(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 110268278 B
(45)授权公告日 2023.04.11

(21)申请号 201780085837.2
(22)申请日 2017.12.21
(65)同一申请的已公布的文献号
申请公布号 CN 110268278 A
(43)申请公布日 2019.09.20
(30)优先权数据
3302-2016 2016.12.22 CL
(85)PCT国际申请进入国家阶段日
2019.08.07
(86)PCT国际申请的申请数据
PCT/CL2017/050081 2017.12.21
(87)PCT国际申请的公布数据
WO2018/112675 ES 2018.06.28
(73)专利权人 智利大学
地址 智利圣地亚哥

(72)发明人 R·芬格尔 D·莫纳斯特里奥 C·穆诺茨 R·丰特斯 N·雷耶斯 L·布朗夫曼 P·曼纳

(74)专利代理机构 中国贸促会专利商标事务所有限公司 11038
专利代理师 李玲
(51)Int.Cl.
G01S 3/72(2006.01)

(56)对比文件
US 2016214534 A1,2016.07.28
CN 102803984 A,2012.11.28
CN 101908936 A,2010.12.08
WO 2016007457 A1,2016.01.14
CN 105824021 A,2016.08.03

审查员 林平

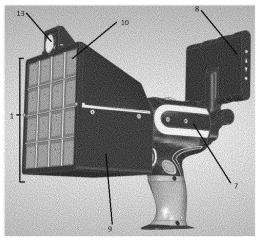
权利要求书1页 说明书38页 附图9页

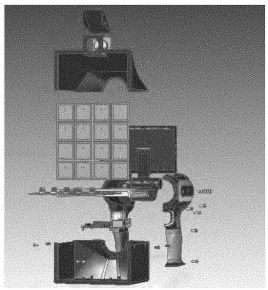
(54)发明名称

无线电视觉设备

(57)摘要

本专利涉及一种用于实时检测和定位由移动设备(蜂窝电话和Wi-Fi/蓝牙设备)使用的频谱中的电磁发射的紧凑且便携的系统。检测和定位的原理基于相控阵列技术,该技术使得能够合成在其形状和方向两个方面都能够被电学地控制的定向辐射波束。这种技术主要用于军事和天文应用。该设备还包括定位和控制算法。这个设备将允许借助于80×80度的视场内的天线波束扫描来检测和定位电磁发射。一旦已经建立了检测和定位,就将结果重叠到由视频相机捕获的视觉图像。





CN 110268278 B

1.一种用于检测电磁和可见发射的设备,其中所述设备包括扁平天线(10)阵列,这些天线被呈现为相控阵列(1),相控阵列(1)使用高速波束扫描来从被扫描的表面捕获射频(RF)信号,其中相控阵列(1)能够合成其形状和方向能够被电学地控制的定向辐射波束;其中每个天线能够改变幅度和相位并共同生成定向辐射波束;其中捕获的RF信号由高频电子设备(2)处理,高频电子设备(2)包括印刷电路板(PCB)或高频电子板:连接到天线的N×N移相器(3),N×N移相器(3)改变接收到的电磁波的相对相位,其中N×N移相器(3)中的每一个的输出被同相地组合(14)成合成波束的单个输出,这个信号被放大(11)并被滤波(12),从而限制和净化信号,到达总功率检测器(4),在总功率检测器(4)中RF信号的总功率被检测为模拟电压,所述模拟电压又借助于模数转换器(5)而被转换成数字信号;此外,在与相控阵列(1)相同的聚焦方向上存在产生视觉图像的视频相机(13);所述设备还包括微型计算机(6),微型计算机(6):第一,负责控制和同步相控阵列、移相器、放大器和滤波器以创建RF图像,对RF图像进行后处理以建立RF发射的位置;第二,负责将由相机获得的视觉图像与RF图像重叠并将它们显示在屏幕(8)上;以及第三,负责执行提供用户界面的算法;自主电源(7),为所有上述部件供能;以及包围所有部件的外壳(9),使得所述设备保持紧凑并且便携。

2.如权利要求1所述的用于检测电磁和可见发射的设备,其中天线(10)阵列包括被布局为矩阵的64至4个天线或辐射元件,这些天线或辐射元件还能够检测在移动设备的发射频率范围中或者在移动设备和/或Wifi设备的频率范围中的多个频带。

3.如权利要求2所述的用于检测电磁和可见发射的设备,其中所述天线(10)阵列包括被布局为矩阵的16个天线或辐射元件。

4.如权利要求2所述的用于检测电磁和可见发射的设备,其中所述多个频带包括700-900MHz、1700-2100MHz、2400-2700MHz和5000MHz频带。

5.如权利要求1至2中任一项所述的用于检测电磁和可见发射的设备,其中通过相控阵列(1)在任意方向上的辐射波束的合成被限定在90×90度。

6.如权利要求1所述的用于检测电磁和可见发射的设备,其中视频相机(13)是数字的并且示出要在其中检测RF信号的区域。

7.如权利要求1所述的用于检测电磁和可见发射的设备,其中微型计算机(6)发送控制信号,所述控制信号允许移位被包括在高频电子板(2)中的移相器(3)中的每一个的相位,其中微型计算机(6)还能够利用所述控制信号同步光学和射频图像的获取。

8.如权利要求1所述的用于检测电磁和可见发射的设备,其中微型计算机(6)在后处理期间使用多个高斯检测算法来建立RF发射的位置,所述多个高斯检测算法确定发射体的数量并定位RF图像中的电磁发射。

9.如权利要求1所述的用于检测电磁和可见发射的设备,其中将由视频相机捕获的视觉图像与RF检测重叠并且将标记放置在RF捕获位点处。

无线电视觉设备

技术领域

[0001] 这种设备的应用范围包括用于救援困在雪崩或倒塌建筑物中的人以及在不允许使用电源或移动设备的地方(诸如监狱或飞机)中检测和测量来自电源或移动设备的电磁发射的工业应用。另一方面,本发明涉及蜂窝电话频谱内的RF(射频)源的检测和定位以及图像形式的视觉表示的实时重叠。

背景技术

[0002] 现有技术提到参考不是保护特定物理系统而是保护允许特定类型操作的设备和方法的集合的若干专利。

[0003] 存在属于天线的一般领域的多个文献,诸如:EP 0 938 156、EP 0 144 867、US 2015/0022421、WO 2016/089464、WO 2011/148248。

[0004] 另一方面,存在与本发展的范围相关的多个专利,如下所述:

[0005] EP 0 887 879:这个专利落入本发展的范围,并旨在减小高增益相控布局(phased layout)的总尺寸。这是通过使用多层结构来实现的,在多层结构中移相器(phase shifter)耦合到天线,从而消除了路由到它们的需要。这个设备提供了用于相位布局的新形状。

[0006] EP 1 921 709:这个专利涉及能够生成没有光栅波瓣(grating lobe)的双波束(double beam)的系统。它还强调它是紧凑且完整的用于波束合成的模块。

[0007] US 7.602.337 B2(EP2100347):这个专利涉及发射阶段中的布局以及控制它的算法。这个设备将其创造性集中在其形状上,但也集中在其算法和软件上。

[0008] EP 2 463 958:这个专利涉及紧凑的多波束天线阵列,其优化封闭空间内的无线连接。该阵列本身就是创新,因为它优化了名为“Van-Atta”的旧设计。

[0009] US 6.504.510、EP1294163、US2015/0022421:这个专利涉及用于移动电信的相控布局。该布局被布置为M×N,但仅需要M+N个移相器用于其操作,因为它还使用选择性功率分配器。这使它能够“控制三个波束路径”(移动波束的方位角和仰角)。

[0010] US 6.693.590 B1、EP1195017:这个专利涉及使用ADC实现数字移相器的相控布局。为此,他们使用IF降低转换。然后信号被数字地组合。这对于大带宽是非常低效的。

[0011] PCT/FI2014/050067、EP29511884:这个专利涉及天线阵列与现有电信系统的集成,从而最小化资源(布线)和尺寸。它还优化了“信号数据”的收集。

[0012] WO 2008/082917:这个专利涉及用于优化或改善由相控布局合成的波束的形成的一系列方法和设备。

[0013] US 2009/0102716:这个专利涉及具有雷达和电信应用的系统,该系统使用波束来递送功率和控制数据。

[0014] US 2009/0027267:这个专利涉及可变相位反射器。想法是利用喇叭天线照射反射器,并通过反射器的相位控制来改变波束的方向。反射器由其中尤其包括变容二极管(varactor)的元件的阵列组成。

[0015] US 2016/0087349:这个专利涉及用于控制特定频带并且还有多频带的波束宽度的方法。该设计使用4x4原型天线,其中单频带和多频带天线被放置在阵列内的特定位置。

[0016] US2008/0106467:这个专利涉及完全安装在多层基板上的相控布局,其中所有对应的电子设备都被放置在后部。这都是紧凑的,并且设备的尺寸仅受基板的尺寸限制。这个系统在两个方向上都工作。

[0017] US 2015/0316641:这个专利与具有多个输出的雷达的发送器对应,其可以被同相控制。这个发送器集成在单个芯片中。它还具有RF发生器。

[0018] 现有技术中最接近本发展的设备是专利EP 2 246 934 A1。这个专利属于本发展的技术范围,并且能够通过使用在其上的RFID标签来检测和跟踪具有主动发射(active emission)的包裹移动(可移动的物体)的方向。它们使用2×2阵列,从而确保其紧凑的尺寸。其应用集中在使用RF标签的库存管理上。它需要外部计算机来进行操作。设备和相关联的方法受到保护。

[0019] 由本发明解决的技术问题

[0020] 如何基于射频和可见场(没有排除诸如近红外和远红外、近紫外和远紫外之类的其它场)的波长生成集成的实时图像。

[0021] 如何生成能够生成上述图像的手动、自支持和便携式的设备。

[0022] 如何被动地定位辐射发射的源。

[0023] 为了能够响应之前的技术问题,一般而言,本发展包括:

[0024] • 实时RF图像的生成。

[0025] • 使用高斯曲线检测算法来定位辐射源的方向。

[0026] • 使用光学视频图像来通过重叠识别最大无线电辐射的标记以定位视觉场中的辐射的源。

[0027] • 允许便携和自主操作的设备和方法的集合。

[0028] 本发明的一般描述

[0029] 本专利涉及一种用于实时检测和定位由移动设备(蜂窝电话和Wi-Fi/蓝牙设备)使用的频谱中的电磁发射的紧凑且便携的系统。检测和定位的原理基于相控阵列(phased array)技术,该技术使得能够合成在其形状和方向两个方面都能够被电学地控制的定向辐射波束,没有任何移动的机械部件。这些机制主要用于军事和天文应用。该设备还包括定位和控制算法。这个设备将允许通过在80×80度的视场(field of view)中扫描天线波束来检测和定位电磁发射。一旦检测和定位就绪,就将结果重叠在由视频相机捕获的视觉图像上。

[0030] 该设备是完全被动的,它在其操作中不发射处于检测频带的电磁辐射。

[0031] 应当理解的是,本发明不限于本文描述的特定方法、化合物、材料、制造技术、用途和应用,因为这些可以变化。还应当理解的是,本文使用的术语用于描述特定表示的惟一目的,并不意图限制本发明的观点和潜力。

[0032] 应当注意的是,这里在权利要求的列表中和贯穿整个文本的用途和方法,单数不排除复数,除非在上下文中明确暗示。因此,例如,对“用途或方法”的引用是对一个或多个用途或方法的引用,并且包括本主题(领域)中技术人员知道的等同物。类似地,作为另一个示例,对“步骤”、“阶段”或“模式”的引用是对一个或多个步骤、阶段或模式的引用,并且可

以包括隐含的和/或接着发生的子步骤、阶段或模式。

[0033] 使用的所有连词必须以其可能的最少限制性和最多包容性的意义来理解。因此，例如，连词“或”必须以其逻辑正统意义来理解，而不能理解为“排除式的或”，除非上下文或文本明确地需要或指示它。所描述的结构、材料和/或元件应理解为还指功能上等同的那些，以避免无休止的穷举列表。

[0034] 用于指示近似或概念的表述应如此理解，除非上下文命令不同的解释。

[0035] 本文使用的所有名称以及技术和/或科学术语均应具有由在这些内容方面具有资格的普通人给出的普遍含义，除非另有指示。

[0036] 描述了方法、技术、元素、化合物和组合物，但是在本发明的实践和/或测试中可以使用或优选与所述那些类似和/或等同的方法、技术、化合物和组合物。

[0037] 所有专利和其它出版物被包括作为参考，以用于描述和/或告知例如这些出版物中描述的与本发明结合可以是有用的方法。

[0038] 这些出版物被包括仅仅是因为它们的信息在本专利申请的注册日期之前。

[0039] 在这方面，不应当将任何内容视为承认或接受、拒绝或排除作者和/或发明人不被如此考虑或者这些出版物被他人或出于其它任何原因提前出版。

[0040] 为了使本发展清晰，以下概念应被定义如下：

[0041] 天线：能够发射或接收电磁辐射的元件。

[0042] 定向辐射：天线不总是在所有方向上发射或接收。存在几种类型的天线，其中有可能将辐射功率集中在有特权的(privileged)方向上。用于测量这个的参数被称为天线增益。

[0043] 相控阵列：空间地分布在平面上的天线阵列。通过移位(shift)每个天线的幅度和相位，可以朝着特定的方向合成直接辐射波束。

[0044] 高频电子设备：在微波频率工作的电子元件。这些元件不同于常规的电子元件，因为它们不再被视为集中参数元件，而是具有分布参数，即，它们之间的位置和距离影响它们的最终行为。

[0045] 射频信号：具有处于无线电和微波频谱的频率的信号。本发展在千兆赫信号上工作。

[0046] 高速波束扫描：由相控阵列执行的扫描，其在整个扫描表面上执行并且其中射频信号被捕获。这种扫描的一般等同物是阴极射线电视的操作。

[0047] 模拟电压信号：可以取其操作范围内的任何值的连续电压信号。但是，在数字信号中，只有用于表示0和1的两个电压电平。

[0048] 微带天线(microstrip antenna)：一种类型的微波天线，其可以安装在扁平(flat)表面上。这些天线易于被构建并放置在安装在地面上的由介电材料层隔开的扁平金属矩形上。

具体实施方式

[0049] 该设备与基于相控阵列技术的操作原理的检测和定位系统对应。这种技术允许通过多个非定向辐射元件(天线)合成定向辐射波束。这种辐射波束能够接收和发射电磁辐射。合成波束的宽度由单独辐射元件之间存在的间隔来定义。可以通过移位每个单独元件

的相位来改变波束方向。对于线性天线阵列,阵列的相邻元件中的每一个之间的相移由等式1给出,其中d是元件之间的距离,λ是电磁信号的波长,并且θ_0是波束旋转的角度。波束方向的改变在图1/10中示出。

[0050] $$\Delta\phi = \frac{2\pi}{\lambda} \mathrm{d} \sin(\theta_0)$$

等式 1

[0051] 如果天线以二维阵列布局(不限制其它类型的布局),那么可以在两个维度上控制波束方向。现在,波束方向的改变将由两个角度控制,并且其相移由两个轴上的重叠给出。

[0052] 这种应用可以使用广泛范围的天线阵列(不排除任何一个)。优选的范围是64个天线至最少4个天线,优选地是16个天线。4个天线的阵列允许二维图像。一般而言,元件数量的变化将影响合成波束的质量,更大数量的元件产生波束的横向波瓣的减小,并且它们之间的间隔影响所述波束的宽度。

[0053] 为了执行检测,波束对区域进行高速扫描。这允许创建电磁发射(优选地在移动设备和Wifi(无线保真)的范围内,具体地是在700-900MHz、1700-2100MHz、2400-2700MHz和5000MHz频带中)的图像,然后将所述图像与通过视频相机获得的数字视觉图像重叠。意图用于本发明的最大视场是90×90度。

[0054] 鉴于技术的本质以及待检测和定位的源的功率强度的不确定性,测量距被发射的物体的距离是不可能的,因此只可以在两个维度上瞬时执行检测,这意味着检测在XY平面中的2D图像上完成,其中Z是深度。虽然有可能估计距源的最大距离,但是不能进行绝对距离测量。

[0055] 图2/10中示出的图示出了设备的不同部件,如下所述:

[0056] (1)扁平天线阵列:天线阵列与64个和4个单独辐射元件之间的范围对应。这些元件能够接收所有待检测频带的电磁发射(多频带天线,类似于蜂窝天线)。这些元件以不同的形状定位,以便改善合成波束的质量。等距元件的规则分布是优选的,尤其是诸如矩阵形状、圆形形状,如图3/10中所示。天线阵列足够紧凑以作为便携式设备的一部分。

[0057] (2)高频电子设备:高频电子器件被集成到一系列PCB(印刷电路板)中。图4/10示出了被包括在设备中的电子元件。

[0058] 以下是每组元件的列表:

[0059] 数量等于天线阵列的移相器(3)连接到天线阵列(1)中的辐射元件中的每一个。它们负责改变接收到的电磁波的相对相位。

[0060] 移相器(3)中的每一个的输出被同相地组合(14)以生成具有合成波束的信息的单个输出。然后利用低噪声放大器(11)模拟地放大信号,并且可选地利用用于那个频带的滤波器(12)模拟地对信号进行滤波,以到达总功率检测器(4),总功率检测器(4)将射频信号转换成直流(DC)信号。后者借助于A/DC(5)(模数转换器)转换成数字信号。

[0061] 视频相机(13)捕获其中示出检测到的电磁辐射的源的光学图像。相机优选地是分辨率高于640×480像素的数字相机。

[0062] 视频相机可以定位得比天线高,它可以被放置得更低或放置在天线之间的中心处,其中,位于天线阵列的相同对称轴上简化了将无线电图像重叠在光学图像上的算法。

[0063] (6)微型计算机:履行多个重要任务,第一个任务是阵列(1)的控制和同步,向移相器(3)中的每一个发送电信号并且因此合成期望方向上的辐射波束。微型计算机(6)负责接收和处理来自检测器(4)的信号,从而构建RF图像。随后对这个图像进行后处理,以找到射频发射的位置。后处理算法包括高斯检测器。然后将检测与来自视频相机的图像重叠并显示在屏幕上。微型计算机(6)还负责执行提供用户界面的程序。

[0064] (7)电源:允许系统保持自主,为射频电子设备、视频相机和微型计算机及其屏幕(8)供能。

[0065] (9)系统保护外壳:整个系统以类似壳的格式包裹,其包含所有上述部件。

[0066] 上述算法包含以下源代码:

[0067]
```
from rv.core import Core
if __name__ == '__main__':
```

```
    logging.basicConfig(filename='radiovision.log', format='%(asctime)s:%(levelname)s:%(message)s' ,level=logging.INFO,
datefmt='%m/%d/%Y %I:%M:%S %p')
    Core().stuff()
"""
This class initializae the GUI and controls the aqusition of frames
"""
class Core(object):
    def __init__(self):
        self.cam = camera.Camera().start()
        # Load markers images
        import os # this allows to lunch the project from any where
        current_path = os.path.dirname(__file__) # and access the data
        self.marker_1 = imread(current_path + '/markers/marker_1.png', -1)
        self.marker_2 = imread(current_path + '/markers/marker_2.png', -1)
        self.marker_3 = imread(current_path + '/markers/marker_3.png', -1)
        self.markers = [self.marker_1, self.marker_2, self.marker_3]
        # First frame
        # imshow('RadioVision', self.cam.read())
        # sync elements
        # self.marker_position = Array('i',2*number_of_gaussians_to_fit)
        self.marker_queue = JoinableQueue(1)
        self.rf_scan_queue = JoinableQueue(10)
[0068]  self.new_acc_len = Value('i',10)
        self.rf_scan = rv.array.Array(31, 31, self.rf_scan_queue, self.new_acc_len)
        self.marker_fit = rv.detection.detect.Detect(31, 31, self.marker_queue, self.rf_scan_queue)
        # minimun amplitud value
        self.amp_error =1e-6
        # capture pictures loop
        self.capture = True
        self.video_thread = threading.Thread(target=self.start_all)
        self.add_pip = False
        # GUI Stuff
        self._job = None
        self.root = tkinter.Tk()
        self.root.geometry("400x225")
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.button_frame = tkinter.Frame(self.root, width=100)
        self.button_frame.pack_propagate(0)
        self.button_frame.pack(fill='y', side=tkinter.LEFT)
        self.video_frame = tkinter.Frame(self.root)
        self.video_frame.pack(fill=tkinter.BOTH, side=tkinter.LEFT)
        self.video_display = tkinter.Label(self.video_frame)#, height=225, width=300)
        self.video_display.pack(fill=tkinter.BOTH, side=tkinter.LEFT)
        self.button_pip = tkinter.Button(self.button_frame, text='PIP OFF', command=self.button_pip_action_on, height=3)
        self.button_pip.pack(fill='x')
```

```
        self.button_acc_reset = tkinter.Button(self.button_frame, text='Reset ACC',height=3,command=self.reset_acc)
        self.button_acc_reset.pack(fill='x')
        tkinter.Label(self.button_frame,text="Acc Size").pack()
        self.acc_length = tkinter.Scale(self.button_frame, from_=1, to=100, orient=tkinter.HORIZONTAL, command=self.updateValue)
        self.acc_length.pack(fill='x')
        self.acc_length.set(10)
        self.root.bind("<<close_me>>",self.last_close)
        self.picture_scale_h_w = 640/480.0
        self.close_mutex = threading.Lock()
        self.close_cond = threading.Condition()
        self.main_loop_closed = False
    def get_picture_size(self):
        width_0 = self.root.winfo_width() - self.button_acc_reset.winfo_width()
        heigth_0 = width_0 / self.picture_scale_h_w
        heigth_1 = self.root.winfo_height()
        width_1 = heigth_1*self.picture_scale_h_w
        # print('s',width_0, heigth_0)
        # print('e',width_1, heigth_1)
        if heigth_0 <= heigth_1:
            return int(width_0), int(heigth_0)
        return int(width_1), int(heigth_1)
    def reset_acc(self):
[0069]      logging.debug('reset')
        self.rf_scan.new_acc()
    def updateValue(self, event):
        if self._job:
            self.root.after_cancel(self._job)
        self._job = self.root.after(500, self._do_something)
    def _do_something(self):
        self._job = None
        logging.debug("new value:" + str(self.acc_length.get()))
        self.new_acc_len.value = self.acc_length.get()
        self.rf_scan.new_acc_len()
    def button_pip_action_on(self):
        self.add_pip = True
        self.button_pip.configure(text='PIP ON')
        self.button_pip.configure(command=self.button_pip_action_off)
    def button_pip_action_off(self):
        self.add_pip = False
        self.button_pip.configure(text='PIP OFF')
        self.button_pip.configure(command=self.button_pip_action_on)
    def on_closing(self):
        self.capture = False
    def stuff(self):
        self.video_thread.start()
```

```
        self.root.mainloop()
    def start_all(self):
        # start array data and fit
        self.rf_scan.start()
        # self.marker_fit.activate_plot()
        self.marker_fit.start()
        # the rest of the programm
        # initial values for the positions of the markers
        markers_coords =    None
        update_counter = 0
        display_color_map = None
        import cv2
        picture_wide = 300
        picture_heigth = int(picture_wide * self.picture_scale_h_w)
        time.sleep(1)
        while self.capture:
            frame = self.cam.read()
            # draw a rectangle over the image
            drawrect(frame, (100, 20), (540, 460), (0,     0,     0), 3, style='else')
            drawrect(frame, (100, 20), (540, 460), (0, 255, 255), 1, style='else')
            # if queue has task
            while not self.marker_queue.empty():
```

[0070]

```
                [markers, data] = numpy.array( self.marker_queue.get())
                self.marker_queue.task_done()
                # extend the data
                y = numpy.arange(0, 31, 1)
                x = numpy.arange(0, 31, 1)
                f = interpolate.interp2d(x, y, data, kind='cubic')
                display_color_map = f(numpy.arange(0, 30, 0.3),numpy.arange(0, 30, 0.3))
                # Update markers
                markers_coords = stretch_coords(markers, self.markers)
            if markers_coords is not None and len(markers_coords) > 0:
                self.paint_markers(frame, markers_coords ,markers[2])
            if display_color_map is not None and self.add_pip:
                frame = self.add_picture_in_picture(frame, display_color_map)
            frame_aux = imutils.resize(frame, width=picture_wide, height=picture_heigth)
            picture_wide,picture_heigth = self.get_picture_size()
            image = cv2.cvtColor(frame_aux, cv2.COLOR_BGR2RGB)

            image = Image.fromarray(image)
            try:
                image = ImageTk.PhotoImage(image)
            except RuntimeError as e:
                break
            self.video_display.configure(image=image)
```

```
                self.video_display.image = image
                # if update_counter > 1000:
                #     print('none')
                #     markers_coords = None
                #     update_counter = 0
                #     display_color_map = None
                # update_counter += 1
            logging.info('core closing')
            self.closing()
        def add_picture_in_picture(self, frame, data):
            # convert the data to uint scale from 0 to 255
            # as if it was a gray scale image
            max_value = numpy.max(data)
            data /= max_value
            # this is have a better distribution of colors
            data *= 255
            data = data.astype(numpy.uint8)
            # generate the color image
            # it must be uint8 data type to work
            data_colormap = cv2.applyColorMap(data, cv2.COLORMAP_JET)
            [frame_x, frame_y, a] = frame.shape
            [pip_x, pip_y, a] = data_colormap.shape
[0071]      frame[frame_x - pip_x:frame_x,frame_y-pip_y:frame_y,:] = data_colormap
            return frame
        def closing(self):
            logging.info('Closing objects...')
            self.cam.stop()
            self.rf_scan.shutdown()
            self.marker_fit.shutdown()
            time.sleep(5)
            while self.cam.thread_update.isAlive():
                pass
            self.root.event_generate("<<close_me>>", when="tail")
            logging.info('** Camera Closed **')
        def last_close(self, *args):
            self.root.destroy()
        def paint_markers(self, frame, marker_cords, markers_amp):
            # print('detected amplitudes: ', markers_amp)
            max_amp = max(markers_amp)
            max_amplitud_ration = 0.1
            font = cv2.FONT_HERSHEY_PLAIN
            constante= 1.0
            distance_funtion = lambda power: str(power) + 'APU'
            draw_distance = lambda _amplitud, Mx, My: cv2.putText(frame, distance_funtion(_amplitud), (Mx, My), font, 1, (0, 255, 255), 1)
            draw_distance_back = lambda _amplitud, Mx, My: cv2.putText(frame, distance_funtion(_amplitud), (Mx, My), font, 1, (0, 0, 0), 2)
```

```
                ampl_cont = 0
                for M1y,M1x in marker_cords:
                    for c in range(0, 3):
                        # paint the markers
                        if markers_amp[0] < max_amplitud_ration*max_amp:
                            pass
                        else:
                            try:
                                frame[M1y:M1y + self.markers[2].shape[0], M1x:M1x + self.markers[2].shape[1], c] = \
                                    self.markers[2][:, :, c] * (self.markers[2][:, :, 3] / 255.0) + frame[M1y:M1y + self.markers[2].shape[0],
                                    M1x:M1x + self.markers[2].shape[1], c] * (1.0 - self.markers[2][:, :, 3] / 255.0)
draw_distance_back(markers_amp[ampl_cont],M1x,int(M1y+self.markers[2].shape[0]/3))
draw_distance(markers_amp[ampl_cont],M1x,int(M1y+self.markers[2].shape[0]/3))
                            except Exception as e:
                                pass #print('marker 1 error' + str(e))
                    ampl_cont += 1
class Camera:
    def __init__(self, src=0):
        # USB camera handler
        self.stream = cv2.VideoCapture(src)  # Don't leave VideoCapture argument empty
        self.grabbed, self.frame = self.stream.read()  # Read first frame
        # Data flow control variables
[0072]  self.stopped = False
        # Thread handler
        self.thread_update = threading.Thread(target=self.update, args=())
        # self.thread_display = threading.Thread(target=self.display, args=())  # Commented due to weird behavior
    def start(self):
        self.thread_update.start()
        # self.thread_display.start()
        return self
    def update(self):
        while self.stream.isOpened():
            if self.stopped:
                return
            self.grabbed, self.frame = self.stream.read()
        self.stream.release()
    def display(self):  # Currently not in use
        cv2.imshow('RadioVision', self.frame)
        while True:
            if self.stopped:
                return
            cv2.imshow('RadioVision', self.frame)
    def read(self):
        return self.frame
    def stop(self):
```

```
            self.stopped = True
    class Array(Process):
        def __init__(self, size_x,size_y, queue, acc_len):
            super(Array, self).__init__()
            x = numpy.linspace(0, size_x-1, size_x)
            y = numpy.linspace(0, size_y-1, size_y)
            self.domain = numpy.meshgrid(x, y)
            self.Power = [] # the vector version of the measuremente
            self.Z = [] # the matrix version of the measuremente
            # sincronize elements
            self.queue = queue # pass the markers to the other process
            self.stop_event = Event()
            self.new_acc_event = Event()
            self.new_acc_len_event = Event()
            self.MY_SOCK_PATH = "/home/pi/comunication"
            self.sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
            self.max_hold = numpy.zeros((size_x,size_y))
            self.number_acc = 10
            self.current_acc = 0
            self.new_acc_index = 0
            self.acc_len = acc_len
            self.max_hold = numpy.zeros((self.number_acc, size_x, size_y))
```

[0073]

```
            try:
                self.sock.connect(self.MY_SOCK_PATH )
            except socket.error as msg:
                sys.stderr.write("bad conection")
                # sys.exit(1)
        def change_acc_length(self, new_acc_length):
            if new_acc_length > 0:
                self.number_acc = new_acc_length
        def reset_acc(self):
            a,b,c = self.max_hold.shape
            self.max_hold = numpy.zeros((self.number_acc, b, c))
            self.new_acc_event.clear()
        def change_acc_len(self):
            # print(self.max_hold.shape, self.new_acc_index)
            length, b, c = self.max_hold.shape
            new_a = self.acc_len.value
            new_max_hold = numpy.zeros((new_a, b, c))
            if self.number_acc >= new_a:
                back_index = self.new_acc_index - new_a
                if back_index < 0:
                    tmp = back_index
                    back_index = 0
                    new_max_hold[0:abs(tmp),:,:] = self.max_hold[length + tmp:length,:,:]
```

```
                    new_max_hold[abs(tmp):new_a,:,:] = self.max_hold[back_index:self.new_acc_index,:,:]
                else:
                    new_max_hold[0:new_a, :, :] = self.max_hold[back_index:self.new_acc_index, :, :]
                self.new_acc_index = 0
            else:
                new_max_hold[0:(length - self.new_acc_index),:,:] = self.max_hold[self.new_acc_index:length,:,:]
                new_max_hold[(length - self.new_acc_index):length, :, :] = self.max_hold[0:self.new_acc_index, :, :]
                self.new_acc_index = length
            self.max_hold = new_max_hold.copy()
            self.number_acc = new_a
            # print(self.max_hold.shape)
            # print(self.new_acc_index)
            self.new_acc_len_event.clear()
        def get_scann(self):
            self.sock.sendall(b'scann\n')
            n = 0
            recieve_data = b''
            while n < 31*31*8:
                recieve_data_aux = self.sock.recv(31*31*8)
                recieve_data += recieve_data_aux
                n += len(recieve_data_aux)
            recieve_data = unpack('<' + str(31*31) + 'd', recieve_data)
[0074]      recieve_data = array(recieve_data)
            return numpy.reshape(recieve_data,(31,31))
        def sweep_once(self):
            self.logging_info( 'New measure...')
            t = time.time()
            magnitudeMat = self.get_scann()
            tf = time.time()
            self.logging_info('Elapsed : ' + str(tf - t) + ' seconds')
            # magnitudeMat = 10**(magnitudeMat/10.0)
            self.max_hold[self.new_acc_index,:,:] = magnitudeMat
            self.new_acc_index = (self.new_acc_index + 1) % self.number_acc
            self.current_acc += 1
            self.Power = numpy.max(self.max_hold, axis=0)
        def shutdown(self):
            self.stop_event.set()
        def new_acc(self):
            self.new_acc_event.set()
        def new_acc_len(self):
            self.new_acc_len_event.set()
        def logging_info(self, msg):
            logging.info('[Array] ' + msg)
        def run(self):
            self.logging_info('Start array.')
```

```
            while 1:
                if self.stop_event.is_set():
                    self.logging_info('Stopping array.')
                    return
                # do one scann
                if self.new_acc_event.is_set():
                    self.reset_acc()
                if self.new_acc_len_event.is_set():
                    self.change_acc_len()
                self.sweep_once()
                try:
                    # pass the scan to fit gaussians
                    self.queue.put(self.Power,block=True,timeout=0.5)
                except Full:
                    # the queue is full
                    self.logging_info('Scan drop.')
def drawline(img, pt1, pt2, color, thickness=1, style='dotted', gap=20):
    dist = ((pt1[0] - pt2[0]) ** 2 + (pt1[1] - pt2[1]) ** 2) ** .5
    pts = []
    for i in np.arange(0, dist, gap):
        r = i / dist
        x = int((pt1[0] * (1 - r) + pt2[0] * r) + .5)
[0075]  y = int((pt1[1] * (1 - r) + pt2[1] * r) + .5)
        p = (x, y)
        pts.append(p)
    if style == 'dotted':
        for p in pts:
            cv2.circle(img, p, thickness, color, -1)
    else:
        e = pts[0]
        i = 0
        for p in pts:
            s = e
            e = p
            if i % 2 == 1:
                cv2.line(img, s, e, color, thickness)
            i += 1
def drawpoly(img, pts, color, thickness=1, style='dotted', ):
    e = pts[0]
    pts.append(pts.pop(0))
    for p in pts:
        s = e
        e = p
        drawline(img, s, e, color, thickness, style)
def drawrect(img, pt1, pt2, color, thickness=1, style='dotted'):
```

```
        pts = [pt1, (pt2[0], pt1[1]), pt2, (pt1[0], pt2[1])]
        drawpoly(img, pts, color, thickness, style)
    def stretch_coords(fitted_coords, punteros):
        """ This functions handles the coordinate
        transmfrmation to fit the image"""
        new_coordinates = []
        # return new_coordinates
        # px = [-0.00000970378811294236, 0.00208172113150949, -0.153201441654720, 8.43103861597864, 118.848944819314]
        # py = [-0.0000442788139443627, 0.00320074167604714, 0.104246823935828, -6.04649560143571, 199.899107062199]
        # ypix[0] = np.polyval(py, fitted_coords[1][0]*3)
        # ypix[1] = np.polyval(py, fitted_coords[1][1]*3)
        # ypix[2] = np.polyval(py, fitted_coords[1][2]*3)
        # xpix[0] = np.polyval(px, fitted_coords[0][0]*3)
        # xpix[1] = np.polyval(px, fitted_coords[0][1]*3)
        # xpix[2] = np.polyval(px, fitted_coords[0][2]*3)
        x_offset = 20
        x_scale_factor = int(440/31)
        y_offset = 100
        y_scale_factor = int(440 / 31)
        puntero1 = punteros[0]
        for cont in range(len(fitted_coords[0])):
            xp = int(fitted_coords[0][cont] * x_scale_factor + x_offset)
[0076]      xp -= int(puntero1.shape[0] / 2)
            yp = int(fitted_coords[1][cont] * y_scale_factor + y_offset)
            yp -=int(puntero1.shape[1] / 2)
            new_coordinates.append((xp,yp))
        return new_coordinates
    class DetectGaussias(object):
        def __init__(self, number_sample_points=150, alfa = 0.05):
            """This class detects an unknow number of gaussians.
            Keyword arguments:
            number_sample_points -- number of samplings to get form the data (default 200)
            """
            self.sample_data = ResampleData()
            self.get_covar_mean = EstimateMeanCovariance(alfa)
            self.number_sample_points = number_sample_points
            self.density_threshold = 0.6 # 0.05and aporiximation considering 20 points in a circunsfere wiht a radius of 20 with no pi
            self.minimum_area = 7
            self.alfa = alfa
        def sample_from_data(self, data):
            sample_index = numpy.zeros([self.number_sample_points, 2])
            for cont in range(self.number_sample_points):
                indy, indx = self.sample_data.sample_data(data)
                sample_index[cont, 0] = indy
                sample_index[cont, 1] = indx
```

```
                cont += 1
                if cont == self.number_sample_points:
                    break
        return sample_index
    def get_bic(self, cluster, means=[], covars=[]):
        """ This functions calculates the bayesian informatio criterion
        for a set of set of points
        cluster -- set of points
        """
        (d,) = cluster[0].shape
        mean = numpy.zeros((d,))
        total = 0
        # for cluster in clusters:
        for point in cluster:
            mean += point
            total += 1
        mean /= total
        covar = numpy.zeros([d,d])
        # for cluster in clusters:
        for point in cluster:
            val = point - mean
            covar += numpy.dot(val[None,:].T,val[None,:])
```

[0077]

```
        covar /= total
        u,s,v = numpy.linalg.svd(covar)
        area = 1
        chi_square_alfa = chi2._ppf(1 - 0.05, 2)
        for cont in range(d):
            area *= s[cont] * chi_square_alfa
        area = sqrt(area)
        # print(area, 'area')
        density = total / area
        # print(density, 'density')
        if density >=    self.density_threshold:
            means.append(mean)
            covars.append(covar)
            return True
        return False
    def detect_number_of_clusters(self, data_set):#, m_near = 18):
        """ this functiona applays a preprocessing of the data
            to try to estimate the number of clusters available.
            Keyword arguments:
            data_set -- a two dimentional array that contains the coordinates of the clusters
            m_near -- the number of neiborhood that are considered to rank the ponts (default 15)
            alfa -- the confidence interval, (1 - alfa) * 100 % (default 0.05)
        """
```

[0078]

```
alfa = self.alfa
reamaining_points = data_set.copy()
detected_clusters = []
cont_fig = 3
plot_all = False
m_near = 10
while 1:
    points_size, dimention = reamaining_points.shape
    real_near = m_near if m_near < points_size else points_size - 1
    near_distance = numpy.ones([points_size, real_near]) * 100
    near_distance_index = numpy.zeros([points_size, real_near])
    if real_near < 2:
        logging.warning('posible_problem')
    # Estimate the ranks from the points
    cont_main = 0
    for y_p, x_p in reamaining_points:
        cont_aux = 0
        for y_aux, x_aux in reamaining_points:
            # print(cont_aux)
            if cont_main == cont_aux:
                cont_aux += 1
                continue
            distance = sqrt((y_p - y_aux) ** 2 + (x_p - x_aux) ** 2)
            for cont in range(int(real_near)):
                if near_distance[cont_main, cont] > distance:
                    if cont != real_near - 1:
                        # print('in_me')
                        near_distance[cont_main, cont + 1:real_near] = near_distance[cont_main, cont:real_near - 1]
                        near_distance_index[cont_main, cont + 1:real_near] = near_distance_index[cont_main, cont:real_near -
1]
                    # else :
                    #     print('to long')
                    near_distance[cont_main, cont] = distance
                    near_distance_index[cont_main, cont] = cont_aux
                    break
            cont_aux += 1
        cont_main += 1
    ranks = numpy.sum(near_distance, axis=1)
    weights = 1 / ranks
    weights /= numpy.sum(weights)
    uniform_dist_value = 1 / (1.0 * weights.size)
    # sample one point to star the cluster detection
    x_i = self.sample_data.sample_data_gen_1d(weights, uniform_dist_value)
    total = numpy.sum(data_set)
    dist_data = data_set / total
```

```
                # add the point itself
                index_to_consider = numpy.concatenate([ near_distance_index[x_i], numpy.array([x_i,])])
                covar, mean = self.get_covar_mean.estimate_mean_variance(reamaining_points, index_to_consider)
                new_cluster_points_index = []
                chi_p_alpha = chi2._ppf(1 - alfa, 2)
                alfa_confidence_covar = chi_p_alpha * covar
                for cont in range(points_size):
                    choosen_point = reamaining_points[cont, :].copy()
                    choosen_point -= mean
                    # check if point is inside hyper-cube
                    points_ = choosen_point[:,None]
                    evalueate_ellipsoid = numpy.dot(numpy.dot(points_.T, numpy.linalg.inv(alfa_confidence_covar)), points_)
                    if evalueate_ellipsoid <= 1:
                        new_cluster_points_index.append(cont)
                # remove the points from the cluster
                # invert array to avoid having problems with the change in size and
                # new indexing of the array
                new_cluster_points_index.sort(reverse=True)
                new_cluster_points = []
                for index in new_cluster_points_index:
                    # delete one point at a time
                    new_cluster_points.append(reamaining_points[index, :])
[0079]              reamaining_points = numpy.concatenate((reamaining_points[0:index, :], reamaining_points[index+1:, :]))
                points_size, dimention = reamaining_points.shape
                # self.plot_points(reamaining_points, new_cluster_points, cont_fig)
                cont_fig += 1
                detected_clusters.append(new_cluster_points)
                # if the are enought point do one more iteration
                # print(points_size)
                if points_size <= m_near :
                    break
            cluster_amount = 0
            means = []
            covars = []
            for cont in range(len(detected_clusters)):#range(len(detected_clusters), 1, -1):
                if self.get_bic(detected_clusters[cont], means, covars):
                    cluster_amount += 1
                    # print('yes')
            return self.fusse_near_clusters(means, covars)
        def make_markers(self, mean, data):
            markers = [[],[],[]]
            for x,y in mean:
                markers[0].append(x)
                markers[1].append(y)
                markers[2].append(data[int(x),int(y)])
```

```
        return markers
    def plot_points(self, sample_index, delated, cont):
        points = numpy.zeros((31, 31))
        for indy, indx in sample_index:
            points[int(indy), int(indx)] = 1
        for indy, indx in delated:
            points[int(indy), int(indx)] = 0.5
        pyplot.figure(cont)
        pyplot.cla()
        pyplot.clf()
        pyplot.imshow(points)
        pyplot.draw()
        # pyplot.show(block=False)
    def fusse_near_clusters(self, means, covars):
        NN = len(means)
        mm = means.copy()
        meged = []
        final_means = []
        # print('in')
        cont = 0
        while cont < NN:
            if cont in meged:
```

[0080]

```
                cont += 1
                continue
            # print('first thing, mean')
            # print(means[cont])
            merge_count = 1
            mean = means[cont].copy()
            for cont1 in range(cont + 1,NN):
                if cont1 in meged:
                    continue
                covar_sum = covars[cont] + covars[cont1]
                distance = means[cont] - means[cont1]
                distance = numpy.dot(distance[:,None].T,distance[:,None])
                sum_distribution = multivariate_normal(means[cont], covar_sum)
                evala = sum_distribution.pdf(means[cont1])
                # print('gaussian dist',evala)
                # print(means[cont1])
                #
                # print('distance', distance)
                if distance <= 40:
                    # print('merge')
                    mean += means[cont].copy()
                    merge_count += 1
                    meged.append(cont1)
```

```
                    final_means.append(mean/merge_count)
                    cont += 1
            # print('out')
            return final_means
class Detect(Process):
    def __init__(self, size_x, size_y, queue_markers, queue_scan, number_of_gauss=3):
        super(Detect, self).__init__()
        # Data & Domain arrays needed for performing the fit
        self.domain = meshgrid(linspace(0, size_x-1, size_x), linspace(0, size_y-1, size_y))
        self.data = zeros((size_x*size_y, 0))
        # Outputs
        self.markers = [zeros((number_of_gauss, 1)), zeros((number_of_gauss, 1)), zeros((number_of_gauss, 1))]
        self.time_elapsed = 0.
        self.queue_scan = queue_scan
        self.queue_markers = queue_markers
        self.stop_event = Event()
        self.display_rf_scan = False
    def activate_plot(self):
        self.display_rf_scan = True
    def deactivate_plot(self):
        self.display_rf_scan = False
    # def fit(self, data):
[0081]  #       it = time.time()
    #       # print('fit')
    #       [x1, x2, x3, y1, y2, y3, a1, a2, a3, s1, s2, s3] = Fit(data, self.domain)
    #       tf = time.time()
    #
    #       Amps = array([a1, a2, a3])
    #
    #       Xpos = array([(x1), (x2), (x3)])
    #       Ypos = array([(y1), (y2), (y3)])
    #       orden = Amps.argsort()
    #
    #       self.markers[0] = [Xpos[orden[2]], Xpos[orden[1]], Xpos[orden[0]]]
    #       self.markers[1] = [Ypos[orden[2]], Ypos[orden[1]], Ypos[orden[0]]]
    #       self.markers[2] = [Amps[orden[2]], Amps[orden[1]], Amps[orden[0]]]
    #
    #       self.time_elapsed = tf - it
    #
    #       return self.markers.copy()
    def log_function_info(self, msg):
        logging.info('[Detect] ' + msg)
    def run(self):
        self.log_function_info('Start detect.')
        from matplotlib import use
```

```
use('TkAgg')
from matplotlib import pyplot
pyplot.ion()
detect = DetectGaussias()
while 1:
    if self.stop_event.is_set():
        self.log_function_info('Stopping detect.')
        return
    data = self.queue_scan.get()
    self.queue_scan.task_done()
    self.log_function_info( 'Data recieve.')
    # the medum filter needs
    mask = data
    mask = mask.astype(numpy.float32)
    data = cv2.medianBlur(mask, 5)# * numpy.max(data) /255.0/256
    # add watershed
    watershed_data = data / numpy.max(data) * 255
    watershed_data = watershed_data.astype(numpy.uint8)
    ret, thresh = cv2.threshold(watershed_data, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
    index_0 = (thresh == 0)
    index_255 = (thresh == 255)
    mask[index_0] = 255
```

[0082]

```
    mask[index_255] = 0
    # data[index_0] = 255
    # data[index_255] = 0
    # end watershed
    resampling = detect.sample_from_data(data)
    means = detect.detect_number_of_clusters(resampling)
    markers = detect.make_markers(means, data)
    if len(markers[2]) > 0:
        max_amplitud = max(markers[2]) * 1.2
        max_index = mask > max_amplitud
        mask[max_index] = max_amplitud
        self.log_function_info( 'Markers ready.')
    try:
        self.queue_markers.put([markers, data],block=True,timeout=0.5)
    except:
        self.log_function_info('Markers drop.')
    if self.display_rf_scan:
        fig = pyplot.figure(42)
        ax = fig.add_subplot(1, 1, 1)
        pyplot.cla()
        circ1 = pyplot.Circle((markers[0][0], markers[1][0]), 2.0, color="white")
        circ2 = pyplot.Circle((markers[0][1], markers[1][1]), 1.0, color="white")
        circ3 = pyplot.Circle((markers[0][2], markers[1][2]), 0.5, color="white")
```

```
                    pyplot.imshow(data)
                    ax.add_patch(circ1)
                    ax.add_patch(circ2)
                    ax.add_patch(circ3)
                    pyplot.draw()
                    pyplot.show()
                    pyplot.pause(0.1)
        def shutdown(self):
            self.stop_event.set()
    def posiciones(minima, maxima):
        x1 = 0
        x2 = 0
        x3 = 0
        y1 = 0
        y2 = 0
        y3 = 0
        dist1 = 0
        dist2 = 0
        dist3 = 0
        while dist1 < minima or dist2 < minima or dist3 < minima or dist3 > maxima or dist1 > maxima or dist2 > maxima:
            x1 = randint(0, 89)
            x2 = randint(0, 89)
            x3 = randint(0, 89)
            y1 = randint(0, 89)
            y2 = randint(0, 89)
            y3 = randint(0, 89)
            dist1 = math.sqrt((x1-x2)**2+(y1-y2)**2)
            dist2 = math.sqrt((x3-x2)**2+(y3-y2)**2)
            dist3 = math.sqrt((x1-x3)**2+(y1-y3)**2)
        return x1, y1, x2, y2, x3, y3
    def maxarg(data):
        # Busca maximos y su posicion en los datos
        x = 0
        y = 0
        maxi = 0.
        i = 0
        j = 0
        aux = data.reshape(90, 90)
        while i < 89:
            while j < 89:
                if aux[i][j] > maxi:
                    x = i
                    y = j
                    maxi = aux[i][j]
                j += 1
```

[0083]

```
                j = 0
                i += 1
        return maxi, y, x      # data.reshape seems to flip axis
def arg_sort(values):
    index = list(range(len(values)))
    for cont in range(len(values)-1):
        change = False
        for cont1 in range(len(values)-1):
            if values[cont1] < values[cont1+1]:
                tmp = values[cont1]
                values[cont1] = values[cont1+1]
                values[cont1 + 1] = tmp
                tmp = index[cont1]
                index[cont1] = index[cont1 + 1]
                index[cont1 + 1] = tmp
                change = True
        if not change:
            break
    return index
class EstimateMeanCovariance(object):
    def __init__(self, alfa = 0.05):
        self.alfa = alfa
```

[0084]

```
    def get_lambda_2(self, lambda_t, lambda_s, radio):
        err = 1e-2
        lambda_s1 = 0
        lambda_s0 = lambda_t * 0.1
        try:
            cont = 0
            while 1:
                ratio = radio/sqrt(lambda_s0)
                upper = (ratio*norm.pdf(ratio))
                lower = norm.cdf(ratio) - 0.5
                lambda_s1 = lambda_t /(1 - upper/lower )
                if lambda_s1 < 0 or lambda_s1 > 1e5:
                    # print('reset')
                    lambda_s1 = lambda_t * 0.1
                    radio = radio * 1.5
                if abs(lambda_s0 -lambda_s1) < err:
                    return lambda_s1
                lambda_s0 = lambda_s1
                cont += 1
                # if (cont % 1000) == 999:
                #         print('*',)
            # print('end')
        except Exception as e:
```

```
            raise  e
    def estimate_mean_covar_u_s_v(self, data, dimention, points_index):
        x_center = numpy.zeros((dimention,))
        covar_matrix = numpy.zeros((dimention, dimention,))
        for index_p in points_index:
            # print(data[index_p, :])
            x_center += data[int(index_p), :]
        # initial mean estimation
        x_center /= len(points_index)
        for index_p in points_index:
            x_i = x_center - data[int(index_p), :]
            x_i = x_i[:, None]
            covar_matrix += numpy.dot(x_i, x_i.T)
        # initial estimation covar matrix
        covar_matrix /= len(points_index)
        U, s, V = numpy.linalg.svd(covar_matrix)
        s = numpy.diag(s)
        return x_center, covar_matrix, U, s, V
    def estimate_mean_variance(self, data, initial_points_index):
        points_size, dimention = data.shape
        x_center, covar_matrix, U, s, V = self.estimate_mean_covar_u_s_v(data, dimention, initial_points_index)
        radio = 0
[0085]  # print('start mean estimation')/
        for cont in initial_points_index:
            aux = data[int(cont)] - x_center
            r = sqrt(numpy.dot(aux[:, None].T, aux[:, None]))
            if  radio < r:
                radio = r
        # initial estimation lambda square
        lambda_0 = numpy.ones((dimention,))*radio
        sqrt_chi_p_alpha = sqrt(chi2._ppf(1-self.alfa,2))
        for cont in range(dimention):
            lambda_0[cont] /= sqrt_chi_p_alpha
        # calculate initial hyperrectanble axes
        hyper_rectangle_thing = numpy.zeros((dimention, dimention,))
        previus_covariance = covar_matrix.copy()
        prev_error = 0
        for watch_dog in range(150):
            for cont in range(dimention):
                hyper_rectangle_thing[cont, :] = sqrt_chi_p_alpha * lambda_0[cont]*U[cont,:]
            points_in = []
            # get the points that are inside the hyper rectangle
            for cont in range(points_size):
                choosen_point = data[cont, :].copy()
                choosen_point -= x_center
```

```
            # check if point is inside hyper-cube
            is_in = True
            for cont1 in range(dimention):
                # print(cont,cont1,'a')
                vector_magnitud = sqrt(numpy.dot(hyper_rectangle_thing[cont1, :].T,hyper_rectangle_thing[cont1, :]))
                unitar_vector = hyper_rectangle_thing[cont1, :]/vector_magnitud
                projection = numpy.dot(choosen_point, unitar_vector)
                if not (projection <= vector_magnitud and projection >= -vector_magnitud):
                    is_in = False
                    break
            if is_in:
                points_in.append(cont)
        # read the documentation on numpy linalg svd
        x_center, covar_matrix, U, s, V = self.estimate_mean_covar_u_s_v(data, dimention, points_in)
        # estimate the values of the norma distribution usin
        # a truncated distribution , see result 2.2 and 2.1
        radio_ = numpy.zeros((dimention,))
        for cont in range(dimention):
            radio_aux = hyper_rectangle_thing[cont,:]
            radio_[cont] = sqrt(numpy.dot(radio_aux,radio_aux))
        o,ss,oo = numpy.linalg.svd(previus_covariance)
        for cont in range(dimention):
[0086]      if isnan(s[cont,cont]):
                logging.ERROR('NaN')
            if s[cont, cont] == 0:
                logging.ERROR('by zero')
            lambda_2 = self.get_lambda_2(s[cont,cont], ss[cont],radio_[cont])
            s[cont,cont] = (lambda_2)
        # estimate the region of 1-alfa confidence
        radio_constant = chi2._ppf(1 - self.alfa, 2) / dimention
        # se the points that are inside this last region
        region_b = []
        A_matrix = numpy.dot(numpy.dot(U, s * radio_constant), V.T)
        for cont in range(points_size):
            choosen_point = data[cont, :].copy()
            choosen_point -= x_center
            # check if point is inside hyper-cube
            is_in = True
            points_ = choosen_point[:,None]
            evalueate_ellipsoid = numpy.dot(numpy.dot(points_.T, numpy.linalg.inv(A_matrix)), points_)
            if evalueate_ellipsoid <= 1:
                region_b.append(cont)
        x_center, covar_matrix, U, s, V = self.estimate_mean_covar_u_s_v(data, dimention, region_b)
        error = numpy.sum(numpy.abs(covar_matrix - previus_covariance))
        if error < 0.1:
```

```
                    break
                if abs(error - prev_error) < 1e-6:
                    break
                prev_error = error
                U, s, V = numpy.linalg.svd(covar_matrix)
                for cont in range(dimention):
                    lambda_0[cont] = sqrt(s[cont])
                previus_covariance = covar_matrix.copy()
            # print('end    mean estimation')
            return covar_matrix, x_center
class ResampleData(object):
    def interpol_data(self, data, y_index, x_index):
        lower_x = int(x_index)
        lower_y = int(y_index)
        upper_x = int(x_index + 1)
        upper_y = int(y_index + 1)
        value = data[upper_y, upper_x]*sqrt((y_index - upper_y)**2 + (x_index - upper_x)**2)
        value += data[lower_y, lower_x]*sqrt((y_index - lower_y)**2 + (x_index - lower_x)**2)
        value /= sqrt((upper_y - lower_y)**2 + (upper_x - lower_x)**2)
        return value
    def sample_data(self, data):
        return self.sample_data_gen(data, 0.005)
[0087]  def sample_data_gen(self, data, dist_val, size_x = 30, size_y =30):
        maximum = dist_val
        total = numpy.sum(data)
        sample_data_ = data/total
        while 1:
            [idx_x, idx_y] = [self.float_random(size_x),self.float_random(size_y)]
            offset = self.float_random(maximum)
            if offset <= self.interpol_data(sample_data_,idx_y,idx_x):
                return idx_y,idx_x
    def sample_data_gen_1d(self, data, dist_val):
        while 1:
            idx_y = random_integers(0, data.size - 1)
            offset = self.float_random(dist_val)
            try:
                if offset <= data[idx_y]:
                    return idx_y
            except:
                logging.ERROR(data.shape,'fail to sample',data.size)
                return 0
    def float_random(self, ends):
        return random()*ends
ADS_CONV_REG    = 0x00
ADS_CONF_REG    = 0x01
```

```
ADS_LTHR_REG  = 0x02
ADS_HTHR_REG  = 0x03
ADS_ADDR_GND  = 0x90
ADS_ADDR_VDD  = 0x92
ADS_ADDR_SDA  = 0x94
ADS_ADDR_SCL  = 0x96
ADS_READ  = 0x01
ADS_WRITE  = 0x00
ADS_OS  = 0x8000
ADS_CONV_CURR  = 0x0000
ADS_CONV_NCURR  = 0x8000
ADS_NOFX  = 0x0000
ADS_CONV_BEGIN  = 0x8000
ADS_MUX  = 0x7000
ADS_AIN0_AIN1  = 0x0000
ADS_AIN0_AIN3  = 0x1000
ADS_AIN1_AIN3  = 0x2000
ADS_AIN2_AIN3  = 0x3000
ADS_AIN0_GND  = 0x4000
ADS_AIN1_GND  = 0x5000
ADS_AIN2_GND  = 0x6000
ADS_AIN3_GND  = 0x7000
```

[0088]

```
ADS_PGA  = 0x0E00
ADS_6144  = 0x0000
ADS_4096  = 0x0200
ADS_2048  = 0x0400
ADS_1024  = 0x0600
ADS_0512  = 0x0800
ADS_0256  = 0x0A00
ADS_MODE  = 0x0100
ADS_CONT_CONV  = 0x0000
ADS_SING_CONV  = 0x0100
ADS_DATA_RATE  = 0xE0
ADS_8SPS  = 0x00
ADS_16SPS  = 0x20
ADS_32SPS  = 0x40
ADS_64SPS  = 0x60
ADS_128SPS  = 0x80
ADS_250SPS  = 0xA0
ADS_475SPS  = 0xC0
ADS_860SPS  = 0xE0
ADS_COMP_MODE  = 0x10
ADS_COMP_TRAD  = 0x00
ADS_COMP_WIND  = 0x10
ADS_COMP_POL  = 0x08
```

```
ADS_COMP_LOW    = 0x00
ADS_COMP_HIGH   = 0x08
ADS_COMP_LAT    = 0x04
ADS_COMP_NLATCH   = 0x00
ADS_COMP_LATCH    = 0x04
ADS_COMP_QUE    = 0x03
ADS_COMP_ONEC    = 0x00
ADS_COMP_TWOC    = 0x01
ADS_COMP_FOURC    = 0x02
ADS_COMP_DISABLE    = 0x03
// Codigo en C, implementacion del servidor local
/* A simple server in the internet domain using TCP
    The port number is passed as an argument */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/mman.h>
```

[0089]

```
#include <sys/stat.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <unistd.h>
#include <math.h>
#include <wiringPiI2C.h>
#include "cnpy.h"
#include "ADS1115.h"
#include <iostream>
#include <time.h>
/*socket related define*/
#define MY_SOCK_PATH "/home/pi/comunication"
#define LISTEN_BACKLOG 50
#define handle_error(msg) do { perror(msg); exit(EXIT_FAILURE); } while (0)
/*scann related define*/
#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)
#define SIZE_N 31
#define MIN_ANG -30
#define MAX_ANG 30
#define STEP_ANG (MAX_ANG-MIN_ANG)/(SIZE_N - 1.0)
#define MEAN_ITER 2
#define WAIT_T 3000
```

```
#define BCM2708_PERI_BASE        0x3F000000
#define GPIO_BASE                (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */
#define WAIT_ADC 500000
using namespace std;
int    mem_fd;
void *gpio_map;
struct timespec tim, tim2;
// I/O access
volatile unsigned *gpio;
// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))
#define GPIO_SET *(gpio+7)  // sets bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0
#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH
#define GPIO_PULL *(gpio+37) // Pull up/pull down
#define GPIO_PULLCLK0 *(gpio+38) // Pull up/pull down clock
/*functoins*/
void setup_io();
void send_data(double* val, unsigned char addr);
void read_from_pipe (int file);
```

[0090]

```
void write_to_pipe (int file);
int do_scan(double data[], double angle[SIZE_N][SIZE_N][16], double theta_array[], int fd);
/*Global variable*/
double angle[SIZE_N][SIZE_N][16];
double theta_array[SIZE_N];
int fd;
void error(const char *msg)
{
    perror(msg);
    exit(1);
}
int handleClientRequest(int clientFD)
{
    char buffer[256];
    int n;
    /*Scanc configureations*/
      double data[SIZE_N*SIZE_N];
       /********************************************************************\
        * You are about to change the GPIO settings of your computer.      *
        * Mess this up and it will stop working!                           *
        * It might be a good idea to 'sync' before running this program    *
        * so at least you still have your code changes written to the SD-card!  *
       \********************************************************************/
```

```
        // Set GPIO pins 18, 23 and 24 to output
        // must use INP_GPIO before we can use OUT_GPIO
      /*End scann configurations*/
        while(0==0)
        {
                bzero(buffer,256);/*clearing the buffer*/
                n = read(clientFD,buffer,255);/*reading from the client connectin*/
                if (n < 0) /*error cheking*/
                {
                        error("ERROR reading from socket");
                }

                /*recieve a close socket message*/
                if(strcmp(buffer,"close\n") == 0 || strcmp(buffer,"") == 0)
                {
                        return 0;
                }
                if(strcmp(buffer,"scann\n") == 0)
                {
                        if(do_scan(data,angle, theta_array, fd)==0)
                        {
                                n = write(clientFD,(char*)data,SIZE_N*SIZE_N*8);/*size of array*number of char per data*/
                        }
                }

        }
      return 0;
}
/*scan functiond*/
void setup_io();
void send_data(float angle, unsigned char addr);
/*main function, here the socket is configured and start to
 * run, waiting a request.*/
int main(int argc, char *argv[])
{
      int sockfd, newsockfd;
      socklen_t clientLen;
      struct sockaddr_un serv_addr, client_addr;
     /*Initialize the socket configuration*/
      sockfd = socket(AF_LOCAL, SOCK_STREAM, 0);
      if (sockfd < 0)
      {
          error("ERROR opening socket");
      }
      /*initialize address structure*/
```

[0091]

```
memset(&serv_addr, 0, sizeof(struct sockaddr_un)); /* Clear structure */
serv_addr.sun_family = AF_LOCAL;
strncpy(serv_addr.sun_path, MY_SOCK_PATH, sizeof(serv_addr.sun_path) - 1);
unlink(serv_addr.sun_path);
/*start bind*/
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(struct sockaddr_un)) == -1)
{
    handle_error("bind");
}
/*set the number of incomming connections*/
if (listen(sockfd, LISTEN_BACKLOG) == -1)
{
    handle_error("listen");
}
clientLen = sizeof(struct sockaddr_un);
/*Configure PI ports and phase shifter values*/
// Set up gpi pointer for direct register access
  setup_io();
  fd = wiringPiI2CSetup(0x54);
  /*wiringPiI2CWriteReg16(fd, 0x02, 0x20);*/
  wiringPiI2CWriteReg16(fd, 0x02, 0x20);
  INP_GPIO(18);
```

[0092]

```
  INP_GPIO(23);
  INP_GPIO(24);
  INP_GPIO(17);
  INP_GPIO(27);
  INP_GPIO(22);
  INP_GPIO(5);
  INP_GPIO(6);
  INP_GPIO(13);
  INP_GPIO(26);
  OUT_GPIO(18);
  OUT_GPIO(23);
  OUT_GPIO(24);
  OUT_GPIO(17);
  OUT_GPIO(27);
  OUT_GPIO(22);
  OUT_GPIO(5);
  OUT_GPIO(6);
  OUT_GPIO(13);
  OUT_GPIO(26);
  GPIO_SET = 1<<18;
  GPIO_CLR = (1<<24)|(1<<23)|(1<<17)|(1<<27)|(1<<22)|(1<<5)|(1<<6)|(1<<13)|(1<<26);
  double dt = 0;
  double dp = 0;
```

```
double dt2 = 0;
double dp2 = 0;
double dt3 = 0;
double dp3 = 0;
for(int i = 0; i < SIZE_N; i++) theta_array[i] = 180.0*sin((MIN_ANG + STEP_ANG*i)*M_PI/180.0);
for(int i = 0; i < SIZE_N; i++)
{
    for(int j = 0; j < SIZE_N; j++)
    {
        dt = theta_array[i];
        dp = theta_array[j];
        dt2 = 2*dt;
        dp2 = 2*dp;
        dt3 = 3*dt;
        dp3 = 3*dp;
        angle[i][j][0] = (int(360 + 360 + 360 - dt3)%360)*255.0/360;
        angle[i][j][1] = (int(360 + 360 + 360 - dt2)%360)*255.0/360;
        angle[i][j][2] = (int(360 + 360 + 360 - dt)%360)*255.0/360;
        angle[i][j][3] = (int(360 + 360 + 360 + 0)%360)*255.0/360;
        angle[i][j][4] = (int(360 + 360 + 360 - dt3 + dp2)%360)*255.0/360;
        angle[i][j][5] = (int(360 + 360 + 360 - dt2 + dp2)%360)*255.0/360;
        angle[i][j][6] = (int(360 + 360 + 360 - dt    + dp2)%360)*255.0/360;
[0093]  angle[i][j][7] = (int(360 + 360 + 360 + 0     + dp2)%360)*255.0/360;
        angle[i][j][8] =    (int(360 + 360 + 360 - dt3 + dp)%360)*255.0/360;
        angle[i][j][9] =    (int(360 + 360 + 360 - dt2 + dp)%360)*255.0/360;
        angle[i][j][10] = (int(360 + 360 + 360 - dt    + dp)%360)*255.0/360;
        angle[i][j][11] = (int(360 + 360 + 360 + 0     + dp)%360)*255.0/360;
        angle[i][j][12] = (int(360 + 360 + 360 - dt3 + dp3)%360)*255.0/360;
        angle[i][j][13] = (int(360 + 360 + 360 - dt2 + dp3)%360)*255.0/360;
        angle[i][j][14] = (int(360 + 360 + 360 - dt    + dp3)%360)*255.0/360;
        angle[i][j][15] = (int(360 + 360 + 360 + 0     + dp3)%360)*255.0/360;
    }
}
/*change the permissions on the unix socket, to allow access without sudo*/
chmod(MY_SOCK_PATH, 0666);
while (true)
{
    printf("Waiting request...\n");
    newsockfd = accept(sockfd, (struct sockaddr *) &client_addr, &clientLen);
    if (newsockfd == -1)
    {
        handle_error("accept");
    }

    if (handleClientRequest(newsockfd) == 1)
```

```
            {
                break;
            }
        }
        printf("by\n");
        unlink(MY_SOCK_PATH);
        return 0;
}
int do_scan(double data[],double angle[SIZE_N][SIZE_N][16],double theta_array[], int fd)
{
        clock_t tStart = clock();
        int conv_reg;
        int result;
        double mean = 0.0;
        double val [8] = {0,0,0,0,0,0,0,0};
        uint8_t buf1 = 0;
        uint8_t buf2 = 0;
        printf("Minimum angle = %d\n", MIN_ANG);
        printf("Maximum angle = %d\n", MAX_ANG);
        printf("Step angle = %f\n", STEP_ANG);
        for(int i = 0; i < SIZE_N; i++)
        {
```

[0094]

```
            for(int j = 0; j < SIZE_N; j++)
            {
                    val[0] = angle[i][j][0];
                    val[1] = angle[i][j][1];
                    val[2] = angle[i][j][2];
                    val[3] = angle[i][j][3];
                    val[4] = angle[i][j][4];
                    val[5] = angle[i][j][5];
                    val[6] = angle[i][j][6];
                    val[7] = angle[i][j][7];
                    send_data(val, 0);
                    val[0] = angle[i][j][8];
                    val[1] = angle[i][j][9];
                    val[2] = angle[i][j][10];
                    val[3] = angle[i][j][11];
                    val[4] = angle[i][j][12];
                    val[5] = angle[i][j][13];
                    val[6] = angle[i][j][14];
                    val[7] = angle[i][j][15];
                    send_data(val, 1);
                    mean = 0.0;
                    result = wiringPiI2CWriteReg16(fd, 0x07, 0);
                    for(int z = 0; z < WAIT_ADC; z++) ((void)0);
```

```
                result = wiringPiI2CReadReg16(fd, 0x07);
                if(result < 0) printf("Error at %d,%d\n", i, j);
                buf1 = (result >> 10) & 0x3F;
                buf2 = result & 0x0F;
                conv_reg = (buf2 << 6) + buf1;
                mean = (conv_reg*3.3/1024 - 1.5269)/0.0174;
                data[j*SIZE_N + (SIZE_N - i - 1)] = mean;
            }
        }
        printf("Time taken: %.2fs\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);
        //const unsigned int shape[] = {SIZE_N, SIZE_N};
        //cnpy::npy_save("arr1.npy", data, shape, 2, "w");
        //printf("saved\n");
        printf("Time taken: %.2fs\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);
        return 0;
    }
    //
    // Set up a memory regions to access GPIO
    // setup_io
    void setup_io()
    {
        /* open /dev/mem */
[0095]  if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0)
        {
            printf("can't open /dev/mem \n");
            exit(-1);
        }
        /* mmap GPIO */
        gpio_map = mmap(
            NULL,                 //Any adddress in our space will do
            BLOCK_SIZE,           //Map length
            PROT_READ|PROT_WRITE,// Enable reading & writting to mapped memory
            MAP_SHARED,           //Shared with other processes
            mem_fd,               //File to map
            GPIO_BASE             //Offset to GPIO peripheral
        );
        close(mem_fd); //No need to keep mem_fd open after mmap
        if (gpio_map == MAP_FAILED) {
            printf("mmap error %d\n", (int)gpio_map);//errno also set!
            exit(-1);
        }
        // Always use volatile pointer!
        gpio = (volatile unsigned *)gpio_map;
    }
    void send_data(double* val, unsigned char addr)
```

```
{
//    unsigned char opt = (int) 0;
    uint8_t data[8];
    data[0] = (uint8_t) val[0];
    data[1] = (uint8_t) val[1];
    data[2] = (uint8_t) val[2];
    data[3] = (uint8_t) val[3];
    data[4] = (uint8_t) val[4];
    data[5] = (uint8_t) val[5];
    data[6] = (uint8_t) val[6];
    data[7] = (uint8_t) val[7];
    for(int j = 0; j < WAIT_T; j++) ((void)0);
    GPIO_CLR = (1<<18);
//    float angle = strtof(argv[1], NULL, 0);
    for(int i = 0; i < 8; i++)
    {
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_CLR = 1<<24;
        if ((data[0] >> i) & 0x01) GPIO_SET = 1<<23;
        else GPIO_CLR = 1<<23;
        if ((data[1] >> i) & 0x01) GPIO_SET = 1<<17;
        else GPIO_CLR = 1<<17;
        if ((data[2] >> i) & 0x01) GPIO_SET = 1<<27;
        else GPIO_CLR = 1<<27;
        if ((data[3] >> i) & 0x01) GPIO_SET = 1<<22;
        else GPIO_CLR = 1<<22;
        if ((data[4] >> i) & 0x01) GPIO_SET = 1<<5;
        else GPIO_CLR = 1<<5;
        if ((data[5] >> i) & 0x01) GPIO_SET = 1<<6;
        else GPIO_CLR = 1<<6;
        if ((data[6] >> i) & 0x01) GPIO_SET = 1<<13;
        else GPIO_CLR = 1<<13;
        if ((data[7] >> i) & 0x01) GPIO_SET = 1<<26;
        else GPIO_CLR = 1<<26;
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_SET = 1<<24;
    }
    for(int j = 0; j < WAIT_T; j++) ((void)0);
//  OPT bit
    GPIO_CLR = 1<<24;
    GPIO_CLR = (1<<23)|(1<<17)|(1<<27)|(1<<22)|(1<<5)|(1<<6)|(1<<13)|(1<<26);
    for(int j = 0; j < WAIT_T; j++) ((void)0);
    GPIO_SET = 1<<24;
    for(int j = 0; j < WAIT_T; j++) ((void)0);
//  Address bit 0
```

[0096]

```
        GPIO_CLR = 1<<24;
        if (addr == 0) GPIO_CLR = (1<<23)|(1<<17)|(1<<27)|(1<<22)|(1<<5)|(1<<6)|(1<<13)|(1<<26);
        else if (addr == 1) GPIO_SET = (1<<23)|(1<<17)|(1<<27)|(1<<22)|(1<<5)|(1<<6)|(1<<13)|(1<<26);
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_SET = 1<<24;
        for(int j = 0; j < WAIT_T; j++) ((void)0);
    //  Address bit 1
        GPIO_CLR = (1<<24)|(1<<23)|(1<<17)|(1<<27)|(1<<22);
        GPIO_SET = (1<<5)|(1<<6)|(1<<13)|(1<<26);
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_SET = 1<<24;
        for(int j = 0; j < WAIT_T; j++) ((void)0);
    //  Address bit 2
        GPIO_CLR = (1<<24)|(1<<23)|(1<<5)|(1<<27)|(1<<13);
[0097]  GPIO_SET = (1<<17)|(1<<6)|(1<<22)|(1<<26);
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_SET = 1<<24;
        for(int j = 0; j < WAIT_T; j++) ((void)0);
    //  Address bit 3
        GPIO_CLR = (1<<24)|(1<<23)|(1<<17)|(1<<5)|(1<<6);
        GPIO_SET = (1<<27)|(1<<22)|(1<<13)|(1<<26);
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_SET = 1<<24;
        for(int j = 0; j < WAIT_T; j++) ((void)0);
    //  Last sequence
        GPIO_CLR = (1<<23)|(1<<24)|(1<<17)|(1<<27)|(1<<22)|(1<<5)|(1<<6)|(1<<13)|(1<<26);
        for(int j = 0; j < WAIT_T; j++) ((void)0);
        GPIO_SET = 1<<18;
    }
```

[0098] 对附图的简要描述

[0099] 图1/10

[0100] 这个图描述了从4个辐射元件(天线)合成的辐射图案。发现该图案与天线之间的相对相移具有16度的倾角。

[0101] (1)扁平天线阵列

[0102] (11)天线或辐射元件

[0103] 图2/10

[0104] 这个图呈现了本发明的一般框图。

[0105] (1)扁平天线阵列

[0106] (2)高频电子设备

[0107] (3)移相器

[0108] (4)总功率检测器

[0109] (5)模数转换器(A/DC)

[0110] (6)微型计算机

[0111] (7)电源

[0112] (8)屏幕

[0113] (9)设备保护外壳

[0114] (10)天线或辐射元件

[0115] (11)放大器

[0116] (12)滤波器

[0117] (13)视频相机

[0118] (14)组合器

[0119] 图3/10

[0120] 这个图表示以矩阵形式排序的辐射元件的几何布局(不排除其它布局)。

[0121] (10)天线或辐射元件

[0122] 图4/10

[0123] 这个图呈现了具有高频电子设备的板的一般框图。

[0124] (1)扁平天线阵列

[0125] (3)移相器

[0126] (10)天线或辐射元件

[0127] (11)放大器

[0128] (12)滤波器

[0129] (14)组合器

[0130] 图5/10

[0131] 这个图呈现了在用于PCS电话频带的微带天线中实现的4×4天线阵列的模型。(以mm为单位)

[0132] (1)扁平天线阵列

[0133] (10)天线或辐射元件

[0134] 图6/10

[0135] 这个图示出了16个天线的应用示例,带有放大器和输出。它还具有使用商业部件的高频电子板的模型(不仅限于这种类型的部件)。

[0136] (1)扁平天线阵列

[0137] (2)高频电子设备

[0138] (3)移相器

[0139] (4)总功率检测器

[0140] (5)模数转换器(A/DC)

[0141] (6)微型计算机

[0142] (7)电源

[0143] (10)天线或辐射元件

[0144] (11)放大器

[0145] (12)滤波器,可选

[0146] (14)组合器

[0147] 图7/10

[0148] 这个图具体图示了使用商业部件的应用示例1至4的天线图(不仅限于这些类型的部件)。(天线5至8、9至13和14至16的图在其分布方面是类似的)

[0149] (3)移相器

[0150] (7)电源

[0151] (10)天线或辐射元件

[0152] (14)组合器

[0153] 图8/10

[0154] 这个图呈现了用于将测得的功率转换成模拟电压的第二电路板的图。

[0155] (4)总功率检测器

[0156] (7)电源

[0157] 图9/10

[0158] 这个图呈现了微型计算机与相机、屏幕、移相器和A/DC的连接图。

[0159] (3)移相器

[0160] (5)模数转换器(A/DC)

[0161] (6)微型计算机

[0162] (8)屏幕

[0163] (13)视频相机

[0164] 图10/10

[0165] 这个图的上部部分示出了实现的系统的模型。

[0166] (1)扁平天线阵列

[0167] (7)电源

[0168] (8)屏幕

[0169] (9)设备保护外壳

[0170] (10)天线或辐射元件

[0171] (13)视频相机

[0172] 下部的图呈现了系统的分解图,尤其包括天线阵列、光学相机、保护外壳、屏幕。

[0173] 应用示例

[0174] 在不限制其部件的情况下,这个设备的一个应用示例是使用16个微带天线的阵列、两个具有商业部件的高频电子板以及商业微型计算机来实现的。

[0175] 这个设备的设计的图像在图6/10、图7/10和图8/10中示出。

[0176] 微带天线与调谐到1.88GHz并且带宽约为50MHz的矩形天线对应。图5/10

[0177] 第一高频电子板具有16个**Analog®**HMC631LP3向量模块,这些模块允许移位由每个天线接收的信号的相位和幅度。这些调制器是利用**Maxim®**的数字电位计DS3930来控制的,其与I2c总线通信。然后使用**Minicircuits®**的14个TCP-2-272功率组合器将信号同相组合。然后用**MACOM®**的低噪声放大器(LNA)型号MAAL-007304放大信号。

[0178] 第二板具有**Linear®**LT5538型功率检测器,并且用于将测得的功率转换成模拟电压。

[0179] 所使用的模数转换器是ADS1115，它的测试板是商业上购买的。后者连接到**Raspberry®**pi 3微型计算机。

[0180] 由于天线仅接收一个频率频带，因此在该设备中滤波器的结合是可选的。

[0181] 另一方面，在这个原型示例中使用的光学数字相机是800×600mm。在图10/10中所呈现的示例中，可以在天线阵列上方看到光学相机。将相机定位在天线的中心也是优选的，因为这个位置简化了用于射频相机的计算。

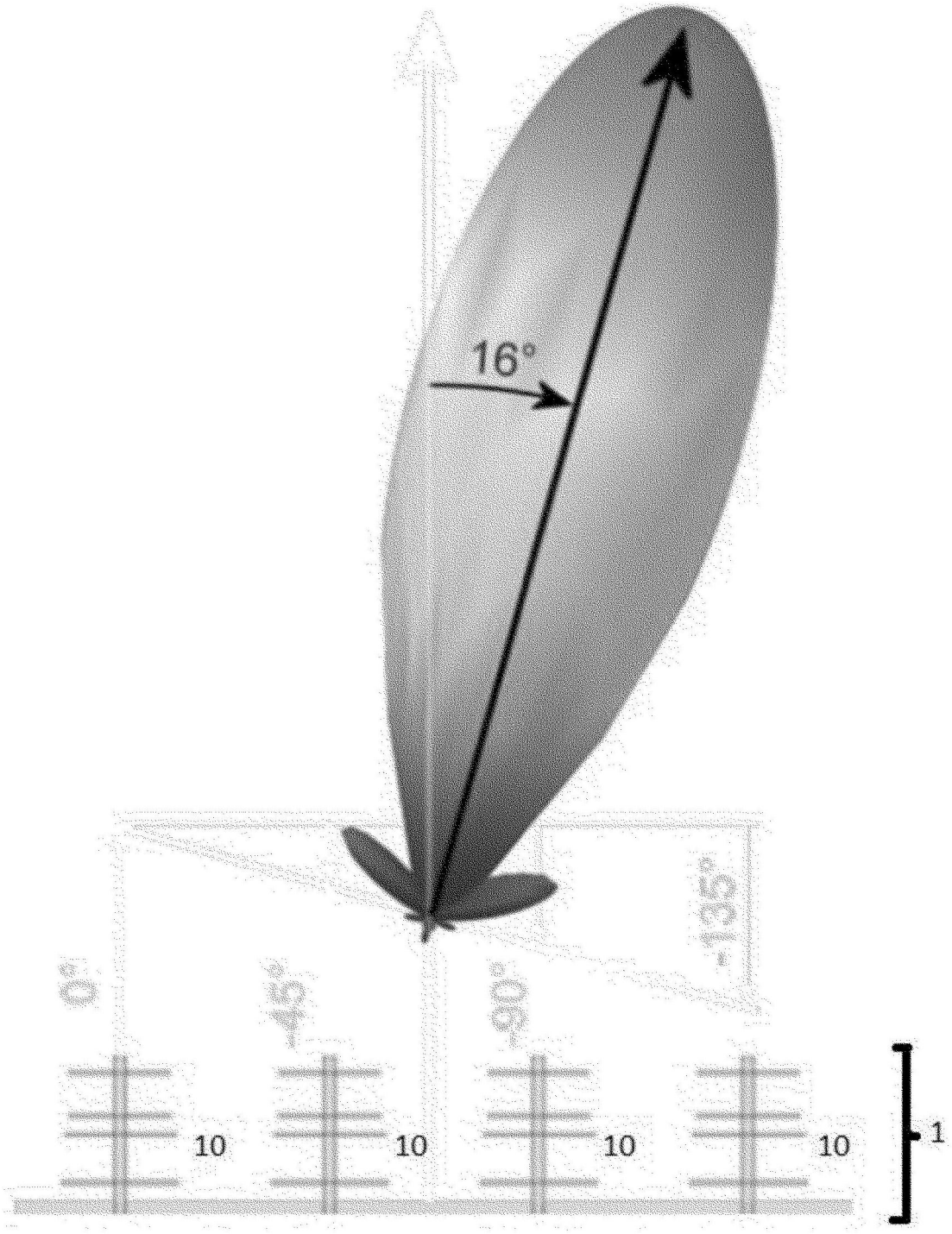


图1

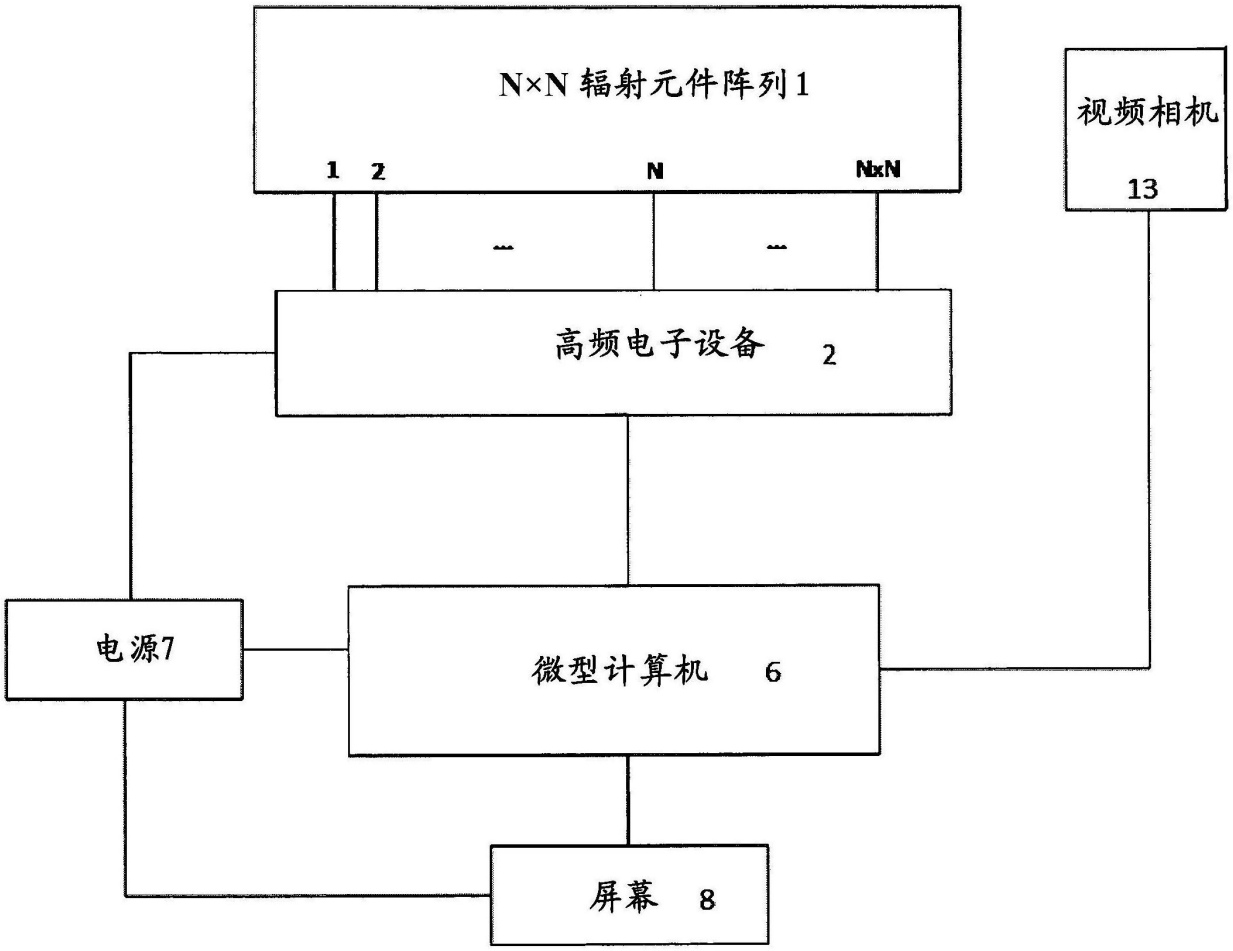


图2

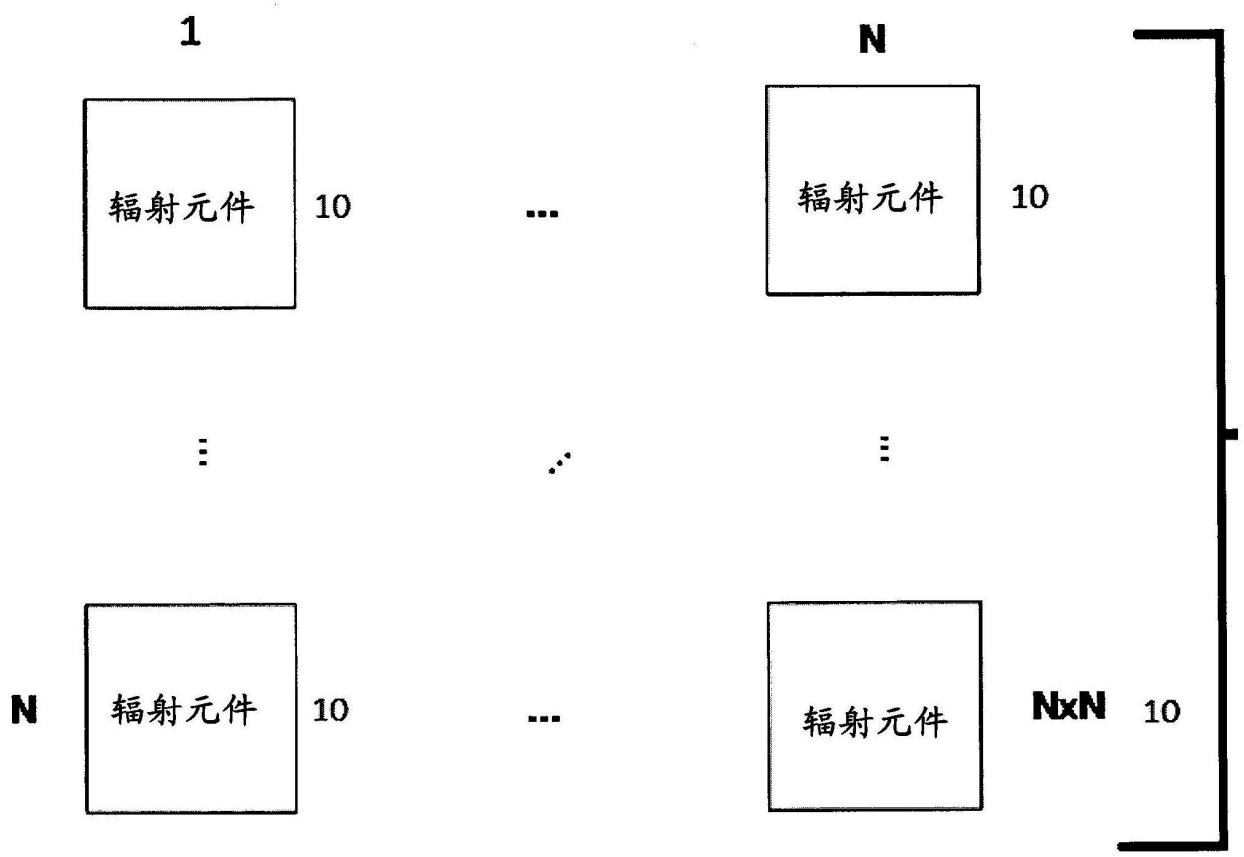


图3

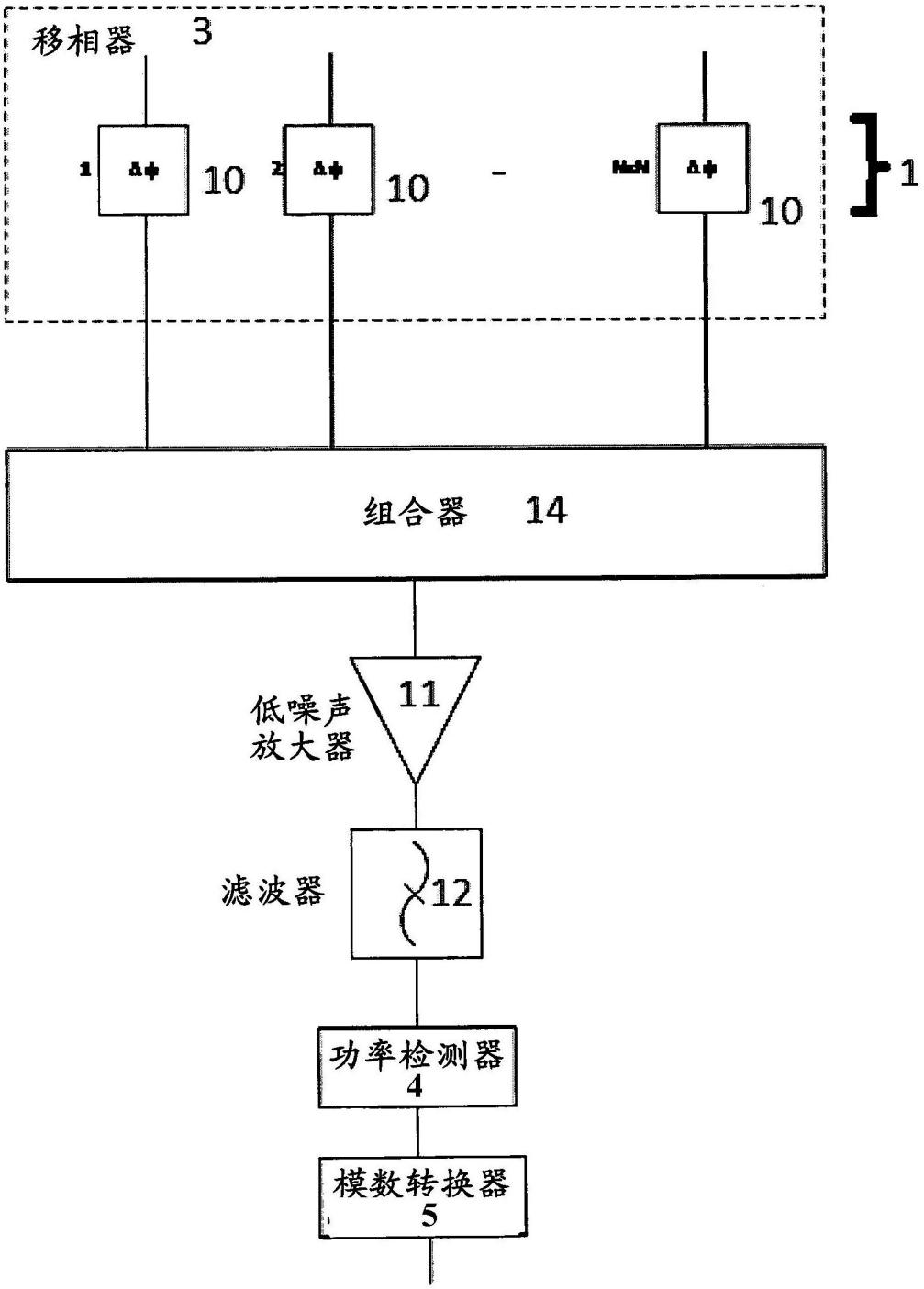

移相器 3
1 Δφ 10
2 Δφ 10
N Δφ 10
1
组合器 14
低噪声放大器 11
滤波器 12
功率检测器 4
模数转换器 5

图4

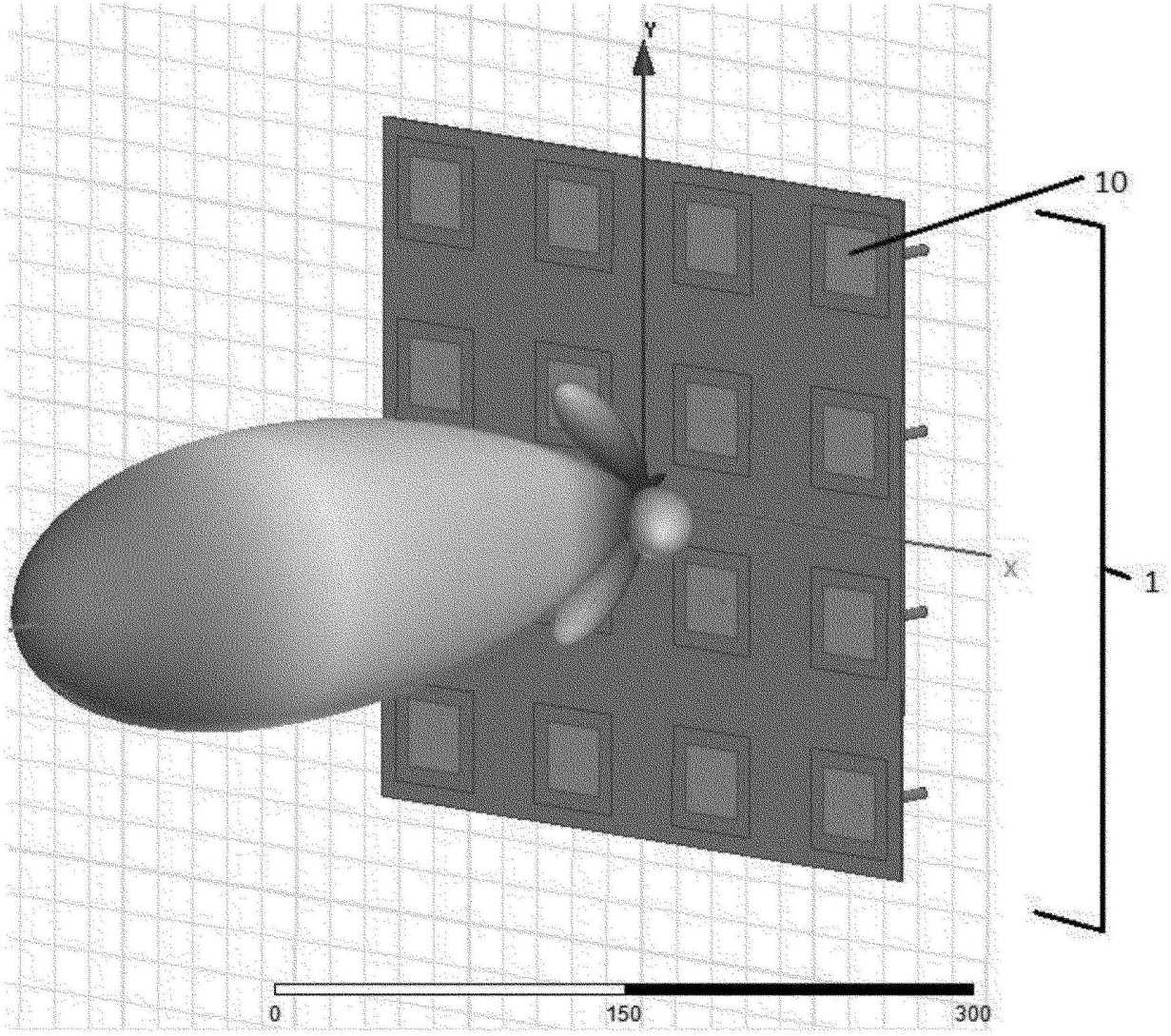


图5

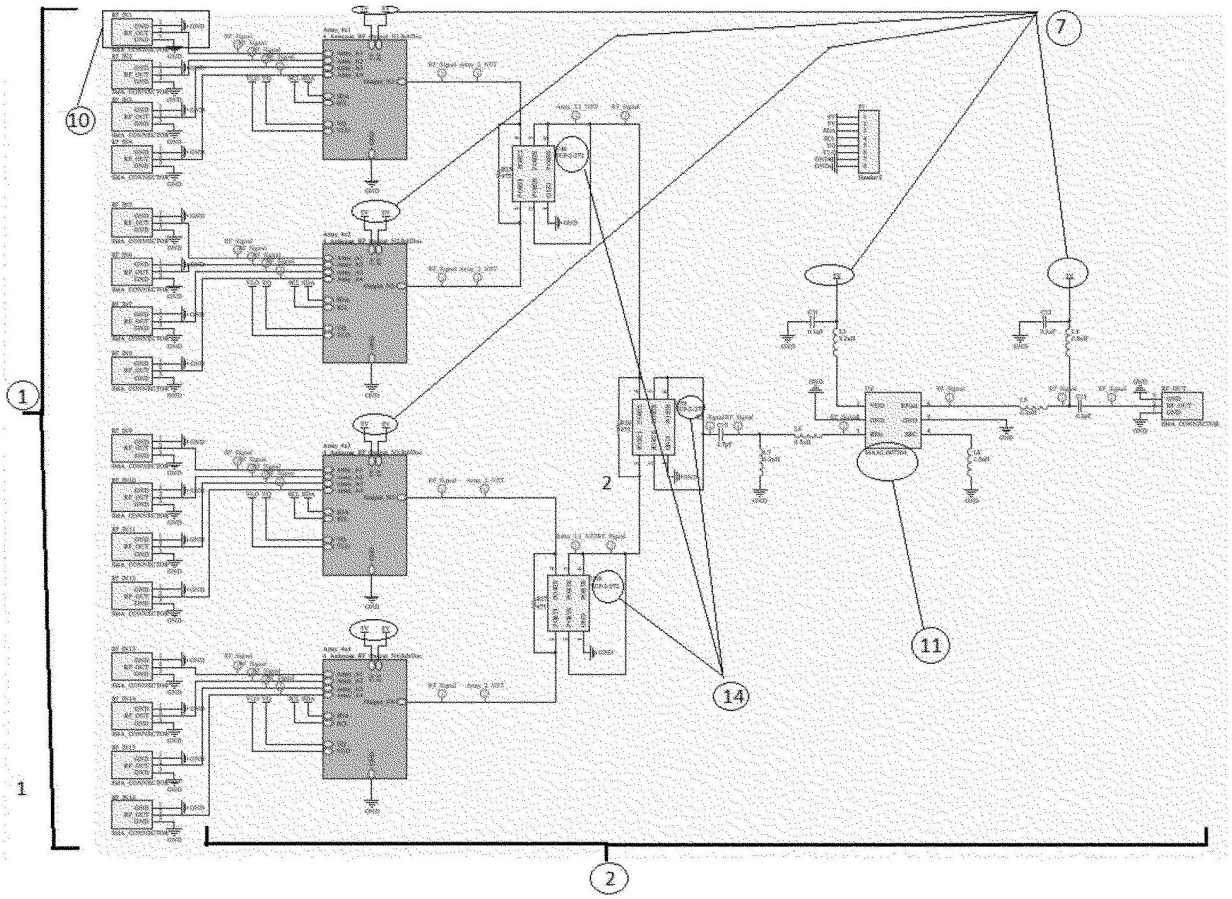


图6

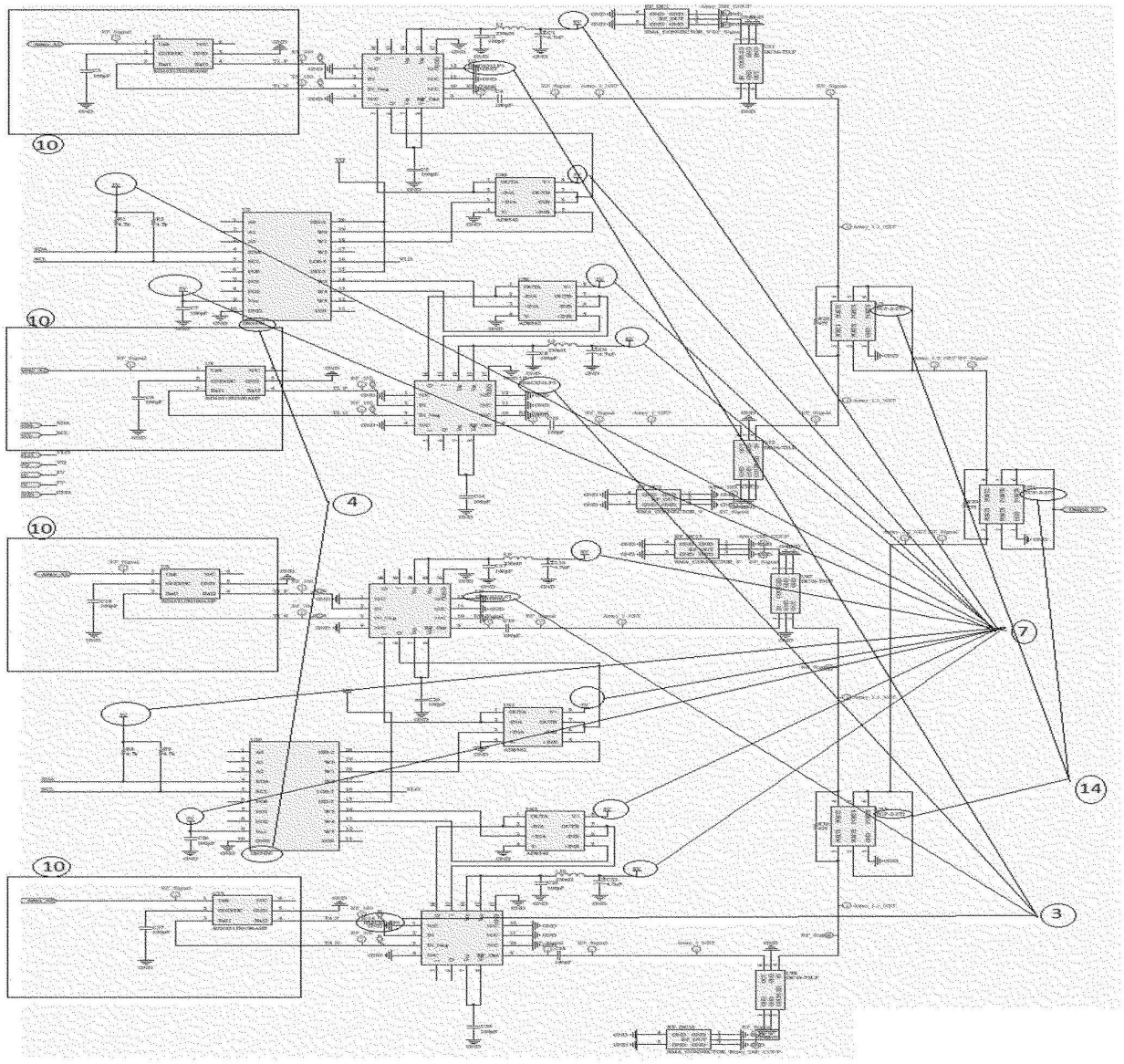


图7

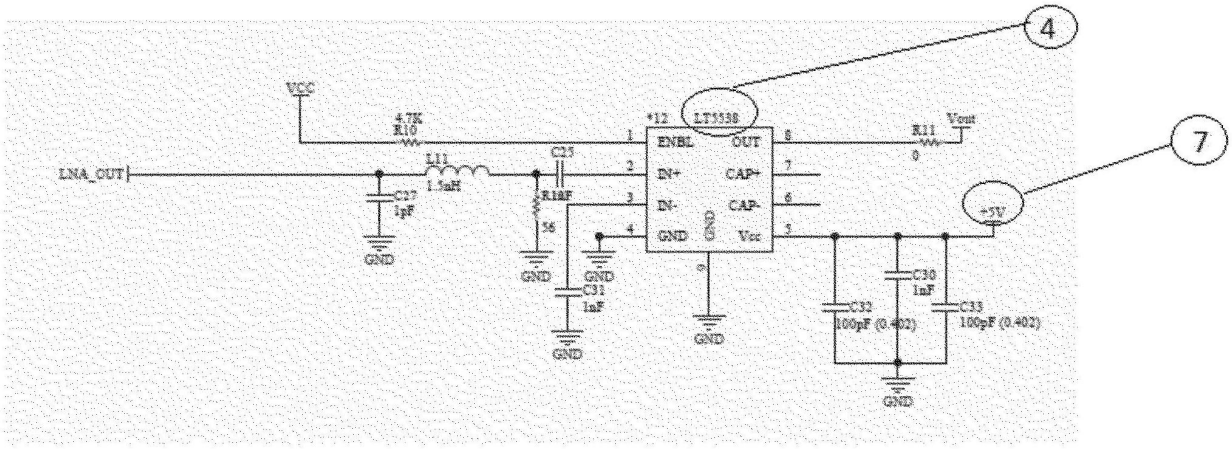

4
7
VCC
4.7K
R10
LNA_OUT
L11
1.5nH
C27
1pF
C25
R18F
56
+12
LT5538
ENBL
IN+
IN-
GND
OUT
CAP+
CAP-
Vcc
R11
0
Vout
+5V
C30
1nF
C32
100pF (0.402)
C33
100pF (0.402)
C31
1nF
GND

图8

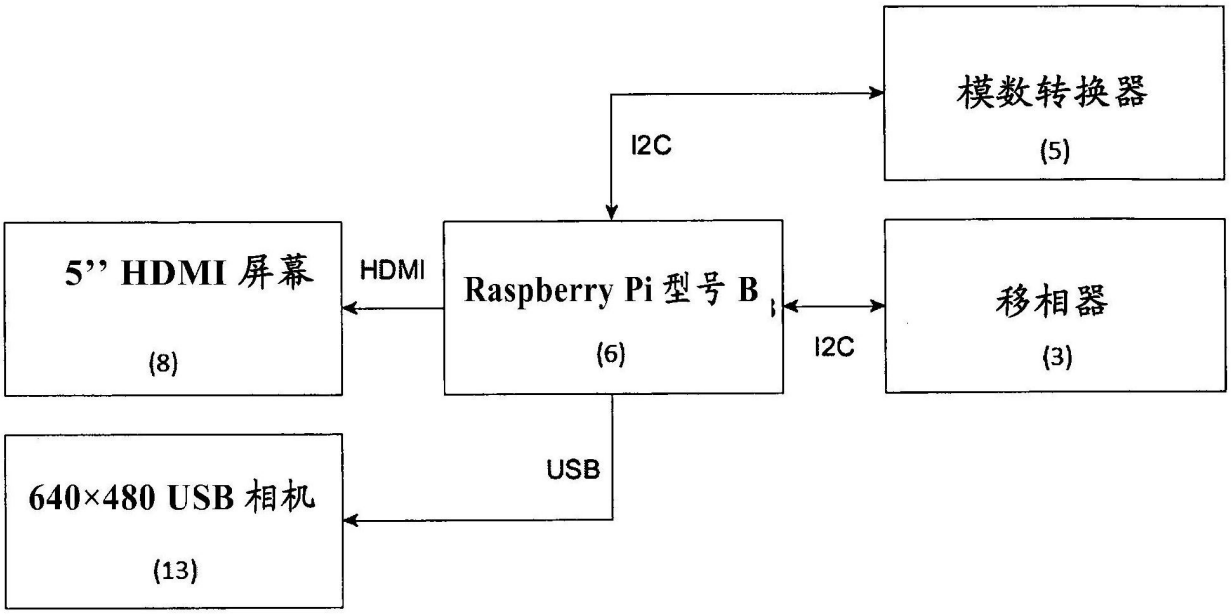
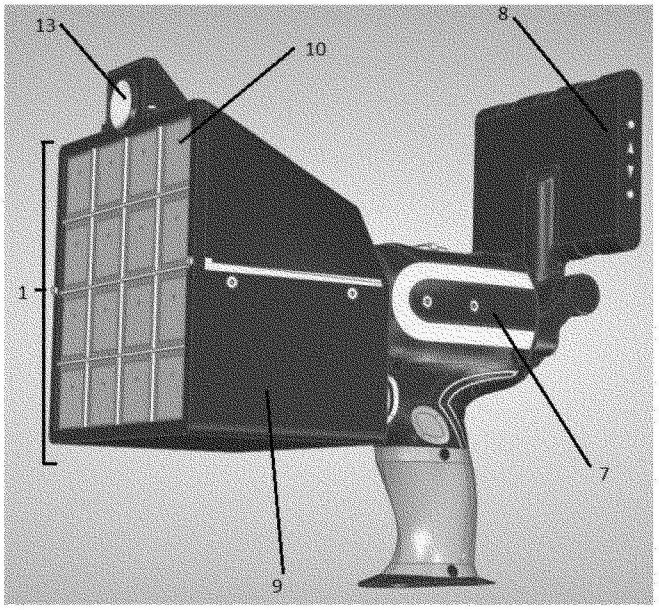

模数转换器
(5)
I2C
5'' HDMI 屏幕
(8)
HDMI
Raspberry Pi 型号 B
(6)
I2C
移相器
(3)
USB
640×480 USB 相机
(13)

图9



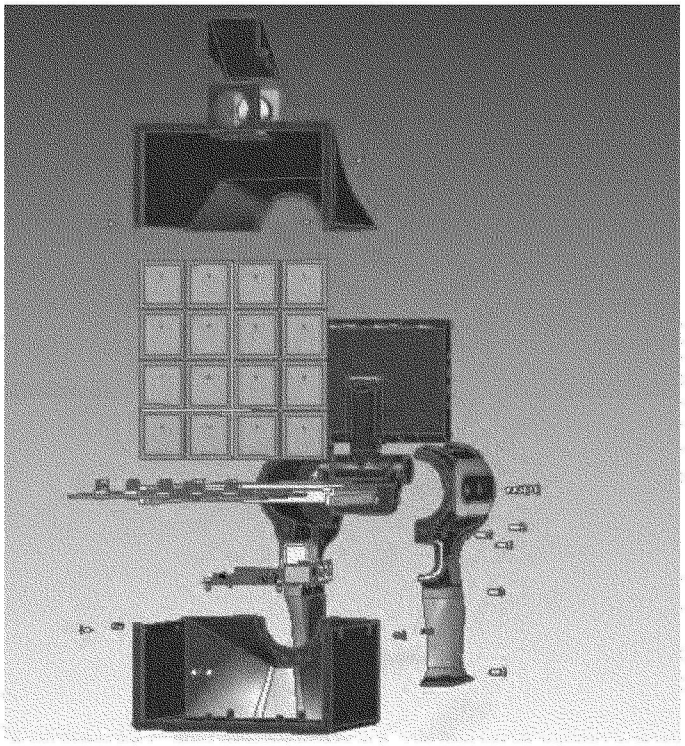


图10