基于非配对数据的惯性传感器信号处理分析及增强

摘要

传感器系统是智能机器人、智能家居、智能驾驶等智能系统不可或缺的组成部分。但在实际情况下，造价昂贵的高精度传感器并不能大规模应用，造价低廉的低成本传感器的信号包含复杂的噪声和误差，不能满足大多数系统的要求。本文分析了两组分别来自高、低成本的惯性传感器采集的三轴加速度和陀螺仪数据。由于获取的数据并不配对，且没有真实值作为参考，故本文采用了常用于分析陀螺仪误差的动态 Allan 方差法（DAVAR），通过对比两组传感器的 Allan 方差值随时间变化的图像，得出高成本传感器和低成本传感器的性能区别在于高成本传感器对于微小的运动状态变化有更明显的反应，受到噪声产生的误差少；而低成本传感器容易受到噪声干扰，产生随机误差，测量精度低。

根据以上结论，本文从处理传感器数据的角度，尝试实现对低成本传感器的信号增强。为了减少低成本传感器信号中的噪声等干扰，本文采用了经典的 Kalman 滤波算法对低成本传感器的信号进行处理，并将处理后的信号与原始信号，以及滤波后信号的动态 Allan 方差值与原始的动态 Allan 方差值分别进行对比，发现处理后的信号变得更加平滑，并且 Allan 方差值突升的问题得到了有效抑制，证明了 Kalman 滤波对于处理低成本传感器的有效性，能使低成本传感器信号具备高成本传感器信号的如噪声干扰较少、误差偏移较小等特征，实现了低成本传感器的信号增强。在实际应用中，低成本传感器也可以通过加入滤波器提升自身信号可靠性，有利于投入到更多智能系统的使用中。

关键词: 惯性传感器；信号处理；Allan 方差；DAVAR 算法；Kalman 滤波

Contents

1 问题重述

1.1 问题背景

随着新一轮科技革命浪潮的到来，高度信息化的社会正愈发接近实现，而一切智能化和自动化都离不开各式各样的高精度传感器。因此传感器的精度对未来社会的智能化程度起到了决定性的作用。虽然低成本的传感器在误差和噪声等方面远远逊色于高精度传感器，但是受制于成本，这些相对低精度的传感器仍然是想要满足大规模使用需求不得不依赖的。因此，如何比较不同传感器的特点并探寻其规律进而实现低成本传感器信号的增强就成了一个具有现实意义的问题。

1.2 问题要求

问题一：现在有若干采集过程彼此独立的低成本与高成本惯性传感器的信号，要求对两者进行对比分析，实现对高精度传感器优势及低成本传感器误差的建模与分析。

问题二：根据问题一的分析，要求借助一定的信号处理方法，使低成本惯性传感器的信号在处理之后能够具备高精度惯性传感器信号的部分特征，从而实现低成本惯性传感器的信号增强。

2 问题分析

2.1 问题一的分析

问题一提供了两种不同精度传感器信号的非配对数据，要求对两者进行对比并给出优势和误差的建模。这个问题的特点在于所提供的数据都是非配对且属于不同过程的，故没有真实数值用来计算传感器信号的离散程度。而在惯性传感器的信号输出中随机误差又是普遍存在的，故本文尝试借助经典的随机误差识别方法——Allan 方差来对传感器输出信号的误差进行分析。同时采用动态 Allan 方差算法（DAVAR）进一步分析信号数据的误差情况。然后将两种传感器的信号质量综合对比，进而得出高精度传感器的优势及低精度传感器的不足。

2.2 问题二的分析

通过问题一的分析，本文已经初步发现低成本和高成本传感器在信号上的特点与区别，即高成本传感器的感知更加灵敏，能够更加精准地反映一些细小的变化，而低成本传感器在灵敏度方面不足，并且低成本传感器输出的信号受噪声等因素干扰的程度明显更重。在查阅了有关文献后，本文选择使用 Kalman 滤波算法对低成本传感器的数据进行处理，减少噪声对传感器的干扰，进而实现低成本传感器信号的增强。

3 模型假设

1. 假设对传感器信号数值产生影响的误差主要来自量化噪声、角度随机游走、零偏不稳定性、角速度随机游走、速率斜坡。在这种情况下，传感器的信号输出变化仅考虑为外界条件变化和上述误差所导致。

2. 假设 Allan 方差和 DAVAR 算法在本题的情景下能够有效反映传感器在题目给出的条件下所产生的误差的情况，能够通过对比结果成为评价传感器性能的指标。

4 符号说明

符号	含义	单位
$\hat{a}$	回归方程预测值	/
ω	传感器加速度测量值	m/s^2
ω	传感器角速度测量值	rad/s
σ	Allan 标准差	/
N_w	Allan 方差矩形框长度	m
t	相关时间	s
$\hat{x}_k$	k 时刻的后验状态估计值	/
$\hat{x}_{\bar{k}}$	k 时刻的先验状态估计值	/
P_k	k 时刻的后验估计协方差	/
$P_{\bar{k}}$	k 时刻的先验估计协方差	/
H	转换矩阵	/
z_k	观察值	/
K_k	滤波增益矩阵	/
A	状态转移矩阵	/
Q	过程激励噪声协方差	/
R	测量激励噪声协方差	/
B	将输入转化为状态的矩阵	/

5 模型建立与求解

5.1 数据预处理

首先对所给出的传感器信号的数据进行预处理，借此来分析可能出现的异常值。在统计学中箱形图是一种常用的分析数据的离散分布情况及异常值的方法，能直观明了地给出了数据批中的异常值和数据的大致分布情况。图中矩形部分数据为下四分位数和上四分位数之间的数据分布，上下的丁字型细线则是根据四分位距计算出的非异常范围内的最大值和最小值（即上下限）。圆点则表示出现的超出上下限的数据。本文采用了对数据绘制箱形图的方法来判断数据的分散程度以及可能出现的异常值/极端值，用来判断数据的可信度。下面是不同数据组别的部分统计图实例。

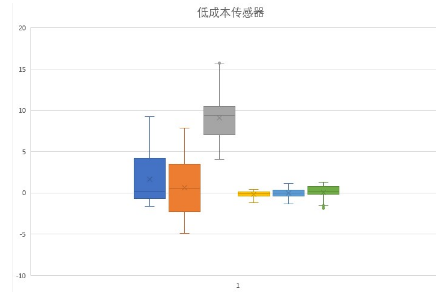

Figure 1: 低成本传感器数据箱形图一

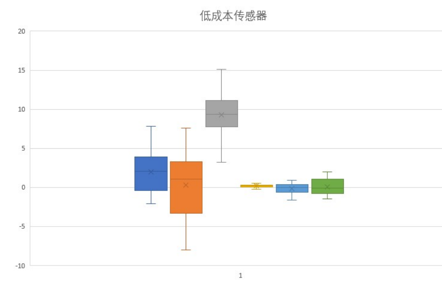

Figure 2: 低成本传感器数据箱形图二

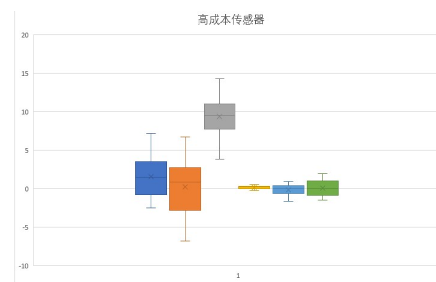

Figure 3: 高成本传感器数据箱形图一

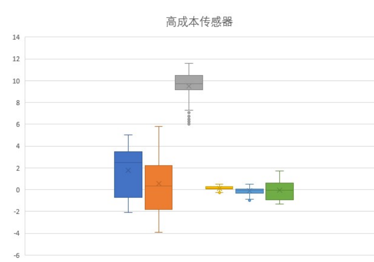

Figure 4: 高成本传感器数据箱形图二

以上图中出现异常点的数目极少，所以通过对箱形图的初步分析可以认为所给出的数据整体上

是可以采用的，可行度较高。并且鉴于极端异常值的占比非常少且整体数据样本量较大，可以认为不用再进行异常数据的去除。

5.2 问题一的模型建立与求解

Allan 方差是 1966 年由美国国家标准局的 David Allan 为了研究原子钟振荡器的稳定性提出的一种计算方法，该方法后来被推广至分析陀螺仪的随机误差。因此本文引进了该算法对两种传感器的数据进行对比分析。

设以 0 为采样时间对陀螺仪输出角速率进行采样，共采样 N 个点，把所获得的 N 个数据分成 K 组，每组包含 M 个采样点，具体如下。

$$\underbrace{\omega_1, \omega_2, \cdots, \omega_M}_{K=1}$$
$$\underbrace{\omega_{M+1}, \omega_M+2, \cdots, \omega_2 M}_{K=2}$$
$$\underbrace{\omega_{N-M+1}, \omega_N-M+2, \cdots, \omega_N}_{K=K}$$

每一组的持续时间 $t=Mt_0$ 称之为相关时间，对于第 k 个子集，其平均值可以表示为

$$\overline{\Omega_k}(t)=\frac{1}{M}\cdot\sum_{i=1}^{M}\omega_{(k-1)M+i}$$

其中

$$k=1,2,\cdots,K$$

对每个不同平均时间计算 Allan 方差：

$$\sigma_A^2(t)=\frac{1}{2}\left\langle\left(\Omega_{k+1}(M)-\Omega_k(M)\right)^2\right\rangle$$

$$=\frac{1}{2(K-1)}\sum_{k=1}^{K-1}\left(\Omega_{k+1}(M)-\Omega_k(M)\right)^2$$

其中 $\langle\cdot\rangle$ 表示对无限时间序列的总体求平均值。对于不同的相关时间，可求得相应的 Allan 方差。Allan 方差的平方根通常被称为 Allan 均方差。

传统 Allan 方差虽然对于分析传感器有效，能对各种误差源统计特性进行细致的表征和辨识，但是其只适用于处理理想的平稳信号。实际情况中，传感器会因为环境变化的敏感性而使其信号的动态误差随时间变化且不易预测 [1]。

为了解决这个问题，本文引进了动态 Allan 方差（DAVAR）算法。DAVAR 算法是 Allan 方差法的扩展和改进，其大致过程如下：首先，对于一个给定的点，采用以其为中心的固定长度截取原始信号，暂不考虑窗口以外的数据，然后，以截取样本为研究对象，计算其 Allan 方差。通过移动窗口分段估计，呈现不同时间的 Allan 方差估计结果，最终将所有的 Allan 方差按顺序排列起来，以一种更直观的方式将信号稳定性体现出来。下面定义 DAVAR 为随时间变化的 Allan 方差曲线簇，即

$$\sigma_A^2(n,m)=\frac{1}{2m^2\tau_0^2}\frac{1}{N-2m}\times$$

$$\sum_{k=n-N_w/2}^{n+N_w/2\ -2m\ -1}\left[\theta(k+2m)-2\theta(k+m)+\theta(k)\right]^2$$

m=1,2,$\cdots$,N_w/2-1[2]。其中 N_w 为所选择的矩形框长度。

DAVAR 算法可以反应惯性传感器的随机误差的时变特性，能够更好地验证惯性传感器动态性能分析的有效性，能够更直观体现设备使用环境变化、设备中的电磁干扰、设备温度变化以及传感器内部模块的性能动态变化，分析其动态性能。

下面给出两种不同传感器的 DAVAR 方差统计图表。

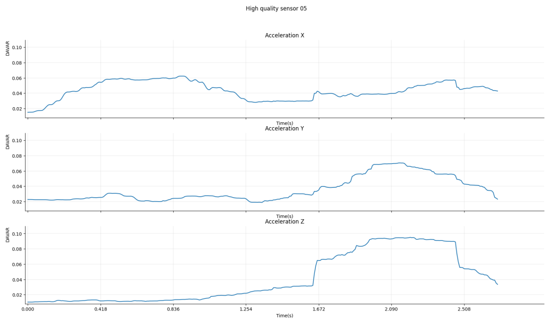

Figure 5: 高成本传感器 05 组加速度的 DAVAR 结果

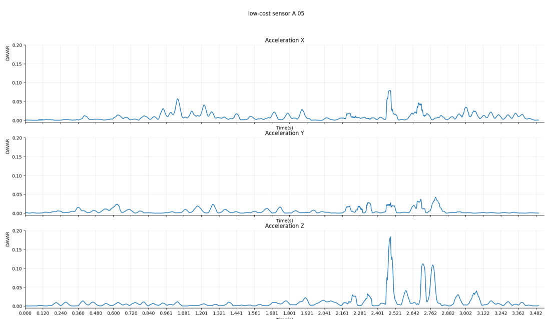

Figure 6: 低成本传感器 A05 组加速度的 DAVAR 结果

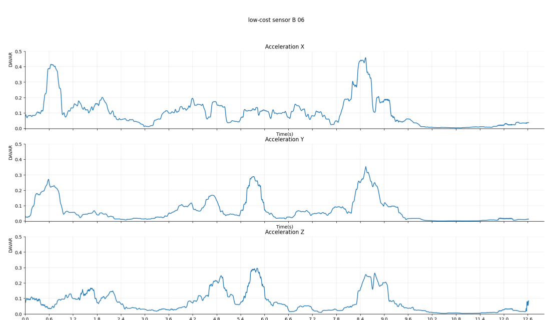

Figure 7: 低成本传感器 B06 组加速度的 DAVAR 结果

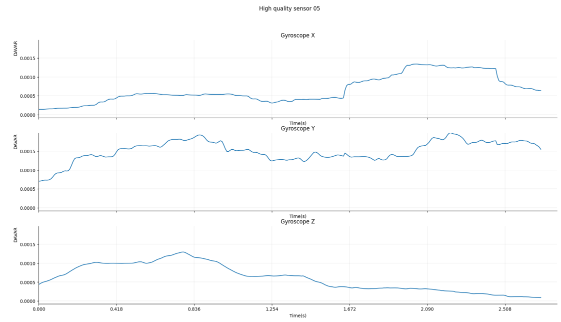

Figure 8: 高成本传感器 05 组角速度的 DAVAR 结果

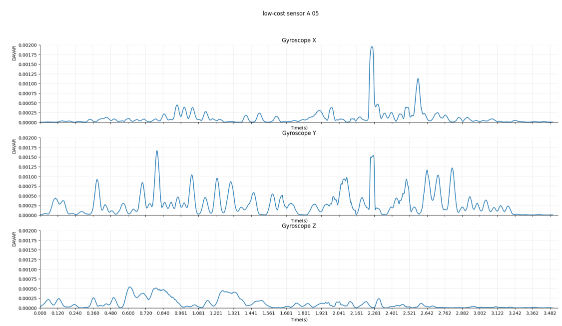

Figure 9: 低成本传感器 A05 组角速度的 DAVAR 结果

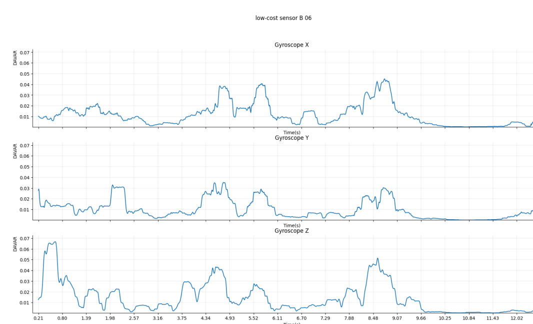

Figure 10: 低成本传感器 B06 组角速度的 DAVAR 结果

分析对比各个图表中 Allan 方差随时间变化的情况，注意到高精度的传感器的 Allan 方差值容

易发生变化，且整体处于较低水平；而低精度传感器的 Allan 方差值常维持在较高水平，且变化不明显。这说明高精度传感器对于微小变化的体现更加明显，能收集到更细微精确的数据，从而实现观测任务。而低成本传感器受到环境变化、使用方式改变等情况的干扰，容易产生随机误差，反应不灵敏，相对更难检测到细微的变化。

5.3 问题二的模型建立与求解

受到制造成本的限制，低成本传感器的硬件难以大幅升级，但是可以将传感器将收集到的数据进行特定处理后再输出，以减少其中的误差，从而使低成本的传感器性能更加接近高成本传感器。

为了尽可能去除传感器信号中的噪声，信号应经过滤波处理。本文采用经典的 Kalman 滤波算法对低成本传感器的数据进行处理。下面给出 Kalman 滤波算法的数学推导。

Kalman 滤波器时间更新方程如下 [3]：

$$\hat{x}_{\bar{k}} = A\hat{x}_{k-1} + Bu_{k-1}$$

$$P_{\bar{k}} = AP_{k-1}A^T + Q$$

Kalman 滤波器状态更新方程如下：

$$K_k = \frac{P_k - H^T}{HP_k - H^T + R}$$

$$\hat{x}_k = \hat{x}_{\bar{k}} + K_k\left(z_k - H\hat{x}_{\bar{k}}\right)$$

$$P_k = \left(I - K_kH\right)P_{\bar{k}}$$

根据以上的公式推导，本文建立了对应的计算机程序对原始数据进行滤波处理。下面给出经 Kalman 滤波处理前后的数据对照图。

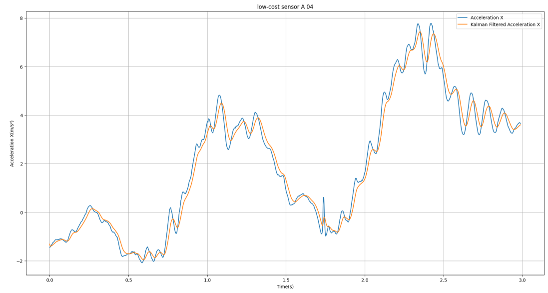

Figure 11: 低成本传感器 A04 组滤波前后的 X 轴加速度对照表

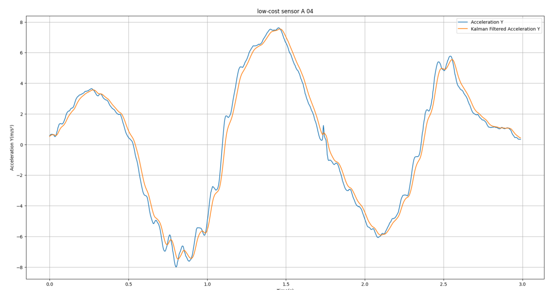

Figure 12: 低成本传感器 A04 组滤波前后的 Y 轴加速度对照表

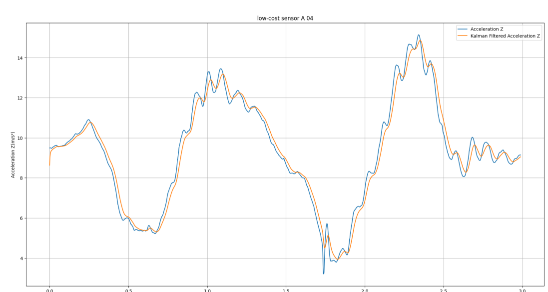

Figure 13: 低成本传感器 A04 组滤波前后的 Z 轴加速度对照表

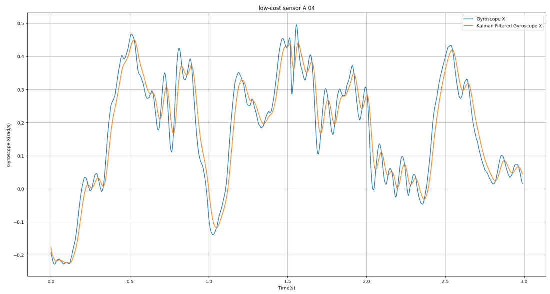

Figure 14: 低成本传感器 A04 组滤波前后的 X 轴角速度对照表

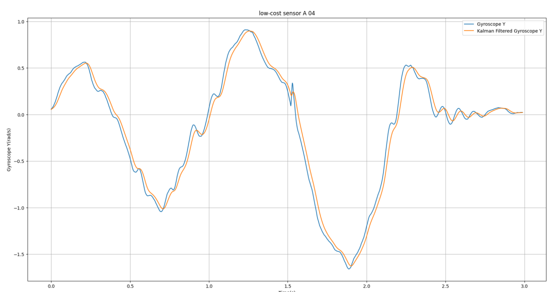

Figure 15: 低成本传感器 A04 组滤波前后的 Y 轴角速度对照表

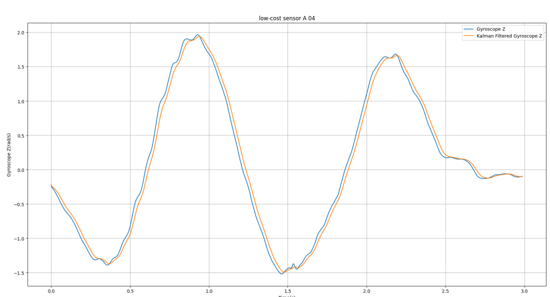

Figure 16: 低成本传感器 A04 组滤波前后的 Z 轴角速度对照表

对比滤波前后的数据图，可以得出使用滤波器处理数据后，信号曲线明显变得更加平滑。当数据出现剧烈抖动时，滤波器能削弱这种抖动，减少可能存在的外界干扰；当数据整体变化平稳但存在小范围抖动时，滤波器并且能完全抑制抖动以消除噪声干扰。

下面给出滤波前后 DAVAR 的数据对比图：

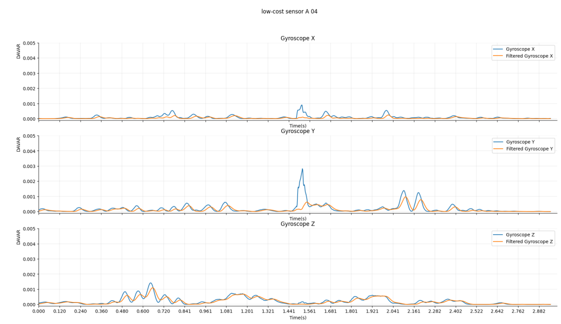

Figure 17: 低成本传感器 A04 组滤波前后的角速度 DAVAR 对照表

Figure 18: 低成本传感器 A04 组滤波前后的加速度 DAVAR 对照表

分析滤波前后的动态 Allan 方差数值，可以发现在不同的数据采集时间和采集环境下采集的数据中各项随机误差在经过滤波器后都有了明显的降低，并且对于高 DAVAR 值的出现有抑制作用。惯性传感器滤波前后的 DAVAR 曲线的变化趋势是相同的，幅值整体下降至较低的水平。

因此，可以得出采用 Kalman 滤波后，低成本传感器的数据可信度得到了提升，信号中的噪声得

到了有效抑制，从而具备了高精度传感器的部分特征的结论。

6 模型的特色、改进与推广

本文根据题目数据的实际情况选取了惯性传感器误差分析中常见的 Allan 方差和 DAVAR 方法对传感器的输出值进行分析，所得到的不同传感器的结果有着明显的特点，便于对传感器的评价。同时本文还采用了经典的 Kalman 滤波法对低成本传感器的信号数据进行处理，对于过滤杂波和减少随机误差的影响有着一定的效果。

此模型的有两处可改进的地方。在问题一中，可以将 DAVAR 中的采样间隔和窗口宽度作为第三个坐标轴，生成三维图形，使分析更具说服力；在问题二中，可以通过调整参数，使 Kalman 滤波后的数据时延尽可能小。

本文所采用的 Kalman 滤波模型在实际工程与制造中也具有滤波抗噪的应用价值，例如可以在实际使用中给传感器加入滤波器，以使处理后的信号输出更能反映真实情况。

参考文献

[1] Zhang Qian,Zhang Qian,et al..Research on Random Errors of Fiber Optic Gyro Based on Dynamic Allan Variance and Algorithm Improvement[J].ACTA OPTICA SINICA,2015.4
张谦，王伟等. 基于动态 Allan 方差的光纤陀螺随机误差分析及算法改进 [J]. 光学学报,2015.4.

[2] Guo Gang.Research on Random Error Analysis and Signal Processing of Fiber Optic Gyroscope[J].University of Chinese Academy of Sciences,2016.5.
郭港. 光纤陀螺随机误差辨识及信号处理技术研究. 中国科学院光电技术研究所,2016.5

[3] ZENG Jingcong.Real-time Structural Displacement Estimation by Fusing Acceleration and Displacement Data with Adaptive Kalman Filter [J].Advanced Engineering Sciences,2022.1.
曾竞骢. 基于自适应卡尔曼滤波加速度与位移融合的结构位移实时估计 [J]. 工程科学与技术,2022.1.

A 附录

以下是本文所用的程序

A.1 Allan 方差及 DAVAR 计算程序一

```python
import csv
import numpy
from matplotlib import pyplot as plt

def Allan(_list, n):
    list = numpy.asarray(_list)
    length = len(list)
    K = length // n
    sigma = []

    for i in range(0, length-1, n):
        s = slice(i,i+n)
        global listtemp
        listtemp = list[s]
```

```
        avg = numpy.mean(listtemp)
        sigma.append(avg)
    del(listtemp)
    sum = 0
    for j in range(0, K-2):
        sum += (sigma[j+1]-sigma[j]) ** 2
    delta = sum / 2 / (K - 1)
    return delta

def DAVAR(_dataList, n, windowWidth):
    AllanList = []
    length = len(_dataList)
    for index in range(windowWidth, length - windowWidth):
        datatemp = _dataList[index - windowWidth:index + windowWidth]
        Allantemp = Allan(datatemp, n)
        AllanList.append(Allantemp)
        del(datatemp)

    return AllanList # 最终，AllanList中将含(length-2windowWidth+1)个元素

def GetSampleTime(timelist):
    sampleNum = len(timelist)
    timeInterval = [timelist[j+1] - timelist[j] for j in range(0, sampleNum-1)]
    _timeInterval = numpy.asarray(timeInterval)
    avgSampleTime = numpy.mean(_timeInterval)
    return avgSampleTime

def DisplayGraph(graph, _title, ylist):
    graph.set(title=_title, xlabel='Time(s)')
    graph.set_ylabel('DAVAR', loc='top')
    graph.axis([timeStart, timeEnd, yMin, yMax])
    graph.plot(timeAxis, ylist)
    graph.spines['top'].set_visible(False)
    graph.spines['right'].set_visible(False)
    graph.grid(linewidth=0.2)

if __name__ == '__main__':
    filename = "low-cost sensor A 05.csv"
    time = []
    accx = []
    accy = []
    accz = []
    gyrox = []
    gyroy = []
    gyroz = []

    # 读取文件
    with open(filename, mode="r", newline="", encoding="utf8") as f:
        reader = csv.reader(f)
        header = next(reader) # 文件的第一行是列名行
        for row in reader:
            time.append(float(row[0]))
            accx.append(float(row[1]))
            accy.append(float(row[2]))
            accz.append(float(row[3]))
            gyrox.append(float(row[4]))
            gyroy.append(float(row[5]))
            gyroz.append(float(row[6]))

    # 计算DAVAR
    n = 2
    windowWidth = 10
    DAVAR_accX = numpy.asarray(DAVAR(accx, n, windowWidth))
    DAVAR_accY = numpy.asarray(DAVAR(accy, n, windowWidth))
    DAVAR_accZ = numpy.asarray(DAVAR(accz, n, windowWidth))

    n = 2
    windowWidth = 10
    DAVAR_gyroX = numpy.asarray(DAVAR(gyrox, n, windowWidth))
    DAVAR_gyroY = numpy.asarray(DAVAR(gyroy, n, windowWidth))
    DAVAR_gyroZ = numpy.asarray(DAVAR(gyroz, n, windowWidth))

    sampleTime = GetSampleTime(time)
    sampleNum = len(DAVAR_accX)
    timeAxis = numpy.linspace(0, sampleNum*sampleTime, sampleNum, True)
```

```

    # 绘图
    timeStart = 0
    timeEnd = time[-1]
    yMin = -0.005
    yMax = 0.2
    xInterval = 60
    figure1, (graph_AccX, graph_AccY, graph_AccZ) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True)

    DisplayGraph(graph_AccX, 'Acceleration X', DAVAR_accX)
    DisplayGraph(graph_AccY, 'Acceleration Y', DAVAR_accY)
    DisplayGraph(graph_AccZ, 'Acceleration Z', DAVAR_accZ)
    plt.xticks(timeAxis[::xInterval])
    plt.suptitle(filename[0:-4])
    plt.show()

    yMin = -0.000
    yMax = 0.002
    figure2, (graph_GyroX, graph_GyroY, graph_GyroZ) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True)
    DisplayGraph(graph_GyroX, 'Gyroscope X', DAVAR_gyroX)
    DisplayGraph(graph_GyroY, 'Gyroscope Y', DAVAR_gyroY)
    DisplayGraph(graph_GyroZ, 'Gyroscope Z', DAVAR_gyroZ)
    plt.xticks(timeAxis[::xInterval])
    plt.suptitle(filename[0:-4])
    plt.show()
```

A.2 Allan 方差及 DAVAR 计算程序二

```
import csv
import numpy
from matplotlib import pyplot as plt

def Allan(_list, n):
    list = numpy.asarray(_list)
    length = len(list)
    K = length // n
    sigma = []

    for i in range(0, length-1, n):
        s = slice(i,i+n)
        global listtemp
        listtemp = list[s]
        avg = numpy.mean(listtemp)
        sigma.append(avg)
    del(listtemp)
    sum = 0
    for j in range(0, K-2):
        sum += (sigma[j+1]-sigma[j]) ** 2
    delta = sum / 2 / (K - 1)
    return delta

def DAVAR(_dataList, n, windowWidth):
    AllanList = []
    length = len(_dataList)
    for index in range(windowWidth, length - windowWidth):
        datatemp = _dataList[index - windowWidth:index + windowWidth]
        Allantemp = Allan(datatemp, n)
        AllanList.append(Allantemp)
        del(datatemp)

    return AllanList

def GetSampleTime(timelist):
    sampleNum = len(timelist)
    timeInterval = [timelist[j+1] - timelist[j] for j in range(0, sampleNum-1)]
    _timeInterval = numpy.asarray(timeInterval)
    avgSampleTime = numpy.mean(_timeInterval)
    return avgSampleTime

def DisplayGraph(graph, _title, ylist, timeList, timeStart, timeEnd):
    graph.set(title=_title, xlabel='Time(s)')
    graph.set_ylabel('DAVAR', loc='top')
```

```python
    graph.axis([timeStart, timeEnd, yMin, yMax])
    graph.plot(timeList, ylist)
    graph.spines['top'].set_visible(False)
    graph.spines['right'].set_visible(False)
    graph.grid(linewidth=0.2)

if __name__ == '__main__':
    time1 = []
    time2 = []
    accx = []
    accy = []
    accz = []
    gyrox = []
    gyroy = []
    gyroz = []

    # 读取文件
    filename1 = "low-cost sensor B 06 Acc.csv"
    with open(filename1, mode="r", newline="", encoding="utf8") as f:
        reader = csv.reader(f)
        header = next(reader)
        for row in reader:
            time1.append(float(row[0]))
            accx.append(float(row[1]))
            accy.append(float(row[2]))
            accz.append(float(row[3]))
    filename2 = 'low-cost sensor B 06 Gyro.csv'
    with open(filename2, mode="r", newline="", encoding="utf8") as f:
        reader = csv.reader(f)
        header = next(reader)
        for row in reader:
            time2.append(float(row[0]))
            gyrox.append(float(row[1]))
            gyroy.append(float(row[2]))
            gyroz.append(float(row[3]))

    # 计算DAVAR
    n = 2
    windowWidth = 20
    DAVAR_accX = numpy.asarray(DAVAR(accx, n, windowWidth))
    DAVAR_accY = numpy.asarray(DAVAR(accy, n, windowWidth))
    DAVAR_accZ = numpy.asarray(DAVAR(accz, n, windowWidth))

    DAVAR_gyroX = numpy.asarray(DAVAR(gyrox, n, windowWidth))
    DAVAR_gyroY = numpy.asarray(DAVAR(gyroy, n, windowWidth))
    DAVAR_gyroZ = numpy.asarray(DAVAR(gyroz, n, windowWidth))

    sampleTime1 = GetSampleTime(time1)
    sampleNum1 = len(DAVAR_accX)
    timeAxis1 = numpy.linspace(0, sampleNum1*sampleTime1, sampleNum1, True)
    sampleTime2 = GetSampleTime(time2)
    sampleNum2 = len(DAVAR_gyroX)
    timeAxis2 = numpy.linspace(time2[0], sampleNum2*sampleTime2, sampleNum2, True)

    # 绘图
    timeStart1 = time1[0]
    timeEnd1 = time1[-1]
    yMin = -0.001
    yMax = 0.5
    xInterval = 60
    figure1, (graph_AccX, graph_AccY, graph_AccZ) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True)
    DisplayGraph(graph_AccX, 'Acceleration X', DAVAR_accX, timeAxis1, timeStart1, timeEnd1)
    DisplayGraph(graph_AccY, 'Acceleration Y', DAVAR_accY, timeAxis1, timeStart1, timeEnd1)
    DisplayGraph(graph_AccZ, 'Acceleration Z', DAVAR_accZ, timeAxis1, timeStart1, timeEnd1)
    plt.xticks(timeAxis1[::xInterval])
    #figure.tight_layout()
    plt.suptitle(filename1[0:20])
    plt.show()

    timeStart2 = time2[0]
    timeEnd2 = time2[-1]
    yMin = -0.000
    yMax = 0.06
    figure2, (graph_GyroX, graph_GyroY, graph_GyroZ) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True)
    DisplayGraph(graph_GyroX, 'Gyroscope X', DAVAR_gyroX, timeAxis2, timeStart2, timeEnd2)
```

```
    DisplayGraph(graph_GyroY, 'Gyroscope Y', DAVAR_gyroY, timeAxis2, timeStart2, timeEnd2)
    DisplayGraph(graph_GyroZ, 'Gyroscope Z', DAVAR_gyroZ, timeAxis2, timeStart2, timeEnd2)
    plt.xticks(timeAxis2[::xInterval])
    plt.suptitle(filename2[0:20])
    plt.show()
```

A.3 Kalman 滤波前后原始数据对比程序

```
import csv
import numpy as np
from matplotlib import pyplot as plt

def Allan(_list, n):
    list = np.asarray(_list)
    length = len(list)
    K = length // n
    sigma = []

    for i in range(0, length-1, n):
        s = slice(i,i+n)
        global listtemp
        listtemp = list[s]
        avg = np.mean(listtemp)
        sigma.append(avg)
    del(listtemp)
    sum = 0
    for j in range(0, K-2):
        sum += (sigma[j+1]-sigma[j]) ** 2
    delta = sum / 2 / (K - 1)
    return delta

def DAVAR(_dataList, n, windowWidth):
    AllanList = []
    length = len(_dataList)
    for index in range(windowWidth, length - windowWidth):
        datatemp = _dataList[index - windowWidth:index + windowWidth]
        Allantemp = Allan(datatemp, n)
        AllanList.append(Allantemp)
        del(datatemp)
    return AllanList

def GetSampleTime(timelist):
    sampleNum = len(timelist)
    timeInterval = [timelist[j+1] - timelist[j] for j in range(0, sampleNum-1)]
    _timeInterval = np.asarray(timeInterval)
    avgSampleTime = np.mean(_timeInterval)
    return avgSampleTime

class Kalman_Filter:
    def __init__(self, Q, R):
        self.Q = Q
        self.R = R

        self.P_k_k1 = 1
        self.Kg = 0
        self.P_k1_k1 = 1
        self.x_k_k1 = 0
        self.ADC_OLD_Value = 0
        self.Z_k = 0
        self.kalman_adc_old=0

    def Kalman(self, ADC_Value):
        self.Z_k = ADC_Value

        if ( abs(self.kalman_adc_old - ADC_Value) >= 60 ):
            self.x_k1_k1 = ADC_Value * 0.382 + self.kalman_adc_old * 0.618
        else:
            self.x_k1_k1 = self.kalman_adc_old;

        self.x_k_k1 = self.x_k1_k1
        self.P_k_k1 = self.P_k1_k1 + self.Q
        self.Kg = self.P_k_k1 / (self.P_k_k1 + self.R)
```

```

        kalman_adc = self.x_k_k1 + self.Kg * (self.Z_k - self.kalman_adc_old)
        self.P_k1_k1 = (1 - self.Kg) * self.P_k_k1
        self.P_k_k1 = self.P_k1_k1

        self.kalman_adc_old = kalman_adc
        return kalman_adc

def DisplayGraph(graph, _title, ylist, filtered_ylist):
    graph.set(title=_title, xlabel='Time(s)')
    graph.set_ylabel('DAVAR', loc='top')
    graph.axis([timeStart, timeEnd, yMin, yMax])
    l1, = graph.plot(timeAxis, ylist, label=_title)
    l2, = graph.plot(timeAxis, filtered_ylist, label='Filtered '+_title)
    graph.legend(handles=[l1,l2])
    graph.spines['top'].set_visible(False)
    graph.spines['right'].set_visible(False)
    graph.grid(linewidth=0.2)

if __name__ == '__main__':
    filename = "low-cost sensor A 04.csv"
    time = []
    accx = []
    accy = []
    accz = []
    gyrox = []
    gyroy = []
    gyroz = []

    # 读取文件
    with open(filename, mode="r", newline="", encoding="utf8") as f:
        reader = csv.reader(f)
        header = next(reader)
        for row in reader:
            time.append(float(row[0]))
            accx.append(float(row[1]))
            accy.append(float(row[2]))
            accz.append(float(row[3]))
            gyrox.append(float(row[4]))
            gyroy.append(float(row[5]))
            gyroz.append(float(row[6]))

    # Kalman Filtering
    AccX_Filter = Kalman_Filter(0.001, 0.1)
    AccY_Filter = Kalman_Filter(0.001, 0.1)
    AccZ_Filter = Kalman_Filter(0.001, 0.1)
    GyroX_Filter = Kalman_Filter(0.001, 0.1)
    GyroY_Filter = Kalman_Filter(0.001, 0.1)
    GyroZ_Filter = Kalman_Filter(0.001, 0.1)

    filteredAccX = []
    filteredAccY = []
    filteredAccZ = []
    filteredGyroX = []
    filteredGyroY = []
    filteredGyroZ = []
    for i in range (0, len(time)):
        filteredAccX.append(AccX_Filter.Kalman(accx[i]))
        filteredAccY.append(AccY_Filter.Kalman(accy[i]))
        filteredAccZ.append(AccZ_Filter.Kalman(accz[i]))
        filteredGyroX.append(GyroX_Filter.Kalman(gyrox[i]))
        filteredGyroY.append(GyroY_Filter.Kalman(gyroy[i]))
        filteredGyroZ.append(GyroZ_Filter.Kalman(gyroz[i]))

    # 计算DAVAR
    n = 2
    windowWidth = 10
    DAVAR_AccX = np.asarray(DAVAR(accx, n, windowWidth))
    DAVAR_AccY = np.asarray(DAVAR(accy, n, windowWidth))
    DAVAR_AccZ = np.asarray(DAVAR(accz, n, windowWidth))
    DAVAR_Filtered_AccX = np.asarray(DAVAR(filteredAccX, n, windowWidth))
    DAVAR_Filtered_AccY = np.asarray(DAVAR(filteredAccY, n, windowWidth))
    DAVAR_Filtered_AccZ = np.asarray(DAVAR(filteredAccZ, n, windowWidth))

    n = 2
```

```
    windowWidth = 10
    DAVAR_GyroX = np.asarray(DAVAR(gyrox, n, windowWidth))
    DAVAR_GyroY = np.asarray(DAVAR(gyroy, n, windowWidth))
    DAVAR_GyroZ = np.asarray(DAVAR(gyroz, n, windowWidth))
    DAVAR_Filtered_GyroX = np.asarray(DAVAR(filteredGyroX, n, windowWidth))
    DAVAR_Filtered_GyroY = np.asarray(DAVAR(filteredGyroY, n, windowWidth))
    DAVAR_Filtered_GyroZ = np.asarray(DAVAR(filteredGyroZ, n, windowWidth))

    sampleTime = GetSampleTime(time)
    sampleNum = len(DAVAR_AccX)
    timeAxis = np.linspace(0, sampleNum*sampleTime, sampleNum, True)

    # 绘图
    timeStart = 0
    timeEnd = time[-1]
    yMin = -0.005
    yMax = 0.16
    xInterval = 60
    figure1, (graph_AccX, graph_AccY, graph_AccZ) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True)

    DisplayGraph(graph_AccX, 'Acceleration X', DAVAR_AccX, DAVAR_Filtered_AccX)
    DisplayGraph(graph_AccY, 'Acceleration Y', DAVAR_AccY, DAVAR_Filtered_AccY)
    DisplayGraph(graph_AccZ, 'Acceleration Z', DAVAR_AccZ, DAVAR_Filtered_AccZ)
    plt.xticks(timeAxis[::xInterval])
    plt.suptitle(filename[0:-4])
    plt.show()

    yMin = -0.0001
    yMax = 0.005
    figure2, (graph_GyroX, graph_GyroY, graph_GyroZ) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True)
    DisplayGraph(graph_GyroX, 'Gyroscope X', DAVAR_GyroX, DAVAR_Filtered_GyroX)
    DisplayGraph(graph_GyroY, 'Gyroscope Y', DAVAR_GyroY, DAVAR_Filtered_GyroY)
    DisplayGraph(graph_GyroZ, 'Gyroscope Z', DAVAR_GyroZ, DAVAR_Filtered_GyroZ)
    plt.xticks(timeAxis[::xInterval])
    plt.suptitle(filename[0:-4])
    plt.show()
```

A.4 Kalman 滤波前后 Allan 方差及 DAVAR 数值对比程序

```
import numpy as np
import matplotlib.pyplot as plt
import csv

class Kalman_Filter:
    def __init__(self, Q, R):
        self.Q = Q
        self.R = R

        self.P_k_k1 = 1
        self.Kg = 0
        self.P_k1_k1 = 1
        self.x_k_k1 = 0
        self.ADC_OLD_Value = 0
        self.Z_k = 0
        self.kalman_adc_old=0

    def Kalman(self, ADC_Value):
        self.Z_k = ADC_Value

        if ( abs(self.kalman_adc_old - ADC_Value) >= 60 ):
            self.x_k1_k1 = ADC_Value * 0.382 + self.kalman_adc_old * 0.618
        else:
            self.x_k1_k1 = self.kalman_adc_old;

        self.x_k_k1 = self.x_k1_k1
        self.P_k_k1 = self.P_k1_k1 + self.Q
        self.Kg = self.P_k_k1 / (self.P_k_k1 + self.R)

        kalman_adc = self.x_k_k1 + self.Kg * (self.Z_k self.kalman_adc_old)
        self.P_k1_k1 = (1 - self.Kg) * self.P_k_k1
        self.P_k_k1 = self.P_k1_k1

```

```
            self.kalman_adc_old = kalman_adc
            return kalman_adc

def GetSampleTime(timelist):
    sampleNum = len(timelist)
    timeInterval = [timelist[j+1] - timelist[j] for j in range(0, sampleNum-1)]
    _timeInterval = np.asarray(timeInterval)
    avgSampleTime = np.mean(_timeInterval)
    return avgSampleTime

if __name__ == '__main__':
    # 读取文件
    filename = "low-cost sensor A 04.csv"
    time = []
    accx = []
    accy = []
    accz = []
    gyrox = []
    gyroy = []
    gyroz = []

    with open(filename, mode="r", newline="", encoding="utf8") as f:
        reader = csv.reader(f)
        header = next(reader)
        for row in reader:
            time.append(float(row[0]))
            accx.append(float(row[1]))
            accy.append(float(row[2]))
            accz.append(float(row[3]))
            gyrox.append(float(row[4]))
            gyroy.append(float(row[5]))
            gyroz.append(float(row[6]))

    # Kalman Filtering
    AccX_Filter = Kalman_Filter(0.001, 0.1)
    AccY_Filter = Kalman_Filter(0.001, 0.1)
    AccZ_Filter = Kalman_Filter(0.001, 0.1)
    GyroX_Filter = Kalman_Filter(0.001, 0.1)
    GyroY_Filter = Kalman_Filter(0.001, 0.1)
    GyroZ_Filter = Kalman_Filter(0.001, 0.1)

    sampleTime = GetSampleTime(time)
    sampleNum = len(time)
    timeAxis = np.linspace(0, sampleNum*sampleTime, sampleNum, True)

    filteredAccX = []
    filteredAccY = []
    filteredAccZ = []
    filteredGyroX = []
    filteredGyroY = []
    filteredGyroZ = []
    for i in range (0, sampleNum):
        filteredAccX.append(AccX_Filter.Kalman(accx[i]))
        filteredAccY.append(AccY_Filter.Kalman(accy[i]))
        filteredAccZ.append(AccZ_Filter.Kalman(accz[i]))
        filteredGyroX.append(GyroX_Filter.Kalman(gyrox[i]))
        filteredGyroY.append(GyroY_Filter.Kalman(gyroy[i]))
        filteredGyroZ.append(GyroZ_Filter.Kalman(gyroz[i]))

    # 绘图
    l1, = plt.plot(timeAxis, accx, label="Acceleration X")
    l2, = plt.plot(timeAxis, filteredAccX, label="Kalman Filtered Acceleration X")
    plt.title(filename[0:-4])
    plt.xlabel("Time(s)")
    plt.ylabel("Acceleration X(m/sš)")
    plt.legend(handles=[l1,l2])
    plt.grid()
    plt.show()

    l1, = plt.plot(timeAxis, accy, label="Acceleration Y")
    l2, = plt.plot(timeAxis, filteredAccY, label="Kalman Filtered Acceleration Y")
    plt.title(filename[0:-4])
    plt.xlabel("Time(s)")
    plt.ylabel("Acceleration Y(m/sš)")
    plt.legend(handles=[l1,l2])
```

```
    plt.grid()
    plt.show()

    l1, = plt.plot(timeAxis, accz, label="Acceleration Z")
    l2, = plt.plot(timeAxis, filteredAccZ, label="Kalman Filtered Acceleration Z")
    plt.title(filename[0:-4])
    plt.xlabel("Time(s)")
    plt.ylabel("Acceleration Z(m/sš)")
    plt.legend(handles=[l1,l2])
    plt.grid()
    plt.show()

    l1, = plt.plot(timeAxis, gyrox, label="Gyroscope X")
    l2, = plt.plot(timeAxis, filteredGyroX, label="Kalman Filtered Gyroscope X")
    plt.title(filename[0:-4])
    plt.xlabel("Time(s)")
    plt.ylabel("Gyroscope X(rad/s)")
    plt.legend(handles=[l1,l2])
    plt.grid()
    plt.show()

    l1, = plt.plot(timeAxis, gyroy, label="Gyroscope Y")
    l2, = plt.plot(timeAxis, filteredGyroY, label="Kalman Filtered Gyroscope Y")
    plt.title(filename[0:-4])
    plt.xlabel("Time(s)")
    plt.ylabel("Gyroscope Y(rad/s)")
    plt.legend(handles=[l1,l2])
    plt.grid()
    plt.show()

    l1, = plt.plot(timeAxis, gyroz, label="Gyroscope Z")
    l2, = plt.plot(timeAxis, filteredGyroZ, label="Kalman Filtered Gyroscope Z")
    plt.title(filename[0:-4])
    plt.xlabel("Time(s)")
    plt.ylabel("Gyroscope Z(rad/s)")
    plt.legend(handles=[l1,l2])
    plt.grid()
    plt.show()
```