专业实践报告

2014011451 计41 甘雨田

【设计题目】

研究Marvelmind Indoor Navigation System 的精度及其使用，并将该室内定位系统与先锋机器人小车相结合。

【实验平台】

系统平台：

Marvelmind Indoor Navigation System ： Windows 10

先锋机器人小车 ： Ubuntu 16.04

实验语言：

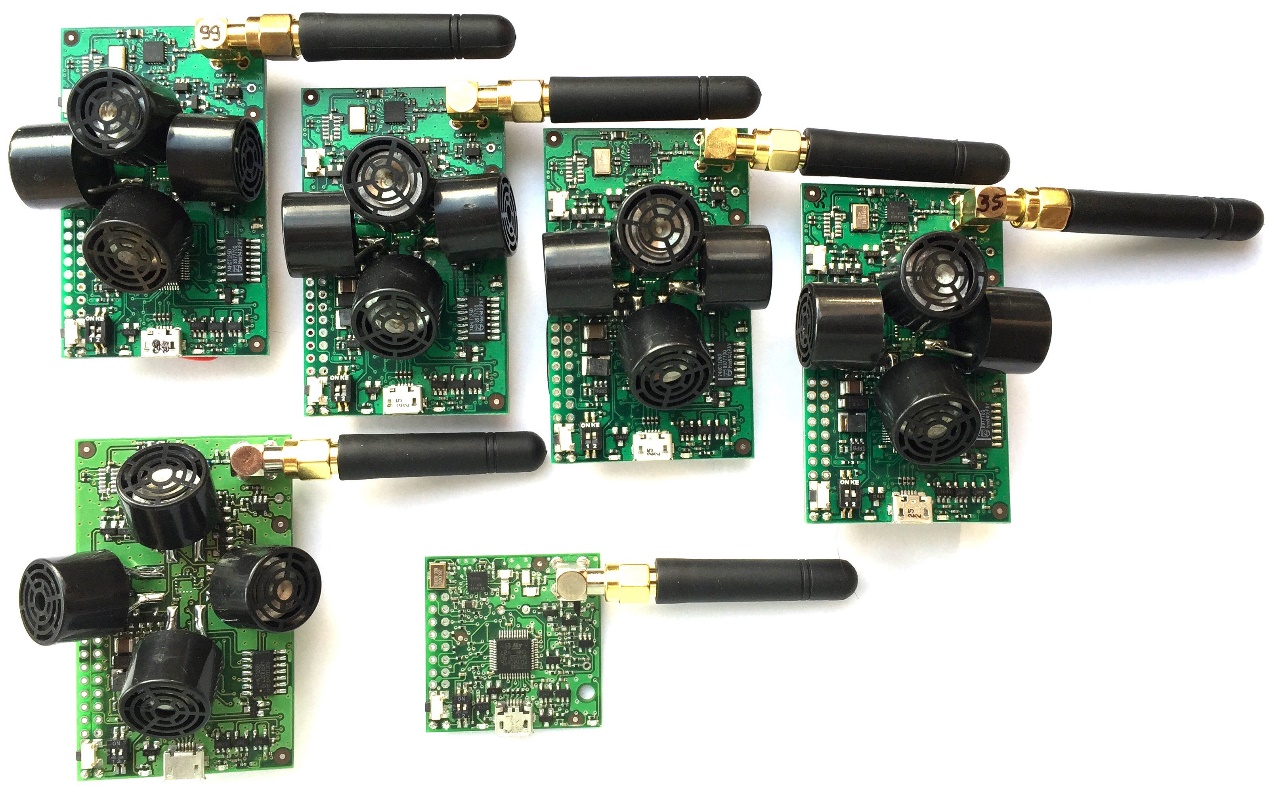
Marvelmind及Socket通信：C

足球机器人小车控制： C++

【实验设计及原理】

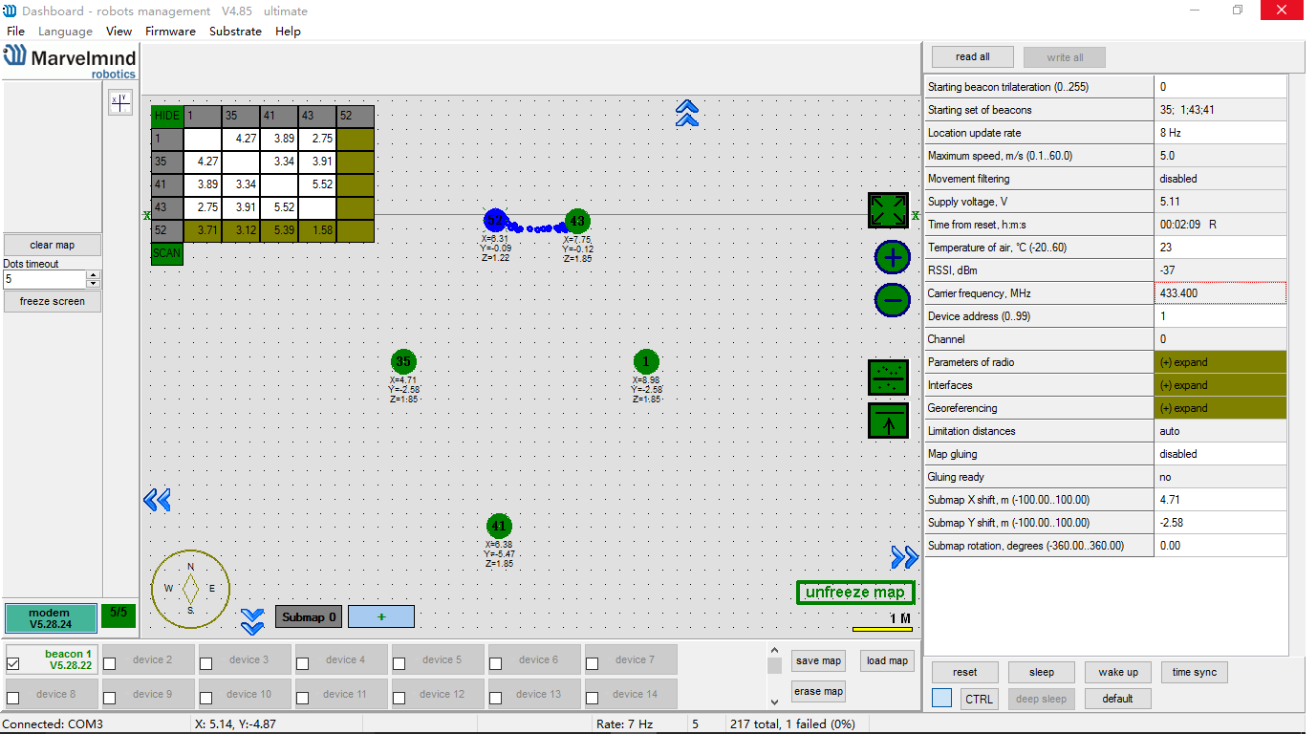
Marvelmind Indoor Navigation System系统是一套高精度的超声室内定位系统，能够解决传统定位系统不能在室内正常工作或精度不够高的问题。该定位系统的最高精度为2厘米，对于本实验中使用的机器人小车来说，该精度足够较成功地完成实验。

在本次实验中，使用了Marvelmind Indoor Navigation System系统的5个Beacon（4个作为固定基站，一个作为移动Beacon）和1个Router（接收位置信息的路由）。



Marvelmind官网上提供了一个自带软件Dashboard及两个C程序（marvelmind.h和marvelmind.c）。

首先研究Dashboard的使用。一开始按照说明文档进行操作，发现仪器总会出现一些奇怪的问题，比如时间延迟特别大，甚至有时仪器在Dashboard上找不到。这时给Marvelmind公司的工程师发送邮件，向他请教这类问题的原因和解决办法，并且在Youtube上能找到少量的关于Dashboard的软件的使用的视频，最后发现是移动Beacon的频率参数要调的比基站Beacon更高，并且将移动Beacon的Hedge Mode参数调整为enable（默认为disable）。



对于官网提供的两个C程序，则几乎没有说明文档和代码注释。研读代码时，总结出其中的接口的作用：

* struct MarvelmindHedge \* createMarvelmindHedge ()

该结构体能够收集Beacon传来的数据。

* void startMarvelmindHedge (struct MarvelmindHedge \* hedge)：

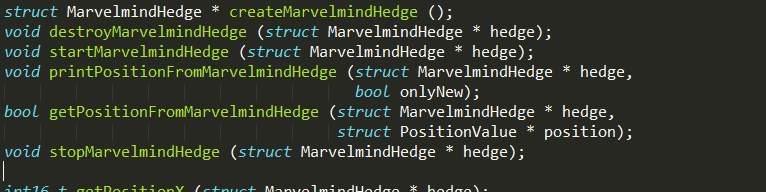
用于声明一个MarvelmindHedge结构并初始化。

* void stopMarvelmindHedge (struct MarvelmindHedge \* hedge)：

停止MarvelmindHedge的一个线程。

* void destroyMarvelmindHedge (struct MarvelmindHedge \* hedge)：

该函数用于释放内存。



利用以上接口，我实现了以下四个C程序接口：

* int16\_t getPositionX (struct MarvelmindHedge \* hedge)：

用于获得移动Beacon的X坐标，数据类型为int16\_t。

* int16\_t getPositionY (struct MarvelmindHedge \* hedge)：

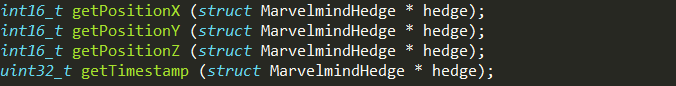
用于获得移动Beacon的Y坐标，数据类型为int16\_t。

* int16\_t getPositionZ (struct MarvelmindHedge \* hedge)：

用于获得移动Beacon的Z坐标，数据类型为int16\_t。

* uint32\_t getTimestamp (struct MarvelmindHedge \* hedge)：

用于获得当前的时间信息，数据类型为uint32\_t。



坐标系分析：通过以上C程序接口，我们能够获得小车（移动Beacon）在坐标系中的坐标信息。为了最大程度地利用该坐标信息，我们还需要知道该坐标系与空间的对应关系，因此需要进行坐标系分析。先将四个基站Beacon固定放置，然后测出基站Beacon之间的位置关系，再将移动Beacon分别放置在四个基站Beacon的位置，测出其坐标信息，由此可以得到该系统建立的坐标系与空间坐标之间的关系。

Marvelmind Indoor Navigation System系统与先锋机器人小车的结合：

我们组的先锋机器人小车实现了自主避障等功能，我们准备了一个“S”型弯跑道，将跑道的入口处设置在定位系统建立的坐标原点，而小车（车上放置移动Beacon）放在该坐标系的任意位置，则小车能通过获取到的实时位置，自动导航到跑道入口处（坐标系的坐标原点）。

由于该室内定位系统目前仅在windows系统下有对应驱动，而同时实验小车的平台是在Ubuntu系统上，则需要让该定位系统在windows系统下运行，得到数据之后，与Ubuntu系统上的server端通过socket进行通信，将数据发送给server端。

【实验结果】

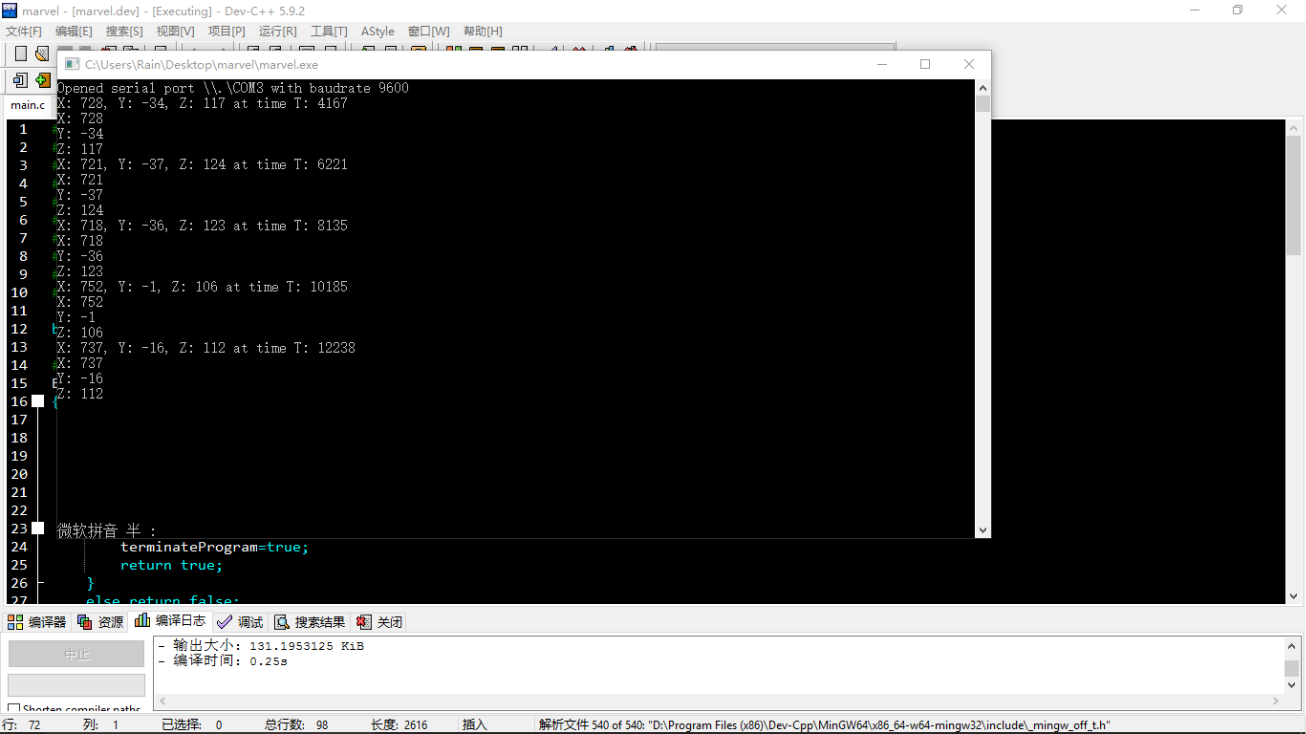
Dashboard的使用： 能够使用该自带软件显示出五个Beacon在坐标系中的相对位置关系，并且能够画出Mobile Beacon的移动轨迹。难点不多，主要难点是解决读取频率太低的问题，通过发邮件请教Marvelmind工程师和在Youtube上学习使用教程视频，问题得到解决。

Marvelmind接口：官方网站上提供的marvelmind.h和marvelmind.c两个C程序，不仅没有说明文档和程序注释，接口也非常难以使用。通过研读代码，自行写出了能够便于使用者直接使用的接口：

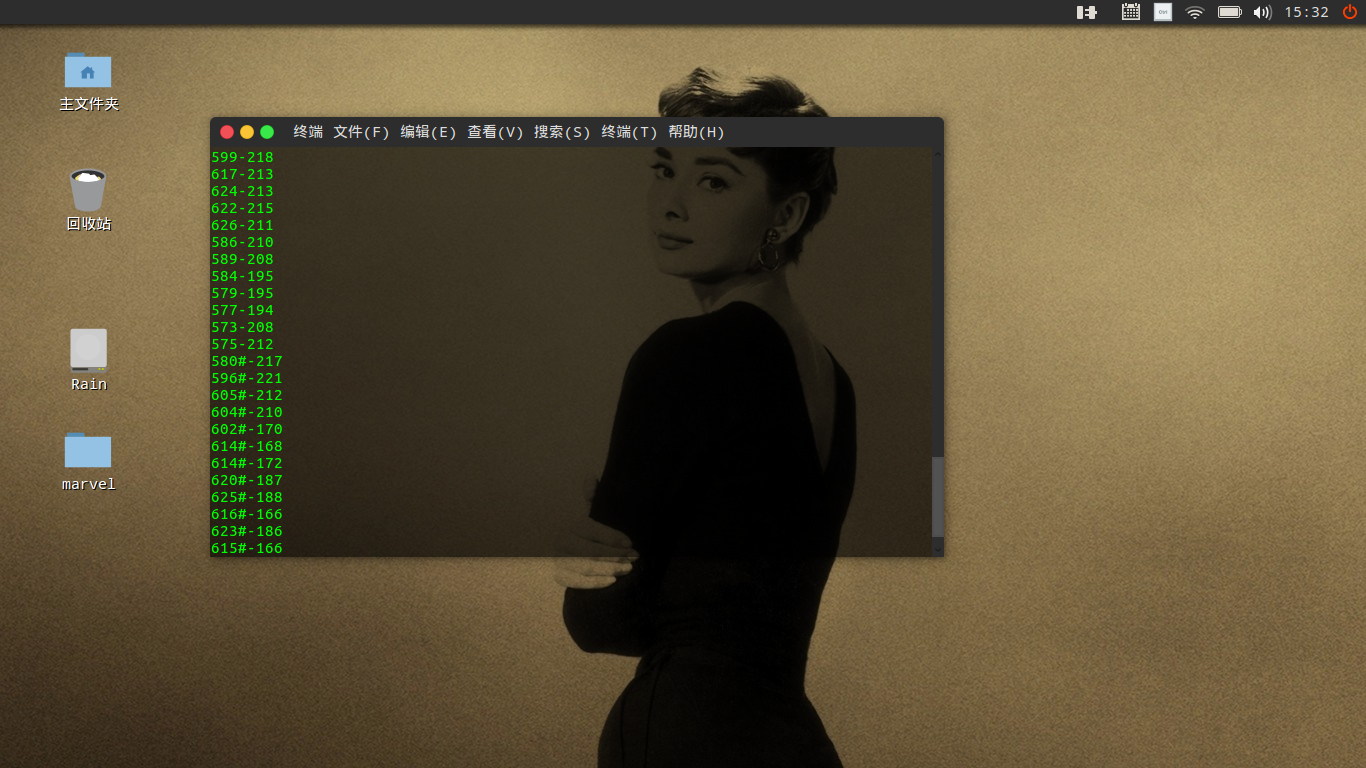
* + int16\_t getPositionX (struct MarvelmindHedge \* hedge)
  + int16\_t getPositionY (struct MarvelmindHedge \* hedge)
  + int16\_t getPositionZ (struct MarvelmindHedge \* hedge)
  + uint32\_t getTimestamp (struct MarvelmindHedge \* hedge)

如果以后有人需要继续使用Marvelmind Indoor Navigation System， 不需要再去阅读代码，直接通过这几个接口就能够获得Mobile Beacon的坐标信息及时间信息。

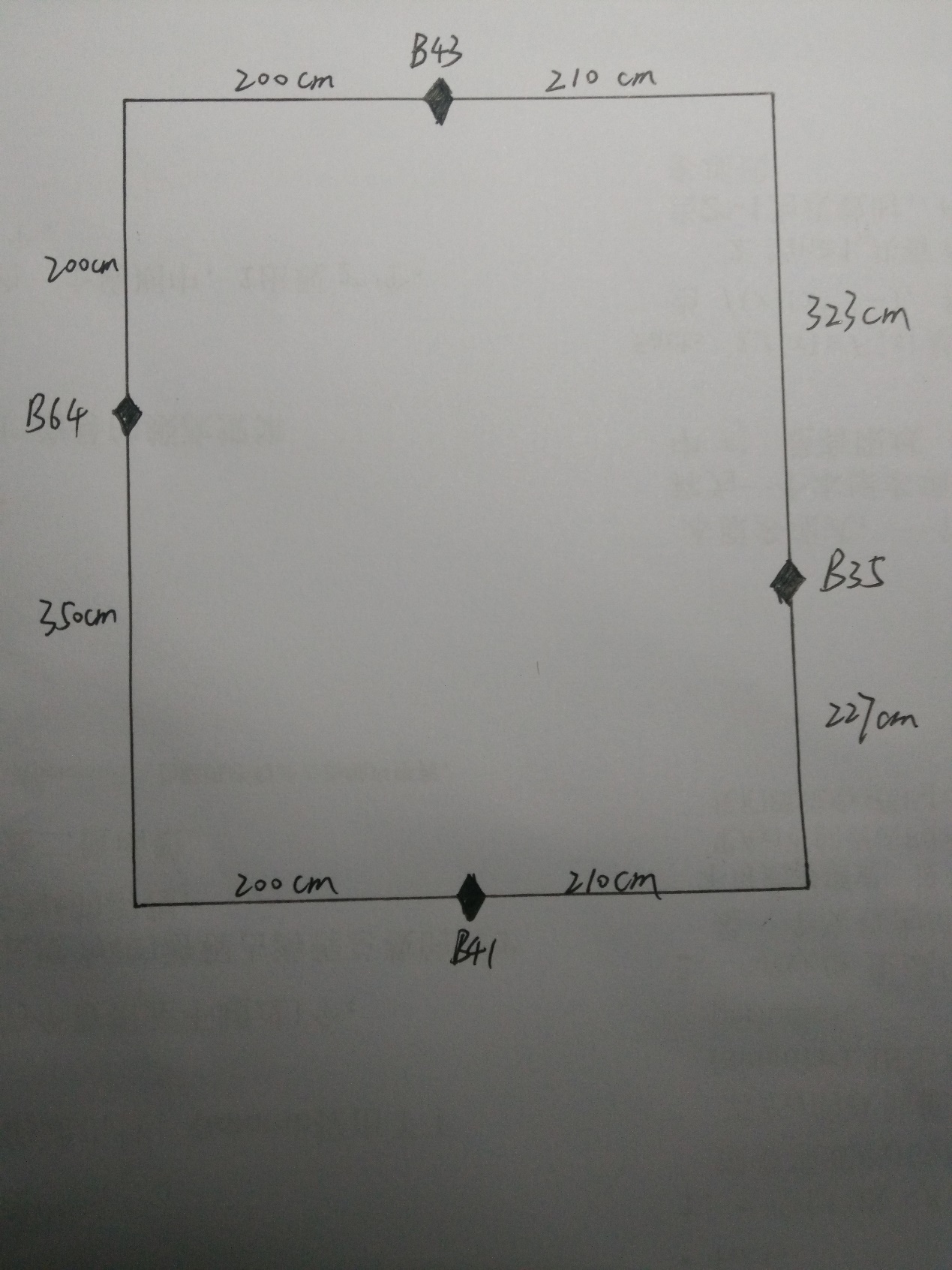
在Windows下通过以上接口，得到实验结果如下：



通过Socket通信，发送到Ubuntu上的server端，解析X，Y坐标信息之后，结果显示如下：



为了测出坐标系与空间坐标之间的对应关系，我在较为空旷的实验室（位于东主楼五楼）进行了测量。将四个基站放置在同一高度（1.80m），对应位置如下：



得到四个基站Beacon的坐标如下：

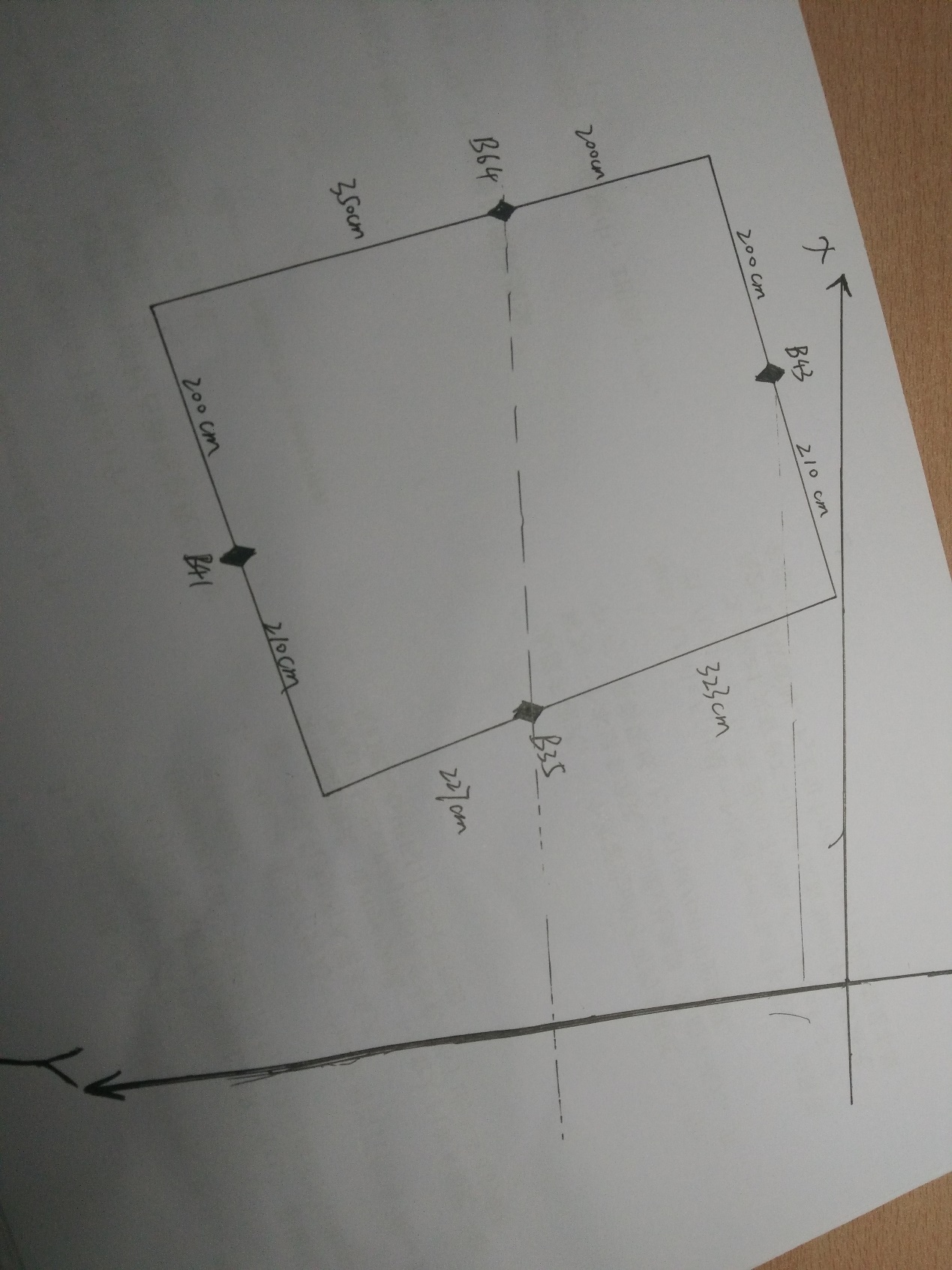
B43：（767，-25，130）

B64：（895，-256，140）

B41： (640, -532, 130)

B35： (467, -258, 142)

由此得到坐标系与空间的对应关系，坐标系如下图。



由此也能看出，在空旷的实验室也很难达到2cm的精度，一般情况下，定位的精度在10cm左右。

在得到空间位置与坐标系的关系之后，我们就能够找出坐标为任意值的点在空间中的位置，由此，我们将“S”型弯跑道的入口处设置在坐标原点处，然后让小车自动导航到该入口处。遗憾的是，由于场地障碍物等干扰，Mobile Beacon的坐标会有很大的概率出现突变，且对于实验结果影响极大。到目前为止，我们组只有2次成功将小车导航到“S”型弯跑道的入口处。

【实验总结】

在此次专业实践中，由于有五周时间，私以为时间非常充裕，于是在前三周里，我专注于研究Marvelmind的使用及其通信，与队友没有过多的交流沟通。但是在最后要与队友的工作合并的时候，才发现有些接口及对实验的理解有矛盾和冲突，导致在接近检查的时候，我们还在进行合并和调试。