Brigham Young University

AUVSI Capstone Team (Team 45)

Geolocation Algorithm Description

ID	Rev.	Date	Description	Author	Checked By
IM-004	1.0	12-12-2018	Initial release	Connor Olsen	Tyler Miller
IM-004	1.0	02-20-2019	Comment / code updates	Connor Olsen	Tyler Miller

1 Introduction

Geolocation of targets is one of our key success measures, whose accuracy is scored by the judges for points. Accuracy is determined by distance of our location estimate from ground truth, at a max of 150ft. Anything further than 150ft will score 0 points. This geolocation algorithm is fast enough to work in real time on the server as targets are cropped.

2 Introduction

Given how short the algorithm is (150 lines), the best way to document it is through the code itself.

```python
'''
          BYU AUVSI-SUAS Capstone
        Target Geolocation Algorithm
             Connor Olsen, 2019
'''

import numpy as np
import math as math

"""
Calculates the meters between two GPS coordinates
@type lat1: float
@param lat1: The latitude of the first GPS coordinate

@type lon1: float
@param lon1: The longitude of the first GPS coordinate

@type lat2: float
@param lat2: The latitude of the second GPS coordinate

@type lon2: float
@param lon2: The longitude of the first GPS coordinate

@rtype: array of floats(4)
@return: Distance north (meters), distande east (meters), total distance
   (meters), angle (0 deg = East)
"""
def GPStoMeters(lat1, lon1, lat2, lon2):
```

```
    d2r = 0.0174532925199433
    dlong = (lon2 - lon1) * d2r
    dlat = (lat2 - lat1) * d2r
    a = (math.sin(dlat/2.0))**2 + math.cos(lat1*d2r) * math.cos(lat2*d2r) *
        (math.sin(dlong/2.0))**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = 6367 * c; #Distance between points in meters
    d_meters = d * 1000
    dy = lat2 - lat1
    dx = math.cos(d2r * lat1) * (lon2 - lon1)
    angle = math.atan2(dy, dx)
    east_dis_meters = d_meters * math.cos(angle)
    north_dis_meters = d_meters * math.sin(angle)
    returnvalues = np.array([north_dis_meters, east_dis_meters, d_meters,
        angle])
    return returnvalues

"""
Calculates GPS coordniates given a starting coordinate and meters north and
    east
@type Lat: float
@param Lat: The latitude of the GPS coordinate

@type Lon: float
@param Lon: The longitude of the GPS coordinate

@type north_displacement: float
@param north_displacement: The distance north of the given coordinates (meters)

@type east_displacement: float
@param east_displacement: The distance east of the given coordinates (meters)

@rtype: array of floats(2)
@return: latitude of new target, longitude of new target
"""
def MeterstoGPS(Lat, Lon, north_displacement, east_displacement):
    # Earth[U+FFFD]s radius, sphere
    R = 6378137
    # Coordinate offsets in radians
    dLat = north_displacement/R
    dLon = east_displacement/(R*math.cos(math.pi*Lat/180))
    # OffsetPosition, decimal degrees
    lat0 = Lat + dLat * 180/math.pi
```

```
    lon0 = Lon + dLon * 180/math.pi
    returnvals = [lat0, lon0];
    return returnvals


'''
The data below will be pulled from the database:
Attitude
MAV Coordinates
Pixel Coordinates of Target
'''
# For now, we will use dummy data
phi_in = 0
theta_in = 60
psi_in = 90
alpha_az = 0 # Assuming the camera is angled with the top facing out the nose
alpha_el = -math.pi/2
lat_mav = 40.2465
lon_mav = -111.6483
lat_gnd = 40.2485
lon_gnd = -111.6458
height = 16

MaxX = 2000 # Max x pixels
MaxY = 2000 # Max y pixels
'''
The top left and bottom right coordinates of the cropped
photo are provided. This section finds the center of the
cropped image and translates it into
'''
TopLeftX = 0
TopLeftY = 0
BottomRightX = 0
BottomRightY = 0
CenterX = BottomRightX - TopLeftX
CenterY = BottomRightY - TopLeftY
AdjustedCenterX = CenterX-(MaxX/2)
AdjustedCenterY = (-1)*(CenterY-(MaxY/2))

M = 4000
Ex = -15 #AdjustedCenterX
Ey = -1028 #-AdjustedCenterY
```

```
fov_ang = 0.872665 #field of View angle --> A6000 83* - 32* (in radians)
f = M/(2*math.tan(fov_ang/2))

l_cusp_c = 1/math.sqrt(Ex**2 + Ey**2 + f**2) * np.array([[Ex],[Ey],[f]])
'''
Convert Roll, Pitch and Yaw to radians
'''
phi = phi_in*math.pi/180
theta = theta_in*math.pi/180
psi = psi_in*math.pi/180

'''
k unit vector in the inertial frame
'''
k_i = np.array([[0],[0],[1]])

'''
Calculates distance between ground station and mav in meters to determine MAV's
relative location. Then creates the position vector [Pn Pe Pd]^T
'''
positionData = GPStoMeters(lat_gnd, lon_gnd, lat_mav, lon_mav)
P_i_mav = np.array([[positionData[0]], [positionData[1]], [-height]])

'''
Trigonometry calculated one time to decrease run time
'''
cphi = math.cos(phi)
sphi = math.sin(phi)
ctheta = math.cos(theta)
stheta = math.sin(theta)
cpsi = math.cos(psi)
spsi = math.sin(psi)
caz = math.cos(alpha_az)
saz= math.sin(alpha_az)
cel = math.cos(alpha_el) #Rreturns 6.123234e-17 instead of 0
sel = math.sin(alpha_el)

'''
Rotation from body to inertial frame
Found on page 15 of Small Unmanned Aircraft
'''
R_v2b = np.array([[ctheta*cpsi, ctheta*spsi, -stheta],\
```

```
[sphi*stheta*cpsi-cphi*spsi, sphi*stheta*spsi+cphi*cpsi, sphi*ctheta],\
[cphi*stheta*cpsi+sphi*spsi, cphi*stheta*spsi-sphi*cpsi, cphi*ctheta]])
R_b2v = np.transpose(R_v2b)
R_b2i = R_b2v

'''
R_g_to_b
Found on page 227 of Small Unmanned Aircraft
'''
R_b2g1 = np.array([[caz, saz, 0],[-saz, caz, 0],[0, 0, 1]])
R_g12g = np.array([[cel, 0, -sel],[0, 1, 0],[sel, 0, cel]])
R_b2g = np.matmul(R_g12g, R_b2g1)
R_g2b = np.transpose(R_b2g)

'''
R_c_to_g
% Found on page 227 of Small Unmanned Aircraft
'''
R_g2c = np.array([[0, 1, 0],[0, 0, 1],[1, 0, 0]])
R_c2g = np.transpose(R_g2c)

'''
% For simplicity, the three Rotation matrices are combined into one below
RbiRbgRcg = R_b_to_i * R_g_to_b * R_c_to_g;
'''
RbiRbgRcg = np.matmul(np.matmul(R_b2i, R_g2b), R_c2g)


l_cusp_i = np.matmul(RbiRbgRcg, l_cusp_c)
P_i_tar = P_i_mav + height*l_cusp_i/l_cusp_i[2]

print("Groundstation coordinates")
print(str(lat_gnd) + " " + str(lon_gnd))

print("MAV coordinates")
print(str(lat_mav) + " " + str(lon_mav))

TargetCoordinates = MeterstoGPS(lat_gnd, lon_gnd, P_i_tar[0], P_i_tar[1])
print("Target coordinates")
print(str(float(TargetCoordinates[0])) + " " +
    str(float(TargetCoordinates[1])))
```