################################################################################################

# Since we'll need a measurement update for both the GNSS and the LIDAR data, let's make

# a function for it.

################################################################################################

def measurement\_update(sensor\_var, p\_cov\_check, y\_k, p\_check, v\_check, q\_check):

# 3.1 Compute Kalman Gain

K = p\_cov\_check@h\_jac.T@np.linalg.inv(h\_jac@p\_cov\_check@h\_jac.T + sensor\_var)

# 3.2 Compute error state

diff = (y\_k-p\_check)

dx = np.dot(K,diff.T)

# 3.3 Correct predicted state

dp = dx[0:3] #error state of position

dv = dx[3:6] #error state of velocity

dth = dx[6:9] #error state of theta

p\_check = p\_check + dp.T

v\_check = v\_check + dv.T

q\_check = Quaternion(axis\_angle=dth).quat\_mult(q\_check,out = 'np')

#print('qqqqqq', q\_check)

#print('qqqqqqsssssssss', q\_check.shape)

# 3.4 Compute corrected covariance

p\_cov\_check = (np.eye(9)- (K@h\_jac))@p\_cov\_check

return p\_check, v\_check, q\_check, p\_cov\_check

#### 5. Main Filter Loop #######################################################################

################################################################################################

# Now that everything is set up, we can start taking in the sensor data and creating estimates

# for our state in a loop.

################################################################################################

for k in range(1, imu\_f.data.shape[0]): # start at 1 b/c we have initial prediction from gt

delta\_t = imu\_f.t[k] - imu\_f.t[k - 1]

# 1. Update state with IMU inputs

Cns = Quaternion(\*q\_est[k-1]).to\_mat() # orientation quaternion at k-1

p\_check = p\_est[k-1] + delta\_t \* v\_est[k-1] + ((delta\_t\*\*2)/2) \* (Cns @ imu\_f.data[k-1].reshape((3,1)) - g.reshape((3,1))).T

v\_check = v\_est[k-1] + delta\_t \* (Cns @ imu\_f.data[k-1].reshape((3,1)) - g.reshape((3,1))).T

q\_check = Quaternion(axis\_angle = imu\_w.data[k-1]).quat\_mult(q\_est[k-1]) # orientation at k

# 1.1 Linearize Motion Model

F\_k\_1 = np.eye(9) #1

F\_k\_1[0:3,3:6] = np.eye(3) \* delta\_t #2

F\_k\_1[3:6,6:9] = -Cns.dot(skew\_symmetric(imu\_f.data[k - 1])) \* delta\_t

L\_k\_1 = l\_jac

Q\_k = np.eye(6)

Q\_k[0:3, 0:3] = Q\_k[0:3, 0:3]\*var\_imu\_f

Q\_k[3:6, 3:6] = Q\_k[3:6, 3:6]\*var\_imu\_w

#Q\_k = (delta\_t\*\*2)\*Q\_k

# 2. Propagate uncertainty

p\_cov\_check = F\_k\_1@p\_cov[k-1]@F\_k\_1.T + L\_k\_1@Q\_k@L\_k\_1.T

# 3. Check availability of GNSS and LIDAR measurements

GNSSavailable = is\_avl(gnss.t,imu\_f.t[k]) #To check if GNSS reading is available

#Lidaravailable = is\_avl(imu\_f.t[k],lidar.t) # To check if LIDAR reading is available

sensorVar\_gnss= np.diag([var\_gnss,var\_gnss,var\_gnss])

sensorVar\_lidar= np.diag([var\_lidar,var\_lidar,var\_lidar])

if GNSSavailable != -1: #valid gnss data

y = gnss.data[GNSSavailable].reshape(3,1)

p\_check, v\_check, q\_check, p\_cov\_check=measurement\_update(sensorVar\_gnss, p\_cov\_check, gnss.data[GNSSavailable], p\_check, v\_check, q\_check)

p\_est[k] = p\_check

v\_est[k] = v\_check

q\_est[k] = q\_check

p\_cov[k] = p\_cov\_check

print(p\_check)