Linux kernel scripts • Android development • Linux rare gaming gems

ODROID Magazine

Year Two
Issue #19
Jul 2015

Popcorn Time

Watch movies and TV shows instantly with your ODROID

Diet Pi

ODROID's lightest distribution ever

- ODROID-C1 Music Stand
- GPIO Pins C1 Control

What we stand for.

We strive to symbolize the edge of 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 device 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

One of the more popular uses of ODROIDs is for a media center, and Popcorn Time is an all-in-one software package that can stream nearly any type of movie or television show. It runs very well on the ODROID-U3, turning it into a very useful, yet inexpensive, set-top box.

As usual, we feature fun gaming options for the ODROID platform, including creating your own video games for the classic Amstrad computer, playing Millenia: Altered Destinies, and enjoying Nubs' Adventure and Kung Fury for the Android platform. For Android enthusiasts, Nanik continues his Android Development series with a guide to building Android Studio, a Java interactive development environment.

For DIY makers, Ivan introduces his innovative electronic music stand, which he uses as a professional musician to access his sheet music and take notes with a modern touchscreen interface. We also detail accessing the GPIO pins of an ODROID-C1 using a Java library called jOdro, explore a lightweight distribution called DietPi, and learn how to compile an ODROID Linux kernel using automated scripts.

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
Hardkernel manufactures the ODROID family of quad-core development boards and the world's first ARM big.LITTLE single board computer.
For information on submitting articles, contact odroidmagazine@gmail.com, or visit http://bit.ly/lyplmXs.
You can join the growing ODROID community with members from over 135 countries at http://forum.odroid.com.
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`.

Robert Cleere, Editor

I am a hardware and software designer currently living in Huntsville, Alabama. While semi-retired from a career in embedded systems design, including more than a decade working on the Space Shuttle program, I remain active with hardware and software product design work as well as dabbling in audio/video production and still artwork. My programming languages of choice are Java, C, and C++, and I have experience with a wide range of embedded Operating Systems. Currently, my primary projects are marine monitoring and control systems, environmental monitoring, and solar power. I am currently working with several ARM Cortex-class processors, but my ODROID-C1 is far and away the most powerful of the bunch!

Bruno Doiche, Senior Art Editor

Hurry Bruno, we need to package the magazine to send it to our readers! Think of something funny to write around here, quick!
....
....
....
....
....
Got it!
"I don't think of myself as an ugly person, but rather as a beautiful monkey!"
also:
"People say that money is not the key to happiness, but I always figured if you have enough money, you can have a key made."

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.

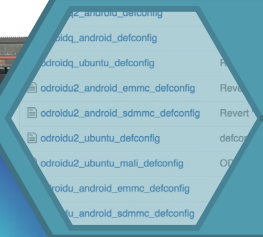
James LeFevour, Art Editor

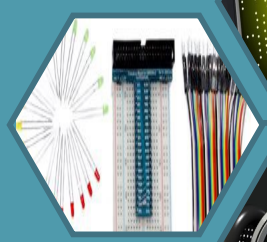
I am a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities, the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored with many aspects that I think most West Coast people take for granted. I live with my lovely wife and our adorable pet rabbit; the latter keeps my books and computer equipment in constant peril, the former consoles me when said peril manifests.

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





MAKING VIDEOGAMES FOR AMSTRAD CPC

HAVE FUN WITH THIS BLAST FROM THE PAST

by Jose Cerrejon

The AMSTRAD CPC is one of those beloved 8-bit relics we will always have room on our hearts

For those who do not know, the AMSTRAD CPC was an 8-bit computer that was popular between 1984 and 1990. CPCtelera is an engine that has been released recently in its first stable version, which facilitates the creation of games for the Amstrad computer using C or assembly code.

The primary developer is Francisco Gallego (@frangallegobr), which is an Informatics engineer, video game developer and professor at the University of Alicante in Spain. For more features of this framework, please refer to the links at the end of this article.

Introducing CPCtelera

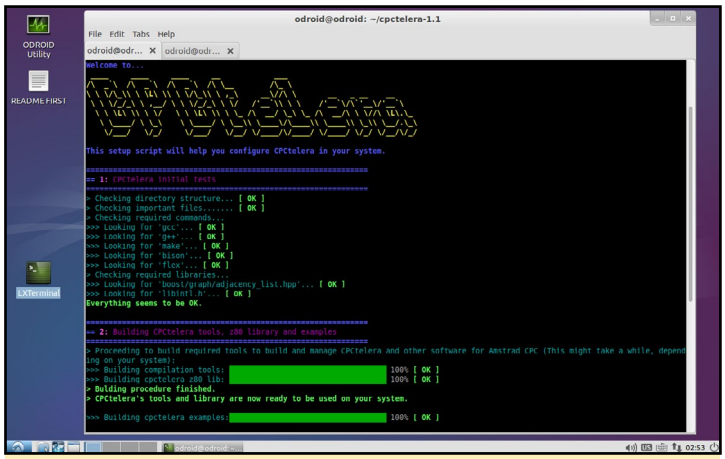
CPCtelera is an integrated development framework for creating Amstrad CPC games and content which includes:

- **A low-level library with support for: graphics, audio, keyboard, firmware, strings, video hardware manipulation and memory management.**
- **An API for developing games and software in C and Assembler**
- **Tools for content authoring (audio, graphics and level editing)**
- **Multi-platform: It works on Windows, Ubuntu, Debian, Arch, and Manjaro operating systems**

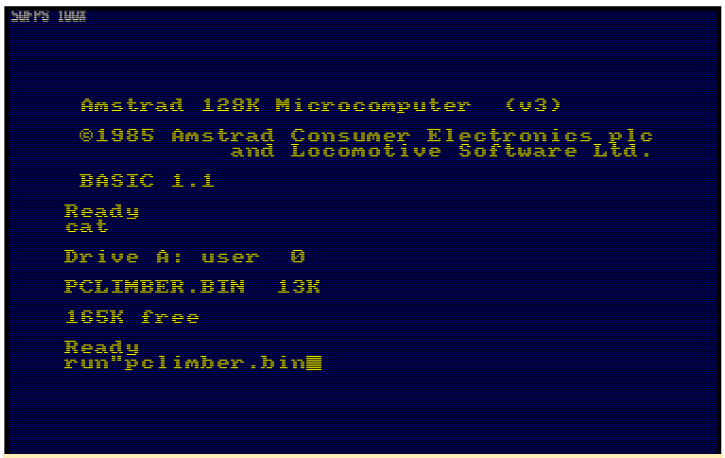
Installing CPCtelera

First, you need to download the source code in order to compile it on your board. To do this, download the stable version like I did, or if you are brave, you can use the latest version from GitHub:

```
$ wget http://bit.ly/1MMdUMA && \
  unzip -nq $(basename $_) && \
  rm $(basename $_) && \
  cd cpctelera-1.1/
# or use the last commit:
$ git clone http://bit.ly/1IPxMOf && \
  cd $(basename $_)
```



Running ./setup.sh after resolving dependencies



Amstrad BASIC, time to dust out a bunch of essential books!

Next, we need to install any missing dependencies, some of which may be already installed:

```
$ sudo apt-get install -y build-essential libboost-
dev flex bison
```

Then, invoke the installation script by running the setup file:

```
$ sudo ./setup.sh.
```

It will warn you about the necessary packages and prepare your system to run the engine. On an ODROID-C1, it will take about 20 minutes to compile.

Starting the Engine

If we browse through the directories, we can see some interesting folders such as docs/, which contains the reference manual, or tools/, which is used to make sprites, compose soundtracks, format converters, and more. Some of these tools are only available for Windows.

To create a new project from Bash, type the following:

```
$ cpct_mkproject [folder_project]
```

Navigate into the directory and you will see two subdirectories:

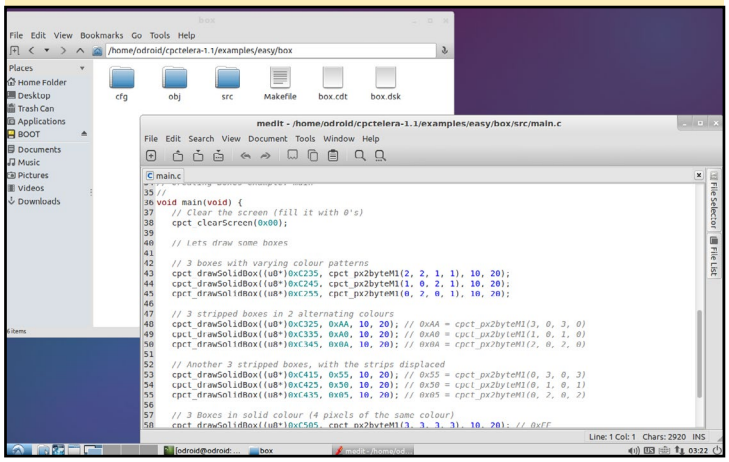
- `src/` with the source code (the first time we just have a main.c). You can create files and directories of your game here.
- `cfg/` contain settings to compile the game through the build_config.mk file.

There are also examples that you can study. Just navigate to the directory, run make, and automatically create .CDT and .DSK files. So cool, isn't it?!

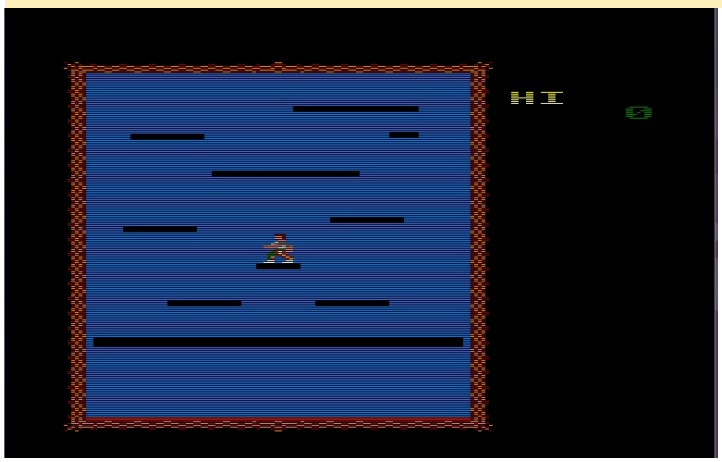
Platform Climber

There is a complete game example included with CPCtelera called Platform Climber. First, you'll need to get an AMSTRAD CPC emulator. I did not find any that were pre-compiled for ODROID, so I had to download and compile it by myself. Don't worry, It's easy:

You have a lot of examples to learn



Scanlines alongside the best nostalgia, get ready to climb!



```
$ wget http://bit.ly/1U2RrjZ
$ unzip caprice*
$ make -f makefile.unix
RELEASE=TRUE
```

In a minute, you will get a binary called cap32. The use is very simple by obtaining any .BAS or .DSK file. To load the game Platform Climber, run the emulator followed with the path of the .DSK file:

```
$ ./cap32 ../cpctelera-1.1/
examples/games/platformClimber/
pclimber.dsk
```

Now you are inside the AMSTRAD! Do you feel the magic? Type cat to see the files inside the previously mounted disk, then run the game:

```
cat
run"game_name.extension
```

Notice that there is no end quote. For example, to run Platform Climber, type the following:

```
run"pclimber.bin
```

Conclusion

Now it's your turn. I recommend you to start studying the examples in `examples/easy/src`, then modify and compile them. Below there is a link to the reference manual with all the info you need to know about functions and methods available. If you know something about SDL, everything will be a little easier.

Happy coding!

More References:

`http://bit.ly/1IPxMOf`
`http://bit.ly/1IpjQvV`
`http://bit.ly/1FMAkrQ`
`http://bit.ly/1IILeAT`

LINUX KERNEL BUILD SCRIPTS

TWEAK YOUR SYSTEM

by Rob Roy

To make Linux kernel compilation easier, I wrote a set of BASH scripts that can be used to download the latest version of any kernel stored in the Hardkernel GitHub branch, compile the source code, and package the completed kernel into a redistributable package. The scripts keep the original kernel on the compilation machine, so that the kernel for any ODROID device may be compiled on a different ODROID device.

The resulting package includes a single-click installation script, and may be shared with others, installed on any compatible ODROID, or added to a repository for distribution via apt-get. The scripts will give prompts for the next step of the process, making it ideal for use by users who are not yet experienced in kernel compilation.

Overview

The build scripts are located at `http://bit.ly/1U6kQcU`, and may be downloaded using wget from the command line or any web browser. Unzip the package after navigating to the download folder:

```
$ wget \
http://bit.ly/1U6kQcU
$ tar -xvzf build.tgz
```

The resulting "build" folder contains the following scripts:

`download.sh` is used to download a particular branch from the Hardkernel repository

`build.sh` is used to launch the kernel compilation process

`install.sh` is copied to the resulting kernel installation package in order to install the kernel on any ODROID device.

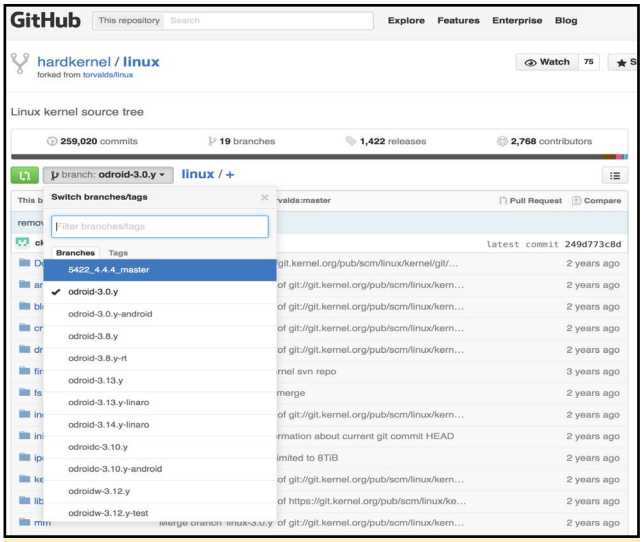
Download script

The script "download.sh" may be used to download any available branch from the Hardkernel GitHub repository by specifying it as the first argument. For instance, to download the most recent version of the odroidc-3.10.y branch for the ODROID-C1, navigate to the "build" folder and type the following:

```
$ sh ./download.sh odroid-3.13.y
```

A list of branches may be obtained by visiting `http://bit.ly/1NvVQa1` and inspecting the "branch" dropdown selection menu, as shown below.

After the download script completes, the "install.sh" and "download.sh" files will automatically be copied to the downloaded branch directory in preparation for the next step.



The place to get the most up-to-date kernel is at the branch dropdown menu on Hardkernel's GitHub repository page

Select your configuration files to have the best build

Build script

After the branch has been downloaded, navigate to the downloaded branch directory, then locate the file in the directory `arch/arm/configs/` corresponding to the ODROID platform that will use the kernel:

```
$ cd odroid-3.13.y
$ ls arch/arm/configs/odroid*
```

For example, when using the odroid-3.13.y branch, configuration files are available for use with the ODROID-Q, ODROID-Q2, ODROID-X, ODROID-X2, and ODROID-U2 as shown in the figure above. Other branches may include configuration files for other ODROID devices, such as the XU, C1 and XU3. When building a kernel for the ODROID-U2/U3 using the odroid-3.13.y branch, the target configuration file would be odroidu2_ubuntu_mali_defconfig, which is supplied as the primary argument for the "build.sh" script. Any changes to the configuration file should be made before launching the build script. I prefer to edit the file using a text editor, but the "make menuconfig" utility may also be used.

The following command launches the build process using the selected configuration file, which requires superuser privileges. Make sure to substitute the name of the target configuration file:

```
$ sudo sh ./build.sh \
  odroidu2_ubuntu_mali_defconfig
```

It may take 10-30 minutes for the kernel to compile, which will result in a folder structure contains the kernel installation assets. This folder structure is contained in a subfolder under the new "release" folder using the name of the configuration file. The build process does not automatically install the kernel locally, but instead creates a portable package which may then be run on the target ODROID device.

To install the kernel on the target machine, copy the "release" directory structure to the ODROID, then navigate to the subdirectory that is named after the selected configuration file. In this example, the installation script may be run by typing the following commands:

```
$ cd release/\
  odroidu2_ubuntu_mali_defconfig
$ sudo sh ./install.sh
```

This process may take 3-10 minutes, after which the new kernel will be ready for use. Reboot the computer to use the updated kernel. A backup of the original kernel is saved as indicated in the output of the installation script, so that the kernel update may be reversed if necessary.

If you have questions, comments, or suggestions regarding the kernel update scripts, please create a new thread on the ODROID forums at `http://forum.odroid.com`.

Branch reference

odroid-3.0.y: Q, U2, U3, X, X2
odroid-3.8.y: U2, U3, X, X2
odroid-3.13.y: XU
odroidc-3.10.y: C1, C1+
odroidw-3.12.y: W
odroidxu3-3.10.y: XU3
odroidxu-3.4.y: XU
odroidxu4-v4.2-rc1: XU3, XU4

NUBS' ADVENTURE

CHALLENGING AND ENJOYABLE 2D PLATFORMER

by Bruno Doiche

Working for ODROID Magazine means that we always access to hundreds of 2D platformer games. After all, we emulate many types of 8 and 16-bit consoles all the time. I often find myself with a recently flashed Android image on my trusty U3 plugged into my lapdock, looking for a brand new adventure to play. Recently, I managed to discover Nubs' Adventure, a very pleasant platformer that took me to amazing interwined worlds in the best 'Metroidvania' flavor. It has great level design, challenging bosses and excellent puzzles to keep you wanting to spend a little more time than you initially expected. Enjoy!

`https://play.google.com/store/apps/details?id=nubs.adventure`

LINUX GAMING

RARE GAMING GEMS PART I

by Tobias Schaaf

This is the original CD-ROM version of Millenia - Altered Destinies

I want to introduce some of my favorite games that probably not everyone has heard of, but are very interesting and fun to play. I hope you enjoy the games that I picked, and I encourage you to try them and play some awesome rare gems of gaming history.

Millennia – Altered Destinies

This game is very unique. You are a freighter pilot on your 6-month trip back to Earth from Jupiter. While you realize it might be a bad sign that you already speak loudly to yourself on the first day of your journey, you are suddenly "abducted" by an alien species. After you have been "abducted", they explain to you that you are in time stasis, and that they are an alien species named the Hood. They are time wardens that need your help in their time and galaxy.

A galaxy called Echelon is currently overrun by a species called Microids. This species is very aggressive, and the time wardens galaxy is about to be overrun as well, and next would be "our" galaxy that's why we are supposed to solve this. You are put in a new space ship which can not only travel through space, but through time as well, and you are being send back 10,000 years ago, into the Echelon galaxy, where only one star system is occupied by the Microids.

The goal is to establish the original four species that lived in the Echelon galaxy, as well as help them to thrive until they are strong enough to withstand the Microids, and therefore save the galaxy. This is also in your interest, not only because you don't want them to attack our galaxy, but the moment that you entered the Echelon galaxy 10,000 years in the past, the technology used to bring you to this place no longer exists, since it was built by the four species that you are supposed to help develop. So what you need to do in order to get back to your own time and galaxy is to help develop the four species until they are able to build the missing parts of the ship again, and help them to defend themselves against the Microids.

The game has very nice features such as full voice acting of your ships computer and AI Agnis, different movie cut scenes of the actions, space combat action, and more. As far as I know, this game is not very well known, but really a rare gaming gem for its unique style and gameplay. It's definitely worth playing and you should definitely give it a try.

Installing and starting

Since this game is a DOS game and not made for Linux, you can't just download a package and run it. I use DOSBox to start the game and configured some options to make it work nicely. First of all, you need the CD or a rip of it, although I would advise to use the CD version for the best experience.

Here is a small step-by-step guide on how to install and start the game based on my ODROID GameStation Turbo image, although it should work the same with the Ubuntu image from HardKernel if you have my all/main and all/testing package lists activated as well. Type the following to install the requirements for DOSBox:

```
$ sudo apt-get install dosbox-
odroid libgl-odroid
```

Configure DOSBox

Start DOSBox once to create the default configuration file, but then exit it right away. Open /home/odroid/.dosbox/dosbox-SVN.conf in a text editor and change the following lines:

```
[sdl]
 fullscreen=true
 fullresolution=1920x1080
 output=opengl

[render]
 frameskip=3

[cpu]
 core=dynamic
 cputype=pentium_slow
 cycles=5000
```

```
cycleup=200
cycledown=200
```

First, I created a folder where I want to place my games:

```
$ mkdir DOS
```

I also copied over the ISO that I created from my Millennia – Altered Destinies game, and placed them into a folder called CDs on my ODROID as well. To make things easier, I added the following lines to the end of the DOSBox configuration file, so I don't need to type them every time I want to play the game:

```
[autoexec]
 mount c: /home/odroid/DOS
 c:
 imgmount d: /home/odroid/CDs/
Mil.iso -t iso
```

Now the system is completely prepared and can launch the emulator. The folder DOS will be automatically mounted as my drive C:, and the CD will mounted as D: as a CD-ROM drive. Install the game as usual under DOS and start it. The intro can be slightly laggy at some scenes, but generally with the settings above you should be able to play the game in full speed.

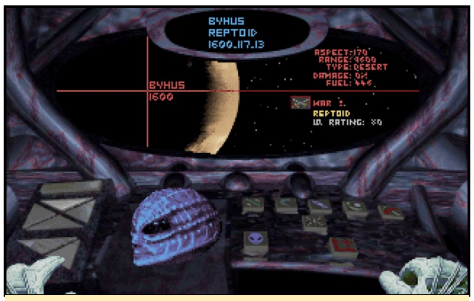
To play the game, launch DOSBox again with the following command, so that DOSbox uses glshim (libgl-odroid) in order to run with OpenGL acceleration.

```
$ LD_LIBRARY_PATH=/usr/local/lib
dosbox
```

The game begins

After the introduction, you find yourself in a distant galaxy a long, long time ago (sounds familiar, doesn't it?). The game does not come with a tutorial, which means that you are pretty much thrown into action without knowing what you must or can do. Therefore, I'm going to explain the basics of the game and hopefully, this will help you understand it.

You have to use your ship's abilities in order to travel through time and space so that you can fix things for the four different species that you should watch over, and help them evolve into an advanced civilization that can help you leave the galaxy, as well as defend themselves against the evil Microids. For this, you have to understand and control your ship in order to complete all of the tasks ahead of you.



The main view of your space ship: from here you start your work

The ship

Inside your ship, tou can see Agnis, the ship's computer and AI, which helps you in your task. He acts as a translator when you talk to the different species. He can give advice and comment on your actions. In fact, his "I am at your service, human" was the starting sound of my Windows machine for a long time.

Most of the ship screens look alike. You have a navigation panel on your left side, which you can control by pressing and holding the right mouse button and moving over the different buttons. A left-click selects the system you want to visit, and a type of elevator moves your cockpit to the selected station. On the right side, you have the action buttons of your current station. Simply move the mouse to the action button you want, and press the left mouse button to activate the action.

In the main view, you can see information about the current planet you are visiting. The name of the planet and the year is shown at the scanner's target cross. On the right side, you can find information about the planet and its inhabitants. Here we can see the race (Raptoids), their current IQ rating (80), and we see a symbol of the current event (war). At the top of the screen, we see again a small information panel with the name of the planet, the race and a clock. The year 1600 is the current century, and the 117.13 symbol is actually a game time clock, which has a very odd format. It's counting seconds, but reaches 100 before the number in front of the decimal changes.

The main view is only for starting a trip to the next century, or to a different place in the galaxy. You can start the engines only from this screen. The second thing you can only do on the main screen is space combat. Occasionally, you have to defend yourself and the planet's inhabitants against invading enemies. Since this game messes around with time, this can sometime be rather confusing, since you might actually end up fighting against your own alter-ego from a different timeline, but you also might battle Microids that try to invade the space where your race settles. The Hoods themselves often try to attack the race that you seeded from a different timeline, and one of the other races that you seeded might even try to invade the planet while they attempt to expand their territory. Therefore, your ship is equipped with some weapons, and can even be upgraded through some of the inventions that you pick up from the races that you have seeded.

In order to fight enemies, you press

A unique twist is fighting off your alter-ego in Millennia

and hold the right mouse button and move in the direction you want to go. A small red dot shows you the direction of the closest enemy. Hitting the left mouse button will fire at your enemy.

Fighting your alter-ego is rather easy, and it normally escapes after a couple of hits. Fighting the other enemies is somewhat harder, since they often come in larger numbers, and while you fight against one, another one can come up from behind and attack you. The red info panel on top will change into a "back mirror", showing you that an enemy is behind you and attacking you. Also, your ship will get damaged, and if the damage is too big, you will automatically flee from the scene and head to the planet in the center of the universe, which is the only planet where you can't seed any species. So you can't really die.

The main view can be reached through the button pointing to north on the upper left side of your navigation bar. Navigation is probably the most important system you have. If you click the button facing east in the upper navigation panel, you can get to the galaxy map and the navigation system.

Galaxy map of the navigation system, not only for space travel but also for time travel

In the navigation system, you gather all the information that you need in order to plan your next step. The galaxy map shows you what the galaxy looks like and how the different species are distributed. Since you start 10,000 years in the past, there is only one planet inhabited with Microids, and none with any other species. You have to choose a planet and where you want to seed the four different races. Each one of them requires another habitat, but more on that later.

Here you can set a course through space and time in order to help the species to develop themselves. Hint: Traveling through time and space costs fuel, and you can only reload fuel on gas giants. That's why it's best practice to seed your race on a planet that has a gas giant near by as well. There's a button

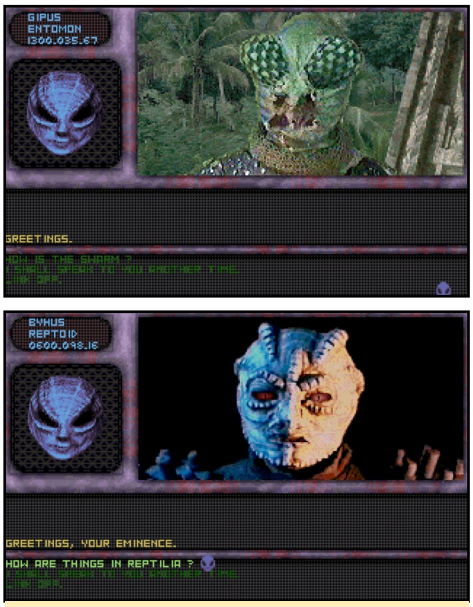
The histograph shows all important events on a races development and is your most important tool

that turns on and off planets that have no gas giant.

In the lower right corner of this screen, you can see the histograph button, which is your best friend. In the histograph, you see all the events that happen, and each step is 100 years in development. New inventions are marked with a green border around the event, and a red border indicates a crisis.

This is where the game gets complicated. You have to help the race though crises, or even prevent crises from happening in the first place. There are often different solutions to a problem with different outcomes, and this is where the game also becomes interesting.

You have different means of interacting with the development of a race, and the most common way is to communicate with a representative of the species. This representative is called an Agent, which is an altered being from the species that you helped to develop. He alone knows of your existence and the greater plan. He knows that you can alter events in time, and therefore



Two of the different species you have to deal with in the game: are the insect-like Entomon and the reptilian-like Reptoids

see what will happen in the future. Still, the Agent is part of the race that you are seeding and will have similar beliefs and needs, which makes them sometimes harder to work with, and each one needs a different form of convincing.

Clicking on the button facing westward in the upper navigation bar brings you to the communication center. Here, you can contact your agents and try to solve their problems by guiding their actions. Agnis will help you translate what your agents are telling you. You actually can see small video flicks of the agents when they talk to you, making it look like a video transmission. With the help of Agnis, you try to solve their issues. This includes topics like politics, where you have to choose who should be the leader of a race, plan assassinations of rebel leaders in order to prevent uprisings, or decide whether to help the rebels and kill the tyrannical emperor.

However, you also have to give advice on how they should protect themselves from a harsh winter, how to prevent wrong decisions that can later lead to catastrophes, or help them come up with new inventions to defend themselves against aggressors. There are so many different topics that you need to address,

and this is the interesting part of the game. Decisions that you make may have tiny effects at first, but also may lead to a big impact thousand years in the future.

Should a minor mutation be eradicated before it pollutes the "pure" society, should it be ignored and later be cured, or does this minor mutation turn out to be important in the future? Should you choose a strong leader, a dictator, that will push development of people through war and domination, or should you choose to be the pacifist, that wants peace and cooperation with others? Will the dictator cause stagnation once he's satisfied and not see room for changes, or will the thousand years of peace with the pacifist lead to a weak race that simply does not want to venture out into space to conquer new worlds? How can you prevent a war between two sentient races on the planet? Or should you rather encourage it to speed development, even if it's through a military arms race? Is it better to fight a climate change with big fires all over the planet that keep a certain level of heat, or will this permanently pollute the environment? Or should they make small fires and gather many people in one space, even though this might lead to shortage of food, cannibalism, or simply false beliefs? Sometimes you even have to solve some religious issues to keep the species on track.

There are many more things you have to decide, but you also have to deal with the different beliefs and characters of your agents. Some might be very helpful and try to do whatever you say, and others might need more convincing at times, through logic explanations, or through threads or using their beliefs against them. Some might even try to trick you.

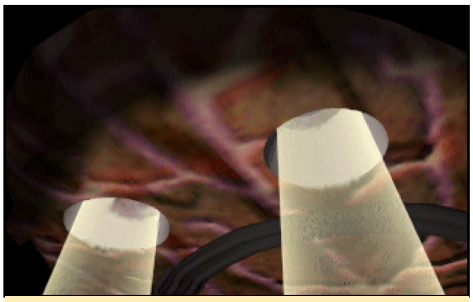
If you are successful in whatever way, a temporal storm will shake your ship, and the history of the species changes. You can then go back to the navigation system and check the histograph again to see how you changed the development of the species. Hint: Since some of the changes you invoked might turn out to be an error a thousand years later, it's best to save the game before every interaction with a species, in case you have to revert what you did.

Another way to interact with the species that you are trying to develop is by using the transporter on your ship. Similar to Star Trek, you can use the transporter to bring objects to your ship and back to the planet. The transporter can be accessed through the southward button in your upper left navigation bar.

First, you click the scan button and watch an animation on how the computer is searching for a temple. This temple is used to interact with you. New inventions will be placed there for you to take away, or if it's empty, you can place an invention there yourself.

First get into orbit, then search for the Temple

Transporting inventions can be useful in different ways. Sometimes an invention that looked like a good idea, such as sonic drillers for mining resources, can cause catastrophes later on like earthquakes, and therefore should be removed from society. But, you can also speed up inventions by extracting an invention, then going back several hundred years and giving them the very same invention, so it takes less time to develop the species. This can be very crucial, since when they hit an IQ of 300, you can give them the blueprints for the missing part of your ship, which may take thousands of years for them to build. And they have to do it before they are attacked by the Microids. Therefore, speeding up the development is often crucial for your goals.

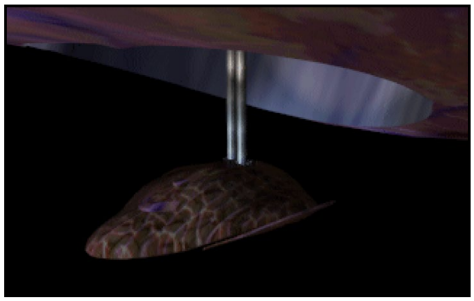


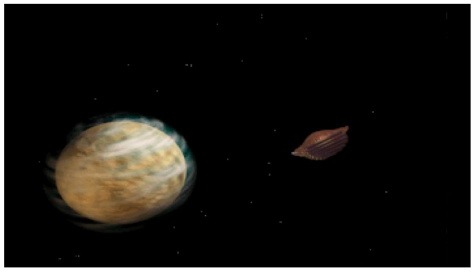
This is how you transport an invention to your ship

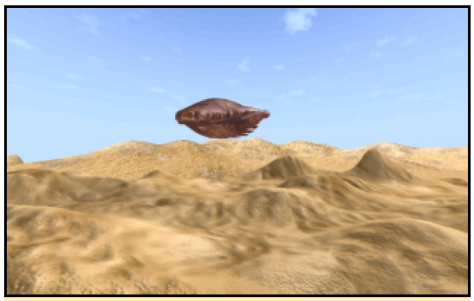
Look at this, we now have a bow and arrows

Hint: If an invention gives your people an IQ of 100, you can extract that invention from its timeline, and can go back to where they had an IQ of 80 and give them the invention a couple of hundred years earlier to speed up their development. Sometimes talking isn't enough to solve an issue on a planet, and you can't stop a dispute between different factions and leaders by taking an invention away. So you occasionally have to take more drastic measures to stop them from killing each other, or just to make a point, even if it means you have to go down to the planet and blow up a building with the leader of an enemy faction inside. Therefore, you have a small dropship that is able to fly directly to the planet's surface where you can attack a building and blow it

up with the weapons of your dropship. This might be necessary to kill an enemy leader which is unreachable for your agent, or to kill an aggressor of a different species that lives on the planet as well and threatens to kill your people.







Sending the dropship to the planets surface is one of the best render videos in the game, and changes depending on the planet you visit

Sometimes it's even necessary to make a point so that your Agent will actually do as you request.

Hint: Attacking the planet with the dropship is very rare. If you attack any building without "the need" to do so, you will kill the entire species, no matter how unimportant the building. Which means you instantly know if attacking the planet is the right thing to do, or if you have to find a different solution. The dropship is launched from the eastward facing button on the lower left panel of your ship.

The last system of your ship is the refill and damage repair station, which can be reached using the northward facing button on the lower panel on the left side. Jumping through time and space will deplete your fuel supplies, and you have to refill them on a gas giant. Therefore, you do a fly-by where you collect fuel from the gas giant itself. Fuel is used to jump through time and space, but also to repair the ship if it was damaged in combat. There are different sizes of gas giants, which will refill different amounts of fuel on a fuel fly-by. This might even exceed the maximum capacity of fuel you can have in your tanks, and also slightly damage you ship if you collect more than you can load. Also, flying close by a gas giant will damage your heat shields, and depending on the size of the gas giant, this will be more or less dangerous.

An important task is refilling your ship on a gas giant: the bubbling liquid is the amount of fuel in your tanks

Hint: It's very practical to settle your species on a Planet that has its own gas giant. On the galaxy map, you can choose to only see planets that have gas giants. That way, you don't have to fly to another system when you have to refill or repair your ship.

Strategy advice

The most important thing to do is to save often and at different save slots, in case something goes wrong, so that you can try again. I found that the best solution is to seed one species after another and concentrate on finishing the development of one species before going to the next. While you could easily speed them all at once and just jump through time and space to fix their issues, it's much easier to concentrate on a single species, since the different species are very different in culture, and it can be hard to switch your thinking between a peaceful hive mind which only thinks of the evolving of the species to a warrior race, which only thrives through permanent conflict and danger of extinction.

Try to empathize with the beliefs of a species. An aggressive species might not follow your orders if you ask them nicely to do what you want, but if you threaten to kill them all if they don't comply, they might be persuaded. Or with a rather religious species, you might have to think about how to guide their beliefs to achieve your goals, or give them a gentle reminder as to why you sent them to that planet.

If possible, extract inventions and place them earlier into the timeline so that the species will develop faster. Moving a handful of inventions one or two hundred years earlier each time

will still give you a thousand years of earlier development, which might be the thousand years that you need in order to finish the device for exiting the galaxy before the Microids try to attack that species.

Take a close look at all events in the histograph, since not all events or inventions seem to be important or have any major influence, but may later cause serious issues. Try to seed the different species as far apart as possible from the Microids, but also from one each other as well. One species might develop spacecrafts earlier than another race, and when they start to expand their territory, they might actually attack the species that you are currently attempting to develop, and you will be forced to fight against your own creation. So, make sure they do not "meet" each other too soon.

Conclusion

I really like this game, and it's one of my all time favorites, although it might not be known to the majority, which is why I call it a rare gaming gem. The game is fascinating, and you have to develop an understanding of political decisions as well as to cope with religious beliefs and other aspects of a society. The different species are very unique, and an approach that works with one species might fail with another. The game is rich in variety, and often captivated me for many hours trying to get a certain species just where I wanted them to be.

The video cut-scenes were very good for the time, and I feel a little pity for the person who had to wear the costumes for each of the species, but it really fits the settings. If you haven't played this game yet, you should really give set aside the time for it, and if you already played it, why not go and give another play through on your ODROID with a giant TV right in your living room?

When you finish that game, you will miss your dropship so much that you will start playing it all over again!

KUNG FURY: STREET RAGE

THE BEST WORST MOVIE EVER MADE ON YOUTUBE IS NOW THE BEST WORST RIPOFF GAME EVER

by Bruno Doiche

If you managed to strand yourself in an island without your ODROID and without access to Youtube, you may have missed the trash phenomenon that was Kung Fury.

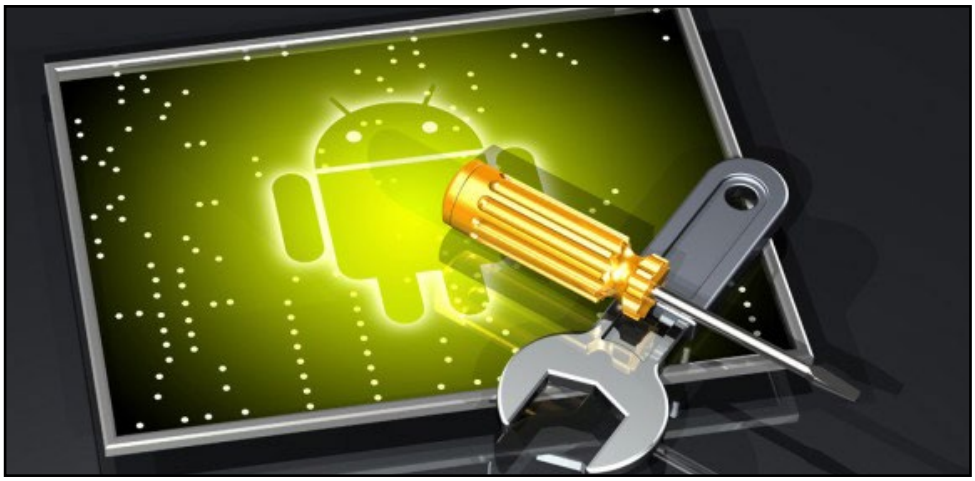
If you still don't know what I'm talking about, go to: `https://www.youtube.com/watch?v=bS5P_LAqiVg`. I'll wait for you to watch and continue to read this. I know! Awesome, right? Imagine if there was a game about this movie!? Well, there is! Go get it!

`https://play.google.com/store/apps/details?id=se.hellothere.kungfurygame&hl=en`

ANDROID DEVELOPMENT

BUILDING ANDROID STUDIO

by Nanik Tolaram



As programmers and developers, we generally use multiple different tools to create applications, and it's not much different with Android. We need tools that will allow us to speed up the code, debug and test cycle, while at the same not to drive us nuts with debugging! Since the beginning of this year, I personally have switched to Android Studio for doing Android development because Google has stop maintaining the ADT (Android Development Toolkit) for Eclipse. The ADT was a plugin for Eclipse that was contributed by Google to allow easy development as it was the de facto IDE (Interactive Development Environment) for Java development. Now anyone that is doing Android development will have to use Studio as their primary tool.

Like any other software, Studio sometimes contains bugs which get fixed in subsequent releases. Because it is an open source project, developers don't have to wait very long for the next release, since as soon as patch or new features are added into the source repository, we can just download and build it locally and start using it. In this article, I will walk through the process of building Studio from source under Linux so you can use it as your day-to-day IDE as well. I'm using Ubuntu 14.04 64-bit, however, you can use any Linux distro or Mac OS for the task.

Prerequisites

You need to have the Oracle JDK installed, which you can download from http://bit.ly/196ebsY. Use the following command to create the relevant symlinks to point the java, javac and the other tools to the correct location of the JDK:

```
$ sudo update-alternatives
--install "/usr/bin/java" "java"
\
  "/home/nanik/Downloads/
jdk1.7.0_79/bin/java" 1071
$ sudo update-alternatives
--install "/usr/bin/javac"
"javac" \
  "/home/nanik/Downloads/
jdk1.7.0_79/bin/javac" 1071
$ sudo update-alternatives
--install "/usr/bin/javaws"
"javaws" \
  "/home/nanik/Downloads/
jdk1.7.0_79/bin/javaws" 1071
$ sudo update-alternatives
--install "/usr/bin/javap"
"javap" \
  "/home/nanik/Downloads/
jdk1.7.0_79/bin/javap" 1071
$ sudo update-alternatives
--install "/usr/bin/javadoc"
"javadoc" \
  "/home/nanik/Downloads/
jdk1.7.0_79/bin/javadoc" 1071
```

The following tools must also be installed. After downloading, extract the Apache Ant and place it in any local folder.

- **git**
- **Apache Ant (downloaded from ant.apache.org)**

Checkout Source

The source code for Studio is hosted in the same place as the Android code at http://bit.ly/1GWeQwC. The step to checkout code is the same as Android, using the repo tool, which may be downloaded using the following command:

```
$ curl https://storage.
googleapis.com/git-repo-
downloads/repo > \
  ~/bin/repo
$ chmod 777 ~/bin/repo
```

Then, use the following commands to checkout the code:

```
$ mkdir studio-1.4-dev
$ cd studio-1.4-dev
```

```
$ repo init -u https://android.googlesource.com/
platform/manifest \
  -b studio-1.4-dev
$ repo sync -j4 --no-clone-bundle
```

Building

Building Studio is straightforward since it uses Ant as its build process, and the snippets of the build.xml are shown below. The build.xml file may be found inside the studio-1.4-dev/tools/idea directory.

```
<!--
 This build script compiles IntelliJ IDEA.  Options
include:
    -Dout=/path/to/out/dir, defaults to ${basedir}/
out
    -Dbuild=123, defaults to SNAPSHOT
    -Dtestpatterns=com.foo.*, defaults to empty
string
    -Dproduct=foo, defaults to studio
 -->
<project name="IntelliJ IDEA Community Edition"
default="all">
  <property name="project.home" value="${basedir}"/>

  <condition property="out.dir" value="${out}"
else="${project.home}/out">
    <isset property="out" />
  </condition>

  <condition property="build.number" value="${build}"
else="SNAPSHOT">
    <isset property="build" />
  </condition>

  <condition property="test.patterns"
value="${testpatterns}"
             else="org.jetbrains.android.*;com.
android.tools.idea.*;com.google.gct.*;com.intellij.
android.*">
    <isset property="testpatterns" />
  </condition>

          …..
          …..
          …..

  <target name="all" depends="cleanup,build,fullupda
ter"/>
</project>
```

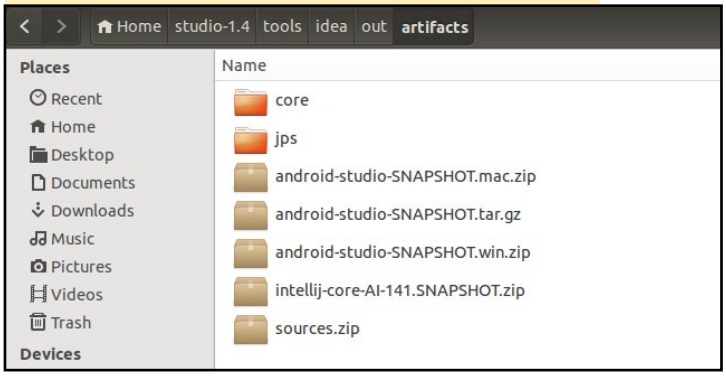
Change the directory to studio-1.4-dev/tools/ideas and make sure the /bin directory of the Apache Ant is included in your PATH environment variable. For example, mine resides in /home/nanik/apache-ant-1.9/bin. Once you are inside the directory, start the build process by executing the ant command:

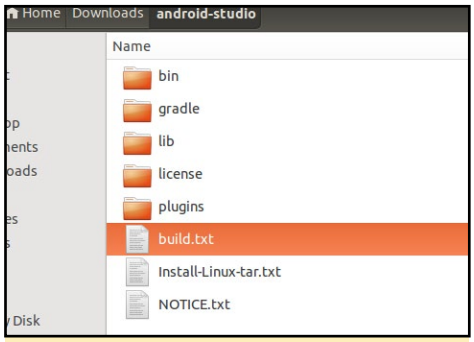
```
$ ant
```

You will see log similar to the one below when you complete the build process.

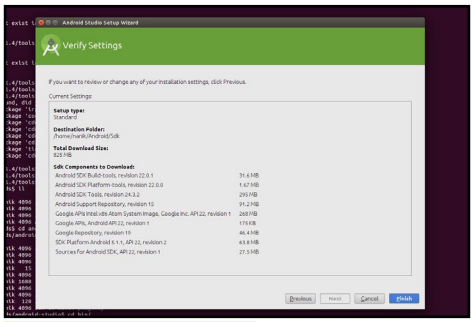
```
Buildfile: /home/nanik/studio-1.4/tools/idea/build.xml

cleanup:

init:
    [mkdir] Created dir: /home/nanik/studio-1.4/
tools/idea/out
    [mkdir] Created dir: /home/nanik/studio-1.4/
tools/idea/out/tmp

build:
     [java] Buildfile: /home/nanik/studio-1.4/tools/
idea/build/gant.xml
     [java]
     [java] doGant:
     [java] 'home' is not defined.  Defaulting to '/
home/nanik/studio-1.4/tools/idea'
     [java] default:
     [java] compile:
          …..
          …..
     [java] Build log (info) will be written to /
home/nanik/studio-1.4/tools/idea/out/tmp/system/
build-log/build.log
     [java] Loaded project /home/nanik/studio-1.4/
tools/idea: 264 modules, 80 libraries
     [java]     [mkdir] Created dir: /home/nanik/
studio-1.4/tools/idea/out/dist.win.ce
```

Figure 1 : Android Studio package for 3 platforms

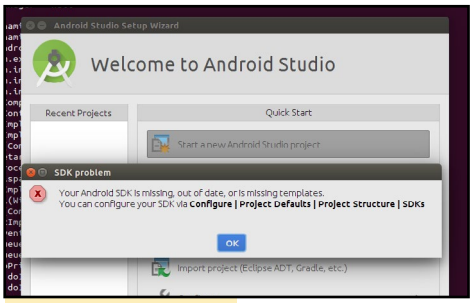




Contents of android-studio-SNAPSHOT.tar.gz



Download the SDK



Setup the SDK

```
     [java]     [mkdir] Created dir: /home/nanik/
studio-1.4/tools/idea/out/dist.all.ce
          ….
          ….
          ….
     [java]       [tar] Building tar: /home/nanik/
studio-1.4/tools/idea/out/artifacts/android-studio-
SNAPSHOT.tar
     [java]      [gzip] Building: /home/nanik/
studio-1.4/tools/idea/out/artifacts/android-studio-
SNAPSHOT.tar.gz
     [java]    [delete] Deleting: /home/nanik/
studio-1.4/tools/idea/out/artifacts/android-studio-
SNAPSHOT.tar
               ….
               ….
     [java]       [jar] Building jar: /home/nanik/
studio-1.4/tools/idea/out/___tmp___/_0/updater.jar
     [java]      [copy] Copying 1 file to /home/nanik/
studio-1.4/tools/idea/out
     [java] ------ default
     [java]
     [java] BUILD SUCCESSFUL
```

Running

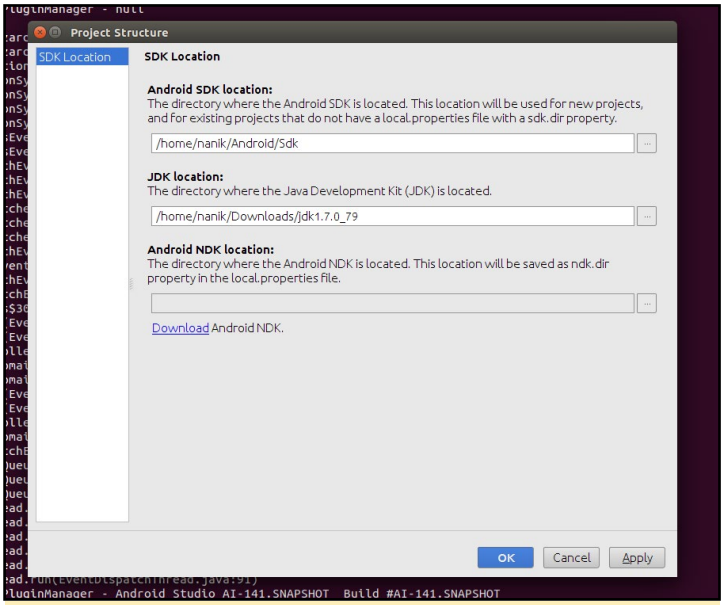
When the build completes successfully, you will get Studio packaged nside /studio-1.4/tools/idea/out/artifacts for 3 different platforms: Windows, Mac and Linux, as shown in Figure 1.

Since I'm using Linux, I extracted the file android-studio-SNAPSHOT.tar.gz, which yielded the contents shown in Figure 2. To run Studio, just run the studio.sh inside the bin/ directory.

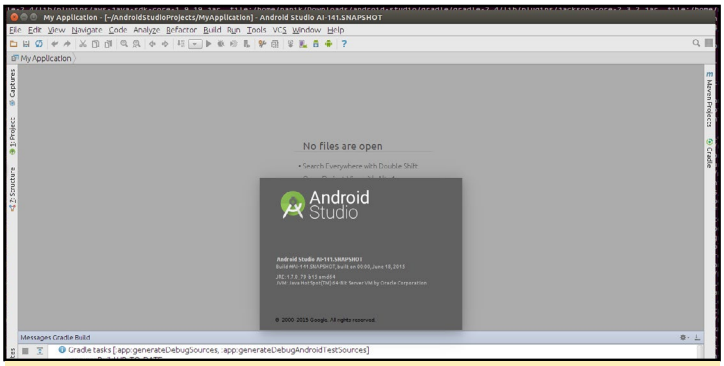
If you have the SDK installed, it will be automatically detected by Studio. Otherwise, you will see a screen that tells you it will need to download the SDK as shown in Figure 3.

Once the download completes, you will be presented with a screen similar to Figure 4. All you have to do is follow the instructions for locating your JDK and SDK when prompted. Complete the screen shown in Figure 5 with the correct location.

Enabling yourself to build the IDE yourself will allow you to always stay up-to-date to the latest changes, which will give you that extra tool or feature to assist you in developing your application much faster.



Setting up the correct location of the SDK and JDK

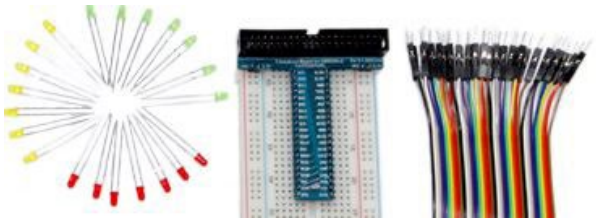


Ready to create your awesome Android application

JODRO

JAVA LIBRARY FOR CONTROLLING THE GPIO PINS OF THE ODROID-C1

by @ChromoDev
edited by Rob Roy



I started writing this Library because I wasn't able to find a Java Library like Pi4j for the ODROID-C1. This project is in development, when you have some ideas for changed or new features please contact me via the link at the end of this article.

Installation

Download the project from http://bit.ly/1RROajs and add the jOdro.jar from the dist folder to your project. Run the following command on your Odroid to give the library the necessary permissions:

```
$ sudo chmod 222 /sys/class/gpio/
export /sys/class/gpio/unexport
```

Usage

At the moment you can set and read a pin. Here is an example project to get you started with using jOdro:

```
public class Tester{
    private static final int delay
= 500;

    GPIOPin led;
    GPIOPin in;

    public void startTest() {
        led = new
GPIOPin(OdroPin.GPIO_24, PinMode.
OUT, PinState.LOW);
        in = new GPIOPin(OdroPin.
GPIO_23, PinMode.IN);

        Runtime.getRuntime().
addShutdownHook(new Thread(() ->
{
            led.shutdown();
            in.shutdown();
        }));

        while(true){
            led.toggle();
            System.out.
println(in.read());
            try {
                Thread.
sleep(delay);
            } catch
(InterruptedException ex) {
                Logger.
getLogger(Main.class.getName()).
log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

First, you have to define a GPIO pin, which is a software representation of a hardware pin. For this pin, you have to define which hardware pin you want to select, for example (such as OdroPin.GPIO_24), in which direction the pin should work (such as PinMode.OUT) and optionally the default value (such as PinState.LOW). Then you can manipulate or read the pin according to Figure 1.

At the end of the program, you have to shut down the pins, which resets them to default (low and input) and unexports them.

GPIOPin class

The GPIOPin class represents the hardware pin in the code.

OdroPin	GPIO #	Funktion	WiringPi
GPIO_00	#74	SDA1	--
GPIO_01	#75	SCL1	--
GPIO_02	#83	--	7
GPIO_03	#88	--	0
GPIO_04	#116	--	2
GPIO_05	#115	--	3
GPIO_06	#107	MOSI	12
GPIO_07	#106	MISO	13
GPIO_08	#105	SCLK	14
GPIO_09	#76	SDA2	--
GPIO_10	#101	--	21
GPIO_11	#100	--	22
GPIO_12	#108	--	23
GPIO_13	#97	--	24
GPIO_14	#113	TXD1	--
GPIO_15	#114	RXD1	--
GPIO_16	#87	--	1
GPIO_17	#104	--	4
GPIO_18	#102	--	5
GPIO_19	#103	--	6
GPIO_20	#117	CE0	10
GPIO_21	#118	--	11
GPIO_22	#77	SCL2	--
GPIO_23	#99	--	26
GPIO_24	#98	--	27

Figure 1 - GPIO mappings

```
// Constructor without default
state (set to LOW)
public GPIOPin(OdroPin pin,
PinMode mode)
```

EXPERIENCE PEACE

WHITE NOISE GENERATOR

by Bruno Doiche

Do you have insensitive coworkers who think that noisy keyboards are fashionable in 2015? Is a roomba robot sweeping your floor automatically while you are trying to manage a couple of spreadsheets? If this is the case, get a hold of a white-noise generator!

```
$ sudo apt-get install sox
```

White noise

```
$ play -n synth 60:00 whitenoise
```

Brown noise

```
$ play -n synth 60:00 brownnoise
```

Pink noise

```
$ play -n synth 60:00 pinknoise
```

Enjoy your own personal space back for an entire hour. If you like it and want to keep it indefinitely, just adjust the 60 minute timer, but mind your eardrums once in a while!


```
// Constructor with default State
public GPIOPin(OdroPin pin,
PinMode mode, PinState state)

// Sets the state of the pin to
low
public void low()

// Sets the state of the pin to
high
public void high()

// Reverses the state of the pin
public void toggle()

// Reads the state of the pin
public PinState read()

// Shuts down the pin
public void shutdown()

// Returns the constant for the
pin
public OdroPin getPin()

// Returns the mode of the pin
public PinMode getMode()
```

PinState

The PinState class represents value of the pin in the code.

```
// Constant for a low value
PinState.LOW

// Constant for a high value
PinState.HIGH

// Returns the value which is
used to control the GPIOs
public string getCode()

// Returns the state state as a
boolean
public boolean toBool()

// Returns the state as an
integer
public int toInt()

// Returns the state as a string
public string toString()
```

PinMode

The PinMode class represents mode of the pin in the code.

```
// Constant for input
PinState.IN

//Constant for output
PinState.OUT

// Returns the value which is
used to control the GPIOs
public string getCode()

// Returns the mode state as a
boolean
public boolean toBool()

// Returns the mode as a int
public int toInt()

// Returns the mode as a String
public String toString()
```

OdroPin

The OdroPin class represents the address of the pin in the code.

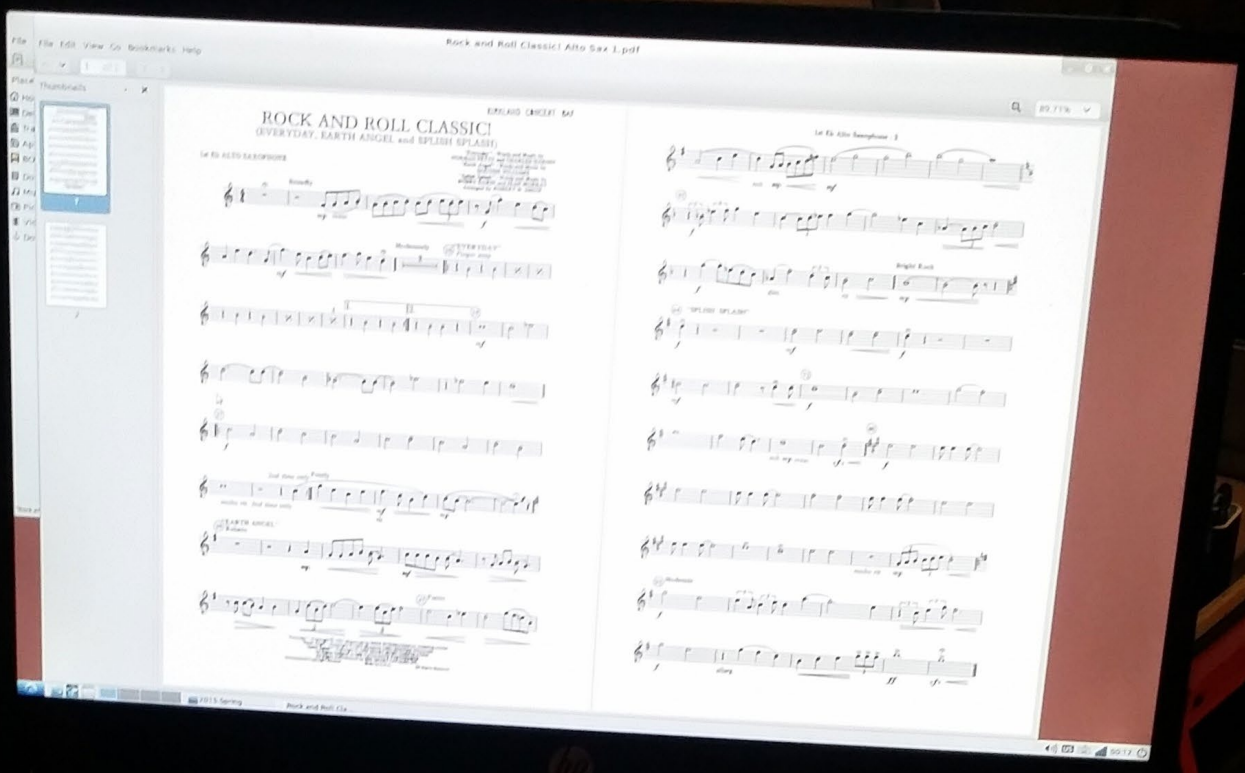
```
// Returns the value which is
used to control the GPIOs
public int getOdroidCode()

// Returns the function of a
pin.  If there is no function, it
returns the number.
public string getLabel()

// Returns the GPIO number in the
WiringPi Protokol
public int getWiringPin()
```

For questions, comments, or suggestions, please visit the jOdro repository on GitHub at `http://bit.ly/1HweC2B`.

ODROID-CI MUSIC STAND

JAMMING WITH STYLE

by Ivan Reede



Being part of a few music bands and an orchestra, I have to carry around many binders of sheet music. I got really tired of toting around all these binders, figuring out which binders I needed for each event, and not forgetting the specific binders required for each specific band practice. To me, paper is a primitive media for sheet music with many attached difficulties. Since I am both an engineer and a musician, I decided to build an electronic music stand.

I first tried using a 10" tablet to replace the paper sheets, but found the screen to be too small. I had a nice app on the tablet that was geared to music playing, but the screen still wasn't large enough. What I wanted was something that would allow me to display two side by side pages, like my music stand. Since I play the saxophone, both of my hands are occupied, and using a touch screen to flip pages was a problem.I tried a foot pedal, but that was just another thing to tote around.

My tablet adventure came to an end when my flimsy music stand was jarred and the tablet went crashing down on a cement floor, cracking the screen. I definitely needed something better. I started by writing a specifications for the music stand. I envisioned using a standard stand, a monitor, a computer and rechargeable batteries, all mechanically fixed together as a single, functional unit.

Stand Specifications

Backplate: 13.5" x 19" back plate with 2" shelf with safe round edges. Holes for a microphone clip holder. Tilt adjustment knobs for customizable viewing angles.

Midpoint clutch adjustment system: Friction locking knob, adjustable height anywhere from 24" (sitting) to 45" (standing)

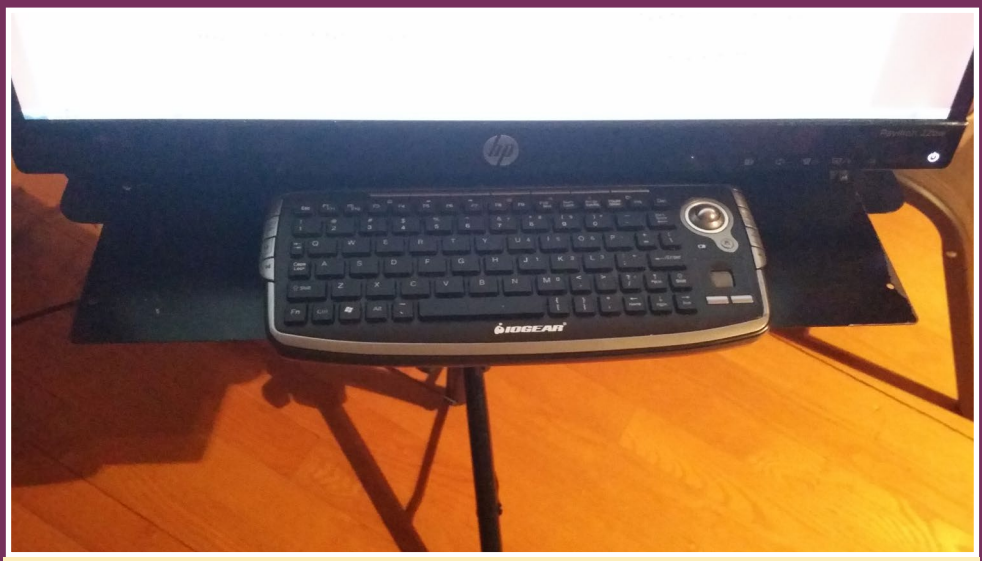
Base: Sturdy tripod with non-slip rubber feet and variable leg spread which fold easily for travel.

Monitor Specifications

I wanted to be able to display at least 2 pages of music at a time, at close to normal 8.5" x 11" size. The monitor had to fit nicely on to the stand and be firmly mounted to the stand. It needed an anti-glare screen to avoid reflection from stage lighting projectors and sunlight. It's image had to be visible outside, in broad daylight, and the monitor had to be able to run on batteries for at least 6 hours between recharges.

Computer Specifications

The computer had to be small and light, with enough storage to hold a large amount of music, with an HDMI output to connect to the monitor and USB inputs to allow easy data transfer, and be able to support WiFi and Bluetooth peripherals. Like the monitor, it also needed



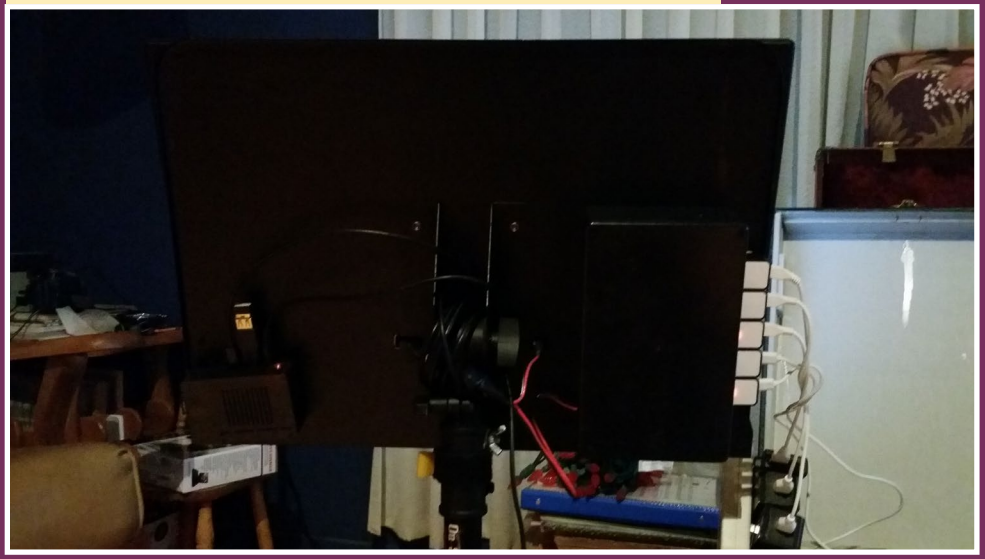
Ivan's choice of control is an awesome integrated keyboard with trackball.

to be able to run on batteries for at least 6 hours between recharges.

I found a 22-inch HMDI monitor with an external 20V DC power supply that was about the same width as the music stand backplate. In order to secure the monitor to the music stand, I removed its pedestal, removed all of its plastic parts, and kept the metal frame inside it. I checked and made sure that the monitor would clip in a sturdy fashion to the pedestal frame without the plastic pieces. With some patience, I drilled the music stand backplate to accept the monitor's base. That was quite easy, as the pedestal frame uses screws to hold the plastic that normally hides the frame, which was one problem solved. I then had a music stand with an LCD screen.

The tablet OS was a limiting factor in practice. Tablet operating systems are oriented toward information consumption rather than information production.

Ivan surely made a great setup and managed the power supply



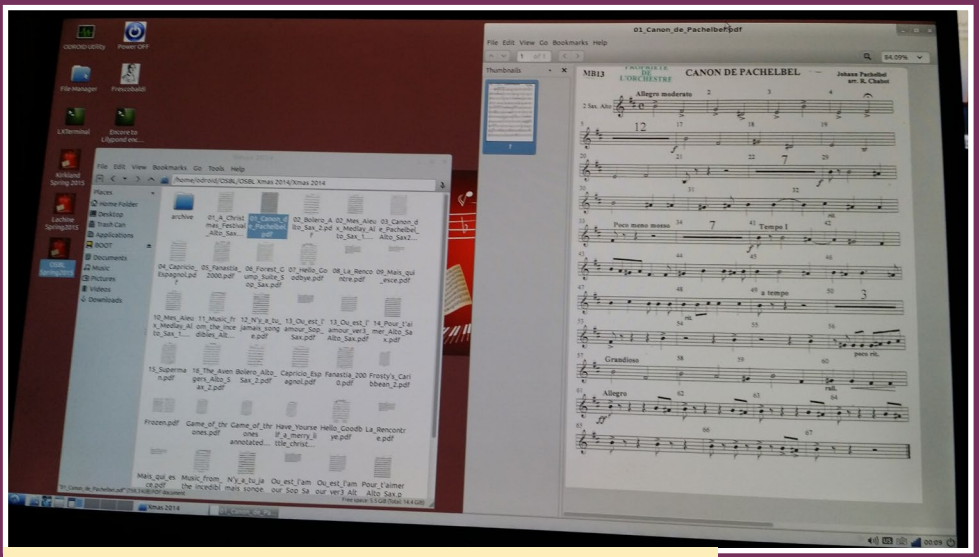
Therefore, I set about finding a suitable replacement. Linux seemed like an interesting OS for this, and I have used Linux for many years now. Mostly, it's open-source, and given some time, you can make it do what you want rather than what the OS people want.

Initially, I tried using a Raspberry Pi computer, which didn't work very well. It ran out of memory quite fast, and, after loading about ten music parts, it would slow down to a crawl. I would take me approximately 20 minutes to load the music sheets I needed for a concert and switching pages could take up to 30 seconds. It was good enough to practice my parts at home, but surely unusable at a concert venue. Still I tried, with my fellow musicians smiling at my slow contraption with tons of wires, power supplies and a really slow computer. They wondered how this could ever be better than the good old paper music sheets.

Then came along a new candidate, the ODROID-C1. It had twice the memory, four CPU cores, about twice the clock rate, and a nice rectangular casing, so I bought three of them and gave them a try. Suddenly, my electronic music stand became much more viable. The 20 minutes need for loading my music parts for a concert dropped down to 45 seconds, which was great. The music stand finally began making practical sense. I indeed could afford 45 seconds of set up time in a show. The pages could be flipped around quite fast, but not fast enough yet.

After a few tries, I finally found a good PDF viewer that was so fast and easy, I could just put all my music in a folder, open that folder and chose my music live from the folder with a simple double click. I then added one desktop icon per concert. I prefixed the file names with a 2 digit number (00, 01, 02) so that my music could be put in playing order, and it was really starting to be fun. Using a mouse, however, proved to be a problem at concerts.

I still had a mobility problem, which was how to power the unit. I had to carry power packs, wires, and more. Running around at each concert venue to find power, string extensions, taping them down to the floor to avoid people tripping on them

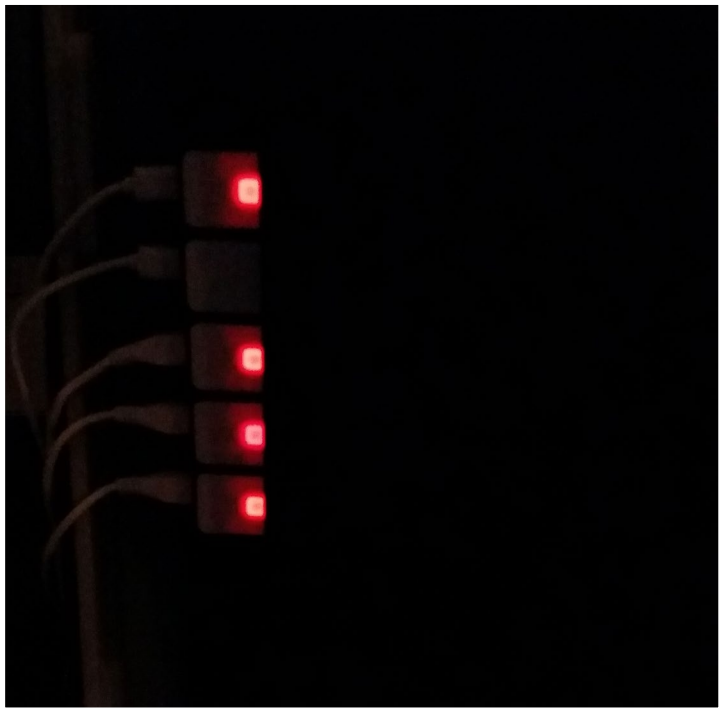


All his partitures at his fingertips in pdf format.

was really inconvenient. I had to have a better solution, so I bought a simple USB battery power pack, 10Ah, 2.1 amp output. Tests showed that the ODROID-C1 would run 19 hours on that. Great! Now for the monitor, which proved to be an unexpected challenge.

The monitor needed 20 volts, so I thought to use four 5 Volt USB batteries in series, but I was wrong. The monitor can pull a quite impressive amount of inrush power when it powers up, when it lights up, and when switching pages. However, when the image is still and nothing changes, the power drain goes down to a very minimal value. The USB power packs turned off at random times while I was playing a part, even though nothing changes on the screen.

And the lighted charging ports are a great styling plus,



The end result was that the USB power packs would go off while I was playing or while I was flipping pages. I discovered that it could shut off either because it wasn't sensing a load at all or, because it was sensing an overload.

The solution was to bypass the USB battery regulating electronics altogether. So, I tore apart the USB power packs and removed the Li-ion batteries. I ganged cells in parallel in order to achieve a battery with the required amp-hour capacity. Connecting 5 of these batteries in series gives 21 volts full charge and 17 volts when discharged. The monitor's internal regulator can work with that. Finally, I made a casing for the batteries and bolted it to the music stand back plate. This gave me a really functional set up with well over 6 hours of autonomy, and no more pesky power cables. Better still, most connections can stay in place, so setting up the music stand is very similar to a normal paper based music stand.

As an emergency backup, I extended the bottom shelf with a clip-on plate, so now, if I want, I can still put paper sheets on the stand. It's also very useful to hold my keyboard. My fellow musicians are starting to find this set up pretty useful. Better still, with a WiFi dongle in the USB port and my cell phone as a hot spot, I can go get any piece of sheet music I need from my home server, even if I don't have it preloaded on the stand. The C1 can house the band's entire music library stored on it with no more printers needed, with no papers flying away in the wind on outside performances. This is really, really cool!

By adding Lilypond and Frescobaldi applications, I can even write music and make corrections to the music on the fly. The music stand can play back music for practice and record sound in performances and practice, which allows me to listen later on and examine where to improve. It also acts as tuning meter.

All in all, thanks to Hardkernel for this little technical marvel! You made my music stand possible and soon, it will be ready to go to production as a commercially available, full size electronic music stand. With my latest experiment, using VNC, an assistant operator can now place sheets and music parts and messages directly on my screen. I can finally concentrate on playing music, rather than flipping pages. The next step is to add a bit of software to the C1 in order to have a wireless network of music stands for band and orchestra.

OS SPOTLIGHT

DIETPI FOR ODROID-C1

by Daniel Knight

What is DietPi?

At its core, DietPi is the "goto image" for a minimal Ubuntu installation:

We've stripped down and removed everything from the official Hardkernel image to provide a bare minimal image that we call DietPi-Core.

With the additions of Ramlog, Dropbear SSH server, and tweaks to reduce memory/cpu usage, the DietPi image comes pre-optimized and ready to run.

Core stats

Automatic filesystem expansion

DietPi will automatically expand your filesystems on the first run. This ensures that you have access to the full capacity of your MicroSD card.

Ramlog

Reduces Filesystem IO and saves SDcard writes by moving /var/log to ram.

Dropbear

Lightweight SSH server installed by default. Can be swapped with OpenSSH-Server by using DietPi-Software if you require SFTP/SCP.

Wifi Support

By using DietPi-Config, you can quickly and easily connect to your Wifi network.

Low Ubuntu memory footprint

< 98MB RAM usage on boot.

Low resources

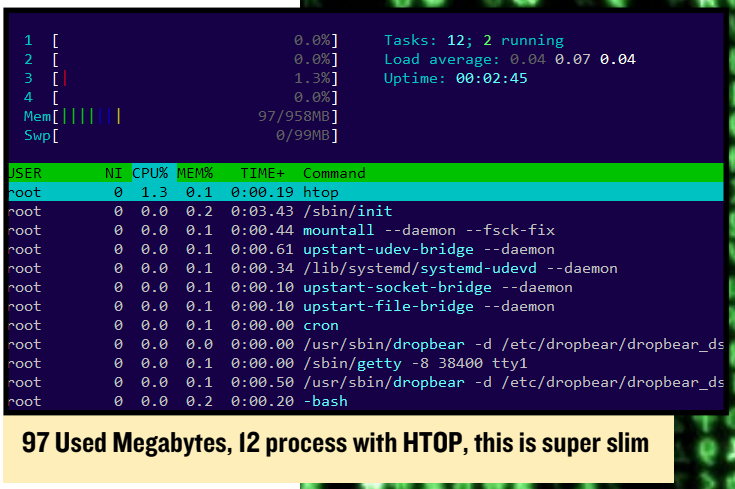
11 total processes on boot.

Swapfile

100mb with swapiness setting 1 (to prevent out of memory errors).

Optional USB dedicated drive

If you plan on using a USB drive with your installation, DietPi will set up your



97 Used Megabytes, 12 process with HTOP, this is super slim

USB drive and automatically configure all future software installed with DietPi-Software to utilize your USB device instead of the MicroSD.

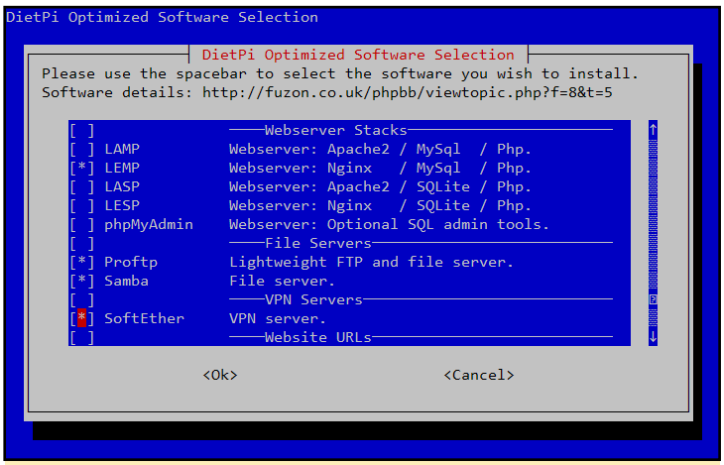
Capabilities

Built from the ground up, DietPi-Software allows for popular, optional install choices. All of which are preconfigured and "ready to run" with all the optimizations and configurations done for you.

If you're looking for a LAMP webserver stack (Nginx/MySql/Php), BitTorrent server and Kodi combo installation, DietPi-Software will install, configure and optimize them all. The optimizations applied include everything from php opcache size, bittorrent server cache size, Nginx/php5-fpm thread counts, and many more.

By automatically applying unique optimizations specific to your hardware, DietPi ensures you get the maximum performance from your ODROID device and the software you choose to install.

The full list of DietPi's software choices can be found online, please goto `http://fuzon.co.uk/phpbb/viewtopic.php?f=8&t=11#p11`



DietPi has its own optimized software library selection

Configuration

From inside DietPi-Config, you can easily change display options, connect to a wifi network, set static IP address, modify CPU governor settings and many more options. With the integration of Samba client, NoIp and CurlFtpFs, you can easily connect to network file shares or give your device a permanent website address with ease. DietPi-Config is a tweaker's paradise.

Setup

Download the DietPi image, setup an optional dedicated USB hard drive, and install the following DietPi optimized software with DietPi-Software:

- **Owncloud - Your own personal backup system**
- **Transmission - BitTorrent server with web interface**
- **Kodi - The pinnacle media center**
- **LAMP webserver - Apache2, MySql, PHP-5 (used by Owncloud)**
- **Samba server - To access your BitTorrent downloads on this device remotely**

Installation

We will also cover the optional setup of NoIp with DietPi-Config and give your ODROID-C1 some lightweight justice. What you'll need:

- **ODROID-C1.**
- **2GB or greater MicroSD card.**
- **Internet Access (Ethernet or Wifi, required to complete the DietPi setup)**
- **A dedicated USB hard drive is recommended for BitTorrent Transmission server and Owncloud installations. DietPi will automatically move your data to the USB hard drive if installed.**

The online guides and documentations are available here:

```
http://fuzon.co.uk/phpbb/viewtopic.php?f=8&t=9#p9
```

Download DietPi for ODROID-C1 at the following link:

```
http://goo.gl/UF6I0f
```

Write the image to your MicroSD card:

- **Unzip/extract the DietODROID.7z image.**
- **Write the DietODROID_vxx.img image file to your MicroSD card.**

The online documentation covers the methods for writing the image with Windows and Linux.

- **Plug the MicroSD card into your ODROID-C1 device and power it on.**
- **DietPi will automatically expand your filesystem and reboot twice when completed.**
- **When the login screen appears, enter username 'root' and password 'raspberry'.**

DietPi also comes preinstalled with a lightweight SSH server (Dropbear). Simply use the IP address of your ODROID device and the login details above.

DietPi will now check for updates. If updates are applied, a system reboot prompt will appear, press enter. When the login screen reappears, log back in.

USB drive

DietPi will prompt you to answer questions regarding your installation, so press enter to continue.

- **The USB dedicated hard drive screen will now appear. If you have a USB hard drive available, select USB Install, press enter, and follow the onscreen instructions.**

If your USB hard drive is already formatted with ext4 or NTFS, you will be given the option to keep the existing data or format to ext4.

Software

From the main menu, select the DietPi Optimized Software option and press enter.

• Use the spacebar to select Kodi, Transmission, Owncloud. Then press enter.
• You do not need to select LAMP, as DietPi will automatically install LAMP for Owncloud.
• When the "File Server Recommended" prompt appears, press enter.
• When the "Boot Options" prompt appears, select Yes and press enter. From here, you can choose which software will automatically start on bootup. Select Kodi from the list and press enter. When you're done, press ESC to return to the main menu.

Selecting a file server

From the main menu, select the File Server option and press enter.

• Select Samba from the list of available file servers and press enter. Confirm when the prompt appears to return back to the main menu.

Both SSH Servers and File Servers can be changed easily at any time by simply running dietpi-software from the terminal. DietPi will automatically install your new choice and remove your previous choice.

Starting the installation

When you are ready to install your selections:

Simply select Go Start Install from the main menu and press enter.

DietPi will now begin the installation process and automatically install, configure, and optimize your choices.

Once DietPi has finished installing your installation choices the system will reboot. This completes the installation of your software.

Using the installed software

Obtaining your IP address:

In this guide, we are using the IP address of 192.168.0.100. This will need to be replaced with the IP address of your ODROID device.

You can obtain your IP address by running dietpi-config. Select the networking options menu, then select either ethernet or wifi.

Using Kodi:

As we selected Kodi for the autoboot option with DietPi, this will load automatically. If you didn't select Kodi to boot from startup, you can run Kodi by typing startkodi. You can also change the autoboot choice by running dietpi-config from the terminal and selecting the AutoBoot option.

Accessing mounts / USB drive:

All of DietPi's mounts can be found in the root filesystem under the folder /mnt/. If you wish to browse your USB drive, simply browse to /mnt/usb_1.

Using Transmission (BitTorrent):

url = `http://192.168.0.100:9091`
username = `root`
password = `raspberry`

Access downloaded data:

Since we installed the Samba server, we can access the downloads remotely. From a Windows based OS, simply hold the Windows key and press R, then enter the address below.

address = `\\192.168.0.100\dietpi or \\dietpi\dietpi`
username = `root`
password = `raspberry`
directory = `downloads`

Using Owncloud:

Access web interface:

url = `http://192.168.0.100/owncloud`

The first time you connect:

Create your admin account by typing in a new username and password.
Click Storage & Database to expand the submenu.
Database type = Select MySQL
DataFolder = Change to `/var/lib/owncloud/data`
Database User = `root`
Database Password = `raspberry`
Database Name = `owncloud`
Click Finish Setup to complete the Owncloud setup.

Using LAMP Webserver:

Access website:

url = `http://192.168.0.100`
local directory = `/var/www`

Access phpinfo:

url = `http://192.168.0.100/phpinfo.php`
Access PHP cache info:
url = `http://192.168.0.100/apc.php`

MySql Details:

username = `root`
password = `raspberry`

Installing NoIp:

Using NoIp will allow you to point a web url address (eg: http://MyWebsite.noip.biz) to your ODROID device, regardless of your Internet IP address.

Registering for NoIp:

Create your free NoIp account by going to https://www.noip.com/sign-up and select a web address for your account.

Activating your account with DietPi:

DietPi-Config is a feature rich configuration tool for your device. One of its main features is the ability to easily setup and install NoIp client, Samba client, and, FTP client.

From the terminal, run dietpi-config
Select the Networking Options menu
Select NoIp from the list and select Install to install it. DietPi will now automatically install NoIp.
When the installation is completed, select NoIp from the menu again.
From here you can enter your NoIp email address and password. Press enter on all the remaining options.

If you entered your details correct, the NoIp current status will change to Online.

Open router ports:

If you want to access your website outside of the local network, you will need to enable port forwarding on your router. This will allow external access to your website. Simply enable TCP port 80 and point it to your ODROID device.

Figure I - "DietPi - Figure I - DietOdroid_htop.png"

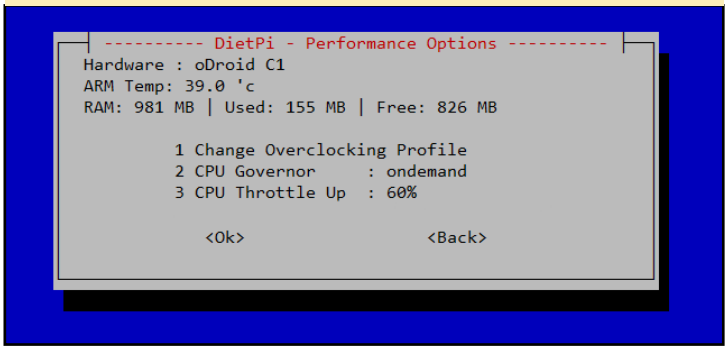
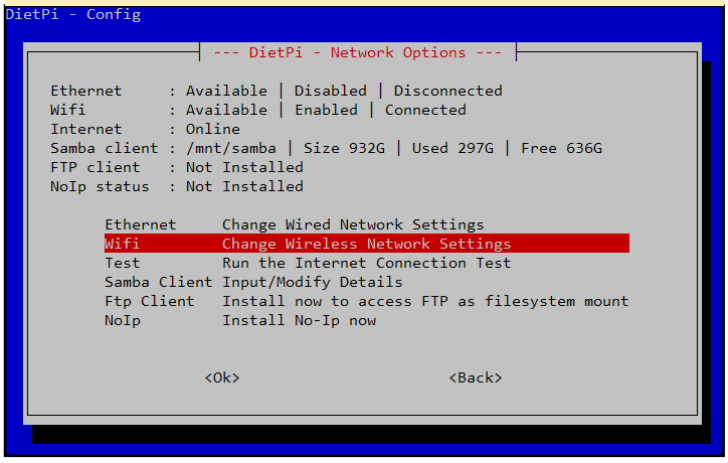


Figure I - "DietPi - Figure I - DietOdroid_htop.png"



POPCORN TIME

A MEDIA LOVER'S DREAM

by László Leonard

Popcorn TIme not only have a cute logo, but also is a revolutionary way to get your media

The original Popcorn Time application was a multi-platform, open-source BitTorrent client which included an integrated media player. The program, and its forks of the same name are free alternatives to subscription-based video streaming services such as Netflix. Popcorn Time uses sequential downloading to play copies of films listed by the website yts.to (earlier yify-torrent.com & yts.re), also known as YIFY (although other trackers can be added and used manually).

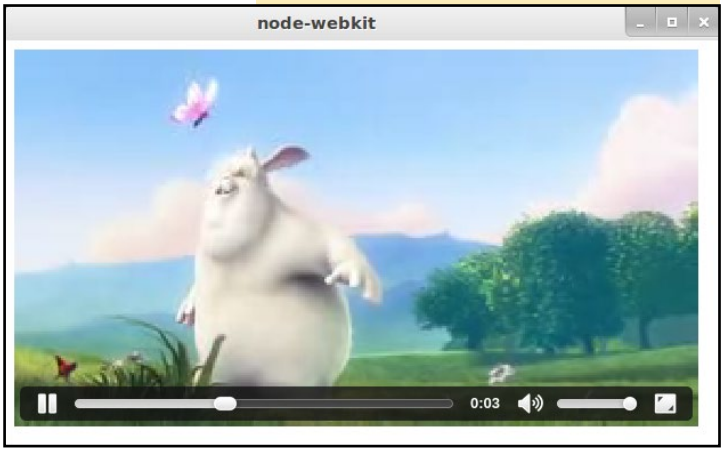
Following its inception, Popcorn Time quickly received positive media attention, with some comparing the application to Netflix due to its ease of use. After this increase in popularity, the program was abruptly taken down by its original developers on March 14, 2014 due to pressure from the MPAA. Since then, Popcorn Time has been forked by several other development teams to maintain the program and produce new features. One of this forks is available at https://popcorntime.io/. The officially supported platforms are Mac, Windows, Linux (32 and 64 bit) and Android.

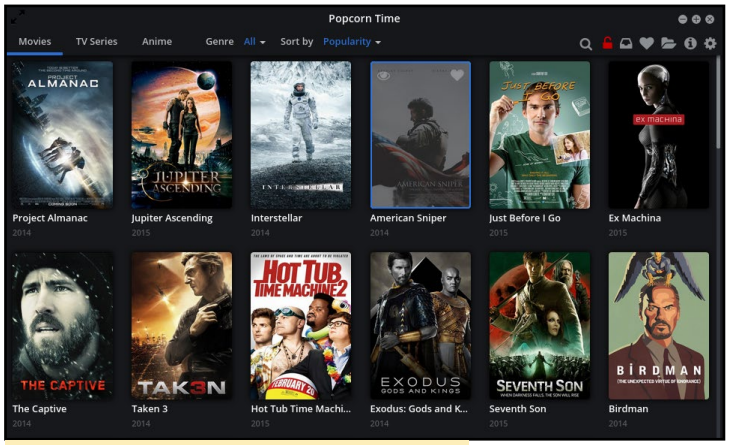
Thanks to its modern and easy to use graphical user interface, the application became very popular around the world. Newer versions of Popcorn Time are able to download and play content provided by other trackers, and are also also to use the media players installed on the system for playing the content. You can also set the application to keep the deleted files for watching later, even with other media players.

Being an ODROID-U3 owner when I found out about this application, the first thing I did was check to see whether it was available on armhf platform. On the Popcorn Time and ODROID forums however, I found that Popcorn Time is not supported on armhf Linux devices. After some small researching on the Internet, I figured out that theoretically there is no reason why this application could not run on my ODROID-U3. All I needed was to find the nw.js (aka node-webkit) application runtime binaries for armhf. This task was quite challenging because none of the binaries I found were able to run hardware accelerated WebGL (or in other words to use OpenGL ES 2.0), demo applications, nor decode video and audio files. So I tried to build my own node-webkit framework binaries based on tutorials that I found on different forums, but each attempt failed because some dependencies were not satisfied, or some error occurred during the build process.

NW.js is an application runtime based on Chromium and node.js. You can write native applications in HTML and JavaScript with NW.js. It also lets you call Node.js modules directly from the DOM and enables a new way of writing native applications with all Web technologies. It was created in the Intel Open Source Technology Center.

Playing a video using HTML 5 with NW.js





Popcorn Time's front page

Since NW.js is Chromium based and my Chromium browser is able to run hardware accelerated WebGL, I thought that the way to get the framework to use OpenGL ES 2.0 instead of OpenGL 2.0 on NW.js would be similar to Chromium. So I ran my demo applications with the --use-gl=egl flag and the magic happened. Suddenly my application was running with hardware acceleration. I was able to run some games developed with WebGL on my ODROID.

Although my framework was running in hardware accelerated mode, I was not able to decode video and audio files so I kept pursuing my research. As one of the Popcorn Time developers pointed out, the problem was in the libffmpegsumo.so file which didn't have most of the codecs implemented. Since my Chromium browser was able to decode a lot of videos, I tried to use the libffmpegsumo.so file located in the Chromium installation directory. Beside replacing this file I also had to replace icudtl.dat file too in the node-webkit archive.

This time the node-webkit framework was able to play videos with HTML 5's video tag as the picture below illustrates:

Popcorn Time playing a video

At this point I had the framework needed for running the Popcorn Time application with hardware acceleration, and was able to decode some video and audio files. All I needed to do now was to port the project to the armhf platform. After downloading the project from the Git repositories I tried to build it, but the build failed due to the unknown architecture. To get it to work, I had to add the arm support to the Popcorn Time desktop project and to its dependency Node-webkit-builder project. To do this, fortunately I only had to modify two files in my Git Project.

The result was a working instance of Popcorn Time for my ODROID-U3. Due to the lack of codecs for Chromium, some videos can't be decoded yet, or the decoding is slow and the video lags especially when I play Full HD videos. I hope that updated drivers for the Mali GPU will solve these issues and that the next releases of Chromium will implement more codecs. Beside these issues, I found out that opening torrent files from other trackers did not function correctly, but I am working on this problem. I really hope that this issue will be fixed before you read this article.

Two projects can be found on my Git repository for enthusiasts. One of the projects contains the necessary files and tutorial for installing the application, and the other contains a tutorial for building the project.

These repositories are available at the following locations:

```
https://git.popcorntime.\
io/laslaul/popcorn-time-\
installation-guide-armv7

https://git.popcorntime.io/\
laslaul/popcorn-time-\
building-guide-armv7
```

This build was only tested on an ODROID-U3 running Ubuntu 14.04.2 and I would really appreciate feedback from users, especially from those who are running the application on a different device or operating system version.

Before downloading and installing Popcorn Time you should check your country's policy on proprietary content, or make sure that the content you are downloading or uploading is free and legal. Note that Popcorn Time is frequently referenced as the pirate version of Netflix.

References

Wikipedia, Popcorn Time, http://en.wikipedia.org/wiki/Popcorn_Time (2015. may)

GitHub, Nw.js project, https://github.com/nwjs/nw.js/ (2015. may)

MEET AN ODROIDIAN

CHRIS MCMURROUGH, ROBOTICS EXPERT

edited by Rob Roy

Although Chris is not yet involved with the creation of the Iron Man armor, he surely is the guy among us that is the closest to get it done

Please tell us a little about yourself.

I am a robotic perception engineer, university instructor, and maker from Texas. I have worked on a wide range of robotic platforms (aerial, ground, underwater, aquatic surface, and industrial) and embedded systems (ODROID, Raspberry Pi, Intel Atom, and microcontrollers). Most of my experience comes from research and development, both in academia and industry. I am always interested in the evolving world of robotics, and the computational approaches required to make them do useful work.

My main focus right now is engineering education, particularly at the university level for students who are nearing the start of their professional career. I enjoy teaching practical, multidisciplinary, high-demand skills that students may not necessarily learn in their core classes. I also work on outreach projects in order to get kids interested in engineering at a young age. The best way that I know of to do this is to show students a robot that does something really cool, and then explain all of the mechanical skills that it takes to design a robot platform, the electrical skills that it takes to establish communication and control, and the computer science skills that it takes to add intelligence and decision making capabilities.

How did you get started with computers?

I grew up with Oregon Trail. Most of my friends died of dysentery.

Seriously, all of the computers that I had while growing up were always second-hand machines that someone else was getting rid of. I had to upgrade half of the components just to get them to be usable, and I was always running into problems that I had to solve myself. The first "new" computer that I ever bought was a PC running Windows ME, which was the worst OS ever made. It also had a design flaw that would cause it to overheat and lock-up randomly, so I was forced to find my own workarounds and tweaks to run the original Starcraft. Basically, I learned a lot about computers just to play Starcraft.

What drew you to the ODROID platform?

I do a lot of computer vision and robotic perception work. Basically, I need Linux and a bunch of computationally intensive libraries to do my work. The ODROID platforms really give you the most "bang for the buck" compared to everything else that I am aware of. All embedded Linux boards have problems when you are porting code developed on a desktop or laptop computer, but I have been able to solve any significant problem I have ever encountered by taking advantage of the wealth of information on forums.

Which ODROID is your favorite?

My all-time favorite is the ODROID-U3. It is small, fast, and offers the highest value when compared to everything else that is currently available on the market. I am also partial to the X2, because I had such a good experience working with it as my first ODROID board.

How did you become so proficient in robotics?

The very first robot that I built was a simple maze solver using the original Lego Mindstorms kit in my high school computer science class. This really was my first experience doing any sort of embedded programming, sensor data acquisition, and motor control. After that, I was hooked. When I started my undergraduate degree, I joined a

Chris enjoys sightseeing in the world, and here he is at the Parthenon in Greece

university robotics team and built a simple ground robot with students from other engineering majors. I enjoyed the interdisciplinary and competitive experience that these competitions provided, and to this day I believe that "right" way to learn robotics is to jump in and start building.

Towards the end of my undergraduate studies, I started working in a control systems research laboratory as an intern. I gained valuable experience in control theory and electronic systems and decided to stick around and work on a master's degree. For the next couple of years, I ended up building a flapping wing micro air vehicle as a proof of concept for a theoretical control law that some people much smarter than me invented. This is where I picked up mechanical and electrical engineering skills, which together with computer engineering, sort complete the robotic "triad".

So at this point, I really enjoyed what I was doing and decided that another 4 or so years of this stuff wouldn't be a bad idea. I went right into a PhD after completing my master's degree and continued to work on robotic platforms. Around this time, a close family member of mine was diagnosed with ALS, a neuro-degenerative disease that renders patients unable to move or speak. They were given a medical eye tracking computer, which at the time required a lot of intervention to calibrate and keep in good working order. I learned a lot about the device's shortcomings, and started to develop my own prototypes and algorithms hoping to drive down cost and improve performance and usability. Towards the end of my PhD, I was experimenting with controlling ground robots (wheelchairs and small UGVs) with eye gaze, which was when I bought an ODROID-X2!

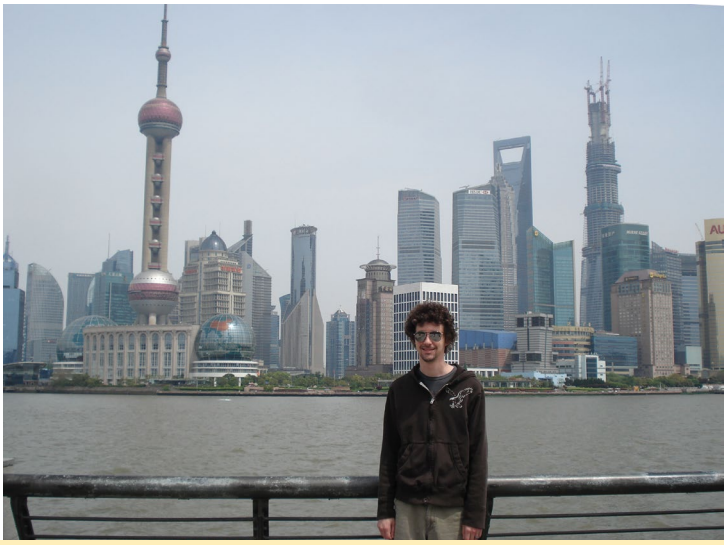
After graduating, I developed perception software for industrial material handling robots. I then accepted a faculty position at my university, and now I teach. I really love my job, and it gives me reason to continue to learn new skills and experiment with new commercial products. After all, no student wants to learn the inside-out workings of an embedded system that was popular 10 years ago.

What motivated you to create the popular Robotics edition image on the ODROID forums?

Most of my research work required Linux, OpenCV, Point Cloud Library, and ROS. When I bought the X2, I quickly realized that setting up my base environment was more complicated than what I was used to while working on my x86 development machines. I did a lot of research and found solutions to each of the compilation and installation problems I was encountering along the way, and finally had a nice stable image with everything set up just how I like it. I made a backup image of the SD card "just in case" and one day I decided to post it to the forums to give back to the community that had generously helped me with tips and ideas. I started getting a lot of replies on the original thread, and kept answering questions as they would come in. Shortly after I posted the X2 image, I bought a U2 and modified my same image, then posted as I did before. The folks at Hardkernel kept making new, innovative products, and before long I was getting requests to release images for other boards.

I was surprised that my images grew in popularity the way that they did. Hardkernel sent me some free stuff as part of their monthly giveaway, and the whole thing sort of took off from there. Hardkernel and the ODROID community in general have been very supportive of my work, and I try to release new images as official Ubuntu releases and new products are made available.

What hobbies and interests do you have apart from computers?



And across the world he went, as we can find our robotics expert chilling on a great skyline in Shanghai

I am an amateur machinist and rapid prototyping enthusiast. Recently, I have been working on various CNC machines (3D printers, routers, milling machines, and lathes). I am very excited by the maker movement happening right now, and having a garage full of automated manufacturing equipment is becoming more and more realistic every day. I am also experimenting with less technologically advanced manufacturing methods, such as woodworking and metal casting. Basically, all I do is make stuff.

Are you involved with any other computer projects unrelated to the ODROID?

I have made a few minimal contributions (a couple of minor bug fixes) to Point Cloud Library. As part of my job as a university instructor, I assign and mentor our senior capstone projects. These change each semester, and I tend to push a lot of robotics and embedded systems based projects. I place a strong emphasis on practical skills in my classes, and I spend about half of the lecture time giving demos and tech talks about exciting new gadgets (ODROIDs, 3D printers, and computer vision). I enjoy seeing the concepts that my students learn become implemented in their final projects, and I continue to make and tinker in my personal time so that I always have something new to teach.

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

I would like to see some progress made in the GPU, particularly with Linux. One of my interests is parallel programming with multicore GPUs, but my only real experience in this area is with the NVIDIA CUDA toolkit. Hardkernel has made good progress in this area, and I am excited to see how this develops.

What advice do you have for someone who wants to learn more about programming?

Start with a programming language that has lots of examples online, and maybe something with some nice GUI tools. Most people who start programming want to make a simple GUI app as their first program after the quintessential "Hello World" console app, but this can be tricky and cumbersome to set up for someone who is not familiar with IDEs, makefiles, compilers, etc. I really like the .NET framework with C#, making a GUI and doing most tasks in a Windows environment is pretty simple. Python is also a good starting language, but personally, I think a good background in C++ is something that separates the professionals from the rest.

The most important thing is not to worry about breaking things. When you are starting out in programming or even embedded development, you will break stuff. Sometimes you break code, sometimes you break your hardware, but if you aren't breaking something, you aren't learning anything!