

---

# 实时嵌入式系统应用

## ——高速飞行器实验结题报告

组长： 刘乾

组员： 王春阳

组员： 刘畅

2016 年 6 月 30 日

---

# 目录

1、框架与综述 .....	6
1.1、总架构 .....	6
1.2、PC 与分区 1 的交互 .....	7
1.3、分区 1 与分区 2 的交互 .....	7
2、组合导航单元 .....	8
2.1、单元实践信息 .....	8
2.2、单元总体设计 .....	8
2.3、惯导计算实施方案 .....	9
2.3.1、数据的存储与矩阵计算 .....	9
2.3.2、惯导计算的主要算法 .....	9
2.3.3、总算法流程 .....	11
2.4、结果分析 .....	11
3、飞行数据管理单元 .....	12
3.1、单元实践信息 .....	12
3.2、单元的总体设计 .....	12
3.3、    具体实现原理 .....	13
3.3.1、Pipe 分区间通信 .....	13
3.3.2、Socket 通信 .....	16
3.3.3、串口通信 .....	22
3.3.4、综合通信 .....	24
3.4、存在问题及后续规划 .....	26
4、飞行姿态控制单元 .....	26
4.1、单元实践信息 .....	26
4.2、飞行姿态控制单元总设计 .....	27
4.3、实施方案 .....	27
4.4、具体实现方法 .....	28
4.4.1、共享数据区分区通信 .....	28
4.4.2、分区间同步控制 .....	30
4.4.3、姿态调整 .....	32
4.4、存在问题及后续规划 .....	36

---

5、图像识别单元.....	37
5.1 环境配置 .....	37
5.2 图像识别过程.....	37
5.3 具体实现 .....	38
5.3.1 相关结构体及函数.....	38
5.3.2 程序流程 .....	40
5.4 结果筛选 .....	40
5.5 输出 .....	42
6、最终结果.....	43

---

## 图目录

图 1 架构图 .....	6
图 2 高度随时间变化仿真图 .....	12
图 3 飞行数据管理架构图 .....	13
图 4 飞行数据管理集成单元图 .....	13
图 5 pipe 分区间通信时序图 .....	14
图 6 pipe 分区间流程图 .....	14
图 7 pipe 分区间通信仿真结果 .....	16
图 8 Socket 通信流程图 .....	17
图 9 多线程实现 socket 通信 .....	21
图 10 单线程 socket 结果图 .....	21
图 11 多线程 Socket 收发动态图 .....	22
图 12 PC 机串口发送流程图 .....	22
图 13 开发板串口接收 .....	24
图 14 综合通信 .....	25
图 15 综合通信动态图 .....	26
图 16 VS 打开串口 .....	26
图 17 交互设计图 .....	27
图 18 具体实施方案图 .....	28
图 19 分区读写数据流 .....	30
图 20 分区间通信结果 .....	30
图 21 机体坐标系与姿态角 .....	32
图 22 球坐标示意图 .....	33
图 23 姿态调整部分流程图 .....	35
图 24 飞行姿态最高点 .....	36
图 25 最终落点 .....	36
图 26 图像识别三步 .....	37
图 27 图像识别程序流程 .....	40
图 28 结果偏差 .....	41
图 29 FAST SIFT FlannBased 对应结果 .....	42
图 30 SURF SIFT FlannBased 对应结果 .....	42
图 31 最终动态图展示 .....	43

## 表目录

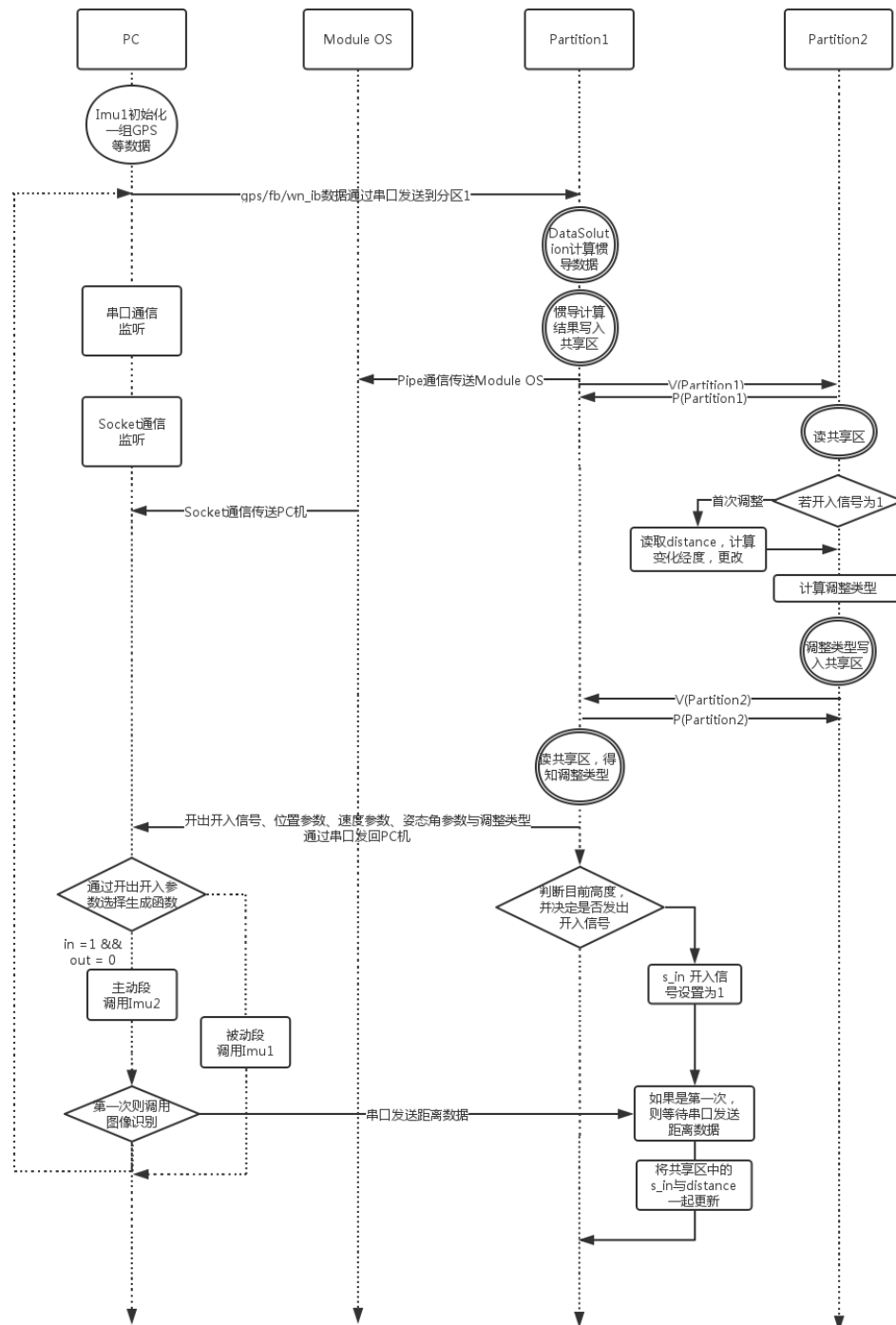
表 1 图像识别 PC 与分区 1 的串口时序 .....	7
表 2 非图像识别 PC 与分区 1 的串口时序 .....	7
表 3 分区 1 与分区 2 的交互时序 .....	8

---

# 1、框架与综述

## 1.1、总架构

图 1 架构图



整个系统的交互关系与工作流程如 图 1 架构图 所示，主要的交互过程有：

- 1. PC 与分区 1 的交互
- 2. 分区 1 与分区 2 的交互

下面来一一说明一下每个部分的交互过程。

1.2、PC 与分区 1 的交互

PC 与分区 1 的交互分为两种情况：

- 1、第一次产生开入信号需要发送图像识别结果时，PC 与分区 1 的串口收发过程如 表 1 图像识别 PC 与分区 1 的串口时序 所示。

PC	分区 1
串口发送 gps 等 9 个浮点数	
	串口接收，计算得到调整类型
	串口发送开入开出信号等
串口接收开入开出信号	
串口发送图像识别结果	
	串口接收，动态更改目标点经纬度

表 1 图像识别 PC 与分区 1 的串口时序

- 2、当不需要发送目标识别的结果时，PC 与分区 1 的串口收发过程如 表 2 非图像识别 PC 与分区 1 的串口时序 所示。

PC	分区 1
串口发送 gps 等 9 个浮点数	
	串口接收，计算得到调整类型
	串口发送开入开出信号等
串口接收开入开出信号	

表 2 非图像识别 PC 与分区 1 的串口时序

1.3、分区 1 与分区 2 的交互

分区 1 与分区 2 的交互在后期实验中是最重要的交互过程,交互过程也是有两种过程,用表来表示即是

分区 1	分区 2
DataSolution 计算惯导数据	
写入共享区	
V(Partition1)	
	P(Partition1)
	读共享区
	判断开出信号，计算调整类型
	写共享区

	V(Partition2)
P(Partition2)	
读共享区	
判断是否发出开出信号	

表 3 分区 1 与分区 2 的交互时序

## 2、组合导航单元

### 2.1、单元实践信息

- 小组成员组成、角色及任务分工：

刘乾：组长，写了一个完整的使用 malloc 的惯性导航程序，文档编写，集成调试。

王春阳：组员，学习龙格库塔方法与信号量的同步控制，文档编写。

刘畅：组员，写了完整的不带 malloc 的惯导程序。

- 实施时间：

2016.4

- 单元任务简介及单元总体完成情况：

本单元主要通过编写惯导计算程序完成导弹在惯性导航阶段的模拟，并在虚拟机上进行模拟。通过与 GPS 加权，得到结果基本满足要求。

### 2.2、单元总体设计

本单元的主要目的是通过给定的惯导计算方法编写相应的计算程序。通过惯导加速度计提供的比力  $f_b$  及陀螺仪提供的  $\omega_b^n$  计算出各个方向上相应的加速度  $\dot{V}$ ，之后设计投票系统，先利用一次积分法、多次积分法、四阶龙格库塔法求解得到速度，再通过投票系统选出最有方案，并对速度按照最优方案进行更新。

有关姿态的微分方程通过四元数微分方程求解。需要注意的是，采用四元数法除第一次利用初始值外，后面迭代均是在前一次循环的基础上进行更新，因此应注意数据的保存。



最后在  $\Delta t$  相当短的时间间隔内，近似认为导弹匀速运动。利用速度  $V$  的数据，更新得到位置信息。但是由于重力加速度的变化，惯导计算的数据存在较大误差，因而最终结果与 GPS 加权进行计算，其中 GPS 的权值应当较高。

## 2.3、惯导计算实施方案

### 2.3.1、数据的存储与矩阵计算

```
typedef double(*Matrix)[3];
typedef double(*Column)[1];
```

### 2.3.2、惯导计算的主要算法

1. 数据初始化与变量声明
2. 计算重力加速度

根据物理学知识可知，重力加速度与高度有关  $g_k = g_0 \left(1 - \frac{2h}{R_e}\right)$

3. 计算  $R_M$  和  $R_N$

$$R_M = R_e (1 - 2e + 3e \sin^2 L)$$

$$R_N = R_e (1 + e \sin^2 L)$$

4. 计算  $\omega_{ie}$  和  $\omega_{en}$

$$\omega_{ie}^n = \begin{bmatrix} 0 \\ \omega_{ie} \cos L \\ \omega_{ie} \sin L \end{bmatrix} = \begin{bmatrix} 0 \\ \omega_{iey} \\ \omega_{iez} \end{bmatrix}$$

$$\omega_{en}^n = \begin{bmatrix} -V_y^n / (R_M + h) \\ -V_x^n / (R_M + h) \\ -V_x^n \tan L / (R_M + h) \end{bmatrix} = \begin{bmatrix} \omega_{enx}^n \\ \omega_{eny}^n \\ \omega_{enz}^n \end{bmatrix}$$

5. 四元数更新

$$\Lambda(k+1) = \left\{ \left[ 1 - \frac{\Delta\theta_0^2}{8} - \frac{\Delta\theta_0^4}{384} \right] \cdot I + \left[ \frac{1}{2} - \frac{\Delta\theta_0^4}{48} \right] \cdot \Delta\theta \right\} \cdot \Lambda(k)$$

$$\Delta\theta = \begin{bmatrix} 0 & -\Delta\theta_x & -\Delta\theta_y & -\Delta\theta_z \\ \Delta\theta_x & 0 & \Delta\theta_z & -\Delta\theta_y \\ \Delta\theta_y & -\Delta\theta_z & 0 & \Delta\theta_x \\ \Delta\theta_z & \Delta\theta_y & -\Delta\theta_x & 0 \end{bmatrix}$$

$$\Delta\theta_0 = \sqrt{\Delta\theta_x^2 + \Delta\theta_y^2 + \Delta\theta_z^2}, \quad \Delta\theta_x = \omega_{nbx}^b \cdot \Delta t, \quad \Delta\theta_y = \omega_{nby}^b \cdot \Delta t, \quad \Delta\theta_z = \omega_{nbz}^b \cdot \Delta t$$

## 6. 计算姿态矩阵

$$C_b^n = \begin{bmatrix} \lambda_0^2 + \lambda_1^2 - \lambda_2^2 - \lambda_3^2 & 2(\lambda_1\lambda_2 - \lambda_0\lambda_3) & 2(\lambda_1\lambda_3 + \lambda_0\lambda_2) \\ 2(\lambda_1\lambda_2 + \lambda_0\lambda_3) & \lambda_0^2 - \lambda_1^2 + \lambda_2^2 - \lambda_3^2 & 2(\lambda_2\lambda_3 - \lambda_0\lambda_1) \\ 2(\lambda_1\lambda_3 - \lambda_0\lambda_2) & 2(\lambda_2\lambda_3 + \lambda_0\lambda_1) & \lambda_0^2 - \lambda_1^2 - \lambda_2^2 + \lambda_3^2 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix}$$

## 7. 计算姿态角加速度

$$\omega_{nb,k}^i = \omega_{ib,k}^n - (C_{b,k}^n)^T \cdot (\omega_{en,k}^n + \omega_{ie,k}^n)$$

$$\omega_{in,k}^n = \omega_{en,k}^n + \omega_{ie,k}^n$$

$$\omega_{nb}[0] = \omega_{ib}[0] - T[0][0]\omega_{in}[0] - T[1][0]\omega_{in}[1] - T[2][0]\omega_{in}[2]$$

## 8. 计算姿态角

$$\psi = \arctan\left(-\frac{T[1][2]}{T[2][2]}\right)$$

$$\theta = \arcsin(T[3][2])$$

$$\gamma = \arctan\left(-\frac{T[3][1]}{T[3][3]}\right)$$

## 9. 计算 $f_n$

$$f_n = C_b^n \cdot f_b$$

## 10. 计算各个方向加速度

$$\begin{bmatrix} \dot{V}_x^n \\ \dot{V}_y^n \\ \dot{V}_z^n \end{bmatrix} = \begin{bmatrix} f_{nx} \\ f_{ny} \\ f_{nz} \end{bmatrix} - \begin{bmatrix} 0 & -2(\omega_{iez}^n + \omega_{enz}^n) & 2\omega_{iey}^n + \omega_{eny}^n \\ 2\omega_{iez}^n + \omega_{enz}^n & 0 & -2(\omega_{iey}^n + \omega_{eny}^n) \\ -2(\omega_{iez}^n + \omega_{enz}^n) & 2\omega_{iey}^n + \omega_{eny}^n & 0 \end{bmatrix} \cdot \begin{bmatrix} V_x^n \\ V_y^n \\ V_z^n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -g_k \end{bmatrix}$$

## 11. 积分求速度

### 1) 一次积分法

$$\Delta V_x = \dot{V}_x \Delta t$$

$$\Delta V_y = \dot{V}_y \Delta t$$

$$\Delta V_z = \dot{V}_z \Delta t$$

### 2) 四阶龙格库塔法

设微分方程为  $\dot{V} = f(V)$ ，用龙格库塔法求解方法如下：

$$k_1 = f(V_0) \quad V_1 = k_1 \Delta t / 2 + V_0$$

$$k_2 = f(V_1) \quad V_2 = k_2 \Delta t / 2 + V_1$$

$$k_3 = f(V_2) \quad V_3 = k_3 \Delta t / 2 + V_2$$

$$k_4 = f(V_3) \quad V_4 = k_4 \Delta t / 2 + V_3$$

$$V_{new} = V_0 + \frac{1}{6}(k_1 + 2k_2 + 3k_3 + k_4) \Delta t$$

对  $V_x$ ， $V_y$ ， $V_z$  分别执行上述操作，即可得到更新后的速度。

## 12. 求纬度、经度和高度增量

$$\dot{L} = \frac{V_y^n}{R_M + h} \quad \dot{\lambda} = \frac{V_x^n}{R_N + h} \quad \dot{h} = V_z^n$$

### 2.3.3、总算法流程

在分区一中定义函数 Datasolution，其函数声明如下：

```
double* DataSolution(double wn_ib[3], double fb[3], double gps[3],  
double Lambda[4]);
```

其中，wn\_ib, fb, gps 分别为传感器给出的数据，Lambda 为四元数的值，用于每次更新使用。

在计算分区中新建任务，在任务中循环调用 DataSolution，直至导航结束。

## 2.4、结果分析

将高度-时间的曲线绘出，如 图 2 高度随时间变化仿真图 所示

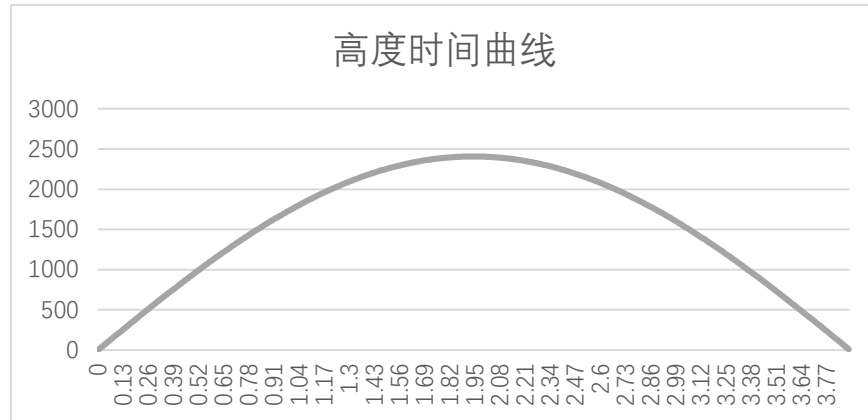


图 2 高度随时间变化仿真图

高度曲线变化符合抛物线，与预期一致。

## 3、飞行数据管理单元

### 3.1、单元实践信息

- 小组成员组成、角色及任务分工：  
刘乾：组长，串口通信资料查阅及代码、文档编写、集成与调试  
王春阳：组员，socket 通信资料查阅及代码、文档编写  
刘畅：组员，分区间通信资料查阅及代码、文档编写
- 实施时间：  
2016 年 5 月
- 单元任务简介及单元总体完成情况：  
完成预期目标，实现对应功能。

### 3.2、单元的总体设计

本单元要实现将串口通信、分区间通信与 Socket 网络通信一起集成起来的飞行数据管理模块，最终实现 PC 机与开发板之间的数据交互功能来模拟实际的弹道导弹航行时的数据通讯。

从 PC 机读取 GPS 等数据，将其通过串口 COM2 以字符串的形式发送给开发板。开发板中的 Partition1 分区启动一个 task，该 task 将不断监听串口 COM2，当收集到一组完整数据时，即写入在 Module OS 中建立的管道 pipe。Module OS 中启动一个 task，该 task 对管道进行读取，当没有可读取的内容时自动阻塞。当 Partition1 分区发送数据到 Module OS 后，Module OS 使用惯导模块对这些数据进行计算，并将本轮次计算得出的高度、纬度和经度等

数据通过网络通信 socket 套接字发送给 PC 机。至此，飞行数据管理模块的任务完成。

架构图如 图 3 飞行数据管理架构图 所示。

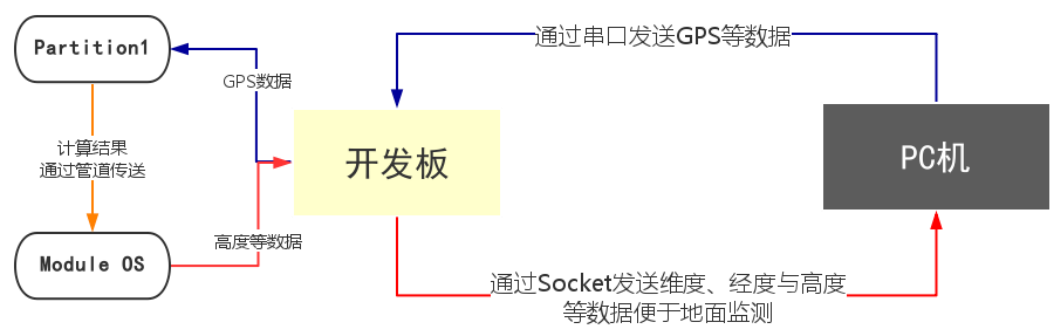


图 3 飞行数据管理架构图

3.3、具体实现原理

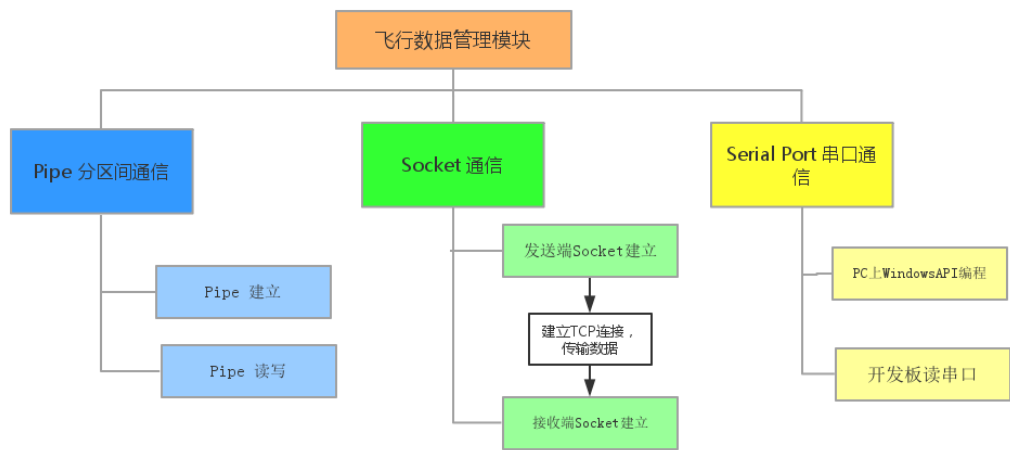


图 4 飞行数据管理集成单元图

3.3.1、Pipe 分区间通信

在 VxWorks 中，管道是一种通过虚拟的 I/O 设备来实现的消息队列通信机制。使用函数 pipeDevCreate()和 pipeDevDelete()来生成和删除管道。它的相关 API 跟统一 I/O 访问接口完全一致，如 open，close，read，write 等。

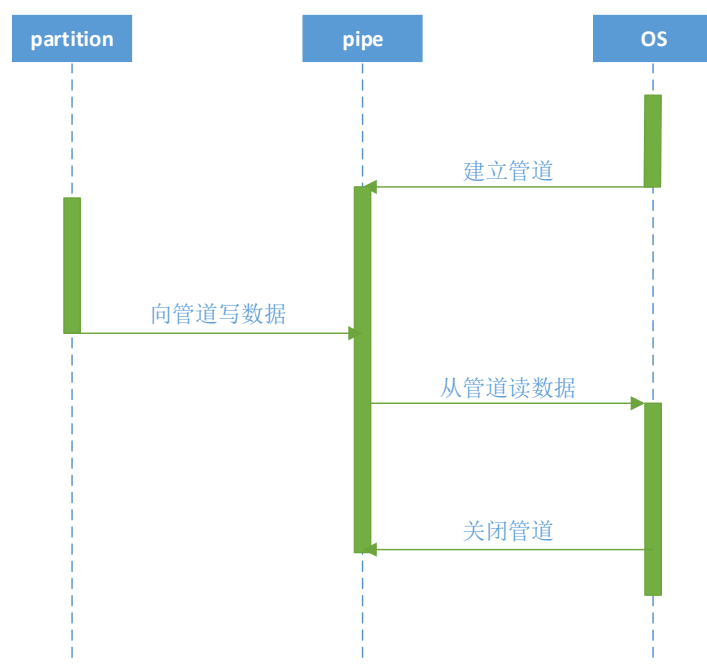


图 5 pipe 分区间通信时序图

如 图 5 pipe 分区间通信时序图 所示，首先在 ModuleOS 里面利用 pipeDevCreate() 建立一个名字为 pipe\_name 的管道。在 partition1 以只写 WRONLY 的方式打开 pipe\_name 管道，并将计算得到的组合导航信息 pipe\_buffer 写入管道中。在 ModuleOS 中新建一个 task，在 task 中以只读 RDONLY 方式打开 pipe\_name 管道。利用 read 对管道进行监听，read 自带阻塞功能，因此不必再使用 select() 函数进行阻塞。一旦串口准备好，就将管道中的数据读入 buffer 中。OS 中的读操作一定要写到 task 里面，否则 read 的阻塞功能会使 OS 挂起，partition 无法执行，程序崩溃。

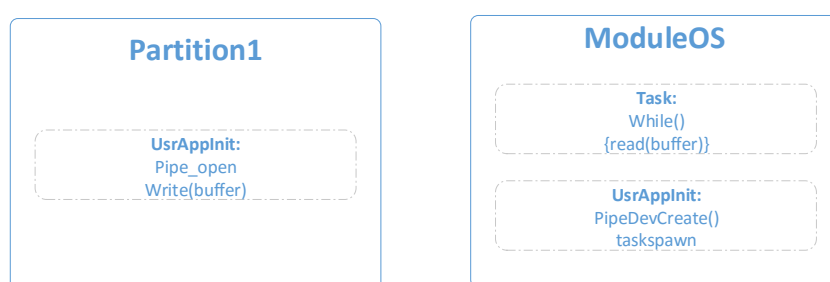


图 6 pipe 分区间流程图

## pipeDevCreate 函数

```
STATUS pipeDevCreate(char* name,int nMessages,int nBytes)
```

<b>name</b>	管道名
<b>nMessages</b>	管道中的最大数目
<b>nBytes</b>	每个消息的字节数

open 函数

int open(const char *path, int access,int mode)		
<b>path</b>	管道名称	
<b>access</b>	访问模式	
<b>nBytes</b>	每个消息的字节数	
<b>O_RDONLY</b>	1	只读打开
<b>O_WRONLY</b>	2	只写打开
<b>O_RDWR</b>	4	读写打开

write 函数

int write(int fd, char * buffer, size_t nbytes )	
<b>fd</b>	open 得到的 pipeld
<b>buffer</b>	要写入的信息
<b>nbytes</b>	sizeof(buffer)

read 函数

int read(int fd, char * buffer, size_t maxbytes )	
<b>fd</b>	open 得到的 pipeld
<b>buffer</b>	读入缓冲区
<b>maxbytes</b>	buffer 最大长度

在虚拟机上进行仿真，将惯导计算得到的数据由 partition1 传到 ModuleOS 中，并逐行打印到屏幕上，结果如 图 7 pipe 分区间通信仿真结果

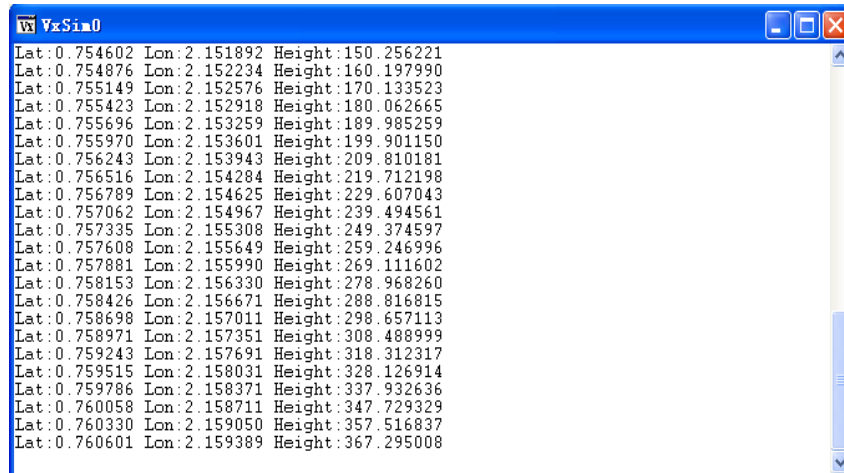


图 7 pipe 分区间通信仿真结果

### 3.3.2、Socket 通信

通信流程如 图 8 Socket 通信流程图 所示



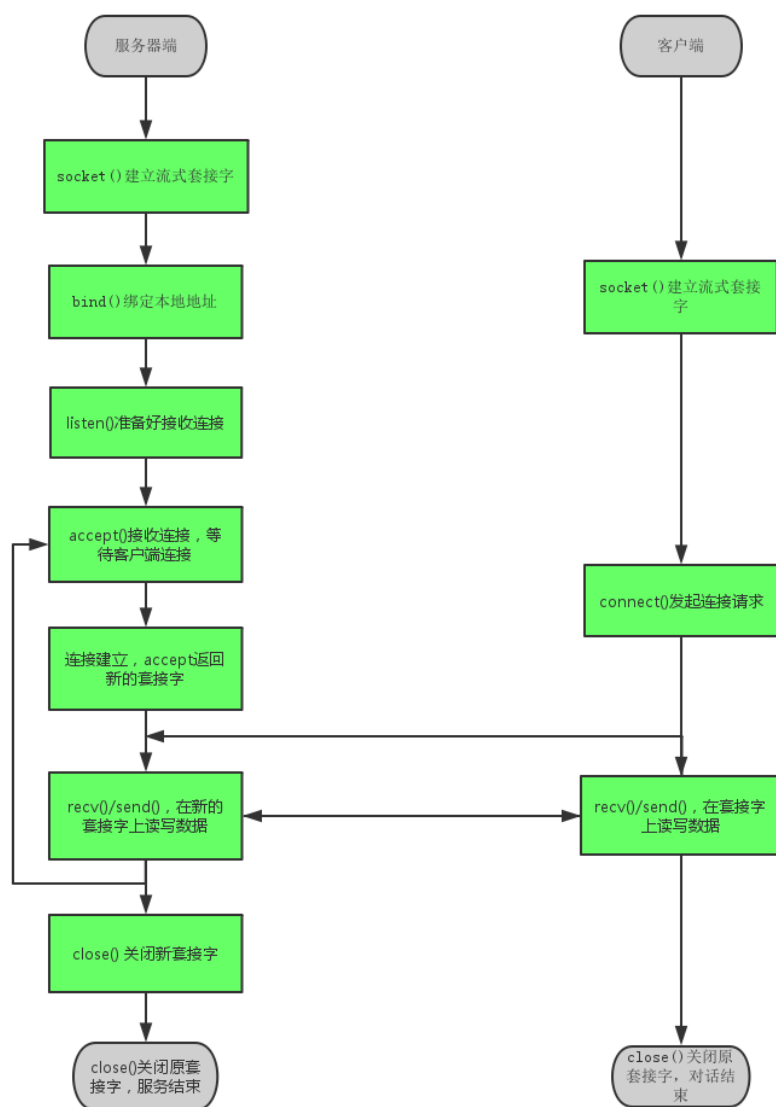


图 8 Socket 通信流程图

### 通信过程概述

PC 机作为接收端（服务器端），通过 socket 建立套接字，再用 bind 绑定本地地址（此处先设置为任意），调用 Listen 开始监听端口，再通过 accept 等待开发板发送端（客户端）进行连接；检测到发送端有链接后（发送端申请套接字，设置负责接收 sockaddr\_in 结构体中的地址成员为 PC 机 IP 地址，并进行 connect 后），返回新的读取数据的套接字，调用 recv 进行读取，若对方没有发送则会阻塞等待。而开发板则调用 send 进行发送。当 recv 检测到对面数据发送完毕后（返回值变为 0），关闭读取套接字，回到 accept 状态，继续等待 connect 进行连接和下一次发送。

主要结构体

```
struct sockaddr_in {
    short    sin_family;
    u_short  sin_port;
    struct   in_addr sin_addr;
    char     sin_zero[8];
};
```

结构体参数	代表含义
sin_family	指代协议族，在 socket 编程中只能是 AF_INET
sin_port	存储端口号（使用网络字节顺序），在 linux 下，端口号的范围 0~65535,同时 0~1024 范围的端口号已经被系统使用或保留。
sin_addr	存储 IP 地址，使用 in_addr 这个数据结构
sin_zero	为了让 sockaddr 与 sockaddr_in 两个数据结构保持大小相同而保留的空字节

主要函数

Socket 相关函数在 Vxworks 与 windows 中几乎是一致的。所以这里统一介绍。

函数名称	函数定义	函数作用
bzero	void bzero((char*)& myaddr,sizeof(myaddr))	初始化套接字结构体
htonl	long htonl(unsigned long hostlong)	函数将一个 32 位的无符号整数从主机字节顺序转换成网络字节顺序。返回网络字节顺序的 32 位无符号整数。
htons	unsigned short htons(unsigned short hostshort)	函数将一个 16 位的无符号整数从主机字节顺序转换成网络字节顺序。返回网络字节顺序的 16 位无符

		号整数。
<i>socket</i>	int socket(int domain,int type,int protocol)	函数建立一个套接字，给指定的地址族，数据类型和协议分配一个套接字描述符和相关的资源。返回 socket 描述符或 ERROR。
<i>bind</i>	STATUS bind (int s,struct sockaddr * name,int namelen)	函数将本地地址与 socket 函数创建的未命名的套接字绑定，建立起套接字的本地连接。
<i>listen</i>	STATUS listen(int s,int backlog)	函数用于服务器端套接字的侦听连接。第二个参数 backlog 指定了在此套接字上排队等待处理的连接请求的最大个数。
<i>connect</i>	STATUS connect (int s , struct sockaddr * name ,int namelen)	函数用来与对方建立一个连接。name 指向一个含有对方套接字地址的结构。
<i>accept</i>	int accept ( int s , struct sockaddr * addr , int * addrlen)	函数 accept 从已完成的连接队列头中获得一个已完成的连接。这个连接队列是调用 listen 建立的。
<i>recv</i>	int recv(int socket,void *buf,size_t len,int flag)	接收与发送函数，buf 为缓存区首地址，flag 为通信参数，一般设为 0
<i>send</i>	int send(int socket,const void*buf,size_t len,int flags)	

## 具体实现

Windows 与 Vxworks 均采用 C 来实现 socket 通信。具体函数调用关系同通信流程一节中的图示。

---

## 初期实现策略

在 moduleOS 中，分区接收数据与 socket 发送数据顺序执行

```
While (条件) {  
    read ( pipeId , pipe_buffer , 50); //分区通信  
    tNetSend() ; //向 PC 发送数据  
}
```

存在的问题：丢包

原因

① 初期 socket 通信的接收与发送均采用单线程，会因为 TCP 建立连接以及传输数据的速率问题发生数据丢包现象。即上次网络传输还未完成，依然占据缓冲区，导致分区通信新传输来的数据丢失。

② 在 PC 端接收程序中有一些用来显示传输状态的输出信息，即 printf. 而它很可能会阻塞 IO，影响网络传输效果。

解决

接收端、发送端均采用多线程机制。

发送端：每次 send 均建立一个任务。

```
taskSpawn( " " ,200, 0x100, 20000, (FUNCPTR)tNetSend);
```

接收端：每次检测到发送端有 connect 链接，均新建线程去接收。

```
CreateThread(NULL, 0, ProcessClientRequests, &clientsocket, 0, NULL);
```

其中每个 ProcessClientRequests 均是一个独立的 socket 接收数据函数。Clientsocket 即为代表开发板的套接字。

实现思想如 图 9 多线程实现 socket 通信

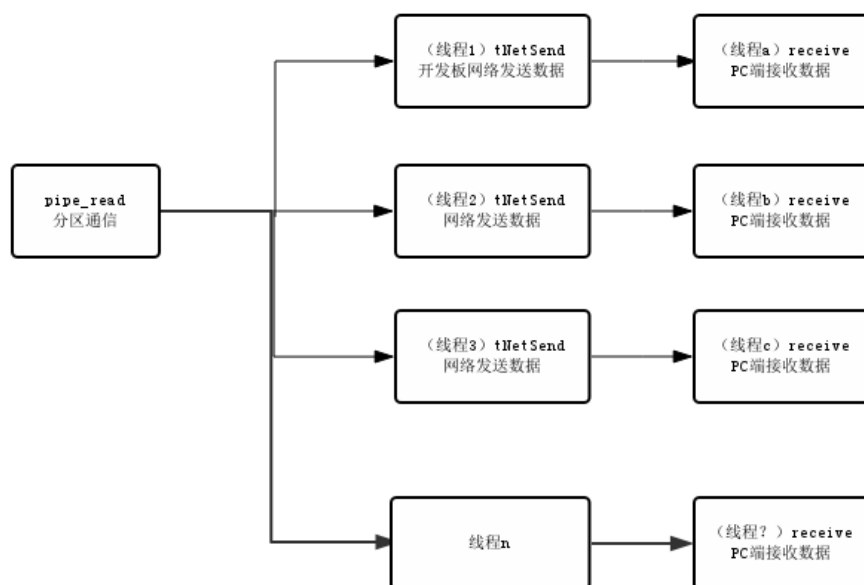


图 9 多线程实现 socket 通信

每次分区通信接收到数据，便新建任务进行 socket 发送；而接收端也每次建立线程去对应接收。尽可能实现每次传输互不干扰，避免丢包现象的产生。

### 实验结果截图

采用单线程时：

```

***Client***    Lat:0.757789 Lon:2.155830 Height:-216.056819
***SYS***      New client touched.
***SYS***      HELLO.
***SYS***      New client touched.
***SYS***      HELLO.
***SYS***      New client touched.
***SYS***      HELLO.
***Client***    Lat:0.756971 Lon:2.154807 Height:-245.726102
  
```

图 10 单线程 socket 结果图

改用多线程后：

数据传输非常稳定，几乎没有丢包现象。（左边为串口发送程序，右边为 PC 端接收程序）。见最终结果展示 图 11 多线程 Socket 收发动态图。

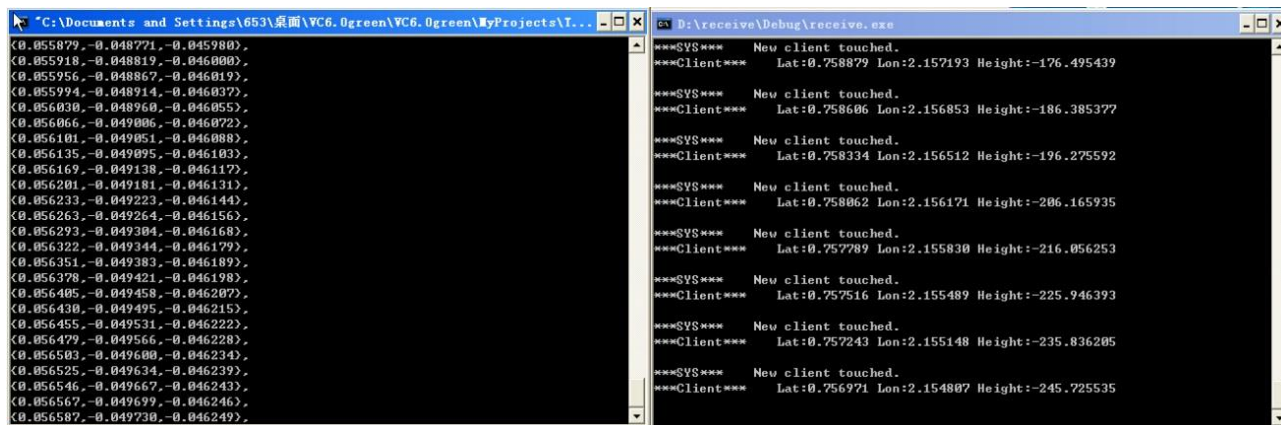


图 11 多线程 Socket 收发动态图

### 3.3.3、串口通信

串口通信这里分为两部分实现，一部分是 PC 主机上的串口发送程序的实现，另一部分是开发板上的串口接收程序。

#### PC 主机串口发送程序

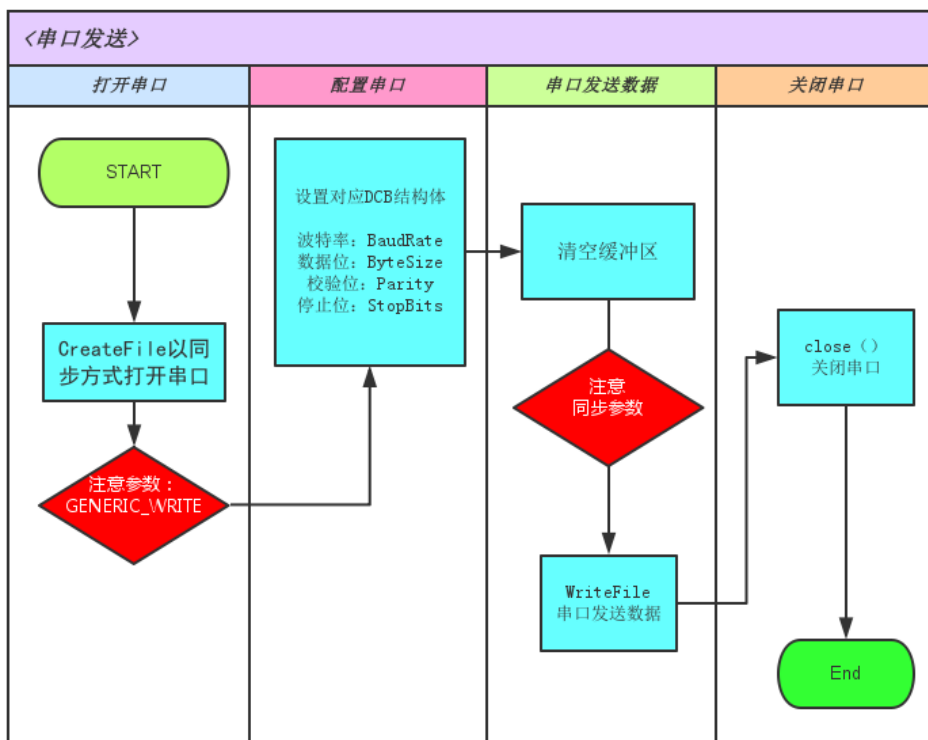


图 12 PC 机串口发送流程图

下面结合代码来具体讲解一下类似过程：

函数名称	函数作用
<b>CreateFile("COM2",           GENERIC_WRITE,           0,           NULL,           OPEN_EXISTING,           0,           NULL);</b>	CreateFile 函数打开了 COM2 串口，在 PC 机上打开的串口是用于发送的。这里需要注意的是如果为同步发送模式，倒数第 2 个参数需要为 0，而不是重叠模式。
<b>SetupComm(hCom, 1024, 1024);</b>	设置缓冲区的大小为 1024。
<b>GetCommState(hCom, &amp;com_dcb);</b>	获取 hCom 的 DCB 结构体指针，用来设置其对应的结构体
<b>com_dcb.BaudRate = 115200;</b>	设置串口发送的波特率为 115200
<b>com_dcb.ByteSize = 8;</b>	数据位为 8 位
<b>com_dcb.Parity = NOPARITY;</b>	没有数据校验位
<b>com_dcb.StopBits = TWOSTOPBITS;</b>	停止位是 2 位
<b>SetCommState(hCom, &amp;com_dcb);</b>	将 DCB 结构体写回对应的 hCom 串口句柄。
<b>PurgeComm(hCom,           PURGE_TXCLEAR             PURGE_RXCLEAR           );</b>	清空 hCom 的缓冲区准备发送。
<b>memset(buffer,0,strlen(buffer));</b>	清空 buffer 缓冲区（不是串口缓冲区）
<b>fgets(buffer,100,file_wn);</b>	从文件中读取对应的一行内容到 buffer 中。
<b>bWriteStat = WriteFile(hCom,           buffer,           dwBytesWritten,           &amp;dwBytesWritten,           NULL);</b>	向 hCom 代表的串口句柄中写入 buffer 数组的内容值。

开发板上的串口接收程序

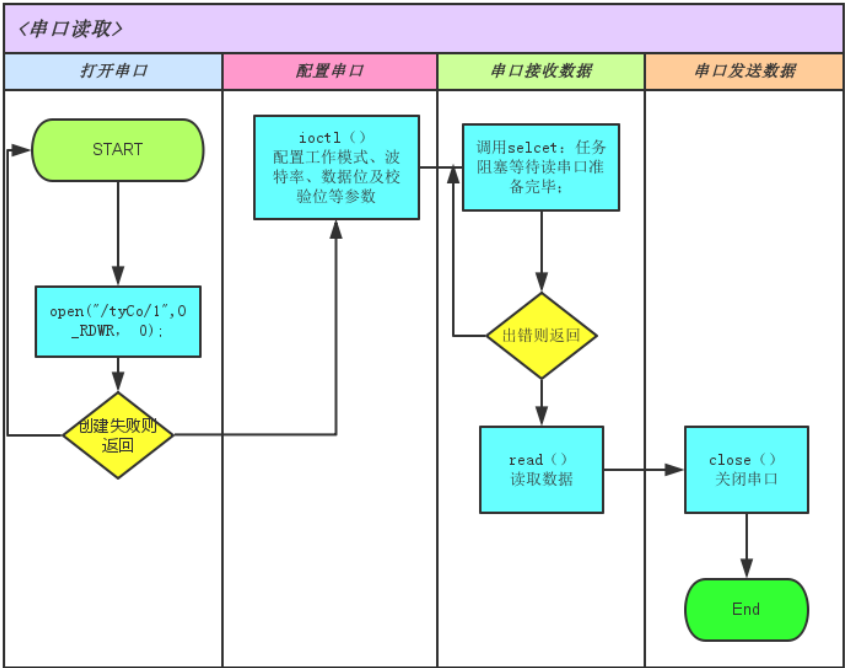


图 13 开发板串口接收

函数	函数作用
com_fd=open("/tyCo/1",O_RDWR,0);	在开发板上打开串口 1，模式是 O_RDWR。串口设备与文件系统的打开方式类似，在其前加前缀/tyCo 即可。
ioctl(com_fd,FIOBAUDRATE,115200);	设置 com_fd 的波特率为 115200
ioctl(com_fd,SIO_HW_OPTS_SET,CS8 STOPB PARENB PARODD);	设置 com_fd 为奇校验，2 位停止位并且 8 位数据位。
FD_ZERO(&fds_data);	清空缓冲区，准备开始读取串口数据。
FD_SET(com_fd,&fds_data);	
read(com_fd,accept_buf,1);	读取串口数据。

3.3.4、综合通信

上面这三部分的结合在综合设计中虽然提到了，但是没有提到具体的实现方式。代码的最终框架大致如图 14 综合通信



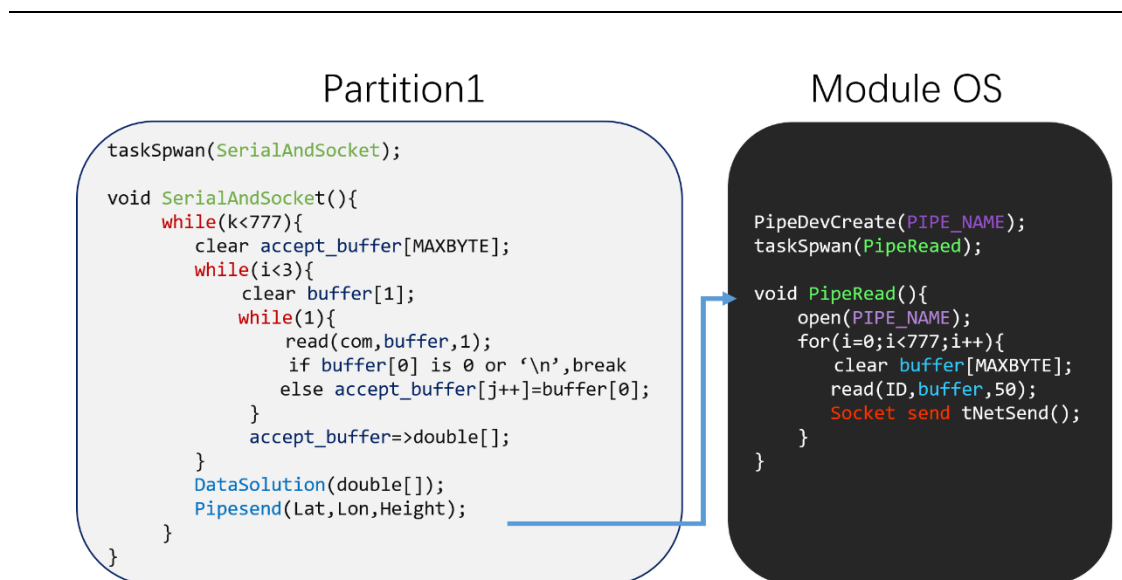


图 14 综合通信

## 执行流程

在 Partition1 中，启动一个 task 用来接收串口的数据，并使用惯导计算得出数据后，再通过分区间通信将该数据发给 Module OS。

在 Moudle OS 中，启动一个 task 用来读取管道中的数据，并将数据通过非阻塞即异步的方式使用 socket 通信与 PC 机进行通信。

## 数据传输过程

- PC 机的串口发送程序从预置的三个文本文件中分别读取了 gps, wn\_ib 与 fb 的数据。之后 PC 机对串口进行了一系列参数的设置，并打开了串口 COM2，准备发送数据。
- 现在开始清除缓冲区，从文件中读取一行的内容到缓冲区，并将缓冲区的内容写入串口。如此进行三次即完成一组数据的传输，此时串口发送程序休眠 0.05s。
- 开发板 Partition1 分区中的串口接收程序接收到了 PC 发送来的数据，每次只接受单个字符，直到接收到换行符或者空白符，即表示一行接受完毕。当一行接受完毕时，将根据 i 值将本行的内容转移到对应的 double 数组中。
- 当三行一起接受完毕后，gps,wn\_ib,fb 的数据已经准备好了，此时调用 DataSolution 进行惯导计算，得出惯导计算的结果：高度，经度与纬度。将它们写入管道，发送给 Module OS 分区。
- 在 Module OS 分区中有一个 task 一直在监听管道，现在监听到管道有新的数据写入，即读出数据，开一个新的 task 将其通过网络发送给 PC 机。
- PC 机上有一个网络监控的程序在运行，此时收到了来自 Module OS 分区发送的网络包，随即将获得的高度、经度与纬度打印。

综合效果展示如 图 15 综合通信动态图

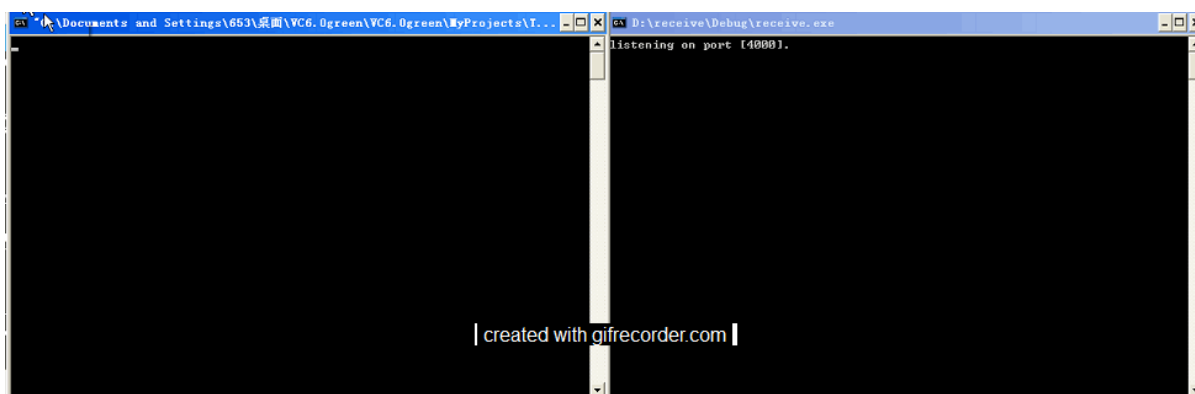


图 15 综合通信动态图

### 3.4、存在问题及后续规划

在本单元中我们发现 VS 里无法打开串口,但后来通过查阅资料解决了这个问题,在 VS 中和 VC6.0 中定义稍有不同,VS 中打开串口的方式如 图 16 VS 打开串口

```
const wchar_t device[5] = L"COM2";

hCom = CreateFile(
    device,
    GENERIC_WRITE | GENERIC_READ,
    0,
    0,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    NULL
);
```

图 16 VS 打开串口

## 4、飞行姿态控制单元

### 4.1、单元实践信息

- 小组成员组成、角色及任务分工：
  - 刘乾：组长，姿态调整部分与串口收发资料查阅及代码、文档编写、最终集成与调试。
  - 王春阳：组员，共享数据区、同步控制资料查阅及代码、文档编写。
  - 刘畅：组员，姿态调整部分及代码、文档编写。
- 实施时间：
  - 2016 年 5 月 23 日-6 月 3 日
- 单元任务简介及单元总体完成情况：

本单元主要涉及飞行姿态控制单元，到目前为止已经完，但结果误差较大。

## 4.2、飞行姿态控制单元总设计

如 图 17 交互设计图 所示，在本单元的总设计中，几个主要交换的数据是：

- [1] 导航分区与控制分区间的导航数据
- [2] 导航分区和控制分区间的调整类型数据
- [3] 导航分区和 PC 端的组合导航数据
- [4] 导航分区和 PC 端的姿态角、调整类型数据

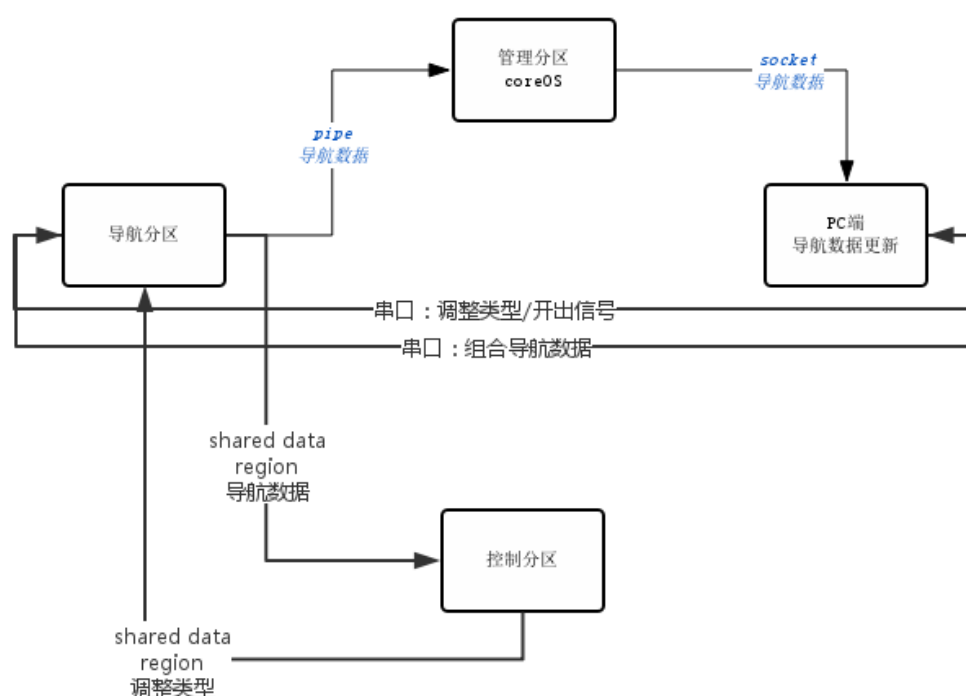


图 17 交互设计图

我们最终的设计即根据 图 17 交互设计图 定义一些数据结构，添加了各种数据交互的过程，使得开发板的分区间可以同步控制读写，PC 与开发板的串口收发也可以同步控制，姿态调整控制使得导弹的轨迹达到预期要求即可完成本单元的目标。

## 4.3、实施方案

我们的实施方案如 图 18 具体实施方案图 所示。此图表示了整个过程中的时序关系，指定了以下几种严格的先后关系：

1. 最初的是串口由 PC 驱动，之后进入循环，由开发板向串口反馈结果来驱动 PC 机发送串口。这样由开发板来驱动串口发送速率，不会使得开发板崩溃。
2. 首先由分区 1 调用 DataSolution 方法得到惯导数据后并写入共享区，分区 2 才可以对共享区先读后写。



## 配置共享数据区

在 default.xml 里对共享数据区进行配置

```
<SharedDataRegions>
  <SharedData Name="sdRgn">
    <SharedDataDescription
      SystemAccess="READ_WRITE"
      DataType="DATABASE"
      CachePolicy="DEFAULT"
      size="0x1000">
    </SharedDataDescription
  </SharedData>
</SharedDataRegions>
```

并且之后要在各分区中声明该分区

```
<SharedDataRegion NameRef="sdRgn" UserAccess="READ_WRITE">
```

数据区的结构由用户程序自行定义，可利用结构体

```
typedef struct test_data
{
    int data1;
    int data2;
} TEST_DATA;
```

这样就将共享数据区定义成为了一个存储两个 int 的内存区域。

## 获取共享区地址

函数名称	参数	物理意义	调用说明
sdRgnAddrGet	sdName	定义的共享数据区名称	每次进行读写之前,先调用该函数获得共享数据区地址,再对该地址表示的结构体的各成员进行赋值或读取。

一个简单的函数调用样例如下：

```
pTest = (TEST_DATA *) sdRgnAddr;
pTest->data1 = data1;
pTest->data2 = data2;
```

如 图 19 分区读写数据流，对于这一模块而言，两个分区分别读写的内容是不同的

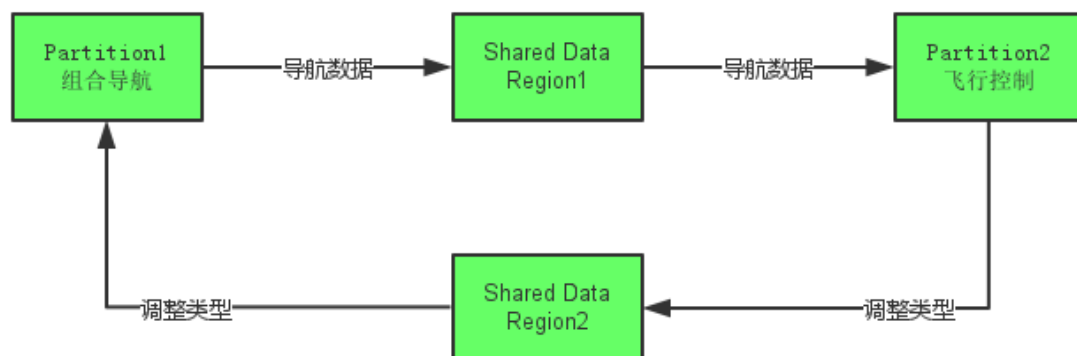


图 19 分区读写数据流

## 结果截图

```
Starting at 0x100000...
Attached TCP/IP interface to mottsec unit 1
Attaching interface lo0...done

VxWorks 653
Copyright (c) 1984-2010 WindRiver Systems, Inc.

CPU: Wind River SBC8349E
Runtime Name: VxWorks 653
Runtime Version: 2.3
BSP version: 2.0/5
Created: May 23 2016, 10:48:23
WDB: Ready.

Hello,Kugou
pipe create success!
data in sdRgn : 1 2 test
```

图 20 分区间通信结果

## 4.4.2、分区间同步控制

本次实验中，我们小组原本打算使用信号量来实现同步控制，实现方法是将信号量要放入共享数据区，供两个分区同时访问。共享数据区包含信号量以及两个分区之间互相通信的数据。

### 通过 CoreOS 共享信号量

目标：采用二进制信号量（semLib）。在 ModlueOS 中对信号量和传输的数据进行初始

---

化，之后切换到两个分区进行调度执行。

**问题**：coreOS 中的共享数据区操作与分区所用的库以及实现方式都不同。CoreOS 需要在代码中手工创建共享数据区，但这个数据区的地址分区通过之前的 `sdRgnAddrGet()` 无法知晓，仍需要别的渠道将地址传输给分区。

## 信号量存放共享区

**目标**：由于信号量不可通过 CoreOS 共享，所以我们希望通过共享数据传递信号量实现两个分区均可使用信号量。因为一开始先调度分区 1，所以直接在分区 1 中进行初始化。同时为了防止初始化未完成时分区 1 就被切换走，再加入一个状态变量标志初始化是否完成。

```
typedef struct test_data
{
    SEM_ID partition1;
    SEM_ID partition2;
    int already;
    SendData* send_data;
    int singal_out;
    int signal_adjust;
} TEST_DATA;
```

**问题**：执行错误。二进制信号量无法跨分区使用。

## 简单的 PV 操作模拟

最终我们使用 `int` 来模拟二进制信号量。为 0 时死循环阻塞，为 1 时赋为 0 并往下执行。

```
typedef struct test_data
{
    int partition1;
    int partition2;
    int already;
    SendData* send_data;
    int singal_out;
    int signal_adjust;
} TEST_DATA;
```

P 操作的简单模拟

```
P: while (partition==0);
    partition=0;
```

---

V 操作的简单模拟

V:   partition=1;

优点：可以实现同步控制，根据目前的实践结果来看，这样的做法是比较好的实践做法。

缺点：无法保证整型赋值操作的原子性，调度时存在风险：若在赋值时切换发生结果不可预知。效率较低。

## 遇到的问题

在使用共享区时，在集成完毕后我们开始分区间进行收发，但是遇到了一个很奇怪的问题：

我们在写入共享区时，使用了类似如下的结构，期望能用简单的一句话来实现共享区的写入：

```
Writetoshare(SendData** send){  
    pTest->send_data = send;  
}
```

但是实践发现，这样传送数据，是完全不正确的。

我们在仔细分析之后得出了这样做的问题所在：如果这样做的话，分区 1 和分区 2 共享的并不是一个原子数据，而是一个指针。指针本身所代表的都是虚拟地址，所以在分区 1 中该指针指向了分区 1 的一片物理内存，但在分区 2 中，同样虚拟空间的指针，并不一定能指向同样的物理内存，并且往往指向不了同样的物理内存，因为分区 1 和分区 2 的虚拟地址到物理地址的映射是不相同的！所以最后我们发现，在 SendData 结构体中的数据也还是需要直接放在 TestData 结构体中，即共享区的结构体中只能使用**原子数据**。

## 4.4.3、姿态调整

滚转角（roll）偏航角（pitch）俯仰角（yaw）

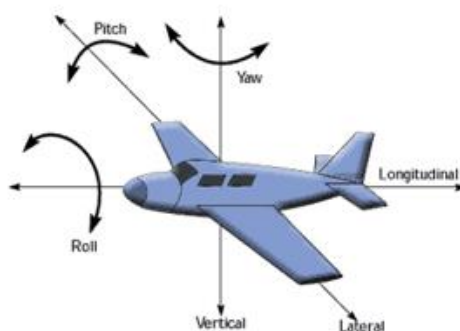


图 21 机体坐标系与姿态角

如 图 21 机体坐标系与姿态角 首先依照下列标准建立机体坐标系：



- 1) 原点 O 取在飞机质心处，坐标系与飞机固连；
- 2) x 轴在飞机对称平面内并平行于飞机的设计轴线指向机头；
- 3) y 轴垂直于飞机对称平面指向机身右方；
- 4) z 轴在飞机对称平面内，与 x 轴垂直并指向机身下方。

在建立了机体坐标系的基础上我们定义姿态角：

**俯仰角 $\theta$  (pitch)：**

机体坐标系 X 轴与水平面的夹角。当 X 轴的正半轴位于过坐标原点的水平面之上（抬头）时，俯仰角为正，否则为负。

**偏航角 $\psi$  (yaw)：**

机体坐标系 xb 轴在水平面上投影与地面坐标系 xg 轴（在水平面上，指向目标为正）之间的夹角，由 xg 轴逆时针转至机体 xb 的投影线时，偏航角为正，即机头右偏航为正，反之为负。

**滚转角 $\Phi$  (roll)：**

机体坐标系 zb 轴与通过机体 xb 轴的铅垂面间的夹角，机体向右滚为正，反之为负。

**导弹速度模型的建立**

球坐标是三维坐标系的一种，用以确定三维空间中点、线、面以及体的位置，它以坐标原点为参考点，由方位角、仰角和距离构成。

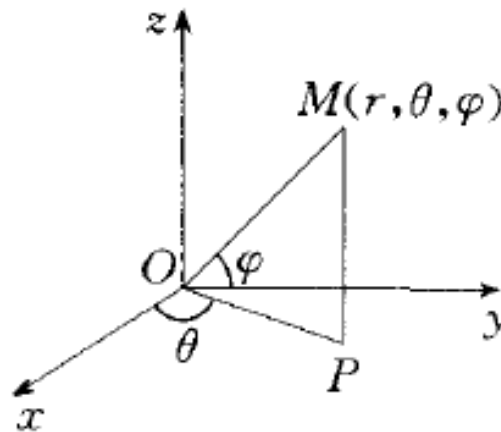


图 22 球坐标示意图

通过比对 图 21 机体坐标系与姿态角 与 图 22 球坐标示意图 发现  $\theta$  表示偏航角 (pitch)， $\varphi$  表示俯仰角(yaw)。由此建立导弹球坐标下速度模型，

$$\begin{cases} v_x = V \cos \theta \sin \varphi \\ v_y = V \sin \theta \sin \varphi \\ v_z = V \cos \varphi \end{cases}$$

---

其中， $V$  表示合速度。调整过程近似认为瞬时完成，且  $V$  的大小不发生改变。

需要注意的是，这个过程中的  $\theta$  和  $\varphi$  并不是姿态角。这是因为，导弹是不断进行调整的，其瞬态的姿态朝向和当前飞行的速度方向很有可能是不一样的。因而具体导弹关于坐标系下的  $\theta$  和  $\varphi$  之能通过三角函数反解出来。如当前通过分区间通信得到三个方向上速度分量  $v_x, v_y, v_z$ ，因而求解出

$$\begin{cases} v = (v_x^2 + v_y^2 + v_z^2)^{\frac{1}{2}} \\ \varphi = \arccos\left(\frac{v_z}{v}\right) \\ \theta = \arcsin\left(\frac{v_y}{v \sin \varphi}\right) \end{cases}$$

### 导弹姿态控制方案确定

导弹姿态调整共有 5 种方案：

- [1] “none”：不调整
- [2] “pitch”：加大俯仰角
- [3] “-pitch”：减小俯仰角
- [4] “yaw”：加大偏航角
- [5] “-yaw”：减小偏航角

通过斜抛运动算出当前落点(公式),

$$\begin{cases} x = v_x t \\ y = v_y t \\ H = v_z t + 1/2 g t^2 \end{cases}$$

其中  $H$  分别表示当前的高度， $t$  表示斜抛所需要的时间， $x$ 、 $y$  表示从当前时刻开始做斜抛运动导弹落点的经度和纬度。

设目标点坐标为  $(x_f, y_f, 0)$ ，则最终落点与目标点距离差值为

$$dis = \sqrt{(x - x_f)^2 + (y - y_f)^2}$$

比较五种调整方案计算所得的  $dis_k$ ，寻找满足使  $dis_k$  最小的  $k$  值，并将其认作当前调整

方案。给定范围阈值  $R$ ，若计算得到的  $(dis)_{\min} < R$ ，认为导弹可以击中目标，调整结束。若调整未结束，则进入下一个循环。同时，由于调整阶段只能在 100km 以上进行，所以若当前高度  $H < 100000m$ ，也认为调整结束。

### 具体工作流程

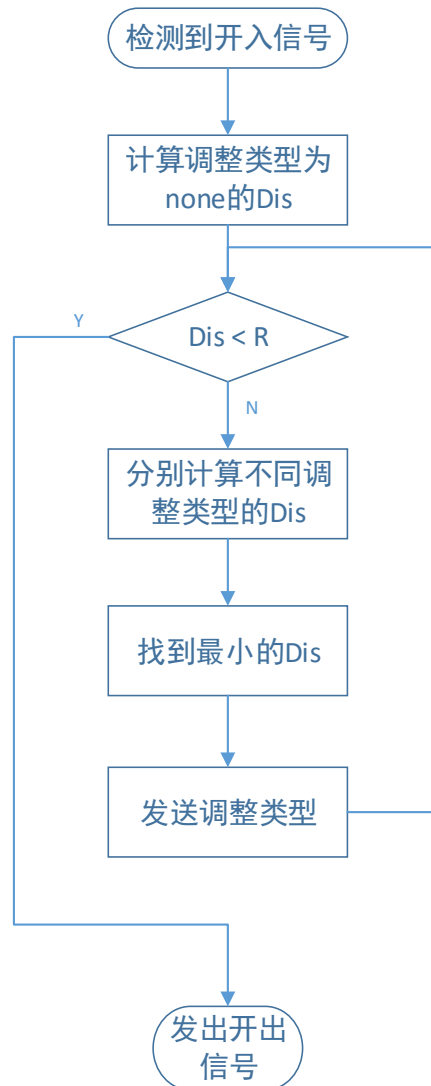


图 23 姿态调整部分流程图

工作流程如 图 23 姿态调整部分流程图 所示。在分区二中循环检测开入信号是否给出，当  $signal\_in=1$ ，即开入信号发出时，导弹进入被动段，执行调整函数  $adjust()$ 。在  $adjust()$  中首先计算不调整时导弹的落点，并判断是否打中目标。若打中目标，则发出开出信号，使  $signal\_out=1$ ；若没有打中，则调用  $pitch\_inc$ ,  $pitch\_dec$ ,  $yaw\_inc$ ,  $yaw\_dec$  计算采用其余四种调整方案时落点的距离差值，并找到差值的最小值，将对应的调整方案发送给 OS。循环运行，直至调整结束。

#### 4.4、存在问题及后续规划

本算法到最终调试集成的时候仍然存在问题，导弹的最高点为 1140km 左右，如图 24 飞行姿态最高点 所示：

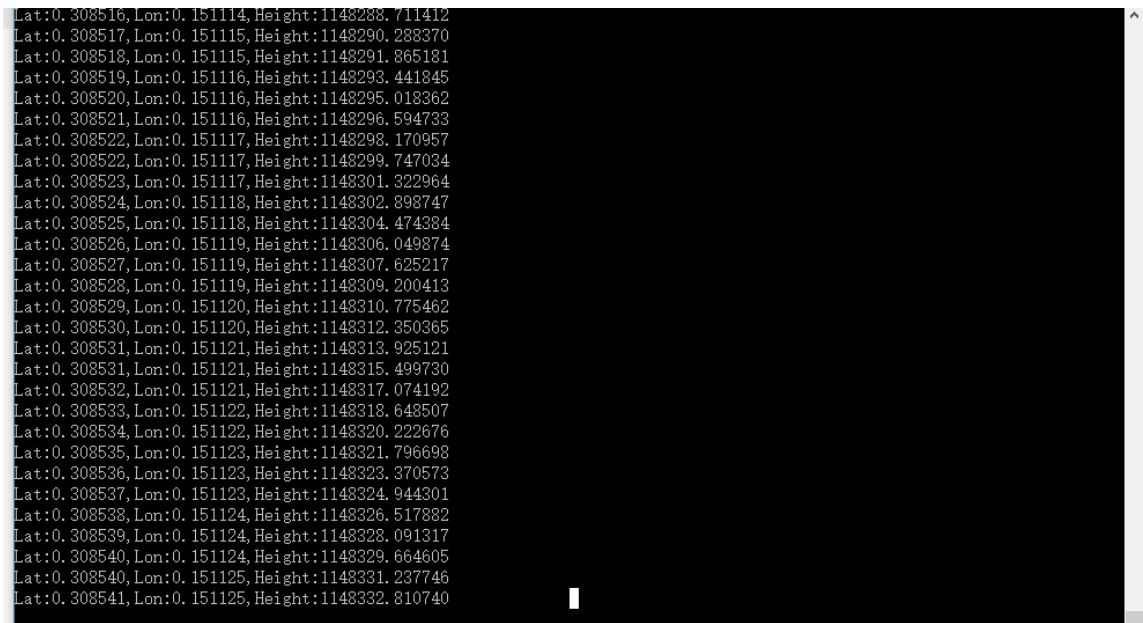


图 24 飞行姿态最高点

其最后落点为大约在(0.4372,0.2154) 图 25 最终落点 的地方

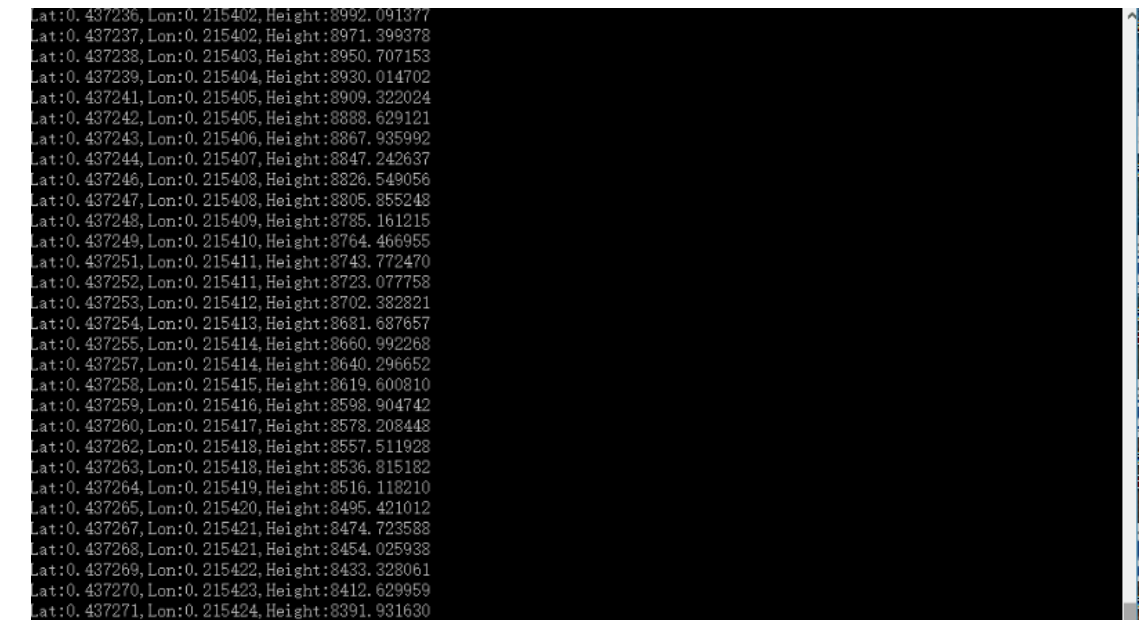


图 25 最终落点

离理想目标点(0.4,0.33)距离为 13.339km。误差较大，但因为时间与调试精力的关系，实在没有时间调试算法来保证最终落点的精确度。

---

## 5、图像识别单元

### 5.1 环境配置

OpenCV 的全称是：Open Source Computer Vision Library。OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法，可以运行在 Linux、Windows 和 Mac OS 操作系统上。

有很多用 CMake 编译 OpenCV 的安装教程，这里建议先不要自己编译，如果使用预编译好的库有问题，再尝试自己编译。

这里使用的是 opencv2.4.4 与 VC2010.操作系统为 XP。

#### 1. 配置环境变量

系统：PATH      用户：OpenCV

#### 2. 配置 VC2010

配置可执行文件目录、包含目录、库目录

配置附加依赖性

### 5.2 图像识别过程

根据匹配算法的基本思想可将图像匹配方法分成两大类，即基于区域的匹配方法和基于特征的匹配方法。而基于特征的图像匹配方法主要包括三步：特征提取、特征描述和特征匹配。



图 26 图像识别三步

以 SIFT 算法为例

基本过程：

1. 建立高斯差分尺度空间 DoG，在 DoG 空间中检测出极值点作为特征点
2. 特征点过滤并进行精确定位，剔除不稳定的特征点；
3. 用梯度方向直方图对提取出特征点进行描述

最后利用欧式距离作为度量对两幅图像中的特征点进行匹配。

## 5.3 具体实现

### 5.3.1 相关结构体及函数

特征点又称兴趣点、关键点，它是在图像中突出且具有代表意义的一些点，是图像特征的局部表达，它能反映图像上具有的局部特殊性。通过这些点我们可以用来识别图像。

特征点对应结构体：

```
class KeyPoint
{
    Point2f pt; //坐标
    float size; //特征点邻域直径
    float angle; //特征点的方向，值为[零,三百六十)，负值表示不使用
    float response;
    int octave; //特征点所在的图像金字塔的组
    int class_id; //用于聚类的 id
}
```

匹配得出的结果结构体

```
struct DMatch
{
    //三个构造函数
    DMatch():queryIdx(-1),trainIdx(-1),imgIdx(-1),
    distance(std::numeric_limits<float>::max()) {}
    DMatch(int _queryIdx,int _trainIdx,float _distance):
    queryIdx(_queryIdx),trainIdx(_trainIdx),imgIdx(-
1),distance( _distance) {}
    DMatch(int queryIdx,int _trainIdx,int _imgIdx,float _distance ):
    queryIdx(_queryIdx),trainIdx(_trainIdx),imgIdx( _imgIdx),distance(
_distance) {}
    int queryIdx; //此匹配对应的查询图像的特征描述子索引
    int trainIdx; //此匹配对应的训练(模板)图像的特征描述子索引
    int imgIdx; //训练图像的索引(若有多个)
    float distance; //两个特征向量之间的欧式距离，越小表明匹配度越高。
    bool operator < (const DMatch &m) const;
```

```
};
```

函数	函数作用
<code>void detectKeypoints(const Mat&amp; image, vector&lt;KeyPoint&gt;&amp; keypoints)</code>	检测特征点
<code>void extractDescriptors(const Mat&amp; image, vector&lt;KeyPoint&gt;&amp; keypoints, Mat&amp; descriptor)</code>	提取特征向量
<code>void bestMatch(const Mat&amp; queryDescriptor, Mat&amp; trainDescriptor, vector&lt;DMatch&gt;&amp; matches)</code>	最近邻匹配
<code>void knnMatch(const Mat&amp; queryDescriptor, Mat&amp; trainDescriptor, vector&lt;vector&lt;DMatch&gt;&gt;&amp; matches, int k)</code>	K 近邻匹配
<code>void saveKeypoints(const Mat&amp; image, const vector&lt;KeyPoint&gt;&amp; keypoints, const string&amp; saveFileName = "")</code>	保存特征点
<code>void saveMatches(const Mat&amp; queryImage, const vector&lt;KeyPoint&gt;&amp; queryKeypoints, const Mat&amp; trainImage, const vector&lt;KeyPoint&gt;&amp; trainKeypoints, const vector&lt;DMatch&gt;&amp; matches, const string&amp; saveFileName = "")</code>	保存匹配结果到图片中

本程序可支持多种不同的算子，将检测算子、提取算子、匹配算法作为参数输入。最后输出得到的匹配、经过筛选得到的移动距离，以及关键点图、匹配图。

特征提取算子：FAST、STAR、SIFT、SURF、MSER、HARRIS；特征描述算子：SIFT、SURF；特征匹配：BruteForce、BruteForce-L1、FlannBased。

### 5.3.2 程序流程

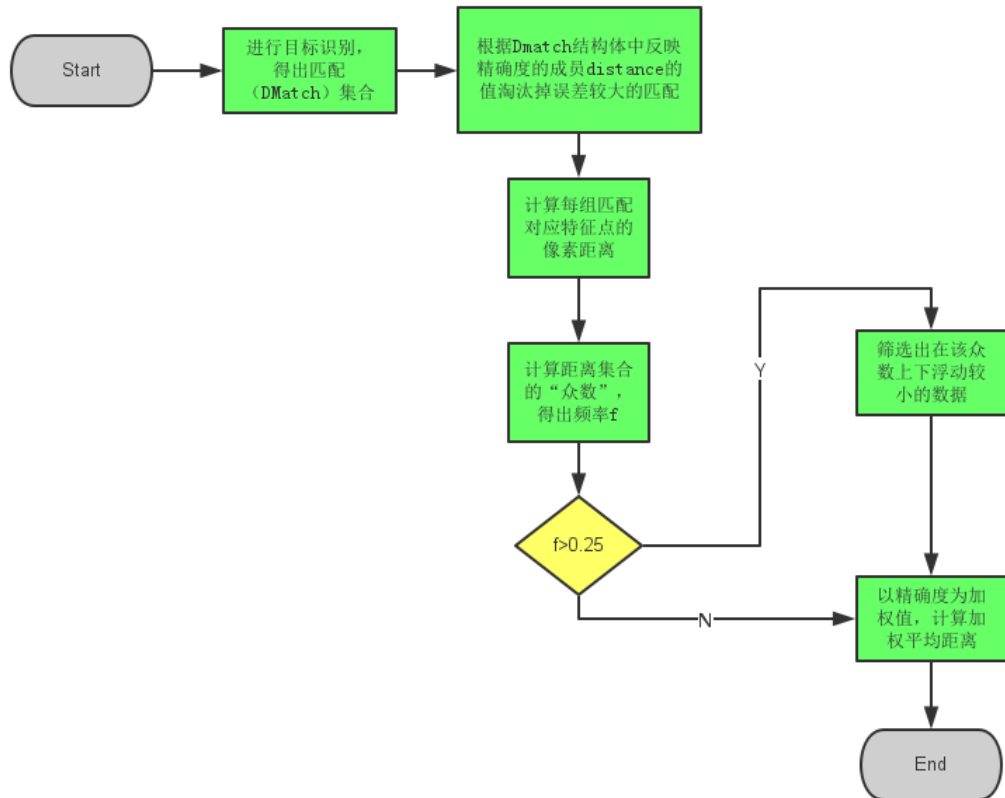


图 27 图像识别程序流程

1. 首先进行目标识别。先探测特征点，继而根据特征点提取出特征向量；最后接连运用最近邻匹配与 knn 匹配得出匹配集合。
2. 对匹配集合根据 distance 的值进行筛选（越小越精确），该值>100 的，全部舍弃。
3. 计算匹配集合中对应特征点的像素距离。
4. 因为像素距离是浮点数，计算每个距离的向下取整，筛选出频率最大的值。若该频率大于 0.25，则以该众数±4 为范围，再进行筛选；否则直接进行第 5 步
5. 因为留下来的匹配的 distance 全部小于 100，而 distance 与精确度有是反相关。所以以 100-distance 为权值，进行加权求平均距离。
6. 将求得平均距离换算为要求的经纬度距离

### 5.4 结果筛选

匹配出的结果会存在很大的方差，要进行筛选。



输出格式为 训练图像特征点序号、查询图像特征点序号、distance、对应特征点像素距离

```
216 281 227.348 28.4299
217 306 136.195 83.6778
218 555 70 239.36
219 313 75.3193 85.1218
220 314 99.3328 85.1218
221 316 65.1844 238.965
222 228 279.492 423.333
223 321 112.889 239.32
224 323 127.515 239.104
225 222 233.609 175.884
226 324 75.5844 238.636
227 271 199.437 144.38
228 272 197.833 144.38
229 330 87.0345 239.343
230 142 359.391 356.956
231 44 218.018 266.197
232 6 186.607 567.549
233 86 174.281 532.347
234 691 205.4 40.4515
235 231 289.955 310.912
236 568 135.222 239.318
237 279 220.645 510.536
238 77 248.544 194.506
239 560 259.035 27.4077
240 340 127.726 238.245
241 284 260.156 684.892
242 541 258.366 121.451
243 284 265.991 386.118
244 541 268.578 227.112
245 284 272.312 77.3756
246 342 12.0416 227.006
247 343 13.6382 227.006
248 344 98.1631 226.94
249 346 52.7162 227.05
250 349 102.255 226.888
251 350 28.9655 226.976
252 355 231.167 114.266
253 412 191.763 45.8652
```

图 28 结果偏差

步骤：

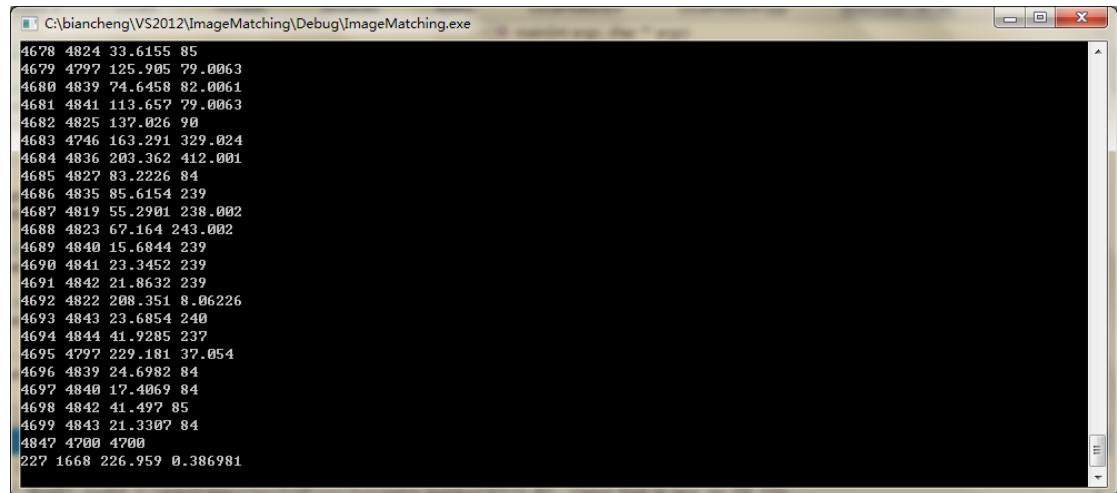
1. 选取 distance 较小 ( $<100$ ) 的匹配, 计算出该匹配对应的查询图像与训练图像点的距离, 组成一个集合
2. 在集合中查找出现频率最大 (取整) 的一个
3. 若该频率  $\geq 1/4$ , 则以该数据  $\pm 4$  为标准, 再筛选出一批数据; 若  $< 1/4$ , 则舍弃
4. 对最终筛选出的数据求平均值 (问题: 权值问题)

## 5.5 输出

输出格式为 训练图像特征点序号、查询图像特征点序号、distance、对应特征点像素距离

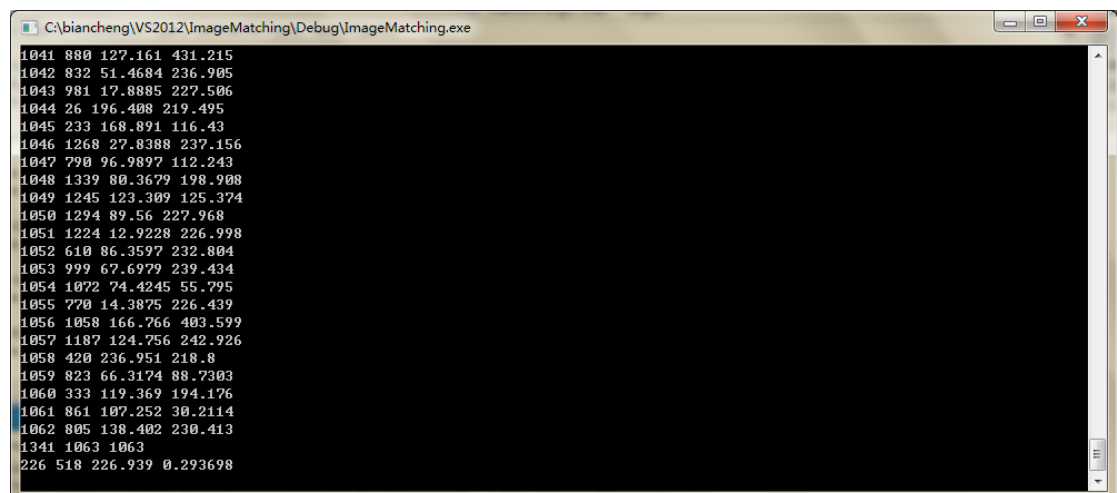
倒数第二行：训练图像特征点总个数、 查询图像特征点总个数、匹配总数

倒数第一行：众数、最终筛选出的匹配个数、求得的最平均像素距离、众数频率



```
C:\biancheng\VS2012\ImageMatching\Debug\ImageMatching.exe
4678 4824 33.6155 85
4679 4797 125.905 79.0063
4680 4839 74.6458 82.0061
4681 4841 113.657 79.0063
4682 4825 137.026 90
4683 4746 163.291 329.024
4684 4836 203.362 412.001
4685 4827 83.2226 84
4686 4835 85.6154 239
4687 4819 55.2901 238.002
4688 4823 67.164 243.002
4689 4840 15.6844 239
4690 4841 23.3452 239
4691 4842 21.8632 239
4692 4822 208.351 8.06226
4693 4843 23.6854 240
4694 4844 41.9285 237
4695 4797 229.181 37.054
4696 4839 24.6982 84
4697 4840 17.4069 84
4698 4842 41.497 85
4699 4843 21.3307 84
4847 4700 4700
227 1668 226.959 0.386981
```

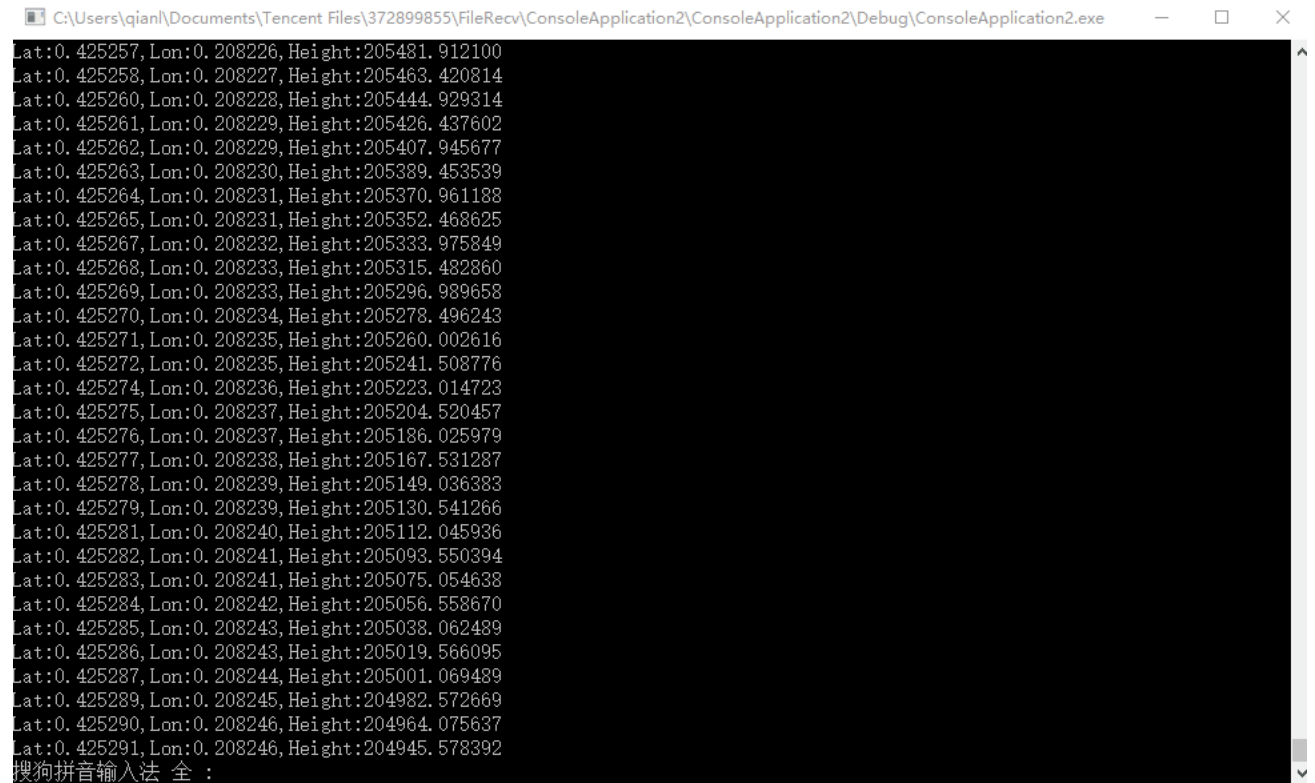
图 29 FAST SIFT FlannBased 对应结果



```
C:\biancheng\VS2012\ImageMatching\Debug\ImageMatching.exe
1041 880 127.161 431.215
1042 832 51.4684 236.905
1043 981 17.8885 227.506
1044 26 196.408 219.495
1045 233 168.891 116.43
1046 1268 27.8388 237.156
1047 790 96.9897 112.243
1048 1339 80.3679 198.908
1049 1245 123.309 125.374
1050 1294 89.56 227.968
1051 1224 12.9228 226.998
1052 610 86.3597 232.804
1053 999 67.6979 239.434
1054 1072 74.4245 55.795
1055 770 14.3875 226.439
1056 1058 166.766 403.599
1057 1187 124.756 242.926
1058 420 236.951 218.8
1059 823 66.3174 88.7303
1060 333 119.369 194.176
1061 861 107.252 30.2114
1062 805 138.402 230.413
1341 1063 1063
226 518 226.939 0.293698
```

图 30 SURF SIFT FlannBased 对应结果

## 6、最终结果



```
C:\Users\qian\Documents\Tencent Files\372899855\FileRecv\ConsoleApplication2\ConsoleApplication2\Debug\ConsoleApplication2.exe
Lat:0.425257, Lon:0.208226, Height:205481.912100
Lat:0.425258, Lon:0.208227, Height:205463.420814
Lat:0.425260, Lon:0.208228, Height:205444.929314
Lat:0.425261, Lon:0.208229, Height:205426.437602
Lat:0.425262, Lon:0.208229, Height:205407.945677
Lat:0.425263, Lon:0.208230, Height:205389.453539
Lat:0.425264, Lon:0.208231, Height:205370.961188
Lat:0.425265, Lon:0.208231, Height:205352.468625
Lat:0.425267, Lon:0.208232, Height:205333.975849
Lat:0.425268, Lon:0.208233, Height:205315.482860
Lat:0.425269, Lon:0.208233, Height:205296.989658
Lat:0.425270, Lon:0.208234, Height:205278.496243
Lat:0.425271, Lon:0.208235, Height:205260.002616
Lat:0.425272, Lon:0.208235, Height:205241.508776
Lat:0.425274, Lon:0.208236, Height:205223.014723
Lat:0.425275, Lon:0.208237, Height:205204.520457
Lat:0.425276, Lon:0.208237, Height:205186.025979
Lat:0.425277, Lon:0.208238, Height:205167.531287
Lat:0.425278, Lon:0.208239, Height:205149.036383
Lat:0.425279, Lon:0.208239, Height:205130.541266
Lat:0.425281, Lon:0.208240, Height:205112.045936
Lat:0.425282, Lon:0.208241, Height:205093.550394
Lat:0.425283, Lon:0.208241, Height:205075.054638
Lat:0.425284, Lon:0.208242, Height:205056.558670
Lat:0.425285, Lon:0.208243, Height:205038.062489
Lat:0.425286, Lon:0.208243, Height:205019.566095
Lat:0.425287, Lon:0.208244, Height:205001.069489
Lat:0.425289, Lon:0.208245, Height:204982.572669
Lat:0.425290, Lon:0.208246, Height:204964.075637
Lat:0.425291, Lon:0.208246, Height:204945.578392
搜狗拼音输入法 全：
```

图 31 最终动态图展示