SOC_1202_T5_120221

Pruebas de algoritmos básicos

Versión: 20120313

Autor: Víctor

Inicio	21/02/2012
Final	13/03/2012
Personas	Víctor Sánchez

Predictive
Control
Systems

Thinking Forward
www.tfxxi.com

VARIACIONES RESPECTO LA VERSIÓN ANTERIOR:

Versión anterior	Ninguna
Variación	1- Ninguna

1- Autor: Vacio. Descripción: Vacio.

1 Pruebas de algoritmos básicos

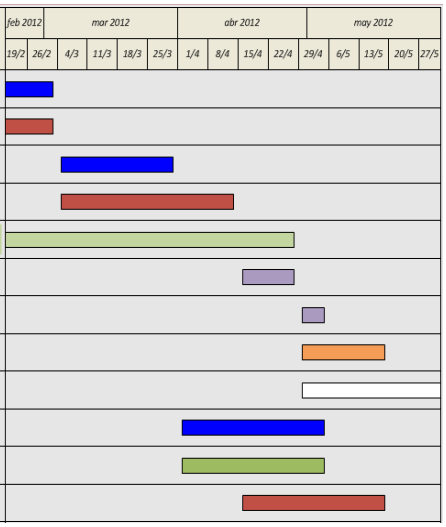
2 Objetivos

El objetivo principal de la tarea consiste en desarrollar un código en C++ (Windows Forms) capaz de detectar la holgura de detección por visión por computador en un motor Dimetronic.

El código debe ser desarrollado teniendo en cuenta que deberá ser implementado en una etapa posterior en un Micro Atmel o similar, por lo que no se deberían utilizar complejas librerías preprogramadas del estilo de OpenCV ya que no serían aplicables en el Microcontrolador.

3 Plan de desarrollo de la tarea

ID	Tarea	Descripción tarea	Responsable tarea	Inicio	Final	Duración
1	SOC_0212_T1_210212	Definir hardware cámara	Rafel	21/02/2012	02/03/2012	9d
2	SOC_0212_T2_210212	Definir el microcontrolador, familirizarse con el entorno de desarroyo, can, etc.	Fran	21/02/2012	02/03/2012	9d
3	SOC_0212_T3_210212	Diseñar prototipo	Rafel	05/03/2012	30/03/2012	20d
4	SOC_0212_T4_210212	Diseñar software prototipo	Fran/Ezio	05/03/2012	13/04/2012	30d
5	SOC_0212_T5_210212	Pruebas de algoritmos básicos	Víctor	21/02/2012	27/04/2012	49d
6	SOC_0212_T6_210212	Definición protocolo de comunicación Servidor/Cliente/SOC	Ezio/Victor/Fran	16/04/2012	27/04/2012	10d
7	SOC_0212_T7_210212	Implementación del protocolo de comunicación Servidor/Cliente/SOC	Ezio/Fran	30/04/2012	04/05/2012	5d
8	SOC_0212_T8_210212	Cambios en TFClient derivados de la aplicación de SOC	Victor/Fran	30/04/2012	18/05/2012	15d
9	SOC_0212_T9_210212	Diseño de algoritmos alternativos mejorados de visión.	Todos	30/04/2012	01/06/2012	25d
10	SOC_0212_T10_210212	Rediseño de hardware SOC	Rafel/Ezio	02/04/2012	04/05/2012	25d
11	SOC_0212_T11_210212	Diseño industrial SOC	Marc/Rafel/Oriol	02/04/2012	04/05/2012	25d
12	SOC_0212_T12_210212	Diseño del software de visión final	Fran/Ezio/Victor	16/04/2012	18/05/2012	25d



Dentro del proyecto SOC, esta tarea (Marcada en verde en el diagrama) ocupa un total de 9 semanas y media que van desde 21/02/2012 hasta 27/04/2012.

4 Desarrollo de la tarea

4.1 Subtareas

Para el desarrollo de esta tarea se han identificado 4 subtareas:

- Preparación de un entorno de desarrollo
- Localización de zonas de trabajo dentro de la imagen
- Localización de las rectas de interés
- Calculo de la distancia entre las rectas de interés

El de talle del desarrollo de cada una de las subtareas se lleva a cabo en los siguientes apartados.

4.1.1 Preparación de un entorno de desarrollo

Desde el primer momento se pensó que la mejor opción de desarrollo sería optar por un entorno basado en Windows Forms dadas las facilidades que ofrece esta tecnología si se compara con tecnologías similares como MFC.

Con la finalidad de preparar el entorno se creó un nuevo proyecto Windows Forms y se empezó a integrar las librerías de captura de imagen y proceso OpenCV. Como se comentó en la introducción, no se pretende hacer uso de funciones de procesado de imagen de estas librerías ya que complicaría luego la migración del software desarrollado a Atmel, pero serán de utilidad a la hora de la adquisición de imágenes y de pruebas rápidas.

La tarea se completó con éxito en un día de trabajo. Se complicó ligeramente debido a que OpenCV no está directamente preparado para trabajar en entornos managed como Windows Forms, por lo que tuve que buscar información sobre como recompilar OpenCV para que pudiera ser utilizado en Windows Forms.

4.1.2 Localización de zonas de trabajo dentro de la imagen

Debido a que el código será finalmente ejecutado por un microprocesador , no es viable procesar constantemente toda la imagen. Por esa razón se consideró que sería interesante detectar ciertas regiones de interés y procesar esas zonas en lugar de procesar toda la imagen constantemente.

En una primera versión se quiso calcular dinámicamente estas zonas basándose en un algoritmo sencillo el cual aplicaba una máscara a toda la imagen resaltando las zonas con bordes rectos y a partir de esta imagen procesada se centraba la zona de trabajo en aquella que se detectaban más líneas rectas.

Se muestra a modo de ejemplo un screenshot de la ejecución de esta versión:

La zona de trabajo seleccionada aparece resaltada sobre el martillo en la imagen de la derecha y las líneas y círculos indican valores calculados utilizados en la selección de la zona de trabajo.

El código desarrollado para esta versión se encuentra en:

<Tarea> \Diseño\Software\C++\SOC_1202_T5_120221_V1.rar

Tras realizar diversas pruebas basadas en esta versión del cálculo de la zona de trabajo se observó que era ligeramente susceptible a cambios de iluminación así como a cambios en la cantidad de grasa en las barras que generaban problemas al calcular la zona.

Tras numerosas pruebas se observo que no tenía mucho sentido calcular de forma dinámica las zonas de trabajo ya que debido a la geometría del motor y a la posición fija de la cámara estas zonas de trabajo nunca variaría más de ciertos límites que podían acotarse por lo tanto, finalmente se ha decidido fijar las 4 zonas de trabajo en base a proporciones de la imagen. Básicamente los cálculos para identificar las 4 zonas son los siguientes:

```
//inputImage

inputImage = photoCaptor->getFrame(false);

//calculamos las ventanas que utilizaremos. En el pic seran variables fijas ajustadas al tipo de camara seleccionada
int centroy = (inputImage->height/2); //centro de la pantalla en y
int centrox = (inputImage->width/2); // centro de la pantalla en x
int centroVentanaSup = centroy-(centroy/3); // centro-1/3 de la pantalla en y
int centroVentanaInf = centroy+(centroy/3); //centro+1/3 de la pantalla en y
int limiteXVentanas = centrox + (centrox/3); //centro+1/3 de la pantalla en x

//definimos el margen de un 5% del tamaño del alto
int margen = inputImage->height*0.05;
int anchoVentanas = (inputImage->height*0.05)/2;

//Definimos finalmente las 4 ventanas iniciales de proceso
Ventana^ arribaIzq = gcnew Ventana(margen,centroVentanaSup-anchoVentanas,(limiteXVentanas/2)-margen,centroVentanaSup+anchoVentanas);
Ventana^ arribaDch = gcnew Ventana((limiteXVentanas/2)+margen,centroVentanaSup-anchoVentanas,limiteXVentanas,centroVentanaSup+anchoVentanas);
Ventana^ abajoIzq = gcnew Ventana(margen,centroVentanaInf-anchoVentanas,(limiteXVentanas/2)-margen,centroVentanaInf+anchoVentanas);
Ventana^ abajoDch = gcnew Ventana((limiteXVentanas/2)+margen,centroVentanaInf-anchoVentanas,limiteXVentanas,centroVentanaInf+anchoVentanas);
```

Con estos sencillos cálculos se generan las 4 zonas de trabajo que son las que procesaremos en busca de líneas de interés. A la hora de migrar este código al Microcontrolador, las zonas podrán ser configuradas de forma fija ya que las propiedades de la cámara serán conocidas y nunca variaran. Las 4 zonas de trabajo seleccionadas se muestran en la siguiente figura:

El código desarrollado para esta versión se encuentra en:

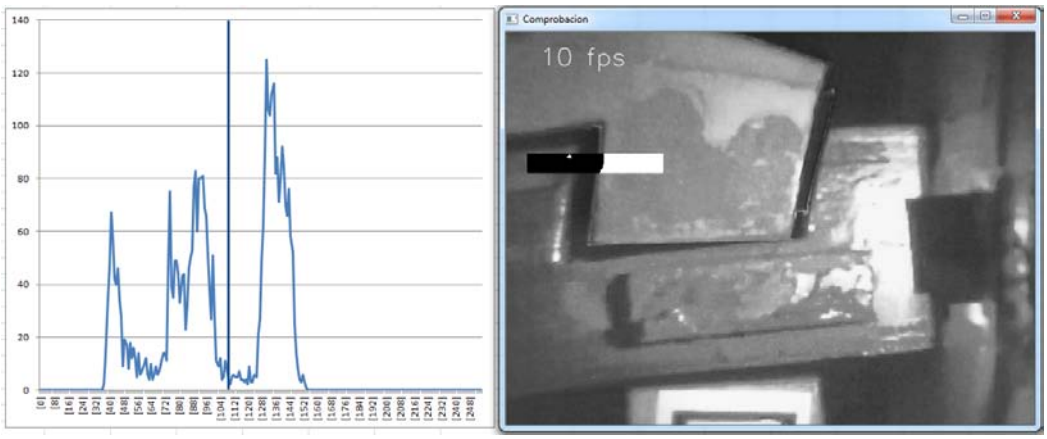
<Tarea> \Diseño\Software\C++\SOC_1202_T5_120221_V1.1.rar

4.1.3 Localización de las rectas de interés

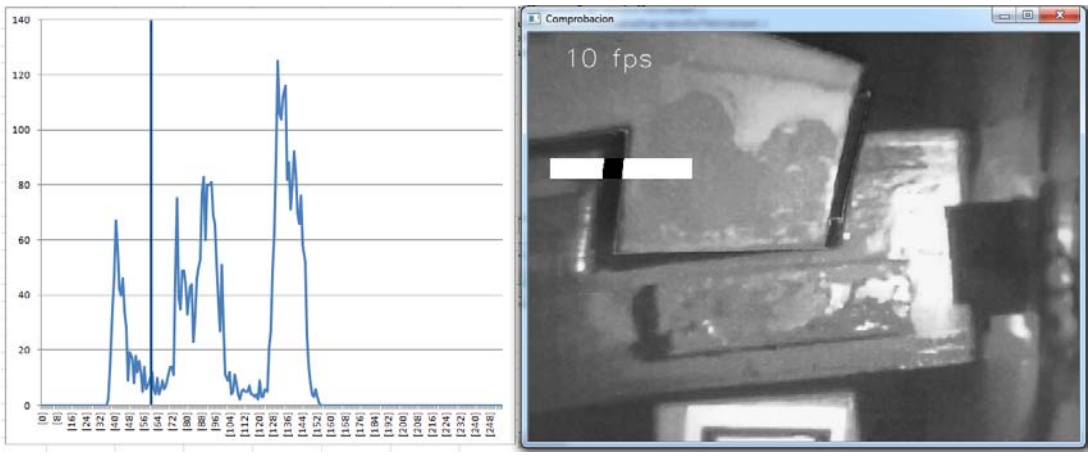
Esta tarea es la más compleja de todo el proyecto y la que requerirá más esfuerzo.

Después de analizar diferentes alternativas, se observa que independientemente del camino a seguir el cálculo del histograma de la imagen así como de algún tipo de filtro será necesario. Se desarrollan por tanto dos funciones de filtro uno gaussiano y otro de mediana. Al implementar la función de histograma se observa que hay algún problema porque el histograma generado es una translación en X del histograma real calculado con matlab. Tras analizar el problema durante algunas horas y probar infinidad de cosas se observa que el problema es debido a que la cámara tarda unos segundos en estabilizar la imagen por lo que los primeros histogramas eran diferentes a los que se obtienen cuando la imagen está estabilizada.

Tras analizar el histograma se observa que es necesario el cálculo dinámico del threshold para binarizar con un threshold diferente cada ventana de trabajo. Despues de estudiar las diferentes alternativas existentes parece que el algoritmo de Otsu es una solución eficiente al problema y se implementa una función que calcula el threshold de Otsu, tras analizar los resultados observamos que no funciona como se esperaba:



En el histograma de la izquierda se muestra con una línea vertical el threshold calculado por el algoritmo de Otsu y en la derecha el resultado de binarizar la ventana de trabajo con este threshold, mientras que el threshold óptimo que necesitaríamos para nuestra aplicación se muestra en la siguiente imagen:



Por lo tanto el siguiente paso es desarrollar un algoritmo que dinámicamente pueda calcular un threshold que se aproxime más a la segunda imagen.

Tras conseguir un algoritmo que generaba un threshold como el deseado basado en el análisis del histograma, se observo que incluso así la variación de luz y de pequeños objetos o manchas de grasas afectaban drásticamente al resultado. Por lo que finalmente se desestimo la idea de usar algoritmos que calculasen dinámicamente el threshold de binarización ya que esto generaba más problemas de los que corregía. Al final se ha optado por una versión basada en un threshold fijo en el que los negros son los pixels que tienen un nivel por debajo de 65. Además para esta aproximación se ha trabajado mucho con la iluminación. El objetivo es conseguir una imagen en la que las zonas del martillo y barras están lo más saturadas posibles mientras que el fondo del motor es muy negro facilitando de este modo la tarea de segmentación de la holgura, para esto fue necesario configurar la cámara de forma manual para poder saturar la imagen sin que se corrigiera automáticamente por lo que esto será igualmente necesario cuando dispongamos de la cámara final.

Finamente se obtuvo una imagen que cumplía las condiciones descritas, pero no era suficiente ya que en las ventanas de procesado seguía apareciendo ruido no deseado debido a las sobras y pequeñas motas de grasa. Esto hizo necesario añadir simples controles para limpiar este ruido basándonos en el conocimiento de la naturaleza de las imágenes que esperamos encontrar.

Con la imagen perfectamente segmentada, el cálculo de las líneas de interés resultó directo.

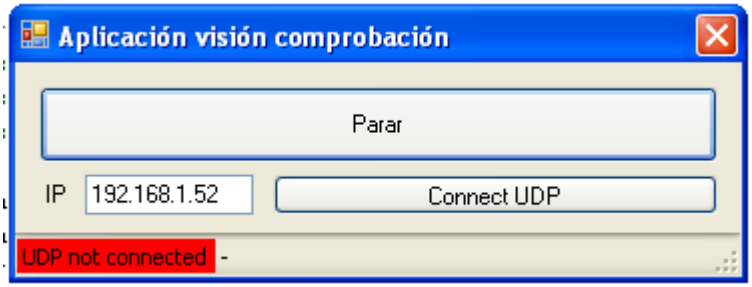
4.1.4 Cálculo de la distancia entre las rectas de interes

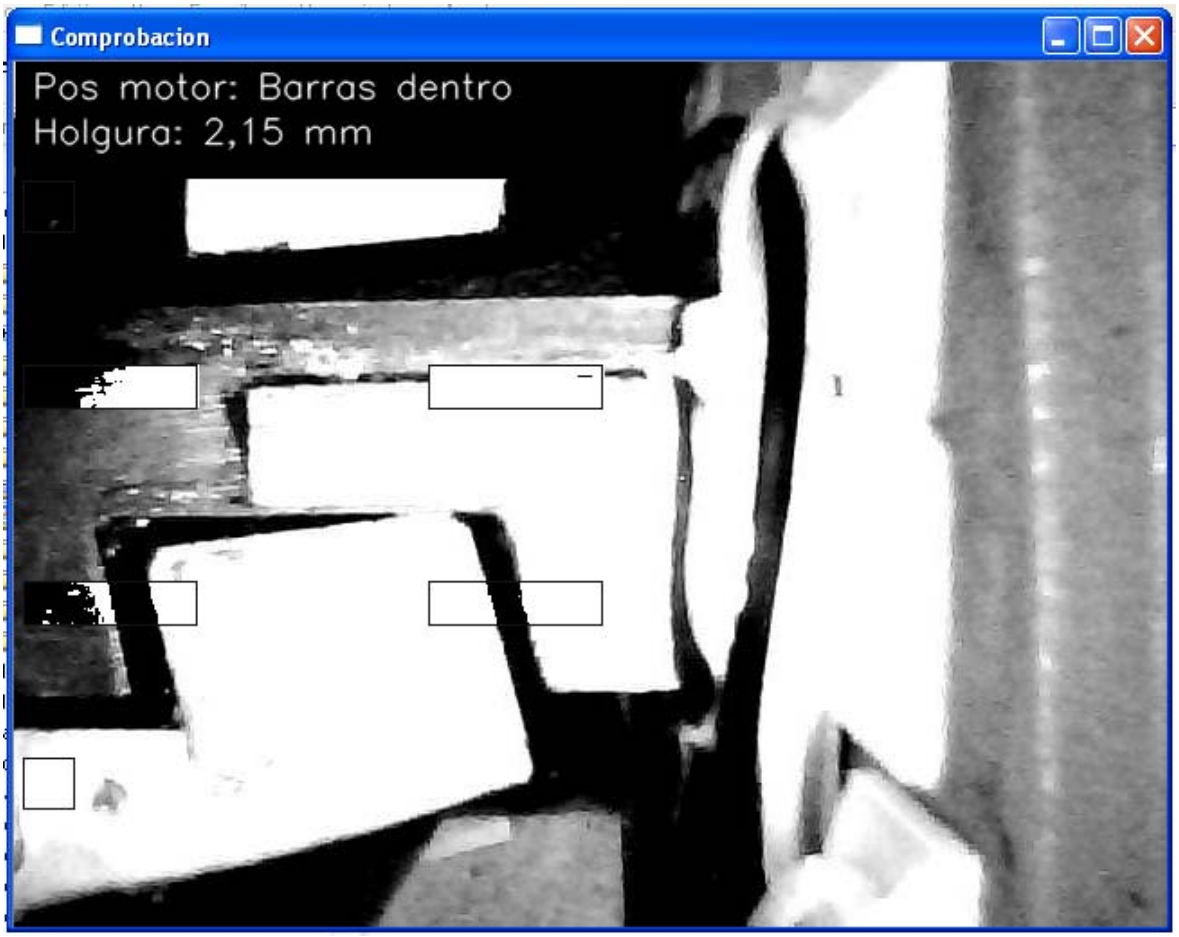
Teniendo las rectas de interés calculadas , también fue una tarea sencilla el calcular las distancias entre las mismas y de ellas la holgura de comprobación, obteniendo precisiones, con el sistema correctamente configurado, de hasta 0.1mm

Llegados a este punto únicamente restaba encontrar la forma de detectar si las barras estaban fuera o dentro para focalizar el cálculo de la holgura en las ventanas superiores o inferiores. Después de probar diferentes alternativas la que resulto más sencilla y funcional fue añadir dos ventanas de procesado más situadas estratégicamente de forma que cuando se cierra un martillo u otro tapa una ventana u otra por lo que es directo saber que martillo está cerrado y, por tanto, donde están las barras.

En caso de que se desee un mayor detalle de los algoritmos desarrollados, todo el código está comentado y se encuentra en <Tarea>\Diseño\Software\C++\SOC_1202_T5_120221_V2.rar

La interface final desarrollada para esta tarea se muestra en la siguiente figura:





Además se ha desarrollado un servidor al que desde este software se puede enviar las imágenes capturadas de forma que pueden enviarse al micro Atmel y de esta forma empezar a hacer desarrollos en el Atmel utilizando las imágenes de la webcam.

5 Conclusiones

Se observa que es mucho más ventajosa la utilización de ventanas de tamaño y posición fijas en vez de ventanas de tamaño y posición dinámica. La cámara será siempre la misma y siempre estará en la misma posición por lo que no tiene sentido el uso de complejos algoritmos de posicionamiento de las ventanas que seguramente podrían introducir factores de error.

Por otro lado también se ha visto que, dado que el entorno así como su iluminación son totalmente controlables, es mucho más ventajoso trabajar con la iluminación de forma que los algoritmos de segmentación sean mucho más simples en lugar de desarrollar complejos algoritmos de visión que podrían no funcionar con la agilidad que necesitamos en el micro Atmel.

Finalmente añadir que la tarea se ha desarrollado en mucho menos tiempo del tiempo planeado ya que se le ha podido dedicar más horas al día de las que se esperaba y a que ha sido necesario un mayor esfuerzo para poder ajustar el Project plan debido a la marcha de Fran.

Fecha de finalización de la tarea 13/03/2012