Instituto Nacional de Astrofísica, Optica y Electrónica.

REPORTE TÉCNICO

No. 627

COORDINACIÓN DE ASTROFISICA

RECEPTOR BÁSICO DE RADIOASTRONOMÍA.

Carpinteyro Ponce Ulises, Palacios Fonseca Juan,
Sánchez Urrieta Susana, Ayala Raggi Salvador Eugenio,
Mendoza Torres Eduardo.

Contenido

Introducción.

En este informe técnico se reporta la actividad que se realizó en el programa de servicio social del alumno Ulises Carpinteyro Ponce de la licenciatura en Electrónica en conjunto con los profesores investigadores Dra. Susana Sánchez Urrieta y Dr. Salvador Eugenio Ayala Raggi de la Facultad de Ciencias de la Electrónica (FCE) de la Benemérita Universidad Autónoma de Puebla, el estudiante de Doctorado Juan Salvador Palacios Fonseca y el investigador Dr. Eduardo Mendoza Torres, ambos del Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE), que consistió en la construcción de un prototipo para la recepción de señales astronómicas.

El trabajo consistió en construir un receptor básico de radioastronomía así como una interfaz gráfica para computadora con la cual se pudo registrar los datos obtenidos por el receptor y al mismo tiempo visualizar de una manera gráfica dichos datos.

El sistema de recepción está basado en tres componentes principales que son un bloque de bajo ruido (LNB) como receptor, un buscador de satélite que acondiciona la señal (amplifica y filtra) y una tarjeta de desarrollo basada en microcontrolador para convertir la señal analógica en digital y para enviar dichos datos directamente a la computadora.

Objetivo General.

Adaptar el bloque de bajo ruido de un receptor de televisión satelital para construir un receptor básico de radioastronomía.

Objetivos Particulares.

- Construir receptor básico a partir de un bloque de bajo ruido de un receptor satelital.
- Adaptar un amplificador y convertidor analógico-digital a dicho bloque para poder mandar los datos de la señal a una computadora.
- Crear una interfaz de computadora para graficar y registrar los datos recibidos del receptor.

Justificación.

Ante el alto costo que conlleva adquirir equipos de observación astronómica, es necesario que existan sistemas de bajo costo, lo que permitirá que estudiantes de nivel medio superior puedan realizar observaciones astronómicas sin tener que invertir una fuerte cantidad de dinero para ello.

Actualmente se usan sistemas comerciales de bajo costo para la recepción de televisión satelital, dichos sistemas se pueden instalar y poner en operación de manera relativamente fácil, por lo que son ampliamente usados tanto en casas habitación como en diversas instituciones y comercios. Estos sistemas pueden ser modificados y adaptados para funcionar como un receptor básico de radioastronomía.

Con la intención de proporcionar un sistema de bajo costo para la observación astronómica, se construyó un receptor básico de radioastronomía utilizando el receptor de sistemas comerciales de televisión de tal manera que el prototipo pueda ser replicado fácilmente.

Descripción del sistema.

Diagrama a bloques del sistema.

El sistema se compone de 3 principales partes que son la recepción de la señal, el acondicionamiento de la señal y el registro de la misma.

Recepción (LNB)

El bloque de bajo ruido LNB (por sus siglas en inglés Low Noise Block) es la base del receptor de TV satelital, el que es semejante a un tubo (Figura 1); en la parte final de su cavidad recibe las señales de satélite en orden de GHz (10.7 – 12.75 GHz). Dicho tubo tiene la función de guiar las ondas hasta dos alambres en los que la energía de las ondas se transforman en energía eléctrica, por tal motivo se le conoce como guía de ondas [1]. El circuito interno está formado por cuatro bloques diferenciados por la función que realizan dentro del proceso de transformación: el primer bloque consta de un resonador discriminador de polaridad con un amplificador; el segundo corresponde al limitador de ruido de entrada; en el tercero se multiplica la señal con una señal de menor frecuencia; y por último el cuarto bloque filtra y amplifica la señal.

Para esté sistema se utilizó un LNB universal de la marca Chaparral [3], el cual tiene las siguientes características:

Frecuencia de entrada:	10.7 - 12.75 GHz
Frecuencia de salida:	950 - 1750 MHz
Ganancia:	58dB (típico)

Figura 1. LNB universal de la marca Chaparral.

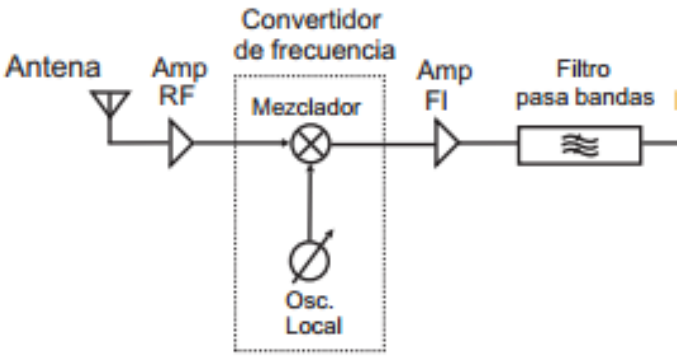


Figura 2. Diagrama a bloques de un LNB

El diagrama general del LNB lo encontramos en la Figura 2, en él podemos ver que a la entrada del sistema tenemos la guía de ondas; el siguiente elemento es un filtro pasa banda de 10.7 - 12.75 GHz que nos permite atenuar todas las señales recibidas con frecuencias fuera de ese rango [3]; a continuación existe un amplificador de bajo ruido seguido de un mezclador, este último va a multiplicar la señal recibida con una señal proveniente de un oscilador local que puede tener frecuencias de 5.15, 9.75, 10, 10.75 GHz entre otras [1]; el propósito de esta multiplicación es modular la señal recibida para mover el espectro de la señal en frecuencia; a la salida del mezclador encontramos un filtro pasa banda de 950 - 1700 MHz que atenúa las señales con una frecuencia fuera de ese rango; y el último elemento del sistema es un amplificador con lo que se contrarresta la pérdida de señal en los cables.

Acondicionamiento

La parte de acondicionamiento es una etapa intermedia que va a comunicar al receptor con la PC. Para dicha comunicación, es necesario dejar lista la señal proveniente del LNB para que la PC pueda recibirla. Es por esto que en esta se realizan tres procesos que van a permitir esa comunicación que son: filtrado de la señal, amplificación y conversión analógica-digital.

Los primeros dos procesos, que son el filtrado de la señal y amplificación, se realizan por medio de un buscador de satélites (Figura 3), el cual es un dispositivo electrónico que puede ponderar la intensidad de una señal con un galvanómetro en una escala de 0 a 10 (donde 0 es menor y 10 es mayor) y emitiendo un sonido donde a mayor intensidad el sonido es más agudo (aumenta su frecuencia). Es utilizado para orientar con precisión una antena parabólica hacia los satélites. Tiene dos conexiones, una para conectarlo hacia el LNB y otra para alimentarlo con una fuente de alimentación (generalmente de 13 a 15 volts).

El buscador de satélites que se usó es de la marca Truspect modelo TSSF-2050, el que recibe la señal proveniente del LNB y nos proporciona una señal que alimenta la bocina con voltajes entre 1.5 - 2 volts y con una frecuencia de hasta 300KHz.

Figura 3. Buscador de Satélites

El tercer paso se realiza por medio de un microcontrolador de uso general Atmega328 en una tarjeta de desarrollo Arduino UNO (Figura 4). Con convertidor analógico – digital que trae integrado el microcontrolador se convierte la señal proveniente del buscador de satélite en datos digitales los cuales serán enviados por protocolo serial RS-232 a la PC por medio de un puerto COM.

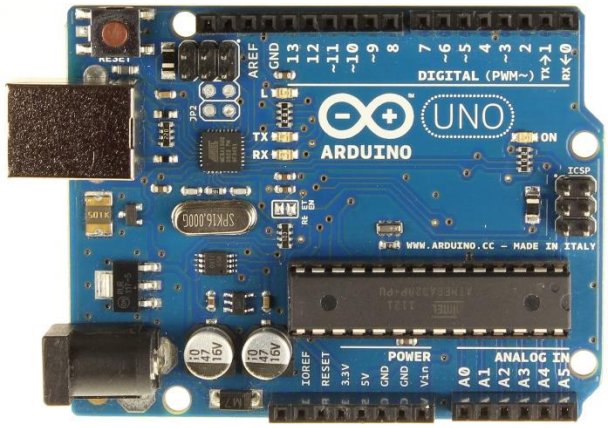


Figura 4. Tarjeta de desarrollo basada en microcontrolador Arduino UNO

Registro (PC)

En esta última parte del sistema se reciben los datos, que envía el microcontrolador a la PC, para que puedan ser graficados en tiempo real y al mismo tiempo registrados en un documento de texto. Para esto se realizó una interfaz de usuario en C# en el entorno de Visual Studio 13 y es por medio de ésta que se da acceso al puerto COM en el que está conectado el microcontrolador, se reciben los datos, se grafican y se les da el formato para poder registrarlos en un archivo de texto (.txt).

Descripción de actividades.

Recibir señal desde el bloque de bajo ruido (LNB).

En esta actividad se buscó verificar el funcionamiento del bloque de bajo ruido (LNB), así como observar el comportamiento de la señal al someter el LNB a diferentes estímulos como apuntarlo a los rayos del sol, a la mano de una persona, entre otros.

Para realizar estas pruebas se necesitó de un LNB, 2 cables coaxiales, un buscador de satélites y una fuente de voltaje de 18 volts que alimente tanto el LNB como el buscador de satélites.

Se comenzó conectando los 3 dispositivos como se muestra en la Figura 5, donde se puede observar que el buscador de satélites tiene dos terminales, una con la leyenda LNB y otra con RCVR, para el LNB y la fuente de alimentación respectivamente.

Figura 5. Conexión para verificar la respuesta del LNB ante diferentes estímulos.

Para comprobar que el ensamble está funcionando, se dirige la cavidad del LNB hacia alguna fuente de radiación, como por ejemplo el sol y verificamos que se registre un cambio en el tono y la ponderación que proporciona el buscador de satélite. Si al apuntar a diferentes fuentes y al obstruir el LNB con la mano el buscador de satélite registra un cambio, podemos decir que el LNB está captando señales.

Obtener señal útil de buscador de satélite.

Dentro de las características del LNB nos dicen que la señal de salida es de 900MHz a 1.7GHz, es por eso que necesitamos bajar la frecuencia a una que pueda ser muestreada con el microcontrolador de la tarjeta arduino, por ejemplo 300MHz; después amplificarla para así poder detectarla y registrarla.

Figura 6. Bocina que emite la alerta sonora de buscador de satélites.

Puesto que el buscador de satélite filtra y amplifica la señal proveniente del LNB, vamos a aprovecharlo para de ahí obtener la señal con frecuencia de máximo 300MHz. Debido a que éste solo tiene dos conexiones (LNB y Alimentación), es necesario a obtener la señal del voltaje de la bocina que emite la alerta sonora (Figura 6). La bocina es alimentada por medio de una señal modulada por ancho de pulso, PWM (por sus siglas en inglés Pulse Width Modulation), la cual consiste en enviar información a través de la variación del ancho de un pulso, por esto último se realizó una integración a la señal de voltaje obtenido de la bocina para obtener el valor promedio de dicho voltaje y posteriormente realizar la conversión analógico-digital que nos permitirá enviar los datos de la señal a la computadora.

Conversión analógico – digital, acondicionamiento de los datos y envío.

Antes de enviar la señal a la computadora, es necesario acondicionarla y una parte de esto es convertirla de analógica a datos digitales. Para dicho propósito aprovechamos el convertidor analógico-digital que incluye el microcontrolador Atmega328 incluido en la tarjeta de desarrollo Arduino UNO, que de acuerdo a las especificaciones del fabricante, es de 10 bits a la salida con un rango dinámico de entrada por default de 0 a 5V [2].

En cuanto al envío de los datos a la PC, se aprovecharon las librerías que proporciona arduino para envió de datos por medio de comunicación serial RS-232. Los datos se capturan a una frecuencia de 600KHz y se envían aproximadamente 40Hz. La frecuencia es aproximada ya que para enviar un paquete de datos en necesario realizar primero el promedio de ellos y las operaciones necesarias para este proceso no tienen un tiempo fijo, lo que provoca un cambio en la frecuencia con la que se envía el dato del valor promedio.

A continuación se describirá el código que se usó para programar el microcontrolador, el que se realizó en el lenguaje C propio de la tarjeta Arduino. Para tener una idea general del sistema, primero se presenta el diagrama de flujo, el que se muestra en la Figura 7 y posteriormente se presenta una descripción detallada a bloques del código utilizado. El código completo se encuentra en el **Anexo 1.**

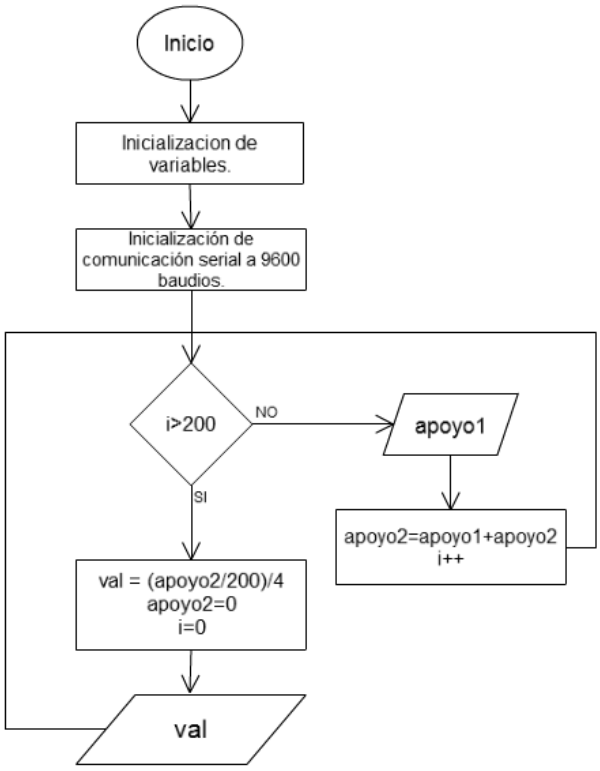


Figura 7. Diagrama de flujo del código programado en arduino.

En el Diagrama 1 podemos ver que lo primero que se hace es iniciar las variables y la comunicación serial RS-232 que en este caso se utilizó una velocidad de 9600 baudios por segundo, que es una de las permitidas por el protocolo RS-232C. Para la captura de datos se inicia un ciclo que permite sumar 200 muestras (i > 200) a través de las variables “apoyo 1” y “apoyo 2”; cuando i > 200, se realiza el promedio con ayuda de la variable “val”, se reinicializan las variables, se envía el dato promediado a la PC y se regresa al inicio del ciclo.

Con respecto al código programado en el microcontrolador, a continuación se da una explicación de una parte de él y después se presenta una figura con sus correspondientes líneas de código.

La primera parte consiste en la declaración de los puertos así como la iniciación del protocolo de comunicación serial RS-232 a 9600 baudios (Figura 8).

```
int val;
float apoyo1,apoyo2;

void setup()
{

  Serial.begin(9600);
}
```

Figura 8. Inicialización de variables y comunicación RS-232.

En la Figura 7 se puede observar que se declararon las variables “val”, “apoyo 1” y “apoyo 2”. En la primera de ellas es en donde vamos a guardar el valor arrojado por el convertidor analógico-digital, es decir, será la variable que representa a la señal obtenida del buscador de satélite. Las otras dos van a funcionar como variables de apoyo en la obtención del promedio, ya que antes de enviar la señal a la PC necesitamos hacer una integración para convertir la señal PWM de la bocina en una señal en DC. En cuanto a la línea “Serial.begin(9600)” se trata de un comando de inicialización en donde configuramos al microcontrolador para realizar una comunicación serial a 9600 baudios.

La segunda parte del programa es un ciclo que permite muestrear, promediar y enviar los datos a la PC (Figura 9).

```
void loop()
{
  for (int i=0; i <= 200; i++){
    apoyo1 = analogRead(0);
    apoyo2 = apoyo2 + (apoyo1);}
  val = (apoyo2/200)/4;
  Serial.write(val);
  apoyo2=0;
}
```

Figura 9. Muestreo, promedio y envío de datos.

En la Figura 8, lo primero que tenemos es un ciclo de 200 repeticiones, donde se realiza un muestreo de la señal y una acumulación en las variables de apoyo. Al terminar el ciclo se realiza una división entre 200 para así obtener el promedio y una división entre 4; esto se hizo porque la señal muestreada es de 10 bits (1024 valores diferentes) mientras que la comunicación serial se hace con 8 (256 valores diferentes) y al dividir entre 4 estamos escalando los valores de manera que el dato pueda ser representado con 8 bits. El resultado de estas divisiones se asigna a la variable “val” la cual definimos como la variable que representa a la señal se salida del buscador de satélite. Con el comando “serial.write()” se envía el promedio de las muestras por comunicación serial a la PC, se reinicia la variable “apoyo 2” y se vuelve a comenzar el ciclo.

Creación de la interfaz gráfica de usuario.

Para la interfaz que recibe los datos y los registra en un archivo, entre la gran diversidad de lenguajes y softwares de programación, se eligió Visual Studio 2013 debido a la facilidad de crear ambientes gráficos y se programa en C# por ser uno de los lenguajes más conocidos [4].

Para iniciar la interfaz, se creó un nuevo proyecto desde la ventana inicial del programa Visual Studio 2013 y se eligió la aplicación “Windows Form Application” del tipo C# como se puede ver en la Figura 10.

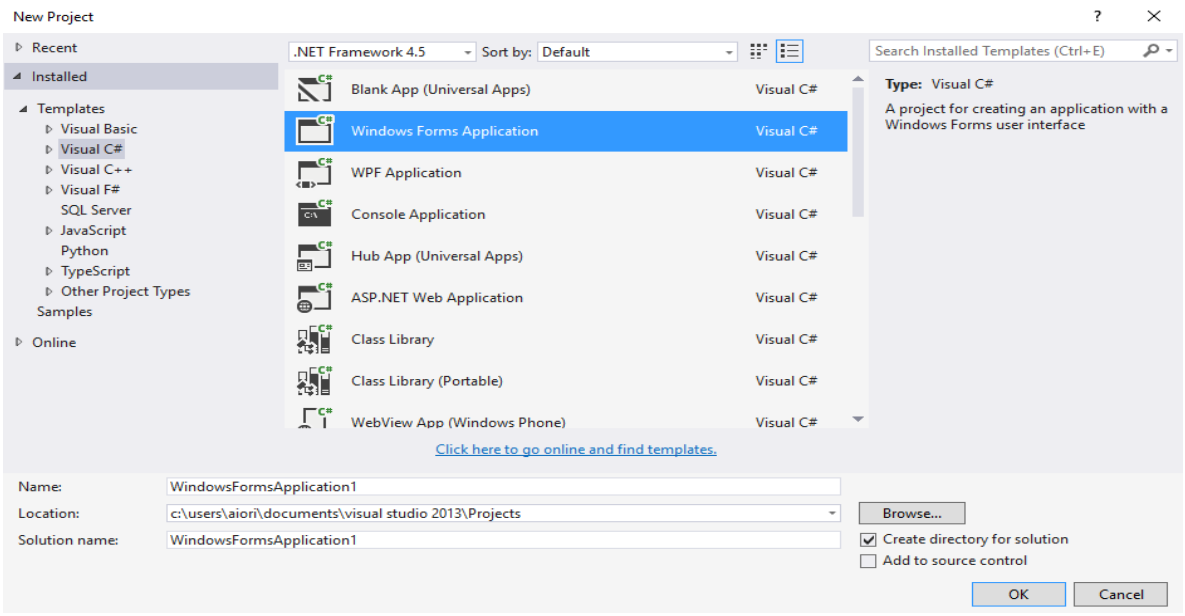


Figura 10. Creación del proyecto en Visual Studio 2013

Una vez creado el proyecto, se proporciona una ventana en blanco que es nuestro espacio de trabajo (Figura 11) y podemos modificarlo a nuestra conveniencia con los objetos que se proporcionan, los que pueden ser botones, etiquetas, cuadros de texto, imágenes, gráficos, etc.

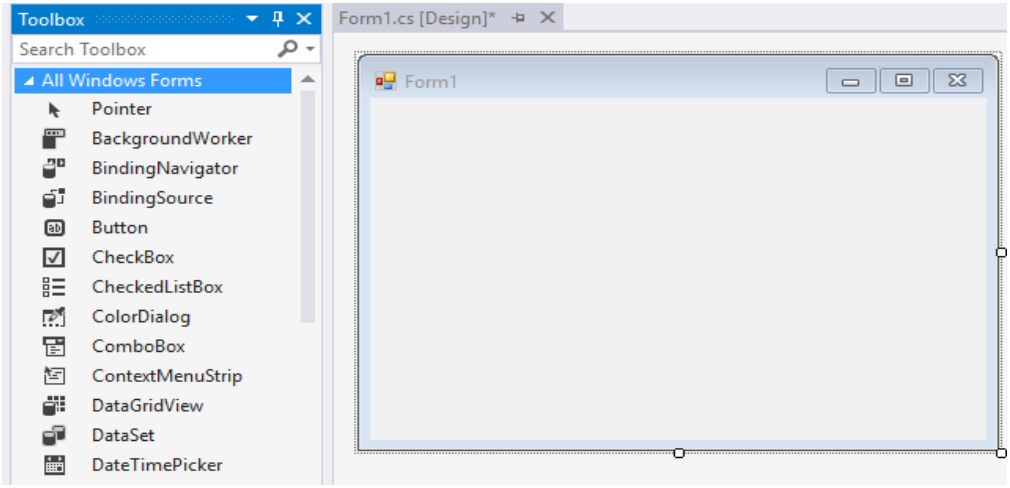


Figura 11. Espacio de trabajo

Para agregar un objeto al espacio del trabajo, primero lo seleccionamos en la barra "Toolbox" y lo arrastramos hasta el lugar donde deseamos dentro del espacio de trabajo. Por ejemplo, para agregar el botón "inicio" en la parte inferior derecha del espacio de trabajo, seleccionamos "Button" en la barra "Toolbox" y lo arrastramos hasta el espacio de trabajo (Figura 12).

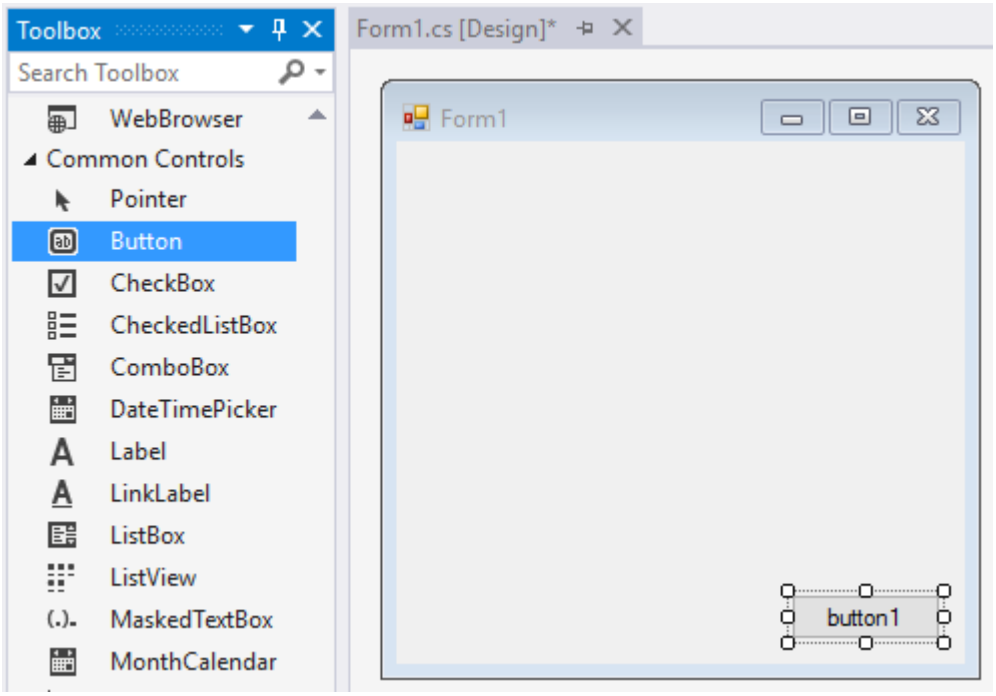


Figura 12. Creación de un botón en el espacio de trabajo.

Si queremos cambiar alguna propiedad referente al objeto, en la barra “Properties” podemos modificar características como el color, imagen de fondo, texto, fuente del texto, entre otros. En este caso vamos a modificar el nombre de botón por lo cual en el apartado “text” escribimos “Inicio” (Figura 13).

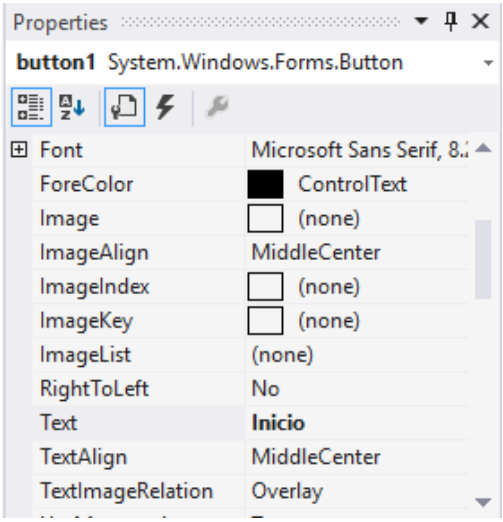


Figura 13. Propiedades de del botón.

Cada vez que se coloca un objeto en el espacio de trabajo también se agrega una función al código principal, el que podemos ver al hacer doble clic en cualquiera de los objetos agregados como se muestra en la Figura 14. Es aquí donde se pondrán las líneas de código correspondientes al objeto colocado.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Receptor_interfaz
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

Figura 14. Función del botón.

En la Figura 14 podemos observar que dentro de la función principal llamada “public partial class Form1 : Form” se encuentran las funciones “Form1()” y “button1_Click”. La primera es una función creada de manera automática cuando creamos el espacio de trabajo y es utilizada para programar sentencias que no están relacionadas con ningún objeto, por ejemplo inicialización de procesos, operaciones aritméticas, entre otros. La segunda es la función correspondiente al botón que le llamamos “inicio” y es aquí en donde posteriormente escribiremos el código que se ejecutará al dar clic en este botón.

En nuestro caso, para realizar la interfaz se utilizaron cuadros de texto, seleccionadores, imágenes que al mismo tiempo servirían como espacio para crear las gráficas de la señal, etiquetas y botones (Figura 15).

Figura 15. Interfaz creada.

Cada objeto que se agregó al espacio de trabajo creó una función en el código principal (Figura 16), que es donde se debe agregar el código correspondiente a la acción que el objeto vaya realizar.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace Receptor_interfaz
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void boton_act_Click(object sender, EventArgs e)
        {
        }
        private void boton_ini_Click(object sender, EventArgs e)
        {
        }
        private void boton_par_Click(object sender, EventArgs e)
        {
        }
        private void serial(object sender,
        {
        }
        public void dibujar(object sender, EventArgs e)
        {
        }
        private void label2_Click(object sender, EventArgs e)
        {
        }
        private void label1_Click(object sender, EventArgs e)
        {
        }
        private void label3_Click(object sender, EventArgs e)
        {
        }
        private void fontDialog1_Apply(object sender, EventArgs e)
        {
        }
        private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
        }
        private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
        }
        private void radioButton3_CheckedChanged(object sender, EventArgs e)
        {
        }
        private void radioButton4_CheckedChanged(object sender, EventArgs e)
        {
        }
        private void radioButton5_CheckedChanged(object sender, EventArgs e)
        {
        }
        private void radioButton6_CheckedChanged(object sender, EventArgs e)
        {
        }
    }
}
```

Figura 16. Funciones vacías generadas por los objetos.

Después de agregar al espacio de trabajo todos los objetos necesarios (Figura 15), debemos programar las acciones que realizarán escribiendo el código dentro de la función correspondiente, estas líneas de código se programan con la misma sintaxis de lenguaje C.

El proceso que realizará el programa se puede observar en el diagrama de flujo presentado en la Figura 17, donde se puede observar que en la primera parte se prepara para graficar, integrar y almacenar los datos recibidos a través de la iniciación de librerías, variables, comunicación serial así como el espacio donde se graficará. La segunda es un ciclo que permite graficar los datos recibidos e integrados mientras exista espacio para ello.

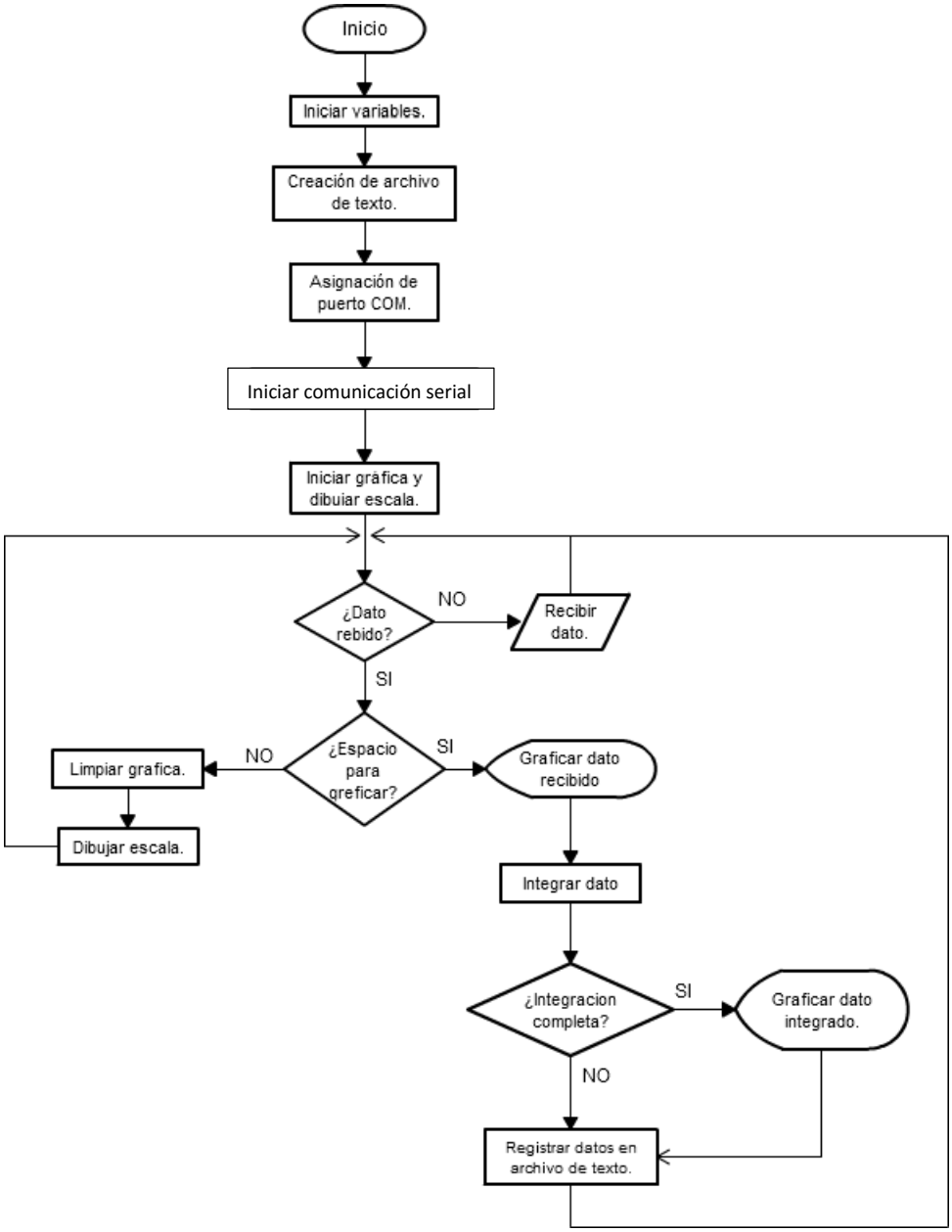


Figura 17. Diagrama de flujo del código en C#.

Con respecto al código en c#, la primera parte consiste en el llamado a las librerías, las que vienen incluidas de manera predeterminada cuando nosotros elegimos el proyecto de Windows Forms del tipo C# (Figura 18).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
```

Figura 18. Iniciación de librerías.

El siguiente paso es declarar e inicializar las variables las que debido a que se usarán en distintas funciones, serán variables globales. Estas se declaran dentro de la función principal como se puede ver en la Figura 19.

```
int valor = 0;
int conta_integrador= 0, valor_i=0;
int cte = 1;
int conta = 0, x0, y0, x1, y1, var = 0, vari = 0, x0i, y0i, x1i, y1i;
string nombre;
double dato;
string hora;
DateTime fecha;
string dato_normalizado = "Datos normalizados";
string dato_integrado;
```

Figura 19. Declarado e inicialización de variables globales.

Las variables donde se irá guardando la señal serán “valor”, ”valor_i”, ”dato_normalizado” y “dato_integrado”. En “valor” y “valor_i” se guarda la señal en formato de 8 bits; mientras que en “dato_normalizado” y “dato_integrado” en formato de cadena de caracteres para guardarlo en un archivo de texto.

Para crear el espacio de trabajo, el archivo de texto y la hora de inicio, en la función “Form1()” se agregan las líneas mostradas en la Figura 20. Donde se puede observar que el archivo se guardara en la ruta c:\\Datos.txt.

```
public Form1()
{
    InitializeComponent();
    StreamWriter escrito = File.CreateText("c:\\Datos.txt");
    fecha = DateTime.Now;
    escrito.WriteLine("Inicio a las " + fecha.ToShortTimeString());
    escrito.WriteLine("");
    escrito.Flush();
    escrito.Close();

}
```

Figura 20. Inicio de espacio de trabajo y creación de archivo de texto.

El archivo de texto se crea con el comando “streamwriter” y las siguientes instrucciones son para colocar la hora de inicio al archivo.

Para las acciones que se van a ejecutar cuando se active cada objeto, comenzaremos con el botón “actualizar” (Figura 21).

```
private void boton_act_Click(object sender, EventArgs e)
{
    label2.Text = tex_nue.Text;
    nombre = label2.Text;
}
```

Figura 21. Botón “actualizar”

Antes de presionar el botón “Actualizar” se debe considerar las instrucciones mostradas en la interfaz respecto a la captura del puerto donde está conectado el arduino, de tal manera que al activarlo, lo que hay en el cuadro de texto aparecerá en la parte superior del botón “Actualizar”, esto se realiza con las líneas mostradas en la Figura 21.

La siguiente función corresponde al botón “inicio” que empieza con el proceso de recepción, graficado y registro de los datos (Figura 22).

```
private void boton_ini_Click(object sender, EventArgs e)
{

    if (!serialPort1.IsOpen)
    {
        serialPort1.PortName = nombre;
        serialPort1.Open();
        serialPort1.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(serial);
    }
}
```

Figura 22. Botón “iniciar”

Al presionar el botón inicio lo primero que hace es verificar que la comunicación serial no esté ocupada (!serialPort1.IsOpen) y en tal caso se asigna y habilita el puerto COM que vamos a usar (serialPort1.PortName, serialPort1.Open). Una vez hecho esto se inicia con la recepción de los datos (seriaPort1.DataReceived), mandando a llamar a la función “serial”, la que llama de manera cíclica a la función “dibujar” como se muestra en la Figura 23.

```
private void serial (object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    try
    {
        this.Invoke(new EventHandler(dibujar));
    }
    catch
    {
    }
}
```

Figura 23. Función “serial”.

Aunque la función “serial” tiene como única tarea llamar a otra función, es importante no omitirla ya que al recibir los datos el programa buscará esta función y si no se encuentra se generará un error de sintaxis.

La función “dibujar” es larga, por lo cual se explicará por partes; en la primera se genera el espacio donde se graficará (Graphics graf), se crea un buffer donde se guardará el dato recibido el cual viene en 8 bits y se convierte a 32 como lo muestra la Figura 24.

```
public void dibujar(object sender, EventArgs e)
{
    try
    {
        Graphics graf = pictureBox1.CreateGraphics();
        int x = 0;
        byte[] buffer = new byte[2];
        serialPort1.Read(buffer, 0, 1);
        valor = (8*Convert.ToInt32(buffer[0]))-500;
```

Figura 24. Inicio de la gráfica.

La siguiente parte del código es iniciar la gráfica. En este caso se definió un color blanco como fondo y se dibujaron líneas horizontales que sirven como escala y con esto el programa está listo para graficar. Las líneas mostradas en la Figura 25 realizan esta función.

```
if (conta == 0)
{
    graf.Clear(Color.White);

    graf.DrawLine(new Pen(Brushes.Green, 1), new PointF(0, 20), new PointF(1200, 20));
    for (int j = 1; j <= 5; j++)
    {
        x = (256 / 5) * j + 20;
        graf.DrawLine(new Pen(Brushes.Green, 1), new PointF(0, x), new PointF(1200, x));
        graf.DrawLine(new Pen(Brushes.Green, 1), new PointF(0, x), new PointF(1200, x));
    }
    var = 275;
    vari = 275;
    valor_i = 0;
    conta_integrador = 0;

}
```

Figura 25. Inicio de la gráfica y escala.

Para graficar los datos que vamos recibiendo del microcontrolador necesitamos dos lecturas que se representarán como puntos en el plano cartesiano unidos por una línea. Cada punto tendrá como coordenada Y el valor que se reciba del microcontrolador y la coordenada X será un número progresivo entre 0 y 1200 que es el máximo de puntos que se pueden graficar. Cuando se inicia la gráfica el primer punto es (0,0) y el segundo es la primera lectura; en adelante el último dato graficado será el primero y la lectura actual será el segundo. Esto ser realiza con las líneas mostradas en la Figura 26.

```
if (conta < 1200)
{
    x0 = conta;
    y0 = var;
    x1 = conta + 1;
    y1 = 275 - Convert.ToInt32(valor);
    graf.DrawLine(new Pen(Brushes.Blue , 1), new PointF(x0, y0), new PointF(x1, y1));
    conta++;
    var = y1;
```

Figura 26. Graficado de los datos recibidos..

Además de graficar los datos recibidos también se grafican al integrarlos, para lo que al recibir la muestra, ésta se suma con la anterior hasta cumplir el tiempo establecido en la interfaz de entrada, el cual puede ser entre 0.1 y 10 segundos (Figura 15); después se divide la suma entre el número de muestras y este es el dato que se grafica aplicando el mismo procedimiento que para los datos sin integrar. El código correspondiente se presenta en la Figura 27.

```
if (conta_integrador < cte)
{
    valor_i = valor_i + valor;
    conta_integrador++;
}
else
{
    conta_integrador = 0;
    x0i = x0 - cte;
    y0i = vari;
    x1i = conta;
    y1i = 275 - (valor_i / cte);
    vari = y1i;
    graf.DrawLine(new Pen(Brushes.Red, 1), new PointF(x0i, y0i), new PointF(x1i, y1i));
    dato_integrado = Convert.ToString(valor_i);
    valor_i = 0;
}
```

Figura 27 Integración y gráfica de datos.

La parte final de la función “dibujar” consiste en registrar los datos, tanto los que se obtienen directamente del microcontrolador como los que se van integrando en un archivo .txt . El código utilizado se muestra en la Figura 28.

```
            dato = Convert.ToDouble(valor);
            dato_normalizado = Convert.ToString(dato);
            StreamWriter escrito = File.AppendText("c:\\Datos.txt");
            fecha = DateTime.Now;
            escrito.WriteLine(fecha.ToLongTimeString() + " \t " + dato_normalizado + " \t " + dato_integrado);
            escrito.Flush();
            escrito.Close();

        }
        else
        {
            conta = 0;
        }
```

Figura 28 Registro de datos en archivo .txt

Debido a que el archivo .txt guarda valores con formato string y los datos a guardan tiene formato entero de 32 bits, fue necesario hacer este cambio usando la instrucción “Convert.toString”.

Para escribir en el archivo de texto primero lo abrimos con el comando “StreamWriter” indicándole la ruta y nombre; posteriormente escribimos una nueva línea que incluye la hora, el dato recibido y el integrado para lo cual usamos el comando “escrito.WriteLine()”; para finalizar se limpia el buffer con el comando “escrito.Flush()” y se cierra el archivo con “escrito.Close()”. Es importante mencionar que este proceso es infinito ya que aunque las pantallas tienen un número finito de datos para graficar, el archivo sigue registrándolos.

Con respecto a la función “botón_par_Click”, ésta se encarga de cerrar el puerto serial. Al dar clic en el botón “parar” primero se verifica que el puerto está abierto, si es así el éste se cierra y se reinicia la posición en la que se va a graficar; en caso contrario no se hace ninguna acción. Las instrucciones para esto se muestran en la Figura 29.

```
private void boton_par_Click(object sender, EventArgs e)
{
    if (serialPort1.IsOpen)
    {
        serialPort1.Close();
        conta = 0;
    }
}
```

Figura 29. Botón “parar”

Las últimas líneas de código pertenecen a las funciones relacionadas con objetos en la interfaz gráfica y todas se encuentran vacías, ya que no esperamos respuesta de ninguna de ellas debido a que su objetivo es hacer que la interfaz sea agradable a la vista, con excepción del selector de constante de tiempo. Con estos botones circulares vamos a definir cuanto tiempo vamos a integrar los datos y la parte del código que hace esto se presenta en la Figura 30.

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
    cte = 4;
}
private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
    cte = (4 * 2);
}
private void radioButton3_CheckedChanged(object sender, EventArgs e)
{
    cte = (4*10);
}
private void radioButton4_CheckedChanged(object sender, EventArgs e)
{
    cte = (4*20);
}
private void radioButton5_CheckedChanged(object sender, EventArgs e)
{
    cte = (4*40);
}
private void radioButton6_CheckedChanged(object sender, EventArgs e)
{
    cte = (4*100);
}
```

Figura 30. Botones para seleccionar las constante de tiempo de integración.

Debido a que cada 4 muestras equivalen a 0.1s en la Figura 30 podemos observar que la variable “cte” está multiplicada por 4, lo que nos permite seleccionar el tiempo de integración de 0.1, 0.2, 1, 2, 4 y 10 segundos.

El código completo se encontra en el **ANEXO 2.**

Resultados.

Una vez terminado el sistema, se realizaron pruebas para verificar el funcionamiento del mismo, es decir, se comprobó que este es capaz de graficar y registrar señales que está captando en ese momento y dependiendo de la constante de integración seleccionada integrarlos, graficarlos y registrarlos.

La primera prueba consistió capturar señales provenientes del Sol, para lo cual se hizo un recorrido del LNB apuntándolo en un arco de aproximadamente 60° durante un intervalo de tiempo de 180 segundos. La valores capturados (grafica azul) e integrados a un segundo (grafica naranja) se muestran en la Figura 31, donde se puede observar que la señal capturada va aumentando su voltaje conforme el LNB apuntaba hacia el sol directamente y va disminuyendo conforme deja de apuntarlo.

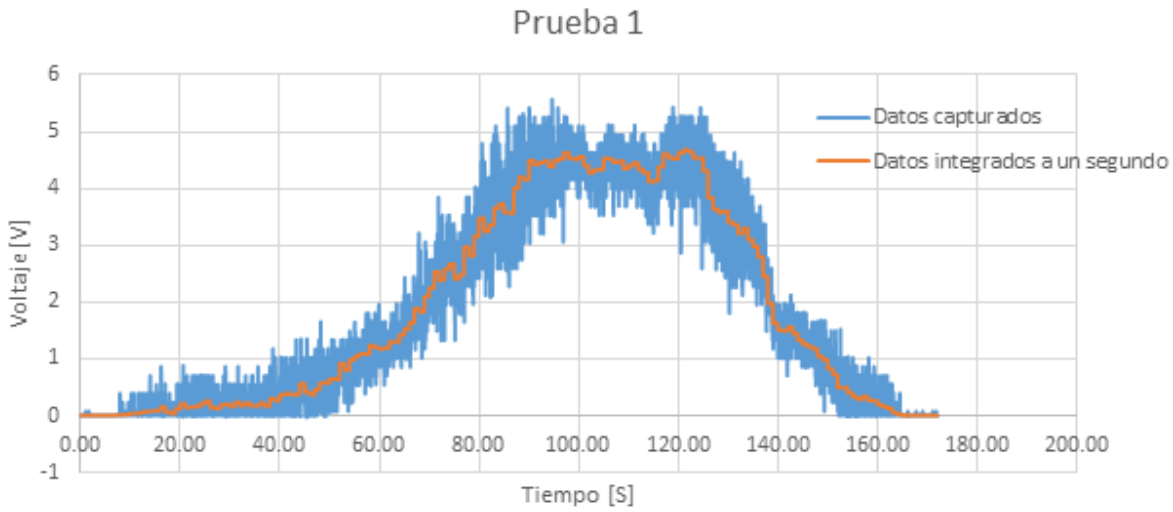


Figura 31. Respuesta del sistema a los Datos capturados e integrados.

En la segunda prueba se realizó el mismo experimento que en la primera, con la diferencia de que después de terminar la trayectoria de los 60° se tapó la cavidad del LNB con la mano 3 veces. La Figura 32 muestra la gráfica en azul para los datos capturados y en naranja para los datos integrados, donde se puede observar que hasta el segundo 175 ambas graficas (Figura 31 y 32) tienen el mismo comportamiento; a partir de este momento éste cambia. Al tapar la cavidad del LNB con la mano, el voltaje capturado es mayor (2.5 volts) que cuando se destapa (0.3 volts), teniendo un comportamiento de pulsos como se observa en la Figura 32.

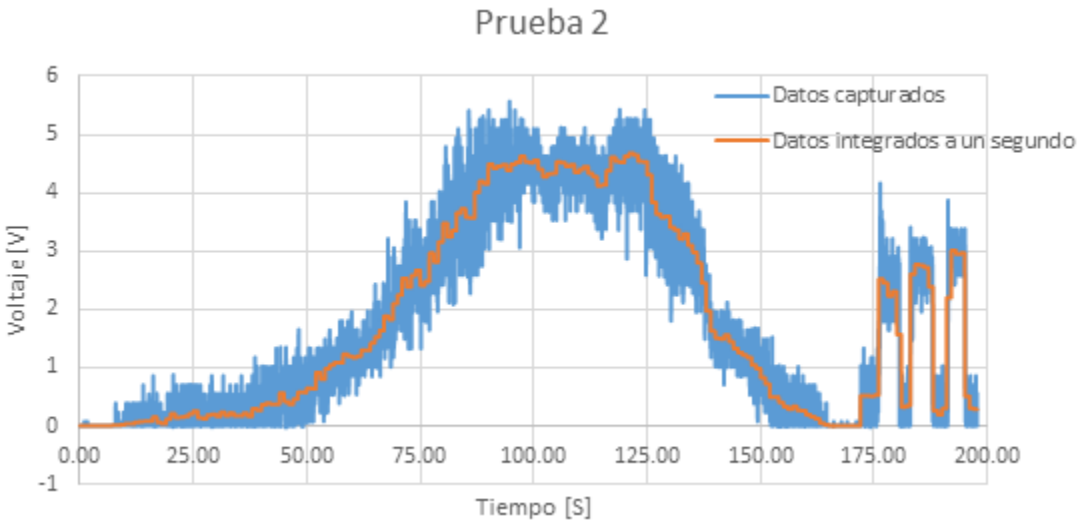


Figura 32. Respuesta del sistema al paso del sol y a la presencia de una mano en la punta del LNB

Conclusiones.

Al finalizar este trabajo de servicio social se logró:

- Adaptar el bloque de bajo ruido de un receptor de televisión satelital para construir un receptor básico de radioastronomía.
- Integrar un bloque que permite adaptar la señal proveniente del LNB para que estas puedan ser recibidas por la computadora.
- Crear una interfaz que permite graficar y registrar los datos recibidos e integrados.
- Hacer dos experimentos para probar el funcionamiento del mismo.

Con lo que se cubre el total de los objetivos planteados para este trabajo.

El prototipo final resulta ser viable como una primera aproximación de un receptor básico de radioastronomía que estudiantes del nivel medio superior puedan usar. Además, consideramos que éste informe técnico es suficientemente detallado para la reproducción del prototipo.

Al prototipo final le llamamos primera aproximación ya que actualmente éste tiene un buscador de satélites y una tarjeta Arduino UNO como parte del bloque de acondicionamiento de la señal, por lo que un posible trabajo a futuro seria poder cambiar estos elementos por un desarrollo propio que podría ser futuro programa de servicio social.

Referencias.

[1] *Satellite Signals* [online]. 2015 Disponible en: http://www.satsig.net/lnb/explanation-description-lnb.htm

[2] A. Brito, *Voltímetro con Arduino y C#* [online]. 2013 Disponible en: http://www.tallerdecontrol.com/index.php/voltimetro-con-arduino-y-c/

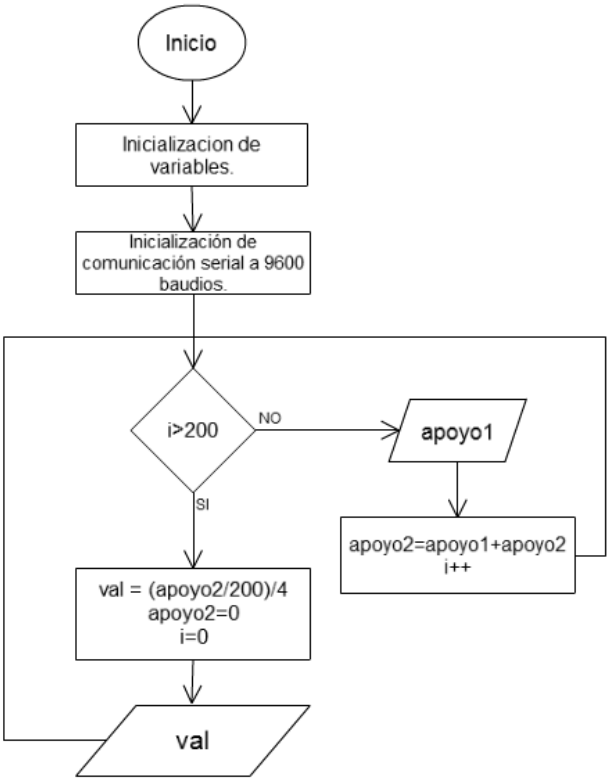
[3] *Universal LNB* [online]. 2012 Disponible en: http://www.chaparral.net/universal-lnb/

[4] J. Melgoza, *C# y Arduino* [online]. 2013 Disponible en: http://jonathanmelgoza.com/blog/c-sharp-arduino-aplicacion-led-rgb/

Anexos.

Anexo1

Diagrama de flujo del código programado en el microcontrolador.



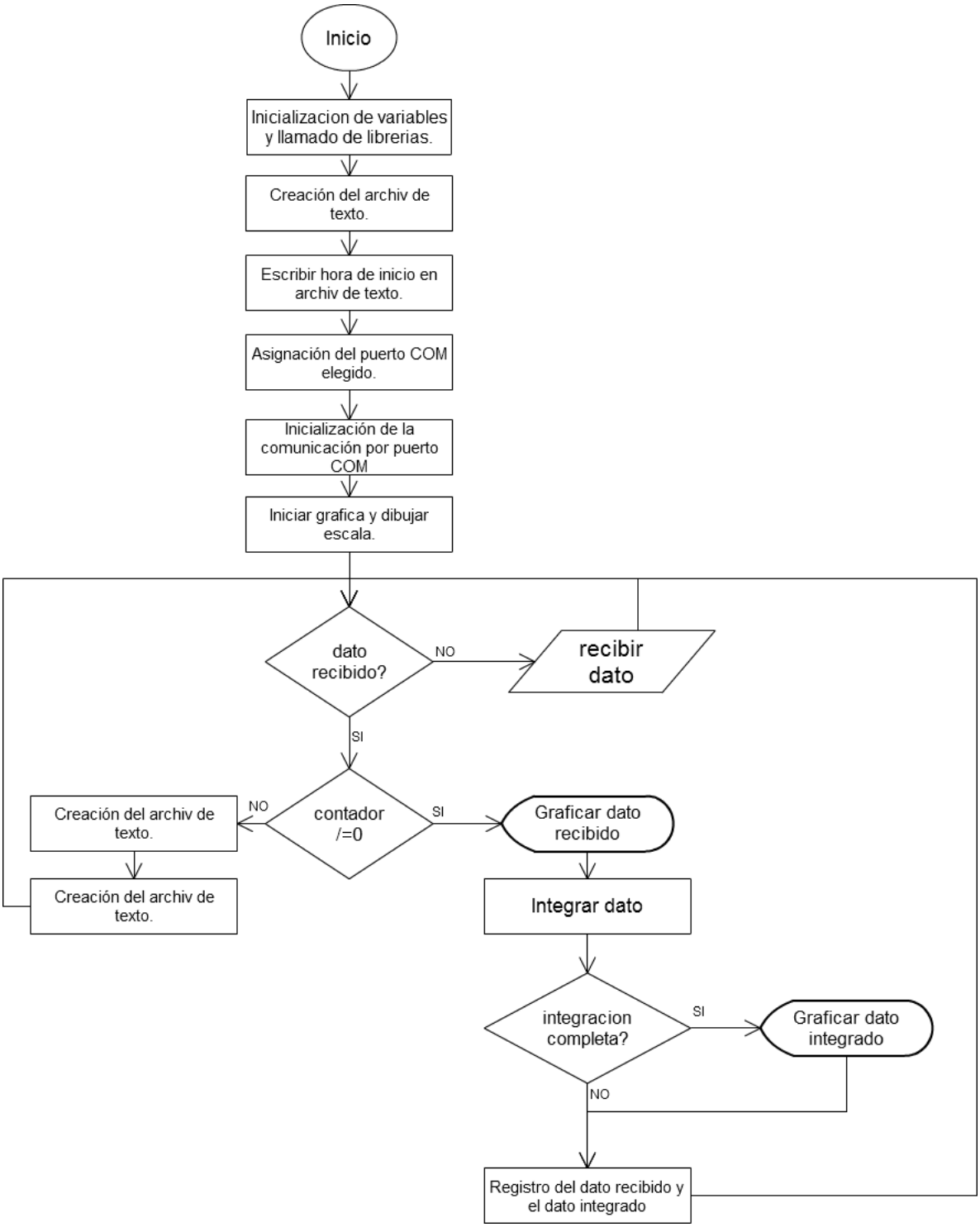
Código programado en el microcontrolador.

```
int val;
float apoyo1,apoyo2;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  for (int i=0; i <= 200; i++){
    apoyo1 = analogRead(0);
    apoyo2 = apoyo2 + (apoyo1);}
  val = (apoyo2/200)/4;
  Serial.write(val);
  apoyo2=0;
}
```

ANEXO 2

Diagrama de flujo del código implementado en Visual Studio 2013.

Este diagrama de flujo corresponde a la función principal de código programado. Las funciones que no se encuentren en este diagrama de flujo son funciones vacías o de una sola instrucción.



Código implementado en Visual Studio 2013.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;



namespace Receptor_interfaz
{
    public partial class Form1 : Form
    {
        int valor = 0;
        int conta_integrador= 0, valor_i=0;
        int cte = 1;
        int conta = 0, x0, y0, x1, y1, var = 0, vari = 0, x0i, y0i, x1i, y1i;
        string nombre;
        double dato;
        string hora;
        DateTime fecha;
        string dato_normalizado = "Datos normalizados";
        string dato_integrado;

        public Form1()
        {
            InitializeComponent();
            StreamWriter escrito = File.CreateText("c:\\Datos.txt");
            fecha = DateTime.Now;
            escrito.WriteLine("Inicio a las " + fecha.ToShortTimeString());
            escrito.WriteLine("");
            escrito.Flush();
            escrito.Close();
        }

        private void boton_act_Click(object sender, EventArgs e)
        {
            label2.Text = tex_nue.Text;
            nombre = label2.Text;
        }

        private void boton_ini_Click(object sender, EventArgs e)
        {
            if (!serialPort1.IsOpen)
            {
                serialPort1.PortName = nombre;
                serialPort1.Open();
                serialPort1.DataReceived += new …
…System.IO.Ports.SerialDataReceivedEventHandler(serial);
            }
        }

        private void boton_par_Click(object sender, EventArgs e)
```

```
 {
     if (serialPort1.IsOpen)
     {
         serialPort1.Close();
         conta = 0;

     }
 }

private void serial (object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
 {
     try
     {
         this.Invoke(new EventHandler(dibujar));
     }
     catch
     {
     }
 }

 public void dibujar(object sender, EventArgs e)
 {
     try
     {
         Graphics graf = pictureBox1.CreateGraphics();
         int x = 0;
         byte[] buffer = new byte[2];
         serialPort1.Read(buffer, 0, 1);
         valor = (8*Convert.ToInt32(buffer[0]))-500;

         if (conta == 0)
         {
graf.Clear(Color.White);
graf.DrawLine(new Pen(Brushes.Green, 1), new PointF(0, 20), new PointF(1200, 20));
for (int j = 1; j <= 5; j++)
{
x = (256 / 5) * j + 20;
graf.DrawLine(new Pen(Brushes.Green, 1), new PointF(0, x), new PointF(1200, x));
graf.DrawLine(new Pen(Brushes.Green, 1), new PointF(0, x), new PointF(1200, x));
}
             var = 275;
             vari = 275;
             valor_i = 0;
             conta_integrador = 0;

         }
         if (conta < 1200)
         {
             x0 = conta;
             y0 = var;
             x1 = conta + 1;
             y1 = 275 - Convert.ToInt32(valor);
graf.DrawLine(new Pen(Brushes.Blue , 1), new PointF(x0, y0), new PointF(x1, y1));
             conta++;
             var = y1;
             if (conta_integrador < cte)
             {
                 valor_i = valor_i + valor;
                 conta_integrador++;
             }
             else
```

```
                    {
                        conta_integrador = 0;
                        x0i = x0 - cte;
                        y0i = vari;
                        x1i = conta;
                        y1i = 275 - (valor_i / cte);
                        vari = y1i;
      graf.DrawLine(new Pen(Brushes.Red, 1), new PointF(x0i, y0i), new PointF(x1i, y1i));
                        dato_integrado = Convert.ToString(valor_i);
                        valor_i = 0;
                    }
                    dato = Convert.ToDouble(valor);
                    dato_normalizado = Convert.ToString(dato);
                    StreamWriter escrito = File.AppendText("c:\\Datos.txt");
                    fecha = DateTime.Now;
 escrito.WriteLine(fecha.ToLongTimeString() + " \t " + dato_normalizado + " \t " +
dato_integrado);
                    escrito.Flush();
                    escrito.Close();

                }
                else
                {
                    conta = 0;
                }
            }
            catch
            {

            }
        }

        private void label2_Click(object sender, EventArgs e)
        {

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void fontDialog1_Apply(object sender, EventArgs e)
        {

        }

        private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            cte = 4;
        }

        private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
            cte = (4 * 2);
        }
```

```
        private void radioButton3_CheckedChanged(object sender, EventArgs e)
        {
            cte = (4*10);
        }

        private void radioButton4_CheckedChanged(object sender, EventArgs e)
        {
            cte = (4*20);
        }

        private void radioButton5_CheckedChanged(object sender, EventArgs e)
        {
            cte = (4*40);
        }

        private void radioButton6_CheckedChanged(object sender, EventArgs e)
        {
            cte = (4*100);
        }
    }
}
```