Notes on Wings and Thrust

Wing Introduction:

Drake includes the ability to incorporate aerodynamic forces into a model through the use of the (Planar)RigidBodyWing class, and is flexible on how these forces are modeled. The programs AVL and Xfoil, written in part by Mark Drela are used to determine proper lift, drag, and moment coefficients for specified 3D airfoils on rectangular wings. The Wing class actually supports four ways of defining an airfoil for the model:

- Empirical data for lift, drag, and moment coefficients. These should be in a .mat file with variable names "CLSpline, CDSpline, CMSpline"
- A 4 or 5-digit NACA airfoil code. i.e. "NACA0012"
- The words "flat plate"
- A .dat file (Xfoil output) specifying x-y points of the surface of the airfoil. See Xfoil's documentation for more information. This file should be located somewhere in the Matlab path.

The properties of the airfoil such as the chord, span, stall angle, and nominal speed are user inputted. The Wing class will display a message during initialization if it detects the wing stalls before the angle the user inputted angle.

Post-stall points:

Before the user-specified stall angle, the lift, drag, and moment coefficients for .dat files and NACA airfoils are calculated by AVL and Xfoil. After the user-specified stall angle, force and moment coefficients are still included all the way to ±180 degrees angle of attack. These are calculated from flat plate theory (which is supported by experimental data taken by Joseph Moore), so there are some inaccuracies due to camber and thickness of the actual airfoil, and the fact that post-stall is extremely difficult to model without taking data in a wind tunnel. However, highly dynamic knife-edge experiments using a "wingeron" aircraft support using this technique as a reasonable starting point in a model such that trajectories and controllers can be developed. Further improvements to the model via system identification on actual data can also be made.

Modeling Flat Plates:

The Wing class makes an improvement to the flat plate model that Rick used for perching which deals with the point of application of lift and drag forces on the wing (pitching moment coefficient). The aerodynamic center of a flat plate airfoil pre-stall is at the quarter-chord,[1] meaning that pre-stall, there is no pitching moment about the quarter-chord point on the plate. However, intuition and research[2] suggests that when a plate is oriented perpendicular to oncoming flow, the zero-moment point on the plate is in the center. This would imply a moment about the quarter-chord point. For any wing, the center of pressure is modeled as moving rearward from the quarter-chord to the middle of the plate starting at stall and ending at 90 degrees angle of attack.

Trick: If you don't want CM's to come into play for flat plates (i.e. Only have lift and drag forces act on the aerodynamic origin as defined in the URDF force_element tag), set the stall angle value in the URDF to 181 degrees. The stall angle is user-defined for flat plates, and since the moment coefficients are only included post-stall, this will set all the moment coefficients to zero.

---

1 https://courses.cit.cornell.edu/mae3050/mae3050ThinAirfoils.pdf

2 Stall flutter and nonlinear divergence of a two-dimensional flat plate wing / John Dugundji, Krishnaswamy Aravamudan. TL570.M41.A25 no.159-6

Generating the Lift, Drag, and Moment splines:

Template files for inputs to AVL and xfoil are provided in the @RigidBodyWing folder. These have properties starting with the $ tag that are replaced by appropriate values calculated from the parameters of the Wing constructor, which originate from the tags in the model's URDF. If for some reason Xfoil and AVL cannot run correctly (the most likely cause being they cannot find a proper input file), then a plaintext warning message will print, but the splines will still be constructed. If AVL fails, then the splines

Both Planar and full 3-D classes of Wing are supported, and these differ almost exclusively in their computeSpatialForce method. The construction of the Wing object is virtually exactly the same.

Other things to note:

- The direction of rotations between standard aerodynamic convention and the coordinate frame used for wings: For a coordinate frame that is directed with X=forwards, Y=out the left of a wing, and Z=up, a positive-Y rotation will be a *pitch downward*. However, this is different than standard aerodynamic convention (which XFOIL and AVL outputs use), which define a pitch up as a positive rotation. This XYZ coordinate frame that the Wing class expects
- RPY support for the wing origin is purposefully excluded. The wing must have the same coordinate axes as its parent body (but not the same xyz origin). If you need rotations, define the parent body with the proper rotations.
- The generated splines are dimensionalized. Lift (and Drag) force is equal to: $L = .5 \cdot C_l(\alpha) \cdot \rho \cdot v^2 \cdot S$ with S=wing area, ρ=air density, v=velocity. That is, the splines include the $.5 \cdot \rho \cdot S$. The moment spline also included the chord length term present in the moment equation. Evaluating a spline at a given angle of attack (α), and multiplying by v^2 will give the appropriate force or torque.

ComputeSpatialForce()

computeSpatialForce returns f_ext (a potentially sparse matrix with manip.getNumBodies columns) and B (a (nq x nu) matrix which contributes a control-affine term + B(q,qd)*u to the manipulator dynamics). Typically one of these will be a matrix of zeros while the other will be nonzero, depending on whether the force is derived from the state (springs, dashpots, wings) or the force is derived from an input (Thrust). The direct_feedthrough_flag can indicate which is the case, and is true for the latter. Check out manipulatorDynamics to see exactly how the f_ext and B_mod are included to the manipulator equations. F_ext gets passed through to the Featherstone model (after a re-mapping to ensure the columns in f_ext are in the proper order that the Featherstone engine expects), while the B_mod simply gets added to the B matrix.
The comments in Bug 1649, and the section below can also be useful in understanding Thrust elements.

Thrust components:

See the Drake documentation for details on how Thrust elements are defined in a URDF. Thrust components have an xyz orientation and XYZ direction which define the point and orientation of the applied force. These vectors should be normalized, and the scaleFactor should be used to translate the input command into Newtons of force. The Thrust class works by returning a B_modifications matrix that is added to B to properly capture the input force's effects on the dynamics of the robot.

Updating Plane.URDF

Whenever the plane model is updated significantly due to moving around weight, increasing the size of wings, or anything else that affects the dynamics or collision geometry, the URDF should be updated.

Some simple changes could be made manually if you know exactly how the model changed, but anything that changes the inertial properties should probably go through the Solidworks Export process again:

1. Go through the steps to export the URDF from the Solidworks model in the Drake documentation.
2. Put the meshes generated in the appropriate location, and make sure the <visual> tags all reference the correct .obj meshes. Copy the material colors from the old URDF to the new URDF if desired.
3. Copy all the collision tags from the old model into the new URDF. Update these as necessary if the geometry of the plane changed.
4. Copy all the <force_element> tags from the old model to the new model. If the geometry of the plane didn't change, you shouldn't have to change anything in these tags. If the chord, span, etc of any of the wings or surfaces changed, then update the corresponding force tag as necessary.

Example: You move the motor 4 cm forwards on the plane, and update the SolidWorks accordingly. You export the new URDF from the updated model, and then make the following changes to the generated URDF:

- Update mesh location for visual tag (and convert meshes to .obj files if necessary)
- Copy all collision properties from old plane. Update the Fuselage collision box because the plane is now 4 cm longer (make the box 4 cm longer (x dimension), and move the collision reference origin. This collision reference origin may or may not move by 2 cm, depending on how the fuselage moved relative to the defined model origin when you exported from SolidWorks.) The collision tags for the other links should not have to be modified.
- Copy all the force elements from the old plane. The wing elements should not have to change, but the xyz origin of the Propellers thrust element should have to be moved forward.

When in doubt, if you're not sure where to place collision or force_element reference origins, set the <visual> tag to Origin:<xyz = “0 0 0”/>, Geometry:<sphere radius =“.005”/>, and observe where the link's origin is. Then grab a ruler, the actual airplane, and figure out where to center the box such that it is in the location of the corresponding surface. Modify the box in the <visual> tag so you can see what is happening, and when you're happy, switch it over to the <collision> tag, and reset the <visual> tag to Origin: <xyz = “0 0 0”/> and Geometry pointing to the appropriate mesh.

Growth of state space from using URDFs:

Many of the aircraft originally simulated employ position control for their control surfaces—This is currently not supported from a URDF. This means that any actuated surfaces will add 2 extra DOF to the model as well as the input required for the actuator. Velocity-controlled joints should add 1 extra DOF on top of the input required. With the inertial properties known, the proper torque can be computed for a desired input position or velocity. Or, the expanded-state model can be used in simulation, as position control of the joints on the actual aircraft is effectively the expanded-state model with high-gain controllers on top of some states to drive them to the commanded values very quickly.

Debugging

If you need to debug the Wing class, it is recommended to step through the system commands that the constructor generates, manually run these, and check their output. There can be some quirks (such as Xfoil cannot handle filenames longer than 64 characters) that might be otherwise hard to track down. Also, check the input and output files that get generated in Matlab's temporary directory to make sure they look okay.

For forces and moments, keep in mind that Xfoil/AVL treat a positive pitch up, while drake will usually treat a positive pitch as a pitch down for a standard X-forward, Z-up axis configuration. Check the jsign property in Planar robots, though.

References and Useful Papers:

C.C. Critzos, H.H. Heyson, R.W. Boswinkle Jr. “Aerodynamic characteristics of NACA 0012 airfoil at angles of attack from 0 to 180 degrees”. NACA TN 3361, 1955

R.E. Sheldahl, P.C. Klimas. “Aerodynamic Characteristics of Seven Symmetrical Airfoil Sections Through 180-Degree Angle of Attack for Use in Aerodynamic Analysis of Vertical Axis Wind Turbines. Report SAND80-2114, Sandia Laboratories, Albuquerque, March 1981.

Stall flutter and nonlinear divergence of a two-dimensional flat plate wing / John Dugundji, Krishnaswamy Aravamudan. TL570.M41.A25 no.159-6

Relevant (resolved) Bugs in Bugzilla:

- 1168—Discussions on how to add aerodynamic forces to URDF's
- 1617—Issues with AVL/Xfoil interaction
- 1625—Simulating the MURI plane in Drake
- 1649—Add thrust capabilities to Drake tools
- 1699—Planar Glider from URDF: dynamics and dircol perching test.
- 1734—Xfoil filenames cannot be longer than 64 characters
- 1735—Mass of elevator in Glider.URDF is significant, as well as pitching moment.
- 1531—Merging URDF fixed joints creates problems with force elements

Notes written by Tim Jenks, and last updated on 08/22/13. With any questions, feel free to contact him at trjenks@gmail.com.