Brigham Young University
AUVSI Capstone Team (Team 45)

Parachute Deployment Drop Simulation

ID	Rev.	Date	Description	Author	Checked By
GV-008	1.0	2-20-2019	Created	Jacob Willis	Derek Knowles

Introduction

To better understand the dynamics of the payload drop with a parachute, a simulation was written based on the dynamics described in 'Dropping an Object on a Target' supplement to *Small Unmanned Aerial Vehicles* by Randal W. Beard. The supplement is found at http://uavbook.byu.edu/lib/exe/fetch.php?media=shared:object_drop.pdf. To more closely simulate the changing dynamics of a parachute opening, the surface area and coefficient of drag were made functions of time, which change between a small value (before parachute opening) and a larger value after the parachute opens. These values were determined from typical values and from t Cd reported by the parachute manufacturer. Surface area is estimated based on the UGV and folded/furled parachute surface areas. Other parameters, such as mass, were determined from estimated UGV system values.

Results

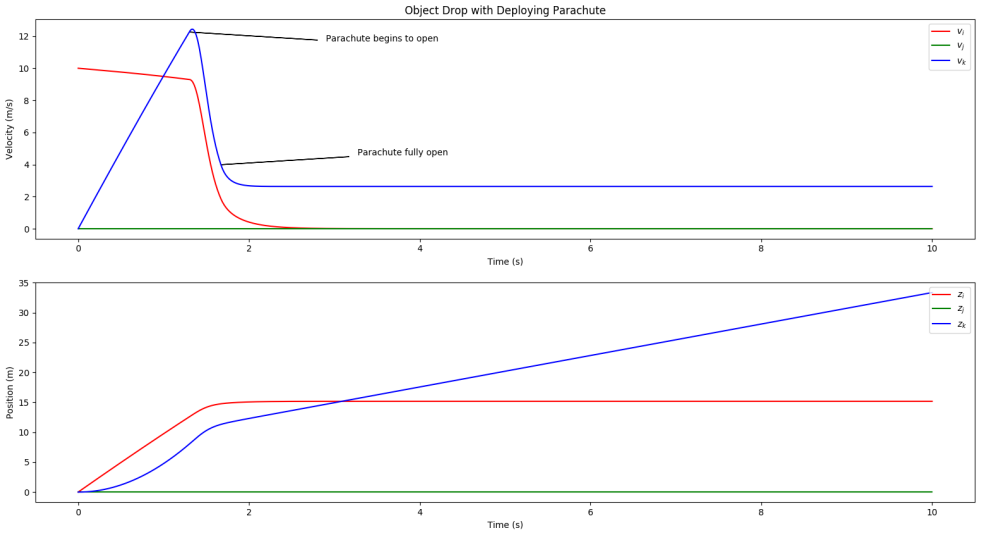

Figure 1: Parachute deployment simulation plots

The main product of this simulation is the plot in Fig 1. Several important results can be seen in this figure. First, the dropped object essentially maintains it original forward velocity until the parachute opens. The parachute's drag occurs in the direction of the object's velocity, so the parachute very quickly reduces horizontal velocities to 0. Another

important result is that the parachute reduces the terminal velocity of the dropped object very quickly, slowing it to a speed low enough for a soft landing.

This informs us of the importance of repeatedly timing the different portions of payload drop sequence in order to accurately predict where our UGV will land.

Simulation Code

```
#! /usr/bin/env python3
"""
    drop.py
    Simulation of parachute unraveling and opening during a UGV drop.
    Author: Jacob Willis
    Date: 29-Jan-2019
"""

import numpy as np
import matplotlib.pyplot as plt


# when the parachute is opening
# times estimated from drop testing with actual parachute
paraTrans = (1.3,1.67) # seconds

# calculate coefficient of drag for opening the parachute after a certain
# time. Piecewise step with a slope between the min and max
def Cd(t):
    minCd = .5 # starting coefficient of drag
    maxCd = 1.5 # final coefficient of drag

    if (t < paraTrans[0]):  # parachute not deployed
        return minCd
    elif (t < paraTrans[1]): # parachute deploying
        return minCd + (t-paraTrans[0])*(maxCd-minCd)/(paraTrans[1] - paraTrans[0])
    else:
        return maxCd # parachute fully deployed

# calculate the parachute area for opening the parachute after a certain
# time. Piecewise step with a slope between the min and max
def Area(t):
    minArea = np.pi*.05**2 # area of closed parachute
    maxArea = np.pi*.5**2 # area of open parachute

    if (t < paraTrans[0]):  # parachute not deployed
        return minArea
    elif (t < paraTrans[1]): # parachute deploying
```

```
        return minArea + (t-paraTrans[0])*(maxArea-minArea)/(paraTrans[1] - paraTrans[0])
    else:
        return maxArea # parachute fully deployed

g = 9.8 # acceleration due to gravity
mass = 450 # grams
rho = 1.2 # density of air

k = np.array([[0], [0], [1]])

t_start = 0
t_end = 10
Ts = .01
tvec = np.linspace(t_start, t_end, (t_end-t_start)/Ts)
v0 = np.array([10, 0, 0]) # initial velocity conditions (i, j, k)
v = np.zeros([3, len(tvec)])
v[:,0] = v0
z = np.zeros([3, len(tvec)])

# run the simulation
step = 0
while step < len(tvec)-1:
  t = tvec[step]
  vs = v[:, [step]]
  v[:, [step+1]] = vs + Ts*(g*k - (rho*Area(t)*Cd(t)*(np.linalg.norm(vs)*vs)))
  z[:, [step+1]] = z[:, [step]] + Ts*vs
  step += 1


# plot the velocity results
plt.subplot(2, 1, 1)
plt.title("Object Drop with Deploying Parachute")
plt.plot(tvec, v[0,:], 'r', label='$v_i$')
plt.plot(tvec, v[1, :], 'g', label='$v_j$')
plt.plot(tvec,  v[2, :], 'b', label='$v_k$')
plt.legend(loc=1)
plt.xlabel("Time (s)")
plt.ylabel("Velocity (m/s)")

# draw arrow for parachute deployment
arrow_x = paraTrans[0]
arrow_y = v[2, int(paraTrans[0]*len(tvec)/(t_end - t_start))]
arrow_dx = 1.5
arrow_dy = -.5
plt.arrow(arrow_x, arrow_y, arrow_dx, arrow_dy)
text_x = arrow_x + arrow_dx + .1
text_y = arrow_y + arrow_dy - .1
plt.text(text_x, text_y, "Parachute begins to open")
```

```
# draw arrow for parachute fully open
arrow_x = paraTrans[1]
arrow_y = v[2, int(paraTrans[1]*len(tvec)/(t_end - t_start))]
arrow_dx = 1.5
arrow_dy = .5
plt.arrow(arrow_x, arrow_y, arrow_dx, arrow_dy)
text_x = arrow_x + arrow_dx + .1
text_y = arrow_y + arrow_dy + .1
plt.text(text_x, text_y, "Parachute fully open")

# plot position results
plt.subplot(2, 1, 2)
plt.plot(tvec, z[0,:], 'r', label='$z_i$')
plt.plot(tvec, z[1, :], 'g',label='$z_j$')
plt.plot(tvec, z[2, :], 'b', label='$z_k$')
plt.xlabel("Time (s)")
plt.ylabel("Position (m)")
plt.legend(loc=1)

plt.show(block=False)
input("Press any key to continue...")
```

Conclusion

The parachute deployment drop simulation successfully allowed us to simulate the drop of our unmanned ground vehicle. Simulation results led us to pick a parachute diameter of 36 inches in order to meet the system requirements for dropping velocity.