Seguimiento de Objetos empleando Aforge.Net y Arduino.
Tracking Objects using Aforge.Net and Arduino.

Ing. José Armando Sáenz Esqueda
Maestro. Facultad de Ingeniería, Ciencias y Arquitectura de la Universidad Juárez del Estado de Durango. México
E-mail: jase1588@gmail.com

Dr. Agustín Sáenz López.
Maestro. Facultad de Ingeniería, Ciencias y Arquitectura de la Universidad Juárez del Estado de Durango. México
Coordinador de Investigación
E-mail: agusgplmx@yahoo.com.mx

M.C. Vicente Reyes Espino
Maestro. Facultad de Ingeniería, Ciencias y Arquitectura de la Universidad Juárez del Estado de Durango. México
Área de investigación

M.C. Rafael Morales Salazar
Maestro. Facultad de Ingeniería, Ciencias y Arquitectura de la Universidad Juárez del Estado de Durango. México
Área de investigación

Recibido: 01-01-14
Aceptado: 25-02-14

Resumen:

En este trabajo se presenta la implementación de un sistema de visión empleando una cámara de video web. El sistema consta de un servomotor Futaba de giro continuo el cual recibe señales de

velocidad. El objetivo es lograr el seguimiento de un objeto empleando la cámara para conocer la posición de este objeto en cada instante. El procesamiento de las imágenes se realizó con las librerías Aforge.NET. Todo el programa se desarrolló empleando un lenguaje de programación C# y para el envío de las señales se emplea un dispositivo Arduino Uno.

Palabras clave: Visión, Aforge.Net, Servomotor Futaba, Arduino Uno

Abstract:

In this paper presents an implementation of a vision system using a video camera web. The system consists of a Futaba servo for continuous rotation which receives signals from speed. The goal is to achieve the tracking of an object using the camera for the position of this object at every moment. The processing of the images was performed with the Aforge libraries .NET. The whole program was developed using a C# programming language and for the sending of the signals is employs a device One Arduino.

Keywords: Vision, Aforge.Net, Servomotor Futaba, One Arduino

Introducción:

Los sistemas basados en visión han tenido una gran cantidad de aceptación en aplicaciones para ingeniería principalmente en la robótica. En la última década se han realizado una gran cantidad de investigaciones respecto a los sistemas visión porque con solamente el sensor que viene siendo una cámara es posible obtener mucha más información que con los sensores típicos como lo son, laser o sonares, que requerían de varios de estos dispositivos para la obtención de la información. Otra gran ventaja que tienen las cámaras respecto a los sensores, es el bajo costo ya que es posible emplear cámaras comunes como lo son las webcam obteniendo con ellas muy buenos resultados, mientras que los otros sensores, necesitan ser construidos específicamente para la aplicación en que se le vayan a usar. Los datos que se pueden obtener con una cámara, pueden ser los colores, contornos o bordes, texturas, distancias, etc. Debido a la gran cantidad de información que se obtienen es necesario aplicar cálculos muy elaborados que requieren de un microcontrolador muy potente o de las mini-pc como son las Beagle Board. Las últimas generaciones de procesadores que ya contienen varios núcleos, son perfectos para esta clase de aplicaciones debido a que la velocidad de procesamiento que tienen es tan alta, que incluso el procesamiento de imágenes que requiere de grande cálculos y que trabaja con matrices de dimensiones de 620x480 pueden ser procesadas en tiempo real.

En este trabajo se va a trabajar con un arquitectura que emplea una laptop, un arduino uno, una cámara PS EYE y un servo motor Futaba de giro continúo. El objetivo es que empleando las librerías de visión de Aforge .NET se procesen las imágenes obtenidas de la cámara y está a su vez mueva la cámara centrando un objetivo que se va a especificar con colores. La técnica para

controlar el motor es conocida en control automático como controlador On/Off el cual se va a encargar de corregir la posición del motor y de la cámara para centrar el objeto.

Algunas de las aplicaciones que se le han encontrado es la de operar robots en zonas de desastre como lo es el robot Finder (Eleazar, y otros, 2012). Este es un robot móvil con ruedas, el cual está equipado con cámara Kinect, la cual tiene una cámara de colores VGA y una cámara infrarroja. La cámara infrarroja sirve para detectar masa e indica la distancia a la que encuentre, y esto combinado con la cámara VGA es posible conocer las dimensiones del objeto u objetos que se tienen enfrente, de esta manera es posible evadirlos. También existen en la Internet (red) una gran cantidad de librerías para detectar movimientos del cuerpo humano con el Kinect. Debido a esto último, ha habido una gran aceptación de esta cámara en la ingeniería, ya que facilita conocer los ángulos de los brazos de una persona y con esto poder controlar brazos robóticos desde una cierta distancia (Teleoperación).

Una de las áreas de visión que han sido investigadas últimamente, es el desarrollo de sistemas que puedan interactuar con las personas. Tal es el caso de los robots ASIMO o el robot NAO (Hernandez & Ibarra, 2012). Estos dos robots son del tipo humanoide y están equipados con un sistema de visión el cual les permite interactuar con las personas. El más famoso es el robot ASIMO ya que fue el primero en caminar tomando de la mano con una persona. Así también en Japón algunas escuelas cuentan con el robot ASIMO para que interactúe con los estudiantes y estos se acostumbren a tratar con un robot ya que se tiene planeado en ese país, que los robots se encarguen de las tareas domésticas más básicas como lo es limpieza, lo más rápidamente posible..

En la referencia (García, Cárdenas, Rendón, & Méndez, 2009) se habla de cómo se han rehabilitado líneas de manufactura que se creían obsoletas empleando sistemas de visión. Debido a que las cámaras son de fácil acceso, algún sistema electromecánico o mecatrónico que se considere obsoleto puede ser rehabilitado. Se consideran obsoletos aquellos robots que tengan que ser teleoperados manualmente, pero añadiendo una cámara a un sistema es posible hacerlo autónomo sin necesidad de agregar algo más a su arquitectura.

Por último para conocer las diversas técnicas de procesamiento de imágenes se pueden consultar las referencias (Hu, Fans, Tang, Xu, & Sun, 2010), (Miyajima, Thomas, & Amano, 2012) y (Ye, Yuetian, Yunhe, Shu, & Yuchen, 2010). En ellas se explican programas realizados con la librería "OpenCV" y que se programa en lenguaje C/C++ y el cual actualmente es la librería de visión por excelencia ya que al ser "Código Libre" es posible modificarlo de acuerdo a las necesidades del usuario y para lo cual existen una cantidad de artículos sobre este tema en la Internet (red). En el apartado **¡Error! No se encuentra el origen de la referencia.** se explica la arquitectura del sistema, haciendo énfasis en el programa que se carga en el Arduino para realizar el movimiento del servomotor. En el apartado **¡Error! No se encuentra el origen de la referencia.** se presenta el procesamiento de imágenes (segmentación e interpretación), así como la interfaz desarrollada para seguir al objetivo y que envía las señales al Arduino. En el apartado 4 se presenta los resultados experimentales y en el apartado 5 se presentan las conclusiones.

Arquitectura

Un sistema que trabaja con visión obtiene las imágenes de una cámara, que en este caso es una PS EYE (Véase Figura 2.- Cámara PS EYE) y que se emplea principalmente como una cámara WebCam. Para el procesamiento de imágenes se utiliza una laptop. El programa que realizara el

procesamiento de imágenes se desarrolló en lenguaje C# utilizando las librerías Aforge .NET y empleando un compilador Visual C# 2010 Express. Para la actuación se va a emplear un Arduino UNO (Véase Figura 1.- Arduino Uno) el cual se comunica vía USB con la laptop emulando un puerto serial. El dispositivo Arduino tiene en algunas de sus terminales la capacidad de generar señales PWM ("Pulse Width Modulation" por sus siglas en ingles "Modulación por Ancho de Pulso") y en se usan estas terminales para indicar la posición del motor. El motor que se conecta es un servomotor Futaba de giro continuo modelo SM-S4303R (Véase Figura 3.- SM-S4303R).

Figura 1.- Arduino Uno

Figura 2.- Cámara PS EYE

Figura 3.- SM-S4303R

La cámara se monta sobre el motor utilizando una corona en forma de hélice que viene como un accesorio del motor. El Arduino va a mandar señales de sentido del giro al motor, de tal manera que el motor va a girar a una misma velocidad solamente que va a cambiar el sentido de giro. El diagrama de conexión se muestra en Figura 4. Para conocer el sentido del giro, la laptop lee las imágenes que son tomadas por la cámara, con las cuales se realizara un procesamiento de imágenes (segmentación e interpretación) el cual se explicara más adelante.

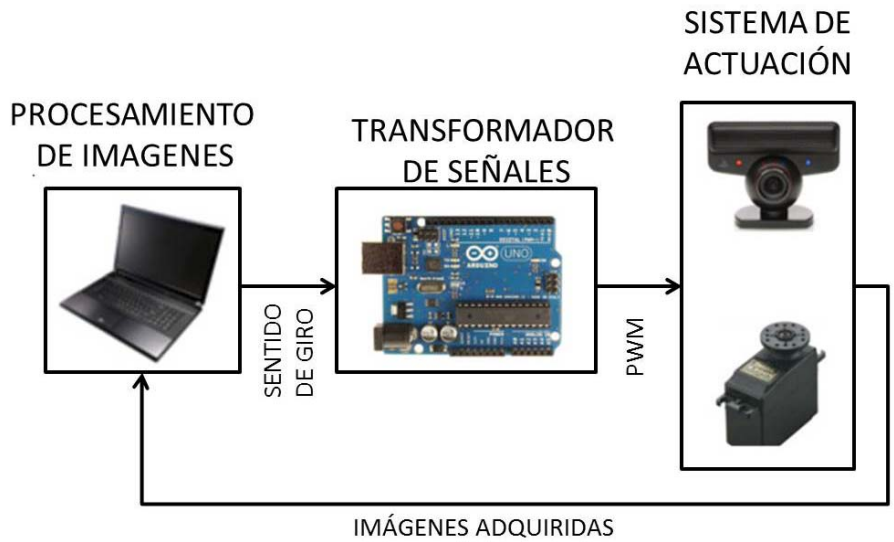

Figura 4.- Arquitectura del sistema

El programa que se instala en el Arduino es el siguiente:

```
# include <Servo.h>

Servo myservo;
```

```
int dato;
void setup()
{
  myservo.attach(9);
  Serial.begin(9600);
}
void loop()
{
  if(Serial.available()>0)
  {
    dato = Seria.read();
    switch((char)dato)
    {
      case 'l':
        myservo.write(80);
      break;
      case 'r':
        myservo.write(100);
      break;
      case 's':
        myservo.write(0);
      break;
    }
  }
}
```

La cabecera del programa debe ser definida como "*Servo.h*", esta librería contiene los métodos para mandar una señal PWM por el puerto que se le indique, con la frecuencia necesaria para mover un servomotor Futaba. Con el comando ***Servo myservo*** se crea un objeto de la clase *Servo* que servirá para establecer el puerto de comunicación e indicar las señales que se deben enviar. Se declara la variable ***dato*** del tipo **int** en donde se va almacenar el carácter que será enviado desde la laptop para indicar el sentido de giro. Después se declara el método ***setup*** que es la primera rutina que corre el microcontrolador del Arduino y en el cual se deben de especificar las condiciones iníciales. El comando ***myservo.attach(9)*** es donde se especifica que la señal PWM saldrá por el puerto digital 9 y es en donde deberá estar conectada la terminal de datos del servomotor. ***Serial.begin(9600),*** sirve para inicializar el puerto serial del Arduino con a una velocidad de **9600 baudios/segundo**.

Enseguida se tiene el método ***loop*** el cual es un ciclo infinito y se empieza a ejecutar una vez que finaliza el método ***setup***. La primera línea de comando es una condición ***if*** que contiene la operación de decision ***Serial.available() > 0, Serial.available(),*** es un método de tipo entero la cual es cero si no se tiene información en el buffer de entrada, una vez que existe información en el buffer de entrada este método regresa un valor diferente de cero. Cuando se detecta que ha llegado información por el puerto serial se vacía en la variable ***dato*** por medio del método ***Serial.read()***, una vez que se ha ejecutado este último comando el buffer de entrada vuelve a ser cero. Cuando ya se tiene el dato almacenado se ejecuta la sentencia de selección múltiple ***switch***. Se pueden recibir 3 caracteres para detectar el sentido de giro del motor o detenerlo. Cuando se recibe el carácter "*l*" significa que el motor debe de girar a la izquierda y se logra escribiendo un 80 en el puerto PWM, si se recibe el carácter "*r*" significa que el motor debe de girar a la derecha escribiendo un 100 en el puerto PWM y por último si se recibe el carácter "s" significa que el motor

debe de detenerse lográndose esto escribiendo un 90 en el puerto PWM. El compilador empleado es la versión 0019 y puede ser descargado de la página http://arduino.cc/.

Procesamiento de Imágenes

El procesamiento de imágenes por computadora se realiza en dos fases que son segmentación e interpretación. La primera fase (segmentación) se realiza para eliminar la información que esta presente en la imagen y no es requerida, la segunda fase (interpretación) se aplica una vez que se tiene la información filtrada por la segmentación y se procede hacer cálculos como el área y la ubicación del centro geométrico o centroide del objeto a seguir. Existen muchos otros procedimientos para la interpretación si se desea conocer más acerca de estos procedimientos se puede consultar la referencia (Pajares & De la Cruz, 2008). La interfaz desarrollada se observa en la Figura 5 consta de tres **PictureBox** nombrados ***pbCámara***, ***pbFiltro*** y ***pbSeguimiento*** y son los objetos donde se van a ir mostrando los procesos de visión. ***pbCámara*** muestra la imagen obtenida directamente de la cámara todavía sin aplicar algún proceso. ***pbFiltro*** es donde se va a mostrar el resultado de la segmentación y donde se elimina toda la información no necesaria, y se muestra únicamente el objeto que se desea seguir. En ***pbSeguimiento*** se muestra el resultado del proceso de interpretación enmarcando en un recuadro de color rojo el objeto que se identificó y las coordenadas del centroide.

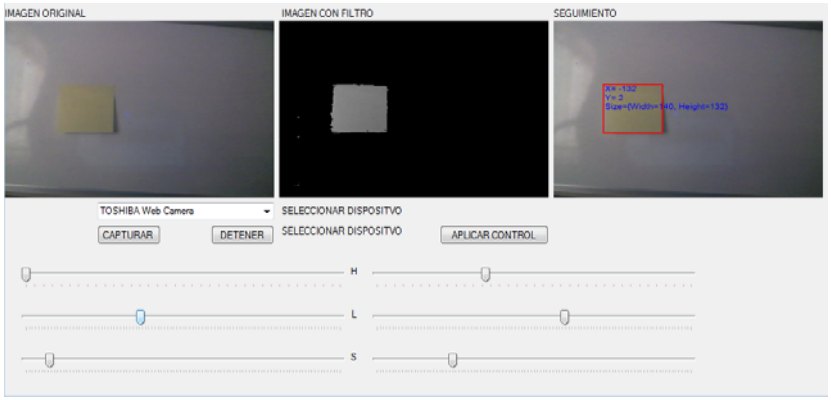

Figura 5.- Interfaz desarrollada en Visual Studio

Debajo de ***pbCámara*** se presenta un ***ComboBox*** llamado ***cbCámara*** y en el se cargan todas las cámaras que se encuentren disponibles en la computadora. Para saber reconocer las cámaras que esta disponibles se manda llamar una rutina en el constructor de la forma llamado ***BuscarDispositivos()*** y también se agregar la instrucción "***CheckForIllegalCrossThreadCalls = false***" para evitara problemas mas adelante al implementar hilos de programación para tratar la imagen. El método ***BuscarDispositivos()*** contiene la siguiente programación:

```
Dispositivos = new FilterInfoCollection(FilterCategory.VideoInputDevice);
if (Dispositivos.Count == 0)
   Existen = false;
else
{
   Existen = true;
   CargarDispositivos();
}
```

En esta rutina se declara una variable global llamada ***Dispositivos*** de la clase ***FilterCollection*** y una variable ***Existen*** del tipo ***Bool***. En la primera línea esta variable crea una instancia con el comando ***new FilterInfoCollection*** dando como parámetro el valor ***FilterCategory.VideoInputDevice***. En la segunda línea de programación esta la instrucción ***if*** teniendo como condición lógica ***Dipositivos.Count*** el cual regresa el número de cámaras detectadas, en caso de ser cero se vacía en la variable ***Existen*** el valor de ***false***, pero en caso de ser diferente de cero en la variable ***Existen*** se vacía el valor ***true*** y se ejecuta la rutina ***CargarDispositivos()***. La programación de este último método es la siguiente:

```
for (int i = 0; i < Dispositivos.Count; i++)
   cbCamara.Items.Add(Dispositivos[i].Name.ToString());
cbCamara.Text = cbCamara.Items[0].ToString();
```

Consta de un ciclo ***for*** el cual va recorriendo el objeto ***Dispositivos*** mediante la instrucción ***cbCámara.Items.Add(Dispositivos[i].Name.ToString())*** va agregando el nombre de las cámaras detectadas en ***cbCámara***. Al finalizar el ciclo pone como selección predefinida la cámara que se encuentra en la posición 0 del objeto ***Dispositivos***.

Tambien se tienen seis ***TrackBar*** llamados ***tbHmin***, ***tbHmax***, ***tbLmin***, ***tbLmax***, ***tbSmin*** y ***tbSmax***. La segmentación que se va a implementar va a utilizar un filtro **HLS** por sus siglas en ingles significa matiz, luminosidad y saturación, y para implementar el filtro se necesita el valor máximo y mínimo **HSL.** ***tbHmin*** y ***tbHmax*** sirven para especificar el valor de mínimo y máximo de matiz en un rango de 0 a 359 cada uno. ***tbLmin*** y ***tbLmax*** son para definir el valor de luminosidad que tienen un rango de 0 a 1000 en cada uno. ***tbSmin*** y ***tbSmax*** son para definir el valor de saturación con un rango de 0 a 1000 cada uno.

Se agregaron tres botones llamados ***btnCapturar***, ***btnDetener*** y ***btnMover***. El primer botón llamado ***btnCapturar*** se etiqueto con la leyenda "**Capturar**" y se cargó la siguiente programación en el evento **Click:**

```
Fuente = new VideoCaptureDevice(DispositivosDeVideo[cbCamara.SelectedIndex].MonikerString );
Fuente.NewFrame += new NewFrameEventHandler ( Frame );
Fuente.Start();
```

Se tiene que definir un objeto global llamado ***Fuente*** de la clase ***VideoCaptureDevice*** y se le crea una instancia por medio del constructor ***VideoCaptureDevice*** que lleva como parámetros la cámara seleccionada en ***cbCamara***. Luego se agrega en ***Fuente*** la rutina que se va ejecutar en el evento ***NewFameEventHandler*** cuyo método es ***Frame***. La rutina ***Frame*** se define a continuación:

```
private void Frame(object sender, NewFrameEventArgs eventArgs)
{
   Imagen = (Bitmap)eventArgs.Frame.Clone();
   video = (Bitmap)eventArgs.Frame.Clone();
   video2 = (Bitmap)eventArgs.Frame.Clone();
   filtro.Hue = new IntRange(tbHmin.Value, tbHmax.Value);
   filtro.Luminance = new Range((float)((float)tbLmin.Value / 1000.00),
(float)((float)tbLmax.Value / 1000.00));
```

```
   filtro.Saturation = new Range((float)((float)tbSmin.Value / 1000.00),
(float)((float)tbSmax.Value / 1000.00));
   filtro.ApplyInPlace(video2);
   grayImage = grayFilter.Apply(video2);
   blobcounter = new BlobCounter();
   blobcounter.MinHeight = 100;
   blobcounter.MinWidth = 100;
   blobcounter.ObjectsOrder = ObjectsOrder.Size;
   blobcounter.ProcessImage(grayImage);
   rects = blobcounter.GetObjectsRectangles();
   if (rects.Length > 0)
   {
      objectRect1 = rects[0];
      g = Graphics.FromImage(video);
      using (Pen pen = new Pen(Color.Red, 3))
      {
         g.DrawRectangle(pen, objectRect1);
         drawPoin = new PointF(objectRect1.X, objectRect1.Y);
         objectX = objectRect1.X + objectRect1.Width / 2 - video.Width / 2;
         objectY = video.Height / 2 - (objectRect1.Y + objectRect1.Height / 2);
         Blobinformation = "X= " + objectX.ToString() + "\nY= " + objectY.ToString()
+ "\nSize=" + objectRect1.Size.ToString();
         g.DrawString(Blobinformation, new Font("Arial", 16), new
SolidBrush(Color.Blue), drawPoin);
         if (serialok == true)
            {
               second = 0;
               offset = 300;
               second = offset - Math.Abs(objectX);
               map = (float)0.85 * second;
               buffer[0] = (byte)Math.Abs((int)map);
            }
         }
         g.Dispose();
      }
      pbCamara.Image = Imagen;
      pbFiltro.Image = grayImage;
      pbSeguimiento.Image = video;
      if (control)
      {
         ex_act = objectX;
         if (serialPort1.IsOpen)
         {
            if (ex_act < 0)
            {
              erx = "-" + Math.Abs(ex_act).ToString().PadLeft(3, '0');
            }
            else
            {
               erx = ex_act.ToString().PadLeft(4, '0');
            }
         }
      }
   }
```

Se deben de declarar tres objetos públicos de la clase En esta rutina se leen los valores indicados en los seis **TrackBar** antes mencionados y que se emplean en el filtro **HSV** mediante las siguientes instrucciones:

```
   filtro.Hue = new IntRange(tbHmin.Value, tbHmax.Value);
   filtro.Luminance = new Range((float)((float)tbLmin.Value / 1000.00),
(float)((float)tbLmax.Value / 1000.00));
   filtro.Saturation = new Range((float)((float)tbSmin.Value / 1000.00),
(float)((float)tbSmax.Value / 1000.00));
   filtro.ApplyInPlace(video2);
```

En las variables ***objectX*** y ***objectY*** se guardan la posición del centroide del objeto detectado. Estas dos variables que son del tipo **entero** guardas la posición donde ***objectX*** guarda la coordenada ***X***, y la variable ***objectY*** guarda la posición **Y**. En la Figura 6 se muestra como se distribuyen los ejes cuando esta es procesada por la computadora.

En la variable ***erx*** que es del tipo **int**¸ se almacena un error en pixeles. Este error dependiendo del signo se sabe si el objeto que se está siguiendo se encuentra a la derecha, si el error es positivo, y está a la izquierda si el error es negativo.

El botón ***btnDetener*** el cual se etiqueto con la leyenda **Detener** es para finalizar la captura de las imágenes y también su procesamiento, Se programó el evento **Click** de este botón con la instrucción ***FuenteDeVideo.Stop()*** el cual suspende la ejecución del evento ***NewFrame*** del objeto ***Fuente***.

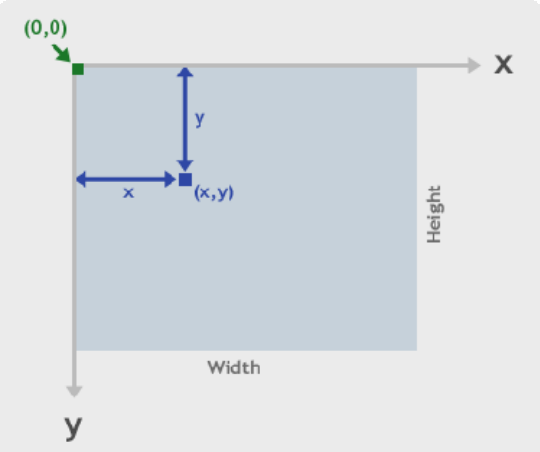

Figura 6.- Coordenas en pixeles de una imagen

El botón ***btnControl*** el cual se etiqueto con la leyenda **Control** sirve para mandar las señales de movimiento al arduino y este a su vez en al motor. Se programó el evento **Click** de este botón con la siguiente rutina.

```
control = control ^ true;
if (control)
{
   serialPort1.Open();
```

```
}
else
{
   serialPort1.Close();
}
```

Se tiene que declarar previamente una variable pública llamada ***control*** del tipo ***Bool***. Cuando se declara una variable tipo ***Bool*** el valor predefino es ***False***. En la primera línea se emplea la instrucción **^** que es el criterio del **Or Exclusivo** de tal manera que cuando **control** sea ***False*** al entrar en **Or Exclusivo** lo convertirá en **True** y viceversa. Cada vez que se presione el botón se estará conmutando el valor de ***control*** y este a su vez se evalúa en una instrucción ***if*** abriendo o cerrando el puerto serie para enviar datos al Arduino.

Resultados Experimentales.

Para los experimentos se empleó una esfera de color naranja (Ver Figura 7). Se escogió este color especialmente porque tiene un rango muy pequeño en el espectro de color (esto ocurre con todos los colores brillantes o fosforescentes) lo que facilita su detección para rangos de color HSV.

Figura 7.- Objeto empleado para las pruebas y el resultado del tratamiento de imágenes

En la Figura **8** se muestran los resultados del seguimiento de la esfera. La prueba se realizó por aproximadamente 4 minutos y se estuvo moviendo la esfera durante ese período. En el segundo 0 se aprecia que un error de 200 pixeles durante los siguientes 50 segundos el robot está corrigiendo su postura hasta alcanza un error próximo a 0 pixeles. Llegado a este punto manualmente se desplazó la esfera a la derecha que es donde se aprecia un nuevo pico de 200 pixeles ahora logrando corregir este error en aproximadamente 30 segundos. Se repite el movimiento de la esfera ahora hacia el lado contrario obteniendo resultados similares. Esto se realiza a lo largo de todo el experimento. Se presentó una anomalía en el cuarto pico en donde la cámara comenzó a ver de manera borrosa la esfera debido un alejamiento de esta. La cámara cuenta con un autoajuste para corregir estas situaciones lo cual genero el ruido que se aprecia entre los segundos 120 y 150.

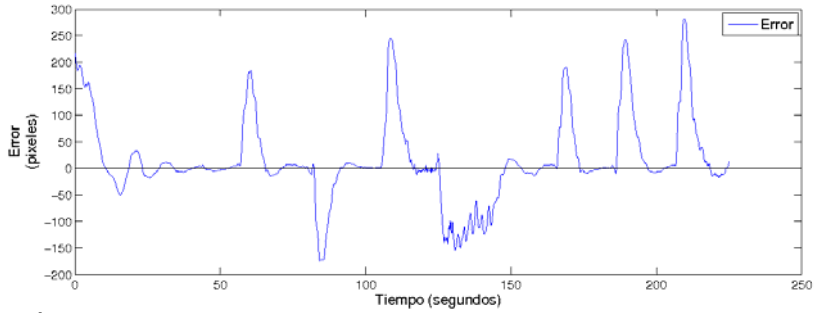

Figura 8.- Gráfica que muestra el desplazamiento del objeto con respecto al centro la imagen

Conclusiones:

Los sistemas que emplean realimentación como los utilizados en este trabajo, presentan una gran ventaja debido a su fácil implementación. Antiguamente se tenían que comprar sensores laser o sensores ultrasónicos, además de ser necesario una DAQ (por sus siglas en ingles Adquisitoria de Datos) que tuviera la capacidad de leer dichos sensores. En cambio ahora una cámara sencilla, que es posible encontrarlas en muchos lugares, y en las cuales ya vienen incluidos los drivers necesarios para que sea detecta por la computadora o bien descargarlos de red. Permite que las investigaciones que sea realicen no requieran de grandes inversiones en cuanto a equipo y facilita en gran medida el se puedan realizar proyectos académicos o personales.

Como se ve en el artículo la mayor parte de esta área tiene que ver con programación ya que al poderse digitalizar fácilmente las imágenes todos se reduce a procesos matemáticos y no se requieren de grandes estudios para poderlos aplicar gracias a las librerías como OpenCV o Aforge .NET que la que se implemento en este trabajo.

Muchos sistemas mecánicos, mecatronicos, electromecánicos o electrónicos, que se creían obsoletos pueden ser rehabilitados implementando visión, esto porque puede ser difícil ingresar nuevos sensores ya que hay que desarmar los mecanismos y acoplarlos. En cambio al utilizar una cámara es posible montarla fácilmente en el sistema o en su defecto colocarla en un lugar fuera del sistema observando el proceso que se planea controlar.

Bibliografía:

Eleazar, G., Sánchez, A., Alejandro, M., Huerta, L., Lozada, R. P., Humberto, J., y otros. (2012) FinDER Robot de búsqueda enambientes de desastre. *COMROB.* Puebla, Puebla, México.

García, M. A., Cárdenas, A., Rendón, J. M., & Méndez, M. M. (2009). Una Plataforma de Control Basado en Visión para la Rehabilitaciónde Robots Manipuladores de Tipo Industrial. *Computación y Sistemas*, 409 - 420.

Hernandez, E., & Ibarra, J. (2012). SLAM Visual Monocular con 1-Point RANSAN IDP y EKF aplicado a un robot humanoide NAO, 1-9. *COMROB.* Puebla, Puebla, México.

Hollinghurst, N., & Cipolla, R. (1993). Uncalibreated Stereo Hand-Eye Coordination. *British Machine Vision Conference*, (págs. 1 - 39).

Hu, S., Fans, Z., Tang, J., Xu, H., & Sun, Y. (2010). Research of Driver Eye Features Detection Algorithm Based on OpenCV. *Secong WRI Global Congress on Intelligent Systems*, (págs. 348-351).

Jaques, D. J., Rodriga, R., Mcisaac, K. A., & Samarabandu, J. (2005). An Object Tracking and Visual Servoing System for the Visually Impaired ∗. *IEEE International Conference on Robotics and Automation*, (págs. 3510-3515). Barcelona, España.

Manuel, J., Zannatha, I., David, Á., Sánchez, G., Enrique, J., Delgado, L., y otros. (2012). Comando gestual de un humanoide Nao usando vision 3D. *COMROB 2012.* Puebla, Puebla, México.

Miyajima, T., Thomas, D., & Amano, H. (2012). A Domain Specific Language and Toolchain for OpenCV Runtime Binary Acceleration Using GPU. *Third International Conference on Networking and Computing*, (págs. 175 - 181).

Pajares, G., & De la Cruz, J. (2008). *Visión por Computador. Imagenes digitales y aplicaciones.* Alfaomega.

Ye, W., Yuetian, S., Yunhe, X., Shu, W., & Yuchen, Z. (2010). The Implementation of Lane Detective Based on OpenCV. *Second WRI Global Congress on Intelligent Systems.*