This pseudo-code description was created for the 2nd Workshop on Theorem Proving in Certification to be held December 5 - 6, 2011 in Cambridge, UK. It has been "seeded" with some intentional defects and other aberrations for the purposes of evaluating and demonstrating the effectiveness of formal analysis as a means of verifying software. All uses of this pseudo-code or variants of this pseudo-code should acknowledge this workshop as the original source.

```
/* ================================================================
   UpdateNGVelocity () - Nose Gear Velocity Estimate Update Function

   Created 2010.07.16
   Updated 2011.02.12
   Updated 2011.08.04

   This function is invoked by the scheduler at least once every 500
   millisecs while the weight-on-wheels condition is true.  It reads
   three global variables (NGClickTime,NGRotations and Millisecs)and
   updates two other global variables (estimatedGroundVelocity and
   estimatedGroundVelocityIsAvailable).

   The minimum measurable velocity is 3 km/hr.  It is not safe to
   use the output of this function to determine that the aircraft
   is stopped.

   Requirement #1:  When the estimatedGroundVelocityIsAvailable flag
   is true (i.e., this global variable is set to a non-zero value),
   the value of the global variable estimatedGroundVelocity shall be
   within 3 km/hr of the true velocity of the aircraft at some moment
   within the past 3 seconds. This requirement may be expressed in a
   semi-formal manner as follows (where behaviour is modelled at the
   millisecond time scale):

      FORALL t,
        IF (estimatedGroundVelocityIsAvailable at time (t + 3000))
        THEN EXISTS n, m SUCH THAT:
         ((1 <= n) AND (n <= 3000)) AND
         ((-3 <= m) AND (m <= 3)) AND
         (estimatedGroundVelocity at time (t + 3000) =
           (trueGroundVelocity at time (t + n)) + m)

   This property must hold if all of the following assumptions are
   true:

   1. The sensor detects the completion of a full rotation of the nose
   gear wheel with a maximum error of 1 cm, i.e., there is a maximum
   variation of +/- 1 centimeter in the location of a fixed position
   on the perimeter of the wheel between the times when a "click"
   is signaled to the computer.

   2.  The two global variables, NGClickTime and NGRotations, are
   updated no more than 2 milliseconds after the sensor detects the
   completion of a full rotation of the nose gear wheel.

   3. The maximum change in velocity (to be tolerated by this function)
   is 20 meters per second per second at speeds above 150 km/hr and no
```

```
   more than 10 meters per second per second at lower speeds.   The
   maximum tolerable jerk (i.e., derivative of acceleration) is 3 meters
   per second cubed.

   4. The wheel diameter is between 12 and 50 inches.

   5. No other part of the software is capable of modifying the values
   of estimatedGroundVelocityIsAvailable and estimatedGroundVelocity.

   6. The aircraft is moving at least 3km/hr.

   7. Once invoked, this update function runs to completion, or at least
   the global variables updated by this function will not be read by any
   any other part of the software between the time when this function is
   entered and when its execution is completed.

   Requirement #2: If all of the above assumptions are true, the
   estimatedGroundVelocityIsAvailable flag shall be true if the values
   of the three global variables NGClickTime, NGRotations and Millisecs
   have been valid when accessed during the two previous invocations of
   this function.
*/

/* SYSTEM ADAPTATION PARAMETERS */

/* WHEEL_DIAMETER is the approximate diameter of the nose gear wheel,
   in inches (accurate within 1 inch of true diameter).
*/

#define WHEEL_DIAMETER 26

/* MIN_SPEED is the minimum measureable speed, in km/hr */

#define MIN_SPEED 3

/* MAX_SPEED is an upper bound on the range of measurable velocity
   while the aircraft is moving on the ground, in km/hr.
*/

#define MAX_SPEED 600

/* MAX_ACCEL is an upper bound on the rate at which the measured speed
   may increase or decrease, in meters per second squared. Note: just
   a quantity, i.e., always a positive value regardless of whether the
   speed is increasing or decreasing.
*/

#define MAX_ACCEL 20

/* MAX_NUM_FAILED_UPDATES is an upper bound on the number of times
   that attempts to update the estimated ground speed can fail
   consecutively (because the estimated velocity or estimated
   (de-)acceleration rate is too high to be valid).
*/

#define MAX_FAILED_UPDATES 5
```

```
/* GLOBAL VARIABLES */

/* NGClickTime (unsigned 16 bits) is the approximate system clock
   time when the nose gear wheel most recently completed a full
   rotation causing NGRotations to be incremented.
*/

extern unsigned NGClickTime;

/* NGRotations (unsigned 16 bits) is the number of completed full
   rotations of the nose gear, i.e., it is incremented when each full
   rotation is completed.
*/

extern unsigned NGRotations;

/* Millisecs (unsigned 16 bits) is the number of milliseconds since
   the start of system execution, i.e., system clock time.
*/

extern unsigned Millisecs;

/* estimatedGroundVelocityIsAvailable (unsigned 16 bits) is
   non-zero if and only if estimatedGroundVelocity is a valid output.
*/

extern unsigned estimatedGroundVelocityIsAvailable;

/* estimatedGroundVelocity (unsigned 16 bits) is the measured velocity,
   in km/hr, of the aircraft while moving on the ground when
   estimatedGroundVelocityIsAvailable is a non-zero value.
*/

extern unsigned estimatedGroundVelocity;

void UpdateNGVelocity () {

      /* private constants */

      /* whcf is circumference in centimeters of the nose gear wheel. */

      static int whcf = (((WHEEL_DIAMETER * 254) / 7) * 22) / 100;

      /* maxMsecs is an upper bound on the time for a full rotation of
        the nose gear wheel.
      */

      static int maxMsecs = (whcf * 36) / MIN_SPEED;

      /* maxUpdates is an upper bound on the number of updates required
         for a full rotation of the nose gear wheel where maximum time
         between updates is 500 millisecs.
      */

      static int maxUpdates = (maxMsecs / 500) + 1;

      /* maxClicks is an upper bound the number of full rotations,
```

```
   i.e., clicks between innovations of this function where the
   maximum time between updates is 500 millisecs.
*/

static int maxClicks = (MAX_SPEED / (whcf * 36)) * 500;

/* unsigned variables on target hardware are 16 bits */

static unsigned prevTime = 0;
static unsigned prevCount = 0;
static unsigned prevCurr = 0;
static unsigned firstTime = 1;
static unsigned numFailedUpdates = 0;
static unsigned updatesWithoutNewClicks = 0;

unsigned thisTime, thisCount, currTime, t1, t2, t3, d1, d2,
  newEGV, deltaEGV, badResult;

currTime = Millisecs;
thisTime = NGClickTime;
thisCount = NGRotations;

/*
      “click”      “update”    “click”        “update”
         |            |           |               |
         |<-----------d1--------->|<------d2----->|
         |            |           |               |
         |<-----------t1--------->|<------t2----->|
         |            |           |               |
         |            |<-------------t3---------->|
   ------|------------|-----------|---------------|----> time
   W            X           Y              Z
         ^            ^           ^              ^
       prevTime   prevCurr     thisTime        currTime
       prevCount               thisCount

  The above diagram shows one possible sequence of events where
  there is a single “click” at point Y after the previous update
  prior to the current update at point Z.

  In general, there will be deltaCount updates between the
  previous update and the current update at point Z, where
  deltaCount is any number less than maxUpdates, including 0.
*/

if (thisTime == prevTime) return; /* too soon, just skip this update */

estimatedGroundVelocityIsAvailable = 0;

if (thisCount == prevCount) {
  /* Okay to skip a few updates at low speed, but eventually it needs
     to be set to unavailable if there are not any new “clicks”.  Note
     that an upper bound on changes in velocity should limit the
     potential inaccuracy that results from missing a few updates.
  */
  if (++updatesWithoutNewClicks > maxUpdates) {
    estimatedGroundVelocityIsAvailable = 0;
```

```
    return;
}
else updatesWithoutNewClicks = 0;

/* t1 is the elapsed time (in milliseconds) between the time at
   the most recent update prior to the previous update occurred
   and the time at which the most recent update occurred.
*/

if (prevTime <= thisTime) t1 = thisTime – prevTime;
else t1 = 65534 – (prevTime – thisTime);

/* t2 is the elapsed time (in milliseconds) between the time at
   the most recent update occurred and the current time.
*/

if (thisTime <= currTime) t2 = currTime – thisTime;
else t2 = 65534 – (thisTime – currTime);

if (t2 > maxMsecs) firstTime = 1;

if (firstTime == 1) {
  prevTime = thisTime;
  prevCount = thisCount;
  badResult = 0;
  firstTime = 0;
  return;
}

if (prevCount < thisCount) deltaCount = thisCount – prevCount;
else deltaCount = 65534 – (prevCount – thisCount);

badResult = (deltaCount > maxClicks) ? 1 : 0;

/* d1 is the distance (in centimeters) travelled since the most
   recent click seen by the most recent update until the most
   recent click, i.e., the distance travelled between prevTime
   and thisTime.
*/

d1 = whcf * deltaCount;

/* d2 is an estimate of the distance (in centimeters) travelled
   since the most recent click.  If there has at least once
   click since the time of the most recent click observed by
   the previous update, i.e., deltaCount > 0, then use d1/t1
   to approximate the current velocity used to calculate d2.
   Otherwise, use the velocity that results from the previous
   update, but make sure that d2 is less than the circumference
   of the wheel (which is important to check in the case of
   rapid deceleration to a low speed).
*/

if (deltaCount == 0) d2 = (estimatedGroundVelocity * t2) / 36;
else d2 = (d1 * t2) / t1;
if (d2 > whcf) d2 = whcf – 1;
```

```
        /*
           d1 + d2 = the total distance estimated to have been travelled
           since the most recent click that occurred prior to the time
           of the most recent update (in centimeters)

           t1 + t2 = the total elapsed time since the most recent click
           that occurred prior to the time of the most recent update (in
           milliseconds)
        */

        newEGV = ((((d1 + d2) * 360) / (t1 + t2)) + 5) / 10;

        badResult += newEGV > MAX_SPEED;

        if (newEGV < estimatedGroundVelocity)
          deltaEGV = estimatedGroundVelocity - newEGV;
        else deltaEGV = newEGV - estimatedGroundVelocity;

        if (prevCurr <= currTime) t3 = currTime – prevCurr;
        else t3 = 65534 – (thisTime – prevCurr);

        badResult += (((deltaEGV * 100) / t3) * 100) > MAX_ACCEL;

        estimatedGroundVelocityIsAvailable = 1;

        /* If the result is bad, then don’t update estimatedGroundVelocity,
           as the previous estimate should be valid for a few more update
           cycles. But if the problem persists, then reset the flag to
           indicate that the estimated velocity is no longer available.
        */

        if ((badResult > 0) && (numFailedUpdates++ > MAX_FAILED_UPDATES)) {
           firstTime = 1;
           estimatedGroundVelocityIsAvailable = 0;
        }
        else if (badResult > 0) numFailedUpdates++;
        else estimatedGroundVelocity = newEGV;

        prevTime = thisTime;
        prevCount = thisCount;
        prevCurr = currTime;
}
```