ACS - Aircraft Software Deployment

This document describes how to load firmware and software onto a brand new aircraft, and how to perform updates to existing aircraft.

Prerequisites

Deployment requires a full development ACS installation on a desktop or laptop computer (see install.md). It is assumed that all repositories have already been set to the desired commits / tags / branches. Note that the ACS installer may reset the state of repositories; be sure to confirm that the desired state is set before proceeding, especially with autopilot firmware updates.

Autopilot firmware deployment requires a USB cable from the computer to the autopilot (which uses a Micro-B connector). A serial telemetry radio may also be useful for troubleshooting.

Payload software deployment requires an 802.11n wireless network device. Either the computer, or a separate device, must act as a wireless gateway to the Internet and to the software repository. For initial payload media preparation, an eMMC / microSD card reader is also required.

Preparing a new aircraft

These instructions prepare the autopilot and payload of a brand-new aircraft for use.

Autopilot

The utility setupPixhawkPlane.sh in the acs-env repository (installed to the path by a standard installation) automates most steps to deploying firmware to a new autopilot. It runs without arguments, and presents a textual menu.

The first step is to (B)uild the firmware. This may require Internet access for fetching updated git submodules. Note that, once the correct firmware is built, the utility can flash multiple autopilots without needing to recompile. Be sure that the firmware is built successfully before proceeding to the next step!

Once the firmware is successfully built, (F)lash an aircraft. Select this operation in the utility BEFORE plugging in an autopilot; the utility must catch some of the boot-up text from an underlying mavproxy.py session in order to complete the procedure.

The script will prompt you for an ID for the autopilot; this is the same ID that will be assigned to the payload
(and which affects the IP address of the payload) below. It then begins flashing and configuring the autopilot. The utility will prompt you to unplug and replug the autopilot (resetting it) approximately twice. **NOTE:** do NOT unplug the autopilot during this process except where prompted, unless the utility crashes or freezes for an extended period of time. Recovery instructions are provided below.

Once the firmware has been flashed, the utility will load a default set of parameters (from `~/ACS/acs-env/data/params/ZephyrII_std.annotated.parm`) onto the aircraft and set its ID. After configuration is completed, it will return to the main menu, at which point it is safe to unplug the aircraft. Additional aircraft can be flashed without having to exit the utility.

Note that the aircraft params file is saved to `~/ACS/acs-env/data/params/` at the end of the installation. This is important for updates and recovery from a failed update, as discussed below.

**Payload**

The payload is a bit more involved, because a base deployment must be made directly to the eMMC / microSD media, and then the main installation is performed with the media booted on the payload computer.

It will be necessary to stand up a wireless gateway connecting the aircraft to the Internet and to any internal repositories. The `wifi_config.sh` script can be useful here (see `wireless.md`). All payloads are configured with `192.168.x.1` as their default gateway. On a computer with `eth0` connected to the Internet and wireless device `wlan0`:

```
wifi_config.sh -R eth0 wlan0 1
```

Note that fresh media is configured for team 1 (using the `192.168.2.0/24` network); aircraft can be configured for either team when running the installer.

**Preparing new media**

If starting with new media, first make sure there is a suitable Linux installation. Note that media purchased for use with ODroids generally comes with Linux pre-loaded, and that the `deploy.sh` script used below accounts for nuances of that pre-load. The payload is currently designed around Ubuntu 14.04 LTS; with minor adjustments, newer versions (or other Debian/apt-based distributions) can be used instead.

Once the media is loaded with Linux, connect the media to the laptop or desktop computer, but do not mount the partitions (if partitions are automatically mounted, they should be unmounted by the `deploy.sh` script). Use the following script to expand the root filesystem to the full space available on the media and to place the installer script, `odroid-installer.sh`, into the default user (odroid) home folder:

```
cd ~/ACS/acs_ros_ws/src/autonomy-payload/deploy
sudo ./deploy.sh -I 14.04 /dev/sdb
```

Adjust `/dev/sdb` to point to wherever the media shows up (`dmesg` | `tail` can be useful here). Be sure to omit the partition number (i.e., use `/dev/sdb` not `/dev/sdb2`). The script is designed to allow copying version-specific installer scripts; you can create a new installer as `odroid-installer-VERSION.sh`
Installation of payload software

Next, unplug the media from the desktop/laptop (it should already be unmounted) and connect it to the payload computer. No networking is set up by default, so set up either a physical keyboard and monitor or a console terminal between the payload computer and the laptop/desktop:

```
miniterm.py /dev/ttyUSB0 115200 --1f
```

Connect a wireless device to the payload (this will be used to access the Internet and code repositories through the wireless gateway set up above). Then, power the payload.

Log in (or `su` to) the user account to be used for installation. Note that the installer was written for systems that have a default `odroid` user, but `should` work with other user accounts:

```
su - odroid
```

Then run the installer:

```
./odroid-installer.sh
```

It will prompt you for aircraft ID (valid IDs range from 2-223), team ID (1 or 2), and an aircraft name (any ascii alphanumeric string, up to 16 characters). The ID should be the same as the autopilot ID set above, for a given aircraft. It will then perform the installation (possibly prompting along the way if more information is needed).

The installer attempts to set the payload clock by NTP, so that timestamp-sensitive build processes work correctly. If the NTP query fails, it will prompt for the date in the form `MMDDhhmmYYYY` (e.g., June 23, 2015 at 9:47am would be written `062309472015`).

Once the install completes, the payload can safely be shut down and is ready for use. The payload is now configured for either the team 1 or team 2 network, with an IP address of `192.168.(TEAM+1).ID` and is SSH-enabled. It will also automatically start the payload software on the next boot cycle.

Updating firmware and software

These instructions are for updating an aircraft that has been through the "Preparing a new aircraft" section above.

Autopilot

For this, follow the same instructions as above, except use the (U)pdate option after (B)uilding the latest firmware. See below for recovery steps in the event of a failed update.
Note that this option also deletes all autopilot dataflash logs (see `logs.md` for details).

**Payload**

Commissioned aircraft have payloads with IP addresses of either 192.168.2.x (team 1) or 192.168.3.x (team 2). First, make sure a corresponding gateway is set up (see above section, and note `-2` option to `wifi_config.sh` for team 2 configuration).

Then, power the payload (it is generally easiest to power the entire aircraft) and give the payload time to boot up. It is possible to run the FTI (see `preflight.md` or `launch.md`) to see when the aircraft has reached WAITING_AP or PREFLIGHT status, which means it is fully-booted.

To manually update a payload, SSH in from a computer on the wireless network and run the installer with an update option:

```bash
ssh odroid@192.168.2.16
... perform login; default password is odroid ...
./odroid-installer.sh -u
```

Specify either the `-u` flag to perform a full update (includes core Ubuntu packages) or the `-q` flag to update just the ACS payload software (faster and generally recommended). Note that whatever version of `odroid-installer-VERSION.sh` that `~/odroid-installer.sh` links to is the script that will be run; there is no check that the install script is the latest version. For this reason, a "remediation" file, in `~ACS/acs_ros_ws/src/autonomy-payload/deploy/` is run by the installer after the repositories are updated. This script can be used to change what steps the installer performs.

Note that there are two kinds of log files stored on the payload: ROS logs in `~/ros/log/`, and ROS bags in `~/bags/`. The odroid installer script does not remove these files.

The script `updateOdroidPayload.exp` automates this process and clears all logs. To run it, make sure the computer is on the correct team's wireless network, then run:

```bash
updateOdroidPayload.exp <team> <id>
```

The clock update, SSH login, installer, and log clearing are all handled. Note again that the updater will use whatever version of the odroid installer that is already loaded on the aircraft.

**Advanced topics**

**Reassigning aircraft ID / team**

To reassign an aircraft payload to a different ID or team, use the odroid installer script with the `-c` option:
ssh odroid@192.168.2.47  # Team 1, Aircraft 47
... perform login ...
./odroid-installer.sh -c
... enter new ID, name, and team number ...
Once the aircraft is power cycled, it will have the new configuration on the payload.

If the aircraft ID is changed, then the autopilot ID and telemetry radio NetID should be changed likewise. The payload should set the autopilot ID on startup, but the autopilot ID can be manually changed using mavproxy.py. See the telem_config.sh utility in telemetry.md for details on changing the NetID.

Using cloned payload media

It is possible to configure a base image with payload software installed and then clone the media, saving installation time. The deployment script can be run with the path to an image:

./deploy.sh -C IMAGE_FILE -I VERSION /dev/sdb
You can then attach the cloned media to a payload, connect by serial console or SSH into the IP set when generating the cloned image, and re-run the installer script with the reconfigure flag (which will prompt for a new ID, team, and name, and reconfigure the payload accordingly):

./odroid-installer.sh -c

Creating an image for cloning

An image file can be generated from existing payload media using the dd utility:

sudo dd if=/dev/sdb of=IMAGE_FILE bs=4M
NOTE: Be very careful when using dd, particularly with the if and of arguments. Misuse can corrupt your computer's filesystem!

It is also possible to trim the size of an image by only copying the used portion of the root filesystem; this is somewhat more advanced and involves a bit of math. First, use a utility such as gparted on the media (attached to the desktop/laptop) to shrink the root filesystem to the minimum allowable size (eliminating all free space).

Next, use fdisk to determine the highest-numbered block (counted in 512-byte sectors) in the shrunken root filesystem (assuming that the root filesystem is in the last partition on the media):

sudo fdisk -l /dev/sdb

Disk /dev/sdb: 62.5 GB, 62537072640 bytes
4 heads, 16 sectors/track, 1908480 cylinders, total 122142720 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000c4046

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In the above example, the highest-numbered block is 7704575. Convert to bytes:

7704575 sectors x 512 bytes/sector = 3944742400 bytes

Then divide by the block size you wish to use with dd (e.g., 4M), rounding up to the next whole integer:

3944742400 / (4 * 1024 * 1024) = 940

Finally, use `dd` with the `count` argument:

```
sudo dd if=/dev/sdb of=mycard.img bs=4M count=940
```

**Autopilot update recovery**

The `setupPixhawkPlane.sh` utility uses scripted `mavproxy.py` sessions to perform aircraft configuration. Occasionally the sessions don't go according to the script and the utility may freeze. In the event of a freeze or crash, the following methods can be used to recover and complete the aircraft update.

If the update freezes during one of the mavproxy configuration sessions, it may be possible to unfreeze the script by forcing an autopilot reboot using a telemetry radio (see `telemetry.md` for details) and a separate mavproxy session:

```
telem_config.sh /dev/ttyUSB0 31
mavproxy --master /dev/ttyUSB0 --baudrate 57600
... wait for mavproxy to load and connect ...
MANUAL> reboot
... wait for autopilot to reset, check script, close mavproxy ...
```

If the update crashes or is irrecoverable and must be stopped using Ctrl+C, there is a "recovery mode" that will attempt to complete the update and restore the aircraft configuration. This is done using the (R)etry menu item in `setupPixhawkPlane.sh`. The recovery mode attempts to look up the saved configuration of the aircraft (saved params overwrite the numbered files in `/ACS/acs-env/data/params/`). The script may prompt for the ID of the aircraft to look up the param file. If it finds a param file, recent or not, it will load that file onto the autopilot. Otherwise, it will load the default parameter set (`ZephyrII_std_annotated.parm`) onto the autopilot.