**数字信号处理课程 Project 实验报告——**

**基于毫米波雷达的无弦琴**

无77 廖宇轩 2017011223 无77 周沛灵 2017011134

无77 陈斯昂 2017011280 无77 张咏镜 2017080063

**一、实验目的**

毫米波雷达工作频段在30~300 GHz，波长在1~10 mm 之间，具有体积小、质量轻、全天时全天候工作的特点。 调频连续波（FMCW， Frequency ModulationContinuous Wave）是毫米波雷达常用的波形之一。通过对 FMCW 信号进行处理，我们可以获取目标信息， 包括距离、速度（多普勒）和方位等，进一步地能实现许多复杂的功能， 例如自动驾驶中的自适应巡航和碰撞预警等，除此之外，还包括手势识别，室内定位以及健康监测等。本次课程 Project 任务就是利用毫米波雷达进行目标探测，通过对回波进行数字信号处理，提取出有用信息加以分析，实现一个简单的功能。

**二、方案设计**

Project的题目创意来源于前苏联物理学家利夫·特尔门教授发明的特雷门琴，特雷门琴是世界第一件电子乐器，它的弹奏不需要任何键或者弦，其原理是利用两个感应人体与大地的分布电容的LC 振荡器工作单元分别产生震荡的频率与大小变化而工作，至今仍是世上唯一不需要身体接触的电子乐器。[1]



Figure 1 特雷门琴

特雷门琴中实现“无弦”的关键就在于测量人体与大地之间的电容从而实现人手位置的判断，并进行发声。而毫米波雷达也可以进行物体的位置判断，所以我们应该也可以利用毫米波雷达来实现一个具有相似功能的电子乐器。

对于毫米波雷达，我们将雷达接收到的信号利用usb数据线传输到电脑上，再利用matlab程序，通过对四根接收天线接收到的回波进行一定的数字信号处理，提取出有用信息，就可以分析出人手的大致角度和距离，然后程序再对于人手所处的不同位置确定不同的声调输出，并最终由发声模块完成发声，如此循环，再辅以一定的UI界面，就可以利用毫米波雷达与与matlab程序实现一个基于毫米波雷达的无弦琴。

Project的代码主要分为以下模块[2]：

Project的主要分工如下：

廖宇轩同学主要完成UI界面的设计，整个程序各个模块的联调测试

周沛灵同学主要完成乐音的设计以及发声模块的设计

陈斯昂同学主要完成毫米波雷达物体检测模块的设计

张咏镜同学主要完成雷达硬件的调试、信号传输和各个模块之间接口的设计与优化

**三、算法实现**

1、人手位置的判断：

为了实现无弦琴的功能，我们首先要利用毫米波雷达四根天线接收到的信号进行物体位置的判断，要实现物体位置的判断，即分别实现物体角度和距离的判断。基于毫米波雷达发射信号的原理，对距离进行判断较为容易，因为雷达发送的是调频连续波，我们只需要将接收信号进行混频和低通滤波之后，我们就可以得到一个信号频率正比于物体距离的信号，通过对信号进行fft确定其峰值功率所在的频率，经过换算就可以得到物体的大致距离。

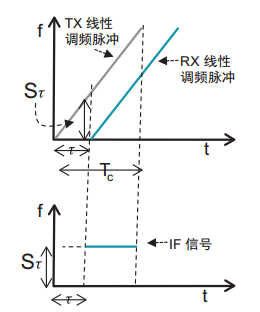


Figure 2 距离测量示意图

对于物体角度的测量就比较麻烦，在上面的距离测量方法中，实际上测量物体距离只需要一根接收天线接收到的信号，但是对于角度得到测量，只利用一根天线的接收信号肯定是不够的。在Project所给的毫米波雷达基础知识的手册中，提供了角度的测量的一个基本方法，利用两根天线接收信号的时间差（相位差），就可以大致计算出接收信号入射的方向（把接收信号看作一个波束，就是对应的来波方向）。

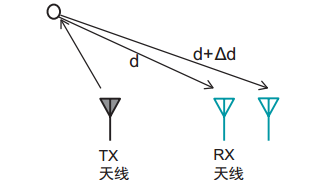


Figure 3 角度测量示意图

但是实际上由于雷达四根接收天线之间的距离相当接近，两根天线之间相位差实际上相当小，这个相位差实际上也很难确定，而且就算对四根天线两两计算后取平均，也不能充分利用四根天线接收到的信息以及它们的空间结构。所以我们在这里考虑将四根接收天线看成一个接收阵列，结合波束形成的方法来确定来波即物体反射的信号的方向。

波束形成算法用于确定多阵元的接受阵列接接收信号的方向性，对于毫米波雷达的接收天线，我们可以将其看做最简单的等距直线接收阵列：

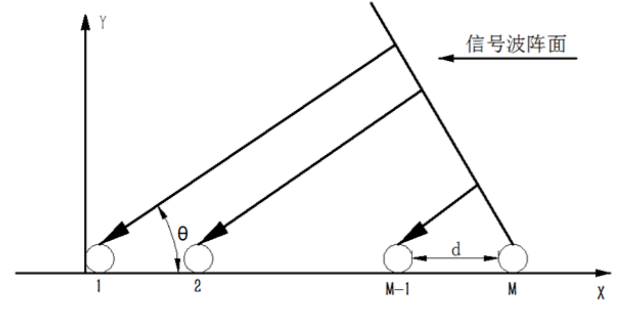
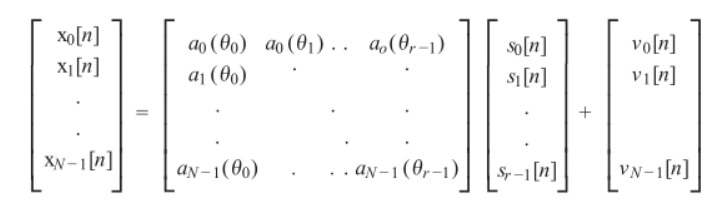


Figure 4 等距直线阵列示意图



C:\Users\MyPC\AppData\Local\Temp\1578829074(1).png

Figure 5 信号模型

s为实际信号，a为入射角度导致的相位变化，v为噪声

传统波束形成算法的目标就是针对不同入射方向的θ，计算出一组加权系数w，用于对不同阵元接收到的信号进行加权，从而得到在θ方向上期望的信号和功率，通过对不同θ进行扫描计算，就可以得到信号的空间谱估计，从而对来波方向进行确定。

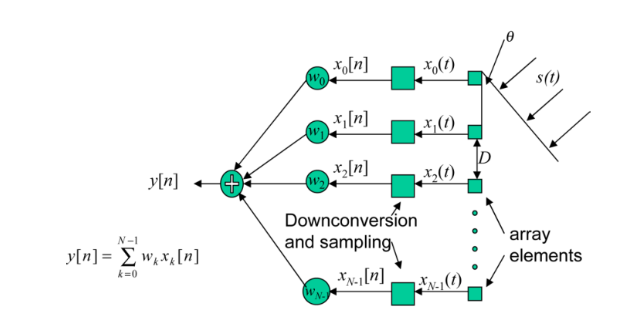
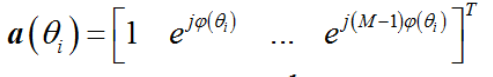


Figure 6 波束成形结构示意图

所以整个算法的关键就在于加权系数的确定，为了实现不同角度上的加权系数的确定，我们采用了两种使用较为广泛的波束成形算法：CBF算法与MVDR算法。

CBF算法的思想比较简单，就是对接收的信号进行延时加权求和，假定入射角度为θ，那么不考虑噪声对相位的延时，只考虑理想情况对应的延时就是：



C:\Users\MyPC\AppData\Local\Temp\1578831969(1).png

Figure 7 CBF算法原理

那么我们只需要取：

C:\Users\MyPC\AppData\Local\Temp\1578831890(1).png

就可以抵消掉延时的影响，理想情况下得到的加权信号就是实际的信号，对应θ方向上的功率大小，即空间功率谱为：

C:\Users\MyPC\AppData\Local\Temp\1578832246(1).png

R为信号的自相关矩阵

接下来我们只需要对不同角度进行扫描，得到不同角度下的加权信号，然后对其进行FFT，得到信号峰值所对应的的频率就是对应方向上的距离；或者计算出信号空间功率谱的峰值角度，得到对应方向即为计算得到的信号方向，再进行FFT处理。

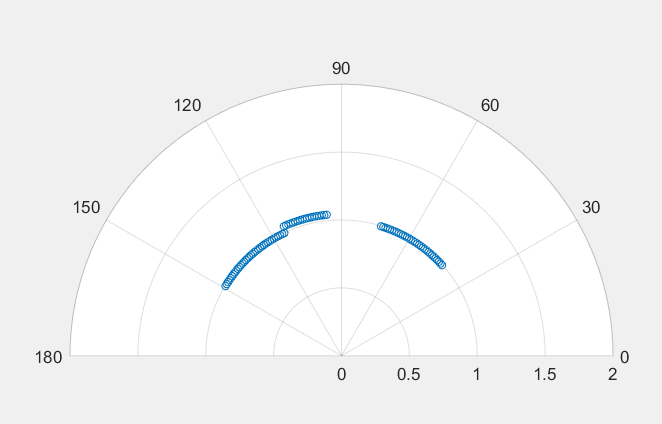
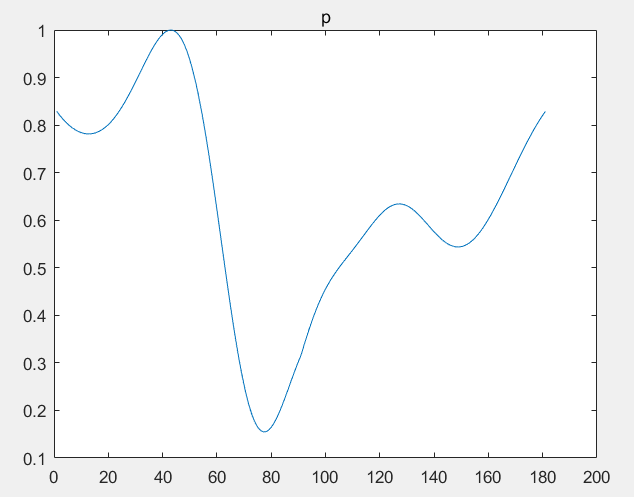


Figure 8 CBF算法直接进行扫描结果（横坐标单位为米，有两个目标，下同）



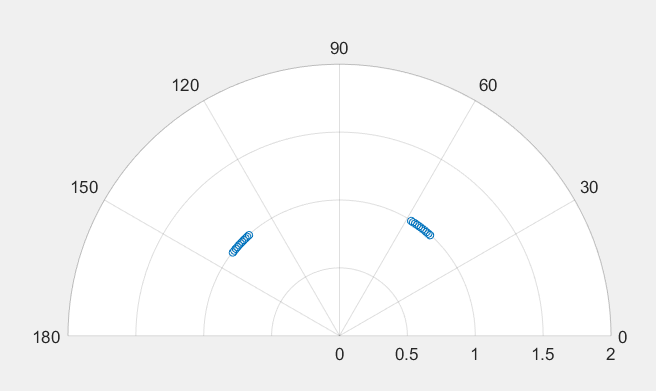


Figure 9 CBF算法峰值检测功率谱与结果

实际上由于物体有一定的大小，为了使之后确定位置比较方便，我们可以对检测结果进行聚类然后取平均值，将有一定大小的物体确定到一个点上，方便后续的处理。

由于信号阵列只有4个阵元，对于角度太大的点的分辨能力比较差，而且由于毫米波雷达的天线增益的影响，角度较小的信号的功率要比两边的信号功率大很多，在实际程序中为了稍微减轻天线增益的影响，给不同角度上的信号乘上了一定大小的增益（体现为代码中的gain，下面的MVDR算法同样也有这么一项）。

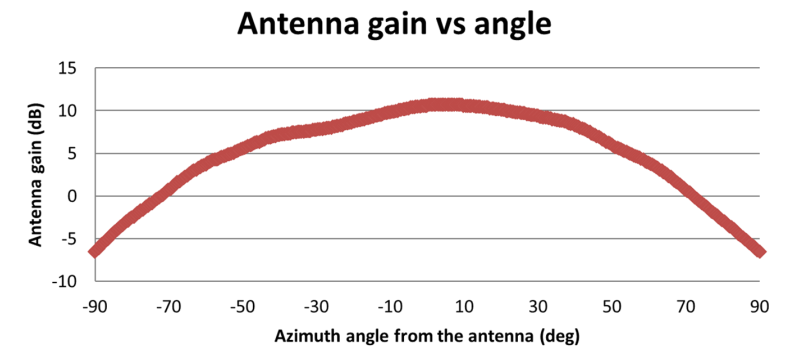


Figure 10 天线增益[3]

由于CBF算法仅仅考虑了相位差，没有考虑噪声等其他因素的干扰，实际实现的效果比较一般，所以考虑使用更好的算法。

MVDR算法实际上是基于线性约束最小方差进行设计的，实际上是一个线性约束最小方差（LCMV, linearly constrained minimum-variance）波束形成器。MVDR滤波器目的是得到在期望方向上的信号功率最小（这样子相当于让噪声充分对消）的系数w，即求解以下的优化问题：

C:\Users\MyPC\AppData\Local\Temp\1578829602(1).png

C:\Users\MyPC\AppData\Local\Temp\1578829750(1).png

Figure 11 MVDR对应的优化问题

我们考虑信号的自相关矩阵：

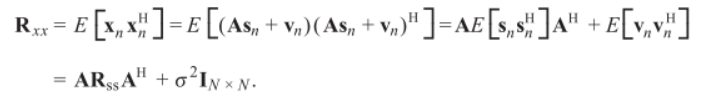


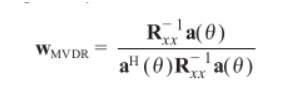
Figure 12 信号自相关矩阵

实际上如果wHw为定值的话，那么噪声对优化问题就没有影响，优化问题转变为：

C:\Users\MyPC\AppData\Local\Temp\1578829222(1).png

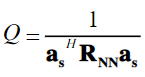
Figure 13 优化问题简化

从而根据LCMV的结果，可以得到我们要的系数w如下：



表达式中需要注意的是实际上我们应该用实际信号的自相关矩阵Rss进行计算，但是由于我们无法直接测得实际信号的自相关矩阵，在噪声较小的情况下，我们用接收信号的自相关矩阵Rxx进行近似计算的误差也是可以接受的，所以我们直接计算出x的自相关矩阵带入计算即可。

MVDR算法对应的空间功率谱密度为：



得到不同角度的加权系数以及空间功率谱之后，我们同样地只需要对不同角度进行扫描，得到不同角度下的加权信号，然后对其进行FFT，得到信号峰值所对应的的频率就是对应方向上的距离；或者计算出信号空间功率谱的峰值角度，得到对应方向即为计算得到的信号方向，再进行FFT处理，就可以得到大致的物体检测结果。

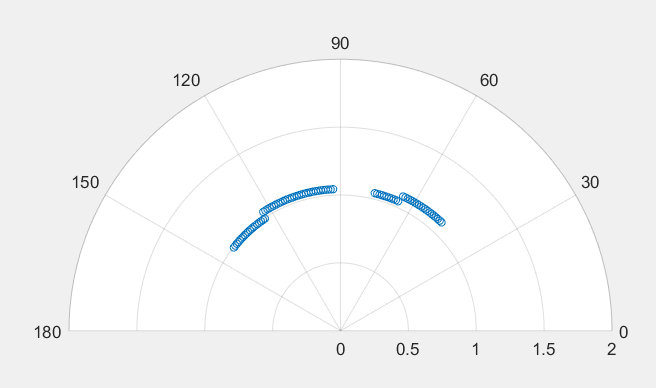
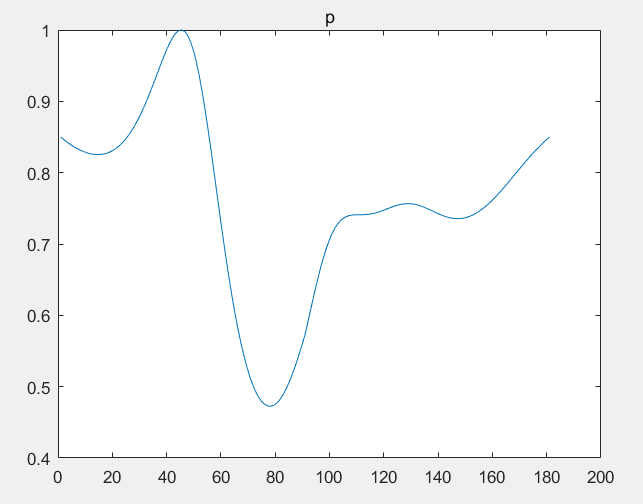


Figure 14 MVDR算法直接进行扫描结果（数据与上面一样，横坐标单位为米，有两个目标，下同）



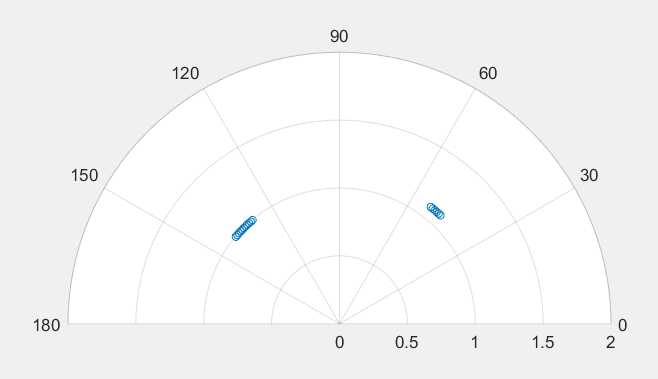


Figure 15 MVDR算法峰值检测功率谱与结果

在实际的算法中，由于使用的阵元数目相对较少，还对MVDR算法进行了一系列优化：在运算时对自相关矩阵进行了对角加载[4]，以加快算法特征值收敛；构造数据矩阵的共轭增广矩阵[5]，在阵元数较少的情况下略微改善解相关性能，具体原理就不再赘述。

C:\Users\MyPC\AppData\Local\Temp\1578833980(1).png

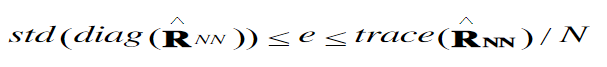


Figure 16 MVDR算法对角加载优化，加载量代码中取了上限

C:\Users\MyPC\AppData\Local\Temp\1578834058(1).png

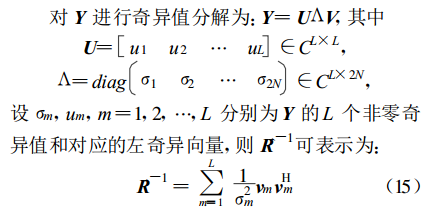


Figure 17 MVDR算法增广矩阵优化

2、程序从端口读取数据到送入get\_point\_plane函数处理至最后量化结果的过程：

① 端口读取：

程序原本的getRawData是一次读取一帧，本任务需要完成连续读取和数据处理，因此我们受首要工作是让程序能连续高速读取雷达数据。下面我们对样例代码进行分析、修改，可以达到连续读取雷达数据的效果。

数据的读取是通过sphandle（第84行）完成的，其具体内容在configureSport函数内定义。当端口一旦收到numSamples\_perRx\_perChirp比特数据后，它就会调用readData函数读取数据。本任务中，使用1个发射天线及4根接收天线，因此每个chirp传回的数据为4\*numAdcSamples\_t，每个chirp数据到达后，readData会将其加入bytevec中。当bytevec收到一帧内numChirp个数据后，就跳出等待循环，进入数据处理。

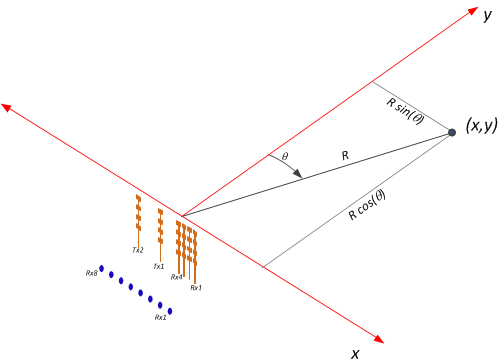


Figure 18 IWR1642信号收发原理图

首先注意到端口建立和拆除时间较长，直接对样例代码做循环达不到本任务的要求，因此将建立和拆除部分单独做成2个模块，分别置于程序的开始和结尾处。

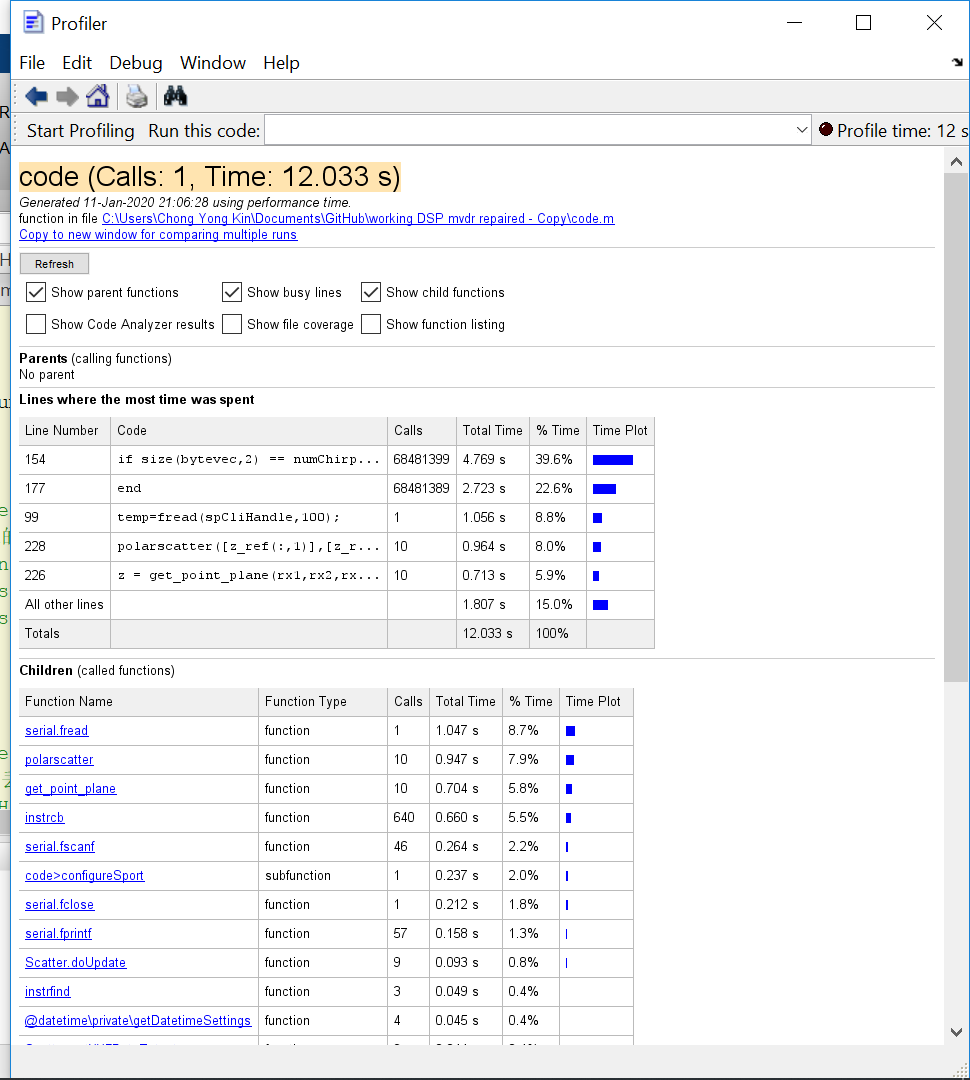


Figure 19 程序运行耗时分析

主要时间花费在等待bytevec完成接收

通过测试发现，数据从端口读入到内存的发生时间很快，为921600bps，但是相对于程序执行速度而言它是非常缓慢的，因此我们尝试寻找一个更快的方法完成读取，以下是我们尝试的方式：

- **修改FrameCfg里number of loops的数量**，因本任务只需要测量出距离和theta角，因此不需要发射多个调频波，当前.cfg下在一帧内会发射16个chirp，但本任务可以将其大大降低，以提高速度。不能成功的原因是因为demo固件要求iwr16xx的chirp数量最小为16。

- **通过读取需要的前几chirp数据直接进行数据处理**，忽略后面的传输数据。但在这样的模式下，数据处理（get\_point\_plane）的耗时远小于readData。最终结果是，数据处理好后，我们还在等待后续不会用到的chirp的返回数据被接收。

- **通过主动传输sensorStart、sensorStop指令来强制新数据传输**，我们发现，当前一轮数据没有完成传输时，第二帧的数据又要被发送时，雷达因为同时需要传输两组数据而出错，因此本程序会报“丢包错误”。

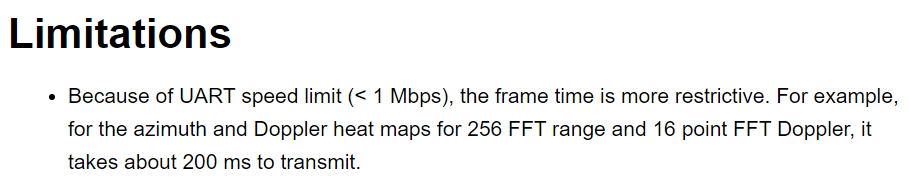


Figure 20 程序运行时间限制[6]

我们尝试了很多程序上的修改，最终也是没能将读取速度提升上去，官方SDK说明文档里指出，读取的速度受制于demo版固件， 我们虽然尝试通过降低chirp数量来减少数据传输时间，但最终还是没能达到理想效果，非常可惜。

②空间位置的量化：



Figure 21 雷达识别的一个例子

雷达会认为较近距离的遮挡时遮挡面积较大，因此我们需要进行分区和量化

对应到普通乐器，本任务的操作同样需要对手在空间中的位置进行量化。本雷达模块的原理中指出他能根据抵达不同接收天线间的相位差计算theta角，而theta角也仅在接近0度时能有较准确的估计，随着角度的增大，theta估计误差越大。【说明文档中的公式】在theta 接近0度（即雷达正上方）时距离估计更准确，随着距离的增大，被手遮挡后反射功率会减小，因此不应设置太多较远距离的音区。

实际测试中发现，40-60cm间定位准确率较高，因此选择在此距离范围内多划分音区。下图给出实际的音区划分，程序并通过pos2ind函数给出雷达监测到目标的最近量化音区。



Figure 22 本项目最终的量化中心，横坐标单位m

3、发音模块

利用位置量化给出的数字产生不同频率的乐音，0对应不产生乐音，1-10分别对应频率从低到高的十个乐音。函数输入为当前帧与上一帧所给出的量化值，若当前帧与上一帧的量化值相同，则维持乐音，若当前帧与上一帧的量化值不同，则停止播放上一帧的乐音，并根据频率对应关系播放新的乐音。

考虑乐音的选择，如果我们选择简单的正弦波乐音，那演奏效果不是特别好；如果直接采用真实音乐的乐音进行截取，在不同乐音之间的拼合可能会有问题，而且这样会使程序要不断地调用外部的音乐文件，对程序的运行效率也有一定的影响。

所以最终我们考虑对正弦的乐音进行一定的优化让其听起来更像真实音乐的乐音，效果很好，而且音调准确，不同乐音之间的衔接也比较自然。程序中的乐音为经过时域加窗的对应频率的正弦函数，通过sound函数进行播放。[7]

具体的加窗方式如下：

利用一个ScaleTable控制频率，利用三角函数的乘积进行加窗可以有较好的效果：

T16 = 0.125;

t16 = [0:dt:T16];

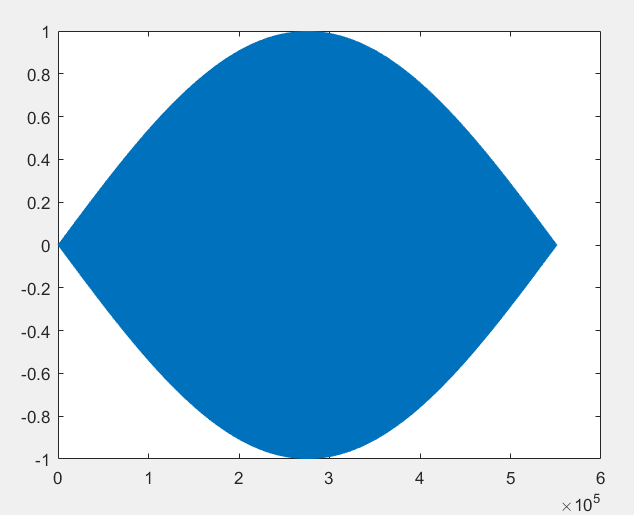
k = size(t16,2);

ScaleTable = [1 2^(1/6) 2^(1/3) 2^(5/12) 2^(7/12) 2^(9/12) 2^(11/12) 2 2^(7/6) 2^(4/3)]

t = linspace(0,100\*T16,100\*k);

mod = sin(pi\*t/t(end));

s = mod.\*cos(2\*pi\*ScaleTable(input)\*f0\*t);



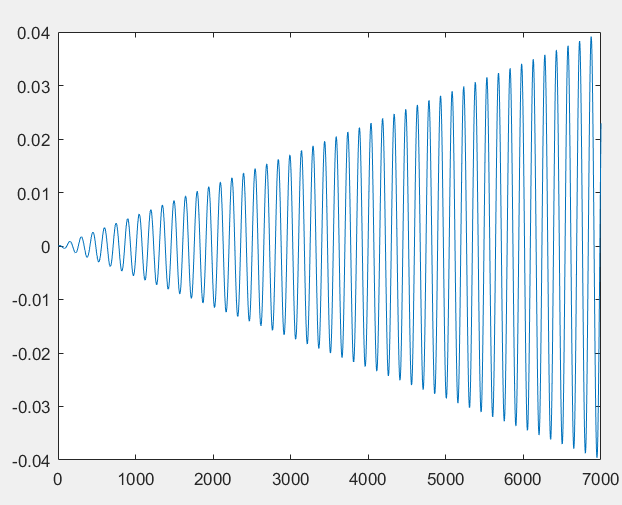


Figure 23 乐音包络与细节

4、GUI部分与程序联调：

我们使用了最新的MATLABR2019a的AppDesigner工具设计一个优美的可视化界面：

主界面：

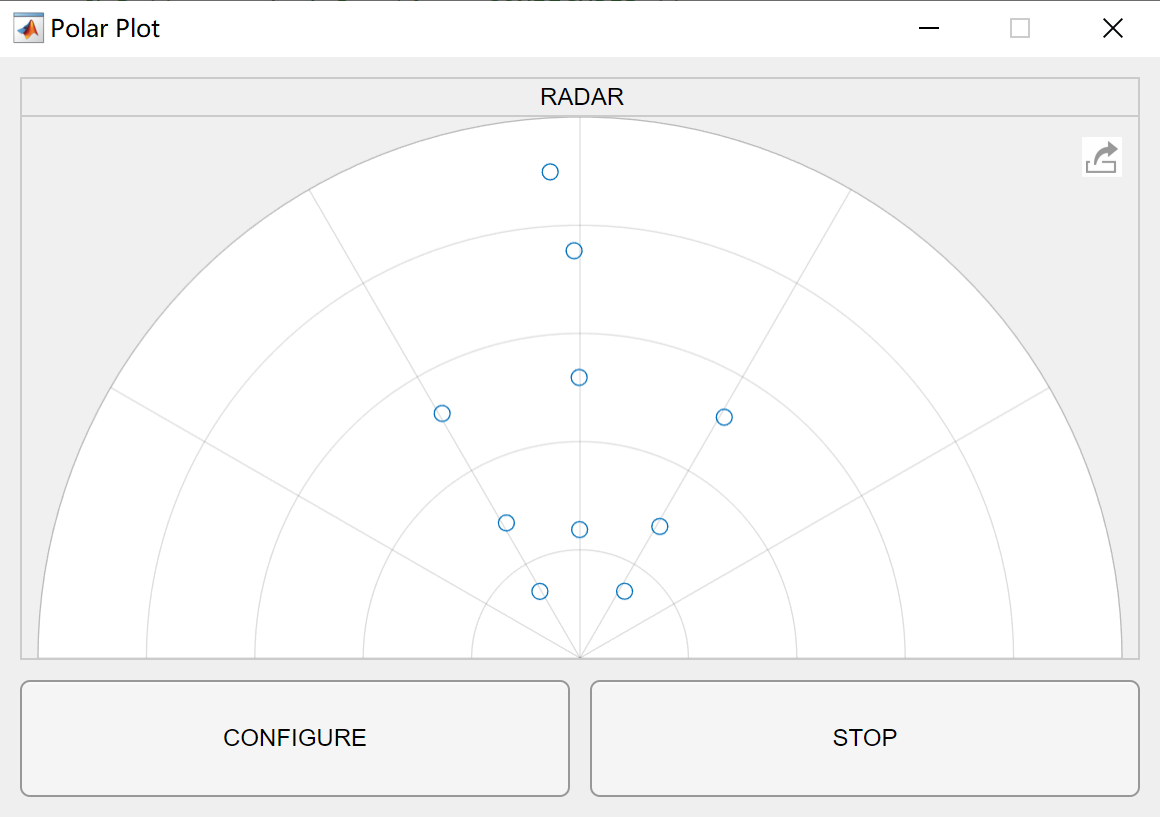


Figure 24 程序主界面

用户首先点击CONFIGURE按钮，配置自己的端口号，

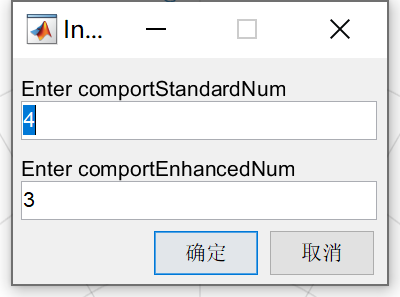


Figure 25 端口设置界面

确认后，CONFIGURE变成START按钮，点击即可开始，

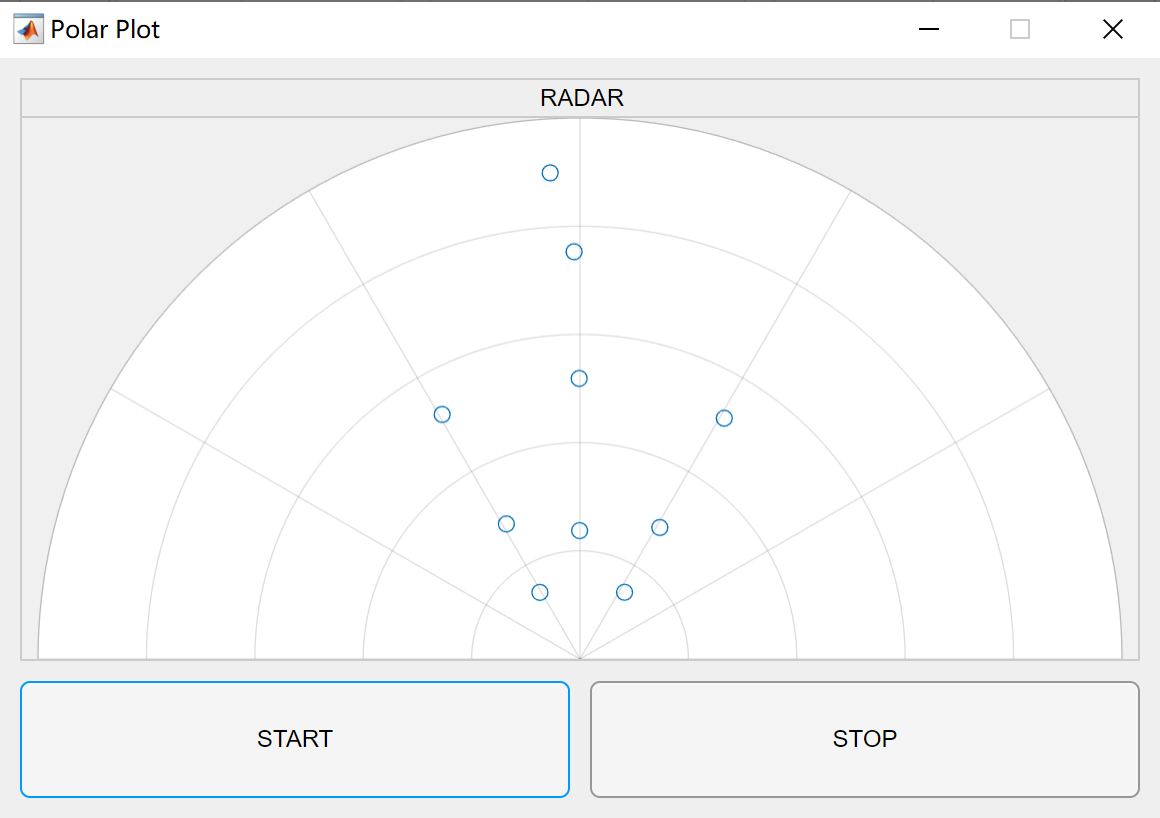


Figure 26 程序运行界面

上方的雷达图显示的是不同音的判定位置，将手放在毫米波雷达上，雷达图会再显示一个实心原点表示，并且程序进行判定并发出对应的声调的声音。

按STOP按钮停止雷达探测与声音产生。

实现机制：

应用MATLAB Parallel Computing Toolbox的并行计算工具，将雷达监测与声音产生放到了Worker之中，并且维护一个跨进程的DataQueue，并为其指定绘图回调函数，由此进行实现。

**四、实验结果展示**

**五、实验总结**

张咏镜：

在这次大作业中，我很幸运能将课堂中所学的知识应用到实践中。尽管距离、角度判断的部分并不是我完成的，但是前期阅读《毫米波雷达基础知识》时也综合了信号与系统和数字信号处理课程所学知识，体验是和上课不一样的。尤其是在阅读样例代码时，不明白的地方就向组员们询问，**从中我更深入地理解了fft和fftshift函数，以及时频变换在雷达信号处理中的重要性。**通过数学推导我们可以从信号中提取出很多有意义的数据，这或许就是数字信号处理所赋予我们的能力吧！

在亲自编写程序时，我主要负责让项目能实时跑起来，能一帧帧地检测手的位置。**除了信号处理技术之外，通信方式的解读和debug的过程也让我获益匪浅。**从一遍遍地排查问题过程中我发现，官方文档里有很多详细的解释，需要仔细阅读，SDK里也有很多我们作为使用者最初没有留意部分的说明。除了理解算法、应用算法之外，这个大作业也让我体验了作为电子工程师一定要阅读说明文档，SDK文档的重要性。

本项目中我尝试不修改固件，通过修改程序读取的方式来加快运行速度。我们的想法非常朴素，尝试了不同的方法最终都以失败告终，感觉非常可惜。不过debug的过程确实非常有趣，通过巧妙的断点、输出分析，我们发现断点执行就不会有丢包错误，而直接跑起来就会出现丢包。最有趣的现象是，程序中常出现丢包的错误，是每次循环到特定次数后就会发生丢包错误。这个问题难以判断其发生机理。最终我们在chirp数据还没有传送完毕时向雷达发送了sensorStart命令，才发现问题的来源：**通过精简程序运行时间，程序就会跑得快，而数据还没传完，程序又进入下一轮sensorStart，结果两组数据需要被同时传回，因此雷达报丢包错误。**

最后将自己的部分和发声部分，GUI多线程部分整合的过程中意识到，**程序的整体架构、统一的输入输出接口非常重要。**一个大工程中，自己的代码能在本地完美运行并不代表你的工作完成得差不多了，也有合在一起完全不能工作的可能。同样的，一个问题有很多种解决方案，如果不和他人合作项目，方案可以自己随便制定，但是若是小组项目，个人方案必须要顾全大局。小组任务并不轻松，但是这是一个体验，也是我们从个人项目向小组项目转型的一次机会。

这个大作业让我非常好得应用了数字信号处理的知识来了解距离、角度估计原理，同时也意识到做好一个任务，不仅仅是需要从自己的程序中找问题，也要从全局的dataPath中看问题、从硬件内部找问题。多参考资料，上网搜类似问题，仔细阅读说明文档，是当今时代解决问题的通用办法。非常感谢我们组员给我的帮助！有很多自己没有做好的地方也是他们的建议让我能继续完善和进步！

陈斯昂：

本次实验中遇到的几个主要困难有：

①雷达对于较大角度信号的分辨率较低，不管是CBF算法还是MVDR算法，对于角度较大的物体，检测成功率与稳定性都不是特别好，这在很大程度上限制了发生的音符数量，分析角度分辨能力受限的原因，可能有以下几个方面：接收天线数量较少，信息量不够；实验空间不够空旷，都是在室内进行实验，有比较大的反射回波干扰；算法选择上不够好，还有更好的选择；雷达发送信号集中在垂直方向，不利于大角度信号的检测（垂直方向上的噪声可能已经足以盖过边上的回波）

②数据传输延迟太大，无法做到即时检测和发生，由于雷达固件的设定限制，雷达无法进行连续收发，只能进行单次一定长度的检测，虽然每次雷达检测的时间不上，但是由于UART传输速度受限，雷达传输数据到电脑需要大约零点几秒的时间（采用不同profile有不同的文件长度，传输时间不同），这在很大程度上限制了检测频率，从而显示了整个程序的灵敏性，音符的频率也不能做的很高，也在一定程度上造成了程序调试的困难。我们也尝试了多种方法尝试提高检测频率（在第2部分中有所体现），但是都有着一定的问题，最后没能成功，确实相当的遗憾，如果时间充裕的话，我们也可以利用TI的mmWave sdk对雷达的固件进行调整以得到更好的效果（SDK已经下好了，东西也在研究，确实没能搞出来，很遗憾）。

虽然DSP课程、DSP的大作业这学期就结束了，但是对于上面这些尚未解决的问题，实际上还有很多值得探究和思考的地方，希望在日后的学习过程中，能够利用学到的新知识，解决这部分的问题，将自己在DSP课上学到的知识真正做到学以致用，这才是这次大作业最重要的意义。

廖宇轩：

做前端的同学听起来简单，实现起来却很繁琐。MATLAB并不是真正支持POSIX多线程的平台。所以将ParPool当做多进程的替代品的代价是，每次启动和关闭都需要等待一定时间，且对进程的操作十分不灵活，通信手段也十分有限。尤其是作为商业软件，MATLAB的文档非常不详细，导致其并行计算工具调用的函数的参数需要在最近上下文显式定义的变量这件事情根本没让用户知道，使我在这件事情上，本来只是一个定义新局部变量赋值拷贝的事情，研究了数个小时。而且MATLAB的 内部运行机制非常不明朗，Debug十分困难（我听书其实是JAVA的皮）。另一个难点是队友的代码并没有考虑被人回调或被丢进次进程等等问题（而要维持表面UI的运行和画布的更新这是必要的），不处理异常抛出或者仔细考虑调用接口，导致协调沟通乃至一起修改代码，也花费了不少力气。不过最后能够达到实验效果，也令人欣慰。我们发现code函数如果不Plot就会引起传输超时，这是我面临的最后问题：当时我并不知道这件事情，但是奇怪怎么老是超时，结果队友说，只要plot就不超时了。我想，明明是因为程序运行太快又没有sleep或者synchronize机制，速度与串口流不匹配，结果问题又落到了我这个最终整合完成的人身上。不过仍然感谢大家通力合作各尽其责，没有甩锅的现象，可谓十分和谐。

**六、参考文献与资料：**

[1] 来源于百度百科“特雷门琴”词条

[2] Project相关代码上传在github上：<https://github.com/HIEROT/DSPFinal>

[3] 来源于TI官方说明文档：Programming Chirp Parameters in TI Radar Devices

[4] 参考论文：MVDR自适应波束形成算法研究

[5] 参考论文：一种改进的 MVDR 相干信源 DOA 估计算法，唐孝国 , 张剑云，洪振清

[6] 来源于TI官方说明文档：mmwave\_sdk\_user\_guide

[7] 乐音窗函数的考虑来自于：<https://blog.csdn.net/stk10/article/details/8922621>