plot_desired_velocity

Mohcine Chraibi

March 31, 2014

Part I

Fast plotting of trajectories

Given a file with the following data:

In [2]:
```
# v0_x   v0_y   pos_x  pos_y   rv0_x   rv0_y   id
# 56.00 101.85 52.40 101.85 56.000000 101.850000 1
# 56.00 100.74 50.60 100.74 56.000000 100.740000 2
# 56.00 100.74 52.10 100.74 56.000000 100.740000 3
# 56.00 102.96 52.40 102.96 56.000000 102.960000 4
# 56.00 100.74 55.40 100.74 56.000000 100.740000 5
# 56.00 103.70 54.80 103.70 56.000000 103.700000 6
# 56.00 101.85 53.00 101.85 56.000000 101.850000 7
# 56.00 102.22 52.10 102.22 56.000000 102.220000 8
# 56.00 101.11 55.40 101.11 56.000000 101.110000 9
# 56.00 101.11 54.80 101.11 56.000000 101.110000 10
# ...
```

The point v^0=(v^0_x, v^0_y) is the direction of pedestrian id.
(rv^0_x, rv^0_y) is the rotation of v^0 around the position of the pedestrion (pos_x, pos_y)

First some imports and check if the program is called with the proper arguments

In [3]:
```python
from multiprocessing import Pool, current_process
import numpy as np
import matplotlib.pyplot as plt
from sys import argv
import pandas as pd
import os, time
#------------ set input ----------
filename = "plot_desired_velocity/log.txt"
nproc = 2
```

The arguments are:

- *filename*: The file containing the data to be ploted
- *proc*: The number of processors to use.

We are going to use multiprocessing, so we better make a function for ploting one frame:

In [4]:
```
def plotFrame(args):
    data, i = args
    fig = plt.figure()
    for line in data:
        v0x = line[0]
        v0y = line[1]
        x = line[2]
        y = line[3]
        Gx= line[4]
        Gy= line[5]
        plt.plot((x), (y), 'og')
        plt.arrow(x, y, v0x-x, v0y-y, head_width=0.1, head_length=0.2, fc=
        plt.arrow(x, y, Gx-x, Gy-y, head_width=0.1, head_length=0.2, fc='k
        #geometry
        plt.plot([65, 62, 62, 60, 60, 56], [104,104,103, 103, 104, 104], '
        plt.plot([65, 62, 62, 60, 60, 56], [100,100,102, 102, 100, 100], '
        plt.xlim((48,66))
        plt.ylim((99,105))
        plt.annotate('%d'%line[6], xy=(x, y), xytext=(x+0.5, y+0.5),
                     arrowprops=dict(facecolor='black', shrink=0.05),
                     )
    fig.savefig("plot_desired_velocity/figs/%.4d.png"%i)
    plt.clf()
    print "---> plot_desired_velocity/figs/%.4d.png"%i
```

In order to make things run faster (we have no time!), we make a pool of **nproc** processes and map the data **FRAME-WISE** to the plot function

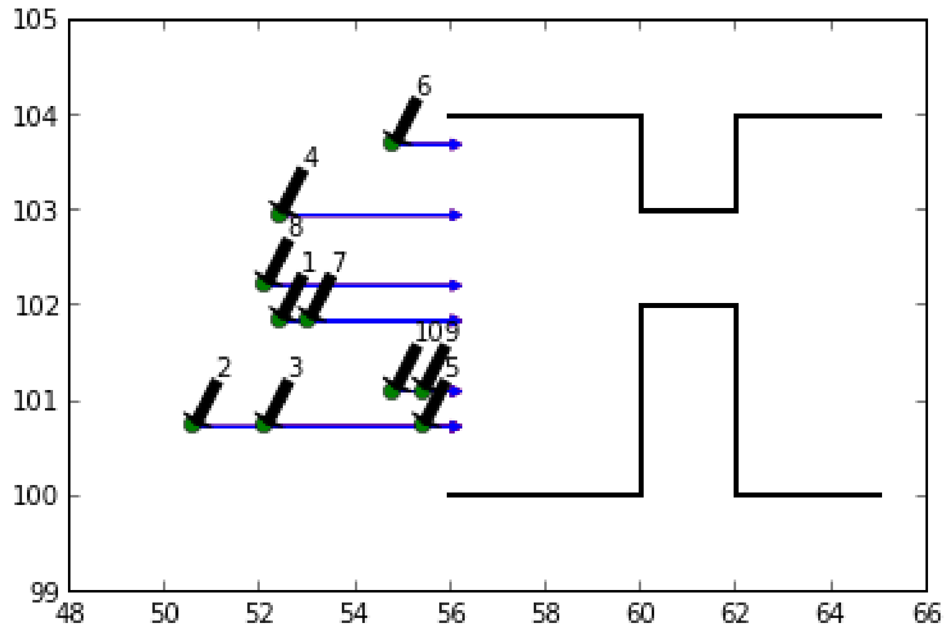
In [5]:
```
if __name__ == "__main__":
    Plot = 1
    Movie = 1

    if Plot:
        start = time.time()
        print "read_csv file ..."
        data = np.array( pd.read_csv(filename, sep = " ", header=None)  )
        readTime = time.time() - start
        m = int( max(data[:,-1]) ) #
        l = data.shape[0]
        pool = Pool(processes = nproc)
        nfigs = range(0, l/m, 100)
        list_data = [ data[i*m:(i+1)*m, :] for i in nfigs ]
        inputData = zip(list_data, range(len(nfigs)))
        start = time.time()
        pool.map(plotFrame, inputData)
        paTime = time.time() - start
        pool.close()
        pool.join()
        print "Read Data in %.4f /s"%readTime
        print "Run Time %.4f /s"%paTime
```

```
read_csv file ...
Read Data in 0.0026 /s
Run Time 0.1747 /s
---> plot_desired_velocity/figs/0000.png
```

In [6]:
```
from IPython.display import Image
Image(filename='plot_desired_velocity/figs/0000.png')
```

Out [6]:



Optional: Make a movie using the pngs produced in the directory **figs/**

In [7]:
```
if Movie:
    cmd = "\"mf://plot_desired_velocity/figs/*.png\" -mf w=800:h=600:fps=2
    print "Run Mencoder ... "
    os.system("mencoder %s" %cmd)
```

```
Run Mencoder ...
```

Todo

Use PyQtGraph insteed of the *very* slow `matplotlib`