Aplicación de Técnicas de Robótica e Inteligencia Artificial sobre un robot móvil utilizando el entorno de programaciónVisual Studio.Net

Directores: **Ing. Claudio Verrastro Ing. Roberto Barneda**
Revisión: **Ing. Juan Carlos Gómez**
www.secyt.frba.utn.edu.ar/gia/

MÓDULO 3: "Desarrollo de algoritmo de detección de líneas mediante visión artificial y control de trayectoria"

Autores: **Leandro M. Di Matteo, Andrea C. Mangone**

Índice general

Figuras

1 Introducción

Se presenta un algoritmo de detección de líneas mediante técnicas de **Visión Artificial** y un algoritmo que permite controlar el desplazamiento del robot a lo largo de la línea sobre el piso.

VISION ARTIFICIAL: La visión artificial es una técnica basada en la adquisición de imágenes, para luego procesarlas digitalmente mediante algún tipo de CPU (computadora, microcontrolador, DSP, etc.), con el fin de extraer y medir determinadas propiedades de las imágenes adquiridas

Para el desarrollo del algoritmo de detección de líneas basado en visión artificial se utilizó el lenguaje de programación C y las librerías OpenCV de Intel (Intel® Open Source Computer Vision Library). Para más información sobre las librerías de Intel:
http://opencvlibrary.sourceforge.net/.
El algoritmo de control de trayectoria para el seguimiento de línea se desarrolló bajo la plataforma .Net, utilizando el entorno de programación de C#.

Figura 1 Armado del robot ER1. Vista frontal

2 Objetivos

El objetivo de este módulo es aplicar un **algoritmo** que permita que el robot se desplace siguiendo una línea delimitada en el piso, empleando técnicas de visión artificial para su reconocimiento y realizar su seguimiento mediante técnicas de control de trayectoria y cálculos de aproximación.

ALGORITMO:
Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

Figura 2 Disposición del campo de acción

3 Requisitos fundamentales para la realización de las prácticas

A continuación se enumeran los requisitos necesarios para realizar las prácticas:

- Un robot ER1 (armado siguiendo las instrucciones del manual que viene con el equipo) con su correspondiente batería cargada.
- Una PC portátil, montada sobre el robot, con 2 puertos USB.
- OpenCV instalado en la PC portátil.
- Librerías de control de motores del ER1 instaladas en la PC portátil.
- Ejecutables de los algoritmos que se detallarán en la práctica.
- Una cinta negra de más de 3 cm de ancho de largo suficiente para señalar el recorrido deseado del robot ER1.

Para el desarrollo de los algoritmos, se necesita contar con los siguientes entornos de programación:

- Visual C++ y C# del Visual Studio .NET 2005

Figura 3 Armado del robot ER1. Vista lateral.

4 Práctica N°1: Desarrollo de un algoritmo para la detección de líneas mediante técnicas de visión artificial

Autores: **Leandro M. Di Matteo, Andrea C. Mangone**

4.1 Introducción Teórica

La **visión artificial** es una técnica basada en la adquisición de **imágenes**, para luego procesarlas digitalmente mediante algún tipo de CPU (computadora, microcontrolador, DSP, etc), con el fin de extraer y medir determinadas propiedades de las imágenes adquiridas. Se trata, por tanto, de una tecnología que combina las computadoras con las cámaras de video para adquirir, analizar e interpretar imágenes de una forma equivalente a la inspección visual humana.

IMAGEN:
Desde un punto de vista amplio es una representación en dos dimensiones de alguna característica del mundo real *f(x,y)*. Por ejemplo: Una imagen pictórica es una representación de la intensidad de luz (y sus colores) recibida sobre un plano a traves de una lente.
En una imagen radiográfica se representa el grado de transparencia de un objeto a los rayos X.

Un sistema de visión artificial se compone básicamente de los siguientes elementos:

• Fuente de luz: es un aspecto de vital importancia ya que se deben proporcionar unas condiciones de iluminación uniformes e independientes del entorno, facilitando además, si es posible, la extracción de los rasgos de interés para una determinada aplicación.

• Sensor de imagen: es el encargado de recoger las características del objeto bajo estudio.

• Tarjeta de captura o adquisición de imágenes: es la interfaz entre el sensor y la computadora o módulo de proceso que permite al mismo disponer de la información capturada por el sensor de imagen.

• Algoritmos de análisis de imagen: es la parte inteligente del sistema. Su misión consiste en aplicar las transformaciones necesarias y extracciones de información de las imágenes capturadas, con el fin de obtener los resultados para los que hayan sido diseñados.

• Computadora o módulo de proceso: es el sistema que analiza las imágenes recibidas por el sensor para extraer la información de interés en cada uno de los casos implementando y ejecutando los algoritmos diseñados para la obtención de los objetivos.

• Sistema de respuesta en tiempo real: con la información extraída, los sistemas de visión artificial pueden tomar decisiones para alcanzar un objetivo propuesto.

DLL:
Dinamically Linked Library.
Un conjunto de subrutinas o funciones que se ligan a un programa en tiempo de ejecución (en contraposición a aquellas que se anexan en la última etapa de complilación).
Esto permite que diferentes tareas accedan al mismo código.

UMBRALI-ZACIÓN:
Método de segmentación de imágenes que en su versión mas simple consiste en convertir a cada pixel de la imagen en blanco o negro según el nivel de gris original supere o no un determinado valor (umbral).

EROSIÓN:
Operación morfológica mediante la cual se examina cada pixel del color del objeto buscado (en este caso los de color negro de la cinta) y se los cambia de color si en su vecindad hay por lo menos uno del color contrario (blanco).

DILATACIÓN:
Operación morfológica dual de la Erosión, mediante la cual se examina cada pixel del color del fondo, (contrario al objeto buscado: Blanco en este caso) y se los cambia de color si en su vecindad hay por lo menos uno del color contrario (negro).

4.2 Desarrollo de la Práctica

En esta práctica se desarrolla una **DLL** que contenga una función para realizar el reconocimiento de una línea sobre el suelo.
Inicialmente se trata la imagen capturada a fines de aislar la línea del fondo. En este sentido se efectuarán **umbralizaciones**[1] por nivel de color para eliminar sombras, también se aplican **erosiones** y **dilataciones** para desaparecer pequeños objetos y falsas líneas.
Un factor importante para lograr un tratamiento exitoso, es que el mismo se adapte a las condiciones de iluminación, es por ello que se utilizan umbrales dinámicos.

Figura 4 Imagen obtenida desde la webcam.

Puede observarse la línea sobre el suelo.

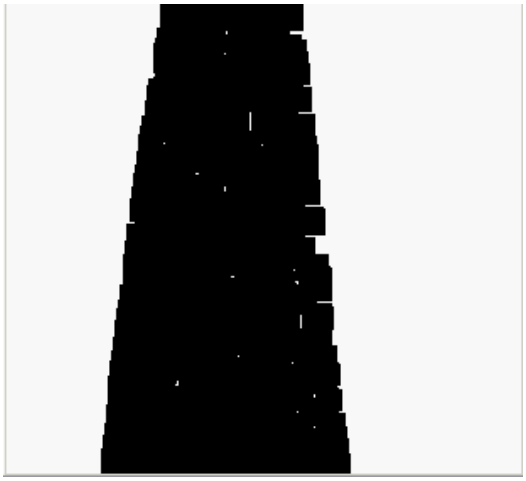

Figura 5 Imagen con post procesamiento.

[1] **R. Gonzalez y R. Woods** *Tratamiento Digital de Imágenes*, Addison-Wesley Publishing Company, 1992, Capítulo 7

Una vez concluido el tratamiento de la imagen para aislar la línea, se aplica el algoritmo de Canny[2] (básicamente un derivador) para resaltar los bordes de dicha línea. Este algoritmo requiere un umbral (en tonos de grises) para filtrar ruidos espurios sobre la imagen derivada, este valor se deja accesible como un parámetro configurable: `int umbral`.

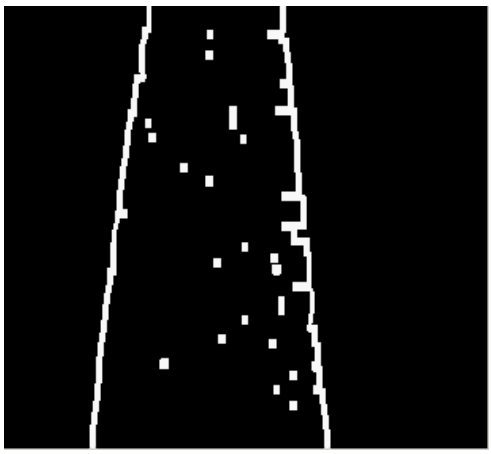

Figura 6 Imagen luego de la derivada

Ahora la idea fundamental es encontrar dos pares de puntos que enmarquen a la línea, es decir, dos pares de puntos que indiquen los bordes de la línea (usando la imagen derivada). Se utilizan dos pares de puntos para luego darles diferentes usos, un par de puntos detectará la línea en la parte superior y el otro par de puntos en la parte inferior.
Dichos puntos serán devueltos en las posiciones de memoria `int* x1, int*y1, int* x2, int*y2, int* x3, int*y3, int* x4, int*y4` pasadas como parámetro por referencia.

La detección de puntos se lleva a cabo trazando dos perfiles sobre la imagen derivada, el ancho de los perfiles puede seleccionarse con el parámetro de entrada `int anchoPerfil` .
Cuanto más ancho sea el perfil más estable es la detección de bordes.

[2] **J. Canny** *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 8, No. 6, Nov 1986.

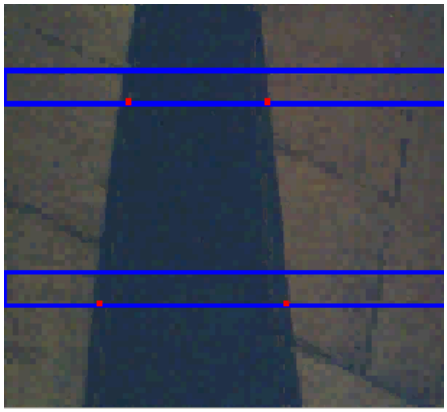

Figura 7 Imagen luego de la detección de bordes.

En color azul se observan los perfiles.

Por último, se agrega un parámetro más `int fin` donde con un 1 se le indica a la función que ese es el último llamado dentro del programa y deberá liberar la memoria tomada dinámicamente.

Para el desarrollo del algoritmo de detección de líneas deben incluirse las librerías OpenCV en el programa en C:

```
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "C:\…\OpenCV\cv\include\cv.h"
#include "C:\…\OpenCV\otherlibs\highgui\highgui.h"
#endif
```

Luego, se definirán los prototipos de las funciones sobre las cuales se trabaja:

```
__declspec(dllexport) int __cdecl giaBuscarPerfilEnLinea(int
anchoPerfil, int umbral, int visualizar, int* x1, int*y1,
int* x2, int*y2, int* x3, int*y3, int* x4, int*y4, int fin);

int giaDetectorBordeHorizontal(IplImage* imagen, int
visualizar, int xini, int yini, int deltaX, int
deltaY, int histeresis, int sentido, int *x, int *y );
```

La expresión `__declspec(dllexport) int __cdecl` indica que esa función podrá ser accedida en forma externa.

Luego, se define el punto de entrada a la DLL:

```
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                             )
{
    switch (ul_reason_for_call)
   {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
               break;
    }
    return TRUE;
}
```

Para la función que devuelve los 4 puntos que delimitan a la línea, es necesario definir tres punteros de imágenes (dos de ellas servirán para hacer cálculos auxiliares). El tipo de puntero a imagen está definido en las librerías de OpenCV y se denomina `IplImage`.
También se definine un puntero de tipo `CvCapture` que apuntará al dispositivo de video (webcam).

```
__declspec(dllexport) int __cdecl giaBuscarPerfilEnLinea(int
anchoPerfil, int umbral, int visualizar, int* x1, int*y1,
int* x2, int*y2, int* x3, int*y3, int* x4, int*y4, int fin)
      {
               IplImage *image = 0, *gray = 0, *edge = 0;
 static CvCapture* captureCAM;
 double minVal;
 double maxVal;
```

Con las siguientes líneas se establece la conexión con la cámara de video y se extrae la imagen.

```
 if(!captureCAM) captureCAM = cvCaptureFromCAM(0);
 if( !cvGrabFrame( captureCAM ))  return -1;
 else
 {
       image = cvRetrieveFrame( captureCAM );
 }
```

Una vez adquirida la imagen se la convierte a tonos de grises y se almacena el puntero a la misma en la variable `gray`.

```
 // Convertir a tonos de grises
 gray = cvCreateImage(cvSize(image->width,image->height),
                      IPL_DEPTH_8U, 1);
 gray->origin = 1;
  cvCvtColor(image, gray, CV_BGR2GRAY);
```

Realizando erosiones y dilataciones consecutivas se eliminan objetos pequeños y aislados que generan ruido en la imagen.

```
//Filtro por tamaño (objetos pequeños)
cvNot(gray,gray); //invierte los colores de la imagen
cvErode(gray,gray,NULL,5);
cvDilate(gray,gray,NULL,5);
```

Luego, se filtran sombras y/o variaciones del color del fondo (tonos de grises claros).
Esto se realiza mediante umbralización binaria.
Para lograr un umbral adaptativo, dependiendo del brillo de la imagen, se ajusta el umbral al 95% del valor máximo detectado.

```
//Filtro por color
cvMinMaxLoc(gray,&minVal,&maxVal,NULL,NULL,NULL);
cvThreshold(gray, gray, maxVal*0.95, 255,
            CV_THRESH_BINARY );
cvNot(gray,gray);
```

Luego, se aplica el algoritmo de Canny para generación de bordes. Se almacena el puntero a esta nueva imagen en la variable `edge`.
Una vez detectados los bordes, se aplica una función de dilatación para engrosarlos.

```
// Umbralización
edge = cvCreateImage(cvSize(image->width,image->height),
                     IPL_DEPTH_8U, 1);
edge->origin = 1;
cvCanny(gray, edge, 0.0f, (float)umbral*5, 3);
cvDilate(edge,edge,NULL,1);
```

Finalmente, se detectan los bordes mediante la aplicación de 4 perfiles:

```
//Detecto los 4 puntos que delimitan la franja
giaDetectorBordeHorizontal(edge,visualizar,0,image-
>height/4, image->width, anchoPerfil, 3, 0, x1, y1);
giaDetectorBordeHorizontal(edge,visualizar,0,image-
>height/4, image->width, anchoPerfil, 3, 1, x2, y2);
giaDetectorBordeHorizontal(edge,visualizar,0,image-
>height*3/4, image->width, anchoPerfil, 3, 0, x3, y3);
giaDetectorBordeHorizontal(edge,visualizar,0,image-
>height*3/4, image->width, anchoPerfil, 3, 1, x4, y4);
```

A continuación se realiza la función que detecta los bordes en un perfil.

```
int giaDetectorBordeHorizontal(IplImage* imagen, int
visualizar, int xini, int yini, int deltaX, int deltaY, int
histeresis, int sentido, int* x, int *y )
{
```

Se definen las variables a utilizar, entre ellas, el vector que contendrá al perfil.

```
int perfil[1000];
int suma = 0;
int i,j,iaux;
```

```
int max, maxPos;
CvScalar pixel;
```

Se genera el perfil requerido tomado de la imagen a analizar, sobre la base de las coordenadas, alto y ancho ingresados como parámetros.

```
//Genero el perfil
for (i=xini;i<xini+deltaX;i++)
{
	for (j=yini;j<yini+deltaY;j++)
	{
		pixel = cvGet2D(imagen,j, i);
		suma = suma + pixel.val[0];
	}
	perfil[i-xini] = suma;
	suma = 0;
}//fin for
```

Se busca el primer máximo dentro del perfil con la finalidad de encontrar el primer borde de la línea.
Se recorre del perfil de derecha a izquierda o en sentido inverso, con el fin de detectar los bordes derecho o izquierdo respectivamente.
Esto se establece con el parámetro de entrada denominado `sentido`.

```
//Busco el máximo
max=0;
maxPos=0;
if (!sentido)
{
	for (i=0;i<deltaX;i++)
	{
		if ( perfil[i] > max)
		{
			max = perfil[i];
			maxPos = i;
		}
		else if ( perfil[i] < max - histeresis)
		{
			break;
		}
	}//fin for

	*x = i + xini;
	*y = yini;
}
else
{
```

```
        for (i=1;i<deltaX;i++)
        {
               iaux = deltaX - i;

               if ( perfil[iaux] > max)
               {
                      max = perfil[iaux];
                      maxPos = iaux;
               }
               else if ( perfil[iaux] < max - histeresis)
               {
                      break;
               }
        }//fin for

        *x = iaux + xini;
        *y = yini;

 }//fin if else

 return 0;
}
```

En x e y se devuelven las coordenadas del máximo encontrado dentro del perfil seleccionado.

A continuación se muestra cómo mediante el algoritmo explicado se reconoce los bordes de una cinta negra situada en el piso mediante la webcam.

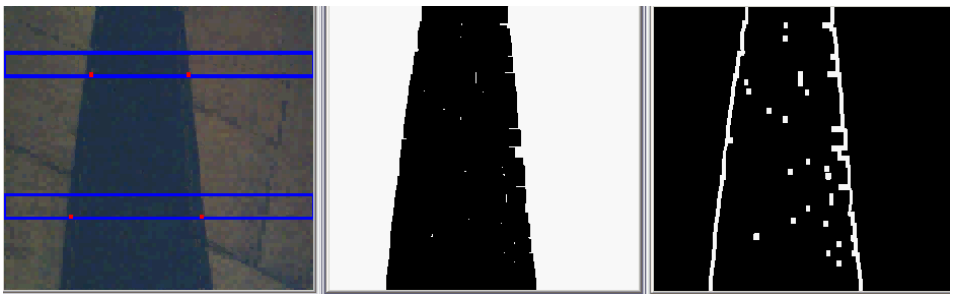

Figura 8 Secuencia de detección de línea.

4.3 Sumario

Durante el desarrollo de esta primera práctica se ha procedido a detallar las funciones necesarias para el desarrollo de un programa de detección de líneas basado en visión artificial que luego, convertido a un archivo DLL permite hacer uso de su funcionalidad en la siguiente práctica y en cualquier otra aplicación que se desee realizar.

5 Práctica N° 2: Desarrollo de algoritmo de control de trayectoria para seguimiento de una línea sobre el piso

Autores: **Leandro M. Di Matteo, Andrea C. Mangone**

5.1 Introducción Teórica

SISTEMA DE CONTROL: Es una combinación de componentes, mecánicos, electrónicos, etc. que actúan coordinadamente para mantener al sistema cerca del punto de funcionamiento deseado .

Un **sistema de control** permite detectar una desviación de los parámetros pre-establecidos del funcionamiento normal de un sistema, para llevar al sistema nuevamente a sus condiciones operacionales normales de funcionamiento.
Existen dos tipos de sistemas de control que se denominan de lazo abierto y de lazo cerrado.
Los sistemas de control de lazo abierto son aquellos donde la salida no tiene efecto sobre la acción de control. En otras palabras, en un sistema de control de lazo abierto la salida ni se mide ni se retroalimenta para compararla con la entrada. Por lo tanto, para cada entrada de referencia corresponde una condición de operación fija. Así, la precisión del sistema depende de la calibración. En presencia de perturbaciones, un sistema de control de lazo abierto no cumple su función asignada. Un control de lazo abierto sólo se puede utilizar si la relación entre la entrada y la salida es conocida, y si no se presentan perturbaciones tanto internas como externas.
Los sistemas de control de lazo cerrado la variable ha ser controlada (Variable controlada) es continuamente medida y así comparada con un valor predeterminado (Variable de referencia). Si existe una diferencia entre estas dos variables (error E o desviación del sistema), se realizan ajustes hasta que la diferencia cuantificada es eliminada y la variable controlada iguala la variable de referencia.
Para retroalimentar el sistema buscando minimizar el error entre el valor deseado y el valor medido, uno de los métodos de control más utilizado es el control Proporcional Integrativo Derivativo (PID).
El control PID trata de minimizar el valor absoluto de la variable de error E. Cuando $E = 0$ significa que el valor de la variable a controlar actual es igual al valor deseado. Si $E > 0$, significa que el valor actual de la variable está por debajo del valor deseado. Si $E < 0$, el valor actual está por arriba del valor deseado.
El control PID consiste en combinar tres tipos de controles: proporcional – integrativo – derivativo, a fin de tener amplia variedad de respuestas y poder lograr ajustes óptimos.

La ecuación utilizada es la siguiente:

$$Y = Kp * E + Ki * \int_0^t E(t).dt + Kd * \frac{dE}{dt}$$

Para llevar a cabo este tipo de control en una PC, microprocesador, etc., debe construirse un software que cumpla con el siguiente diagrama en bloques:

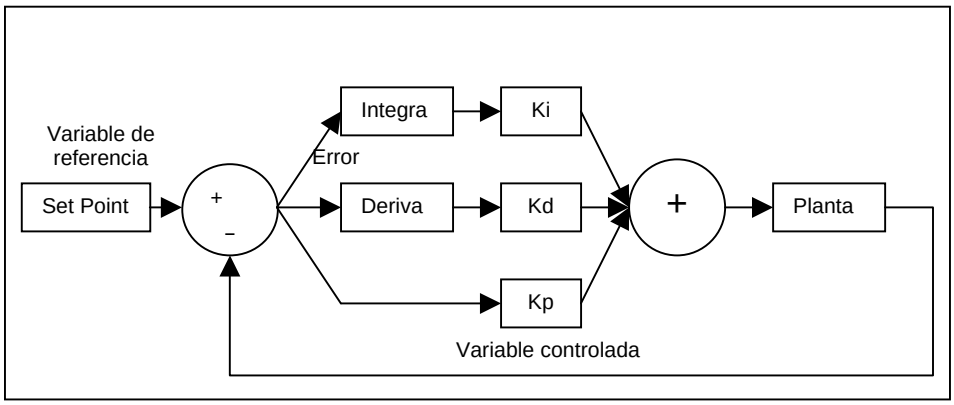

Figura 9 Lazo de control PID.

Ajustando las constantes Kp, Ki y Kd, se obtienen diversos tipos de respuestas, como las que se ilustran a continuación:

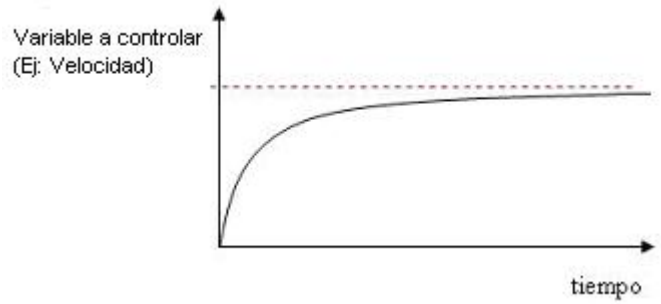

Figura 10 Respuesta de un control proporcional.

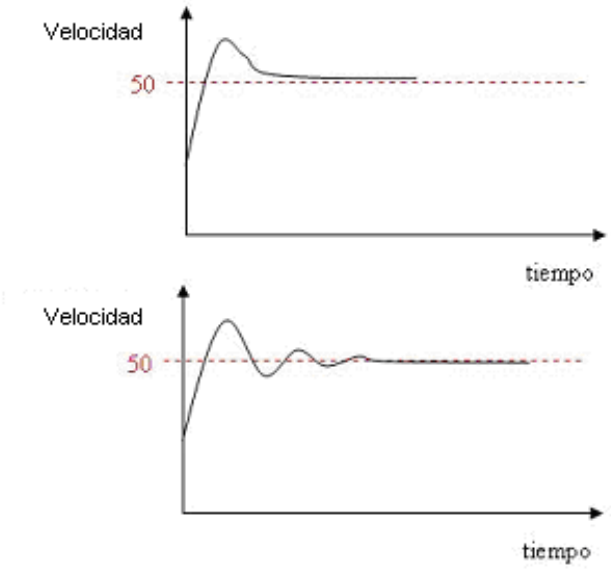

Figura 11 Diferentes tipos de respuestas de un control PID.

Existe una determinada combinación de las constantes, donde se logra una situación de compromiso entre velocidad de respuesta y precisión, logrando la combinación óptima, obteniendo una respuesta del tipo:

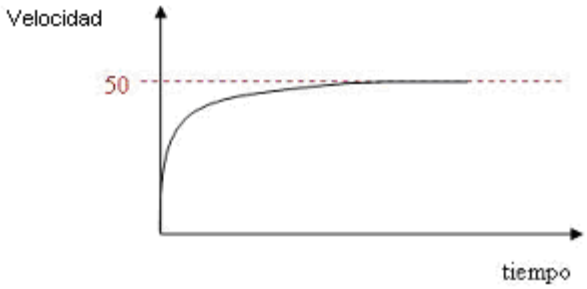

Figura 12 Respuesta de un control PID ajustado para amortiguamiento crítico.

El ajuste adecuado de estas constantes para obtener la curva anterior, denominada amortiguamiento crítico, se logra teniendo un excelente conocimiento del medio y su respuesta a estímulos.

5.2 Desarrollo de la práctica.

Se desarrolla un programa en C# para que el robot ER1 siga la línea indicada en el suelo.
Para ello se utiliza la DLL confeccionada en el trabajo práctico anterior.
En este programa se trabaja sobre el lazo de control para ajustar dinámicamente las velocidades de las ruedas para que el robot se mantenga sobre la línea.

Se deberá crear un formulario con los siguientes objetos:

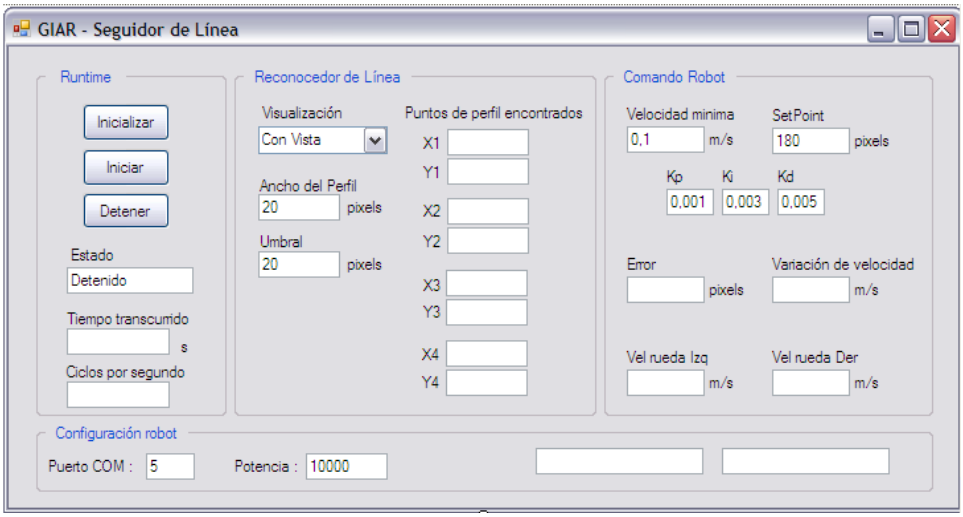

Figura 13 Interfaz del programa.

En el botón Inicializar, se programan las sentencias de inicialización de los motores,

```
private void button1_Click(object sender, EventArgs e)
```

En el botón Iniciar, se genera un bucle permanente ejecutando las rutinas de visión artificial y control, para que el robot se desplace sobre la línea. Este bucle se detendrá únicamente al pulsar el botón detener,

```
private void button2_Click_1(object sender, EventArgs e)
```

Y el botón detener, para los motores y escribe el nuevo estado en el campo de texto correspondiente, lo cual indica al bucle iniciado con el botón Iniciar que debe detenerse,

```
private void button3_Click_1(object sender, EventArgs e)
```

Se usa una simple función para la conversión de tiempo en formato hh:mm:ss a segundos, para calcular el tiempo transcurrido desde que se inició la corrida y también permite calcular la cantidad de ciclos por

segundo que se están ejecutando. Los ciclos por segundo brindan una buena indicación de la regularidad de ejecución del proceso,

```
int ConvertirAtiempoAbsoluto(DateTime Horafecha)
```

Dentro del bucle de ejecución se llama a una rutina troncal, `proceso`, la cual será la encargada de llamar a la rutina de visión artificial, procesar sus datos y ejecutar el lazo de control,

```
private void proceso(Int32 fin)
```

Por lo tanto, se desarrolla una función donde se implementa un control PID para corregir la trayectoria del robot, a fin de mantenerlo corriendo sobre la línea.
La subrutina que se realiza estará preparada para trabajar en un lazo PID, es por ello que pasamos como parámetro las constantes de ajuste Kp, Ki y Kd.
Una vez ejecutado el lazo de control, se adiciona una velocidad mínima constante a cada rueda (velMin) para que el robot no solamente se posicione sobre la línea sino que se mantenga en movimiento todo el tiempo hacia delante,

```
private void control(double setPoint, double velMin, double
        Kp, double Ki, double Kd, double x1, double x2,
        double x3, double x4)
```

Una vez definidas las velocidades que adoptará cada rueda, en m/s, se debe construir una rutina que escale estos valores a unidades legibles por el driver del ER1.
Además, aquí se limitan las velocidades para no dañar los motores o generar resbalamiento,

```
int ComandoER1(double vl, double vr, double potencia)
```

Desarrollo del software.

En las inclusiones se considera el driver de control de motores `DriverER1`:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices; //Agregar esta linea
para usar dll import
using DriverER1;  //Agregar esta línea para usar el driver
del ER1

            namespace WindowsApplication1
      {
            public partial class Form1 : Form
              {
```

Dentro de la clase form1 se realiza la llamada a la función contenida dentro de la DLL creada en el trabajo práctico anterior.
La definición de parámetros con `[MarshalAs(UnmanagedType.LPArray)] Int32[]` permite trabajar con pasaje de variables por referencia.

```
[DllImport("borrar4.dll", EntryPoint =
"giaBuscarPerfilEnLinea", ExactSpelling = false,
CallingConvention = CallingConvention.Cdecl)]

static extern int giaBuscarPerfilEnLinea(int anchoPerfil,
int umbral, int visualizar,
[MarshalAs(UnmanagedType.LPArray)] Int32[] x1,
[MarshalAs(UnmanagedType.LPArray)] Int32[] y1,
[MarshalAs(UnmanagedType.LPArray)] Int32[] x2,
[MarshalAs(UnmanagedType.LPArray)] Int32[] y2,
[MarshalAs(UnmanagedType.LPArray)] Int32[] x3,
[MarshalAs(UnmanagedType.LPArray)] Int32[] y3,
[MarshalAs(UnmanagedType.LPArray)] Int32[] x4,
[MarshalAs(UnmanagedType.LPArray)] Int32[] y4,
int fin);
```

Se crea un objeto instanciado de la clase de control de motores.

```
public RCMotion RCMDev = new RCMotion();
```

Dentro del evento del Botón 1 (Inicializar), se programa la inicialización del driver de los motores, donde el puerto COM instalado por dicho driver puede configurarse en el `textBox23`.

```
private void button1_Click(object sender, EventArgs e)
      {

          //Inicializar ER1
          ulong dwErr;
          int puerto;

        /* Para inicializar el objeto, se pasa el numero de
        puerto serie (COM) por el cual se controlaran el
        driver de los motores y se inicializa  el mismo */

          puerto = Convert.ToInt16(textBox23.Text);
          dwErr = (ulong)RCMDev.RCMInit((int)puerto);
          RCMDev.WheelInit();
       }
```

Con el evento del Botón 2 (Iniciar), se genera un bucle donde correrá el programa bajo estudio. Dicho bucle finalizará cuando se pulse el botón 3 (Detener).
Dentro del bucle se calcula el tiempo transcurrido (en segundos) y los ciclos de programa ejecutados por segundo (cps). Allí es donde se llama a la rutina principal del programa, Proceso( ), la cual dispara la lectura de los puntos sobre le línea y ejecuta el comando de los motores para que el robot se mantenga sobre la línea.
Se agrega una sentencia `Application.DoEvents();` para que no se bloqueen las restantes tareas corriendo sobre la computadora.

```
private void button2_Click_1(object sender, EventArgs e)
{
      DateTime Tiempoinicial;
      Int32 Tiempotranscurrido;
      Int32 contador;
      double cps;

      Tiempoinicial = DateTime.Now;
      contador = 0;
      textBox9.Text = "corriendo ...";

      while (!string.Equals(textBox9.Text, "Detenido"))
      {
        contador = contador + 1;

        Tiempotranscurrido =
        ConvertirAtiempoAbsoluto(DateTime.Now) –
        ConvertirAtiempoAbsoluto(Tiempoinicial);

        cps = contador /
        (Convert.ToDouble(Tiempotranscurrido)+0.01);

        textBox21.Text =
        Convert.ToString(Tiempotranscurrido);

        textBox22.Text = Convert.ToString(Math.Round(cps));

        Application.DoEvents();
        proceso(0);
      }
 }
```

Con el botón 3 (Detener) se detiene el proceso en ejecución, llevando a cero la potencia y velocidad de las ruedas del robot y escribe `"Detenido"` en el `textBox9` que es utilizado como indicador para cortar el bucle mencionado en la rutina mostrada anteriormente.
También se observa el llamado por última vez a la rutina `proceso(1)` , donde el 1 pasado como parámetro le indica que debe liberar la memoria que está utilizando.

```
private void button3_Click_1(object sender, EventArgs e)
{

    textBox9.Text = "Detenido";
    proceso(1);

    RCMDev.WheelSetVelocity(0);
    /*
     * setea la potencia de ambos motores del ER1 a 0
     */
    RCMDev.WheelSetPower(0);
    /*
     * manda el comando al driver de los motores
     */
    RCMDev.WheelUpdate();


}
```

Esta es la rutina auxiliar, utilizada anteriormente para llevar el tiempo de ejecución de programa a una base de tiempo común, medido en segundos:

```
int ConvertirAtiempoAbsoluto(DateTime Horafecha)
{
        return Horafecha.Second + Horafecha.Minute * 60 +
        Horafecha.Hour * 60 * 60;
}
```

Ahora se trabaja sobre la función que controla el seguimiento de la línea:

```
private void proceso(Int32 fin)
      {
```

Se definen las variables a utilizar,

```
Int32 vista;
Int32 AnchoPerfil;
Int32 Umbral;
Int32 i;

System.Double setPoint, velMin, Kp, Ki, Kd;
```

Donde las siguientes variables se definen como vector de una posición, a los efectos de poder pasarlas por referencia a la rutina de visión artificial,

```
Int32[] x1 = new Int32[1];
Int32[] y1 = new Int32[1];
Int32[] x2 = new Int32[1];
Int32[] y2 = new Int32[1];
Int32[] x3 = new Int32[1];
Int32[] y3 = new Int32[1];
Int32[] x4 = new Int32[1];
Int32[] y4 = new Int32[1];
```

Luego se realizan las conversiones correspondientes para adaptar los parámetros de entrada de la rutina de la búsqueda de la línea.
La variable `vista` indicará si la rutina de procesamiento de imágenes mostrará el procesamiento en pantalla o no.
Y las variables `AnchoPerfil` y Umbral se utilizarán para determinar el ancho de la zona de búsqueda sobre la imagen y el umbral de corte para la eliminación de ruido, respectivamente.

```
if (string.Equals(comboBox1.Text, "Con Vista"))
{
  vista = 1;
}
else
{
  vista = 0;
}


AnchoPerfil = Convert.ToInt32(textBox10.Text);
Umbral = Convert.ToInt32(textBox11.Text);
```

Aquí se llama a la rutina de detección de línea, a la cual se le pasa los parámetros correspondientes, en particular x1, y1, x2, y2, x3, y3, x4, y4 son pasados por referencia pues es allí donde se depositarán los valores retornados por esta rutina:

```
giaBuscarPerfilEnLinea(AnchoPerfil, Umbral , vista, x1, y1,
x2, y2, x3, y3, x4, y4, fin);
```

Se muestran dichos valores en los cuadros de texto correspondientes:

```
textBox1.Text = Convert.ToString(x1[0]);
textBox2.Text = Convert.ToString(y1[0]);
textBox3.Text = Convert.ToString(x2[0]);
textBox4.Text = Convert.ToString(y2[0]);
textBox5.Text = Convert.ToString(x3[0]);
textBox6.Text = Convert.ToString(y3[0]);
textBox7.Text = Convert.ToString(x4[0]);
textBox8.Text = Convert.ToString(y4[0]);
```

Entonces, se toman los parámetros de control y se realizan las conversiones correspondientes antes de llamar a la rutina de control.

```
setPoint = Convert.ToDouble(textBox20.Text);
velMin  = Convert.ToDouble(textBox12.Text);
```

```
            Kp = Convert.ToDouble(textBox13.Text);
            Ki = Convert.ToDouble(textBox14.Text);
            Kd = Convert.ToDouble(textBox15.Text);

            control(setPoint, velMin, Kp, Ki, Kd,
                   Convert.ToDouble(x1[0]),
                   Convert.ToDouble(x2[0]),
                   Convert.ToDouble(x3[0]),
                   Convert.ToDouble(x4[0]));

        }
```

Se desarrolla ahora la subrutina de control, a la cual se le pasa la ubicación de la línea (x1, x2, x3, x4) y la posición que se desea mantenga el robot sobre ella (setPoint).
Dentro de esta rutina se puede implementar un lazo cerrado de control tipo PID o bien algún lazo de control relacionado a la inteligencia artificial, por ejemplo por lógica difusa.
La subrutina que realizaremos estará preparada para trabajar en un lazo PID, es por ello que pasamos como parámetro las constantes de ajuste Kp, Ki y Kd.
Una vez ejecutado el lazo de control, se adiciona una velocidad mínima constante a cada rueda (velMin) para que el robot no solamente se posicione sobre la línea sino que se mantenga en movimiento todo el tiempo hacia delante.

```
        private void control(double setPoint, double velMin, double
        Kp, double Ki, double Kd, double x1, double x2, double x3,
         double x4)
        {

            double  error, error1, error2;
            double  vl, vr;
            double potencia;
```

A fin de mantener el robot centrado sobre la línea se calcula el error a minimizar de la siguiente manera:

```
            error2 = (x3 + x4) / 2 - setPoint;
            error = error2;
```

Por simplicidad se implementa un control proporcional (P), donde a las velocidades mínimas de cada rueda se le adicionará y restará velocidad proporcionalmente con el error, así de esta manera se encausará el robot a quedar sobre el setpoint indicado:

```
            vl = velMin + Kp * error;
            vr = velMin - Kp * error;

            potencia = Convert.ToDouble(textBox24.Text);
```

Finalmente, se llama a la rutina que le otorga la velocidad a ambos motores, derecho (vr) e izquierdo (vl), a una potencia dada que es ajustable por el usuario.

```
        //Movimiento de motores
        ComandoER1(vl, vr, potencia);

        //Visualización
        textBox16.Text = Convert.ToString(error);

    }
```

Se desarrolla ahora el módulo para enviar la velocidad a los motores:

```
    int ComandoER1(double vl, double vr, double potencia)
    {
        ushort pote;
        Int32 vlLINEALIZADA, vrLINEALIZADA;
```

Se limitan las velocidades de entrada (-1.5 a -0.03) y (0.03 a 1.5) m/s.

```
        //Limito el rango de operación
        if (vl < -1.5) vl = -1.5;
        else if (vl > -0.03 && vl < 0.03) vl = 0.0;
        else if (vl > 1.5) vl = 1.5;
        if (vr < -1.5) vr = -1.5;
        else if (vr > -0.03 && vr < 0.03) vr = 0.0;
        else if (vr > 1.5) vr = 1.5;
```

Se muestran dichas velocidades en los correspondientes cuadros de texto

```
        textBox18.Text = Convert.ToString(vl);
        textBox19.Text = Convert.ToString(vr);
```

y se ajusta la potencia de los motores,

```
        //Ajuste de potencia
        pote = Convert.ToUInt16 (potencia);
        RCMDev.WheelSetPower(pote);
```

Luego se escalan los valores en m/s a la unidad de medida del driver del ER1, estos valores han sido relevados con un tacómetro, hallando la relación aproximada que se indica debajo,

```
        //Linealizo velocidad de ruedas
        vlLINEALIZADA = Convert.ToInt32(2000000.0 * vl);
        vrLINEALIZADA = Convert.ToInt32(2000000.0 * vr);
```

Se muestran dichas velocidades en los correspondientes cuadros de texto

```
        textBox25.Text = Convert.ToString(vlLINEALIZADA);
        textBox26.Text = Convert.ToString(vrLINEALIZADA);
```

y por último se envían los valores de velocidad a los motores,

```
            //Ajusto velocidad en ambos motores
            RCMDev.SendCmd_DW(1, 17, -vlLINEALIZADA);
            RCMDev.SendCmd_DW(0, 17, vrLINEALIZADA);

            //Actualizo velocidades
            RCMDev.WheelUpdate();

            return 0;
        }
```

Se obtiene lo siguiente,

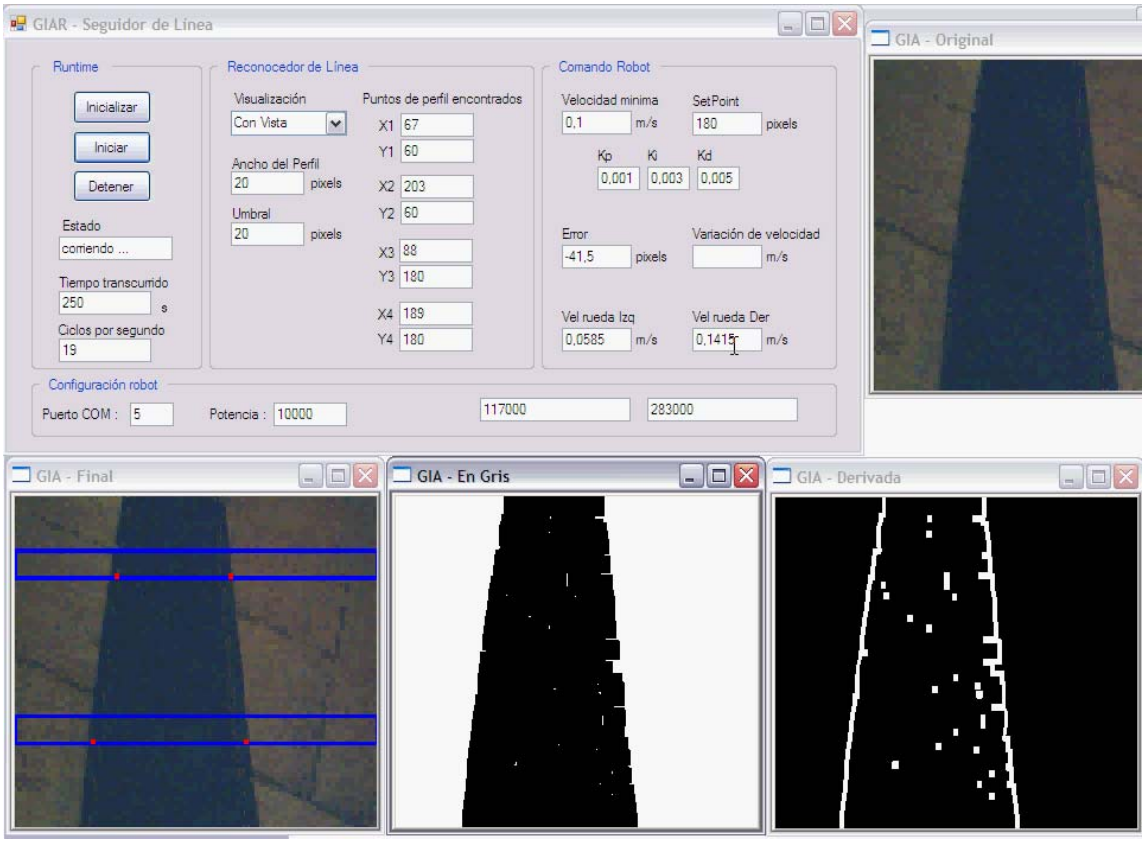

Figura 14 Programa en funcionamiento.

5.3 Sumario

En esta práctica se ha explicado cómo realizar un programa de control de trayectoria siguiendo una línea proyectada en el piso, donde el camino no es conocido de antemano, sino que el entorno se conoce y evalúa en tiempo real a través del uso de funciones de visión artificial (archivo dll). En esta práctica se integran visión artificial y acciones de control para el seguimiento de una trayectoria en un entorno dinámico.