Quaternion Approach Method

Euler from Quaternion

```
def euler_from_quaternion(x, y, z, w):
    if w == 1:
        return 0,0,0
    roll_x = np.arctan2(y*z + x*w,0.5 - z*z - w*w)
    pitch_y = np.arcsin(2.0 * (y*w - x*z))
    yaw_z = np.arctan2(2*(w*z+x*y),1-2*(y*y+z*z))
        return roll_x, pitch_y, yaw_z
```

Computing New Pose

```
for row in data_reader:
        if counter == 0:
            q = [1,0,0,0] #[w,x,y,z]
        else:
            quat_gyro = [0, float(row[0]), float(row[1]), float(row[2])]
            q_grad = 0.5 * quaternionProduct (pose_initial, quat_gyro)
            new_pose = quaternionNormalize (pose_initial, q_grad, dt=dt)
            pose_initial, q = new_pose, new_pose
        counter += 1
```

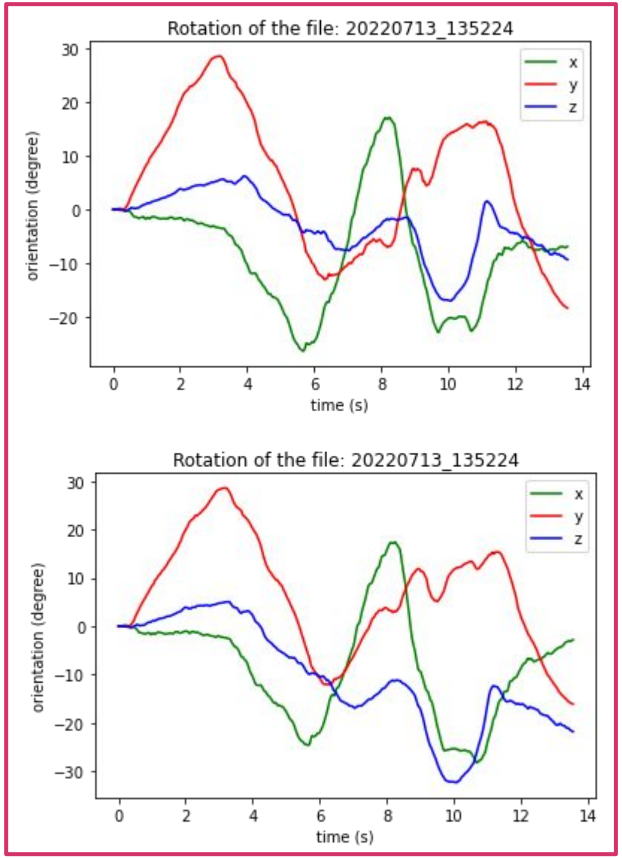
Quaternion Product

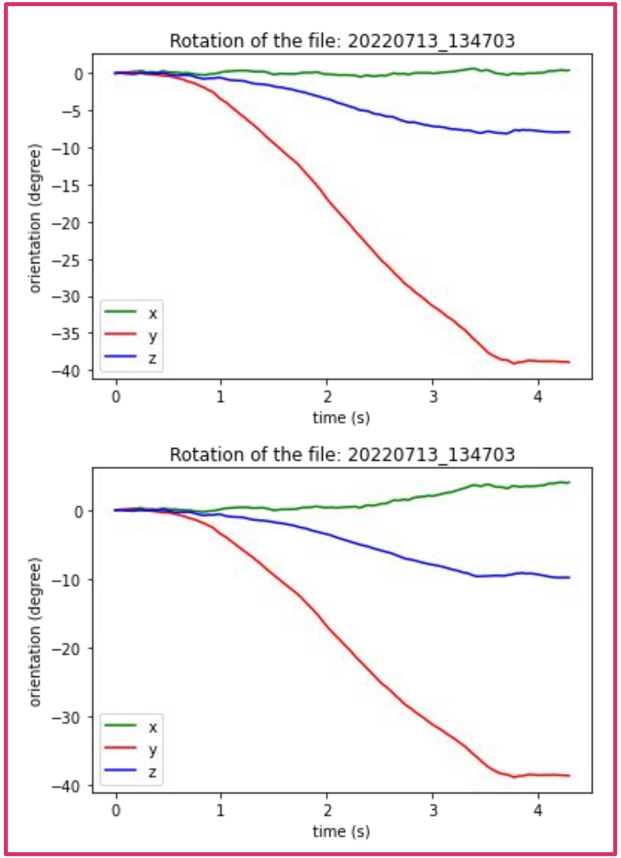
```
def quaternionProduct (q1, q2):
    w0, x0, y0, z0 = q1
    w1, x1, y1, z1 = q2
    return np.array([-x1 * x0 - y1 * y0 - z1 * z0 + w1 * w0,
                     x1 * w0 + y1 * z0 - z1 * y0 + w1 * x0,
                     -x1 * z0 + y1 * w0 + z1 * x0 + w1 * y0,
                     x1 * y0 - y1 * x0 + z1 * w0 + w1 * z0],
dtype=np.float64)
```

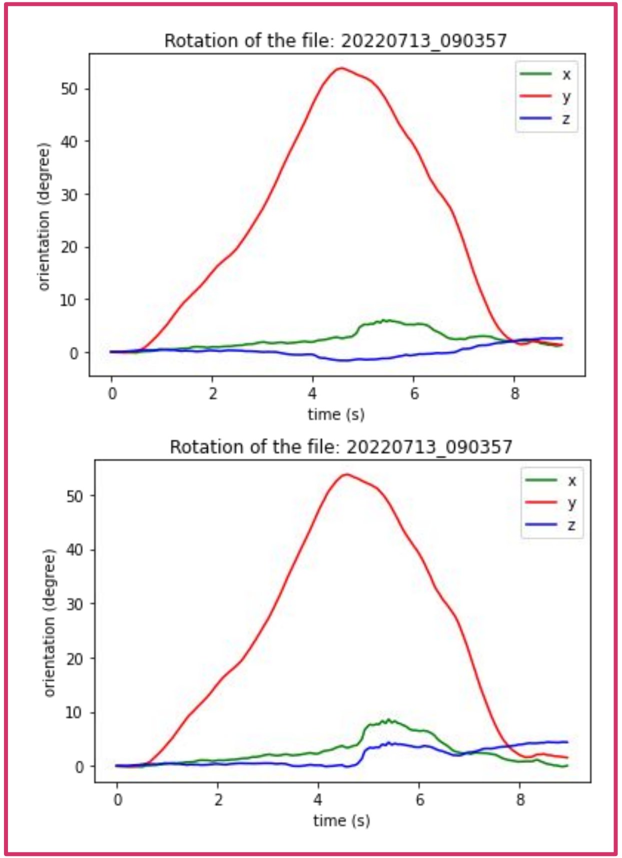
Quaternion Normalization

```
def quaternionNormalize (pose_initial, q_grad, dt=1/400):
    quat = mathutils.Quaternion (np.asarray(pose_initial) + dt * q_grad)
    return quat.normalized()
```

General Output [Gyroscope Integration (top), Quaternion (bottom)]







Error in the Quaternion Approach