OpenKITE: User Manual

Petr Listov, Colin Jones

April 26, 2018

OpenKITE [Kite Identification Tracking & Estimation] is being developed at the Automatic Control laboratory EPFL as a part of the European Airborne Wind Energy Systems Optimization and Control (AWESCO). The program was originally aimed to test and validate optimal estimation and control algorithms for the indoor kite prototype built in the laboratory. It has is a number of routines for modeling, simulation as well as model-based nonlinear control, estimation and identification of rigid-wing kites. The software is aimed to reduce development time of both research-oriented and commercial AWE applications.

Modeling

OpenKITE [Kite Identification Tracking & Estimation] package contains implementation of several wind energy kite models:

- ✓ Rigid-wing single line kite [reference to future paper]
- ✓ Kinematic model ("tricycle on a sphere") [reference to Sanket]

Rigid-wing single line kite

The primary interest is the rigid-wing single-line kite model since it has very complex and highly-nonlinear dynamics. OpenKITE allows rather flexible configuration of the flying vehicle inertia and geometry, as well as some of the cable properties. For any particular kite the parametrization can be obtained using CAD and CFD analysis software. In our work we relied on the XFLR5 tool that is capable of performing aerodynamic and stability analysis of the wing for comparable low Reynolds numbers. The following parameters are necessary to specify for a kite simulation:

Geometry:

- b: wing span [m]
- c: mean aerodynamic chord (MAC)[m]
- AR: aspect ratio []
- S: wing surface area [m2]
- lam: taper ratio []
- St: horizontal tail surface area [m2]
- lt: tail level arm [m]
- Sf: fin surface area [m2]
- lf: fin level arm [m]
- Xac: aerodynamic centre [1/c]

Inertia:

- mass: [kg]
- Ixx: [kg * m2]
- Iyy: [kg * m2]

- Izz: [kg * m2]
- Ixz: [kg * m2]

Aerodynamics:

- aerodynamic:
- CL0: lift coefficient with zero angle of attack (AoA) []
- CLa_total: 5.483 total lift coefficient derivative with respect to AoA [1/rad]
- e_oswald: Oswald efficiency number []
- CD0_total: total zero lift drag coefficient []
- CYb: total side force coefficient [1/rad] (in BRF !!!)
- Cm0: zero AoA pitching moment
- Cma: pitching moment coefficient sensitivity wrt to AoA [1/rad]
- Cn0: zero yawing moment for symmetric aircrafts []
- Cnb: yawing moment sensitivity wrt to sideslip [1/rad]
- Cl0: rolling moment for symmetric aircrafts
- Clb: rolling moment sensitivity wrt to sideslip angle [1/rad]
- CLq: pitch-rate lift coefficient [1/rad]
- Cmq: pitch-rate moment coefficient [1/tad]
- CYr: yaw-rate side force effect []
- Cnr: yaw-rate yawing moment effect []
- Clr: yaw-rate rolling moment effect []
- CYp: roll-rate side force coefficient []
- Clp: roll-rate rolling moment coefficient []
- Cnp: roll-rate yawing moment coefficient []
- CLde: lift sensitivity to elevator deflection [1/rad]
- CYdr: side force sensitivity to rudder deflection [1/rad]
- Cmde: pitch moment sensitivity wrt to elevator deflection [1/rad]
- Cndr: yawing moment sensitivity wrt to rudder deflection [1/rad]
- Cldr: rolling moment sensitivity wrt to rudder deflection
- CDde: drag force sensitivity wrt elevator deflection

Tether:

- length: tether length [m]
- Ks: spring coefficient [N/m]
- Kd: damping coefficient [N m / s]
- rx: x-displacement of the tether mouting point wrt CoG [m]
- ry: y-displacement of the tether mouting point wrt CoG [m]
- rz: z-displacement of the tether mouting point wrt CoG [m]

Configuration file has YAML format, an example can be found in the "*data*" folder. There are two ways of using the modelling functionality. User may use the ROS integrated "black-box" simulator and access navigation data by subscribing to specific topic, or get direct access to the model through C++ API.

C++ API of the simulator

Listing (1) shows how the kite model should instantiated using C++ API. It is possible to obtain numerical and symbolic expressions of the right-hand side (RHS) ODE describing kite dynamics, Jacobian and Integrator, that can be used in estimator and controller design.

```
#include "openkite/kite.h"

/** load kite properties */
std::string kite_config_file = "path_to_config_file";
KiteProperties kite_props = kite_utils::LoadProperties(kite_config_file);

/** (optional) specify properties of the built-in integrator */
AlgorithmProperties algo_props;
algo_props.Integrator = RK4;

/** instantiate kite model */
KiteDynamics kite(kite_props, algo_props);

/** get RHS of the ode and Jacobian */
Function ode = kite.getNumericDynamics();
Function jac = kite.getNumericJacobian();
```

Listing 1: Instantiation of the kite model

It is possible then use on the pre-implemented ODE solvers to simulate the system, or to implement a custom integration method. Listing (2) demonstrates this functionality.

```
#include "openkite/kite."h
#include "openkite/integrator."h"

. . ".

KiteDynamics kite(kite_props, algo_props);
Function ode = kite.getNumericDynamics();

    /** compare three ode solvers */
Dict opts;
```

```
opts["tf"]          = 1.0;
opts["poly_order"] = 21;
opts["tol"]         = 1e-4;

/**Create solvers */
/** Runge-Kutte 4ord method without step size control */
opts["method"]      = IntType::RK4;
ODESolver rk4_solver(ode, opts);

/** BDF based integrator from CVODES package *
opts["method"] = IntType::CVODES;
ODESolver cvodes_solver(ode, opts);

/** Chebyshev collocation method: uses Newton method with line search */
opts["method"] = IntType::CHEBYCHEV;
ODESolver chebychev_solver(ode, opts);

/** Pseudospektral Chebyshev collocation: uses IPOPT to solve constraints */
double tf = 1.0;
PSODESolver<10,4,13,3>ps_solver(ode, tf);

/** solve a problem */
DM rk4_sol, cheb_sol, cv_sol, ps_sol;

DM init_state = DM::vertcat({vx, vy, vz, wx, wy, wz, x, y, z, q, qx, qy, qz});
DM control = DM::vertcat({throttle, elevator, rudder, aileron});

rk4_sol  = rk4_solver.solve(init_state, control, tf);
cv_sol   = cvodes_solver.solve(init_state, control, tf);
cheb_sol = chebychev_solver.solve(init_state, control, tf);
ps_sol = ps_solver.solve(init_state, control, FULL);
```

Listing 2: Instantiation of the kite model

ROS kite simulator

Kite simulator can be accessed smiply be calling the command:

```
roslaunch openkite simulator.launch
```

User may edit the launch file to specify the initial point of the simulation:

```
<rosparam param="init_state"> [vx,vy,vz, wx,wy,wz, x,y,z, q,qx,qy,qz]</↩
    rosparam>
```

Configuration file describing the wing and tether properties:

```
<param name="kite_params" value="path_to_configuration_file" />
```

To setup 3D visualisation it is necessary to run RViz from ROS distribution prior to simulator:

```
rosrun rviz rviz
```

Published topics:

+ *kite_state* (sensor_msgs::MultiDOFJointState) : navigation data

Subscribed to topics:

+ *kite_control* (openkite::aircraft_controls) : control input