Drone Ballistics

Brian Beckman, Amazon Prime Air, 02 Nov 2016

How far away from a flying drone should you stand so that you don't get hit if the drone goes ballistic? The answer depends on initial vector position and vector velocity, vector wind, and parameters like mass m, area A presented to the wind, and coefficient of drag C_d. The solution here consists in several parts:

1. Yellow Trajectory¨ --- plotted as downrange versus altitude: deterministic path accounting for drag from three sources:
   1.1. resistance opposing the tangential motion of the vehicle (along its trajectory curve)
   1.2. constant prevailing downrange and crossrange wind in knots, input to the simulation¨ from the UI sliders

If the vehicle begins with no forward velocity, it will be blown off the vertical fall plumb-line by prevailing and gust wind. Prevailing and gust wind have no vertical components.

2. Blue Trajectory --- plotted as downrange versus altitude: deterministic path with no drag or relative effect from the wind at all. This is the "high-school-physics" solution provided as a sanity check against the yellow trajectory curve.
3. Monte-Carlo CEP (Circular Error Probable) --- scatter-plotted separately on a grid, downrange versus crossrange: a collection of strike points on the ground derived by varying launch angles and gust wind velocities according to given input¨ standard deviations.

The initial conditions are two-dimensional: altitude (positive up in the trajectory diagram) and downrange ground speed in knots.

To interact with the program, you will need Wolfram's free CDF player from here: https://www.wolfram.-com/cdf-player/

Looks like with a 5-knot wind, a 55-lb drone going at a climb angle of 28 degrees, a speed of 50 mph at a height of 90 feet will land somewhere between 125 and 200 feet from the point where it goes ballistic.

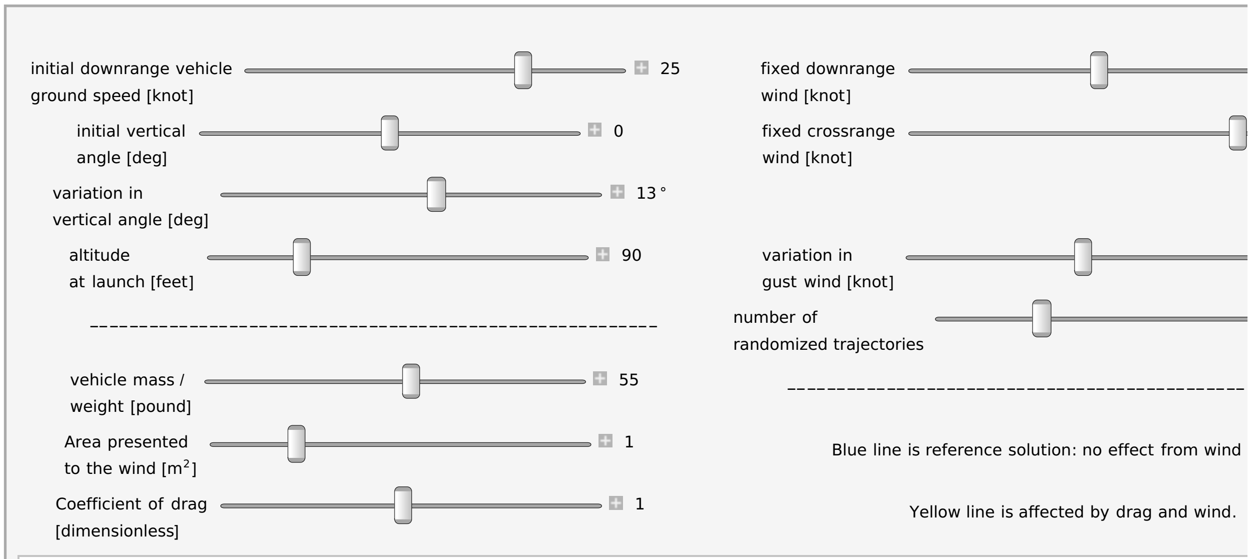
The coordinate system is Earth-fixed: x = downrange (positive to the right), y = altitude (positive upward), z = crossrange (positive direction is "out of the page."). Ground zero is the launch point where the drone goes ballistic with whatever residual velocity it had.

References

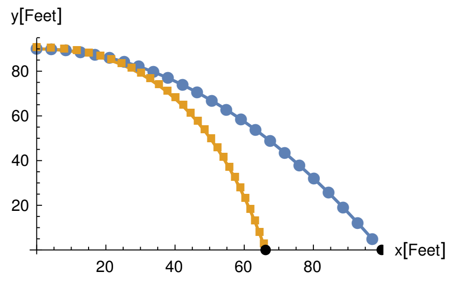
http://tinyurl.com/o9q654r
http://tinyurl.com/h9sbzd5
http://tinyurl.com/je8wq8q
http://tinyurl.com/zxubzcc

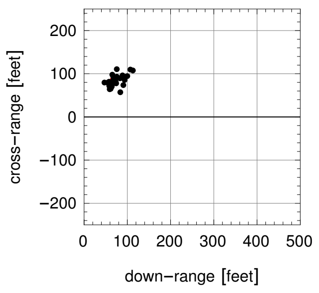
```
Manipulate[
 (* Units of measure (could be handled in a variety of ways;
   this is the easiest to understand but maybe not the most elegant.) *)
 With[{mpsPerMph = 0.44704,
   mpsPerKnot = 0.514444,
   feetPerMeter = 3.28084,
   kgPerPound = 0.453592,
   gMpS2 = 9.81,
   ρKgPerM3 = 1.25},
  With[{
    (* v0mps is the 2-vector initial vehicle ground velocity in meters per second,
    at elevation angle αVerticalDeg upward from the horizontal in degrees. *)
    v0mps = mpsPerKnot * vVxKnot {1.0, Sin[αVerticalDeg °] / Cos[αVerticalDeg °],
       0(*crossrange*)},

    (* p0m is the initial position vector,
    starting off as pure altitude (no downrange component),
    in meters. Ground zero is the point at which the vehicle goes ballistic. *)
    p0m = {0, hFeet / feetPerMeter, 0},

    (* Mass of the vehicle in kilograms,
    converted from pounds mass (approximately 1/32 of a slug) *)
    mKg = mPound kgPerPound,

    (* Density of dry air multiplied by area-
     in-square-meters presented to the wind. *)
    ρPerM = AMeter2 ρKgPerM3,

    (* Run for maxTSeconds of simulation time with a time-
     step of dtSeconds. Smaller time steps run slower but
      produce more faithful answers. To tune the simulation,
    reduce dtSeconds until the solution does not change character. *)
    maxTSeconds = 20, dtSeconds = 0.1,

    (* Get n random launch angles,
    with mean αVerticalDeg and standard deviation σVerticalAngleDeg. *)
    anglesRad = RandomVariate[NormalDistribution[
       αVerticalDeg °, σVerticalAngleDeg °], nSamples],

    (* Get random gust wind: mean speed zero,
    given standard deviation, and uniformly random azimuth
  angle. TODO: Weibull or Rayleigh distribution is more standard. *)
 randomGustAzimuthRad = Function[RandomReal[{0.0, 2.0 π}]],
 randomGustSpeedMps =
  Function[RandomVariate[NormalDistribution[0, mpsPerKnot * σvKnotGust]]]},

With[{randomGustVectorMps = Function[
    randomGustSpeedMps[] * With[{a = randomGustAzimuthRad[]},
      {(*downrange*)Cos[a], (*vertical*)0.0, (*crossrange*)Sin[a]}]]},

 With[
  {(* The initial vector wind velocity: (positive downrange (x) means blowing left-
        to-right; positive crossrange (z) means blowing out of the page).  *)
   vWvectorMps = mpsPerKnot {(*downrange*)vWxKnot, (*vertical*)
      0, (*crossrange*)vWzKnot},
   vWspeedMps = mpsPerKnot * Sqrt[vWxKnot * vWxKnot + vWzKnot * vWzKnot]},

  (* Get random variations in¨ gust wind and launch angle. *)
  With[{vZerosMps =
     (v0mps + randomGustVectorMps[] +
         {(*downrange*)Cos[#], (*vertical*)0.0, (*crossrange*)Sin[#]}) & /@
      anglesRad},

   (* Given initial position p0 and velocity v0, solve for a trajectory,
   triples {qx, qy, qz} and their time derivatives {qx', qy' qz'}. *)
   (* TODO: add random winds. This is 'closed over'
     the constant prevailing wind. *)
   With[{solver = {p0, v0} ⟼ NDSolve[(* Arrow notation
         means 'function of p0 and v0.' *)
        (* Vehicle downrage groundspeed qx' is measured against the downrange
         coordinate axis x¨. rx is downrange airspeed, needed for drag.  If qx'
         is positive (vehicle moving to the right wrt ground) and vWvectorMps
         has the same magnitude and is positive (blowing to the right),
        then the downrange airspeed, namely rx is zero. That means that
          rx must be qx' - vWvectorMps⟦x⟧. Likewise for qy' and qz'. *)
        With[{rx = qx '[t] - vWvectorMps⟦1⟧, ry = qy '[t] - vWvectorMps⟦2⟧,
          rz = qz '[t] - vWvectorMps⟦3⟧},
         With[{v = Sqrt[rx rx + ry ry + rz rz]}, (* This speed is
           always positive, by construction. *)
```

```
            {mKg qx''[t] == -(1/2 ρPerM Cd v rx) (* Force of drag opposes airspeed,
              hence the extra minus signs. If rx is positive,
              then the vehicle is moving rightwards with respect to the wind
               and the drag force must be leftwards, that is, negative. *)
              mKg qy''[t] == -(1/2 ρPerM Cd v ry) - gMpS2 mKg, (*vertical*)
              mKg qz''[t] == -(1/2 ρPerM Cd v rz) (*crosswind*),
              (* The following are the initial conditions. *)
              qx[0] == p0⟦1⟧, qy[0] == p0⟦2⟧, qz[0] == p0⟦3⟧,
              qx'[0] == v0⟦1⟧, qy'[0] == v0⟦2⟧, qz'[0] == v0⟦3⟧}]],
          {qx[t], qy[t], qz[t]}, (* Solve for these quantities... *)
          {t, 0, maxTSeconds}], (* for these times. *)
      (* Give me a solution 'soln', which has a complicated structure,
      and I will pick out the x, y, z, and t. *)
      solPicker = soln ↦ <|(* Arrow notation means 'function of soln.' The result
           is a hashmap bracketed with <|...|> ('Association' in Wolfram-speak). *)
          "x" → soln⟦1, 1, 2⟧,
          "y" → soln⟦1, 2, 2⟧,
          "z" → soln⟦1, 3, 2⟧,
          "t" → Floor[soln⟦1, 1, 2, 0, 1, 1, 2⟧, 0.001]|>},
     (* Solve for the main, yellow trajectory, qs. *)
     With[{qs = solPicker[solver[p0m, v0mps]]},
      (* Solve for nearby, randomized trajectories, qss. *)
      With[{qss = solPicker[solver[p0m, #]] & /@ vZerosMps},
       (* Pick out the main series of positions and times. *)
       With[{qx = qs["x"], qy = qs["y"], qz = qs["z"], t1m = qs["t"]},
        (* The following, qpd,
        is a high-school solver for a ¨trajectory with no wind¨. It's included
          just for visual reference for the parabolas and does not produce the
          CEP. The CEP comes only from the more accurate numerical solution. *)
        With[{qpd = {p0, v0, vW, tm} ↦ p0 + (v0) tm + 1/2 {0, -gMpS2, 0} tm^2,
          (* Here is a function to pick out time
   series from any trajectory, reference or numerical¨. *)
  qpf = {picked, tm} ↦ ({picked["x"], picked["y"], picked["z"]} /.
      {t → Min[tm, picked["t"]]})},
(* Get a reference curve D from the same initial
  conditions as the numerical solution. *)
With[{trajAndTimesD = Module[{i = 0}, Table[{++i, t2,
       qpd[p0m, v0mps, vWvectorMps, t2]},
      {t2, 0, maxTSeconds, dtSeconds}]],
   (* Get a realistic curve E from the main numerical solution qs. *)
   trajAndTimesE = Module[{i = 0}, Table[{++i, t2, qpf[qs, t2]},
      {t2, 0, maxTSeconds, dtSeconds}]],
   (* Get trajectoies Fs for the randomized numerical solutions qss. *)
   trajAndTimesFs = Module[{i = 0}, Table[{++i, t2, qpf[#, t2]},
        {t2, 0, maxTSeconds, dtSeconds}]] & /@ qss},
  (* The rest of this is plotting code. D is the reference
   trajectory (analytical, unrealistic). E is the main numerical
   solution. Fs are the randomized solutions near E. *)
  With[{
    tibbleD = Select[trajAndTimesD, #[[3, 2]] ≥ 0 & ],
    tibbleE = Select[trajAndTimesE, #[[3, 2]] ≥ 0 & ],
    tibbleFs = Select[#, #[[3, 2]] ≥ 0 & ] & /@ trajAndTimesFs,
    (* A function to find landing times so we
     can stop drawing the line when we hit the ground. *)
    landingTime = {trajTime} ↦
      With[{lastUp = Last@Select[trajTime, #[[3, 2]] ≥ 0 &]},
       With[{indexLastUp = lastUp[[1]],
         lastUpHeight = lastUp[[3, 2]], lastUpTime = lastUp[[2]]},
        With[{firstDownTime = trajTime[[indexLastUp + 1, 2]],
          firstDownHeight = trajTime[[indexLastUp + 1, 3, 2]]},
                                       firstDownTime - lastUpTime
         lastUpTime + lastUpHeight  ─────────────────────────────────]]]},
                                     lastUpHeight - firstDownHeight
   (* Make display units from the internal SI / SCDU units. *)
   With[{
     trajD = feetPerMeter (#[[3]] & /@ tibbleD),
     trajQ = feetPerMeter (#[[3]] & /@ tibbleE),
     trajFs =  feetPerMeter ((tibbleF ↦ #[[3]] & /@ tibbleF) /@ tibbleFs),
     tD = landingTime[trajAndTimesD],
     tE = landingTime[trajAndTimesE],
                  tFs = landingTime /@ trajAndTimesFs},
                 With[{
                   landingPointD = feetPerMeter * qpd[p0m, v0mps, vWvectorMps, tD],
                   landingPointE = feetPerMeter * qpf[qs, tE],
                   landingPointFs =
                    feetPerMeter * ((qpf @@ # &) /@ Transpose[{qss, tFs}])},
                  With[{
                    cpe = GraphicsRow[
                      {ListLinePlot[
                        {Drop[#, -1] & /@ trajD, Drop[#, -1] & /@ trajQ},
                        AxesLabel → {"x[Feet]", "y[Feet]"},
                        PlotMarkers → {{●, 8}, {■, 8}},
                        ImageSize → Medium,
                        Epilog →
                         {PointSize@0.03, Point[Drop[landingPointD, -1]],
                          Point[Drop[landingPointE, -1]]}],
                       With[{ex = 500},
                        Graphics[{Red, Point[{landingPointE⟦1⟧, landingPointE⟦3⟧}],
                          Black, Point /@ ({#⟦1⟧, #⟦3⟧} & /@ landingPointFs)},
                         Axes → True,
                         FrameLabel →
                          {{"cross-range [feet]", ""}, {"down-range [feet]", ""}},
                         Frame → True,
                         GridLines → Automatic,
                         PlotRange → {{0, ex}, {-ex / 2, ex / 2}}]]}]},
                   Show[{cpe}]
                  ]]]]]]]]]]]]]],
     Grid[
      {{Control[{{vVxKnot, 25, "initial downrange vehicle\nground speed [knot]"}, -50, 50, 1,
          Appearance → "Labeled"}],
        Control[{{αVerticalDeg, 0, "initial vertical\nangle [deg]"},
          -89, 89, 1, Appearance → "Labeled"}],
        Control[{{σVerticalAngleDeg, 15 °, "variation in\nvertical angle [deg]"},
          1 °, 22 °, 1 °, Appearance → "Labeled"}],
        Control[{{hFeet, 90, "altitude\nat launch [feet]"}, 1,
          400, 1, Appearance → "Labeled"}],
        "------------------------------------------------------",
        Control[{{mPound, 55, "vehicle mass /\nweight [pound]"},
          1, 100, 1, Appearance → "Labeled"}],
        Control[{{AMeter2, 1, "Area presented\nto the wind [m^2]"},
          0, 5, 0.01, Appearance → "Labeled"}],
  Control[{{Cd, 1, "Coefficient of drag\n[dimensionless]"},
    0, 2.1, .05, Appearance → "Labeled"}]},
 {Control[{{vWxKnot, 0, "fixed downrange\nwind [knot]"},
    -50, 50, 1, Appearance → "Labeled"}],
  Control[{{vWzKnot, 0, "fixed crossrange\nwind [knot]"},
    -50, 50, 1, Appearance → "Labeled"}],
  "",
  Control[{{σvKnotGust, 5, "variation in\ngust wind [knot]"},
    1, 25, 1, Appearance → "Labeled"}],
  Control[{{nSamples, 5, "number of\nrandomized trajectories"},
    1, 100, 1, Appearance → "Labeled"}],
  "--------------------------------------------------------",
  "Blue line is reference solution: no effect from wind or drag.",
  "Yellow line is affected by drag and wind."
 }} // Transpose]]
```



```
initial downrange vehicle ground speed [knot]   25        fixed downrange wind [knot]
initial vertical angle [deg]                    0         fixed crossrange wind [knot]
variation in vertical angle [deg]               13°
altitude at launch [feet]                       90        variation in gust wind [knot]
                                                          number of randomized trajectories
---------------------------------------------------       ---------------------------------------------
vehicle mass / weight [pound]                   55
Area presented to the wind [m^2]                1         Blue line is reference solution: no effect from wind
Coefficient of drag [dimensionless]             1         Yellow line is affected by drag and wind.
```


y[Feet]
x[Feet]


cross−range [feet]
down−range [feet]