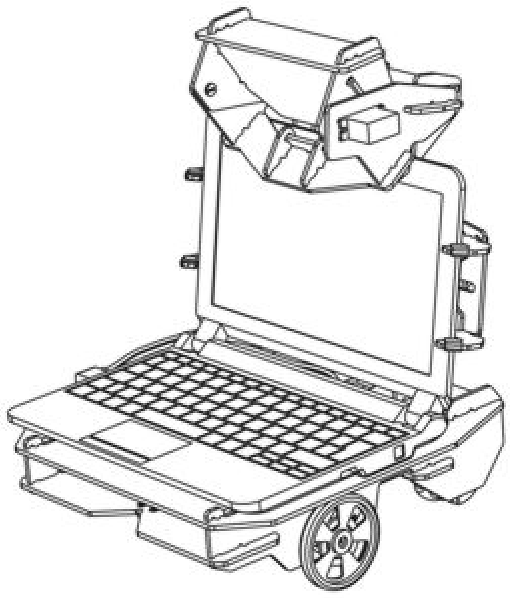
CONTENTS

INTRODUCING OCULUS

The Oculus system converts a small, reasonably modern, standard clamshell style laptop computer into a telerobotic, remotely operated vehicle. It uses the laptop cpu to run the control software, the laptop's battery to power the motors, the laptop's webcam and microphone for sensory input, and a wifi network for untethered communication. The adjustable frame fits best with 11" laptops or smaller.

The frame has powered wheels, and a tilting periscope aligned with the laptop's built-in webcam. Control software runs on Windows XP/Vista/7 and Linux, and is simple to install and configure. (The only potentially tricky part, for some, will be forwarding the 2 required ports thru your wifi network router, so the bot can be accessed over the internet.) Power and communication to the motors and controller is supplied entirely via the laptop's USB port. When not in use, the system is charged using the laptop's stock charger, routed through the included charging dock.

Once setup and running, you can log-in to your Oculus from any PC connected to the internet, from anywhere in the world, using just a web-browser. Usually no extra software install is required (the browser must have the adobe flash 10 or higher plugin, which is already installed in over 98% of the worlds PCs, according to Adobe). Remote (client) PCs can be running Windows, OS X, Linux or other operating systems, so long as they have a reasonably modern browser with the Flash 10 plugin. Android and iOS handheld client apps are also available.



After you've logged in, you can stream real-time video (captured from the Oculus laptop's built-in-webcam), and audio (captured from the built-in microphone). And of course, you can *drive around*. You'll have surprisingly precise and responsive steering control, and you can look up and down by tilting the upper mirror of Oculus's periscope.

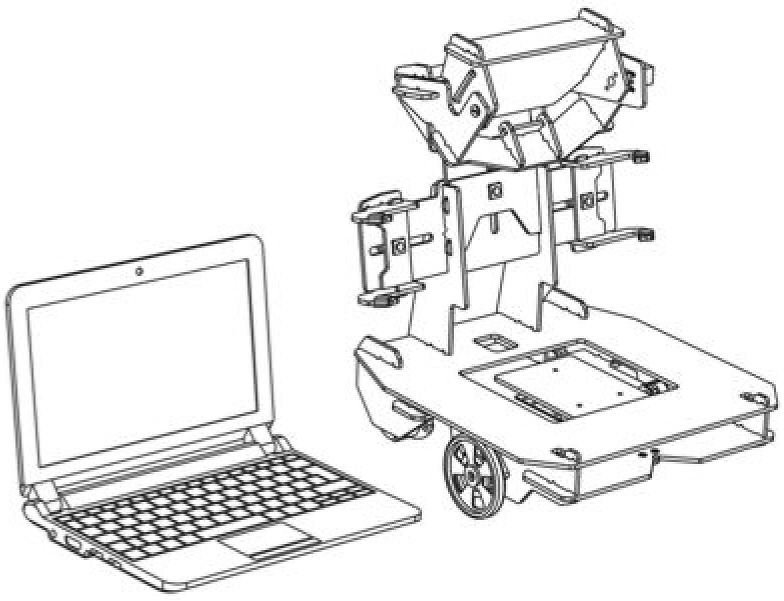
If you have a webcam on the PC you're connecting from you can also broadcast your own image and voice onto the screen and through the speakers of your Oculus laptop, for a true telepresence/video chat experience. Also, you can add multiple accounts so you can have many people connected to Oculus at once: one driver, and one or more passengers along for the ride, all seeing the same video stream.

The control software is free and open source, and easily extensible for hackers/enthusiasts who want to add their own robotic functionality. Also, the software will work just fine if installed on a desktop or laptop Windows PC *without* the Oculus hardware present: you won't be able to drive around (obviously), but you'll be able to broadcast audio and video to remote browsers as long as your machine has a webcam and mic.

SETUP

This section walks through the procedure of getting up-and-running with a fresh server software installation on the laptop/netbook you plan on mobilizing, and attaching and configuring the Oculus hardware. Don't attach the USB cable just yet, or mount your laptop in the frame (you'll want an unfettered view of the screen and comfortabe access to the keyboard for software installation).

ASSEMBLY



1. Assemble the periscope assembly to the frame with the thumbscrew and nut provided.
2. Attach the periscope servo cable to the servo cable extension, assuring that the lightest-colored wires of each line up (eg., white to white, or white to yellow).
3. If necessary, plug the servo extension cable into the Oculus controller board, with the light colored cable (eg., white) facing to the *center* of the board.
4. If necessary, plug in the USB cable into the controller board.

If you have a **preconfigured package** and are using it for the 1st time, you can skip ahead to mounting laptop in frame. The recommendations under operating system tuning notes has already been done. Basic calibration has also been done, but you may fine you need some fine tuning.

SOFTWARE INSTALLATION (Windows)

(For LINUX specific install info, see linux notes)

Java runtime 32-bit is required. If you don't have it, get it here. The 32 bit version is required, even if you're running 64 bit Windows. You may have to set your PATH to point to where java is installed (usually something like "C:\Program Files (x86)\Java\jre6\bin"). For info on how to set your PATH environment variable, see here.

Flash browser plugin v10 or higher is required for the server. If you don't have it already, get it here.

Download the Oculus full win32 software install package ZIP file from here. Unzip the file so the 'Oculus' directory and its contents is on the root of laptop's hard drive (eg., C:\Oculus).

Next, with the Oculus frame sitting next to the laptop, and the laptop connected to the internet, connect the USB cable.

DRIVER INSTALLATION (Windows)

When you connect the cable, Windows should initiate the driver installation process. On **Windows Vista or 7 the driver should be automatically downloaded and installed** (if this fails for some reason, download and install appropriate drivers from here).

On **Windows XP**, the Add New Hardware wizard will open:

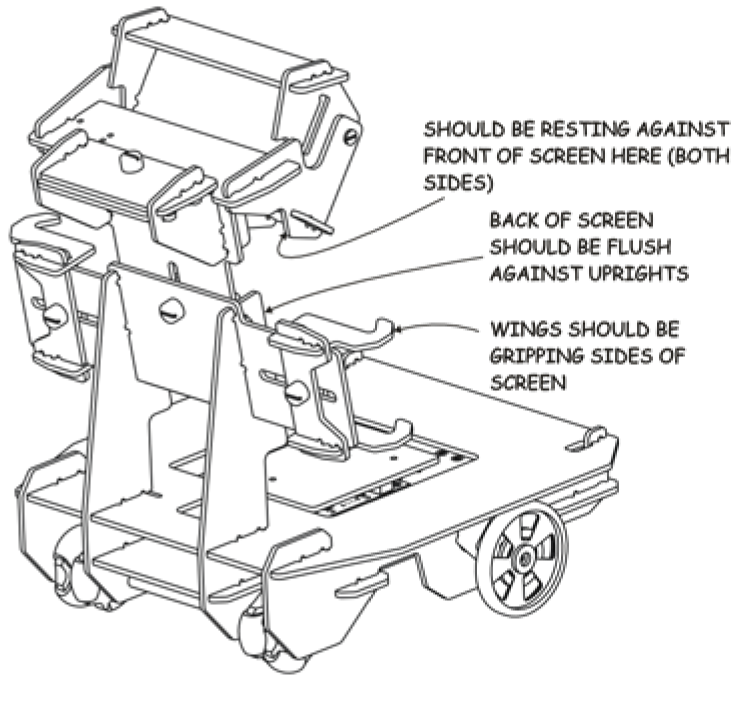
1. When asked "Can Windows connect to Windows Update to search for software?" select "No, not this

time." Click "Next."
2. Select "Install from a list or specified location (Advanced)" and click "Next."
3. Make sure that "Search for the best driver in these locations" is checked; uncheck "Search removable media"; check "Include this location in the search" and browse to the "\Oculus\FTDI USB Drivers" directory. Click "Next."
4. The wizard will search for the driver and then tell you that a "USB Serial Converter" was found. Click "Finish."
5. The new hardware wizard will appear again. Go through the same steps and select the same options and location to search. This time, a "USB Serial Port" will be found.
   You can check that the drivers have been installed by opening the Windows Device Mananger (in the Hardware tab of System control panel). Look for a "USB Serial Port" in the Ports section; that's the Oculus microcontroller board.
   Now that software and drivers are installed, you can mount the laptop in the Oculus frame.

MOUNTING LAPTOP IN FRAME

Loosen the 4 adjustment knobs, and extend the wings and the vertical periscope sliders to their maximum position.

1. Place the laptop in position, and reconnect the power and USB cable if necessary.
2. Move the adjusting slides into position, be careful to center the laptop within the frame so that the webcam at the top of the screen lines up with the center of the lower mirror.
3. Don't worry too much about the vertical position of the mirror now, you can take care of that in the next step.
4. Move the periscope assembly forward and back so that the back of it is resting firmly against the front of the mirror.
5. Tighten all the adjustment knobs. Don't over-tighten, finger tight is OK.
   With the hardware in place, you're ready to run the Oculus server software for the first time.

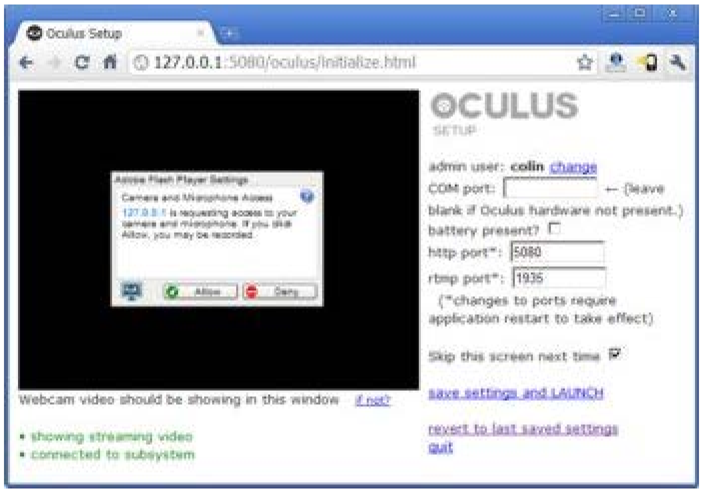

SHOULD BE RESTING AGAINST FRONT OF SCREEN HERE (BOTH SIDES)
BACK OF SCREEN SHOULD BE FLUSH AGAINST UPRIGHTS
WINGS SHOULD BE GRIPPING SIDES OF SCREEN

STARTING THE SERVER (Windows)

From within the "Oculus" directory, run "OCULUS_START.bat". A console window will open and the server will start. After a few moments, your default web-browser will open, showing the server initialize dialog.

Check if the video/webcam is streaming: if there is no image showing, there is probably a Flash dialog box that says "127.0.0.1 is requesting access to your camera and microphone". Click "ALLOW." You should see live video in the window.

1. Right-click in the video window and choose "Settings".
2. Check the "Remember" box.
3. Go to the microphone and camera tabs, and double-check that the correct cam and mic are selected.
4. Under the microphone tab, make sure the "reduce



Oculus Setup

127.0.0.1:5080/oculus/initialize.html

OCULUS
SETUP

admin user: **colin** change
COM port: ← (leave blank if Oculus hardware not present.)
battery present? ☐
http port*: 5080
rtmp port*: 1935
(*changes to ports require application restart to take effect)

Skip this screen next time ☑

save settings and LAUNCH

revert to last saved settings
quit

Webcam video should be showing in this window if not?

- showing streaming video
- connected to subsystem

Adobe Flash Player Settings
Camera and Microphone Access
127.0.0.1 is requesting access to your camera and microphone. If you click Allow, you may be recorded.

Server initialize dialog

echo" box is checked. Mic volume should be as high as possible without creating too much echo/feedback.

5. Adjust the vertical position of the periscope so that the top and bottom edges of the lower mirror are centered in the video frame
6. Enter an administrator username and password.
7. Make sure the "battery present" check box is checked (unless you're testing the software on a desktop PC)
8. Leave http port at 5080 and the rtmp port at 1935, unless you want to change them from the default (more info on this below).
9. Make sure the "skip this screen next time" checkbox is checked.
10. Finally, click "save settings and LAUNCH."

NETWORK SETUP

Network setup of Oculus involves a bit of know-how about IP addresses and wifi routers; it should be simple if you have some experience already. This section doesn't go into the nitty-gritty details, as there are already good resources on the web to learn from.

To be able to access Oculus from a remote PC on your local network, you need to know the correct address to point the browser to. By default, the IP address will likely be dynamic; that is, it is subject to change periodically. It will make your life simpler to set the IP address as static.

To access Oculus from a remote PC outside your home or office network, you'll need to setup your wifi router to forward ports 5080 and 1935 (or to different ports if you like, just remember to change them in the initialize dialog, as described above, and restart the application). Port 5080 is for the Apache Tomcat web server, and port 1935 is for the RTMP (real time messaging protocol) service used by Oculus to stream audio and video, and send and receive commands in real time.

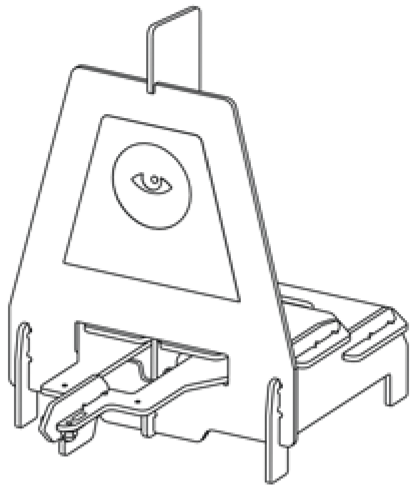
A good place to start looking for information on port forwarding is here: http://en.wikipedia.org/wiki/Port_forwarding. Once you have this setup, you no longer point to Oculus's address directly, you should point at the router's address. Note that some firewalls block pretty much all ports, so you may not be able to connect to Oculus if the remote PC is on a heavily locked-down network. The current version of Oculus software offers no solution for this (using dedicated ports allows maximum real-time responsiveness).

To avoid having to remember the numeric IP address all the time, a good tip is to use the DynDNS free domain name service (or similar). DynDNS is a good one because its so popular that some wifi routers have built in support to update it automatically if your internet service provider changes your IP address.

POSITION THE CHARGING DOCK

Oculus uses computer vision to automatically dock itself into the charging dock, and there are a few dock placement factors to know about:

1. Place it on the floor, near an AC wall outlet.
2. Floor should be level.
3. Floor should be smooth and not carpeted, for best results (or at least on fairly smooth, hard carpet).
4. It's best to place it with its back against a wall or some other immovable object, that won't be able to be nudged by Oculus when docking.
5. Oculus needs a clear view of the black and white graphic, to dock itself automatically—it's best for the dock to be placed in an *evenly* lit area, ideally such that Oculus doesn't cast a harsh shadow on the graphic when it gets close
6. When un-docking, Oculus automatically backs straight out about 3/4 meters, so make sure there is adequate clearance.



OPERATING SYSTEM TUNING NOTES (Windows)

Windows needs a few tweaks to be able to act as a real time operating sytem, suitable for robotic control.

1. Disable sleep/hibernate, while plugged-in and running on battery. If you don't, you won't be able to access Oculus while the system is sleeping.
2. Under power settings, set monitor ON AT ALL TIMES while running on battery. If nothing else, the monitor will shed a bit more light on the situation, and it will ensure the monitor will be ON if you're using 2-way video chat/telepresence. While plugged in, it's OK to allow the monitor to turn off automatically after a short delay.
3. For maximum performance, the web-browser tab running the server page always needs top focus (ie., it has to be the top-most window, no other applications can have higher priorty). If it doesn't, there may be a significant performance lag in video frame-rate, and control responsiveness. One of the biggest culprits of focus-stealing by the operating system is pop-up reminders in the system tray (eg, for java updates, system updates, and scheduled tasks). These might only come up every few days or weeks, but if you want to leave Oculus running and available for log-in while you're away on vacation, these reminder pop-ups could be a pain. Disable as necessary.
4. For maximum performance, close all other instances of the flash plugin (ie., no other web-pages should be open that could have a flash component in it).
5. Auto-start on reboot: put a shortcut to the "OCULUS_START.bat" file in the "Startup" folder in the start menu, and make sure no manual login is required on system startup (see here for instructions on that). This way, if you remotely reboot Oculus for whatever reason, the server will start up automatically and you'll be able to re-connect. Reboot the system a couple times to test your settings.
6. Setting Oculus to reboot the computer periodically really helps with long-term availability (if you want to leave it running for weeks/months). In the remote-control-web interface, goto menu > server settings, and set "reboot every 48 hours" to YES.

That's it for this setup section, but *technically*, you're not quite finished. Some calibration needs to be done, which is discussed in the next section.

CONTROLLING OCULUS

This section covers remotely controlling Oculus from a PC on the local, or on far-distant network, using just a web-browser. Some configuration and calibration is carried out through this interface, as well.

CONNECTING FROM A REMOTE PC

To access the PC client interface from a remote PC, fire up your favorite web browser. It needs to have Adobe's Flash plugin 10+ installed. Enter the URL in your web browser address bar. The url takes the format:

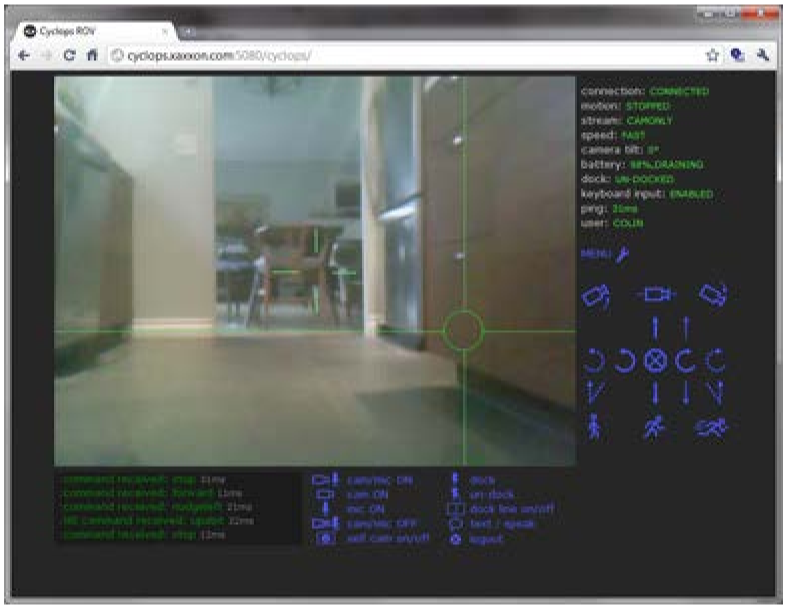
`http://`*ip-address-or-domain-name*`:`*port*`/oculus/`

replace *ip-address-or-domain-name* with the appropriate text (see section on network setup for more details). Replace *port* with the default port 5080, or other if you chose a different port in setup.

For example:
`http://192.168.0.111:5080/oculus/`

Once you got it right, you should be connected to Oculus and be facing the login screen. Login using the administrator username and password that you created in setup.
Once logged in, you should be looking at the command console page.

COMMAND CONSOLE WEB PAGE



```
connection: CONNECTED
motion: STOPPED
stream: CAMONLY
speed: FAST
camera tilt: 0°
battery: 98%,DRAINING
dock: UN-DOCKED
keyboard input: ENABLED
ping: 31ms
user: COLIN
```

```
command received: stop 21ms
command received: forward 11ms
command received: nudgeleft 21ms
[illegible] command received: [illegible] 22ms
command received: stop 12ms
```

This page is comprised of 5 sections:

1. Top left is the main video window.
2. To the right of the video window, on top, is the status section. This section lists various statuses of the system, which are updated in real-time by Oculus.
3. Below that, to the right of the video window, is the mouse controls area. When you hover the cursor over each icon in this area, it will show a description of what each does, underneath. And under each description, if available, there is shown a keyboard shortcut.
4. Lower left is the communication feed, primarily listing detailed feedback from Oculus, when you send commands.
5. Lower right is a set of controls for starting and stopping video, docking, and some other commonly used commands.

Extra features and advanced commands are accessed by hitting the “MENU” link.

NOTE: you can also access the main controls page on screen of Oculus itself, by hitting “launch controls” on the Oculus Server page. This is a useful option for calibration, particularily of the mirror tilt, as described below.

CALIBRATION

CAMERA TILT

After unboxing and setup, you probably won’t be able to do much of anything without calibrating the camera tilt, because the default values may be all wrong. To do this, first turn the video on by clicking “cam on.” Then hit “MENU” from the main controls page, then under the “settings” section, click “camera tilt settings.”

The “max clicksteer angle” doesn’t need to be tampered with at this stage. The horiz/max/min angles are what you’re trying to define. Valid numbers range from 0-180 (corresponding to degrees of tilt of the mirror servo). Start by entering a value of, say, 90 in the “Test at position” box and click “go.” Keep repeating this process until you’ve found a more-or-less horizontal position, and maximum and minimum tilt positions that don’t quite cause the mirror to actually hit the physical limit stops. Once you’re satisfied, make sure the values are in their corresponding boxes, and click “send values.”

DRIVING SETTINGS

To access the driving settings calibration tool, click “MENU” then “driving settings.”

STEERING COMP should be set first (if required). Drive Oculus forward and backward on smooth, level floor. It doesn’t have to be perfect, but if you find that there is a noticeable, annoying drift to the left or the right, enter a compensation value (start small at first). Click “send values” then test some more, and repeat as necessary.
NOTE: the motors may break in over time, you may find you have to adjust the steering comp value once in a while.

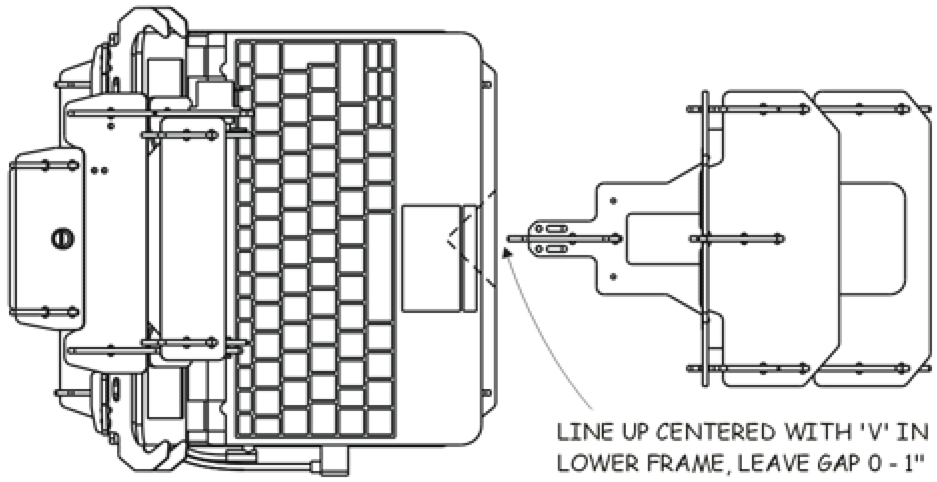
SPEED OFFSET values set the “slow” and “medium” speeds. Valid numbers range from 0-255, with 255 being maximum speed. The default values are probably OK initially, though you might later want to increase the “slow” speed a bit, if you find you don’t have enough low-speed torque. NOTE: docking uses “medium” speed, so if you change that, make sure docking still works.

NUDGE DELAY determines how far to steer when issuing a “nudge” command. The default value is likely OK.

MAX CLICKSTEER and MOMENTUM sets the *horizontal* accuracy of where Oculus points when you mouse-click in the main video window.
For “max clicksteer ms,” enter the time, in milliseconds, for Oculus to steer from the center to the very left or right edge of the screen. To determine if you’ve got it right, click on an object within view that is right at the left or right edge of the screen. If Oculus steers it to almost right in the center of the crosshairs, you’re set.
The “momentum x” setting usually takes a fractional value between 0-2. Set it higher for heavy rigs that fall short of the mark at targets closer to the horizontal center of the video window.
For *vertical* clicksteer accuracy, you can tweak the “max clicksteer angle” setting under “camera tilt settings.”

AUTO-DOCK

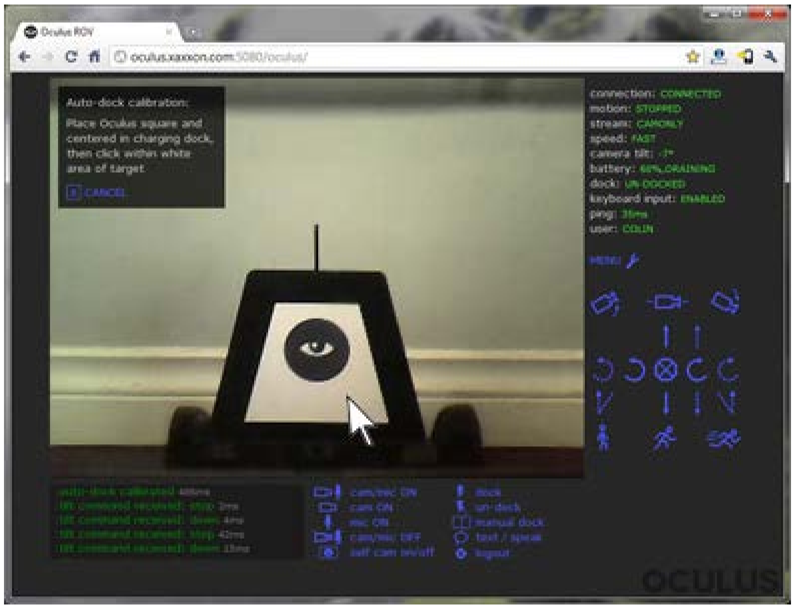
The visual docking system needs to be calibrated, with Oculus *manually* placed square and centered right in front of the dock, as shown in the diagram below.



Calibration placement

Once it is in position, follow these steps:

- Turn video on
- Make sure room is brightly lit, with even light (no shadows) cast on the dock graphic.
- Tilt camera to its minimum position, so it's looking down as low as it goes
- Click "MENU" then "calibrate auto-dock"
- Click somewhere within the white area of the dock graphic.


Auto-dock calibration:

Place Oculus square and centered in charging dock, then click within white area of target

connection: CONNECTED
motion: STOPPED
stream: CAMONLY
speed: FAST
camera tilt: -7°
battery: 60%,DRAINING
dock: UN-DOCKED
keyboard input: ENABLED
ping: 35ms
user: COLIN

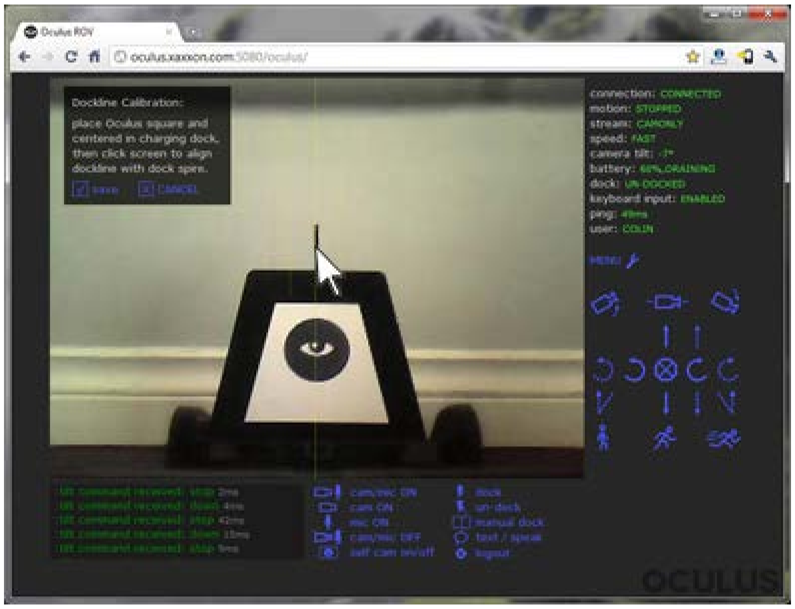
auto-dock calibrated 485ms
tilt command received: stop 2ms
tilt command received: down 4ms
tilt command received: stop 42ms
tilt command received: down 15ms

Auto-dock calibration

And you're done—you should only have to do this once. This writes the specs of the dock graphic to oculus_settings.txt

MANUAL DOCK LINE
In the rare event where you need to dock manually (eg., if *someone* has scribbled all over the dock graphic with a black marker), it helps to have the manual dock-line lined up with the center of the dock. Follow these steps:

- Place Oculus and the dock in the same position as described above
- Turn video on
- Click "MENU" then "calibrate dock line"
- Click in the video window, to line up the center yellow line with the "spire" at the top of the dock
- Click "save"



Dockline Calibration:
place Oculus square and centered in charging dock, then click screen to align dockline with dock spire.

connection: CONNECTED
motion: STOPPED
stream: CAMONLY
speed: FAST
camera tilt: -7°
battery: 60%,DRAINING
dock: UN-DOCKED
keyboard input: ENABLED
ping: 49ms
user: COLIN

```
tilt command received: stop 2ms
tilt command received: down 4ms
tilt command received: stop 42ms
tilt command received: down 15ms
tilt command received: stop 9ms
```

Dock line calibration

DRIVING

To the lower right of the video window, is the mouse controls area. When you hover the cursor over each icon in this area, it will show a description of what each does, underneath. And under each description, if available, there is shown a keyboard shortcut.
Rather than explaining here what each control does, it’s best that you just try each one out, and see for yourself. It takes a bit of practice if you’ve never driven a video ROV before, but you’ll be a master in no time.

STATUS DESCRIPTIONS

To the right of the video window, on top, is the status section. This section lists various statuses of the system, which are updated in real-time by Oculus:

CONNECTION: Describes state of your connection to oculus. Should generally say “connected” or “passenger”. If it doesn’t, try reloading the web page.

MOTION: Let’s you know if the wheel motors are moving or not. If it says “disabled,” it usually means you’re parked in the charging dock. Once you click “un-dock,” motion will be enabled.

ROV STREAM: Indicates if Oculus is transmitting a video and/or audio stream, or stopped.

SELF STREAM: Indicates if remote client browser is transmitting a video and/or audio stream through Oculus, or stopped.

SPEED: Indicates what wheel speed Oculus is set at.

CAMERA TILT: Indicates what angle the webcam periscope upper mirror is at.

BATTERY: Shows the latest charge status of the battery. It updates every few minutes. To update this status immediately, go to “MENU” > “battery status refresh”

DOCK: Indicates whether Oculus is parked in the charging dock or not.

KEYBOARD INPUT: Indicates if keyboard input is enabled or disabled.

PING: An indication of network connection speed. It is the round trip time, in milliseconds, for a message to be sent to Oculus and a response to be received. The longer this time is, the slower the controls will respond. Generally things will work nicely as long as ping *averages* less than about 300ms.

USER: Indicates who is logged in.

VIDEO AND AUDIO STREAMING

Under the video window are a set of controls for starting and stopping video and audio streaming on Oculus. To change the video quality setting, go to "MENU" and select one of the video quality options, or set a custom option.
If you've got a slow connection and are noticing significant video lag, try turning the sound off (ie., select "CAM ON"), this will give you a boost in available bandwidth.

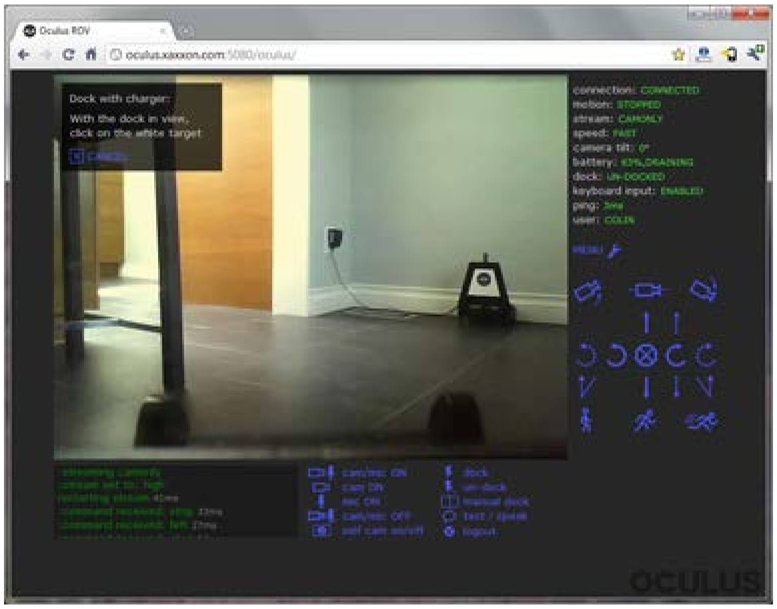
TWO-WAY VIDEO CHAT

With Oculus streaming video and audio set to ON, if your remote PC has a webcam you can enable two-way video chat by hitting "SELF CAM ON/OFF." If this pops up a Flash dialog box that says "... requesting access to your camera and microphone". Click "ALLOW." You should see your image showing in the lower left corner of the video window.
Notes about this feature:

1. The monitor of Oculus should be ON for this feature to be of much use. If you haven't enabled the monitor to be "always on while on battery" under power settings, you can remotely turn the monitor on by hitting "MENU" and "MONITOR ON." You might have to keep doing this periodically.
2. Excessive volume will cause echo issues. You might have to experiment with this and dial-in the volume settings on Oculus, and your remote PC. For best results, use a headset on the remote PC.
3. Results will vary between laptops, their microphone and speaker qualities, and mounting locations within the laptop. If you're *really* serious about 2-way video chat, it might be beneficial to add an external mic.

DOCKING WITH THE CHARGER

To park in the charging dock when not in use, drive within a couple meters of it, make sure it's showing in the video window, select "DOCK," then press "GO."



Dock with charger:
With the dock in view, click on the white target

connection: CONNECTED
motion: STOPPED
stream: CAMONLY
speed: FAST
camera tilt: 0°
battery: [illegible]
dock: UN-DOCKED
keyboard input: ENABLED
ping: [illegible]
user: COLIN

Auto-docking start

Oculus will begin the process of parking itself. It may back up and try again if it docks and doesn't find a charge, or if it gets in too close and finds itself too far off-target.
It will pause every few moments while docking, to analyse the webcam image. If the view of the dock is ever blocked while it's doing this (for example, if an absent-minded cat walks between), it may get stuck in a pause for a long time, then show a "target lost" message in the communication feed. If this happens, just try again.
To cancel the auto-docking process at any time, select "CANCEL" or "STOP".

UPDATING SOFTWARE

Oculus software is constantly undergoing development; adding new features, and fixing bugs.

To check if an update package is available, connect to oculus using the remote-web client as the admin user, click on “MENU” > “server commands” > “check for software update”
Follow instructions and restart server software when prompted.

LIMITATIONS

What? Oculus has limitations?

Yes, unfortunately it does, primarily because it runs *only* on power coming from the laptop’s USB cable. Most laptops limit this to 2.5 watts. Oculus goes great on smooth floors, but is subject to the following shortcomings:

- Some obstacles can be impassable, such as ‘steps’ between floor-types, often found in doorways between rooms. If the steps have short ramps, they might be OK.
- Stay away from thick/deep carpet. Smooth, hard carpet is not so bad, but turning can be slow.
- Auto-docking on carpet may not work.
- Laptops over 5lbs/2.3Kg may be too heavy.

Other Limitations:

- Oculus communicates over your wireless network. If you drive out of range, you will experience major lag or lose the connection.
- FIREWALLS: Oculus relies on two custom TCP ports for snappy, real-time performance (ports 5080 and 1935 are default). If Oculus or the remote PC is behind a heavily locked-down firewall on the network, connection may not be possible. To test if its going to work for your situation, install the free software.
- Oculus relies heavily on the laptop for proper functionality. It is designed to work with a wide variety of small laptops, but if the laptop itself isn’t functioning properly, Oculus may not either.

If you’re unsure about the above “other” limitations, it’s highly encouraged to install the free software only, without the Oculus hardware, on the laptop you would use it on, *before* buying. Try streaming video and audio; this will let you know if you’re likely to have problems with the laptop or your network.

LAPTOP COMPATIBILITY

Oculus works with all **11.6"** and **10.1"** laptops from **Acer, Asus, Dell, Toshiba, Fujitsu, MSI,** and **Gateway**. It is compatible with *some* 11.6" and 10.1" models from **Lenovo** and **HP/Compaq**. Small laptops from **Sony** and **Samsung** have incompatible DC power jacks, but work fine otherwise.

Laptop compatibility generally depends on overall size, and power jack size. Virtually all laptops with 11.6" and smaller screens fit fine, as long as they weigh less than about 5lbs/2.5kg.

The laptop webcam must be at the top of the screen, and centered (up to 1" offset-from-center webcams will fit OK for laptops with 10" screens).
Laptops must be running the Windows 8/7/XP/Vista, or Linux operating system.

While the latest laptops offer better performance, older systems work fine: even the original netbook, the Asus EeePC 701 with Windows XP, runs well with Oculus. Manufacturing date of 2007 and newer is recommended.

The Oculus system ships with plug adapter tips to fit the following DC barrel jack sizes (outer diameter x inner diameter):

- 5.5mm x 2.5mm
- 5.5mm x 1.7mm
- 4.8mm x 1.7mm
- 4.0mm x 1.7mm
- 2.3mm x 0.7mm

CONTACT US IF YOU'RE UNSURE
If you're unsure of what size DC connector your laptop has, or if it will fit in the frame — *please* contact us with your laptop make, model and model no., and we will promptly get back to you with an answer, no obligation implied. (Some of the DC connector measurements are very close so it can be tricky to define).

JUST RETURN IT IF IT DOESN'T FIT
If you purchase an Oculus system and find your laptop doesn't fit with the system, you will of course be able to return it no-charge.

OR JUST SPLICE THE CHARGER IN YOURSELF

If your laptop fits the frame (most with screens 11" and under do), but your power plug tip isn't one of the 5 sizes listed here, and you know how to do some very basic wiring, and don't mind chopping your charger cable in half (or getting another one and chopping that in half), you can easily make it work yourself. Basic instructions are here.

COMPATIBILITY TABLES

DETAILED COMPATIBILITY TABLE: CLICK HERE

Our research shows that current model 11.6" and 10.1" laptops have the plug sizes shown below, and are **compatible** with the Oculus system:

Size	Manufacturer, 11.6" and 10.1" laptops, Oculus compatible
5.5 × 2.5	Toshiba, some Lenovo, Packard Bell, Fujitsu, MSI
5.5 × 1.7	Acer, Dell, Gateway

4.8 × 1.7	older Asus Eee PCs
4.0 × 1.7	some HP/Compaqs
2.3 × 0.7	newer Asus Eee PCs

Known to be **incompatible** at this time: 10.1" laptops from Samsung, Sony

SYSTEM REQUIREMENTS

Oculus Server software requires the following to run properly:

- Windows 7/XP/Vista or Linux operating system 32 or 64 bit
- 500Mb RAM or more recommended
- Wifi 802.11 g/n
- Java runtime environment (32 bit recommended)
- Web browser (for server graphical user interface).
- Adobe Flash plugin
- USB 2.0 port
- Webcam (built-in, center mounted at top of screen recommended for auto-docking capability)
- Microphone recommended (built-in or external)

The free and open-source software does not require the Oculus hardware to run—feel free to download and install, to give it a try (movement commands won't work without an Oculus frame, but 2-way streaming audio and video should work fine)

LINUX NOTES

To run software on Oculus with Linux OS, download and unpack the OCULUS FULL package.

You'll need rxtx java libraries for serial over USB connection to Oculus hardware (even if if you don't have Oculus hardware attached). Download from here. Extract only the appropriate "librxtxSerial.so" from under the Linux subfolder, and put it in ../jre/lib/[machine type] (i386 for instance). If you don't know your JRE path you can use the javaHome utility under /Oculus (type "java javaHOME").

To run, from within the /Oculus folder enter "oculus_start.sh" NOTE: in some installations of Ubuntu 12.04, apparently you have to run the app as root, to be able to access the USB ports (or, change permissions of the ports under /dev eg., /dev/ttyUSB0)

The Oculus software uses Adobe Flash browser plugin for video streaming and communications. You need to get your webcam (and microphone too, ideally) operating in Flash. In some versions the Flash "settings" context menu is disabled, so you have to go to "Global Settings" and set "always allow" webcam/mic access from domain 127.0.0.1

Troubleshooting / Frequently Asked Questions

Microphone doesn’t seem to transmit from remote PC web client

Hold down ‘T’ to talk. (This feature was added as an anti-feedback measure)
If that doesn’t help, check flash settings (on remote PC) to make sure you have the correct microphone selected
NOTE: A pending software update will make this feature optional

Auto-Docking – Oculus has a real hard time docking itself. What could be wrong?

A number of factors have to be right for Oculus to dock itself automatically dock itself, with minimal retries. Double check these things:

- “Clicksteer” has to be accurate. Check the calibration notes under Camera Tilt and driving settings
- Auto-Dock calibration has to be done, and ideally *re-calibrated* if the dock has been moved to a floor surface slightly out-of-level, if the netbook has been re-installed within the frame, in a slightly different position, or if the periscope height/position has been changed slightly.
- The dock target has to be evenly lit. If there are stark shadows or glare falling across the graphic, or if it is extremely dim, Oculus may not be able to recognize it.
- Some laptops supply slightly more juice to the USB port than others. If you have an unusually high-powered bot (which is otherwise a good thing!), you might need to increase the time it takes for Oculus to come to a full stop, so auto-docking isn’t dealing with movement blur when analyzing the dock image. In oculus_settings.txt, try adding this line under the “manual settings” section: “stopdelay 800”
  (or replace 800 with any number higher than the default of 500—this corresponds to the time in milliseconds it takes Oculus to come to a full stop)
- Try re-calibrating the auto-dock with the bot sitting slightly further out from the dock
- Auto-docking on smooth, hard floor is going to work best. On any floor surface plusher than hard/dense carpet, it might not work at all, as turning may be too slow.

I keep getting ‘the system cannot find the path specified.’ error when running OCULUS_START.bat

Newer versions of Java for Windows don’t fully install themselves; sometimes you have to add to the PATH environment variable to point to the folder where the java executable is. (Usually something like “C:\Program Files (x86)\Java\jre7\bin”.) Info on how to set your PATH is here:
http://www.java.com/en/download/help/path.xml

How can you tell which version of ArduinOculus firmware is loaded on Oculus?

The firware will output the version number every time it is reset. The easiest way to read it is to login to Oculus through a web-browser, go to MENU > Advanced Menu and click ‘reset arduinoculus board.’ The version information will appear in the message window.

What if I find the eyeball graphic on the dock kind of creepy?

Just fill in the white areas with a black marker, so it looks like this – the auto-docking image tracking (DMVAM) will still work fine.

Is it possible to extract a single image from the video stream?

YES – In the web-browser client, go to ‘MENU > advanced menu > frame grab.’ Or, just point your browser to the frame grab url:

http://*ip-address-or-domain-name:port*/oculus/frameGrabHTTP

You can also use the ‘`framegrabtofile`’ command to save images to disk

Programming Oculus – Introduction

The Oculus-Java application is an extension of the open source Red5 streaming media server, which includes the Apache Tomcat web server.

Because the Oculus hardware is an open Arduino based platform controlled by a full desktop OS, there are numerous approaches to programming it and adding your own functionality, including:

• Interact with the Oculus Java Application using almost *any programming language,* via the powerful socket based **Telnet Programming Interface**. With it you can use dozens of Oculus text commands, reference Oculus STATE variables, and manipulate configuration settings.

• Extend the open-source Oculus Java software itself. A guide to getting started in Eclipse is here

• Run RoboRealm on the robot, *instead of* or *along-with* Oculus-Java, using the Oculus Module included with purchase of RoboRealm. RoboRealm is a rich computer vision toolkit, with a powerful scripting language. It also offers a telnet/socket API, so you can can use any programming language to interact with RoboRealm *and* Oculus-Java *simultaneously*

• Take a lower-level direct approach, and modify the ArduinOculus microcontroller firmware (source). You can upload new firmware to it with the Arduino IDE, choosing 'Duemillenove' as board type

• Many other approaches are available. With Windows or Linux as the controlling OS you can explore the Microsoft Robotics Developer Studio, or ROS – The Robot Operating System, or take advantage of the powerful OpenCV – Open Source Computer Vision libraries.

Also in this section:

The Telnet Programming Interface

```
Telnet 127.0.0.1
<telnet> Welcome to Oculus b
<telnet> LOGIN with admin us
```

INTRO

The Oculus Telnet Interface allows you to converse with the Oculus-Java application via TCP socket connection. You can write automation scripts using any programming language that supports TCP socket connections and is capable of deciphering text strings; eg., Python, Ruby, C#, Visual Basic, C, Java, Perl, and many others. You can also use it interactively, with any telnet client.

When you or your program/script logs into the Oculus Telnet Interface, you have complete control over all the robot's functions, and access to its real-time status output stream. A program can be running on the robot itself, or on a remote machine. The interface allows multiple connections, so you can have multiple programs running simultaneously, and your program can launch other programs.

NOTE: the other 2 ways clients communicate with the Oculus-Java application are through RTMP and HTTP protocols. The web-browser and Android/iOS clients primarily communicate via RTMP (a protocol originally developed by Macromedia and is used for streaming video and text)

GETTING STARTED

The telnet interface accepts any of the Oculus Text Commands, and has access to all of the Oculus-Java STATE variables and configuration settings.

The easiest way to familiarize yourself with the Oculus Telnet Interface is to log into it manually from another machine, and play around. By default, the telnet server is enabled on port 4444. To change this, you can edit the 'oculus_settings.txt' file and change the value for the 'commandport' setting to another port (or 'disabled' if you want to disable it).

With Oculus-Java running, connect to it using a telnet client. In Windows, you can use the 'telnet.exe' program if it's installed, or putty.exe is another popular option. If you're using Putty, set host to the robot's IP address, port to '4444,' and the connection type to 'telnet.' Under the 'connections > telnet' configuration category, set 'telnet negotiation mode' to 'passive.'
If you're using Linux, at a command prompt type something like '`telnet 192.168.0.99 4444`' (replacing the IP address with the correct one).

You should be greeted with something like this:

```
Trying 192.168.0.99...
Connected to 192.168.0.99.
Escape character is '^]'.
<telnet > Welcome to Oculus build 639
<telnet > LOGIN with admin user:password OR user:encrypted_password
```

At this point you can enter your login credentials as ‘`username:password`’ OR replace the plain-text password with the encrypted version, found at the bottom of the file ‘Oculus/conf/oculus_settings.txt,’ at the line ‘pass0’. So, you would login with something like ‘`username:Mup8F325S5CzeG6XM3xJug1AOeU=`,’ if you have any concerns about sending a plain-text password over the network.

Once logged in correctly, a good thing to do now is to also log in with the web-browser client, start controlling the robot, and watch the telnet output stream.

SENDING COMMANDS AND UNDERSTANDING OUTPUT TAGS

With the telnet interface you have complete control of the robot. To see a list of all available commands, enter ‘`help`’ or enter ‘`help` *command*’ to see extended help on a particular command.

Try sending a command: with the robot camera turned off, enter:

```
     publish camera
```

You will notice the video stream start in the web client, and the telnet server will output:

```
     < messageclient > command received: publish camera
     < state > stream camera
     < messageserverhtml > streaming camera
```

Notice the XML style tags preceding each line—these are added by the telnet server to identify specific message types, to make the output easier to understand and be parsed by programs. There are currently 4 primary output tags:

1. `< telnet >` – These are messages sent by the telnet server to the currently connected telnet user only. For example, the login prompt *always* starts with ‘`< telnet > LOGIN`,’ so your script can know that the server is waiting for login information.

2. `< messageclient >` – These are messages sent to all connected users—they will appear in the message window of a RTMP-connected driver (not passengers), and any telnet output streams

3. `< messageserverhtml>` – These are messages sent only to the javascript running in the web-browser on the robot (either initialize.html or server.html), and to any telnet output streams

4. `< state>` – These are messages pertaining to Oculus STATE variables

Sometimes `< messageclient >` and `< messageserverhtml>` tags will be followed by the `< status>` tag. This represents information *not* sent to the message window, but for other purposes required by the web page (eg., in the case of the web-client, to update the upper-right status readouts)

Lastly, messages that span more than one line will be surrounded by the `< multiline>< /multiline>` tags.

NEXT: an example telnet program

Example Python Telnet Program – React to Noise

Below is a script that uses the Oculus Telnet Interface, written in Python. This will log into the Oculus-Java server running on the robot, startup the microphone, and synth-voice complain whenever it hears a loud noise.

```
# onsound.py

import socket
import re
import time

host = "127.0.0.1"
username = "admin"
password = "KurwGDyd+oy+ZZ3S4qMn/wjeKOQ=" #encrypted password
port = 4444

#FUNCTIONS

def sendString(s):
        oculusock.sendall(s+"\r\n")
        print("> "+s)

def waitForReplySearch(p):
    while True:
        servermsg = (oculusfileIO.readline()).strip()
        print(servermsg)
        if re.search(p, servermsg):
            break
    return servermsg

#MAIN

#connect
```

```
oculusock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
oculusock.connect((host, port))
oculusfileIO = oculusock.makefile()

#login
waitForReplySearch("^<telnet> LOGIN")
sendString(username+":"+password)

sendString("publish mic")
time.sleep(2)
while True:
    sendString("setstreamactivitythreshold 0 10")
    waitForReplySearch("streamactivity: audio")
    sendString("speech please be quiet")
    time.sleep(5)

oculusfileIO.close()
oculusock.close()
```

Let's walk through this program bit by bit to break down what it's doing:

```
import socket
import re
import time
```

These import the Python modules we need: 'socket' to communicate via TCP/telnet protocol, 're' for regular expression string parsing, and 'time' to measure time.

```
host = "127.0.0.1"
username = "admin"
password = "KurwGDyd+oy+ZZ3S4qMn/wjeKOQ=" #encrypted password
port = 4444
```

This declares variables we need to connect and login. In this case, 'host' is set to 127.0.0.1, for the program to be running on the robot's laptop, but you could also run it remotely and change 'host' to the IP address of the robot. This script is using the encrypted version of the password, found next to 'pass0' in 'Oculus/conf/oculus_settings.txt.'

```
def sendString(s):
    oculusock.sendall(s+"\r\n")
    print("> "+s)
```

This declares a function that sends text to the Oculus-Java server, followed by line-ending '`\r\n`' characters, so the server knows to process it as a single chunk of text.

```
def waitForReplySearch(p):
    while True:
        servermsg = (oculusfileIO.readline()).strip()
        print(servermsg)
        if re.search(p, servermsg):
            break
    return servermsg
```

This function reads a line of output from the robot and compares it to a regular expression. If it matches, it returns the whole line. If not, it keeps reading lines forever until it finds a match. The two above functions set the program up for 2-way ‘conversation’ with the server.

```
#connect
oculusock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
oculusock.connect((host, port))
oculusfileIO = oculusock.makefile()
```

This makes a TCP/socket connection with the server, and converts it into fileIO so it interprets single lines as individual chunks of text to be processed

```
#login
waitForReplySearch("^<telnet> LOGIN")
sendString(username+":"+password)
```

This waits for the server to output a login string, then sends login info.

```
sendString("publish mic")
time.sleep(2)
while True:
    sendString("setstreamactivitythreshold 0 10")
    waitForReplySearch("streamactivity: audio")
    sendString("speech please be quiet")
    time.sleep(5)
```

And finally, this is the ‘main loop’ of the program that takes command of the robot and reacts to events. First, it turns on the microphone and waits a couple of seconds for it to start up. Then it enters a ‘do forever’ loop where it first sends the ‘`setstreamactivitythreshold`’ command with parameters 0, 10. This tells it to ignore video and react to audio levels higher than 10 (out of 100). Then it waits, possibly for a long time, for the robot to hear something, and output a line containing ‘streamactivity: audio’. At this, our program sends the ‘speech’ command to politely ask for silence, waits a few seconds for the synth-voice ouput to end, then the loop restarts and it goes back into listening mode.
(Note that the ‘`setstreamactivitythreshold`’ command needs to be re-sent every time, at the beginning of the loop, since the listener is automatically turned off every time it is triggered.)

To make this script a bit more *useful,* instead of ‘synth-voice nagging’ you could, for example, post a new RSS news item using the ‘`rssadd`’ command and have a service like IFTTT.com react to it and send a text message. Or, since IFTTT only checks at 15 minute intervals, you could use a more immediate method and send an alert via email using the ‘`email`’ command, as in the next example

Example RUBY Telnet Program – Email if Draining Battery

The Ruby program below is similar in structure to, and defines the same functions as the previous Python example.

The main loop at the end runs every 10 minutes and logs in/out each time to check battery status (this is more reliable than connecting only once at program start, since the TCP connection could timeout after a while).

It uses the '`who`' command to check if there are any active RTMP users connected; that is, users connected via the web-client or Android/iOS apps (if this was the case there would be no reason to panic and send a battery-draining email). To see if the battery is draining, the program reads the STATE variable 'batterystatus'.

Email is not configured by default — for info on how to configure it, search for the '`email`' command in the Oculus Command Reference, and associated settings info in the oculus_settings.txt config file.

```
# batt_draining_email.rb

require 'socket'

host = "127.0.0.1"
username = "admin"
password = "KurwGDyd+oy+ZZ3S4qMn/wjeKOQ=" #encrypted password
port = 4444
$oculussock
emailsentonce = false

#FUNCTIONS

#send text over socket
def sendString(str)
    $oculussock.puts str
    puts "> "+str
end

#receive text over socket, waiting for pattern
def waitForReplySearch(pattern)
    while true
        servermsg = $oculussock.gets.strip
        puts servermsg
        if servermsg =~ pattern then break end
        end
    return servermsg
end
```

Example Python Telnet Program – Rotate until Find Dock

Here is another Python Example that incrementally rotates the robot, looking for the dock. If it finds it, it starts the auto-docking routine, and exits.

It sets up the same `sendString()` and `waitForReplySearch()` text conversation functions as the previous example, and adds the `rotateAndCheckForDock`() function. This function rotates the robot a bit with the ‘`move right`’ command, then checks ambient lighting using the ‘`getlightlevel`’ command—if it’s too dark, it will turn on the floodlight (if connected).
It then sends the ‘`dockgrab`’ command to check if the dock target is in view: if the STATE variable `dockxsize` > 0 (target greater than 0 pixels in width), it knows it has found it, and will start the autodocking routine by sending ‘`autodock go`’.

```
# findock.py

import socket
import re
import time
```

```
host = "127.0.0.1"
username = "admin"
password = "KurwGDyd+oy+ZZ3S4qMn/wjeKOQ=" #encrypted password
port = 4444


#FUNCTIONS

def sendString(s):
        oculusock.sendall(s+"\r\n")
        print("> "+s)

def waitForReplySearch(p):
    while True:
        servermsg = (oculusfileIO.readline()).strip()
        print(servermsg)
        if re.search(p, servermsg):
            break
    return servermsg

def rotateAndCheckForDock():
        # rotate a bit
        sendString("move right")
        time.sleep(0.6)
        sendString("move stop")
        time.sleep(0.5) # allow to come to full stop

        # if too dark, turn floodlight on
        sendString("getlightlevel")
        s = waitForReplySearch("getlightlevel:")
        lightlevel = int(re.findall("\d+",s)[0])
        if lightlevel < 25:
                sendString("floodlight on")

        # check if dock in view
        sendString("dockgrab")
        s = waitForReplySearch("<state> dockxsize")
        dockwidth = int(re.findall("\d+",s)[0])
        if dockwidth > 0:
                return True
        else:
                return False

#MAIN

#connect
oculusock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
oculusock.connect((host, port))
oculusfileIO = oculusock.makefile()

#login
waitForReplySearch("^<telnet> LOGIN")
sendString(username+":"+password)

#tell any connections what's about to happen
sendString("messageclients launching finddock.py")
```

```
#turn camera on if it isn't already
sendString("state stream")
s = waitForReplySearch("<state> stream")
if not re.search("(camera)|(camandmic)$", s):
        sendString("publish camera")
        time.sleep(5)

#set camera to horizontal
sendString("cameracommand horiz")

#keep rotating and looking for dock, start autodock if found
dockfound = False
for i in range(20):
        sendString("messageclients attempt #: "+str(i))
        if rotateAndCheckForDock():
                dockfound = True
                break

if dockfound:
        sendString("autodock go")
        waitForReplySearch("<state> dockstatus docked")
else:
        sendString("messageclients finddock.py failed")

oculusfileIO.close()
oculusock.close()
```

OCULUS COMMANDS

The following listing is of text commands used by clients (javascript, Android, iOS and telnet clients) to communicate with the Oculus Java server applicatin. They can be grouped into the following categories:

MOVEMENT

move *left | right | forward | backward | stop*
Wheel motors, continuous movement

nudge *left | right | forward | backward*
Move for amount of milliseconds (specified by 'nudgedelay' setting)

slide *left | right*
Rearward triangular movement macro, that positions robot slightly to the left or right of starting spot

speedset *slow | med | fast*
Set drive motor speed

clicksteer *{INT} {INT}*
Camera tilt and drive motor movement macro to re-position center of screen by x,y pixels

motionenabletoggle *(no arguments)*
Enable/disable robot drive motors

getdrivingsettings *(no arguments)*
Returns drive motor calibration settings

drivingsettingsupdate *{INT} {INT} {INT} {INT} {DOUBLE} {INT}*
Set drive motor calibration settings: slow speed, medium speed, nudge delay, maxclicknudgedelay (time in ms for robot to turn 1/2 of screen width), momentum factor, steering compensation

PERISCOPE MIRROR TILT

cameracommand *stop | up | down | horiz | downabit | upabit*
Camera periscope tilt servo movement

tilttest *{INT}*
Move persicope tilt servo to specified position

holdservo *true | false*
Set/unset use of power break for persicope servo

gettiltsettings *(no arguments)*
Returns camera calibration settings

tiltsettingsupdate *{INT} {INT} {INT} {INT} {INT}*
Set camera calibration settins: horiz tilt, max tilt, min tilt, maxclickcam (time in ms for tilt to move 1/2 screen height), video scale %

VIDEO / AUDIO

publish *camera | camadnmic | mic | stop*
Robot video/audio control

speech *{STRING}*
Voice synthesizer

streamsettingsset *low | med | high | full | custom*
Set robot camera resolution and quality

streamsettingscustom *{STRING}*
Set values for 'custom' stream: resolutionX_resolutionY_fps_quality

videosoundmode *low | high*
Set robot video compression codec_

playerbroadcast *camera | camadnmic | mic | stop*
Client video/audio control (to be broadcast thru robot screen/speakers)

pushtotalktoggle *true | false*
When broadcasting client mic through robot speakers, always on or mute until keypress

setstreamactivitythreshold *{INT} {INT}*
Set video motion, audio volume detection threshold, 0-100 (0=off)

getlightlevel *(no arguments)*
Returns average pixel greyscale value (0-255) of frame from current stream

setsystemvolume *{INT}*
Set robot operating system audio volume 0-100

muterovmiconmovetoggle *(no arguments)*
Set/unset mute-rov-mic-on-move' setting

framegrabtofile *(optional) {INT}*
Saves a frame from video stream to JPG in folder Oculus/webapps/oculus/framegrabs (creating folder if not already present). Folder is publicly view-able via web url: http://*ip-address-or-domain-name:port*/oculus/framegrabs. Eg.:

http://69.163.206.138:5080/oculus/framegrabs

NOTE: You can also access single frames without saving to disk, by going to 'MENU > advanced menu > frame grab' from within the web-client. Or, just point your browser to the frame grab url:

http://*ip-address-or-domain-name:port*/oculus/frameGrabHTTP

USERS / ACCOUNTS

password_update *{STRING}*

Set new password for currently connected user

new_user_add *{STRING} {STRING}*
Add new user with ‘username’ ‘password’

user_list *(no arguments)*
Returns list of user accounts

delete_user *{STRING}*
Delete user ‘username’

extrauser_password_update *{STRING} {STRING}*
Set new password for user with ‘username’ ‘password’

username_update *{STRING} {STRING}*
Change non-connected username with ‘oldname’ ‘newname’

beapassenger *{STRING}*
Be passenger of current driver, specify passenger ‘username’

playerexit *(no arguments)*
End rtmp connection with robot

disconnectotherconnections *(no arguments)*
Close rtmp connection with all user connections other than current connection

assumecontrol *{STRING}*
Assume control from current driver, specify new driver ‘username’

who *(no arguments)*
Returns information on current driver connected via RTMP (eg., flash/web/mobile clients), and number of users connected via telnet

loginrecords *(no arguments)*
Returns list of RTMP driver (eg., flash/web/mobile clients) login history for current server session

DOCK/CHARGING

dock *dock | undock*
Start manual dock routine

autodock *cancel | go | dockgrabbed | dockgrabbed {STRING} | calibrate | getdocktarget*
Autodocking system control. Camera must be running

dockgrab *(no arguments)*
Find dock target within robots camera view, writes target data to corresponding STATE variables (if found, will be non-zero values). Robot camera must be running

docklineposupdate *{INT}*
Set manual dock line position within camera FOV in +/- pixels offset from center

autodockcalibrate *{INT} {INT}*
Start autodock calibration, by sending xy pixel position of a white area within the dock target

battstats *(no arguments)*
Gets battery charging state and charge remaining, writes to corresponding STATE variables and client

status

SYSTEM

systemcall *{STRING}*
Execute OS system command, with 'Oculus/' as the current working directory

showlog *(no arguments)*
Returns partial Oculus/logs/jvm.stdout

monitor *on | off*
Robot display sleep/wake control

softwareupdate *check | download | versiononly*
Robot server software update control

restart *(no arguments)*
Restart server application on robot

shutdown *(no arguments)*
Quit server application on robot

memory *(no arguments)*
Returns percent used, total memory, and free memory in use by the Java Virtual Machine. NOTE: JVM memory is not to be confused with *system* memory totals.

uptime *(no arguments)*
Returns time in milliseconds since the server application started

OTHER

relaunchgrabber *(no arguments)*
Launch new server.html browser page on robot

chat *{STRING}*
Send text chat to all other connected users

statuscheck _(optional) battstats _
Request current statuses for various settings/modes. Call with 'battstats' to also get battery status. NOTE: mainly used by rtmp clients only for initial status populating, and frequent 'PING' status updates. Output to telnet clients is suppressed

arduinoecho *true | false*
Set ArduinOculus microcontroller to echo all commands

arduinoreset *(no arguments)*
Reset ArduinOculus microcontroller

spotlight *0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100*
Set main spotlight brightness. (0=off)

floodlight *on | off*
Controls wide angle light

writesetting *{STRING} {STRING}*

Uses format: *key value*
Write setting to oculus_settings.txt

opennisensor *on | off*
Kinect/Xtion Primesense sensor control. NOTE: developer mode has to be enabled by setting 'developer' to 'true' in oculus_settings.txt

email *{STRING}*
Uses format: *email-address [subject] body*
Send email to *email-address*, with subject *[subject]* and body text *body*
Example:

```
email bob@example.com [Test email] This is an email test
```

Uses mail server settings specified in oculus_settings.txt configuration file values for 'email_smtp_server, email_smtp_port,' (and 'email_username, email_password' if required)
To use Gmail to send mail: set the server to 'smtp.gmail.com,' port to 587, and use Gmail username and password

state *(optional) {STRING} (optional) {STRING}*
Uses format: *key*, or *key value*
If called with no arguments: returns list of all non-null robot STATE variables
If called with one argument: returns value for single *key*
If called with two arguments: sets *key* to *value*

quit *(no arguments)*
Quit telnet session (telnet connection only)

help *(optional) {STRING}*
If called with no arguments: returns list of all commands
If called with argument *command*, returns quick help for single command

settings *(no arguments)*
Returns list of all settings from oculus_settings.txt

analogwrite *{INT} {INT}*
Sends command 'a' followed by two bytes (pin #, value) to ArduinOculus microcontroller. (Requires firmware version 0.5.5 or higher)

digitalread *{INT}*
Sends command 'd' followed by byte (pin #) to ArduinOculus microcontroller, requesting status of digital pin. (Requires firmware version 0.5.5 or higher.) Returns 1 or 0 within output string. Example output, reading HIGH value on pin 12:

```
< digitalread 12: 1 >
```

messageclients *{STRING}*
Send text to all other connected users. Similar to 'chat,' but without preceding user info

rssadd *{STRING}*
Uses format: *[title] description*
Adds item to Oculus RSS feed file: Oculus/webapps/oculus/rss.xml (creating it if not already present). Feed is publicly view-able via web url:
http://ip-address-or-domain-name:port/oculus/rss.xml Example:

```
http://69.163.206.138:5080/oculus/rss.xml
```

STATE Variable Reference

STATE variables are used extensively by the Oculus Java application to store real-time information about the current status, or state, of the robot's various functions and systems. Some are very handy to access from the Telnet Programming Interface, when programming the robot via an external script. The 'state' command description is as follows:

state *(optional) {STRING} (optional) {STRING}*
Uses format: *key*, or *key value*
If called with no arguments: returns list of all non-null robot STATE variables
If called with one argument: returns value for single *key*
If called with two arguments: sets *key* to *value*

Whenever a state variable changes, it is immediately written to the telnet output stream, preceded by the `< state >` tag. External scripts can set up event-handlers to react to these.

State Variable Groups:
Motors
Dock
Lights
RTMP Users
System
Video/Audio

MOTORS

firmware *{STRING}*
Name returned by microcontroller connected to motors (typically ArduinOculus, which returns 'OculusDC')

serialport *{STRING}*
Name of serial-over-USB port communicating with microcontroller connected to motors

motionenabled *{BOOLEAN}*
TRUE if wheel motors are allowed to move (typically FALSE when docked)

speed *{STRING}*
Current motor speed setting

tempdirection *{STRING}*
Temporary direction of motion, used by wheel movement macros

moving *{BOOLEAN}*
TRUE if wheel motors are moving

sliding *{BOOLEAN}*
TRUE if 'slide' wheel movement macro is in motion

movingforward *{BOOLEAN}*
TRUE if motors are moving wheels in forward direction

camservopos *{INT}*
Actual position, in degrees, of mirror tilt servo motor

DOCK

dockgrabbusy *{BOOLEAN}*
TRUE if auto-docking system is in the process of grabbing single frame from video feed, and trying to find the dock target within the image

docking *{BOOLEAN}*
TRUE if final docking macro is in progress, where robot is moving straight forward in increments, looking for battery charging status

dockstatus *{STRING)*
Current state of dock. Can be 'docked,' 'un-docked,' or 'docking'

autodocking *{BOOLEAN}*
TRUE if overall auto-docking routine is in progress

dockxsize *{INT}*
Last detected width of dock target, in pixels (320×240 resolution)

dockslope *{FLOAT}*
Last detected slope of dock target, in degrees

dockxpos *{INT}*
Last detected center X position of dock target, in pixels (320×240 resolution)

dockypos *{INT}*
Last detected center Y position of dock target, in pixels (320×240 resolution)

LIGHTS

floodlighton *{BOOLEAN}*
TRUE if flood light is on

spotlightbrightness *{INT}*
Brightness of spotlight: 100=full, 0=off

lightport *{STRING}*
Name of serial-over-USB port communicating with OcuLED lights microcontroller

RTMP USERS

driver *{STRING}*
Current driver username, connected via RTMP

logintime *{LONG}*
Last time driver RTMP client logged in (in milliseconds since epoch)

pendinguserconnected *{STRING}*
Username of rtmp connected user waiting to be driver/passenger

SYSTEM

boottime *{LONG}*

Time of server application startup (in milliseconds since epoch)

batterystatus *{STRING}*
Current status of battery. Can be ‘charging’ or ‘draining’

batterylife *{INT}*
Current battery charge percentage

localaddress *{STRING}*
IP address of robot system within the local wifi network. NOTE: this value can be inaccurate if the system has networking complexities such as virtual networks, or multiple NICs

externaladdres *{STRING}*
External IP address of robot system, obtained from ‘http://checkip.dyndns.org/’

httpPort {INT}
Robot’s web-server HTTP port (default 5080)

streamActivityThreshold {STRING}
Current value of video/audio activity thresholds, set by ‘setstreamactivitythreshold’ command

streamActivityThresholdEnabled *{LONG}*
Time streamActivityThreshold was set (in milliseconds since epoch), NULL if no activity detection is set

VIDEO/AUDIO

stream *{STRING}*
Current mode of robot camera/microphone stream. Can be ‘camera,’ ‘camandmic,’ ‘mic,’ or ‘stop’

videosoundmode *{STRING}*
Video encoding mode employed by robot Flash/Actionscript camera/mic capture. ‘LOW’ refers to h263 video with Nelly-Moser audio, and ‘HIGH’ refers to h264 video with SPEEX audio. This varies automatically depending on the capabilities of various server and client operating systems

driverstream *{BOOLEAN}*
TRUE if driver (rtmp client) ‘self’ camera/microphone stream is active

muteOnROVmove *{BOOLEAN}*
TRUE if system is set to mute robot mic on wheel motor movement

volume *{INT}*
(percent) system volume

framegrabbusy *{BOOLEAN}*
TRUE if waiting for single-frame pixel data to be fetched from robot Flash/Actionscript camera/mic capture

THE oculus_settings.txt CONFIG FILE

The oculus_settings.txt file is located under /Oculus/conf/, and contains all settings pertaining to Oculus. This file is generally maintained by the application but a few advanced settings are changed manually. It is split into 3 sections: 'GUI Settings,' 'Manual Settings,' and 'User List.'
GUI settings are set within the application itself, as is the user list. Each line represents a single value, with the setting name, followed by a space, then the value.
If the file is deleted, or 'restore factory settings' is selected from initialize.html, it is re-generated with default values, on application restart.

GUI SETTINGS

skipsetup *yes | no*
indicates whether the server application starts by launching initialize.html, or server.html. After initial setup, typically server.html is always launched

speedslow *{INT}*
(0-255) PWM value used by ArduinOculus to drive the wheel motors at slow speed

speedmed *{INT}*
(0-255) PWM value used by ArduinOculus to drive the wheel motors at medium speed. (speedfast is always 255)

steeringcomp *{INT}*
(0-255) used to set wheel motors steering compensation, 128 is no compensation

camservohoriz *{INT}*
(0-255) mirror tilt servo position at horizontal

camposmax *{INT}*
(0-255) maximum mirror tilt servo position

camposmin *{INT}*
(0-255) minimum mirror tilt servo position

nudgedelay *{INT}*
time in milliseconds for wheel motors nudge move

docktarget *{STRING}*
dock calibration proportions/metrics

vidctroffset *{INT}*
manual dock line offset from center, in pixels

vlow *{INT}*
low video settings width, height, fps, and bandwidth(0-100)

vmed *{INT}*
medium video settings width, height, fps, and bandwidth(0-100)

vhigh *{INT}*
high video settings width, height, fps, and bandwidth(0-100)

vfull *{INT}*
full video settings width, height, fps, and bandwidth(0-100)

vcustom *{INT}*
  custom video settings width, height, fps, and bandwidth(0-100)

vset *{STRING}*
  current video setting

maxclicknudgedelay *{INT}*
  Clicksteer time in milliseconds for wheels to shift video image horizontally from the center to the very left or right edge of the screen

clicknudgemomentummult *{DOUBLE}*
  Momentum multiplier affecting horizontal clicksteer accuracy

maxclickcam *{INT}*
  vertical (mirror tilt) clicksteer setting

muteonrovmove *true | false*
  mutes robot mic on wheel motor movement

videoscale *{INT}*
  scale of web browser client video

volume *{INT}*
  (percent) system volume

holdservo *true | false*
  mirror tilt servo power brakes

loginnotify *true | false*
  controls voice synthesizer announcement of user login

reboot *true | false*
  set to reboot laptop every 48 hours (Windows only)

selfmicpushtotalk *true | false*
  enable/disable push 'T' to un-mute self mic in web browser client

MANUAL SETTINGS

email_smtp_server {STRING}
  SMTP server for outgoing email. 'Disabled' if unused

email_smtp_port {STRING}
  SMTP server port for outgoing email

email_username {STRING}
  username if SMTP server for outgoing email requires authorization. 'Disabled' if unused

email_password {STRING}
  password if SMTP server for outgoing email requires authorization. 'Disabled' if unused. WARNING: PLAIN TEXT

email_from_address {STRING}
  return email address for outgoing email. 'Disabled' if unused

developer *true | false*

enable (alpha) developer features

debugenabled *true | false*
enable verbose logging

commandport {INT}
TCP port for socket client connection. ‘Disabled’ if unused

stopdelay {INT}
time in milliseconds for wheels to come to full stop from max speed

vself {INT}
web browser client self video settings width, height, fps, and bandwidth(0-100)

arduinoculus {STRING}
ArduinOculus microcontroller serial port. Set to ‘discovery’ for auto-detection, ‘disabled,’ or port name

oculed {STRING}
OcuLED Lights serial port. Set to ‘discovery’ for auto-detection, ‘disabled,’ or port name

USER LIST

salt {STRING}
randomly generated key used in password encryption

user0 {STRING}
admin username

pass0 {STRING}
admin password, encrypted

user1..2..3.. {STRING}
additional usernames (non admin)

pass1..2..3.. {STRING}
additional passwords, encrypted (non admin)

ADAPTING AN INCOMPATBILE CHARGER TO WORK WITH OCULUS

If your netbook/laptop is not compatible with one of the 5 DC plug tip sizes, and you're comfortable with doing some basic wiring, you can follow this 10 minute procedure:

You will need:

- Charger compatible with your laptop (use the one that came with your laptop, OR buy an aftermarket one, so you have a 2nd one to use when your laptop is NOT being a robot).
- Wire stripper/cutter
- Pliers and/or small wrench, short phillips driver tip with pliers to hold it
- Voltmeter (optional)
- 1 small zap-strap (optional)
- #6 ring terminals (optional)

Procedure:

1. Cut the charger cable 2 ft (60cm) from where the tip plugs into the laptop.
2. Strip the outer housing away to expose 2" (5cm) of the + and – leads (usually white and black, or red and black respectively). You can ignore any ground leads (sometimes green). Then strip 1/4" (6mm) of the housing off the end of each lead. Do this for each half of the cut cable.
3. Wire the side attached to the AC brick into the charging dock — if you look at the dock PCB, you can wire the + lead into the screw terminal marked 'CTR', and the – lead into the other one, sharing the terminals with the existing wires.
4. Take the other 2ft half and snake it through the Oculus chassis (remove the other one), but make note of which side the WHITE wire of the existing Oculus chassis power lead is attached to: this is the + side.
5. You can attach the leads to fresh #6 ring terminals if you have any, and screw them into the dock contacts, or you can sandwich the bare wire ends under the nuts.
6. Reverse polarity can damage your laptop. Before testing charge, its a good idea to check polarity with a voltmeter if you have one—-positive should be the center pin.