Untitled

November 15, 2017

```
In [1]: import numpy as np
        import pickle
        import sys
        import time
        import matplotlib.pyplot as plt
        import mpl_toolkits.mplot3d as a3
        from mpl_toolkits.mplot3d import Axes3D
        from matplotlib.patches import Polygon
        from transforms3d.quaternions import quat2mat,mat2quat
        %matplotlib inline
```

1 Quaternion operation and averaging

```
In [14]: # q n x 4, p 4
         def opMul(q, p):
             if type(q) is list or type(p) is list:
                 q, p = np.asarray(q, dtype='float'), np.asarray(p, dtype='float')
             assert(q.shape[-1]==4 and p.shape[-1]==4)
             return np.concatenate(( q[...,:1]*p[...,:1] - q[...,1:].dot(np.transpo
                                    (q[...,:1]*p[..., 1:] + p[..., :1]*q[..., 1:]+n
                                   axis = -1)
         def opNorm(q):
             return np.linalg.norm(q, axis=-1).reshape(-1,1) + 1e-20

         def opInv(q):
             if type(q) is list:
                 q = np.asarray(q, dtype='float')
             assert(q.shape[-1]==4)
             return np.divide(opConjugate(q), opNorm(q)**2)

         def opLog(q):
             if type(q) is list:
                 q = np.asarray(q, dtype='float')
             assert(q.shape[-1]==4)
             n = opNorm(q)
             norm_qv = opNorm(q[..., 1:4])
             return np.concatenate((np.log(n),
                           np.divide(q[..., 1:4], norm_qv)*np.arccos(q[...
                          ), axis=-1)

def opExp(q):
    if type(q) is list:
        q = np.asarray(q, dtype='float')
    assert(q.shape[-1]==4)
    norm_qv = opNorm(q[..., 1:4])
    return np.exp(q[...,:1])*np.concatenate( (np.cos(norm_qv),
                                              np.divide(q[...,1:4], norm_q
                                             ), axis=-1 )


def opConjugate(q):
    if type(q) is list:
        q = np.asarray(q, dtype='float32')
    assert(q.shape[-1]==4)
    return np.concatenate((q[...,:1],-q[...,1:]), axis=-1)


def hatmapping(w):
    return np.asarray([[0,    -w[3],w[2]],
                       [w[3], 0   ,-w[1]],
                       [-w[2],w[1] ,0]
                      ])
def opRotate(q,s):
    if q.shape[-1] == 4 and  q.ndim==1:
        R = quat2mat(q)
        return np.transpose(R.dot(np.transpose(s)))
    else:
        assert(0)

def test():
    p = np.asarray([[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,0]])
    q = np.asarray([1,2,3,4])
    print opMul(opInv(p),q)
    print 2*opLog(opExp(p/2.))

test()



def weightedAverageQuaternions(Q, w, T, q_average = np.asarray([1,0,0,0],
    # Number of quaternions to average
    Q = np.asarray(Q, dtype='float')
    w = np.asarray(w, dtype='float')

    M = Q.shape[0]
    for i in range(0, T):
        #print '1',opLog(q_average)*2/np.pi*180
        tmp_q = opMul(opInv(q_average[:]),Q)
        #print '2',opLog(q_average)*2/np.pi*180

        #Error rot. vector from quaternion
        tmp_q= 2.*opLog(tmp_q)
        norm_qv =opNorm(tmp_q[...,1:])
        assert(len(tmp_q)==M)
        #print norm_qv/np.pi*180
        # normalize
        qv = np.divide((-np.pi + np.remainder(norm_qv+np.pi, 2*np.pi))*tmp
        #print q_average[1]
        #print qv[:,1]/np.pi*180
        qv = np.sum(qv * w.reshape(-1,1), axis=0)
        qv[...,0] = 0
#         print '+'*10
#         print opLog(q_average)*2/np.pi*180
        if opNorm(qv) < 1e-10:
            return q_average
        q_average = opMul(q_average, opExp(qv/2.))
        #print opLog(q_average)*2/np.pi*180
    return q_average

def test_average():
    q1 = opExp([0,0,0,170./180*np.pi])[0]
    q2 = opExp([0,0,0,270./180*np.pi])[0]
    q3 = opExp([0,0,0,-101./180*np.pi])[0]
    p = np.asarray([q1,q2,q3])
    a = weightedAverageQuaternions(p, [0.33]*3,100)
    return opLog(a)*2/np.pi*180
print test_average()


def plot(path='../report/test.png',test_mode=False):
    plt.figure(figsize=(25,6))
    plt.subplot(1,3,1)
    plt.title('roll')
    if not test_mode:
        start_time = min(vicd['ts'][0,0],imud['ts'][0,0])
    else:
        start_time = imud['ts'][0,0]

    if not test_mode:
        plt.plot(vicd['ts'][0]-start_time, ground_true[:, 0], label='groun
    plt.plot(imud['ts'][0]-start_time, w[:, 0], label='integration result'
    plt.legend(prop={'size': 8})
    plt.xlabel('time')
    plt.ylabel('rad')
```

```
             plt.subplot(1,3,2)
             plt.title('pitch')
             if not test_mode:
                 plt.plot(vicd['ts'][0]-start_time, ground_true[:, 1], label='groun
             plt.plot(imud['ts'][0]-start_time, w[:, 1], label='integration result'
             plt.legend(prop={'size': 8})
             plt.xlabel('time')
             plt.ylabel('rad')

             plt.subplot(1,3,3)
             plt.title('yaw')
             if not test_mode:
                 plt.plot(vicd['ts'][0]-start_time, ground_true[:, 2], label='groun
             plt.plot(imud['ts'][0]-start_time, w[:, 2], label='integration result'
             plt.legend(prop={'size': 8})
             plt.xlabel('time')
             plt.ylabel('rad')
             plt.savefig(path)
```

```
[[ 1.          0.          0.          0.        ]
 [ 1.          0.          0.          0.        ]
 [ 1.          0.          0.          0.        ]
 [ 1.          0.          0.          0.        ]
 [ 1.          0.85714286 -0.57142857  0.28571429]]
[[ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  0.]]
[[   0.    0.    0.  106.]]
```

2 Read data

```
In [207]: def tic():
            return time.time()
          def toc(tstart, nm=""):
            print('%s took: %s sec.\n' % (nm,(time.time() - tstart)))

          def read_data(fname):
            d = []
            with open(fname, 'rb') as f:
              if sys.version_info[0] < 3:
                d = pickle.load(f)
              else:
                d = pickle.load(f, encoding='latin1')  # need for python 3
            return d
          test = True
          dataset="13"
          if test:
            cfile = "../testset/cam/cam" + dataset + ".p"
            ifile = "../testset/imu/imuRaw" + dataset + ".p"
          else:
            cfile = "cam/cam" + dataset + ".p"
            ifile = "imu/imuRaw" + dataset + ".p"
            vfile = "vicon/viconRot" + dataset + ".p"
          ts = tic()
          camd = read_data(cfile)
          imud = read_data(ifile)
          if not test:
            vicd = read_data(vfile)
          toc(ts,"Data import")
```

```
Data import took: 0.983999967575 sec.
```

3 Calcuating bias and helper function to convert imu data

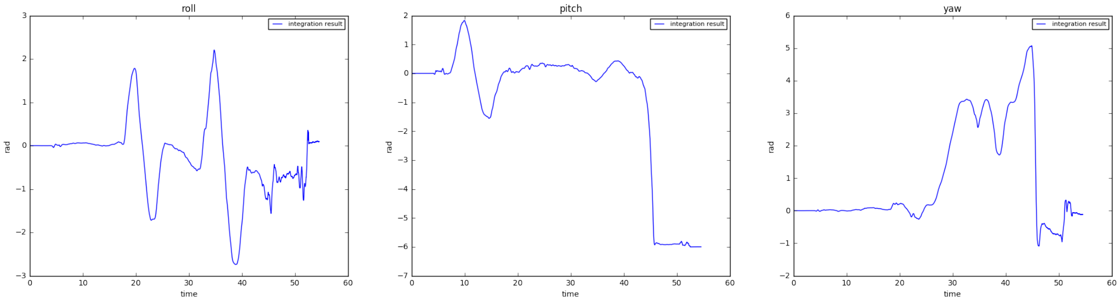
```
In [45]: print camd.keys(), imud.keys()
         imud['vals'] = np.asarray(imud['vals'], dtype='float64')
         #calculating bias
         scaler_w = 3300./1023*np.pi/180/3.33
         scaler_v = 11./1023
         bias = np.zeros(6)
         s = np.sum(imud['vals'][:,:100],axis=-1)/100.
         bias = np.concatenate((s[:3]-np.asarray([0,0,1])/scaler_v, s[3:]))
         def convertData(X):
             assert(X.ndim==1)
             # -Ax, -Ay, -Az, Wz, Wx, Wy
             return (X-bias)*np.concatenate(([scaler_v]*3, [scaler_w]*3))
         print bias
```

```
['ts', 'cam'] ['vals', 'ts']
[ 510.14  501.64  512.94  369.6   373.61  375.39]
```

4 Simple Integration

```
In [208]: def intAngleV(val, time):
              q = [np.asarray([1.,0.,0.,0.])]
              assert(len(time)==len(val))
              for i in range(len(time)-1):
```

```
        t = time[i+1]-time[i]
        w = convertData(np.asarray(val[i],dtype='float64'))[[4,5,3]]
        w = np.concatenate(([0],w))
        tmp_q = opMul(q[-1], opExp(w/2.*t))
        q.append(tmp_q[0])
    return q
q = intAngleV(np.transpose(imud['vals']), imud['ts'][0])
w = (opLog(q)*2)[..., 1:]
if not test:
    ground_true = 2*opLog([ mat2quat(item) for item in np.transpose(vicd

plot('../report/simple'+dataset+'.png',test)
```



5 sampling sigma points and predict

```
In [35]: def sampling(mu, sigma,num=3):
             if type(mu) is list:
                 mu = np.asarray(mu, 'float64')
             # computer how many data points we need
             d = mu.shape[-1]
             # restore data points
             res = [np.asarray(mu,dtype='float64')]
             # compute Cholesky factorization
             lower = np.linalg.cholesky(sigma)
             #print lower*np.sqrt(d)
             assert(lower[0,1]==0)
             for i in range(d):
                 tmp = lower[:,i]*np.sqrt(d)
                 res.append(mu+tmp)
                 res.append(mu-tmp)
             assert(len(res)==(2*d+1))
             return res

         def oneStepTransform(W, control, t=1):
             ## computer transformation Q by control
```

```
    assert(control.ndim==1 and len(control)in[3,4])
    assert(W.shape[-1]==4)
    if len(control) == 3:
        control = np.concatenate(([0],control))
    else:
        control[0] = 0
    return opMul(opExp(W/2.), opExp(control/2.*t))

def predict(data_points, q_t_t, control):
    #data points nx3 control 1x3
    n = len(data_points)
    assert(n==7)
    # weight
    weight_m = np.asarray([0]+[1./(n-1)]*(n-1))
    weight_c = np.asarray([2]+[1./(n-1)]*(n-1)).reshape(-1,1)
    # go one step
    data_points_tt = oneStepTransform(np.concatenate((np.zeros((n,1)),data
    q_tt_t = opMul(q_t_t, data_points_tt)
    ave_q = weightedAverageQuaternions(q_tt_t, weight_m, 200, q_average=q_
    assert(ave_q.shape[-1]==4)
    # compute Sigma
    e = opMul(opInv(ave_q), q_tt_t)
    e = 2*opLog(e)[...,1:]
    sigma_tt_t = np.transpose(e*weight_c).dot(e)
    return ave_q, sigma_tt_t, q_tt_t
```

6 Using predict result to do integration

```
In [51]: def intAngleUsingPredict(val, time):
             q_0 = np.asarray([1,0,0,0])
             sigma_0_0 = np.eye(3)*0.0001
             q = [q_0]
             sig = [sigma_0_0]
             sigma_tt_t = sigma_0_0
             sigma_noise = np.eye(3)*0.0001
             assert(len(time)==len(val))
             for i in range(len(time)-1):
                 t = time[i+1]-time[i]
                 w = convertData(np.asarray(val[i], dtype='float64'))[[4,5,3]]
                 dps = sampling(np.asarray([0,0,0]), sigma_tt_t+sigma_noise)
                 ave_q, sigma_tt_t,_ = predict(dps, q[-1], w*t)
                 q.append(ave_q)
             return q

         num_of_use = 10000
         q = intAngleUsingPredict(np.transpose(imud['vals'])[0:num_of_use], imud['t
         w = (opLog(q)*2)[..., 1:]
         if not test:
             ground_true = 2*opLog([ mat2quat(item) for item in np.transpose(vicd['

         #plot(test_mode=test)
```

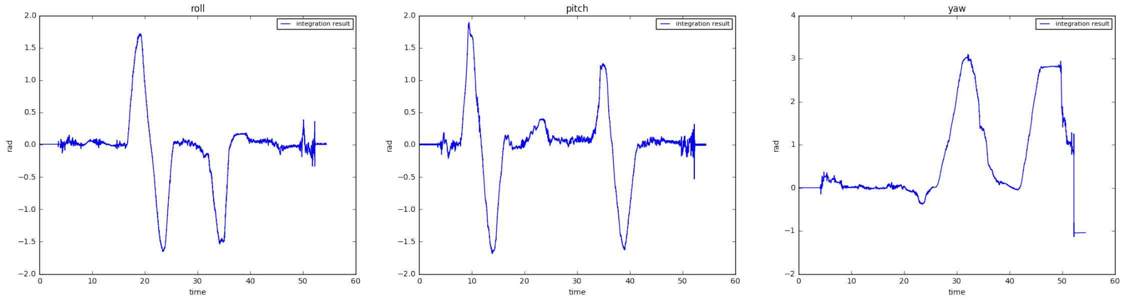
7 update

```
In [22]: def measure_model(q_tt_t, g):
             g = np.asarray(g, dtype='float64')
             Z = []
             for i in range(len(q_tt_t)):
                 Z.append(opMul(opMul(q_tt_t[i], g)[0], opInv(q_tt_t[i]))[0])
             return np.asarray(Z)

         def update(q_tt_t, ave_q_tt_t, Cov_tt_t, z_acc, g=[0,0,0,1], noise=np.eye(
             n = q_tt_t.shape[0]
             g = np.asarray(g)
             # error
             e = 2*opLog(opMul(opInv(ave_q_tt_t), q_tt_t))[...,1:]
             Z_tt = measure_model(q_tt_t, g)[..., 1:]
             #print Z_tt,z_acc
             assert(Z_tt.shape[-1]==3)
             weight_c = np.asarray([2.]+[1./(n-1)]*(n-1)).reshape(-1,1)
             Ave_Z_tt = np.mean(Z_tt[:,:], axis=0)

             Cov_Z_tt = np.transpose((Z_tt-Ave_Z_tt)*weight_c).dot(Z_tt-Ave_Z_tt) +
             Cov_xz_tt = np.transpose(weight_c*e).dot(Z_tt-Ave_Z_tt)
             K_tt = Cov_xz_tt.dot(np.linalg.inv(Cov_Z_tt))
             q_tt_tt = opMul(ave_q_tt_t,
                             opExp(np.concatenate(([0],(K_tt.dot(z_acc-Ave_Z_tt)).r
                             )
             Cov_tt_tt = Cov_tt_t - K_tt.dot(Cov_Z_tt).dot(np.transpose(K_tt))
             return q_tt_tt, Cov_tt_tt
```

```
In [186]: def intAngleUsingPredictUpdate(val, time):
              q_tt_tt = np.asarray([1,0,0,0])
              q = [q_tt_tt]
              # hyper parameter
              Cov_tt_tt = np.eye(3)*0.01
              Oberservation_noise = np.eye(3)*0.001
              Motion_noise = np.eye(3)*0.001
              assert(len(time)==len(val))
              for i in range(len(time)-1):
                  t = time[i+1]-time[i]
                  w = convertData(np.asarray(val[i], dtype='float64'))[[4,5,3]]
```

```
        z_acc = convertData(np.asarray(val[i+1], dtype='float64'))[:3]
        z_acc[:2] = -z_acc[:2]
        if np.linalg.norm(z_acc)>1.5:
            print np.linalg.norm(z_acc),i
        z_acc /= np.linalg.norm(z_acc)
        dps = sampling(np.asarray([0,0,0]), Cov_tt_tt+Motion_noise)
        ave_q_tt_t, Cov_tt_t,q_tt_t = predict(dps, q_tt_tt, w*t)
        q_tt_tt, Cov_tt_tt = update(q_tt_t, ave_q_tt_t, Cov_tt_t, z_acc,
        q.append(ave_q_tt_t)
    return q

num_of_use = 10000
q = intAngleUsingPredictUpdate(np.transpose(imud['vals'])[:num_of_use], i
w = (opLog(q)*2)[..., 1:]
if not test:
    ground_true = 2*opLog([ mat2quat(item) for item in np.transpose(vicd[

plot('../report/update'+dataset+'.png',test)
```

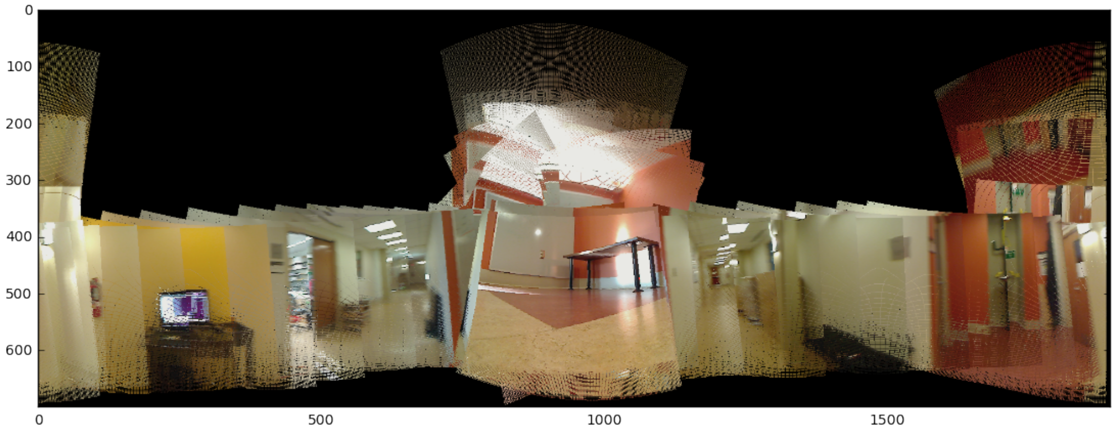
```
1.5105744662 5074
1.51700549668 5075
1.97499903918 5153
1.86696868376 5174
1.55499905478 5190
2.87486491562 5218
1.63706747506 5221
```



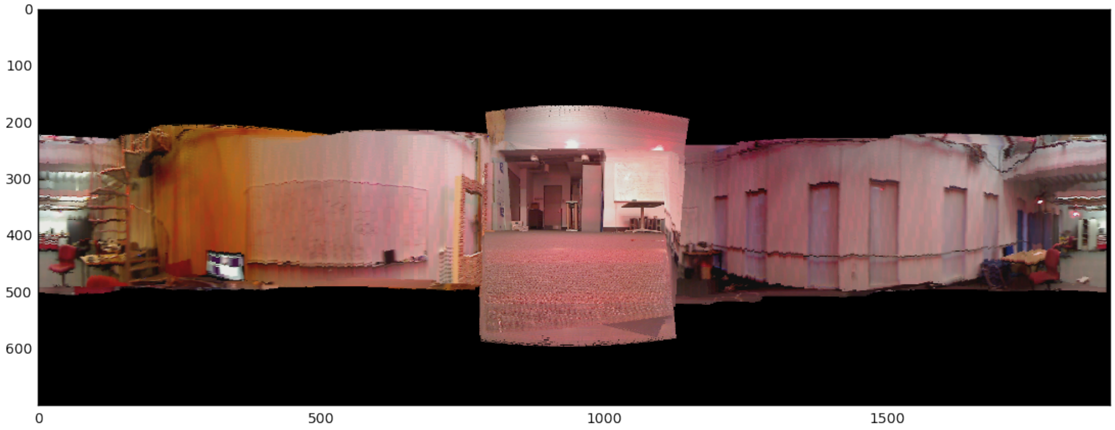
8 Plot panoramic image

```
In [24]: import matplotlib.pyplot as plt
         %matplotlib inline

In [149]: def row2Sep(row, column):
              return ((159.5-column)/320./3*np.pi,(119.5-row)/240.*45/180*np.pi)
```

```
          def sep2Cart(k):
              alpha, beta = k
              # alpha(-30,30) --- beta(-22.5,22.5) ||
              z = np.sin(beta)
              x, y = np.cos(beta,dtype='float64')*np.cos(alpha,dtype='float64'), np
              return x, y, z

          def align(t1, t2):
              return [[i, np.argmin(np.abs(t2-t1[i]))] for i in range(len(t1))]

          def cart2Cylinderm(points,r=200):
              points = np.asarray(points, dtype='float64')
              alpha = -np.arctan(1.0*points[..., 1]/points[..., 0])
              alpha[points[:,0] > 0] += np.pi
              alpha[alpha < 0] += 2*np.pi
              #print np.max(points[:,2]), np.min(points[:,2])
              return np.vstack((np.round(alpha*r),
                                np.round(1.0*(points[:,2])*r/np.linalg.norm(points[:,
                                )

          def draw_panoramic(time_pair, Q, img_set,pic, r=300):
              pic_3d =  np.asarray([sep2Cart(row2Sep(i,j)) for i in range(240) for
              for item in time_pair[::]:
                  img = img_set[..., item[0]].reshape(-1,3)
                  q = Q[item[1]]
                  X_after_rotate = opRotate(q, pic_3d)
                  X_2d = np.transpose(cart2Cylinderm(X_after_rotate,r))
                  X_2d = np.asarray(X_2d, dtype='int')
                  X_2d = X_2d - np.asarray([[0,-300]])
                  if np.max(X_2d[:,1]) > 700 or np.min(X_2d[:,1])<=0:
                      continue
                  pic[X_2d[:,1], X_2d[:,0], :] = img
              plt.figure(figsize=(18,5))
              plt.imshow(pic[::-1,:,:])

In [209]: time_pair = align(camd['ts'][0], imud['ts'][0])
          canvas = np.zeros((700,int(np.floor(2*np.pi*r))+10, 3), dtype='uint8')
          draw_panoramic(time_pair[:], np.asarray(q), camd['cam'], canvas)
```



```
In [64]: # used for ground truth
         time_pair = align(camd['ts'][0], vicd['ts'][0])
         pic_3d =  np.asarray([sep2Cart(row2Sep(i,j)) for i in range(240) for j in
         r = 300
         #pic = np.zeros((700,int(np.floor(2*np.pi*r))+10, 3), dtype='uint8')
         for item in time_pair[::]:
             img = camd['cam'][..., item[0]].reshape(-1,3)
             q = vicd['rots'][...,item[1]]
             q = mat2quat(q)
             X_after_rotate = opRotate(q, pic_3d)
             X_2d = np.transpose(cart2Cylinderm(X_after_rotate,r))
             X_2d = np.asarray(X_2d, dtype='int')
             X_2d = X_2d - np.asarray([[0,-300]])
             if np.max(X_2d[:,1]) > 700 or np.min(X_2d[:,1])<=0:
                 continue
             pic[X_2d[:,1], X_2d[:,0], :] = img
         plt.figure(figsize=(18,5))
         plt.imshow(pic[::-1,:,:])

Out[64]: <matplotlib.image.AxesImage at 0xd2f8128>
```


0
100
200
300
400
500
600
0
500
1000
1500