test_coordonees_perceptives

Laurent Perrinet (INT, UMR7289)

February 14, 2014

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from parametres import VPs, volume, p, d_x, d_y, d_z, calibration, DEBUG
from modele_dynamique import arcdistance, orientation, xyz2azel, rae2xyz
```

```
DEBUG parametres , position croix:  [ 0.          3.76999998  1.37
]
```

In [3]:
```
print xyz2azel.__doc__
```

```
    renvoie le vecteur de coordonnées perceptuelles en fonction des
coordonnées physiques

    xyz = 3 x N x ...

    Le vecteur OV désigne le centre des coordonnées sphériques,
    - O est la référence des coordonnées cartésiennes et
    - V les coordonnées cartesiennes du centre (typiquement du
videoprojecteur).

    cf. https://en.wikipedia.org/wiki/Spherical_coordinates
```

1 Testing spherical coordinates

Plotting function:

In [4]:
```
def plot_xyz2azel(motion, vp=VPs[0],  axes_perc=['t', 'r', 'az', 'el', 'rec'], axes_xy
    """
    Let's define a function that displays for a particular motion
    (a collection of poistions in the x, y, z space --- usually
    continuous) the resulting spherical coordinates (wrt to vp).

    """
    fig = plt.figure(figsize=(18,10))
    t = np.linspace(0, 1, motion.shape[1])[:, np.newaxis]*np.ones((1, motion.shape[2])
    rae_VC = xyz2azel(motion, np.array([vp['x'], vp['y'], vp['z']]))
    motion_rec = rae2xyz(rae_VC, np.array([vp['x'], vp['y'], vp['z']]))
    #print rae_VC.shape
    for i_ax, axe_perc in enumerate(axes_perc):
        for j_ax, axe_xyz in enumerate(axes_xyz):
            #print i_ax, j_ax, 3*i_ax + j_ax
            ax = fig.add_subplot(len(axes_perc), len(axes_xyz), 1 + len(axes_xyz)*i_ax
```

```
            if axe_perc == 't':
                #ax.plot(t, rae_VC[i_ax, :, :])
                #print motion_x[j_ax, :, :].shape, t.shape
                ax.plot(motion[j_ax, :, :], t)
            elif axe_perc == 'rec':
                ax.plot(motion_rec[j_ax, :, :], t)
            else:
                if axe_perc in ['az', 'el']: scale = 180./np.pi
                else: scale = 1.
                #print motion_x[j_ax, :, :].shape, rae_VC[i_ax-1, :, :].shape
                ax.plot(motion[j_ax, :, :], rae_VC[i_ax-1, :, :]*scale)
            ax.plot([vp[axe_xyz]], [0.], 'D')
            ax.set_xlabel(axe_xyz)
            ax.set_ylabel(axe_perc)
    plt.show()
    return rae_VC
print plot_xyz2azel.__doc__
```

```
    Let's define a function that displays for a particular motion
    (a collection of poistions in the x, y, z space --- usually
    continuous) the resulting spherical coordinates (wrt to vp).
```

In [5]:

```
vp = VPs[1]
print vp
```

```
{'cz': 1.36, 'cy': 4.0, 'cx': 11.057115384615384, 'pc_min': 0.001,
'address': '10.42.0.52', 'y': 4.0, 'x': 0.9428846153846154, 'pc_max':
1000000.0, 'z': 1.36, 'foc': 21.802535306060136}
```

1.1 Motion in depth

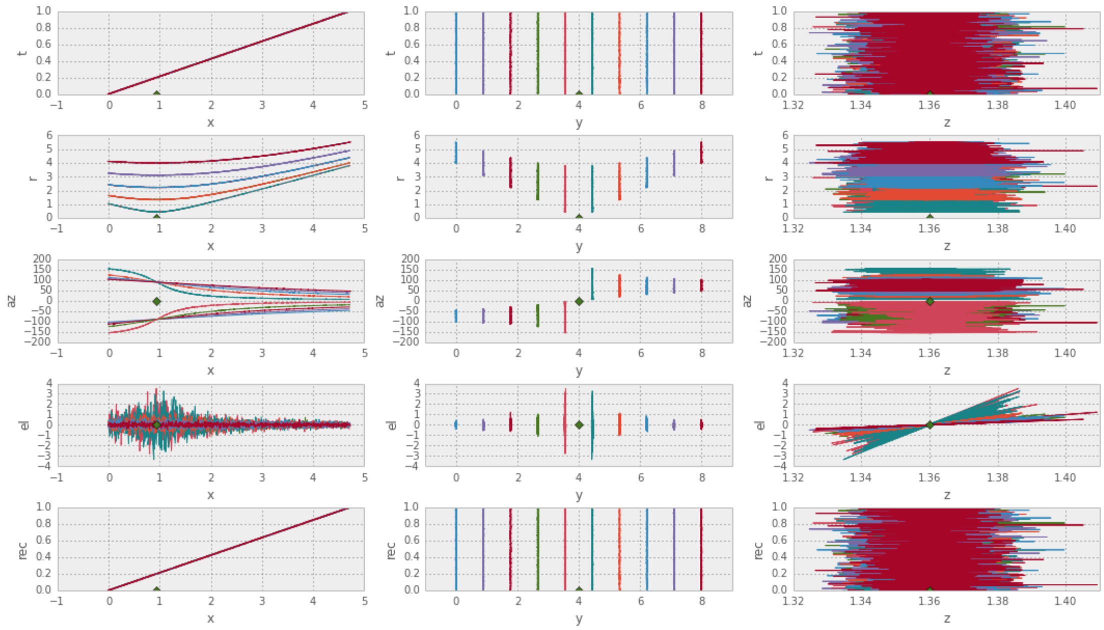
Let's define N_player trajectories of N_t points, where players are distributed in the width (y) and move in the axis of the VP (x):

In [6]:

```
N_player, N_t = 10, 1000
```

In [13]:

```
x = vp['x']*(np.linspace(0., 5., N_t)[np.newaxis, :, np.newaxis]*np.ones((1, 1, N_play
y = vp['y']*np.ones((1, N_t, 1))*np.linspace(0, 2, N_player, endpoint=True)[np.newaxis
z = vp['z']*np.ones((1, N_t, N_player)) # constant
#print x.shape, y.shape, z.shape
motion_x = np.vstack((x, y, z))
motion_x += .01*np.random.randn(3, N_t, N_player)
motion_x.shape
rae_VC = plot_xyz2azel(motion_x, vp=vp)
```



```
note : when azimuth is uperior to 180° in a
```

In [ ]:

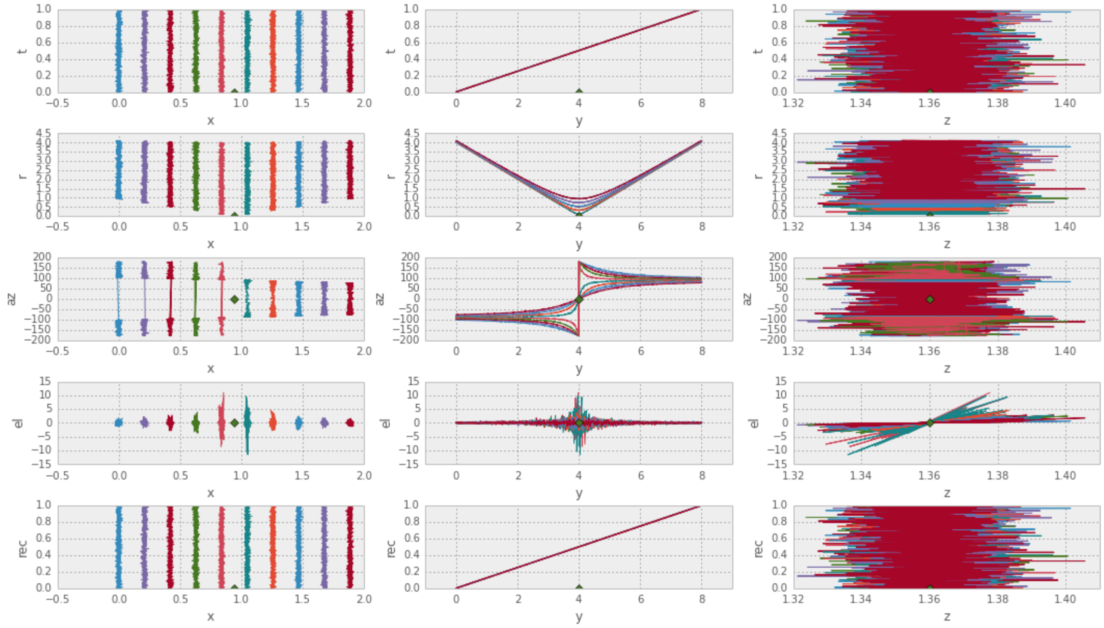
Note the small elevation:

In [8]:
```
el = rae_VC[2,:,:]
print el.min(), el.max(),  el.mean(),  el.std()
```

```
-0.0509226184347 0.0601199550504 -9.90535830335e-05 0.00584778894562
```

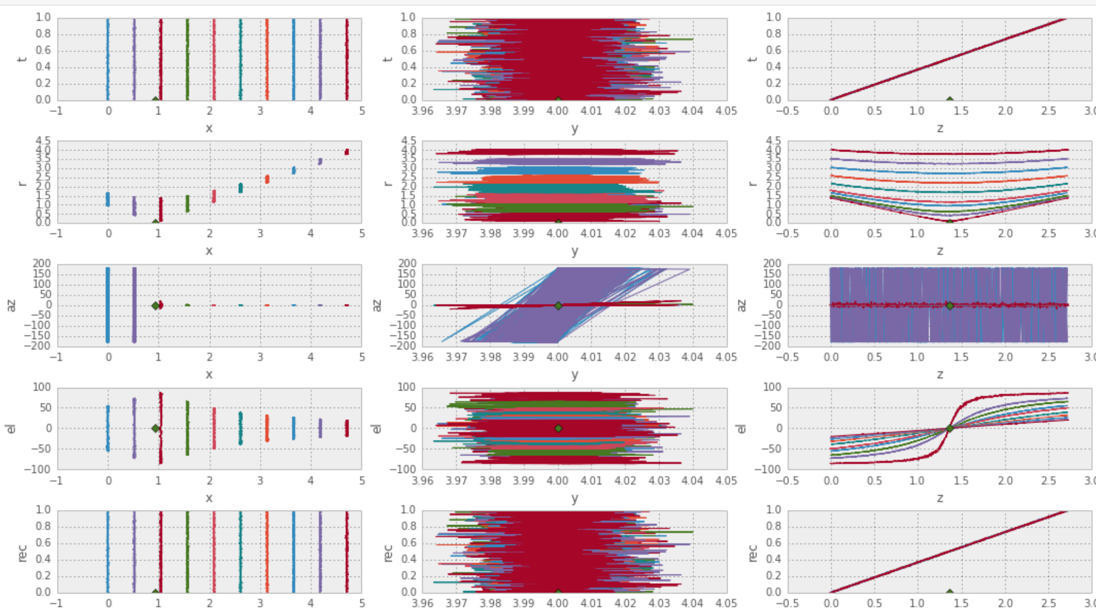
1.2 Motion in width

In [9]:
```
x = vp['x']*np.ones((1, N_t, 1))*np.linspace(0., 2., N_player, endpoint=True)[np.newax
y = vp['y']*(np.linspace(0, 2, N_t)[np.newaxis, :, np.newaxis]*np.ones((1, 1, N_player
z = vp['z']*np.ones((1, N_t, N_player)) # constant
motion_y = np.vstack((x, y, z))
motion_y += .01*np.random.randn(3, N_t, N_player)
rae_VC = plot_xyz2azel(motion_y, vp=vp)
```



1.3 Motion in height

In [10]:
```
x = vp['x']*np.ones((1, N_t, 1))*np.linspace(0, 5., N_player, endpoint=True)[np.newaxi
y = vp['y']*np.ones((1, N_t, N_player)) # constant
z = vp['z']*(np.linspace(0, 2., N_t)[np.newaxis, :, np.newaxis]*np.ones((1, 1, N_playe
motion_z = np.vstack((x, y, z))
motion_z += .01*np.random.randn(3, N_t, N_player)
rae_VC = plot_xyz2azel(motion_z, vp=vp)
```



2 Testing great circle navigation

Similar tests, but now looking at the `arcdistance` and `orientation` functions:

In [11]:
```
print arcdistance.__doc__
```

```
    renvoie l'angle sur le grand cercle (en radians)

    # rae1 ---> rae2

     r = distance depuis le centre des coordonnées sphériques (mètres)
     a =  azimuth = declinaison = longitude (radians)
     e =  elevation = ascension droite = lattitude (radians)

    http://en.wikipedia.org/wiki/Great-circle_distance
    http://en.wikipedia.org/wiki/Vincenty%27s_formulae
```

In [12]:
```
print orientation.__doc__
```

```
    renvoie le cap suivant le grand cercle (en radians)

     r = distance depuis le centre des coordonnées sphériques (mètres)
     a =  azimuth = declinaison = longitude (radians)
     e =  elevation = ascension droite = lattitude (radians)

     http://en.wikipedia.org/wiki/Great-circle_navigation
                #http://en.wikipedia.org/wiki/Haversine_formula
```

In [12]:
```

```