昵称：rubbninja
园龄：1年11个月
粉丝：18
关注：3
+加关注

< 2017年4月 >

日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

搜索

找找看

谷歌搜索

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

我的标签

机器学习(13)

学习笔记(12)

室内定位(7)

室内定位系列(7)

卡尔曼滤波(2)

C++(2)

ipython(1)

ipython notebook(1)

Java(1)

python(1)

更多

室内定位系列（六）——目标跟踪（粒子滤波）

进行目标跟踪时，先验知识告诉我们定位轨迹是平滑的，目标当前时刻的状态与上一时刻的状态有关，滤波方法可以将这些先验知识考虑进来得到更准确的定位轨迹。本文简单介绍粒子滤波及其使用，接着卡尔曼滤波写，建议先阅读室内定位系列（五）——目标跟踪（卡尔曼滤波）。

原理

这里跟卡尔曼滤波进行对比来理解粒子滤波。

目标跟踪中的卡尔曼滤波的简化版解释：

定位跟踪时，可以通过某种定位技术（比如位置指纹法）得到一个位置估计（观测位置），也可以根据我们的经验（运动目标常常是匀速运动的）由上一时刻的位置和速度来预测出当前位置（预测位置）。把这个观测结果和预测结果做一个加权平均作为定位结果，权值的大小取决于观测位置和预测位置的不确定性程度，在数学上可以证明在预测过程和观测过程都是线性高斯时，按照卡尔曼的方法做加权是最优的。

上面提到的"线性高斯"需要从概率上来理解。

- 预测过程中，预测的当前位置（或状态）$\hat{x}_k^-$与上一时刻的位置（或状态）是线性高斯关系，即所谓的运动方程：

  $$\hat{x}_k^- = A\hat{x}_{k-1} + u_k + \omega_k$$

  其中的外界输入u_k有时可以是零，预测不一定准确，因此存在一个高斯的误差ω_k。从概率分布上来看，当前位置在预测点附近的概率较大，越远概越小。

- 观测过程中，观测值z_k与真实状态x_k之间也是线性高斯关系，即观测方程：

  $$z_k = H_k x_k + n_k$$

  目标跟踪中的观察矩阵H可能就是1，观察值也并不完全准确，因此也存在一个高斯误差n_k。从概率分布上来看，当前位置在观测点附近的概率较大，越远概越小。

预测过程和观测过程都有一个目标位置的概率分布，我们应该取一个联合概率最大的位置作为估计值。如果经验预测和实际观测满足这样的线性高斯，比如经验中目标是匀速运动的，那么直接使用卡尔曼滤波做加权就可以了。

然而，很多情况下并不是线性高斯的，比如以下这种情形：我们除了知道目标常常匀速运动，还知道地图信息，如果目标的前方有一堵墙，就不应该继续用匀速运动（加高斯噪声）来进行预测，应该有一个完全不一样的预测值的分布（比如反向，或者朝墙的同侧随机一个方向，或者沿着墙的边缘运动），这个分布取决于我们经验是怎样的，它有可能是某个预测值附近的高斯分布，也可能是某些用公式无法描述的分布。

既然用公式无法描述，那就用蒙特卡洛方法来模拟：模拟大量的粒子，每个粒子都有一个状态（位置）和权重，所有这些粒子的状态分布和权值共同模拟了目标位置（或状态）的概率分布，可以直接把所有粒子的状态做加权平均得到估计值。在预测过程中，根据经验给粒子设置一些规则（进行状态转移），比如让每个粒子匀速运动，遇到墙就反向，同时再加上一点随机性，这样就能完美地模拟各种经验和规则了。然后，用观察结果带来的概率分布去更新各个粒子的权重，更加匹配观测结果的粒子应该赋予更大的权重。

更多参考：
filterpy文档——粒子滤波
Particle Filter Tutorial 粒子滤波：从推导到应用

步骤

随笔档案

2017年1月 (1)

2016年12月 (4)

2016年11月 (2)

2016年2月 (2)

2016年1月 (2)

2015年11月 (3)

2015年10月 (4)

2015年7月 (3)

2015年5月 (2)

C++基础

最新评论

1. Re:室内定位系列（五）——目标跟踪（卡尔曼滤波）

博主可以加我微信吗？ 我的名字叫吴志国，微信号：18813157365

--gguo_2017

2. Re:室内定位系列（五）——目标跟踪（卡尔曼滤波）

@gguo_2017不错哦！...

--rubbninja

3. Re:室内定位系列（五）——目标跟踪（卡尔曼滤波）

您好！我是北邮研二在读学生，研究方向室内定位，目前有个国内室内定位交流群，主要成员都是在读的研究生和博士生，还有部分的国外研究生博士生。如果博主有兴趣，欢迎加入！希望和博主多交流学习。室内定位交流群号.....

--gguo_2017

4. Re:室内定位系列（一）——WiFi位置指纹（译）

@king-blues没有。很多分类器都可以用来代替最基本的knn，但感觉没这个必要。后续会尝试各种分类器来做这个，不过都用现成的包。...

--rubbninja

可以将粒子滤波理解成一个滤波的框架，框架内部应该根据实际问题具体实现。

1. **t = 0时，粒子初始化。**随机生成粒子集并设置权值。
2. **t = 1, 2, ...， 重复以下步骤：**
   a. **预测。**根据系统的预测过程预测各个粒子的状态。
   b. **更新。**根据观测值更新粒子权值。
   c. **重采样。**复制一部分权值高的粒子，同时去掉一部分权值低的粒子。
   d. **输出：状态估计。**使用粒子和权值估计当前的状态。

实践

使用python工具包filterpy来实现卡尔曼滤波和粒子滤波，对knn位置指纹法的定位结果进行滤波。数据来源说明

Github地址

导入数据

```
# 导入数据
import numpy as np
import scipy.io as scio
offline_data = scio.loadmat('offline_data_random.mat')
online_data = scio.loadmat('online_data.mat')
offline_location, offline_rss = offline_data['offline_location'],
offline_data['offline_rss']
trace, rss = online_data['trace'][0:1000, :], online_data['rss'][0:1000, :]
del offline_data
del online_data
```

```
# 定位精度
def accuracy(predictions, labels):
    return np.mean(np.sqrt(np.sum((predictions - labels)**2, 1)))
```

KNN + Kalman Filter

```
# knn回归
from sklearn import neighbors
knn_reg = neighbors.KNeighborsRegressor(40, weights='uniform', metric='euclidean')
knn_reg.fit(offline_rss, offline_location)
knn_predictions = knn_reg.predict(rss)
acc = accuracy(knn_predictions, trace)
print "accuracy: ", acc/100, "m"
```

```
accuracy:  2.24421479398 m
```

```
# 对knn定位结果进行卡尔曼滤波

from filterpy.kalman import KalmanFilter
from scipy.linalg import block_diag
from filterpy.common import Q_discrete_white_noise
def kalman_tracker():
    tracker = KalmanFilter(dim_x=4, dim_z=2)
    dt = 1.
    # 状态转移矩阵
    tracker.F = np.array([[1, dt, 0,  0],
                          [0,  1, 0,  0],
                          [0,  0, 1, dt],
                          [0,  0, 0,  1]])
    # 用filterpy计算Q矩阵
    q = Q_discrete_white_noise(dim=2, dt=dt, var=0.001)
    # tracker.Q = block_diag(q, q)
    tracker.Q = np.eye(4) * 0.01
    # tracker.B = 0
    # 观测矩阵
    tracker.H = np.array([[1., 0, 0, 0],
                          [0, 0, 1., 0]])
    # R矩阵
    tracker.R = np.array([[4., 0],
                          [0, 4.]])
    # 初始状态和初始P
    tracker.x = np.array([[7.4, 0, 3.3, 0]]).T
    tracker.P = np.zeros([4, 4])
    return tracker
```

5. Re:室内定位系列（一）——WiFi位置指纹（译）

你好，请问目前是否用到PSO训练ANN的方式来作指纹定位算法。

--king-blues

阅读排行榜

1. 室内定位系列（一）——WiFi位置指纹（译）(1789)

2. 统计信号处理-简单看看克拉美罗界(1016)

3. Tensorflow使用环境配置(896)

4. 室内定位系列（〇）——从人耳听觉定位原理到室内定位技术(689)

5. 室内定位系列（五）——目标跟踪（卡尔曼滤波）(634)

评论排行榜

1. 统计信号处理-简单看看克拉美罗界(3)

2. 室内定位系列（五）——目标跟踪（卡尔曼滤波）(3)

3. 室内定位系列（一）——WiFi位置指纹（译）(2)

推荐排行榜

1. 统计信号处理-简单看看克拉美罗界(2)

2. 室内定位系列（一）——WiFi位置指纹（译）(2)

3. 使用Java练习算法常用的基本操作(1)

4. 室内定位系列（〇）——从人耳听觉定位原理到室内定位技术(1)

5. 小技能——markdown(1)

```
tracker = kalman_tracker()
zs = np.array([np.array([i]).T / 100. for i in knn_predictions]) # 除以100，单位为m
mu, cov, _, _ = tracker.batch_filter(zs) # 这个函数对一串观测值滤波
knn_kf_predictions = mu[:, [0, 2], :].reshape(1000, 2)
acc = accuracy(knn_kf_predictions, trace / 100.)
print "accuracy: ", acc, "m"
```

```
accuracy:  1.76116239607 m
```

KNN + Particle Filter

```
# knn回归
from sklearn import neighbors
knn_reg = neighbors.KNeighborsRegressor(40, weights='uniform', metric='euclidean')
knn_reg.fit(offline_rss, offline_location)
knn_predictions = knn_reg.predict(rss)
acc = accuracy(knn_predictions, trace)
print "accuracy: ", acc/100, "m"
```

```
accuracy:  2.24421479398 m
```

```
# 设计粒子滤波中各个步骤的具体实现

from numpy.random import uniform, randn, random, seed
from filterpy.monte_carlo import multinomial_resample
import scipy.stats
seed(7)

def create_particles(x_range, y_range, v_mean, v_std, N):
    """这里的粒子状态设置为（坐标x，坐标y，运动方向，运动速度）"""
    particles = np.empty((N, 4))
    particles[:, 0] = uniform(x_range[0], x_range[1], size=N)
    particles[:, 1] = uniform(y_range[0], y_range[1], size=N)
    particles[:, 2] = uniform(0, 2 * np.pi, size=N)
    particles[:, 3] = v_mean + (randn(N) * v_std)
    return particles

def predict_particles(particles, std_heading, std_v, x_range, y_range):
    """这里的预测规则设置为：粒子根据各自的速度和方向（加噪声）进行运动，如果超出边界则随机改变方向再次尝试，"""
    idx = np.array([True] * len(particles))
    particles_last = np.copy(particles)
    for i in range(100): # 最多尝试100次
        if i == 0:
            particles[idx, 2] = particles_last[idx, 2] + (randn(np.sum(idx)) *
std_heading)
        else:
            particles[idx, 2] = uniform(0, 2 * np.pi, size=np.sum(idx)) # 随机改变方向
        particles[idx, 3] = particles_last[idx, 3] + (randn(np.sum(idx)) * std_v)
        particles[idx, 0] = particles_last[idx, 0] + np.cos(particles[idx, 2] ) *
particles[idx, 3]
        particles[idx, 1] = particles_last[idx, 1] + np.sin(particles[idx, 2] ) *
particles[idx, 3]
        # 判断超出边界的粒子
        idx = ((particles[:, 0] < x_range[0])
                | (particles[:, 0] > x_range[1])
                | (particles[:, 1] < y_range[0])
                | (particles[:, 1] > y_range[1]))
        if np.sum(idx) == 0:
            break

def update_particles(particles, weights, z, d_std):
    """粒子更新，根据观测结果中得到的位置pdf信息来更新权重，这里简单地假设是真实位置到观测位置的距离为高斯分布"""
    # weights.fill(1.)
    distances = np.linalg.norm(particles[:, 0:2] - z, axis=1)
    weights *= scipy.stats.norm(0, d_std).pdf(distances)
    weights += 1.e-300
    weights /= sum(weights)

def estimate(particles, weights):
    """估计位置"""
    return np.average(particles, weights=weights, axis=0)

def neff(weights):
    """用来判断当前要不要进行重采样"""
```

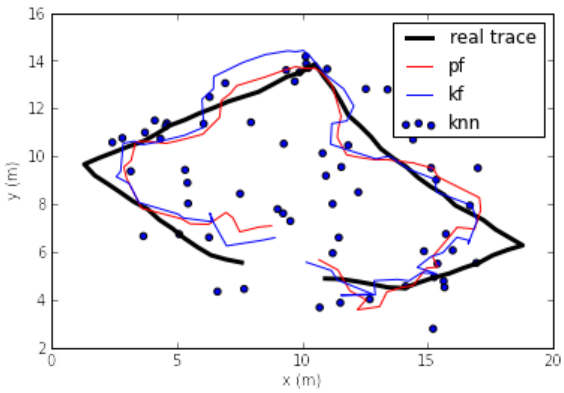
```
    return 1. / np.sum(np.square(weights))

def resample_from_index(particles, weights, indexes):
    """根据指定的样本进行重采样"""
    particles[:] = particles[indexes]
    weights[:] = weights[indexes]
    weights /= np.sum(weights)

def run_pf(particles, weights, z, x_range, y_range):
    """迭代一次粒子滤波，返回状态估计"""
    x_range, y_range = [0, 20], [0, 15]
    predict_particles(particles, 0.5, 0.01, x_range, y_range) # 1. 预测
    update_particles(particles, weights, z, 4) # 2. 更新
    if neff(weights) < len(particles) / 2: # 3. 重采样
        indexes = multinomial_resample(weights)
        resample_from_index(particles, weights, indexes)
    return estimate(particles, weights) # 4. 状态估计
```

```
# 对knn定位结果进行粒子滤波

knn_pf_predictions = np.empty(knn_predictions.shape)
x_range, y_range = [0, 20], [0, 15]
n_particles = 50000
particles = create_particles(x_range, y_range, 0.6, 0.01, n_particles) # 初始化粒子
weights = np.ones(n_particles) / n_particles # 初始化权重

for i, pos in enumerate(knn_predictions):
    pos = pos.copy() / 100.
    state = run_pf(particles, weights, pos, x_range, y_range)
    knn_pf_predictions[i, :] = state[0:2]

acc = accuracy(knn_pf_predictions, trace / 100.)
print "final state: ", state
print "accuracy: ", acc, "m"
```

```
final state:  [  8.16137026  12.49569879   4.06952385   0.54954716]

accuracy:  1.80881825483 m
```

轨迹对比

```
import matplotlib.pyplot as plt
%matplotlib inline
x_i = range(220, 280)
tr, = plt.plot(trace[x_i, 0] / 100., trace[x_i, 1] / 100., 'k-', linewidth=3)
pf, = plt.plot(knn_pf_predictions[x_i, 0], knn_pf_predictions[x_i, 1], 'r-')
kf, = plt.plot(knn_kf_predictions[x_i, 0], knn_kf_predictions[x_i, 1], 'b-')
knn_ = plt.scatter(knn_predictions[x_i, 0] / 100., knn_predictions[x_i, 1] / 100.)
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.legend([tr, pf, kf, knn_], ["real trace", "pf", "kf", "knn"])
plt.show()
```



作者：rubbninja
出处：http://www.cnblogs.com/rubbninja/
关于作者：目前主要研究领域为机器学习与无线定位技术，欢迎讨论与指正！

标签：室内定位，室内定位系列，卡尔曼滤波，粒子滤波

好文要顶 关注我 收藏该文

rubbninja
关注 - 3
粉丝 - 18
+加关注

0 0

« 上一篇：室内定位系列（五）——目标跟踪（卡尔曼滤波）

posted @ 2017-01-06 14:00 rubbninja 阅读(462) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

发表评论

昵称：1bit

评论内容：

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]

最新IT新闻:
· 固态硬盘要长命绝招：千万别点磁盘碎片！
· GitLab：为什么我们开源了员工手册
· 计算机预测技术可搭配新型磁性材料
· 百度大脑给李彦宏新书写序，你看写得如何？
· 开发者Peter Molyneux：50多岁重学代码
» 更多新闻...

最新知识库文章:
· 程序员，如何从平庸走向理想？
· 我为什么鼓励工程师写blog