
Makani

Makani was a project to develop a commercial-scale airborne wind turbine, culminating in a flight test of the Makani M600 off the coast of Norway. All Makani software has now been open-sourced. This repository contains the working Makani flight simulator, controller (autopilot), visualizer, and command center flight monitoring tools. Additionally, almost all avionics firmware is also included, albeit potentially not in a buildable state, due to the removal of some third-party proprietary code. We hope that this code will be inspirational and useful to the kite-based windpower and wider communities.

For more information about Makani, please visit <https://x.company/projects/makani/>

A brief tour of the code

- [analysis](#) - Miscellaneous analysis scripts. The most important files here are [analysis/control/crosswind.py](#), which generates the gains for the crosswind inner loop controller, and [analysis/control/generate_hover_controllers.m](#), which generates the gains for the hover controller.
- [avionics](#) - Firmware for the winch, ground station, motors, network, servos, batteries, network switches, GPS, strobe lights, etc.
- [control](#) - The hover, transition-in, crosswind, and off-tether flight controllers.
- [common](#)
- [config](#) - A Python-based configuration system produces a JSON dict specifying all system, controller, and simulation parameters. This is translated into a read-only C structure at compile-time.
- [database](#) - Aero tables, Pitot calibration tables, etc.
- [documentation](#)
- [vis](#) - OpenGL-based visualizer that depicts the state of the system during simulation and real flight.

How to build

This code base was originally designed to run on Linux systems running the Debian Stretch distribution. For the convenience of future users, we are shipping this open source release with a script to create the necessary environment within Docker.

Using Debian Stretch natively or as a VM

1. Run `./lib/scripts/install/install_packages.sh`

-
2. Run `source ~/.bashrc`
 3. Run `cd ${MAKANI_HOME}`
 4. To build everything, run `bbuild_x86`
 5. To test everything, run `btest_all`

Using Docker

We have tested the Docker solution in some Linux systems. We also tested it on macOS but we were unable to get the visualizer running due to an issue with libGL (the rest of the simulation software did work).

1. Run `./docker_build.sh`
2. Run `./docker_run.sh`

Inside of docker:

1. Run `cd ${MAKANI_HOME}`
2. To build everything, run `bbuild_x86`
3. To test everything, run `btest_all`

How to run the Makani flight simulator

1. Run `sudo route add -net 239.0.0.0 netmask 255.0.0.0 dev lo`
2. Run `cd ${MAKANI_HOME}`
3. Run `run_sim -S -M flight`

This command will open the visualizer. If running natively or a VM, and have Chrome installed, the webmonitor should open automatically. If not, open <http://localhost:8000> on your browser.

For a full list of available flags when running the simulator, see the file `sim/run_sim.py`.

Other commands

For a list of preconfigured commands, see `lib/scripts/operator` and `lib/scripts/developer`.

How to read the logs

The Control Telemetry Users' guide, included as a PDF with this distribution, gives a full description of log file data structures. Below are examples for how to load and plot log data using Python or MATLAB.

Python

Here's a small example showing how to load an h5 log file in Python and plot a variable (in this case, the kite's altitude). Accessing the log files is made much less painful by enabling tab-completion of telemetry fields; instructions are in [lib/python/ipython_completer.py](#).

```
1 import h5py
2 import pylab
3
4 log = h5py.File('20161121-142912-flight01_crosswind.h5', 'r')
5 c = (log['messages']['kAioNodeControllerA']
6      ['kMessageTypeControlTelemetry']['message'])
7 pylab.plot(c['time'], -c['state_est']['Xg']['z'])
8 pylab.show()`
```

Additionally, by starting Python with `bazel-bin/lib/bazel/pyembed ipython`, you will be able to access some Makani library functions (like `DcmToAngle`) directly from Python.

You can also explore the field names using `.items()` or `.keys()` and `.dtype` as appropriate:

```
1 log.keys() # Shows [u'bad_packets', u'messages', u'parameters']
2 log['messages'].keys() # Shows [u'kAioNodeBattA', ... ]
3
4 # The number of messages of this type in the h5 file:
5 log['messages/kAioNodeBattA/kMessageTypeSlowStatus'].len()
6
7 # The first message:
8 log['messages/kAioNodeBattA/kMessageTypeSlowStatus'][0]
9
10 # The nested field names:
11 log['messages/kAioNodeBattA/kMessageTypeSlowStatus'][0].dtype`
```

MATLAB

Here's a small example showing three ways to load an h5 log file in MATLAB and plot a variable (in this case, the kite's altitude).

Method 1 A quick way to load a specific telemetry dataset to the workspace using a MATLAB built-in function.

```
1 c = h5read('20161121-142912-flight01_crosswind.h5', '/messages/  
    kAioNodeControllerA/kMessageTypeControlTelemetry');  
2 time = c.message.time;  
3 altitude = -c.message.state_est.Xg.z;  
4 figure;  
5 plot(time, altitude)
```

Method 2 A slower way to load the entire telemetry to the workspace. NOTE: This method only works in MATLAB 2016a and earlier. Find out your MATLAB version by running “ver” on the console. You need the makani repository loaded to your computer.

Open MATLAB and navigate to the following directory on the console:

```
1 $MAKANI_HOME/analysis
```

Run the following script in the MATLAB console to set all relevant paths:

```
1 SetMatlab
```

Now you can run the following code on console to access telemetry data:

```
1 log = h5load('20161121-142912-flight01_crosswind.h5');  
2 c = log('/messages/kAioNodeControllerA/kMessageTypeControlTelemetry');  
3 Time = c.message.time;  
4 Altitude = -c.message.state_est.Xg.z  
5 figure;  
6 plot(Time, Altitude)
```

Method 3 The best of both worlds! A lazy way to loads all datasets quickly. You need the makani repository loaded to your computer.

Open MATLAB and navigate to the following directory on the console:

```
1 $MAKANI_HOME/analysis
```

Run the following script in the MATLAB console to set all relevant paths:

```
1 SetMatlab
```

Now open the H5Plotter (a GUI interface for opening and plotting H5 log data) by running the following in the MATLAB console:

```
1 H5Plotter
```

Load a H5 log file using the ‘Choose’ button in the top right corner. Once the file is loaded, datasets appear in ‘AIO Nodes’ panel box. Click on one or more of these nodes to access the corresponding datasets in the ‘AIO Messages’ panel box. Only datasets common to all selected AIO nodes are shown.

Once you have selected data to plot in the ‘AIO Messages’ panel box, use the ‘plot’ button at the bottom right corner to visualize the data. Holding ctrl or shift allows multiple fields to be selected and selecting a node in the tree will plot all data contained beneath that node. Data can also be exported by right clicking.

NOTE: You can plot multiple datasets simultaneously on the same time axes. How many datasets you can plot at the same time is only limited by your machine’s RAM; be judicious about this.

Matlab Example: Plot roll, pitch, and yaw Here’s a small example that converts the `dcm_g2b` matrix into Euler angles.

```
1 % Read the log file.
2 filename = '20161121-142912-flight01_crosswind.h5';
3 c = h5read(filename, ...
4     '/messages/kAioNodeControllerA/kMessageTypeControlTelemetry');
5
6 % Fetch the dcm_g2b matrix
7 dcm_g2b = c.message.state_est.dcm_g2b.d;
8
9 % Transpose the dcm_g2b matrix.
10 dcm_g2b = permute(dcm_g2b, [2 1 3]);
11
12 % Compute Euler angles.
13 [yaw, pitch, roll] = dcm2angle(dcm_g2b, 'ZYX');
14
15 % Plot the results.
16 plot(c.message.time, roll * 180/pi, '.', ...
17     c.message.time, pitch * 180/pi, '.', ...
18     c.message.time, yaw * 180/pi, '.')
19 legend('roll', 'pitch', 'yaw')
20
21 ylim([-180 180]);
22 set(gca, 'YTick', -180:30:180);
23 grid on;
24 xlabel('controller time [s]');
25 ylabel('angle [degrees]');
26 title(['flight attitude (' filename ')'], 'interpreter', 'none');
```