浙江大学 实验报告

课程名称：智能移动技术　　指导老师：　熊蓉，王越　　成绩：
实验名称：EKF 定位　　姓名：潘力豪

一、实验内容

在 Matlab 中，利用 ekf_localization.m 文件，实现移动机器人的 EKF 定位算法。

二、设计思路：

语言：matlab

①算法设计思路：

该实验需要应用扩展卡尔曼滤波实现里程计和 GPS 的融合。原理 PPT 所示如下：

Algorithm Extended_Kalman_filter $(\mu_{t-1}, \Sigma_{t-1}, u_{t-1}, z_t)$

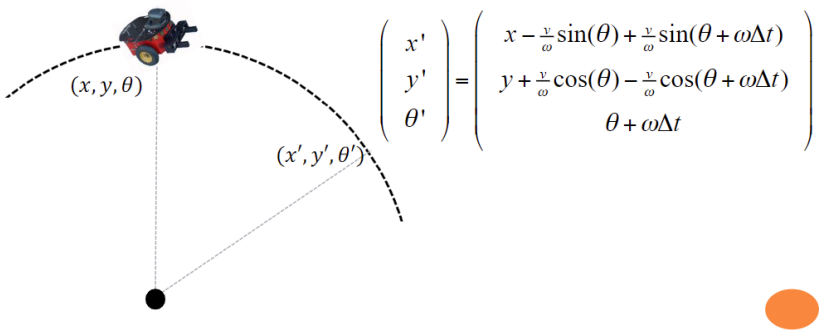
$$\bar{\mu}_t = g(\mu_{t-1}, u_{t-1})$$
$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \longrightarrow O(k^{2.376})$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = [I - K_t H_t] \bar{\Sigma}_t \longrightarrow O(n^2)$$
$$return \ \mu_t, \Sigma_t$$

其中前两行属于运动预测，后三行属于观测更新。

对于小车的运动模型，采用了之前讲过的模型进行代码编写，原理如下：

机器人的速度控制运动模型



对于 Jacobian 矩阵以及观测矩阵的代码编写如 PPT 中所写。

②代码详细解释

对于扩展卡尔曼滤波的主要步骤如主循环的代码中所示：

```
% ------ Kalman Filter --------
% Predict
% ?
x_p = doMotion(xEkf,u);
G_t = jacobF(xEkf,u);
convQ = G_t * convR *G_t' + noiseQ;
% Update
% xEkf=?
H = jacobH(x_p);
S = H*convQ*H'+noiseR;
K = convQ*H'*inv(S);
xEkf = x_p +K*(z - doObservation(z,x_p));
convR = (eye(3)-K*H)*convQ;
```

而在之前的初始化中定义了初始的运动模型和观测模型的协方差矩阵 noiseR 和 noiseQ，并在该代码中不断进行更新。

运动模型，协方差矩阵等的求解都如下列代码所示，原理与之前所述的相同：

```
function x = doMotion(x, u)
    global dt;
    %?
    vel = sqrt(u(1)*u(1)+u(2)*u(2));
    vel_w = vel / u(3);
    temp_1 = -vel_w*sin(x(3))+vel_w*sin(x(3)+u(3)*dt);
    temp_2 = vel_w*cos(x(3))-vel_w*cos(x(3)+u(3)*dt);
    temp_3 = dt*u(3);
    temp = [temp_1 temp_2 temp_3]';
    x = x + temp;
end

% Jacobian of Motion Model
function jF = jacobF(x, u)
    global dt;
    vel = sqrt(u(1)*u(1)+u(2)*u(2));
    vel_w2 = vel / u(3);
    %dist = sqrt(u(1)*u(1) + u(2)*u(2))*dt;
    jF = [1 0 -vel_w2*cos(x(3))+vel_w2*cos(x(3)+u(3)*dt)
          0 1 -vel_w2*sin(x(3))+vel_w2*sin(x(3)+u(3)*dt)
          0 0 1  ];
    %?
end

%Observation Model
function x = doObservation(z, xPred)
    m = [1 0 0
        0 1 0
        0  0 1];
    x=m*xPred;
    %?
 end

%Jacobian of Observation Model
function jH = jacobH(x)
    %?
    jH = [1 0 0
         0 1 0
         0 0 1 ];
end
```

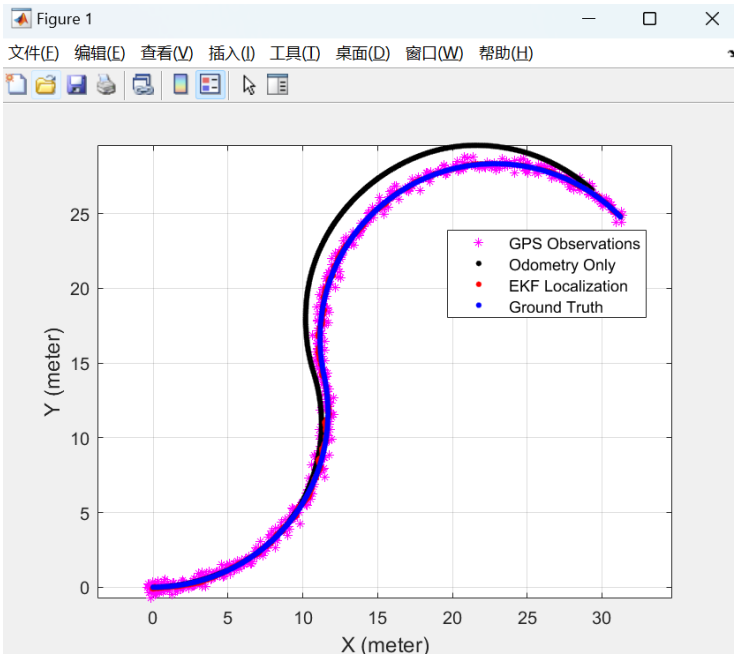
实验最后在命令行窗口输出终点的真值，EKF 值与纯里程计的坐标值，并根据轨迹输出纯里程计与 EKF 融合后的路径平均误差。代码如下：

```
num = size(estimation.xTruth,1);
fprintf("终点站的真值：x = %f,y = %f\n",estimation.xTruth(num,1),estimation.xTruth(num,2));
fprintf("终点站的EKF：x = %f,y = %f\n",estimation.xEkf(num,1),estimation.xEkf(num,2));
fprintf("终点站的里程计Odom：x = %f,y = %f\n",estimation.xOdom(num,1),estimation.xOdom(num,2));
miao = 1;error_Odom = 0;error_Ekf = 0;

while miao <= num
    error_Odom = error_Odom + sqrt((estimation.xOdom(miao,1)-estimation.xTruth(miao,1)).^2 + (estimation.xOdom(miao,2)-estimation.xTruth(miao,2)).^2);
    error_Ekf = error_Ekf +sqrt((estimation.xEkf(miao,1)-estimation.xTruth(miao,1)).^2 + (estimation.xEkf(miao,2)-estimation.xTruth(miao,2)).^2);
    miao = miao + 1;
end
disp(["num=",num]);
fprintf("Odom累计平均误差：%f\n",error_Odom/num);
fprintf("Ekf累计平均误差：%f\n",error_Ekf/num);
```

三、实验结果

实验后的路径图如下：



控制台输出：

```
>> ekf_localization
EKF Start!
终点站的真值：x = 31.227747,y = 24.782587
终点站的EKF：x = 31.254028,y = 24.782574
终点站的里程计Odom：x = 29.334766,y = 26.578839
    "num="    "600"

Odom累计平均误差：1.028250
Ekf累计平均误差：0.071028
```

由于仿真中存在随机误差，因此每一次代码运行平均误差都不同，但是每一次 Ekf 累计平均误差相比于 Odom 的累计平均误差要小一个数量级。可以看出 Ekf 相比于纯里程计能够很好的减少定位误差。