Observability Analysis: Rotary-Wing UAV with Aerodynamic Drag

Spencer Folk

System Parameters

Inertia, Mass, and Gravity

In this version of the dynamics, we consider a body frame (B) and inertial frame (W). Without loss of generality, we assume the body frame is the principal axes such that the inertia tensor is diagonal.

In[1158]:=

```
j = {jx, jy, jz}; (* Measured a priori,
either using CAD or something more complicated *)
jmat = DiagonalMatrix[j]
jmatInv = Inverse[jmat];
```

Out[1159]=

```
{{jx, 0, 0}, {0, jy, 0}, {0, 0, jz}}
```

In[1161]:=

```
m; g; (* Measured a priori *)
```

Motor Coefficients

In[1162]:=

```
τm; (* first order time constant of the motor response *)
```

Propeller Coefficients

In[1163]:=

```
kη; (* Thrust coefficient *)
km; (* Yaw moment coefficient *)
kd;  (* Rotor drag coefficient *)
kz;  (* Loss of thrust due to inflow, inflow coefficient *)
kh; (* Translational lift coefficient *)
```

Propeller Position Vectors

These vectors are written in the body frame (note the B in front of each vector). Since we choose the body frame, we can assume that these values can be easily measured in CAD and therefore known.

Without loss of generality, we consider a planar symmetric configuration.

In[1168]:=
```
l; d;
```

In[1169]:=
```
BpBR1 = {l, 0, d};
BpBR2 = {0, l, d};
BpBR3 = {-l, 0, d};
BpBR4 = {0, -l, d};
```

Propeller Rotation Directions

These reflect the direction of each motor,observability result should be invariant to these but it’s here for completeness. These results should hold without loss of generality.

In[1173]:=
```
ϵ = {1, -1, 1, -1};
```

IMU Position Vector and Rotation

The position and rotation of the IMU frame with respect to the body frame. Without loss of generality, choose identity with zero offset from CoM.

In[1174]:=
```
BpBI = {0, 0, 0}; (* xyz position of the IMU w.r.t. the body axes,
in which case we assume to be known *)
ψI = 0; (* Yaw angle of IMU frame from body axes*)
IRB = {{Cos[ψI], -Sin[ψI], 0}, {Sin[ψI], Cos[ψI], 0}, {0, 0, 1}}
 (* Recall that we assume that IMU z axis and principal z axis are parallel *)
```

Out[1176]=
```
{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

IMU biases

Typically these biases can be calibrated at the beginning of the flight.

In[1177]:=
```
ba = {bax, bay, baz};
bg = {bgx, bgy, bgz};
```

System States

Inertial Position

In[1179]:=
```
Wpos = {xpos, ypos, zpos};
```

Body Attitude (Represented with Euler Angles)

The body attitude is represented using Euler angles.

In[1180]:=
```
θ; (* Pitch angle about the y axis *)
ψ;  (* Yaw angle about the z axis *)
ϕ; (* Roll angle about the x axis *)
Θ = {ϕ, θ, ψ};
```

Construct the elementary rotation matrices

In[1184]:=
```
Rϕ = {{1, 0, 0}, {0, Cos[ϕ], -Sin[ϕ]}, {0, Sin[ϕ], Cos[ϕ]}};
```

In[1185]:=
```
Rθ = {{Cos[θ], 0, Sin[θ]}, {0, 1, 0}, {-Sin[θ], 0, Cos[θ]}};
```

In[1186]:=
```
Rψ = {{Cos[ψ], -Sin[ψ], 0}, {Sin[ψ], Cos[ψ], 0}, {0, 0, 1}};
```

Now construct the rotation from the inertial frame (W) to the body frame (B). This is a ZYX rotation.

In[1187]:=
```
WRB = Rψ.Rθ.Rϕ; MatrixForm[WRB]
```

Out[1187]//MatrixForm=

$$\begin{pmatrix} \mathrm{Cos}[\theta]\,\mathrm{Cos}[\psi] & \mathrm{Cos}[\psi]\,\mathrm{Sin}[\theta]\,\mathrm{Sin}[\phi]-\mathrm{Cos}[\phi]\,\mathrm{Sin}[\psi] & \mathrm{Cos}[\phi]\,\mathrm{Cos}[\psi]\,\mathrm{Sin}[\theta]+\mathrm{Sin}[\phi]\,\mathrm{Sin}[\psi] \\ \mathrm{Cos}[\theta]\,\mathrm{Sin}[\psi] & \mathrm{Cos}[\phi]\,\mathrm{Cos}[\psi]+\mathrm{Sin}[\theta]\,\mathrm{Sin}[\phi]\,\mathrm{Sin}[\psi] & -\mathrm{Cos}[\psi]\,\mathrm{Sin}[\phi]+\mathrm{Cos}[\phi]\,\mathrm{Sin}[\theta]\,\mathrm{Sin}[\psi] \\ -\mathrm{Sin}[\theta] & \mathrm{Cos}[\theta]\,\mathrm{Sin}[\phi] & \mathrm{Cos}[\theta]\,\mathrm{Cos}[\phi] \end{pmatrix}$$

And get the inverse, which rotates a vector in the inertial frame (W) to the body frame (B)

In[1188]:=
```
BRW = Transpose[WRB]; MatrixForm[BRW]
```

Out[1188]//MatrixForm=

$$\begin{pmatrix} \mathrm{Cos}[\theta]\,\mathrm{Cos}[\psi] & \mathrm{Cos}[\theta]\,\mathrm{Sin}[\psi] & -\mathrm{Sin}[\theta] \\ \mathrm{Cos}[\psi]\,\mathrm{Sin}[\theta]\,\mathrm{Sin}[\phi]-\mathrm{Cos}[\phi]\,\mathrm{Sin}[\psi] & \mathrm{Cos}[\phi]\,\mathrm{Cos}[\psi]+\mathrm{Sin}[\theta]\,\mathrm{Sin}[\phi]\,\mathrm{Sin}[\psi] & \mathrm{Cos}[\theta]\,\mathrm{Sin}[\phi] \\ \mathrm{Cos}[\phi]\,\mathrm{Cos}[\psi]\,\mathrm{Sin}[\theta]+\mathrm{Sin}[\phi]\,\mathrm{Sin}[\psi] & -\mathrm{Cos}[\psi]\,\mathrm{Sin}[\phi]+\mathrm{Cos}[\phi]\,\mathrm{Sin}[\theta]\,\mathrm{Sin}[\psi] & \mathrm{Cos}[\theta]\,\mathrm{Cos}[\phi] \end{pmatrix}$$

Velocity (in both frames)

In[1189]:=
```
Wvel = {xdot, ydot, zdot};
Bvel = {Bxdot, Bydot, Bzdot};
```

In[1191]:=
```
Ω = {p, q, r}; (* The rates expressed in the Body Frame!! *)
```

Wind Vector (in both frames)

In[1192]:=
```
Wwind = {wx, wy, wz};
Bwind = {Bwx, Bwy, Bwz};
```

Motor Speeds

In[1194]:=
```
η = {η1, η2, η3, η4};
```

Control Forces and Moments

Commanded Motor Speeds

In[1195]:=
```
u = {u1, u2, u3, u4};
```

Rotor Thrusts

In[1196]:=
```
thrusts = kη * (η^2)
```

Out[1196]=

$\{k\eta\, \eta 1^2, k\eta\, \eta 2^2, k\eta\, \eta 3^2, k\eta\, \eta 4^2\}$

In[1197]:=
```
BTR1 = {0, 0, thrusts[[1]]};
(* These are the individual thrust vectors in the body frame coordinates *)
BTR2 = {0, 0, thrusts[[2]]};
BTR3 = {0, 0, thrusts[[3]]};
BTR4 = {0, 0, thrusts[[4]]};
```

Rotor Yaw Moments

In[1201]:=
```
dragmoment = ∈ * km * (η^2)
```

Out[1201]=

$\{km\, \eta 1^2, -km\, \eta 2^2, km\, \eta 3^2, -km\, \eta 4^2\}$

In[1202]:=
```
ByawmomentR1 = {0, 0, dragmoment[[1]]};
(* These are the drag moments in the body frame coordinates *)
ByawmomentR2 = {0, 0, dragmoment[[2]]};
ByawmomentR3 = {0, 0, dragmoment[[3]]};
ByawmomentR4 = {0, 0, dragmoment[[4]]};
```

Control Wrench

The total moments cause by each rotor is the drag moment + the moment arm

In[1206]:=
```
BcontrolmomentR1 = BpBR1 × BTR1 + ByawmomentR1;
BcontrolmomentR2 = BpBR2 × BTR2 + ByawmomentR2;
BcontrolmomentR3 = BpBR3 × BTR3 + ByawmomentR3;
BcontrolmomentR4 = BpBR4 × BTR4 + ByawmomentR4;
```

In[1210]:=

```
Btotcontrolmoment =
  BcontrolmomentR1 + BcontrolmomentR2 + BcontrolmomentR3 + BcontrolmomentR4 ;
(* This is the total moment expressed in the body frame! *)
Btotcontrolforce = {0, 0, Total[thrusts]};
```

Aerodynamic Forces and Moments

Aerodynamic forces consist of rotor drag, loss of thrust due to inflow, and parasitic (frame) drag. These are all defined in the body frame. Note that we are assuming that each propulsive unit sees the same airspeed (i.e. small body rates) and that they have the same describing coefficients (rotor drag, blade flapping, etc.). This enables us to compute the rotor drag faster.

Airspeed Calculation

In[1212]:=

```
Wva = Wvel - Wwind;
Bva = Bvel - Bwind;
airspeed = (Bva⟦1⟧^2 + Bva⟦2⟧^2 + Bva⟦3⟧^2)^(1/2);
vh2 = Bva⟦1⟧^2 + Bva⟦2⟧^2
```

Out[1215]=

$$(-Bwx + Bxdot)^2 + (-Bwy + Bydot)^2$$

Rotor Drag

In[1216]:=

```
BrotordragR1 = -η1*DiagonalMatrix[{kd, kd, kz}].Bva;
BrotordragR2 = -η2*DiagonalMatrix[{kd, kd, kz}].Bva;
BrotordragR3 = -η3*DiagonalMatrix[{kd, kd, kz}].Bva;
BrotordragR4 = -η4*DiagonalMatrix[{kd, kd, kz}].Bva;
```

In[1220]:=

```
BH = BrotordragR1 + BrotordragR2 + BrotordragR3 + BrotordragR4;
```

Moment Due to Rotor Drag

In[1221]:=

```
BrotordragmomentR1 = BpBR1 × BrotordragR1;
BrotordragmomentR2 = BpBR2 × BrotordragR2;
BrotordragmomentR3 = BpBR3 × BrotordragR3;
BrotordragmomentR4 = BpBR4 × BrotordragR4;

Btotrotordragmoment =
  BrotordragmomentR1 + BrotordragmomentR2 + BrotordragmomentR3 + BrotordragmomentR4;
```

Translational Lift

In[1226]:=

```
BTL = {0, 0, 4 * kh * vh2};
```

Total Aerodynamic Wrench

In[1227]:=

```
Btotaeroforce = BH + BTL;
Btotaeromoment = Btotrotordragmoment;
```

Total Forces and Moments

Compute the total forces and moments in the body frame

In[1229]:=

```
Btotforce = Btotcontrolforce + Btotaeroforce;
Wtotforce = WRB.Btotforce;
```

In[1231]:=

```
Btotmoment = Btotcontrolmoment + Btotaeromoment;
```

Filter States

In[1232]:=

```
constants = {kη, km, kd, kz, kflap, τm};
```

In[1233]:=

```
x = Join[Bvel, Θ, Ω, η, Bwind]
```

Out[1233]=

```
{Bxdot, Bydot, Bzdot, ϕ, θ, ψ, p, q, r, η1, η2, η3, η4, Bwx, Bwy, Bwz}
```

In[1234]:=

In[1235]:=

```
n = Length[x]
```

Out[1235]=

```
16
```

In[1236]:=

```
statevec = x;
inputvec = u;
```

Process Model

Kinematics

In[1238]:=
```
Wposdot = Wvel;
MatrixForm[Wposdot];
```

In[1240]:=
```
Θdot = {{1, Sin[ϕ] * Tan[θ], Cos[ϕ] * Tan[θ]},
    {0, Cos[ϕ], -Sin[ϕ]}, {0, Sin[ϕ] * Sec[θ], Cos[ϕ] * Sec[θ]}}.Ω;
MatrixForm[Θdot];
```

Rotational Dynamics

In[1242]:=
```
Ωdot = jmatInv.(Btotmoment - Cross[Ω, jmat.Ω]);
MatrixForm[Ωdot];
```

Translational Dynamics

In[1244]:=
```
Wveldot = (1 / m) * (Wtotforce + {0, 0, -m * g});
MatrixForm[Wveldot];
```

In[1246]:=
```
Bveldot = (1 / m) * (Btotforce + BRW.{0, 0, -m * g}) - Cross[Ω, Bvel];
MatrixForm[Bveldot];
```

Motor Dynamics

In[1248]:=
```
ηdot = (1 / τm) * (u - η);
MatrixForm[ηdot];
```

Wind Dynamics

In[1250]:=
```
Wwinddot = ConstantArray[0, 3];
Bwinddot = ConstantArray[0, 3];
MatrixForm[Bwinddot];
```

Parameter Dynamics

Assume that the parameters do not change over time.

In[1253]:=
```
f = Join[Bveldot, Θdot, Ωdot, ηdot, Bwinddot];
```

In[1254]:=
```
(*For[i=0,i≤Length[x],i++,Print[f⟦i⟧]]*)
```

In[1255]:=
```
(*For[i=0, i≤Length[x], i++,Print[Grad[f,x][[i]]]*)
```

In[1256]:=
```
vecfield = f;
```

Measurement Model

The measurement model includes the gyro and accelerometer measurements.

In[1257]:=
```
aAct = (1 / m) * Btotforce;
```

In[1258]:=
```
hAcc = IRB.aAct;
hGyro = IRB.Ω;
hVel = Bvel;
hPos = Wpos;
hAttitude = Θ;
hMotor = η;
```

In[1264]:=
```
h = Join[hAcc, hGyro, hVel, hAttitude, hMotor];
```

In[1265]:=
```
outputvec = h;
```

In[1266]:=
```
(*For[i=0,i≤15,i++,Print[h[[i]]]*)
```

In[1267]:=
```
(*For[i=0, i≤Length[h], i++,Print[Grad[h,x][[i]]]*)
```

Observability Analysis

Automated Analysis

In[1268]:=
```
lieseq = {{}, {0}, {1}, {2}, {3}, {4}};
(* Which Lie derivatives to take. Empty set is the output function itself *)
```

In[1269]:=
```
rule0 = Table[inputvec[[i]] → 0, {i, 1, Length[inputvec]}];
```

In[1270]:=
```
vecfield0 = vecfield /. rule0;
```

In[1271]:=
```
rules = Table[Table[inputvec[[j]] → KroneckerDelta[i, j], {j, 1, Length[inputvec]}],
   {i, 1, Length[inputvec]}];
```

In[1272]:=
```
cavecfields = Table[vecfield - vecfield0 /. rules[[i]], {i, 1, Length[inputvec]}];
```

In[1273]:=
```
vecfields = Join[{vecfield0}, cavecfields];
```

In[1274]:=
```
LieIterate[h_, i_] := Grad[h, statevec].vecfields〚i + 1〛;
```

In[1275]:=
```
LieDerivative[h_, seq_] := Fold[LieIterate, h, seq];
```

In[1276]:=
```
lies = Table[LieDerivative[outputvec, lieseq〚i〛], {i, 1, Length[lieseq]}];
```

In[1277]:=
```
obsvec = DeleteCases[Flatten[lies], 0]; MatrixForm[obsvec];
```

In[1278]:=
```
obsmat = Grad[obsvec, statevec];
```

In[1279]:=
```
params = statevec;
```

In[1280]:=
```
psub = Table[i, {i, 1, Length[params]}];
```

In[1281]:=
```
psubrules = Table[params〚i〛 → psub〚i〛, {i, 1, Length[params]}];
```

In[1282]:=
```
obsmateval = obsmat /. psubrules;
```

In[1283]:=
```
(*NullSpace[obsmateval];*)
```

In[1284]:=
```
MatrixRank[obsmateval]
```

Out[1284]=
```
16
```

In[1285]:=
```
Length[statevec]
```

Out[1285]=
```
16
```