

DISPOSITIVO DE RADIOVISIÓN

Campo de aplicación

5 El campo de aplicación del presente dispositivo, comprende aplicaciones industriales en el rescate de personas atrapadas en avalanchas de nieve o edificios colapsados y la detección y medición de emisiones electromagnéticas de fuentes eléctricas o dispositivos móviles en lugares donde no está permitido su uso, como cárceles o aviones. Por otro lado, esta invención está relacionada
10 con la detección y localización de fuentes RF (Radio Frecuencias) en el rango de telefonía celular y la superposición en una imagen visual en tiempo real.

Descripción del estado del arte previo

15 En general el estado del arte menciona en particular varias patentes que protegen no un sistema físico particular sino un conjunto de dispositivos y métodos que permiten una funcionalidad particular.

Existen una serie de documentos los cuales caen dentro del campo
20 general de las antenas tales como: EP 0 938 156, EP 0 144 867, US 2015/0022421, WO 2016/089464, WO 2011/148248.

Por otro lado, existen una serie de patentes relacionadas del campo del presente desarrollo mencionadas a continuación:

25

EP 0 887 879: Esta patente cae dentro del campo del presente desarrollo y pretende disminuir el tamaño total de una disposición en fase de alta ganancia.

Esto se logra utilizando una estructura multicapa donde los desfasadores son acoplados a las antenas, eliminando el ruteo a ellas. Este dispositivo da una nueva morfología de la disposición en fase.

5 EP 1 921 709: Esta patente corresponde a un sistema capaz de generar un haz doble sin lóbulos de rejilla (grating lobes). También da hincapié a que es un módulo compacto y completo para sintetizar haces.

10 US 7,602,337 B2 (EP2100347): Esta patente corresponde a un disposición en fases emisoras, y los algoritmos que la controlan. Este dispositivo centra su inventiva en su morfología, pero además en sus algoritmos y software.

15 EP 2 463 958: Esta patente corresponde a un arreglo compacto de antenas multi-hazes que optimiza las conexiones sin cable dentro de un recinto cerrado. El arreglo en sí la innovación pues optimiza un diseño antiguo llamado “Van-Atta”.

20 US 6,504,510, EP1294163, US2015/0022421: Esta patente corresponde a una disposición de fases para telecomunicaciones móviles. El arreglo es de $M \times N$, pero solo necesita $M+N$ desfasadores para su funcionamiento, pues utiliza además divisores selectivos de potencia. Con esto es capaz de hacer “el control de tres caminos de haces” (Mover el haz en el azimuth y la elevación).

25 US 6,693,590 B1, EP1195017: Esta patente corresponde a una disposición de fases que utiliza ADCs para implementar desfasadores digitales. Para esto utilizan una conversión reductora a IF. Después la señal es combinada digitalmente. Esto es muy poco eficiente para grandes anchos de banda.

30 PCT/FI2014/050067, EP29511884: Esta patente está ligada a la integración de un arreglo de antenas con sistemas de telecomunicaciones actuales minimizando recursos (cableado) y tamaño. Además de optimizar la recolección de “signal data”.

W0 2008/082917: Esta patente corresponde a una serie de métodos y dispositivos para optimizar o mejorar la formación del haz sintetizado por una disposición de fases.

5

US 2009/0102716: Esta patente corresponde a un sistema con aplicaciones para radares y telecomunicaciones que utiliza un haz para entregar potencia y datos de control.

10

US 2009/0027267: Esta patente corresponde a un reflector de fase variable, la idea es que iluminar el reflector con una bocina y a través de control de fase del reflector cambiar la dirección del haz. El reflector está compuesto por un arreglo de elementos que incluyen entre otros elementos varactores.

15

US 2016/0087349: Esta patente corresponde a un método para controlar el ancho del haz para una banda específica y también para multibanda. El diseño utiliza un prototipo de antena de 4x4 donde antenas de banda simple y multibanda fueron colocadas en ubicaciones específicas dentro del arreglo.

20

US2008/0106467: Esta patente corresponde a una disposición de fases montada completamente sobre un sustrato multicapa, con toda la electrónica correspondiente en él, a su parte trasera. Todo esto de forma compacta donde el tamaño del dispositivo solo queda limitado por el tamaño del sustrato. Este sistema funciona en ambas direcciones.

25

US 2015/0316641: Esta patente corresponde a un transmisor para radares con múltiples salidas, las cuales se puede controlar en fase. Este transmisor está integrado en un solo chip. Además posee un generador RF.

El dispositivo más cercano del estado del arte al presente desarrollo, es la patente EP 2 246 934 A1. Esta patente cae dentro del campo técnico del presente desarrollo y es capaz de detectar y seguir con emisiones activas la dirección de movimiento de paquetes (objetos movibles) utilizando una etiqueta RFID en ellos. Utilizan un arreglo de 2 x 2 elementos, además aseguran que es compacto. Su aplicación está centrada en manejo de inventario mediante tag RF. Requiere de un computador externo para operar. Se protege el dispositivo y los métodos asociados.

10 **Problemas técnicos que resuelve la presente invención**

Como poder generar una imagen integrada en tiempo real basadas en longitudes de ondas de la radiofrecuencia y en el campo del visible, sin excluir otros campos, tales como: el infrarrojo cercano y lejano, el ultravioleta cercano y lejano.

Como poder generar un dispositivo manual, auto-sustentado y portátil que posea la capacidad de generar las imágenes previamente mencionadas.

20 Como poder localizar las fuentes de emisión de radiación de forma pasiva.

Para poder responder a los problemas técnicos previamente planteados, en general, el presente desarrollo incluye:

- 25 • La generación de una imagen RF a tiempo real.

- La utilización de algoritmos de detección de curvas gaussianas para localizar la dirección de las fuentes de radiación.

5

- La utilización de una imagen de video óptica para posicionar la fuente de la radiación en el campo visual mediante la superposición de marcadores que identifican los máximos de radiación de radio.

10

- El conjunto de dispositivos y métodos que permite la operación de forma portable y autónoma.

Descripción general de la invención

El presente desarrollo corresponde a un sistema, auto-energizado, compacto y portable que permite la detección y localización de emisiones electromagnéticas en el espectro utilizado por dispositivos móviles (teléfonos celulares y dispositivos Wi-Fi/Bluetooth) a tiempo real. El principio de detección y localización está basado en la tecnología de arreglos de fase, la cual permite la síntesis de un haz de radiación directivo, al que se le puede controlar su forma y dirección de manera eléctrica, por lo que no posee elementos móviles mecánicos. Estos mecanismos son usados principalmente en aplicaciones militares y astronómicas. El dispositivo incluye además algoritmos de localización y control. Este dispositivo permitirá la detección y localización de emisiones electromagnéticas mediante un barrido del haz de la antena en un campo de visión de 80 x 80 grados. Una vez ocurrida la detección y localización, esta es superpuesta en una imagen visual captada por una cámara de video.

El dispositivo es completamente pasivo, no emite radiación electromagnética en la bandas de detección para su funcionamiento.

Debe entenderse que la presente invención no está limitada a la metodología particular, compuestos, materiales, técnicas de manufactura, usos y aplicaciones aquí descritas, pues éstas pueden variar. También debe
5 entenderse que la terminología empleada aquí es usada con el solo propósito de describir una representación particular, y no intenta limitar la perspectiva y el potencial del presente invento.

Debe notarse que el uso y método, aquí, en el pliego de reivindicaciones
10 y en todo el texto que el singular no excluye el plural, salvo que en el contexto claramente lo implique. Entonces, por ejemplo, la referencia a un “uso o método”, es una referencia a uno o más usos o métodos e incluye equivalentes conocidos por quienes conocen de la materia (el arte). Similarmente, como otro ejemplo, la referencia a “un paso”, “una etapa” o a “un modo”, es una referencia a uno o más
15 pasos, etapas o modos y que puede incluir sub pasos, etapas o modos, implícitos y/o sobrevinientes.

Todas las conjunciones usadas han de entenderse en su sentido menos restrictivo y más inclusivo posible. Así, por ejemplo, la conjunción “o” debe
20 entenderse en su sentido lógico ortodoxo, y no como un “o excluyente”, salvo que el contexto o el texto expresamente lo necesite o indique. Las estructuras, materiales y/o elementos descritos han de entenderse que también se refieren a aquellos equivalentes funcionalmente y así evitar enumeraciones taxativas interminables.

25

Las expresiones usadas para indicar aproximaciones o conceptualizaciones deben entenderse así, salvo que el contexto mande una interpretación distinta.

Todos los nombres y términos técnicos y/o científicos aquí empleados tienen el significado común que le otorga una persona común, calificada en estas materias, salvo indicación expresa, distinta.

5 Los métodos, técnicas, elementos, compuestos y composiciones son descritos aunque métodos, técnicas, compuestos y composiciones similares y/o equivalentes a los descritos pueden ser usados o preferidos en la práctica y/o pruebas de la presente invención.

10 Se incorporan todas las patentes y otras publicaciones como referencias, con el propósito de describir y/o informar, por ejemplo, las metodologías descritas en dichas publicaciones, que puedan resultar útiles en relación con el presente invento.

15 Se incluyen estas publicaciones sólo por su información previa a la fecha de registro de la presente solicitud de patente.

A este respecto nada debe considerarse como una admisión o aceptación, rechazo o exclusión, de que los autores y/o inventores no estén legitimados de
20 serlo, o de estar ante-fechadas dichas publicaciones en virtud de otras anteriores, o por cualquier otra razón.

Para aportar claridad al presente desarrollo se definirán los siguientes conceptos:

25

Antenas: Son elementos capaces de emitir o recibir radiación electromagnética.

Radiación direccional: Una antena no siempre emite o recibe en todas las direcciones. Existen varios tipos de antenas donde es posible concentrar la potencia irradiada en una dirección privilegiada. El parámetro que mide esto es conocido como la ganancia de antena.

Arreglos de fase: Corresponde a un arreglo de antenas distribuidas espacialmente sobre un plano. Cambiando la amplitud y fase de cada una de las antenas que conforman es posible sintetizar un haz de radiación directivo en una dirección específica.

Electrónica de alta frecuencia: Son elementos eléctricos que funcionan en frecuencia de microondas. Estos elementos difieren de los elementos electrónicos convencionales debido a que ya no se tratan como elementos de parámetros concentrados, sino que con parámetros distribuidos, es decir la posición y distancia entre ellos afecta su comportamiento final.

Señal de radio frecuencia: Es una señal que posee una frecuencia en el espectro de radio y microondas. Para el presente desarrollo se trabaja en señales del orden de Gigahertz.

Barrido del haz a alta velocidad: Corresponde al barrido realizado por el arreglo de fase, el cual se realiza sobre toda la superficie de escaneo y en donde se captura la señal de radio frecuencias. Un equivalente general de este barrido es el funcionamiento de las televisiones de rayos catódicos.

Señal de voltaje analógico: Son señales del voltaje continuas que pueden tomar cualquier valor dentro de su rango de funcionamiento. En las señales digitales en cambio existen solamente dos niveles de voltaje para representar 0 y 1.

- 5 Antenas de microcinta: Es un tipo de antena de microondas que puede ser montada sobre una superficie plana. Estas antenas son de fácil construcción y en un rectángulo plano de metal montado sobre un plano a tierra separado por una capa de un material dieléctrico.

10 Descripción detallada de la invención

El dispositivo corresponde a un sistema de detección y localización que se basa en el principio de funcionamiento de la tecnología de arreglos de fase. Esta tecnología permite poder sintetizar un haz de radiación direccional a través de múltiples elementos radiantes (antenas) no direccionales. Este haz de radiación es capaz de recibir y emitir radiación electromagnética. El ancho del haz sintetizado está definido por la separación que existe entre los elementos radiantes individuales. Es posible cambiar la dirección del haz mediante un cambio en la fase de cada uno de los elementos individuales. Para un arreglo lineal de antenas, el cambio de fase entre cada uno de los elementos contiguos del arreglo va dado por la ecuación 1. Donde d es la distancia entre los elementos, λ es la longitud de onda de la señal electromagnética y θ_0 es el Angulo que se desea girar el haz. El cambio en la dirección del haz se muestra en la figura 1/10.

25

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

Ecuación 1

Si las antenas se ordenan en un arreglo bi-dimensional, sin restringir otro tipo de ordenamientos, es posible controlar la dirección del haz en dos dimensiones. El cambio en la dirección del haz estará regido ahora por dos ángulos y su cambio de fase está dado por la superposición en dos ejes.

5

La presente solicitud puede usar un amplio rango de arreglos de antenas, sin excluir ninguna cantidad, un rango de preferencia es el rango de las 64 antenas a un mínimo de 4 antenas, de preferencia 16 antenas. Un arreglo de 4 antenas permite imágenes bidimensionales. En general la variación en el número de elementos afectará la calidad de haz sintetizado, un número mayor de elementos produce una disminución en los lóbulos laterales del haz y la separación entre ellos afecta el ancho del mismo haz.

Para realizar la detección se realiza un barrido del haz a alta velocidad del área que se desea observar, esto permite la creación de una imagen de emisiones electromagnéticas, de preferencia en el rango de los dispositivos móviles y Wifi (fidelidad inalámbrica), específicamente en las bandas 700-900MHz, 1700-2100 MHz, 2400-2700 MHz y 5000 MHz, las cuales posteriormente son superpuestas con una imagen visual digital obtenida mediante una cámara de video. El campo de visión máximo que se pretende observar con esta invención es de 90 x 90 grados.

Dada la naturaleza de la tecnología y la incertidumbre en la intensidad de potencia de la fuente a detectar y localizar, no es posible medir la distancia que existe al objeto emitido, por lo que la detección solo se puede realizar en dos dimensiones en forma instantánea, lo cual significa que la detección se hace sobre la imagen 2D en el plano XY, donde Z sería la profundidad. Es posible hacer una estimación de la distancia máxima de la fuente, más no una medición de la distancia absoluta.

30

El esquema mostrado en la figura 2/10 muestra los diferentes componentes del dispositivo, descritos a continuación:

(1) Arreglo plano de antenas: El arreglo de antenas corresponde a un rango entre los 64 y los 4 elementos radiantes individuales, estos elementos tiene la capacidad de recibir la emisión electromagnética de todas las bandas que se desean detectar (antenas multi-banda, de manera similar a los celulares). Estos elementos están posicionados en diferentes formas con el fin de mejorar la calidad del haz sintetizado, de preferencia se utiliza una distribución regular de elementos equidistantes, tal como en una forma matricial, en una forma circular, entre otras, como se muestra en la figura 3/10. El arreglo de antenas es lo suficientemente compacto para ser parte de un dispositivo portable.

(2) Electrónica de alta frecuencia: La electrónica de alta frecuencia está integrada en una serie de PCBs (Printed Circuit Board). La figura 4/10 detalla la electrónica incluida en el dispositivo.

A continuación se detalla cada grupo de elementos:

Los desfasadores (3) que están en la misma relación a la cantidad de arreglos planos de antenas, van conectados con cada uno de los elementos radiantes del arreglo de antenas (1). Estos se encargaran de cambiar la fase relativa de las ondas electromagnéticas recibidas.

A la salida de cada uno de los desfasadores (3) es combinada (14) en fase para generar una salida única con la información del haz sintetizado. La señal después es amplificada análogamente (11) con un amplificador de bajo ruido y es filtrada opcionalmente de forma análoga con filtros para esa banda (12), para llegar a un detector de potencia total (4), el cual convierte la señal de

radio frecuencia en una señal de corriente directa (DC), esta última es convertida en una señal digital mediante un A/DC (5) (Conversor análogo digital).

La cámara de video (13) se encarga de captar la imagen óptica en la cual se muestran las fuentes de radiación electromagnética detectadas. La cámara es de preferencia, una cámara digital con una resolución superior a 640x480 píxeles.

La posición de esta cámara de video puede ser superior a la zona donde están las antenas, puede estar colocado de manera inferior o colocada en el centro entre las antenas, donde al estar en el mismo eje de simetría del arreglo de antenas, simplifica el algoritmo que superpone la imagen de radio sobre la imagen óptica.

(6) El microcomputador: cumple una serie de funciones importantes, donde la primera es el control y sincronización del arreglo (1), enviando señales eléctricas a cada uno de los desfasadores (3) y de esta manera se puede sintetizar el haz de radiación en la dirección deseada. El microcomputador (6) se encarga de recibir y procesar la señal que viene del detector (4) y de esta forma se construye la imagen RF. Esta imagen es posteriormente post procesada para encontrar la localización de las emisiones de radiofrecuencias. El algoritmo de post procesamiento incluye un detector de gaussianas. Las detecciones son después superpuestas con la imagen proveniente de la cámara de video y son mostradas en la pantalla. El microcomputador (6) se encarga además de ejecutar el programa que permite la interfaz con el usuario.

(7) El suministro de poder: permite que el sistema tenga autonomía propia, energizando la electrónica de radiofrecuencias, la cámara y el microcomputador con su pantalla (8).

(9) Cascara de protección del sistema: todo el sistema está envuelto en un formato tipo cascara que contiene todos los componentes antes mencionados.

5

El algoritmo mencionado previamente presenta el siguiente código fuente:

```

from rv.core import Core

10  if __name__ == '__main__':

    logging.basicConfig(filename='radiovision.log', format='%(asctime)s: %(levelname)s: %(message)s', level=logging.INFO,
    datefmt='%m/%d/%Y %l:%M:%S %p')

    Core().stuff()

    """

15  This class initialize the GUI and controls the acquisition of frames

    """

    class Core(object):

        def __init__(self):

            self.cam = camera.Camera().start()

20      # Load markers images

            import os # this allows to launch 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)

25      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

30      # self.marker_position = Array('i', 2*number_of_gaussians_to_fit)

            self.marker_queue = JoinableQueue(1)

            self.rf_scan_queue = JoinableQueue(10)

            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)

# minimum amplitud value

self.amp_error = 1e-6

5   # capture pictures loop

self.capture = True

self.video_thread = threading.Thread(target=self.start_all)

self.add_pip = False

# GUI Stuff

10  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)

15  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)

20  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')

25  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)

30  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):

35  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.wininfo_height()

    width_1 = heigth_1*self.picture_scale_h_w

    # print('s',width_0, heigth_0)
5    # 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):
10    logging.debug('reset')

    self.rf_scan.new_acc()

def updateValue(self, event):

    if self._job:

        self.root.after_cancel(self._job)
15    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()
20    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)

25    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):
30    self.capture = False

def stuff(self):

    self.video_thread.start()

    self.root.mainloop()

def start_all(self):
35    # start array data and fit

```

```

self.rf_scan.start()

# self.marker_fit.activate_plot()

self.marker_fit.start()

# the rest of the programm

5   # initial values for the positions of the markers

    markers_coords = None

    update_counter = 0

    display_color_map = None

    import cv2

10   picture_wide = 300

    picture_heigth = int(picture_wide * self.picture_scale_h_w)

    time.sleep(1)

    while self.capture:

        frame = self.cam.read()

15   # 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():

20   [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)

25   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:

30   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()

35   image = cv2.cvtColor(frame_aux, cv2.COLOR_BGR2RGB)

```

```

        image = Image.fromarray(image)

        try:

            image = ImageTk.PhotoImage(image)

5         except RuntimeError as e:

            break

        self.video_display.configure(image=image)

        self.video_display.image = image

        # if update_counter > 1000:

10         #   print('none')

        #   markers_coords = None

        #   update_counter = 0

        #   display_color_map = None

        # update_counter += 1

15     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

20     max_value = numpy.max(data)

    data /= max_value

    # this is have a better distribution of colors

    data *= 255

    data = data.astype(numpy.uint8)

25     # 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

30     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()

35     self.rf_scan.shutdown()

```

```

self.marker_fit.shutdown()

time.sleep(5)

while self.cam.thread_update.isAlive():

    pass

5     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):

10     # print('detected amplitudes: ', markers_amp)

    max_amp = max(markers_amp)

    max_amplitud_ration = 0.1

    font = cv2.FONT_HERSHEY_PLAIN

    constante= 1.0

15     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)

20     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:

25                 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],

30                        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))

35     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

5 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):
10 self.thread_update.start()

# self.thread_display.start()

return self

def update(self):

while self.stream.isOpened():
15 if self.stopped:

return

self.grabbed, self.frame = self.stream.read()

self.stream.release()

def display(self): # Currently not in use
20 cv2.imshow('RadioVision', self.frame)

while True:

if self.stopped:

return

cv2.imshow('RadioVision', self.frame)

25 def read(self):

return self.frame

def stop(self):

self.stopped = True

class Array(Process):
30 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)

35 self.Power = [] # the vector version of the measurements

```

```

self.Z = [] # the matrix version of the measurements

# synchronize elements

self.queue = queue # pass the markers to the other process

self.stop_event = Event()

5 self.new_acc_event = Event()

self.new_acc_len_event = Event()

self.MY_SOCKET_PATH = "/home/pi/communication"

self.sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

self.max_hold = numpy.zeros((size_x,size_y))

10 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))

15 try:

    self.sock.connect(self.MY_SOCKET_PATH )

except socket.error as msg:

    sys.stderr.write("bad conection")

    # sys.exit(1)

20 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

25 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

30 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:

35 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:

5         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, :, :]

10        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)

15    self.new_acc_len_event.clear()

def get_scann(self):

    self.sock.sendall(b'scann\n')

    n = 0

    recieve_data = b''

20    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)

25    recieve_data = array(recieve_data)

    return numpy.reshape(recieve_data,(31,31))

def sweep_once(self):

    self.logging_info( 'New measure...')

    t = time.time()

30    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

35    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()

5    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):

10    logging.info('[Array] ' + msg)

    def run(self):

        self.logging_info('Start array.')

        while 1:

            if self.stop_event.is_set():

15                self.logging_info('Stopping array.')

                return

            # do one scann

            if self.new_acc_event.is_set():

                self.reset_acc()

20            if self.new_acc_len_event.is_set():

                self.change_acc_len()

            self.sweep_once()

            try:

                # pass the scan to fit gaussians

25                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):

30    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)

35    y = int((pt1[1] * (1 - r) + pt2[1] * r) + .5)

```

```

    p = (x, y)

    pts.append(p)

    if style == 'dotted':

        for p in pts:

5         cv2.circle(img, p, thickness, color, -1)

        else:

            e = pts[0]

            i = 0

            for p in pts:

10             s = e

                e = p

                if i % 2 == 1:

                    cv2.line(img, s, e, color, thickness)

                    i += 1

15 def drawpoly(img, pts, color, thickness=1, style='dotted', ):

    e = pts[0]

    pts.append(pts.pop(0))

    for p in pts:

        s = e

20         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)

25 def stretch_coords(fitted_coords, punteros):

    """ This functions handles the coordinate

    transmfmrnation to fit the image"""

    new_coordinates = []

    # return new_coordinates

30 # 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)

35 # 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)
5 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)
10 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
15 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)
20 """
        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
25 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):
30 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:
35 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

5    cluster -- set of points
    """

    (d,) = cluster[0].shape
    mean = numpy.zeros((d,))
    total = 0

10    # for cluster in clusters:
    for point in cluster:
        mean += point
        total += 1

    mean /= total

15    covar = numpy.zeros((d,d))
    # for cluster in clusters:
    for point in cluster:
        val = point - mean
        covar += numpy.dot(val[None,:].T, val[None,:])

20    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

25    area = sqrt(area)

    # print(area, 'area')

    density = total / area

    # print(density, 'density')

30    if density >= self.density_threshold:
        means.append(mean)
        covars.append(covar)

        return True

    return False

35 def detect_number_of_clusters(self, data_set):#, m_near = 18):

```

```

""" this function applies a preprocessing of the data

to try to estimate the number of clusters available.

Keyword arguments:

data_set -- a two dimensional array that contains the coordinates of the clusters
5   m_near -- the number of neighborhood that are considered to rank the points (default 15)
    alfa -- the confidence interval, (1 - alfa) * 100 % (default 0.05)
"""

alfa = self.alfa
remaining_points = data_set.copy()
10 detected_clusters = []

cont_fig = 3
plot_all = False
m_near = 10

while 1:
15   points_size, dimension = remaining_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:
20       logging.warning('possible_problem')

       # Estimate the ranks from the points

       cont_main = 0

       for y_p, x_p in remaining_points:

           cont_aux = 0
25       for y_aux, x_aux in remaining_points:

           # print(cont_aux)

           if cont_main == cont_aux:

               cont_aux += 1

               continue
30       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')
35       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

5     near_distance_index[cont_main, cont] = cont_aux

    break

    cont_aux += 1

    cont_main += 1

ranks = numpy.sum(near_distance, axis=1)

10    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)

15    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)

20    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()

25        choosen_point -= mean

        # check if point is inside hyper-cube

        points_ = choosen_point[:,None]

        evaluate_ellipsoid = numpy.dot(numpy.dot(points_.T, numpy.linalg.inv(alfa_confidence_covar)), points_)

        if evaluate_ellipsoid <= 1:

30            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)

35    new_cluster_points = []

```

```

for index in new_cluster_points_index:

    # delete one point at a time

    new_cluster_points.append(remaining_points[index, :])

    remaining_points = numpy.concatenate((remaining_points[0:index, :], remaining_points[index+1:, :]))

5   points_size, dimension = remaining_points.shape

    # self.plot_points(remaining_points, new_cluster_points, cont_fig)

    cont_fig += 1

    detected_clusters.append(new_cluster_points)

    # if there are enough points do one more iteration

10  # print(points_size)

    if points_size <= m_near :

        break

    cluster_amount = 0

    means = []

15  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')

20  return self.fusse_near_clusters(means, covars)

def make_markers(self, mean, data):

    markers = [[], [], []]

    for x,y in mean:

        markers[0].append(x)

25  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))

30  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)

35  pyplot.cla()

```

```

pyplot.clf()

pyplot.imshow(points)

pyplot.draw()

# pyplot.show(block=False)

5 def fusse_near_clusters(self, means, covars):

    NN = len(means)

    mm = means.copy()

    meged = []

    final_means = []

10 # print('in')

    cont = 0

    while cont < NN:

        if cont in meged:

            cont += 1

15         continue

        # print('first thing, mean')

        # print(means[cont])

        merge_count = 1

        mean = means[cont].copy()

20 for cont1 in range(cont + 1, NN):

        if cont1 in meged:

            continue

        covar_sum = covars[cont] + covars[cont1]

        distance = means[cont] - means[cont1]

25 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])

30 #

        # print('distance', distance)

        if distance <= 40:

            # print('merge')

            mean += means[cont].copy()

35 merge_count += 1

```

```

        meged.append(cont1)

        final_means.append(mean/merge_count)

        cont += 1

        # print('out')
5         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
10        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.
15        self.queue_scan = queue_scan

        self.queue_markers = queue_markers

        self.stop_event = Event()

        self.display_rf_scan = False

    def activate_plot(self):
20        self.display_rf_scan = True

    def deactivate_plot(self):

        self.display_rf_scan = False

    # def fit(self, data):

    #     it = time.time()

25    #     # 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])
30    #

    #     Xpos = array([(x1), (x2), (x3)])

    #     Ypos = array([(y1), (y2), (y3)])

    #     orden = Amps.argsort()

    #

35    #     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

5 #

# return self.markers.copy()

def log_function_info(self, msg):

    logging.info('[Detect] ' + msg)

def run(self):

10     self.log_function_info('Start detect.')

    from matplotlib import use

    use('TkAgg')

    from matplotlib import pyplot

    pyplot.ion()

15     detect = DetectGaussias()

    while 1:

        if self.stop_event.is_set():

            self.log_function_info('Stopping detect.')

            return

20     data = self.queue_scan.get()

    self.queue_scan.task_done()

    self.log_function_info('Data recieved.')

    # the medium filter needs

    mask = data

25     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)

30     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

    mask[index_255] = 0

35     # data[index_0] = 255

```

```

# data[index_255] = 0

# end watershed

resampling = detect.sample_from_data(data)

means = detect.detect_number_of_clusters(resampling)

5 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

10 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.')

15 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")

20 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)

25 ax.add_patch(circ3)

    pyplot.draw()

    pyplot.show()

    pyplot.pause(0.1)

def shutdown(self):

30 self.stop_event.set()

def posiciones(minima, maxima):

    x1 = 0

    x2 = 0

    x3 = 0

35 y1 = 0

```

```

y2 = 0

y3 = 0

dist1 = 0

dist2 = 0

5   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)

10   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)

15   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

20   y = 0

    maxi = 0.

    i = 0

    j = 0

    aux = data.reshape(90, 90)

25   while i < 89:

        while j < 89:

            if aux[i][j] > maxi:

                x = i

                y = j

30         maxi = aux[i][j]

        j += 1

        j = 0

        i += 1

    return maxi, y, x # data.reshape seems to flip axis

35 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):
5        if values[cont1] < values[cont1+1]:

            tmp = values[cont1]

            values[cont1] = values[cont1+1]

            values[cont1 + 1] = tmp

            tmp = index[cont1]
10        index[cont1] = index[cont1 + 1]

            index[cont1 + 1] = tmp

            change = True

    if not change:

        break
15    return index

class EstimateMeanCovariance(object):

    def __init__(self, alfa = 0.05):

        self.alfa = alfa

    def get_lambda_2(self, lambda_t, lambda_s, radio):
20        err = 1e-2

        lambda_s1 = 0

        lambda_s0 = lambda_t * 0.1

        try:

            cont = 0
25            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 )

30            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:
35                    return lambda_s1

```

```

        lambda_s0 = lambda_s1

        cont += 1

        # if (cont % 1000) == 999:

        #     print('*',)

5         # 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,))

10    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

15    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)

20    # 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

25    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

        # print('start mean estimation')/

30    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

35    # initial estimation lambda square

```

```

lambda_0 = numpy.ones((dimension,))*radio

sqrt_chi_p_alpha = sqrt(chi2._ppf(1-self.alfa,2))

for cont in range(dimension):

    lambda_0[cont] /= sqrt_chi_p_alpha
5    # calculate initial hyperrectangle axes

    hyper_rectangle_thing = numpy.zeros((dimension, dimension,))

    previus_covariance = covar_matrix.copy()

    prev_error = 0

    for watch_dog in range(150):
10        for cont in range(dimension):

            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):
15                choosen_point = data[cont, :].copy()

                choosen_point -= x_center

                # check if point is inside hyper-cube

                is_in = True

                for cont1 in range(dimension):
20                    # 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):
25                        is_in = False

                        break

                if is_in:

                    points_in.append(cont)

            # read the documentation on numpy linalg svd
30            x_center, covar_matrix, U, s, V = self.estimate_mean_covar_u_s_v(data, dimension, points_in)

            # estimate the values of the norma distribution usin

            # a truncated distribution , see result 2.2 and 2.1

            radio_ = numpy.zeros((dimension,))

            for cont in range(dimension):
35                radio_aux = hyper_rectangle_thing[cont,:]

```

```

        radio_[cont] = sqrt(numpy.dot(radio_aux,radio_aux))

o,ss,oo = numpy.linalg.svd(previous_covariance)

for cont in range(dimension):

    if isnan(s[cont,cont]):

5         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)

10    # estimate the region of 1-alfa confidence

    radio_constant = chi2._ppf(1 - self.alfa, 2) / dimension

    # se the points that are inside this last region

    region_b = []

    A_matrix = numpy.dot(numpy.dot(U, s * radio_constant), V.T)

15    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

20        points_ = choosen_point[:,None]

        evaluate_ellipsoid = numpy.dot(numpy.dot(points_.T, numpy.linalg.inv(A_matrix)), points_)

        if evaluate_ellipsoid <= 1:

            region_b.append(cont)

    x_center, covar_matrix, U, s, V = self.estimate_mean_covar_u_s_v(data, dimension, region_b)

25    error = numpy.sum(numpy.abs(covar_matrix - previous_covariance))

    if error < 0.1:

        break

    if abs(error - prev_error) < 1e-6:

        break

30    prev_error = error

    U, s, V = numpy.linalg.svd(covar_matrix)

    for cont in range(dimension):

        lambda_0[cont] = sqrt(s[cont])

        previous_covariance = covar_matrix.copy()

35    # 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)

5       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)

10      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)

    def sample_data_gen(self, data, dist_val, size_x = 30, size_y = 30):

15      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)]

20      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:

25      idx_y = random_integers(0, data.size - 1)

            offset = self.float_random(dist_val)

            try:

                if offset <= data[idx_y]:

                    return idx_y

30      except:

                logging.ERROR(data.shape,'fail to sample',data.size)

                return 0

    def float_random(self, ends):

        return random()*ends

35  ADS_CONV_REG = 0x00

```

```

ADS_CONF_REG = 0x01

ADS_LTHR_REG = 0x02

ADS_HTHR_REG = 0x03

ADS_ADDR_GND = 0x90
5  ADS_ADDR_VDD = 0x92
   ADS_ADDR_SDA = 0x94
   ADS_ADDR_SCL = 0x96
   ADS_READ = 0x01
   ADS_WRITE = 0x00
10  ADS_OS = 0x8000
    ADS_CONV_CURR = 0x0000
    ADS_CONV_NCURRE = 0x8000
    ADS_NOFX = 0x0000
    ADS_CONV_BEGIN = 0x8000
15  ADS_MUX = 0x7000
    ADS_AIN0_AIN1 = 0x0000
    ADS_AIN0_AIN3 = 0x1000
    ADS_AIN1_AIN3 = 0x2000
    ADS_AIN2_AIN3 = 0x3000
20  ADS_AIN0_GND = 0x4000
    ADS_AIN1_GND = 0x5000
    ADS_AIN2_GND = 0x6000
    ADS_AIN3_GND = 0x7000
    ADS_PGA = 0x0E00
25  ADS_6144 = 0x0000
    ADS_4096 = 0x0200
    ADS_2048 = 0x0400
    ADS_1024 = 0x0600
    ADS_0512 = 0x0800
30  ADS_0256 = 0x0A00
    ADS_MODE = 0x0100
    ADS_CONT_CONV = 0x0000
    ADS_SING_CONV = 0x0100
    ADS_DATA_RATE = 0xE0
35  ADS_8SPS = 0x00

```

```

ADS_16SPS = 0x20

ADS_32SPS = 0x40

ADS_64SPS = 0x60

ADS_128SPS = 0x80
5  ADS_250SPS = 0xA0

ADS_475SPS = 0xC0

ADS_860SPS = 0xE0

ADS_COMP_MODE = 0x10

ADS_COMP_TRAD = 0x00
10 ADS_COMP_WIND = 0x10

ADS_COMP_POL = 0x08

ADS_COMP_LOW = 0x00

ADS_COMP_HIGH = 0x08

ADS_COMP_LAT = 0x04
15 ADS_COMP_NLATCH = 0x00

ADS_COMP_LATCH = 0x04

ADS_COMP_QUE = 0x03

ADS_COMP_ONEC = 0x00

ADS_COMP_TWOC = 0x01
20 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 */
25 #include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

#include <fcntl.h>
30 #include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/mman.h>

#include <sys/stat.h>
35 #include <sys/un.h>

```

```

#include <netinet/in.h>

#include <unistd.h>

#include <math.h>

#include <wiringPiI2C.h>
5  #include "cnpy.h"

#include "ADS1115.h"

#include <iostream>

#include <time.h>

/*socket related define*/
10 #define MY_SOCKET_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)
15 #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)
20 #define MEAN_ITER 2

#define WAIT_T 3000

#define BCM2708_PERI_BASE    0x3F000000

#define GPIO_BASE            (BCM2708_PERI_BASE + 0x2000000) /* GPIO controller */

#define WAIT_ADC 500000
25 using namespace std;

int mem_fd;

void *gpio_map;

struct timespec tim, tim2;

// I/O access
30 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))
35 #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

5  /*functoins*/

void setup_io();

void send_data(double* val, unsigned char addr);

void read_from_pipe (int file);

void write_to_pipe (int file);

10 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;

15 void error(const char *msg)

{

    perror(msg);

    exit(1);

}

20 int handleClientRequest(int clientFD)

{

    char buffer[256];

    int n;

    /*Scanc configureations*/

25     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      *

30     * 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*/

35     while(0==0)

```

```

{

    bzero(buffer,256);/*clearing the buffer*/

    n = read(clientFD,buffer,255);/*reading from the client connectin*/

    if (n < 0) /*error cheking*/
5        {

            error("ERROR reading from socket");

        }

    /*recieve a close socket message*/

10    if(strcmp(buffer,"close\n") == 0 || strcmp(buffer,"") == 0)

        {

            return 0;

        }

    if(strcmp(buffer,"scann\n") == 0)
15    {

        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*/
20                }

            }

        }

    return 0;

25 }

/*scan functiond*/

void setup_io();

void send_data(float angle, unsigned char addr);

/*main function, here the socket is configured and start to
30 * run, waiting a request.*/

int main(int argc, char *argv[])

{

    int sockfd, newsockfd;

    socklen_t clientLen;

35    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");
5   }

/*initialize address structure*/

memset(&serv_addr, 0, sizeof(struct sockaddr_un)); /* Clear structure */

serv_addr.sun_family = AF_LOCAL;

strncpy(serv_addr.sun_path, MY_SOCKET_PATH, sizeof(serv_addr.sun_path) - 1);
10  unlink(serv_addr.sun_path);

/*start bind*/

if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(struct sockaddr_un)) == -1)
{
    handle_error("bind");
15  }

/*set the number of incoming connections*/

if (listen(sockfd, LISTEN_BACKLOG) == -1)
{
    handle_error("listen");
20  }

clientLen = sizeof(struct sockaddr_un);

/*Configure PI ports and phase shifter values*/

// Set up gpi pointer for direct register access

    setup_io();
25    fd = wiringPiI2CSetup(0x54);

    /*wiringPiI2CWriteReg16(fd, 0x02, 0x20);*/

    wiringPiI2CWriteReg16(fd, 0x02, 0x20);

    INP_GPIO(18);

    INP_GPIO(23);
30    INP_GPIO(24);

    INP_GPIO(17);

    INP_GPIO(27);

    INP_GPIO(22);

    INP_GPIO(5);
35    INP_GPIO(6);

```

```

INP_GPIO(13);

INP_GPIO(26);

OUT_GPIO(18);

OUT_GPIO(23);

5  OUT_GPIO(24);

OUT_GPIO(17);

OUT_GPIO(27);

OUT_GPIO(22);

OUT_GPIO(5);

10 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);

15 double dt = 0;

double dp = 0;

double dt2 = 0;

double dp2 = 0;

double dt3 = 0;

20 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++)

25     {

        dt = theta_array[i];

        dp = theta_array[j];

        dt2 = 2*dt;

        dp2 = 2*dp;

30        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;

35        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;
angle[i][j][7] = (int(360 + 360 + 360 + 0 + dp2)%360)*255.0/360;
5   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;
10  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;

    }

}

15  /*change the permissions on the unix socket, to allow access without sudo*/
    chmod(MY_SOCKET_PATH, 0666);

    while (true)
    {
        printf("Waiting request...\n");
20  newsockfd = accept(sockfd, (struct sockaddr *) &client_addr, &clientLen);

        if (newsockfd == -1)
        {
            handle_error("accept");
        }

25

        if (handleClientRequest(newsockfd) == 1)
        {
            break;
        }

30  }

    printf("by\n");

    unlink(MY_SOCKET_PATH);

    return 0;

}

35  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;

5    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);

10    printf("Maximum angle = %d\n", MAX_ANG);

    printf("Step angle = %f\n", STEP_ANG);

    for(int i = 0; i < SIZE_N; i++)
    {
        for(int j = 0; j < SIZE_N; j++)

15        {

                val[0] = angle[i][j][0];

                val[1] = angle[i][j][1];

                val[2] = angle[i][j][2];

                val[3] = angle[i][j][3];

20                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);

25                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];

30                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;

35                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;

5         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;

    }

10 }

    printf("Time taken: %.2fs\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);

    //const unsigned int shape[] = {SIZE_N, SIZE_N};

    //cnp::np::save("arr1.npy", data, shape, 2, "w");

    //printf("saved\n");

15 printf("Time taken: %.2fs\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);

    return 0;

}

//

// Set up a memory regions to access GPIO

20 // setup_io

void setup_io()

{

    /* open /dev/mem */

    if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0)

25 {

        printf("can't open /dev/mem \n");

        exit(-1);

    }

    /* mmap GPIO */

30 gpio_map = mmap(

        NULL,          //Any address in our space will do

        BLOCK_SIZE,    //Map length

        PROT_READ|PROT_WRITE, // Enable reading & writing to mapped memory

        MAP_SHARED,     //Shared with other processes

35 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) {
5     printf("mmap error %d\n", (int)gpio_map); //errno also set!
    exit(-1);
}

// Always use volatile pointer!

gpio = (volatile unsigned *)gpio_map;
10 }

void send_data(double* val, unsigned char addr)
{
    // unsigned char opt = (int) 0;

    uint8_t data[8];
15     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];
20     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);
25     // float angle = strtod(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;
30         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;
35         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;

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;

10      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);

15 // 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;

20 for(int j = 0; j < WAIT_T; j++) ((void)0);

// Address bit 0

    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);

25 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);

30 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

35 GPIO_CLR = (1<<24)|(1<<23)|(1<<5)|(1<<27)|(1<<13);

```

```

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);

5 // 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;

10 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;

15 }
```

Descripción breve de las figuras

Figura 1/10

20

Esta figura describe el patrón de radiación sintetizado proveniente de 4 elementos radiantes (antenas). El patrón se encuentra con una inclinación de 16 grados proveniente de los desfases relativos entre las antenas.

25 (1) Arreglo plano de antenas.

(11) Antena o elemento radiante

Figura 2/10

Esta figura presenta un diagrama de bloques general de la invención.

- (1) Arreglo plano de antenas.
- (2) Electrónica de alta frecuencia
- 5 (3) Desfasadores
- (4) Detector de potencia total
- (5) Conversor análogo digital (A/DC)
- (6) Microcomputador
- (7) Suministro de Poder
- 10 (8) Pantalla
- (9) Carcasa de Protección del dispositivo
- (10) Antena o elemento radiante
- (11) Amplificador
- (12) Filtrador
- 15 (13) Cámara de video.
- (14) Combinador

Figura 3/10

- 20 Esta figura representa la disposición geométrica de los elementos radiantes, ordenados en forma matricial, sin excluir otras disposiciones.

(10) Antena o elemento radiante

Figura 4/10

5 Esta figura presenta un diagrama de bloques general de la placa con la electrónica de alta frecuencia.

(1) Arreglo plano de antenas.

(3) Desfasadores

(10) Antena o elemento radiante

10 (11) Amplificador

(12) Filtrador

(14) Combinador

Figura 5/10

15

Esta figura presenta un modelo de un arreglo de antenas de 4x4 implementado en antenas de microcinta para la banda de telefonía PCS. (Escala en mm)

(1) Arreglo plano de antenas.

20 (10) Antena o elemento radiante

Figura 6/10

Esta figura presenta el ejemplo de aplicación de 16 antenas, con amplificadores y salidas, además cuenta con su modelo de placa de electrónica de alta frecuencia usando componentes comerciales, sin ser restrictiva a solo este tipo de componentes.

5

(1) Arreglo plano de antenas.

(2) Electrónica de alta frecuencia

(3) Desfasadores

(4) Detector de potencia total

10 (5) Conversor análogo digital (A/DC)

(6) Microcomputador

(7) Suministro de Poder

(10) Antena o elemento radiante

(11) Amplificador

15 (12) Filtrador, opcional

(14) Combinador

Figura 7/10

20 Esta figura presenta específicamente, el esquema de las antenas del ejemplo de aplicación 1 a 4 usando componentes comerciales, sin ser restrictiva a solo este tipo de componentes. (Los esquemas de las antenas 5 a 8, 9 a 13 y 14 a 16, son similares en su distribución)

25 (3) Desfasadores

- (7) Suministro de Poder
- (10) Antena o elemento radiante
- (14) Combinador

5 Figura 8/10

Esta figura presenta el esquema de una segunda placa usada para convertir la potencia medida en voltaje analógico.

- 10 (4) Detector de potencia total
- (7) Suministro de Poder

Figura 9/10

- 15 Esta figura presenta el diagrama de conexión del microcomputador con la cámara, la pantalla, los desfasadores y el A/DC.

- (3) Desfasadores
- (5) Conversor análogo digital (A/DC)
- 20 (6) Microcomputador
- (8) Pantalla
- (13) Cámara de video.

Figura 10/10

Esta figura presenta en su parte superior un modelo del sistema implementado.

(1) Arreglo plano de antenas.

(7) Suministro de Poder

5 (8) Pantalla

(9) Carcasa de Protección del dispositivo

(10) Antena o elemento radiante

(13) Cámara de video.

10 El esquema inferior presenta un despiece del sistema donde se incluyen el arreglo de antenas, la cámara óptica, la carcasa de protección, la pantalla, entre otros elementos.

Ejemplo de aplicación

15

Un ejemplo de aplicación de este dispositivo sin restringir sus componentes, se implementa utilizando un arreglo de 16 antenas en microcinta, dos placas de electrónica de alta frecuencia con componentes comerciales y un microcomputador comercial.

20

Imágenes del diseño de este dispositivo se muestran en las figuras 6/10, 7/10 y 8/10.

25 Las antenas de microcinta corresponden a antenas rectangulares sintonizadas a 1,88 GHz y con un ancho de banda aproximado de 50 MHz. (Figura 5/10)

La primera placa electrónica de alta frecuencia posee 16 módulos vectoriales HMC631LP3 de Analog®, los cuales permiten cambiar la fase y amplitud de la señal recibida por cada una de las antenas. Estos moduladores son controlados con potenciómetros digitales DS3930 de Maxim®, lo cuales se comunican con un bus I2c. La señal es después combinada en fase mediante 14 combinadores de potencia TCP-2-272 de Minicircuits®. La señal posteriormente es amplificada con un amplificador de bajo ruido (LNA) modelo MAAL-007304 de MACOM®.

10

La segunda placa posee un detector de potencia modelo LT5538 de Linear® y es usado para convertir la potencia medida en un voltaje analógico.

El conversor análogo digital usado es el ADS1115, el cual se compró comercialmente con su placa de prueba. Este último es conectado con una microcomputador Raspberry® pi 3.

15

La incorporación de filtros es opcional en este dispositivo, pues las antenas solo reciben una banda de frecuencia.

20

Por otro lado, la cámara digital óptica, utilizada en este ejemplo prototipo fue de 800 x 600 mm. En el ejemplo presentado en la figura 10/10, se ve la cámara óptica posicionada sobre el arreglo de antenas, también es de preferencia, posicionar la cámara en el centro de las antenas ya que esta posición simplifica los cálculos para la acción de la cámara de radiofrecuencia.

25