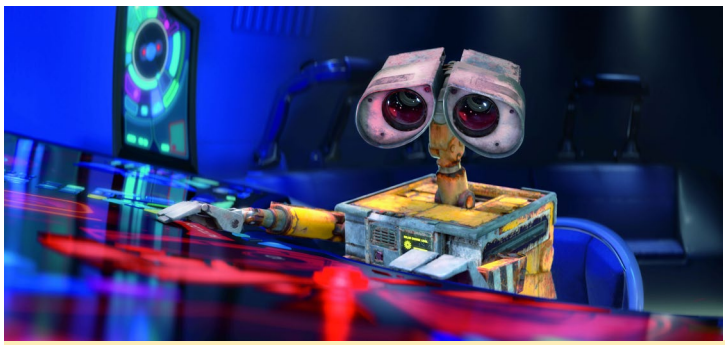
**Closeup of the servo motors used to control Wall-E's eye movements**



**Wall-E's eyes are his signature feature, and need to be very expressive!**



```
        buf += newbuf
        count -= len(newbuf)
    return buf
# http://stupidpythonideas.blogspot.it/2013/05/sock-
ets-are-byte-streams-not-message.html

TCP_IP = "192.168.1.107"
TCP_PORT = 5002

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
while (True):

    length = recvall(s,16)
    stringData = recvall(s, int(length))
    data = numpy.fromstring(stringData,
dtype='uint8')

    decimg=cv2.imdecode(data,1)
    cv2.imshow('SERVER',decimg)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
s.close()
cv2.destroyAllWindows()
```



**Wall-E using an ODROID-VU touchscreen to program his new buddy**

```
# http://stackoverflow.com/questions/20820602/image-
send-via-tcp

The Arduino code is very simple, and does noth-
ing more than receiving the messages from ODROID and
sending them to the servos.  Note that the servos of
the head are connected to pin 5 and 6.

#include <Servo.h>

Servo myservo1;
Servo myservo2;

byte rx = 0; // variabile per contenere il carattere
ricevuto.
int pan = 90;
int tilt = 77;

void setup()
  {
Serial.begin(115200); // imposto la seriale per
lavorare a 115200 baud
myservo1.attach(6);
myservo1.write(pan);
myservo2.attach(5);
myservo2.write(tilt);
Serial.flush(); // svuoto il buffer di ricezione se-
riale.
delay(100);
  }

void loop()
  {

  if (Serial.available() >0) // Controllo se il buf-
fer di ricezione contiene qualcosa
    {
      rx = Serial.read(); // leggo il carattere
ricevuto e lo memorizzo in rx
      Serial.flush(); // svuoto il buffer di ricezione
seriale
      if (rx != '0')
       {
        if (rx=='1')
          {
            if (pan >= 35)
            {
              pan = pan - 2;
            }
          }
        if (rx=='2')
          {
            if (pan <= 135)
            {
              pan = pan + 2;
            }
          }
        if (rx=='3')
          {
            if (tilt >= 35)
            {
              tilt = tilt - 2;
            }
          }
        if (rx=='4')
          {
            if (tilt <= 125)
            {
              tilt = tilt + 2;
            }
          }
       }
        myservo1.write(pan);
        myservo2.write(tilt);
  }
}
```

As my next step, I want to add speech and voice recognition to Wall-E, and then connect the servos to the Arduino to make my robot mobile!

**Wall-E was inspired by 3Dponics and started his own garden!**

# WEATHER FORECAST ON THE DESKTOP

## WHAT CHANCE IS THERE TO CATCH FISH NEXT WEEKEND?

by Jussi Opas

Should one plan to go fishing next the weekend, play golf, or just stay inside and use a computer? That is a question that outdoor hobbyists are always asking during the work week! To help answer this question, Linux users can can install a forecast directly onto the desktop. In this article, I present a couple of applications that display the current weather and do some forecasting in order to help to plan future outdoor activities.

### XFCE

The XFCE desktop contains a weather plugin and it can be added to the bottom panel. One can add a Weather Update widget into panel, as is shown in the first screenshot. To install the XFCE weather plugin, type:

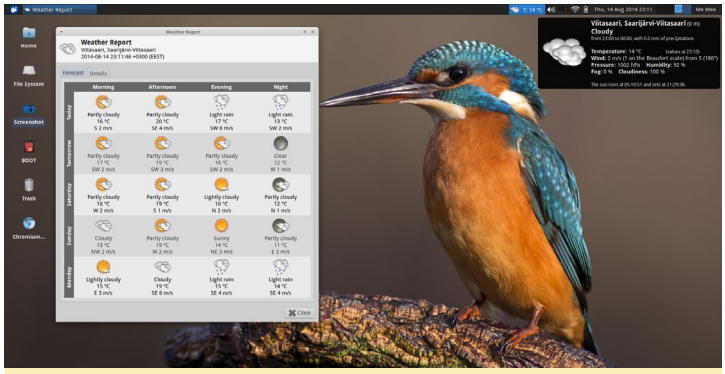
```
$ sudo apt-get install xfce4-weather-plugin
```

From the panel, one can open the properties by right-clicking to access the configuration dialog.

The Change button opens a second dialog that can be used to select a location. The city closest to our fishing river is "Viitasaari", which can be located by entering it into the text box and pressing the Search button. From the results list, one can select the aimed location and get its forecast.

Hovering over the weather widget with the mouse shows a tooltip with more information such as temperature, wind, humidity, and cloudiness. By clicking the widget, one can get a forecast for the next few days.

In order to catch trout, then the weather should be cloudy or somewhat rainy, rather than bright. Trout are active late in the evening or in the early night hours. Meanwhile, winds are hostile for fly-fishing, since it affects the ability to cast, and one can not see fish activity on a windy water surface.



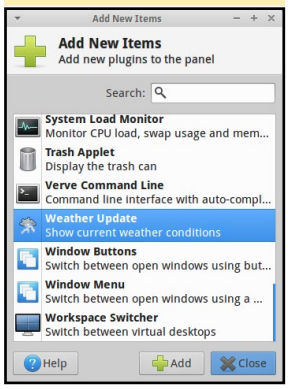
**In this desktop background from http://bit.ly/IlGnGrj, there is a kingfisher as wallpaper. Maybe it is predicting something about our chance of catching fish?**

Several widgets such as wind and temperature may be added to help us make our fishing decisions. In our example shown in the screenshot, the aimed fishing days are Saturday and Sunday, and it seems that the best fishing weather is on the beginning of Saturday night, as there is practically no wind, somewhat cloudy and dark. The Sunday morning is also promising, since it will be cloudy.

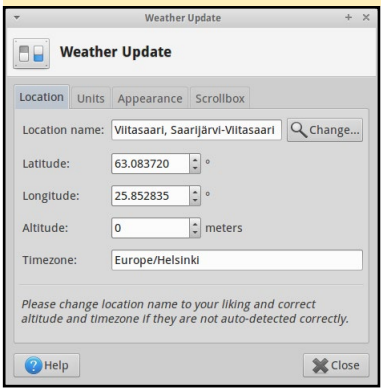
### Gnome and Unity

The users of almost all other desktops, including Gnome and Unity, also have weather application available from several software sources. Search for "weather" in the Ubuntu or

**Weather Update**



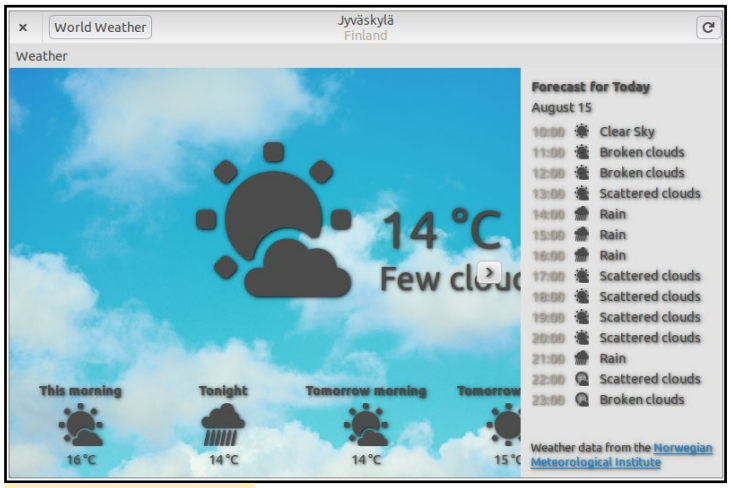

**Configuration dialog for the Weather Update application**



Lubuntu Software Center. For instance, gnome-weather is available in Lubuntu Software Center, as shown in the screenshot, and may be installed by typing:

```
$ sudo apt-get install gnome-weather
```

Using gnome-weather, the user can choose a certain location, and see the weather for that area. One can see a weather forecast for the current day, and also view a reasonably accurate prediction for the next week.




**Gnome Weather**

## Gone Fishing

To continue our story, the weekend came and we travelled to the river, based on the information that we had from our weather widget. The sky was cloudy on Saturday evening, but instead of being cloudy on Sunday morning, it was bright, and there were only a few clouds. The predicted weather took place but it happened earlier than expected. The interpretation here is that the air masses were moving faster than what was forecast.

On Saturday evening, when it was cloudy, just before a light rain, the fish were active and responded to a tinsel streamer. Unfortunately, the fish escaped when the tippet line broke!

Soon it became dark, so we had to stop fishing. It was better to go to sleep and start fishing again on Sunday morning. A tinsel was used because it had some luck on Saturday, but on Sunday there were no bites at all! So, we started using a fly called a "hassle" in the deep pools. The chance of catching any fish with the hassle fly was poor, since our experience is that the trout ignore it. After some time, and despite the bright weather, a trout was finally caught!

**A hassel fly lure has two partridge feather hackles, and the used pupa hook is weighted so that it floats upside down in a pool bottom**

**Tinsel lure - A sample tinsel streamer trout lure that imitates a small fish**

## Fishing Prediction Model

**This trout was caught with the hassle fly and then released**

After we came home from our trip, we formulated a computational model of probability to catch fish based on certain variables such as the weather and type of lure:

```
if (weather is cloudy) {
      probability = good;
}
if (it is evening) {
      increase probability;
}
if (tinsel is used) {
      increase probability;
} else if (hassle is used) {
      decrease probability;
}
if (fisherman is skilled) {
      increase probability;
}
```

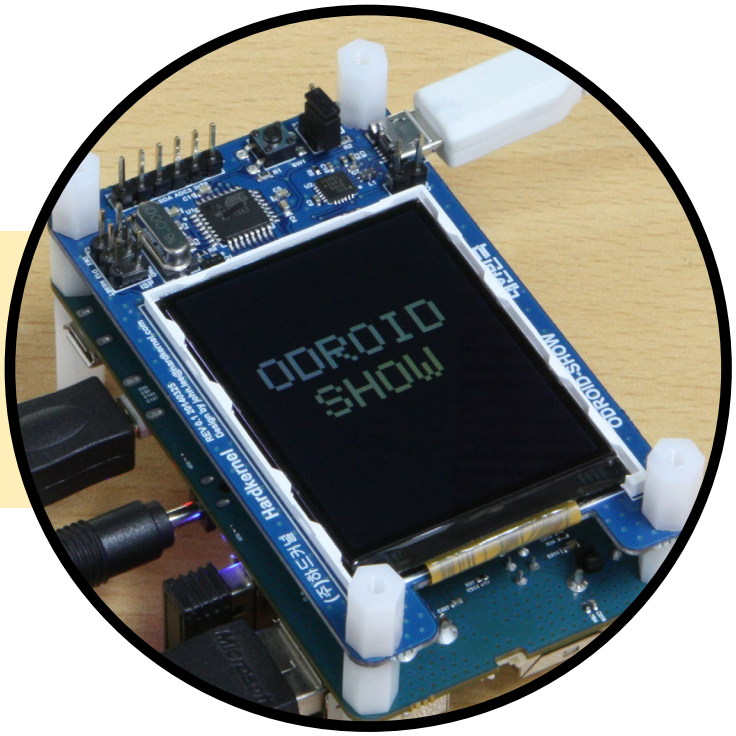
We can test this method against what actually happened; the productive fishing happened on a bright Sunday morning, with a hassle fly and an unskilled fisherman. Despite these disadvantages, a fish was still eventually caught.

If one wants to build a better model for possibility of catching fish, it could be based on Bayes' Theorem (`http://bit.ly/1nwkWIA`), which describes how computations can be made with conditional probabilities. A more advanced model could use also some history data from actual weather. For instance, one should know whether it has been rainy in last two weeks or how warm it has been. This reference data could be collected with an ODROID weather board, located at the fishing resort, with its data available in the Internet.

# DIGGING (INTO) THE ODROID-SHOW

## PART 2: MAKING CONNECTIONS

*by Declan Malone*




**The two pin headers at the top right area of the board can be used to connect several external components**

In the first part of this series of articles about the ODROID-SHOW, I introduced the basics of the software running on it with enough to get you started on your own programs. In this issue, I'll focus on using the two pin headers located up at the top right of the board to connect up some simple components to the SHOW.

### Pins

If you're just starting out with Arduino-like hardware, the information above may seem a little overwhelming, so I'll take a few paragraphs here to break it down into more manageable chunks.

To begin with, the two most important columns are the "Label", which is what is printed on the board and lets you identify the pin, and the "Function" column, which is the primary use for that particular pin. Looking at the Function column, you should be able to see these distinct types of pin:

*** power pins, which supply 3.54 volts and count as a "high" logic level**

*** a ground pin, which is needed to form an electrical circuit and counts as a "low" logic level**

*** digital pins, which can be in either a "high" or "low" state**

*** analog pins, which are connected internally to an analog to digital converter and can measure any voltage value between 0 (ground) and 3.45 volts (high)**

*** a reset pin, which is usually unused**

**Summary of the functions of each ODROID-SHOW pin**

| Label | Function | Alt. Function | ATMega/Port | Pin Change Interrupt |
|---|---|---|---|---|
| 3v45 | Power | - | | |
| A5 | Analogue pin 5 | SCL (I2C clock) | 28/PC5 | 13 |
| A4 | Analogue pin 4 | SDA (I2C data) | 27/PC4 | 12 |
| A3 | Analogue pin 3 | - | 26/PC3 | 11 |
| D2 | Data pin 2 | Hardware Interrupt 0 | 4/PD2 | 18 |
| GND | Ground | | | |
| RESET | Bring low to reset/program chip | - | 1/PC6 | 5 |
| D11 | Digital Pin 11 | SPI MOSI / PWM / Output Compare Timer 2A | 17/PB3 | 3 |
| D12 | Digital Pin 12 | SPI MISO | 18/PB4 | 4 |
| D13 | Digital Pin 13 | SPI SCK | 19/PB5 | 5 |

Any of the analog pins can be programmed to operate just like a digital pin if that is what you want, but the reverse is not true. Typically, digital pins are connected to switches or buttons of some kind, while analog pins are connected to things like variable resistors (potentiometers) or light-dependent resistors. Note also that pins can only work in analog mode when they are set as inputs; when a pin is set as an output, it can only be on or off but not some value in between.

### Alternate pin functions

Continuing on, we see that pins can have alternate functions, which can be divided into groups. They can be used for SPI, I2C, Pulse-Width Modulation (PWM, which is often used to drive motors or vary the brightness of LEDs), externally-triggered interrupts, and timer-based interrupts. The first two of these are different protocols allowing the SHOW's ATMega chip to communicate

with more complex integrated circuits (ICs), sensors and so on. Refer to the "I2C and SPI" sidebar for an overview.

Not all of the possible alternative functions are available on the ODROID-SHOW due to the fact that some of the pins are also used to communicate with the TFT hardware. In particular, pins D11, D12 and D13 are wired up to the TFT, which means that pin D11 cannot really be used for either PWM or for triggering timer-based interrupts. With the correct wiring and programming, however, it can still be used to communicate to another SPI device other than the TFT display.

The only other alternative function left to describe is for externally-triggered interrupts. I'll cover that later on when I talk about using interrupts to respond to button presses.

The "ATMega/Port" column shows alternative ways of addressing the pins. If you decide to use "avr-gcc" to write and compile programs for the SHOW instead of the "arduino" IDE, then you will need to refer to the pins using the numbering in the ATMega documentation: "avr-gcc" does not recognise Arduino-style pin numbering such as D11.

The "Port" value is another way to refer to pins. The ATMega processor groups pins into four separate "ports" or "banks" and includes methods for reading or writing any number of pins in the same port at once. It's also possible to set up interrupt handlers to monitor for changes in pin states, but (with the exception of pin D2) this can only be done with a full port instead of individual pins. Ports are named "A" to "D". For example, the pin labelled A5 on the board has the port name PC5, which means that it is in port "C", and bit 5 is used to access its value.

Finally, each pin has an associated interrupt number which is shown in the last column. As the name suggests, when a "pin change" interrupt is enabled, it causes an interrupt when the pin goes from "high" to "low" or vice-versa.

## Sensor circuits

With some of the theory out of the way, now we can get down to actually connecting up some sensors and reading their values on the SHOW. While these are admittedly very simple circuits, they are often all that is needed to add some interactivity to a program, particularly if you want to use the SHOW for a stand-alone application.

To save space, I'll describe just the main points of each Arduino circuit.

## Button press circuit

A button or switch can be wired in series between a digital pin and a 3v45 pin. In this way, when the button is pressed or the switch closed, the pin registers a 'high' value.

## Reading a potentiometer

Potentiometers (or "pots") are a form of variable resistor. They can come in rotary or linear forms, like a volume control knob or a slider in a mixing desk, respectively. They can also be "linear", where the resistance is proportional to how far the "wiper" is turned, or "log", where the resistance is proportional to the logarithm. Linear pots are generally more useful.

## Reading a joystick

Most joysticks, except for really old ones, are analog devices, with one pot for the X axis, and another for the Y. They usually have one or more buttons as well.

## Using interrupts for button presses

While it's simple to connect up discrete components like variable resistors, LEDs, switches and relays and so on to a microcontroller, as more complex components became available, people realised the need to come up with standard ways of interfacing with them. The I2C and SPI protocols were developed independently to address this problem. Most microcontrollers, and even some more powerful processors or SOCs (System On a Chip) will support one or both.

Both protocols have the concept of a shared bus, over which data travels to and from the peripheral device. Both also use a master-slave arrangement, with the microcontroller (MCU) operating as the master, telling the slave devices what to do. They also allow for connecting several different devices to the shared bus, although they differ in exactly how individual devices are addressed, meaning how the device knows that the MCU is talking to it at any given time.

## SPI

With SPI, different devices are addressed by dedicating a separate "slave select" line for each individual device. By bringing the appropriate slave select line low, the device attached to it knows that the MCU is now talking to it. With I2C, on the other hand, each device must have a unique address, which is usually a 7-bit number. Most I2C devices can have their addresses configured by means of jumpers or solder pads. The master device prefixes each message with the address of the device it wishes to talk to.

SPI has the advantage of higher data transfer rates with slave devices, so it is often used in applications like TFT displays and SD card modules. It's also capable of synchronous operation, meaning that data can be transferred in both directions at once over the MOSI ("Master Out, Slave In") and MISO ("Master In, Slave Out") pins. I2C's main advantage is that it only requires two pins even if addressing the maximum number of devices. This feature makes it a very popular choice for a wide variety of exotic sensors including the ODROID Weatherboard, controllers such as motor controllers, and I/O expanders such as those embedded in the U3 I/O shield.

There's also a wealth of modules available for the Arduino platform that could be connected to the I2C or SPI pins of

the ODROID-SHOW. The only thing to be careful with is making sure that the device can operate correctly at the 3.45v logic level that the SHOW uses. You can even connect SPI and I2C devices at the same time, though you will need to dedicate an unused pin (A3 or D2) as a slave select line for each SPI device to avoid conflicts with the TFT controller.

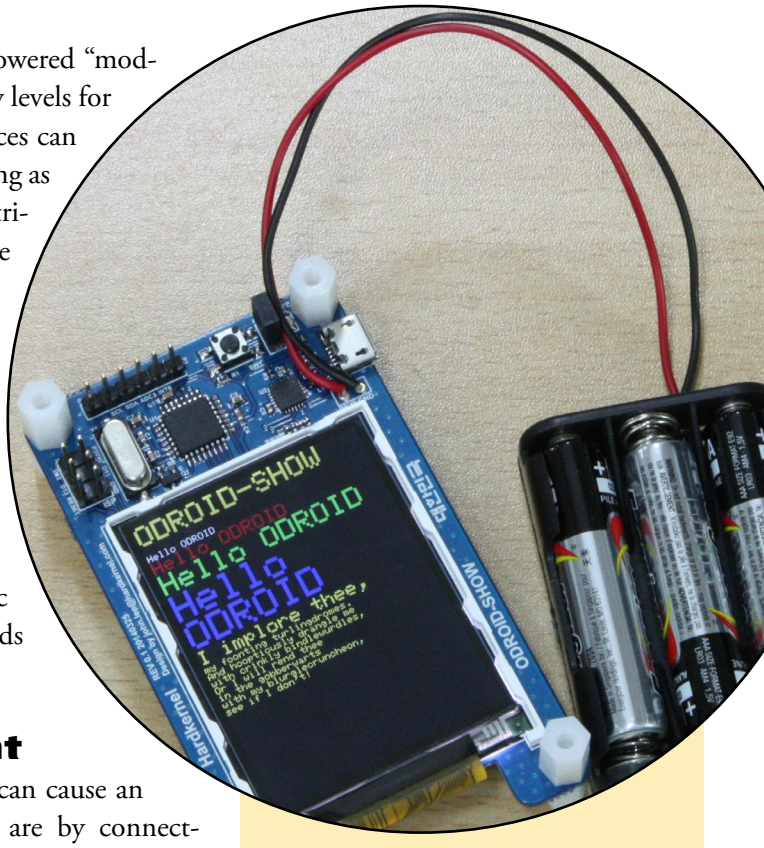
Most hardware modules designed for connecting with an Arduino or similar system will have libraries and demo code available for it, so using them in your SHOW program will often be as simple as including the correct library and modifying the demo to make it do what you need. The libraries will usually hide all the details of the actual protocols used for communication. This makes it a cinch to use most modules in projects of your own design, since you can concentrate on the bigger picture of what it's supposed to do and let the libraries handle the boring bits.

## Basic electrical safety

Due to the tiny currents and voltages used by the ODROID-SHOW, there's practically zero chance of accidentally giving yourself a shock when connecting passive components to the board. However, the sensitive electronics components in the SHOW are a different matter, and it's quite possible damage the board if you wire something up incorrectly. There are essentially three ways to damage the electronics in the SHOW:

*** create an over-voltage situation**
*** create an over-current situation (short circuit)**
*** connect an external power supply with the wrong polarity**

Over-voltage situations arise from connecting an externally-powered device that supplies more than 3.45 volts to any of the SHOW's pins. This includes other Arduino boards, which generally operate at 5v, or what's known as "TTL" levels, and some externally powered "modules" which may use 5v levels for signalling. These devices can still be connected as long as the 3.45v pins are electrically isolated from the higher voltage, such as by using an appropriate voltage divider, level converter or opto-isolator. Static electricity can also cause damage to some components, so use appropriate anti-static or grounding methods when handling them.




**The ODROID-SHOW LCD Panel with optional battery pack**

## Over-current

The two ways you can cause an over-current situation are by connecting devices that draw too much current from the pins or by forming a short-circuit. The pins on a standard Arduino (which operates at 5V) are rated for a maximum of 40mA on any individual pin, with a maximum draw of 200mA. The SHOW, however, operates at 3.45V, so the maximum current is less. I wasn't able to find exact details on this, so to be safe I would recommend not exceeding 20mA on any one pin, or 100mA overall. In fact, since the SHOW is also driving a TFT screen containing an LED backlight, it may be that even 100mA is optimistic. Some devices, like LEDs, could draw too much current if connected directly and need a limiting resistor, while others, like motors or other inductive loads should never be driven directly.

## Short circuit

A short circuit is formed when you create a path from one of your +3.45v pins directly to ground. Besides the obvious short-circuit where you accidentally connect your power pin directly to ground, it's also possible to form a short with GPIO pins that have been configured as OUTPUT. An output pin that's high counts as a +3.45v supply, and one that's low counts as GND, so there's a risk of short circuit if you wire them up to a ground or +3.45v pin respectively. Pins configured as INPUT, however, are safe to wire directly to either your positive pins or ground.

The simple fix for over-current problems and short-circuits, assuming they aren't an oversight, is to put a current-limiting resistor in series with any path that has a risk of drawing too much current or forming a short circuit. Using Ohm's law, a value of 220 Ohms, for example, would limit the current to 3.45v/220R = 15.7mA, which is safely below the maximum current of 20mA.

However, as long as you recognise the hazards and check your circuit before connecting anything, you are unlikely to cause any damage.

**Be safe when working with electricity!**

# HEADLESS 10-NODE ODROID-U3 CLUSTER

## THE ULTIMATE AFFORDABLE HOME SUPERCOMPUTER

*by Tynan Overstreet*

It has always been a dream of mine to own a supercomputer. Accordingly, I entered a field where powerful compute clusters abound: algorithmic trading. In the "algo" world, it is not uncommon for companies to spend six to seven figures on their compute clusters with each node costing thousands of dollars and consuming copious quantities of electricity.

Fortunately for those of us without piles of cash, the ARM-based ODROID U3 offers a compelling chance to build a compute cluster at a fraction of the cost of a traditional x86-based solution. In my first step towards this goal, I built a prototype 10-node cluster out of U3's, and will now subject it to a series of performance tests against one of my current x86 based nodes:

**Backtesting a new strategy idea**
**Real-time signal filtering**
**Generating Random Walks**
**The Challenge**

This article will cover the result of the first test: backtesting a new strategy idea using Response Surface Methodology (RSM), a powerful technique for estimating the shape of hard to evaluate functions. In this case, it is a 4-dimensional hyper-surface. Each point on the surface represents a possible strategy configuration. Three (3) of the dimensions represent configurable parameters in the strategy; think of them as knobs or dials that control the strategy's behavior. The final dimension represents the historical profitability of that particular configuration.

**Every node in Tynan's cluster is a powerful U3 with its own cooling fan**

The hyper-surface can be visualized as a 3-dimensional cubic swimming pool, where we measure the temperature at each point, recording the x,y,z position as well. The warmer the temperature we measure, the more money the strategy makes. We are thus trying to find the warmest, or most profitable, regions of the pool to swim in. You can assume that warm water is a good thing in this example, and not the result of some unsupervised children in our pool!

Each point-measurement takes some non-trivial amount of time to complete, and we must complete thousands of measurements to map the surface accurately. In this example, we are limited by the time it takes to actually evaluate the strategy logic against historical data and measure the resulting profit/loss from any hypothetical trades.

### The contenders

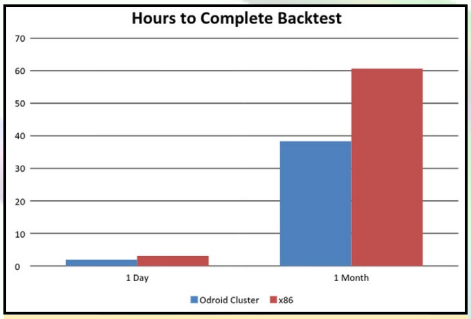
My current node consists of an 8-core 4.2 GHz x86-64 based CPU running Xubuntu with 8GB of memory, a 500GB SSD, dual R9 270x GPU's for OpenCL processing (useful for generating random walks), and an 850W ATX power supply. Not including the rack case or shipping, the node cost about $1150. Similarly, after purchasing 10 x U3's, corresponding 8gb eMMC modules pre-loaded with Linux, power supplies, cooling fans, and a network switch, the ODROID cluster cost about $1250, which minus $100 in shipping puts it right at the cost of each of my x86 nodes. I prefer having some local disk space per node, but if you wanted to forgo the eMMC modules and do a network boot, you could save $250 and build an 10-node ODROID cluster for around $900, excluding shipping.

Additionally, each U3 is overclocked to 1.92GHz by adding the following line to /etc/rc.local right before exit 0:

```
echo 1920000 > \
/sys/devices/system/cpu/cpu0/\
cpufreq/scaling_max_freq
```

## Results

I wanted to see how the ODROID cluster would perform with essentially unmodified code. Other than setting up a python Twisted application to coordinate work across the cluster, the code used for backtesting was the same as my x86 node. As you can see from Figure 1, the ODROID cluster completes a 1-day backtest a full 1 hour faster than the x86 node.




**Speed results of ODROID cluster vs x86**

Backtesting a single day is not very helpful, however, so I performed a 20-day backtest and recorded the elapsed time: the ODROIDs deliver an almost 37% decrease in the time taken to perform the RSM, from running for over 60 hours on the x86 to about a day and a half for the ODROID cluster.

There are also a host of less obvious benefits of running the ODROIDs vs the x86 node. For one thing, the cluster is virtually silent compared to the x86. Once you turn the blue heartbeat LEDs off you barely even notice you have a computer running, let alone 10. Moreover, the U3 cluster draws less power under full-load than just the x86 CPU, not to mention the dual GPU's.

The only downside to the cluster is the added complexity of managing 10 nodes as opposed to a single computer. As such, I am writing a simple cluster management interface that is specific to the needs of a ODROIDs and will release it on www.ODROIDCluster.com when it's ready for public use.

Following are some common pitfalls to avoid when setting up the cluster for the first time:

**The total power needed for this 10-node U3 cluster is less than 70 watts**

First, I had to regenerate the MAC address for each ODROID in order to get the network to recognize each node as a unique device. This was done by deleting the file `/etc/smsc_95xx_addr` and rebooting, which generated a new MAC-Address and allowed me to assign a static IP to the node.

Next, The passive heat sinks needed to be removed in order to add active cooling fans, but they do not give easily. I had to let each U3 watch YouTube for 5-10 minutes prior to prying off the original sinks. This process provided some unneeded digital exfoliation!

Finally, installation of nfs-common did not work until I upgraded the kernel using the ODROID Utility. To access the utility from a headless U3, I had to type sudo ODROID-utility.sh in my remote command line.

## Future improvements

The code used in this speed test was as unmodified as possible in order to assess the differences in hardware in a quick and dirty fashion. Improvements in software design made possible by the increased memory space, for example, were not considered in this test. The ODROID cluster has a total of 20GB of memory vs 8GB on the x86 node, meaning that the software could be refactored to use the hard drive less, which should provide even further speed improvements.

## Going forward

The next test compares the ODROIDs with the x86 in real-time signal processing of incoming market data. The computers will apply Bayesian filters to real-time prices from US futures markets. There is the potential for significant improvements in the number of symbols that the cluster can process simultaneously vs the x86 node. Finally, I will test the ability of each hardware configuration to generate random walks, which is necessary for real-time Monte-Carlo options pricing.

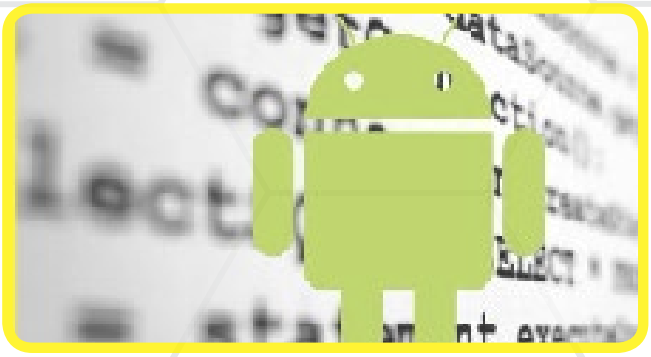
## Conclusion

The ODROID cluster presents a compelling alternative to traditional computing solutions. In a head-to-head speed trial using Response Surface Methodology, the ODROID cluster was 37% more time efficient than my incumbent, x86 based node. Moreover, extra benefits including decreased noise and power consumption make the U3's far superior in this specific application. While operating a cluster adds administrative complexity, the benefits far outweigh the downsides. For up to date information on the ODROID cluster, please visit http://www.ODROIDCluster.com.

# ANDROID DEVELOPMENT

## INSIDE THE ANDROID APK

*by Nanik Tolaram*



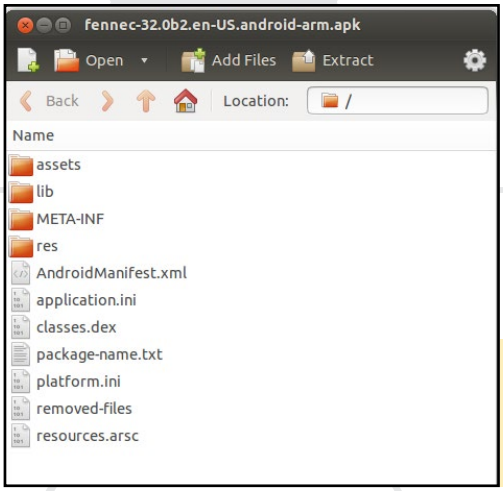
In the last few issues, I gave an overview of the different low level parts of Android, and how they work together to create the Android operating system. In this article, I begin a series of articles on how to develop applications for Android. The best thing about programming on Android is that you can develop and test your applications using any Android device, and it will run (most of the time) on most Android hardware in the wild.

In conventional Android development, you need to test your application with different versions of Android, but you don't need to do this if you are using ODROID hardware. The ODROID family of microcomputers is a great platform on which test your application, since it's able to run different Android versions. However, keep in mind that Hardkernel does not have official support of older versions like Honeycomb, but does publish images for Ice Cream Sandwich 4.0, JellyBean 4.1/4.2, and KitKat 4.4.

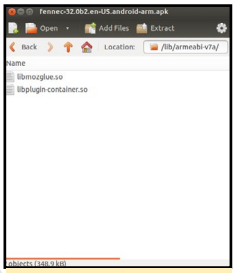
Throughout this article I will be using the open source Fennec (Mozilla) browser application as an example, which is available at http://mzl.la/1lxftFA.

### What is an .apk?

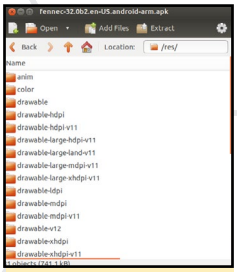
An APK (Android Package File) is a single compressed package that contains all the different files required for your application. If you have developed in Java before, this is similar to a .jar file.



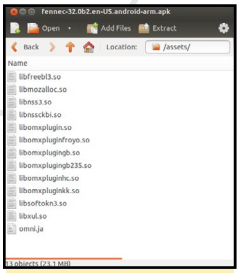

**Folder structure of an APK file**



**/lib**
**Contains native libraries that are architecture dependant**



**/res**
**Contains resources that are used by the application such as images, text, and layout**



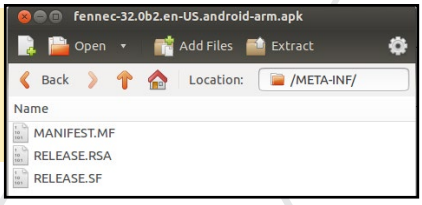
**/assets**
**Any file that will be used by the app can be put here. Normally files inside this folder are read as byte streams and suitable for content like large images, videos and other binary formats**

The following list describes the different folder and files:

**AndroidManifest.xml**
This is the XML configuration file that Android reads in order to find the structure of the app and other properties.

**classes.dex**
This file contains the compiled version of the app's code.

**META-INF**
**Contains MANIFEST.MF and the certificate of the app**




**resources.arsc**
In general, all APKs that run on Android have the same folder structure, which is only slightly different for APKs that are generated as part of a system image. This contains the binary compiled version of resources.

## Java vs Native (C/C++)

Android apps are normally built using Java language, but there are many apps, such as games, that are written in a native language, but are still available for Android. Applications that are ported to Android normally do have some Java code, but it is used as a wrapper layer for the native code. An example of this are many ported games, which run the original version of the console game inside an Android wrapper.

It is recommended to write all Android apps in the Java language, as this will make it more portable across different Android versions, and keeps the maintenance of the code base to a minimum. Using a native app requires that it be recompiled for each version of Android.

## The .dex file

Android APK files have a file called classes.dex which contains your app code in binary compiled format. The format of this file has been defined by Google, and it is not the same as the .class format in the Java world. The dex file is more compact than a normal class file, which is necessary because Android need to run on older devices with limited storage.


**Format of a dex file**

By typing the following command, you will see the information of classes.dex dump into fennec.txt file and will look the snapshot below. If you compare the output with the .dex layout format you can inspect the information available on each layer.

```
<sdk tool>/dx \
--dex --verbose-dump \
--dump-to=fennec.txt \
fennec-32.0b2.en-US.android-arm.apk
```

## AOSP tools

There are a number of APK-related tools inside Android, including aapt (Android Asset Packaging Tool), which is the main tool for packaging Android apps. The included flowchart shows the sequence of steps that need to be done to make an application run on Android.


**Illustration of the APK packaging process**

The compiling and packaging step is normally done by an Interactive Development Environment (IDE), such as Eclipse or Android Studio, which both use aapt internally.

## Dexdump

The Dexdump tool is used to dump the content of your classes.dex file. Shown below is an example of the output:

```
Processing 'classes.dex'...
Opened 'classes.dex', DEX version '035'
Class #0            -
  Class descriptor  : 'Landroid/support/v4/accessibilityser-
vice/AccessibilityServiceInfoCompat$AccessibilityServiceInfo
VersionImpl;'
  Access flags      : 0x0600 (INTERFACE ABSTRACT)
  Superclass        : 'Ljava/lang/Object;'
  Interfaces        -
  Static fields     -
  Instance fields   -
  Direct methods    -
  Virtual methods   -
    #0              : (in Landroid/support/v4/accessibility-
service/AccessibilityServiceInfoCompat$AccessibilityServiceI
nfoVersionImpl;)
      name          : 'getCanRetrieveWindowContent'
      type          : '(Landroid/accessibilityservice/Acces-
sibilityServiceInfo;)Z'
      access        : 0x0401 (PUBLIC ABSTRACT)
      code          : (none)
….
….
….
      access        : 0x0401 (PUBLIC ABSTRACT)
      code          : (none)
   source_file_idx  : 1418 (AccessibilityServiceInfoCompat.
java)
….
….
```

## Compatibility

With hundreds of kinds of Android devices in the world, it's no surprise that some applications work flawlessly on one device but don't work on other devices. There is no single answer or solution to this problem, as it can be caused by incompatible code anywhere from the kernel all the way up to the front-end Android code. Most of the time, vendors modify the Android code to suit what they want to achieve for their devices, which creates more complications.

Most of the time, compatibility issues arise with applications that interact with device's peripherals, since they may have different a behavior when running on a spectrum of hardware. The problem is not with the Android code itself, but with the device driver. Since nearly all of the Android hardware drivers are closed source, there is nothing much that can be done to address the issue except to file a bug with the vendor.

The best approach for a robust application is to test it on as many devices as you can, ii you have access to the devices, or use 3rd party services that test your app on different devices for a fee. Another way, adopted by many software developers, is release the application untested, and let users be the beta testers, giving you access to real life testing environment on a large range of devices. Although this approach is not recommended, in return you can provide the users with free upgrades.

For more detailed information on the Android APK file, please visit http://bit.ly/1A2T0l1, http://bit.ly/1uw6Xqc, and http://bit.ly/1rLAfUK.

# MEET AN ODROIDIAN

## TOBIAS SCHAAF: LINUX NINJA AND ODROID ENTHUSIAST

***edited by Rob Roy***

**Tobias Schaaf, our resident gaming expert, riding a Canada Bushplane**

*Please tell us a little about yourself.*

My name is Tobias Schaaf and i'm from Germany. I'm 31 years old, and will be 32 by the end of August. I'm a sysadmin in a software company which concentrates on software solutions for smart metering, smart home, and smart grid. I've been interested in everything related to computers since early childhood. I love gaming, reading books, watching movies and animes, listening to music and audiobooks. I also like to swim and, of course, everything related to the ODROID.

*How did you get started in computers?*

The first thing that got me hooked was an old Atari 2600 that my parents gave to me as a present in my early childhood. Later, my dad got a Commodore 64, and later, a couple of Amigas with hard drives. At the age of 14, I got my first PC with Windows 95, which I broke within two weeks, and then had my uncle rebuild with a dualboot PC-DOS and Win95 for me. I preferred the DOS way over Windows. Ever since then, I have made sure to constantly upgrade my computer hardware, and normally have about 2-3 PCs actively running besides some older ones that are standing around. Nowadays, there a lot of ODROIDs being added to my PC hardware.

*What is your favorite ODROID?*

Up until now, the X2 was my favorite one, with lots of USB ports, a nice design, and very good quality. It had a switch for the eMMC and SD, with lots of features that I very much liked. That's why it was my first ODROID, although I could have got a cheaper U2 instead! But seeing the new XU3, that's the kind I was looking forward to, so once I get my hands on one of these babies, I might have a new favorite!

**Tobias with a friend in Germany, preparing a barbecue shortly before his most recent trip to the United States**

*Your GameStation Turbo image is very popular on the forums. What other software have you produced for the ODROID?*

I wonder if anyone else produces "software" for the ODROID. Yes, there is the XBMC port, and a couple of different OS versions of Linux (Ubuntu, Debian, Arch, etc.) which many people worked on, but actual software seems to be rarely ported.

I port mostly games and emulators, which now can be found in the Games & Emulators section of the forum, but I also ported a few programs, such as clipgrab, sdl2, or ffmpeg. In addition, I maintain my own kernel builds from Hardkernel as .deb files for installing

and updating. I also created .deb files for XBMC to cleanly install and update xbmc without using the update script provided by @mdrjr, which simply copies the XBMC files over an existing installation.

Most of my work goes directly in the GameStation Turbo image, so I work mostly on new games and updating cores for Retroarch. As far as I know, I have the largest collection of working cores out there for ODROID and always try to improve. I work on stabilizing them and adding new features at the same time. Lately, @AreaScout has done a very good job in helping me with these tasks.

*What type of hardware innovations would you like to see for future Hardkernel boards?*

**1. SATA port(s)**
**2. 1000 Mbit LAN port**
**3. Two 100 MBit LAN ports to have native routing abilities**
**4. Together with the SATA port, a custom ODROID case which allows the addition of hard drives to the ODROID to build a powerful NAS**
**5. A portable ODROID PC like the OpenPandora.**
**6. A U3 underclocked to maybe 1.2 GHz to reduce power consumption together with a nice battery pack, a small keyboard and a simple screen. That would be a nice project!**
**6. Working power and reset buttons. I think they worked rather well on the X2.**
**7. Better cases that allow access to all parts of the ODROID without the need to remove the case (for example, the eMMC, SD and I/O port) or a case that allows a bigger slower turning cooling system, perhaps a 60mm fan with a bigger heatsink.**

**On board the Nargoma, a museum ship that agreed to let Tobias steer for a while**

*What are your future plans for ODROID and your GameStation Turbo Image?*

I have a lot of projects running currently. The GameStation Turbo Image is constantly under development and improvement. I plan to do more updates on the cores and adding a few new ones. I want to exchange the Amiga core from Retroarch for fs-uae (another Amiga emulator) as a standalone emulator, since it comes in a OpenGL ES version which runs much faster than the Retroarch core.

I also plan to include the N64 emulator that @AreaScout and I were working on. There is still room for improvement. I also noticed an issue with PPSSPP under Debian Wheezy, that the system pauses every now and then within emulation. It happens on the X86 version as well, so it's an issue with Debian itself.

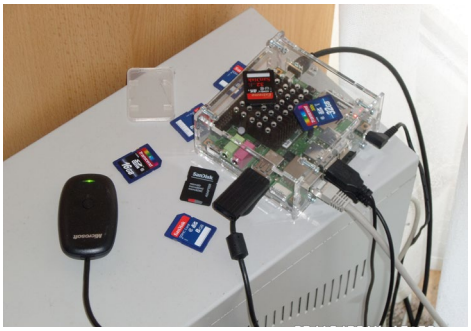
I'm also considering moving to Ubuntu 12.04 or 14.04. But, that might not be necessary. PPSSPP made some improvements in the last few months, and the newest version even supports ARMHF operating systems, which all previous versions did not. They also fixed some graphical issues as well, so I have to check on how this influences my image.

Besides the GameStation Turbo, I'm working on a project to use scripts for installing different kinds of servers on an ODROID, for example, DNS, DHCP, Samba and Active Directory, which will come with an easy to use menu.

I'm also building a "real" Debian repository, which allows you to install and update the software that I ported to the ODROID by using the `apt-get` command. This also includes the newest kernel verion, so you can update the ODROID kernel by simply typing `apt-get dist-upgrade`.


I am always looking around for more awesome games to port to the ODROID. I still have about 50 tabs open of games and programs that I'm considering working on next. There is plenty more to do!

Pictured below is my beloved X2 which I mostly use for gaming since iI-have the XBox 360 controller hooked up



to it, but as you can see by the SD Cards laying around, it's also a base for testing, along with my two U3s.

Also shown is my build slave, which is a U3 hooked up to a 1TB external HDD, where I perform my software magic for my latest ODROID projects.