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] = -C\_ns.dot(skew\_symmetric(imu\_f.data[k - 1])) \* delta\_t

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

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

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

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

# 3.1 Compute Kalman Gain

sensor\_var = sensor\_var\*np.eye(3)

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

# 3.2 Compute error state

delta\_x = np.dot(Kk,(y\_k - p\_check))

# 3.3 Correct predicted state

delta\_p = delta\_x[0:3]

delta\_v = delta\_x[3:6]

delta\_phi = delta\_x[6:9]

p\_check = p\_check + delta\_p

v\_check = v\_check + delta\_v

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

# 3.4 Compute corrected covariance

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

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

In the motion model:

dTheta=delta\_t\*imu\_w.data[k-1]

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

In the measurement update process:

# 3.1 Compute Kalman Gain

Kk=p\_cov\_check@h\_jac.T@np.linalg.inv(h\_jac@p\_cov\_check@h\_jac.T+sensor\_var) #KK 9\*3

# 3.2 Compute error state 9\*3 \* 3\*1 =9\*1

dXk=np.dot(Kk,(y\_k-p\_check))

# 3.3 Correct predicted state

p\_check=p\_check+ dXk[0:3]

v\_check=v\_check+ dXk[3:6]

dPhi=dXk[6:9]

q\_check = Quaternion(axis\_angle= imu\_w.data[k-1]\*delta\_t).quat\_mult(q\_est[k-1])) #Q

# 3.4 Compute corrected covariance

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