PRÁCTICA PROFESIONAL SUPERVISADA

Visualización y Adquisición de Datos de sensor Ultrasónico

Fernando Emanuel ELIZONDO
TUTORES
Dr Ing Luis CANALI
Ing Gonzalo PEREZ PAINA

INTRODUCCIÓN

Una de las tareas más importantes de los sistemas autónomos es adquirir conocimiento de su entorno, lo que se logra mediante la utilización de sensores de diferentes tipos. En el caso de robótica móvil se utilizan una gran variedad de sensores, ya sea para obtener información interna del robot o externa del entorno, los cuales se puede clasificar como propioceptivos/exteroceptivos y activos/pasivos.

Los sensores propioseptivos miden valores internos del robot, por ejemplo la velocidad de los motores de tracción, carga en las ruedas, tensión de las baterías, etc. Los sensores exteroceptivos adquieren información del entorno del robot, por ejemplo medición de distancia, intensidad de luz, etc. Para obtener una información útil del entorno el robot debe "interpretar" la información de los sensores exteroceptivos. Los sensores pasivos miden la energía que proviene del entorno; ejemplos de sensores pasivos son temperatura, micrófono, cámaras CCD o CMOS, etc. Los sensores activos emiten energía al entorno, y miden como regresa esa energía al robot alterada por el entorno. Puesto que estos sensores permiten controlar la interacción con el entorno, generalmente permiten mejor desempeño, aunque la señal puede sufrir interferencia; ejemplos de sensores activos son sensores de ultrasonido, láser, etc.

Se plantea en consecuencia la necesidad de equipar el robot móvil de tracción diferencial RoMAA (Robot Móvil de Arquitectura Abierta) con sensores de uso común en robótica necesarios para obtener información del entorno. Estos pueden ser anillos de sensores de ultrasonido, unidad inercial creada a partir de acelerómetros y giróscopos tipo MEMS, compás electrónico, etc. Para lo cual es necesario realizar un estudio de las diferentes tecnologías disponibles de cada sensor, características, tipo de interfaz, disponibilidad en el mercado y costo. Finalmente, se debe construir la electrónica necesaria para la interfaz con cada sensor, además de la fuente de alimentación para cada sensor, e integrarlos bajo un mismo controlador embebido basado en microcontrolador ARM7 de 32bits que permite comunicarse con la computadora a bordo del robot.

DESARROLLO

Hoy en día los robots móviles están en laboratorios de investigación, así como en la industria, hospitales, museos,y un tema importante es tener un robot móvil potente y fiable. Desafortunadamente la investigación robótica móvil en Argentina es mayormente, hacer uso de robots móviles importados. Sin embargo, estos robots tiene varias desventajas: 1) que son caros, 2) mantenimiento es lento y caro, y 3) normalmente ellos utilizan sistemas cerrados, sin flexibilidad para agregar nuevas capacidades.

Para superar estos inconvenientes, se desarrolla un: abierto, flexible, diferencial, fiable, potente y ampliable móvil robot, diseñado y construido localmente. Se basa en el sistema operativo Linux y hardware comercial para sensores y actuadores. Todo el software requerido por el robot ya estaba parte de Linux, o que se ha desarrollado utilizando herramientas bajo la licencia GNU Licencia Pública General. En otras palabras, no hay código propietario.

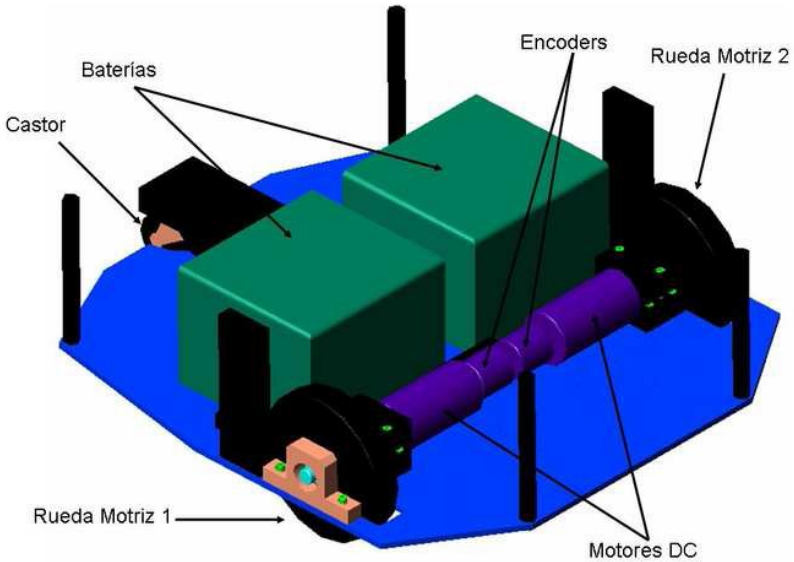
El robot móvil utiliza dos motores con controlador de los dos ruedas del robot móvil y sus respectivos encoders. Ahora bien, este artículo presenta los detalles de un anillo de la ultrasónico diseñado y construido como una parte de nuestro móvil del robot. Estos sensores aumentan la funcionalidad del robot para detectar obstáculos cercanos al mismo.

RoMAA - Robot Móvil de Arquitectura Abierta

Es una plataforma móvil de arquitectura abierta. Desarrollado en el CIII (Centro de Investigaciones Informáticas para la Ingeniería). El objetivo principal de su proposición como proyecto fue disponer de un vehículo adecuado para el ámbito de la investigación capaz de adaptarse a distintos experimentos en el marco de la robótica móvil y visión por computadora. Se obtuvo así un robot de característica modular en el que resulta sencillo agregar diferentes tipos de sensores y actuadores.

Desde el punto de vista mecánico se considero importante poder acceder de forma fácil y rápida a los distintos componentes del sistema, como las baterías/cargador, los motorreductores, encoders, y drivers de potencia para el control de tracción del vehículo. Esto permite reemplazar componentes en forma ágil ya sea por avería, falla y para realizar algún experimento particular que requiere ajustar ciertos componentes de la plataforma. Los paneles superiores del vehículo se retiran fácilmente para acceder de manera individual al sistema de energía o a la tracción del vehículo.

Desde el punto de vista eléctrico/electrónico el bus de alimentación es de tensión estándar de 12V, permitiendo conectar distintos tipos de dispositivos. Los componentes internos se comunican por medio un bus RS-485 sobre el que se implementa el protocolo de comunicaciones MODBUS. La figura muestra un diagrama en bloques del vehículo.

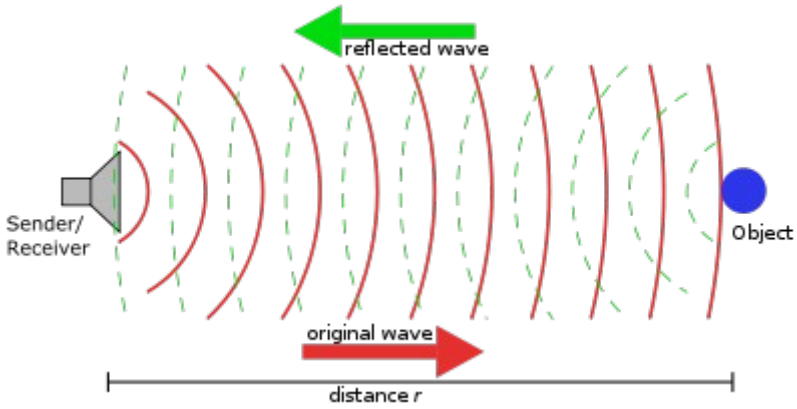


SENSORES

El semianillo de sonar esta compuesto de 9 transductores SensComp/Polaroid t de Grado Medio Ambiente y un módulo controlador serie 6500-Ranging Polaroid, tienen un rango de medición efectiva desde aproximadamente 4 pulgadas a 30 pies, más de un campo de visión de aproximadamente 15 grados.

En la figura a continuación podemos muestra el principio de funcionamiento de sensores ultrasónicos, o sonares. En un primer paso, el transductor ultrasónico emite un sonido de alta frecuencia y, a continuación; en un segundo paso, el sensor espera a que un eco y el mismo transductor ahora funciona como un micrófono para escuchar los

ecos. La distancia a la objeto más cercano puede ser calculado a partir de el tiempo transcurrido entre la emisión de sonido y la recepción del primer eco.



Serie 600 - Sensor inteligente de grado

Descripción

El nuevo Sensor Inteligente SensComp esta basado en la línea de transductores ultrasensibles electrostáticos de la serie 600 con una versión mejorada de los módulos 6500. La nueva circuitería de voltaje regulado permite al sensor operar desde 6 a 24 Vdc.

La salidas compatibles TTL de colector abierto incluyen resistencias pull-up. La configuración del nuevo oscilador permite a la unidad ser externamente disparada o sensar continuamente a 5 Hz.

La ganancia controlada digitalmente y el amplificador de ancho de banda variable minimiza el ruido y la detección de lóbulos laterales en aplicaciones de sonar. La exactitud absoluta típica es ±1% de la lectura sobre todo el rango.

Características Sensor

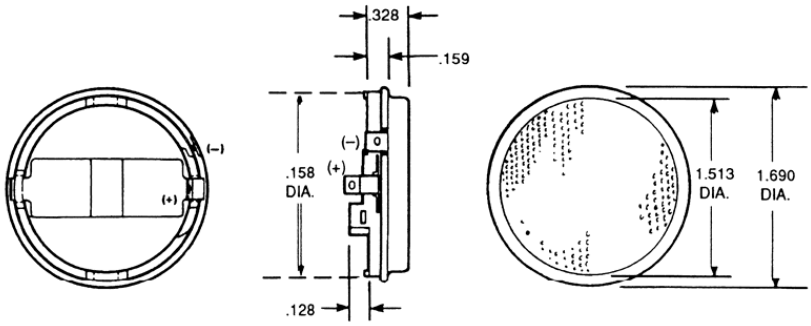
- Sensor de 50KHz
- Compatible TTL
- Operación en modo monoestable y astable

- Voltaje regulado
- Ángulo de apertura 15° a -6dB
- Excelente sensibilidad de recepción
- Específicamente para operación en aire a frecuencias de ultrasonido
- Diseño compacto
- Fácil interfase con mínimos componentes externos
- Puede ser disparado internamente o externamente
- Puede operar con fuente de voltaje de 6 a 14 Vdc
- Rango de Distancia 0,15 a 17 m

Aplicaciones

- Mediciones de nivel
- Detección de Proximidad
- Detección de Presencia
- Robótica
- Productos Educativos

Dimensiones:



Especificaciones:

Sensibilidad de Transmisión..... 110dB min

a 50kHz; 0dB re 20uPa a 1m

(300VACpp, 150VDC polarización)

Sensibilidad de Recepción..... -42dB

a 50kHz; 0dB = 1volt/uPa

(150VDC polarización)

Rango de Distancias..... 15 cm a 10,7m

Resolución..... ± 3mm a 3m

Peso..... 8,2gm

Tensión de Polarización sugerida..... 200V

Tensión de Control Alterna sugerida..... 200Vpico

Voltaje combinado..... 400Vmax

Capacidad a 1kHz (típico)..... 400 - 500pf

(a 150VDC polarización)

Temperatura de trabajo..... -30 a +70°C

Dimensiones

Ancho..... 11,684mm

Diámetro..... 42,926mm

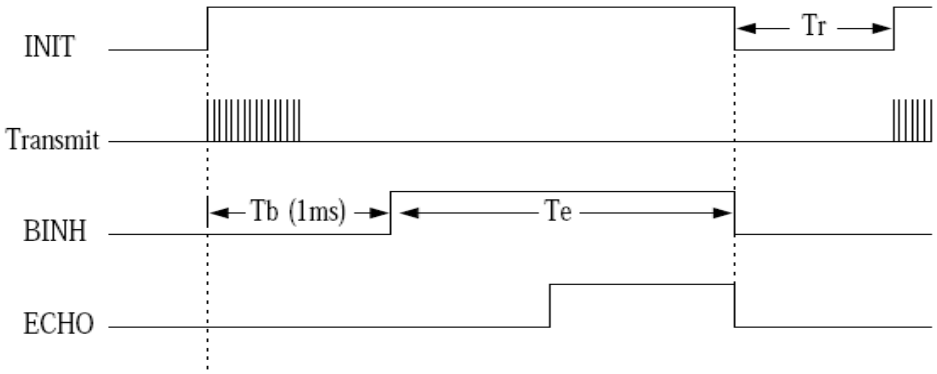
OPERACIÓN DEL SONAR

La serie 6500 es modulo sonar económico que puede manejar todos los transductores electrostáticos SensComp/Polaroid y operan sobre rango de alimentación de 4,5 a 6,8 Voltios. Este módulo, con una simple interfase, es capaz de medir distancias desde 0.1524 m(6 pulgadas) a 10,668 m (35 pies). La exactitud absoluta típica es ±1% de la lectura sobre el rango total. La salida de transmisión del sonar es de 16 ciclos a 49,4 Kilohertz.

Hay 2 modos básicos de operación para el modulo sonar 6500: Eco-Simple y Múltiple-Eco. Cuando INIT es puesto en alto, se activa la salida al transductor. 16 pulsos a 49,4kHz con 400 voltios de amplitud excitaran el transductor y ocurre la transmisión. En el final de la transmisión de los 16 pulsos, permanece una tensión continua de 200 voltios como se recomienda para una operación óptimo. Para eliminar la oscilación de los transductores en la detección como en el retorno de señal, la entrada de la

recepción del CI del modulo de control es deshabilitada internamente por 2,38 milisegundos después de la señal de inicialización. Si se desea reducir el tiempo de deshabilatación , la entrada BINH puede ser puesta en alto para terminar el blanqueo de la entrada de recepción en cualquier tiempo previo al blanqueo interno. Esto puede desearse para detectar objetos mas cercanos que 1,33 pies q corresponden a 2,38ms y puede ser realizado si el rechazo del transductor es suficiente para que la oscilación no sea detectada como señal de retorno.

En el modo de simple eco, todo lo que se debe hacer es esperar el retorno de la señal transmitida, que viaja aproximadamente a 0,9 m/s por pie ida y vuelta. La señal de retorno es amplificada y aparece como un nivel lógico alto en salida ECHO. El tiempo entre que INIT se pone en alto y ECHO lo hace también, es proporcional a la distancia al objetivo desde el transductor. Si se desea, el ciclo puede ser repetido, retornando INIT a un nivel bajo y después en alto cuando se desea la siguiente transmisión.



6500 - Ranging Module

La Serie 6500 es un módulo sonar económico que puede controlar a todos los transductores electrostáticos SensComp. Este módulo, con una interfaz sencilla, es capaz de medir las distancias de 6 pulgadas a 35 pies. La precisión típica es de +/-1% de la lectura en toda la gama.

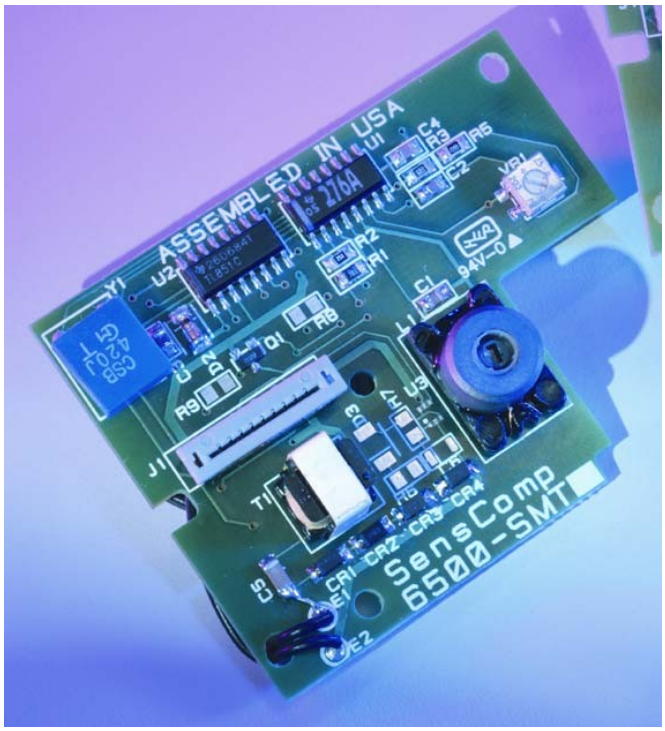
Este módulo tiene una entrada de borrado externo que permite la exclusión eco

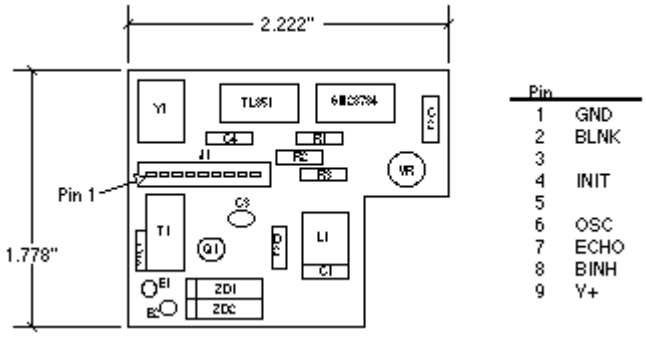
selectivo para la operación en un modo de múltiple-eco.

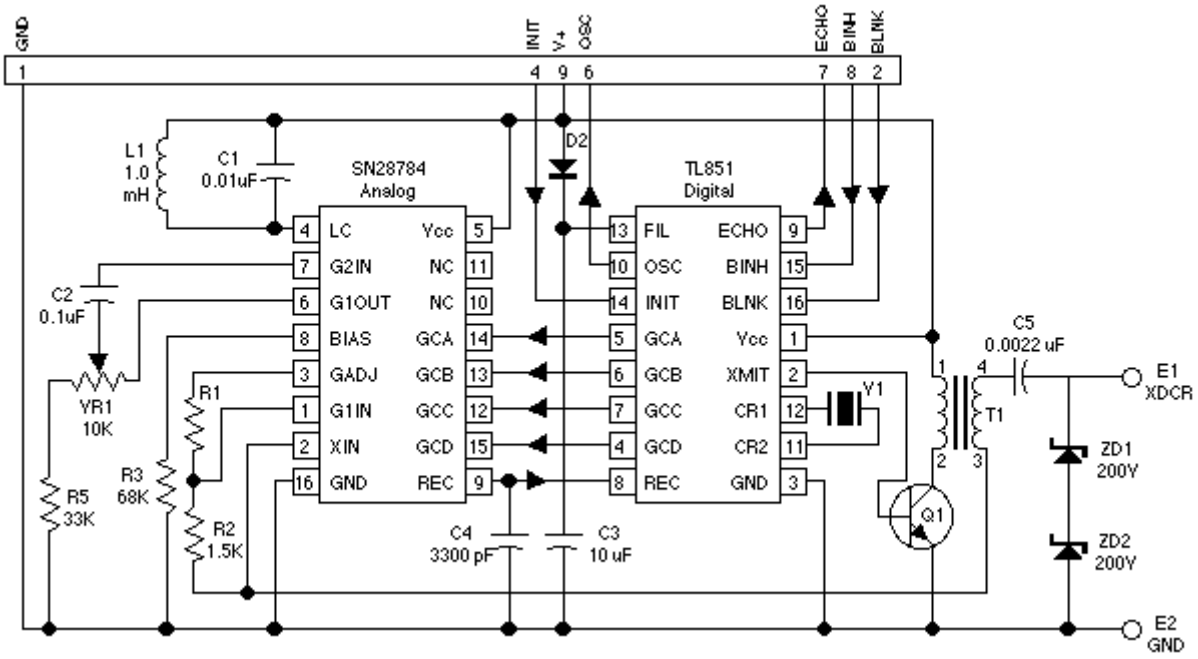
El módulo es capaz de diferenciar los ecos de los objetos que son sólo tres pulgadas de distancia. La ganancia controlada digitalmente y el amplificador de ancho de banda variable minimiza el ruido y los lóbulos laterales de detección en las aplicaciones de sonar.

El módulo tiene un preciso resonador cerámico controlado de 420kHz como generador de base de tiempo. Una salida basada en los 420kHz se proporcionan para uso externo. La salida de transmisión del sonar es de 16 ciclos a una frecuencia de 49,4kHz.

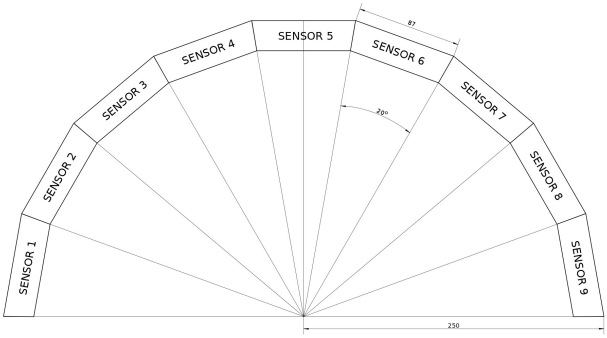
El módulo de la Serie 6500 funciona en un rango de alimentación de CC de 4,5 voltios a 6,8 voltios (5 voltios nominal) y es caracterizado para la operación de 0° C a 40°C.



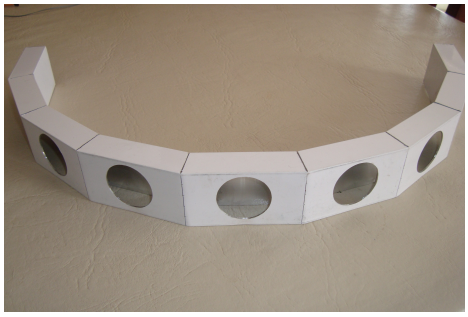


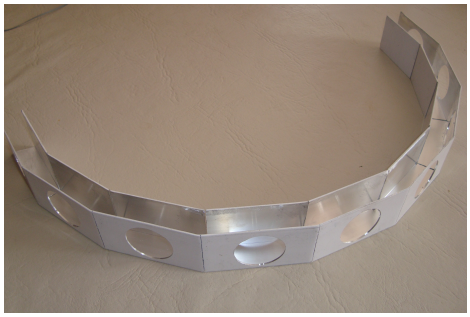


Disposición Mecánica de la estructura de soporte para sensores

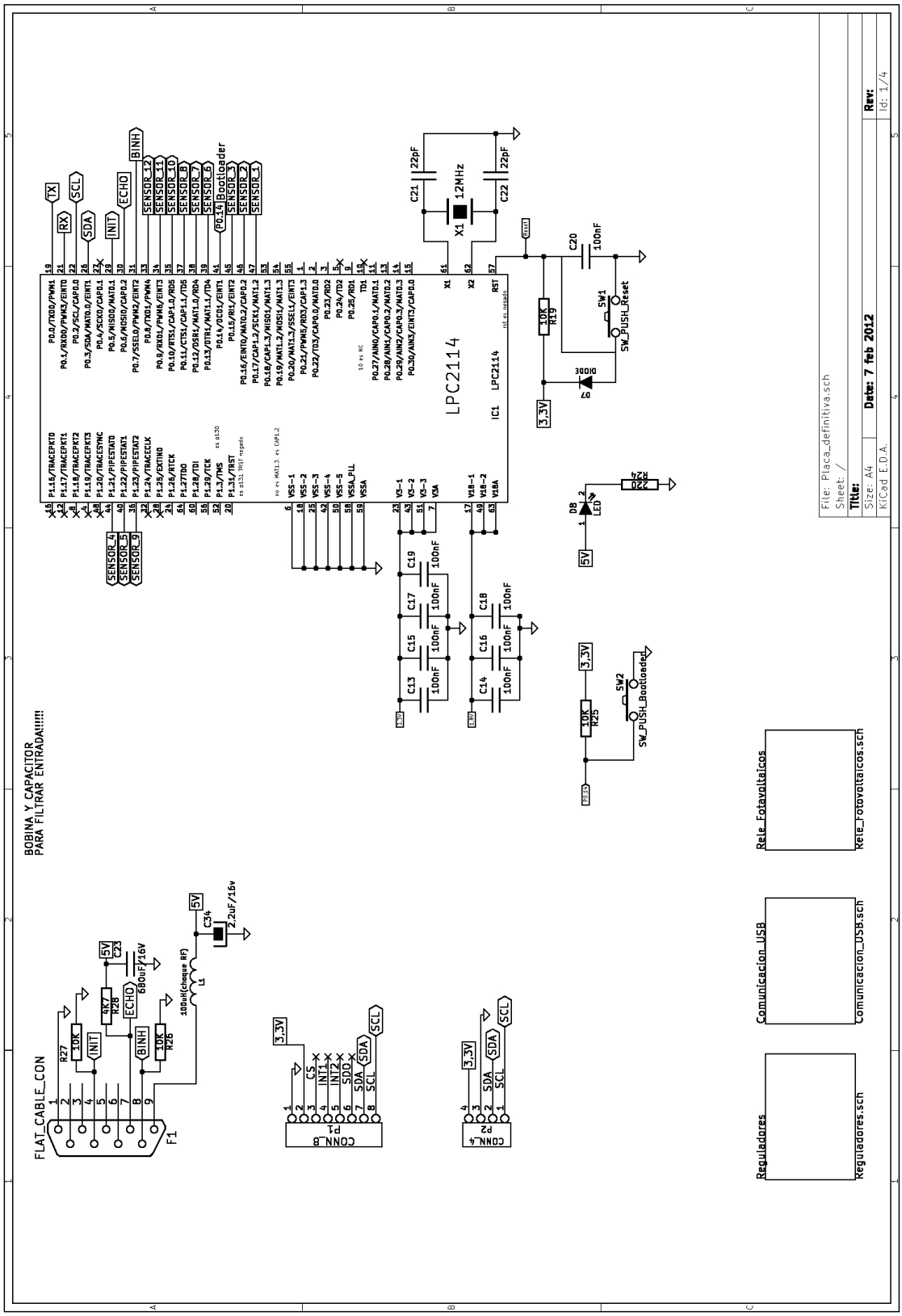


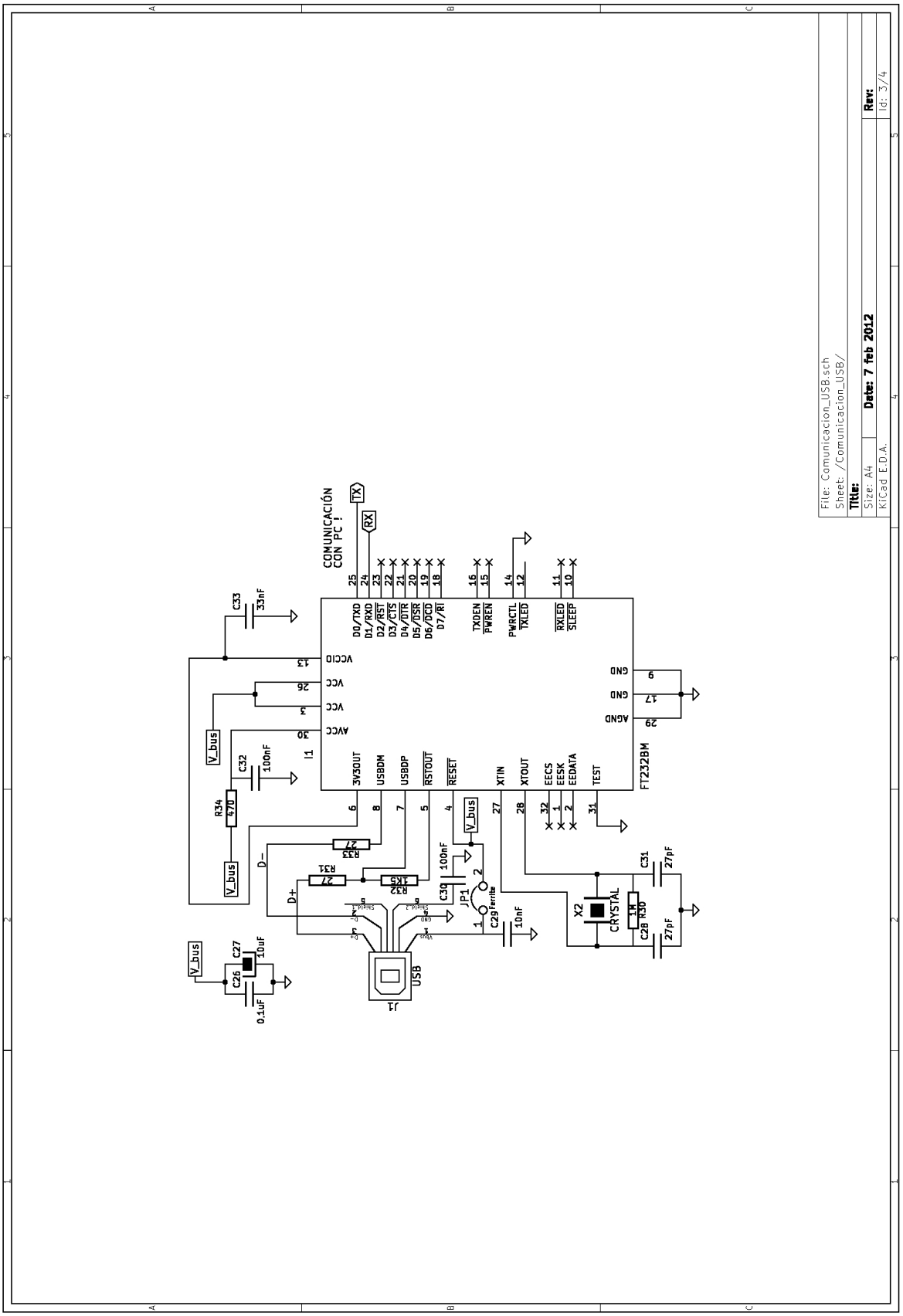
Estructura de soporte





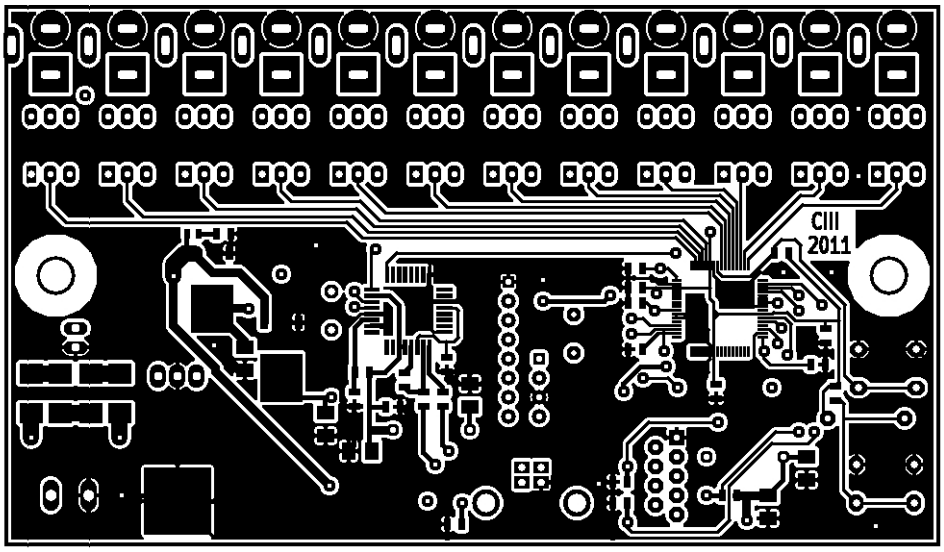
CIRCUITOS


LPC2114
IC1
X1
12MHz
C21
22pF
C22
22pF
C20
100nF
SW1
SW_PUSH_Reset
R19
10K
DIODE
D7
3.3V
D8
LED
5V
R20
470
SW2
SW_PUSH_Bootloader
R25
10K
P0.14
FLAT_CABLE_CON
F1
R27
10K
R28
4K7
C23
680uF/16V
R26
10K
L1
10Uuh(Cheque RF)
C24
2.2uF/16v
BOBINA Y CAPACITOR
PARA FILTRAR ENTRADA!!!!!
CONN_8
P1
CS
INT1
INT2
SDO
SDA
SCL
CONN_4
P2
Rele_Fotovoltaicos
Rele_Fotovoltaicos.sch
Comunicacion_USB
Comunicacion_USB.sch
Reguladores
Reguladores.sch
File: Placa_definitiva.sch
Sheet: /
Title:
Size: A4
Date: 7 feb 2012
Rev:
KiCad E.D.A.
Id: 1/4

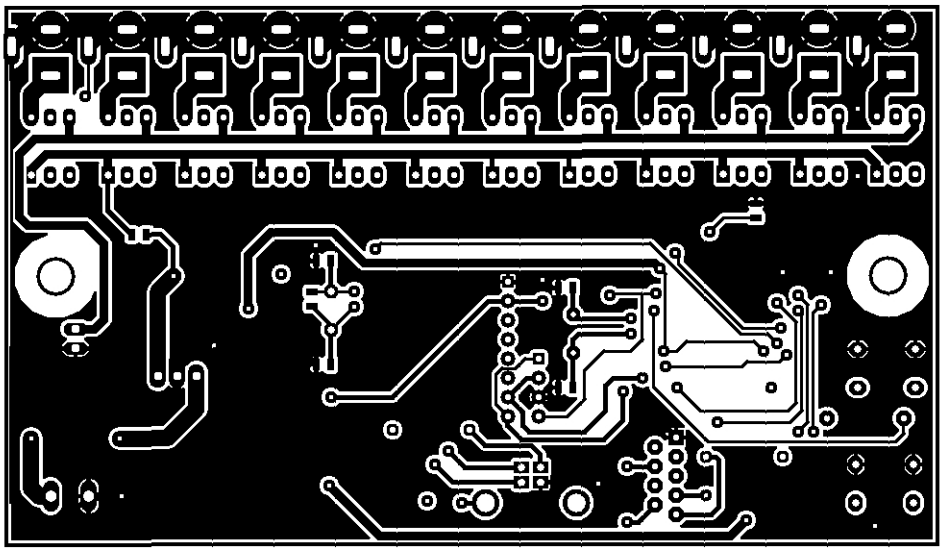

COMUNICACIÓN CON PC !
TX
RX
D0/TXD
D1/RXD
D2/RST
D3/CTS
D4/DTR
D5/DSR
D6/DCD
D7/RI
TXDEN
PWREN
PWRCTL
TXLED
RXLED
SLEEP
VCCIO
VCC
AVCC
GND
AGND
3V3OUT
USBDM
USBDP
RSTOUT
RESET
XTIN
XTOUT
EECS
EESK
EEDATA
TEST
FT232BM
I1
C33
33nF
C32
100nF
R34
470
V_bus
R33
27
R31
27
R32
1K5
C30
100nF
JP1
Ferrite
C29
10nF
X2
CRYSTAL
C31
27pF
C28
27pF
R30
1M
D+
D-
J1
USB
C26
0.1uF
C27
10uF
File: Comunicacion_USB.sch
Sheet: /Comunicacion_USB/
Title:
Size: A4
Date: 7 feb 2012
Rev:
KiCad E.D.A.
Id: 3/4

DISEÑO PCB PLACAS

CAPA SUPERIOR



CAPA INFERIOR



SOFTWARE

El software consta de 2 partes, uno es el embebido dentro del microcontrolador y otro es el que se ejecuta en la computadora para el usuario. Están escritos en 2 lenguajes de programación distintos, el software embebido esta en C mientras que el de la PC esta en Python. Se eligieron estos lenguajes respectivamente porque son los que ofrecen mayor flexibilidad para cada una de las aplicaciones, mientras C permite un control tanto a bajo como a medio nivel lo que lo hace ideal para microcontroladores; Python permite un control a alto nivel y con muchos complementos para lograr una ágil y rápida programación.

Parte del Software que se desarrollo para el microcontrolador se ve en estas lineas:

```
#define INIT_LOW    5.0
#define BINH_LOW    5.7
#define PERIODO     60.0  //PERIODO en ms, valores por defecto. (cubre 20metros)
#define INIT PIN0_5
#define BINH PIN0_7
#define ECHO CAP02_P06
#define SENSOR_1 PIN0_17
#define SENSOR_2 PIN0_16
#define SENSOR_3 PIN0_15
#define SENSOR_4 PIN1_21
#define SENSOR_5 PIN1_22
#define SENSOR_6 PIN0_13
#define SENSOR_7 PIN0_12
#define SENSOR_8 PIN0_11
#define SENSOR_9 PIN1_23
#define SENSOR_10 PIN0_10
#define SENSOR_11 PIN0_9
#define SENSOR_12 PIN0_8

unsigned int pclk_distancia=0, pclk_distancia_2, pclk_distancia_1;
int  sonar_listo, log_stop=FALSE, log_count, log_scan, num_sonar_scan=0, bandera_scan;
float PERIODO_MATCH, INIT_LOW_MATCH, BINH_LOW_MATCH, distancia, log_time_MATCH;
char comando_uart[20]={0};
float distancias[12]={0};

//--- Copiar los PARAMETROS -------------------------------
float sonar_angles[12]={ SONAR_ANGLE1,SONAR_ANGLE2,SONAR_ANGLE3,SONAR_ANGLE4,
                        SONAR_ANGLE5,SONAR_ANGLE6,SONAR_ANGLE7,SONAR_ANGLE8,SONAR_ANGLE9,
                         SONAR_ANGLE10,SONAR_ANGLE11,SONAR_RADIO12};
float sonar_radios[12]= {SONAR_RADIO1,SONAR_RADIO2,SONAR_RADIO3,SONAR_RADIO4,
                        SONAR_RADIO5,SONAR_RADIO6,SONAR_RADIO7,SONAR_RADIO8,SONAR_RADIO9,
                         SONAR_RADIO10,SONAR_RADIO11,SONAR_RADIO12};
float sonar_position[3]= {SONAR_CENTRO_X, SONAR_CENTRO_Y, SONAR_CENTRO_Z};
int sonar_count = SONAR_COUNT;
float sonar_min_range = SONAR_MIN_RANGE;
float sonar_max_range = SONAR_MAX_RANGE;
float log_time= SONAR_LOG_TIME;

//--- Interrupciones -------------------------------------
void callbacks_reset_BINH_INIT();
```

```
void callbacks_set_BINH();
void callbacks_set_INIT();
void irq_timer0_ECHO(void);
void callbacks_LOG_TIME();
//--- Funciones -------------------------------------------
void cm_get_scan(void);
void cm_get_range(char);
void cm_error();
void log_funcion(void);
void config_UART(void);
void config_TIMER(void);
void config_GPIO(void);
//------------------------------------ funcion principal -------------------------------
int main(void){
    int count_char;
    char num_sonar, XoYoZ;
    int a = 1;
    config_UART();
    config_TIMER();
    config_GPIO();
   while(TRUE)
    {
        count_char = receive_packet(comando_uart);  // Lectura del buffer de recepcion
       if(count_char > 0)
        {
               switch (comando_uart[0])
                {
                    case CMD_GET_COUNT:
                        send_packet ((char *)&sonar_count, 4);
                        break;

                    case CMD_GET_SCAN:
                        cm_get_scan();
                        break;

                    case CMD_GET_RANGE:
                        bandera_scan = FALSE;
                        num_sonar = comando_uart[1];
                        cm_get_range(num_sonar);
                        break;
```

Parte del Software que se desarrollo para la PC se ve en estas líneas:

```
#!/usr/bin/env python
import sys
from PyQt4 import QtCore, QtGui
from formulario import * #Ui_Form
from menu_comandos import *
import os
import time
#----------
import Image
from pygame.locals import *
import opencv
#this is important for capturing/displaying images
from opencv import highgui
#---------
from PyQt4.Qwt5 import QwtDial          # Colocar en formulario.py
from PyQt4.Qwt5 import QwtCompass       # Colocar en formulario.py

from PyQt4.Qwt5 import QwtCounter
```

```
from PyQt4.Qwt5 import QwtCompassMagnetNeedle
from math import sin, cos, pi
import pygame
import numpy as np


class MyForm(QtGui.QMainWindow):
    def __init__(self, parent=None):
        """ Inicializo la ventana y los Timer"""
        QtGui.QWidget.__init__(self, parent)
        #imagen de fondo
        self.widget = QtGui.QWidget(self)
        self.widget.setStyleSheet("background-image: url(fondo.png)")
        self.widget.setGeometry(QtCore.QRect(0, 0, 965, 650))  # x,y,ancho,alto ventana
        self.ui = Ui_Form()
        self.ui.setupUi(self)
        self.inicializar_pygame()
        self.Timer1= QtCore.QTimer()
        self.connect(self.Timer1,QtCore.SIGNAL("timeout()"),self.actualizar_video)
        self.Timer1.start(50) # cada estos ms actualiza el video.
        self.Timer2= QtCore.QTimer()
        self.connect(self.Timer2,QtCore.SIGNAL("timeout()"),self.actualizar_sonar)
        self.Timer2.start(200)  #cada estos ms actualiza el dibujo del radar.
        self.Timer3= QtCore.QTimer()
        #self.connect(self.Timer3,QtCore.SIGNAL("timeout()"),self.mouse)
        #self.Timer3.start(100)
        self.Timer3= QtCore.QTimer()
        self.connect(self.Timer3,QtCore.SIGNAL("timeout()"),self.comenzar)
        self.brujula()

    def inicializar_pygame(self):
        """ Inicializa el lugar donde se ve la camara y el radar"""
        self.zoom = 50
        self.ancho_pygame = 320
        self.alto_pygame = 550
        self.alto_video = 240
        self.alto_radar = 320
        self.centro = (self.ancho_pygame/2 , self.alto_video + self.alto_radar/2)
        os.environ['SDL_WINDOWID'] = str(self.ui.widget_pygame.winId()) #comprobas el ID
       Window con: xwininfo -all
        self.camera = highgui.cvCreateCameraCapture(0) # Aca va el numero de la camara
        pygame.init()
        pygame.mouse.set_visible(True)
        self.fuente = pygame.font.Font(None, 20)
        window = pygame.display.set_mode((self.ancho_pygame,self.alto_pygame)) #,
         pygame.NOFRAME )
        pygame.display.set_caption("ROMAA")
        pygame.display.flip()
        self.screen = pygame.display.get_surface()

   def actualizar_video(self):
        """ Cada un cierto tiempo alctualiza la imagen de la camara"""
        im =opencv.adaptors.Ipl2PIL(highgui.cvQueryFrame(self.camera))
        # Add the line below if you need it (Ubuntu 8.04+)
        #im = opencv.cvGetMat(im)
        #convert Ipl image to PIL image
        pg_img = pygame.image.frombuffer(im.tostring(), im.size, im.mode)
        #pg_img = pygame.transform.rotate(pg_img, 180) #para rotar la camara
        pg_img2 = pygame.transform.scale(pg_img, (self.ancho_pygame,self.alto_video))
        self.screen.blit(pg_img2, (0,0))
        self.dibujar_video()
        #pygame.display.flip()
        pygame.display.update(0,0,self.ancho_pygame, self.alto_video)
```

Una vista al trabajo terminado: