The item to be cleared is a low-fidelity software simulation model of a hypothetical free-flying robot designed for use in zero gravity environments.

This simulation model works with the HCC simulation system that was developed by Xerox PARC and NASA Ames Research Center. HCC has been previously cleared for distribution.

When used with the HCC software, the model computes the location and orientation of the simulated robot over time. Failures (such as a broken motor) can be injected into the simulation to produce simulated behavior corresponding to the failure.

Release of this simulation will allow researchers to test their software diagnosis systems by attempting to diagnose the simulated failure from the simulated behavior.

This model does not contain any encryption software nor can it perform any control tasks that might be export controlled.
This file is the HCC code for the PSA simulation. PSA was officially
renamed to SMR, but the name PSA has stuck for the project. Both names
are used for the project, and both might be around in code below. */

/*define constants used in code below */
define m 5 // mass of the PSA kg
define rad 0.1 // radius of the PSA m

define constants used to control behavior of HCC */
/* see HCC programmer's manual for explanation of these */
/**MEMORY_MAX 8000000
/**EPSILON 1.0e-11
/**INTEGRATION_INIT 0.0001
/**SAMPLE_INTERVAL_MIN 0.01
/**SAMPLE_INTERVAL_MAX 0.01 // Good number = 0.01

class CFan {

/* Description of variables:
k1 Curvefit constant for thrust of fan as
function of fan speed
k2 Curvefit for modeling aerodynamic torque
on motor as function of fan speed
inertia Rotational inertia of fan along spin axis
tm Torque constant of motor
    (torque = tm * current)
vel Angular velocity of fan (rad/s)
current Current sent to motor by controller
force Thrust on PSA due to fan
moment Total torque provided by motor
aero_torque Torque of fan used to provide thrust,
    overcome drag, overcome friction, etc.
h Angular momentum of fan */

public interval vel, force, moment, aero_torque, h;

public CFan(interval i, interval k1, interval k2, interval inertia,
    interval tm, interval current) {

    vel = 0.0;

    always { /* Fan model with fan dynamics. k1 and k2 are two constants
determined empirically and passed in to the object. This model has the torque created by the fan motor being
I always
/
1_moment:=(F5.force-F4.force)*radius - F2.aero_torque - F3.aero_torque;
  m_moment:=(F1.force-F0.force)*radius - F4.aero_torque - F5.aero_torque;
  n_moment:=(F3.force-F2.force)*radius - F0.aero_torque - F1.aero_torque;
/
/* Balance of moments in each component direction, used
to find angular velocities (p, q, r). */
  p' := (1_moment - F2.h' - F3.h' + F4.h*r + F5.h*r
        - F0.h*q - F1.h*q) / mom_of_in;
  q' := (m_moment - F4.h' - F5.h' - F2.h*r + F3.h*r
        + F0.h*p + F1.h*p) / mom_of_in;
  r' := (n_moment - F0.h' - F1.h' + F2.h*q + F3.h*q
        - F4.h*p - F5.h*p) / mom_of_in;
/
/* Calculate orientation based on (p, q, r). Orientation is
  in quaternion form. */
  q1' := (q4*p-q3*q+q2*r)/2;
  q2' := (q3*p+q4*q-q1*r)/2;
  q3' := (-q2*p+ql*q+q4*r)/2;
  q4' := (-ql*p-q2*q-q3*r)/2;
/
/* Calculate total force in each component direction */
  x_force:=F2.force + F3.force;
  y_force:=F4.force + F5.force;
  z_force:=F0.force + F1.force;
/
/* Balance of forces in each component direction, used
to find (u, v, w). Note the coriolis forces added as a
result of the body-fixed coordinate system used in the
force balance. */
  u' := (x_force)/mass-2*(q*w-r*v);
  v' := (y_force)/mass-2*(r*u-p*w);
  w' := (z_force)/mass-2*(p*v-q*u);
/
/* Convert the body-fixed velocities to global velocities
  and integrate to find global position (x, y, z) */
  x' := u*(1-2*q2^2-2*q3^2)+v*q1*(q1*q2-q3*q4)+w*q2*(q3*q1+q2*q4;
  y' := u*2*(q1*q2+q3*q4)+v*2*(1-2*q3^2-2*q1^2)+w*2*(q2*q3-q1*q4;
  z' := u*2*(q3*q1-q2*q4)+v*2*(q2*q3+q1*q4)+w*(1-2*q1^2-2*q2^2);
#endif EXTERNAL_CONTROLLER
  junk = update_state(time, u, v, w, p, q, r/*, u', v', w', p', q', r'*/);
#endif
/* Apply a conversion factor to calculate the motor current for each fan, based on the desired thrust of the fan. This current is then sent to the fan of the PSA. */
fan0current = 8.19 * fan0force; //3.631
fan1current = 8.19 * fan1force;
fan2current = 8.19 * fan2force;
fan3current = 8.19 * fan3force;
fan4current = 8.19 * fan4force;
fan5current = 8.19 * fan5force;
} // end always
#endif

} //end CController constructor

} // end CController class def.

().'/*****************'*/
class CPlanner {
    public interval xdes, ydes, zdes;
    public interval roll_des, pitch_des, yaw_des;

    public CPlanner() {
        always {
            /* Set all values to 0 as the default, for all times where they are not defined in the list below. */
            /* I had a problem with this working - after 43 seconds, these values end up being undefined. HCC won't crash, but the fan's force values become undefined, and the PSA will coast in the last direction it is facing. */
            unless ((time >= 0) || (time <= 43)) {
                xdes = 0.0; ydes = 0.0; zdes = 0.0;
                roll_des = 0.0; pitch_des = 0.0; yaw_des = 0.0;
            }
            if ((time >= 0.0) && (time < 1.0)) {
                xdes = 0.0; ydes = 0.0; zdes = 0.0;
                roll_des = 0.0; pitch_des = 0.0; yaw_des = 0.1;
            }
        } /* Send PSA forward */
    }
}
```cpp
fan0 = new CFan(1.0, 4.0e-9, 2.0e-10, 1.0e-7, 0.0004, controller.fan0current);
fan1 = new CFan(2.0, 8.0e-8, 2.0e-10, 1.0e-7, 0.0004, controller.fan1current);
fan2 = new CFan(3.0, 8.0e-8, 2.0e-10, 1.0e-7, 0.0004, controller.fan2current);
fan3 = new CFan(4.0, 8.0e-8, 2.0e-10, 1.0e-7, 0.0004, controller.fan3current);
fan4 = new CFan(5.0, 8.0e-8, 2.0e-10, 1.0e-7, 0.0004, controller.fan4current);
fan5 = new CFan(6.0, 8.0e-8, 2.0e-10, 1.0e-7, 0.0004, controller.fan5current);

#else
interval junk = init_external_controller();
#endif

/****************************
Section to change depending on if using visualization
****************************/

#ifndef VISUALIZATION
/*Global variable*/
CPSA psa0;
psa0 = new CPSA(0, 0, 0, 0, 0, 0, 1);
#endif
```