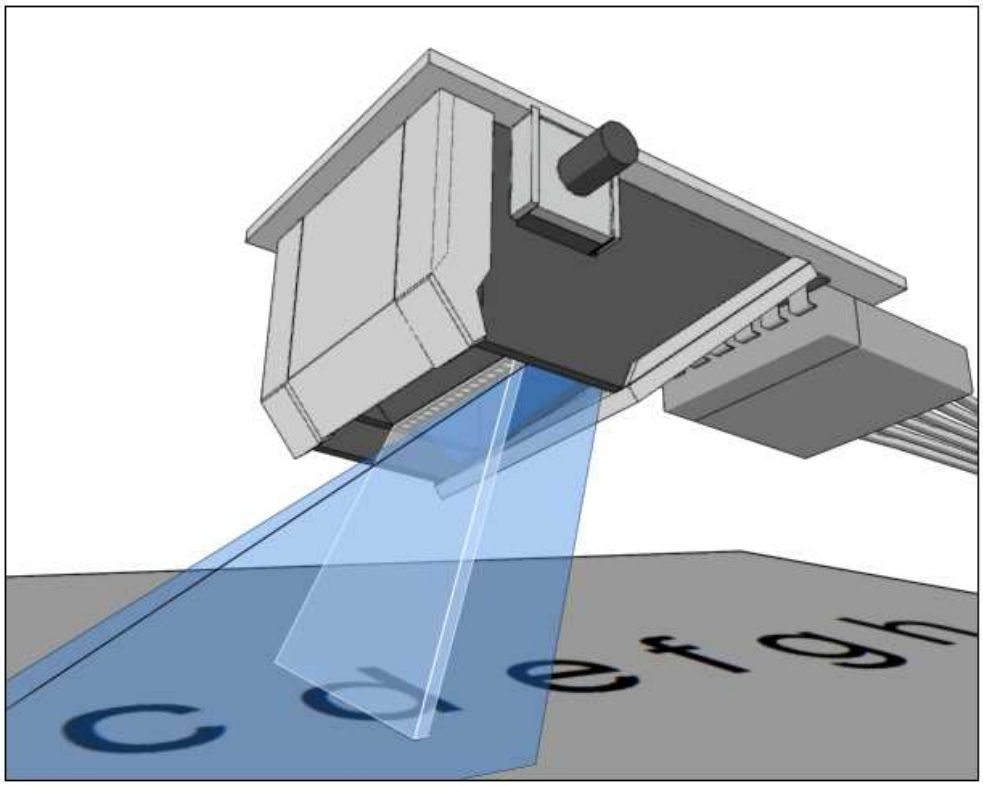
Pontificia Universidad Católica Argentina
Facultad de Ciencias Fisicomatemáticas e Ingeniería

Aplicación de reconocimiento óptico de caracteres con sensor lineal, Versión 1

Trabajo Final



Alumno *Ignacio Ghersi*

Carrera *Ingeniería en Electrónica y Comunicaciones*

Profesor *Ing. Norberto Heyaca*

Tutor *Ing. Ricardo Vecchio*

Año *2009*

TABLA DE CONTENIDOS

1. INTRODUCCIÓN

1.1 RESUMEN

El siguiente proyecto consiste en el desarrollo de una aplicación de reconocimiento óptico de caracteres impresos como resultado de la operación conjunta de un programa en un microcontrolador y otro en una computadora. Se explicará a lo largo del siguiente texto el proceso que se siguió a lo largo del desarrollo de esta aplicación, así como las dificultades encontradas durante ese proceso, las soluciones que se tomaron para resolverlas, y los compromisos que se hicieron para lograr un sistema tan robusto y eficiente como fuera posible.

Más adelante, se explicará la lógica que rige los programas desarrollados para el microcontrolador y la computadora, y la forma en que ambos equipos operan cooperando para aprovechar las facilidades que ofrece cada uno de forma más eficiente.

La última parte de este trabajo trata los resultados obtenidos de la aplicación funcionando sobre un sistema prototipo, las conclusiones cierre del trabajo y las posibles mejoras a incorporar al sistema.

1.2 OBJETIVOS

El objetivo de este proyecto es diseñar una aplicación que permita ejecutar reconocimiento óptico de caracteres (OCR) sobre caracteres impresos, mediante la utilización de un sensor óptico lineal, así como construir un prototipo de lector que permita probar la aplicación y mejorarla.

Se busca que la aplicación sea robusta y eficiente, y que logre una tasa baja de errores en la detección de caracteres sobre las distintas fuentes para las que se la entrene. Cabe destacar que, partiendo del hecho de que el texto a explorar puede estar degradado, esperar que el sistema sea infalible sería absurdo. Será necesario tener en cuenta las limitaciones del sistema a desarrollar para contenerlas y tratar de reducir su efecto en la eficacia final del sistema.

Básicamente, el sistema consiste en un lector que un usuario desplace sobre una línea de texto que quiera traducir. El sistema guardará información acerca de esa línea, la procesará, ejecutará OCR sobre esta información y guardará los resultados en un destino accesible para el usuario. Con esta idea en mente, surgen requisitos que se deberán alcanzar para que el sistema funcione de forma aceptable:

- *Calidad de imagen:* El sistema de lectura tendrá que realizar mediciones de buena calidad para no perjudicar la eficacia del reconocimiento. Se tratará de superar cualquier deficiencia del lector con correcciones de software.
- *Velocidad:* Se busca que el sistema sea operado por un usuario, de modo que no es aceptable ni atractivo un sistema que obligue al usuario a desplazar el lector lentamente sobre la línea a explorar
- C*ontrol de desplazamiento:* No es necesario si se asume que la velocidad de desplazamiento del lector es relativamente constante, pero medir el desplazamiento del lector permite asegurar lecturas igualmente espaciadas, algo valioso porque garantiza la uniformidad de la información

- *Segmentación de caracteres:* Se deberá desarrollar un algoritmo que detecte cuándo se termina de recibir un símbolo y que entregue información que permita encontrar ese símbolo en la información recibida
- *Memoria:* Como se está tratando con imágenes de texto (aunque se trabajara en blanco y negro), se asume que la cantidad de memoria a utilizar para no perder información valiosa será alta. Será necesario encontrar un mecanismo para guardar esta información valiosa de forma rápida
- *Utilidad de reconocimiento óptico de caracteres:* Será necesario incorporar en la aplicación una utilidad que permita decidir cuál es el símbolo que se leyó a partir de la información recopilada sobre éste

Adicionalmente, se analizarán las posibles evoluciones para el desarrollo propuesto, así como sus usos potenciales.

1.3 ESQUEMA INICIAL DE FUNCIONAMIENTO DE SISTEMA

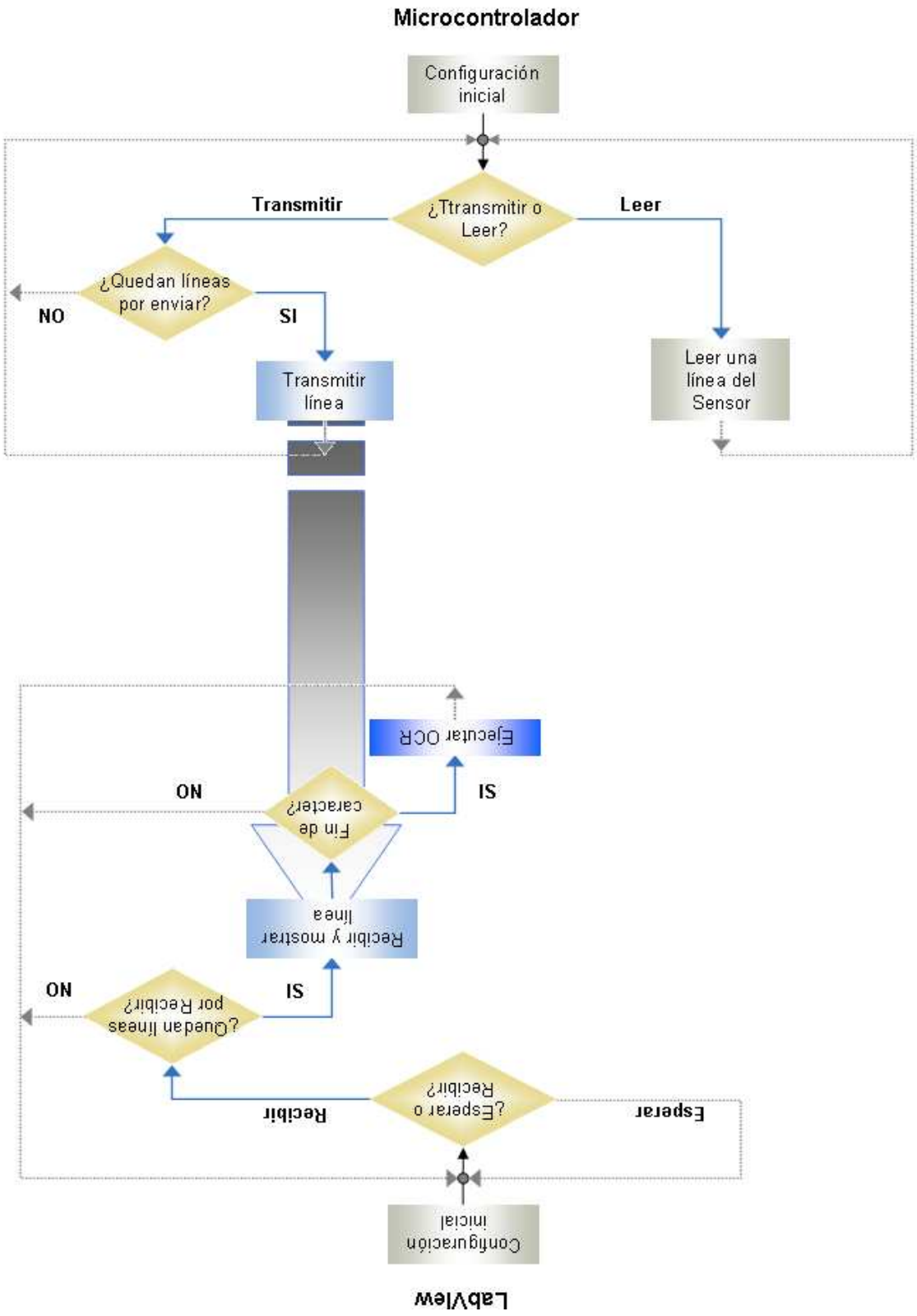
La idea básica que dio origen al proyecto era la de construir un lector que se pudiera utilizar a modo de un “lápiz” que el usuario deslice sobre el texto impreso que quiera reconocer. Este requisito definió la estructura física que se buscó para el prototipo, y justifica la elección de un sensor lineal (de una dimensión) para una aplicación de reconocimiento de imágenes (de dos dimensiones). Por lecturas sucesivas del sensor lineal se formará la imagen del texto.

Desde el inicio del proyecto se contemplaron varias alternativas en cuanto a la forma de implementarlo, en particular en cuanto a cuál sería el sistema utilizado para hacer el reconocimiento de patrones propiamente dicho. Desde un principio se buscaba que existiera una comunicación entre el microcontrolador y una computadora, para mostrar las imágenes que se recibían del sensor, y poder configurarlo correctamente. El programa utilizado para este propósito fue el LabView.

Más adelante, se tomó conocimiento de que era posible implementar una expansión sobre ese programa (Vision Development Module) que permite implementar funciones de visión y reconocimiento de patrones, con capacidad para reconocimiento óptico de caracteres. Sería posible entonces incorporar estas funcionalidades como bloques de programa en aplicaciones creadas para LabView. En este punto se entendió que sería favorable utilizar este módulo para el reconocimiento, en lugar de dejar todas las tareas en manos del microcontrolador, ya que el rendimiento y la velocidad del módulo funcionando en la computadora tendrían menos limitaciones que una función de reconocimiento diseñada para operar sobre un microcontrolador o un DSP.

Por otro lado, la naturaleza del sistema (controlado por la mano de un usuario) suponía que la uniformidad en las mediciones sería difícil de lograr, y como el lector sería construido para este proyecto, se entendió que podrían presentarse otras dificultades de nivel físico/óptico que podrían degradar la calidad de imagen. Es por esto que se creyó que lo mejor sería implementar una función de reconocimiento óptico de mayores capacidades.

En el siguiente diagrama se muestra un esquema *inicial* para ilustrar el funcionamiento básico que se busca para el sistema. En grandes rasgos, se sabe en este punto que el microcontrolador deberá tomar imágenes desde el sensor, y que en la computadora se ejecutará el reconocimiento de caracteres sobre esas imágenes. Más adelante se tratará cada aspecto del desarrollo con mayor detalle, y se mostrará la lógica final que se implementó en el proyecto (con mayor cooperación y un aporte más importante de parte del microcontrolador).



Esquema de funcionamiento de sistema en inicio de etapa de diseño

1.4 *INTRODUCCIÓN AL RECONOCIMIENTO OPTICO DE CARACTERES*

Se entiende por reconocimiento óptico de caracteres (OCR) al proceso por el cual se convierte una imagen de texto impreso o manuscrito a un formato digital, comprensible para una máquina (normalmente en un formato editable con un procesador de texto). Se trata entonces de traducir una distribución de pixels a un número que la representa, indicando el símbolo al que corresponde esa distribución (en formato ASCII, por ejemplo).

Un reconocimiento de imágenes por medios ópticos supone que se tomará una imagen del texto que se quiere traducir *una vez que fue escrito o impreso* (se denomina a esto reconocimiento off-line). Existen otras aplicaciones de reconocimiento de caracteres (no óptico) que analizan *texto a medida que se lo escribe* (denominado reconocimiento on-line), y se observan generalmente en computadoras de mano, consolas de videojuegos, teléfonos, etc., que disponen de pantallas táctiles. En estas aplicaciones se vuelve un poco más sencillo el reconocimiento de caracteres manuscritos, ya que se tiene información acerca del sentido de los trazos y su dirección, además del aspecto visible del símbolo que escribe el usuario.

El reconocimiento óptico de caracteres impresos es un caso especial de reconocimiento de patrones, con características únicas vinculadas con el tipo de información que maneja (existe una gran variante de caracteres posibles, pero están distribuidos de forma ordenada). Las etapas básicas de un sistema que ejecuta OCR se muestran a continuación:

Etapas de un sistema de OCR

El **entrenamiento** de caracteres es necesario para indicar a la aplicación que ejecuta OCR qué caracteres interesa reconocer, y qué parámetros buscar para poder identificar a cada símbolo deseado. Esta información es utilizada más adelante para leer y procesar el texto que se quiera convertir. Una vez terminada la etapa de entrenamiento, se pueden leer imágenes, y analizar si se encuentran patrones en las imágenes de entrada que asemejen a los caracteres entrenados.

Una aplicación de OCR deberá **obtener la imagen** que se quiera analizar, y **procesarla** con el fin de darle un formato uniforme y corregir defectos en ésta. Luego es tiempo de separar la imagen cargada en unidades de texto, sean palabras o caracteres, dependiendo de la lógica de la aplicación. Este proceso se denomina **segmentación**.

Para cada unidad de texto registrada, **se extraen parámetros** que la representan, y **se determina el texto** que mejor representa la imagen analizada.

En la actualidad, el desarrollo de algoritmos lo suficientemente complejos para el reconocimiento de caracteres impresos del alfabeto latino ha permitido alcanzar niveles de efectividad cercanos al 100%. El reconocimiento de patrones más complejos, como el caso del reconocimiento off-line de caracteres manuscritos en cursiva, sigue siendo investigado con el objeto de mejorar los resultados.

Aplicaciones generales

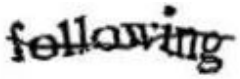
El reconocimiento óptico de caracteres puede utilizarse como utilidad en ramas de aplicación diversas:

Puede hacerse uso de un sistema de reconocimiento óptico de caracteres en aplicaciones industriales que involucran sistemas de visión robótica, mayormente para funciones de control, como por ejemplo:

- Verificación de códigos en encapsulados de circuitos integrados en la industria de semiconductores
- Inspección de etiquetas de envases y códigos de lote en industria farmacéutica
- Control de calidad de sellos sobre partes de máquinas
- Lectura de caracteres alfanuméricos en partes de automóviles
- Ordenamiento y seguimiento de paquetes de correo

Entre otras aplicaciones posibles, existe la de utilizar un programa de OCR para hacer lectura de tarjetas por medios ópticos, o también implementarlo en un sistema de reconocimiento y ordenamiento de documentación con formato definido.

Una aplicación indirecta de la lógica de OCR se encuentra en métodos que tratan de impedir que el reconocimiento pueda lograrse. Normalmente puede observarse al navegar por sitios seguros de Internet, que solicitan al usuario repetir un texto que se ve alterado y distorsionado, como el siguiente:



Test de Turing para seguridad contra ataques

Esta versión de un test de Turing (donde un juez intenta identificar si está tratando con un humano o una máquina, para determinar la capacidad de la segunda), es denominada CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) y puede observarse en sistemas protegidos contra accesos automáticos de computadoras. Para crear las imágenes de este sistema, es necesario conocer la lógica de reconocimiento que puede implementar una computadora sobre ésta. Una forma sencilla de evitar que un sistema pueda hacer OCR consiste, por ejemplo, en impedir que pueda segmentar la imagen en caracteres, como en la imagen superior.

En cuanto a variantes de aplicación orientadas al uso personal, existen sistemas que se destacan por reconocer caracteres de tipo especial o por sumar al reconocimiento otras funcionalidades:

Los caracteres del sistema Braille pueden ser reconocidos mediante un lector óptico, siempre y cuando éste sea capaz de detectar variaciones en el relieve de una superficie (con un sistema de iluminación apropiado, se puede lograr esto fácilmente).

Un sistema con estas características podría servir como utilidad de entrenamiento para una persona no vidente, por ejemplo.

Con una interfaz apropiada, un sistema de reconocimiento de texto impreso sumado a un conversor de texto a voz también puede ser útil para personas no videntes, permitiéndoles tener acceso a fuentes de texto en formatos distintos al Braille.

El conjunto de notas musicales. como son escritas en una partitura, puede ser entendido como un alfabeto. Un sistema de reconocimiento de notas musicales permite convertirlos a un formato edtable, o bien convertirlos a la música que representan.

Otra utilidad posible para el reconocimiento óptico de caracteres consiste en el reconocimiento automático de texto en señales viales y patentes. En estos casos la posición relativa de los objetivos respecto del lector es más variable, y su orientación espacial relativa también puede cambiar mucho, por cuestiones de perspectiva. Sin embargo, existen investigaciones en curso y proyectos en funcionamiento que hacen uso de cámaras para aplicaciones de este tipo.

Por último, existen aplicaciones para consumo con forma de lectores portátiles que permiten a un usuario convertir y guardar texto sin necesidad de transcribirlo. Aplicaciones más desarrolladas de este tipo cumplen funciones de traducción y corrección de errores con diccionarios incorporados en la lógica de reconocimiento.

2 *DESARROLLO*

2.1 MICROCONTROLADOR PIC18F452

El PIC18F452 es un microcontrolador de alto rendimiento, capaz de operar a 10 millones de instrucciones por segundo (10MIPS) cuando su configuración de oscilador lo permite (con 40MHz de frecuencia interna de reloj). Posee 32KBytes de memoria FLASH de programa, equivalente a 16384 instrucciones simples (cada instrucción simple ocupa 2 bytes), y dispone de 1536 bytes de memoria RAM. También se dispone de una memoria EEPROM de datos de 256 bytes.

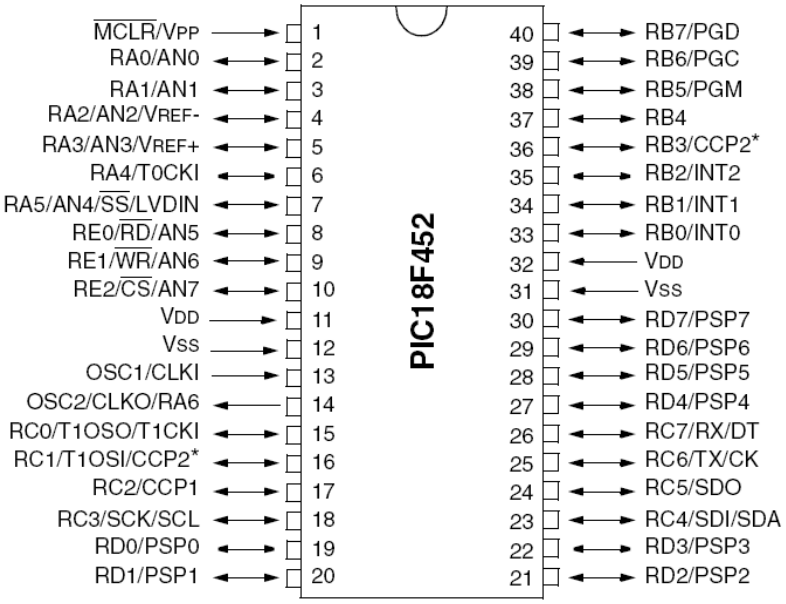


Diagrama de pines, PIC18F452
Fuente: Hoja de datos PIC18F452, Microchip

Además de disponer de puertos de propósito general, el microcontrolador posee varios módulos periféricos programables:

- 4 Timers con características y funcionalidades diferentes
- Módulo conversor A/D con 10 bits de resolución
- 2 módulos Capture/Compare/PWM
- Módulos serie
  MSSP – Master Synchronous Serial Port
  USART – Universal Synchronous Asynchronous Receiver Transmitter
- Módulo PSP (Parallel Slave Port)

Algunas características especiales de este microcontrolador son:

- Posibilidad de seleccionar un multiplicador de frecuencia de oscilador externo, 4x Phase Lock Loop (PLL)
- Disponibilidad In-Circuit Serial Programming (ICSP)
- Modo Sleep para ahorro de energía

Se eligió el PIC18F452 para esta aplicación por su velocidad de operación (se utilizará la máxima disponible en este caso, 10MIPS), disponibilidad de múltiples periféricos para un amplio rango de propósitos (que permitiría hacer pruebas, modificaciones y ampliaciones futuras), y por el tamaño de su memoria de programa y RAM (desde un principio se estimaba que el programa final sería exigente en este aspecto, especialmente en memoria RAM). Por otro lado, la disponibilidad de programar este microcontrolador en lenguaje de alto nivel (con ayuda del compilador C18 del MPLAB) es valiosa –aunque no sea este el único microcontrolador que puede ser programado en este lenguaje por este método o alguno similar.

Módulo USART para interfaz con computadora

La USART del PIC18F452 involucra los pines RC6/TX y RC7/RX del microcontrolador, y al activarlo es necesario configura a RC6 como interfaz de salida y a RC7 como interfaz de entrada, de acuerdo con la especificación del microcontrolador.

El módulo puede configurarse **para realizar comunicaciones full-duplex asíncronas** (para comunicación con una PC por ejemplo, como se utiliza en esta aplicación), o en un modo half-duplex síncronizado (master o slave) para comunicarse con otros periféricos con interfaz serie.

El módulo USART operando en modo asíncrono trabaja con señales del formato NRZ: envia un bit de Start y un bit de Stop antes y después de cada 8 bits de datos (o 9 según configuración). Se puede configurar el sistema para que se controle la comunicación con interrupciones en la transmisión y en la recepción, o evitarlas. Un flag de interrupción se activa aunque no se habiliten las interrupciones, de modo que el control sin interrupciones de la comunicación es posible. Es posible configurar el módulo para recepción continua de datos, y se dispone de señales para el control de errores de flujo. El noveno bit puede ser utilizado por software para control de paridad, pero no se lo utiliza en esta aplicación.

Generador de Baud Rate de módulo USART

Para la operación en modo asíncrono, el registro SPBRG, junto con el bit BRGH, determinan el baud rate para la comunicación serie, de acuerdo con las fórmulas expresadas a continuación:

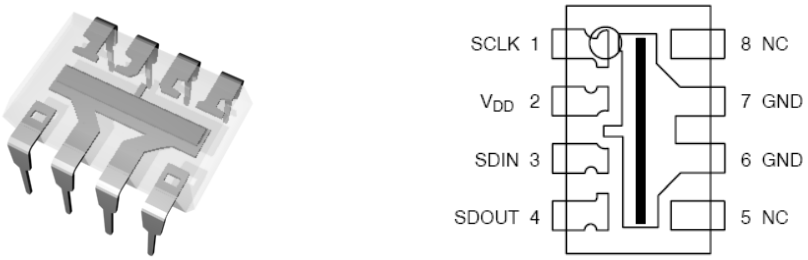
BRGH = 0 *(Baja velocidad)* **Baud Rate = F_{osc} / (64 (SPBRG + 1))**
BRGH = 1 *(Alta velocidad)* **Baud Rate = F_{osc} / (16 (SPBRG + 1))**

En esta aplicación se tratará de alcanzar la mayor velocidad posible para la comunicación entre el microcontrolador y la computadora, por motivos que se harán evidentes más adelante. Se utilizará en esta interfaz un cable blindado y relativamente corto, para que las características de la capa física no limiten la velocidad máxima de la transmisión.

2.2 *SENSOR TSL3301-LF*

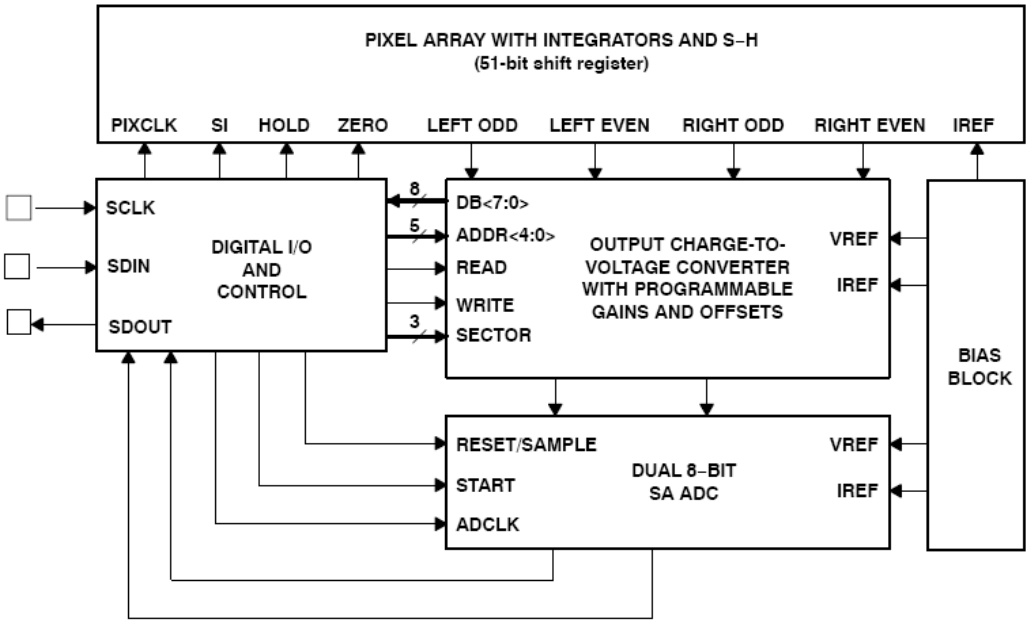
2.2.1 Características

El sensor elegido para medir intensidad de luz es el TSL3301-LF de la empresa Taos Inc. Este sensor es una matriz de 102x1 fotodiodos de alta sensibilidad y tiene una resolución de 300dpi. No sólo posee una interfaz serie que permite programarlo y controlar su funcionamiento con un microcontrolador, sino que también tiene incorporado un conversor A/D interno de 8 bits y acepta tensiones de alimentación entre 3V y 5.5V. Este integrado acepta un reloj de hasta 10MHz.



TSL3301-LF DIP

Al momento de la realización de este informe, este sensor es el único en su serie que posee un conversor A/D interno y, en consecuencia, salida digital para las mediciones de intensidad de luz a través de una interfaz serie de alta velocidad. Esta característica resulta ventajosa a la hora de implementarlo en esta aplicación, principalmente en términos de complejidad de código del microcontrolador, pero también en términos de velocidad de operación del sistema (se busca que el tiempo que se tarde en ejecutar una lectura y procesarla sea reducido, ya que esto se traduce en que el usuario pueda desplazar el sensor más rápidamente sobre el texto).



TSL3301-LF Diagrama en bloques
Fuente: Hoja de datos provista por el fabricante

Como consecuencia de la funcionalidad superior de este sensor, su costo es ligeramente superior respecto de otros de su serie que poseen características similares: el precio al por menor del TSL3301-LF ronda los U$S10, mientras que el sensor TSL1401R-LF, que posee salida analógica con una resolución de 400dpi y 128 pixels (en el mismo encapsulado) tiene un costo que ronda los U$S8. Si bien esta comparación parece dejar al integrado seleccionado en una posición de inferioridad, cabe destacar que:

- salvo para aplicaciones que utilicen un microcontrolador con conversor A/D incorporado, se tendría que sumar el costo de un conversor externo al del sensor que no lo incluye
- para esta aplicación, una resolución vertical de 300dpi resulta más que suficiente para poder observar caracteres en una amplia gama de tamaños
- una matriz con un gran número de pixels y alta resolución puede parecer más atractiva, pero supone tener que transmitir, procesar y guardar una mayor cantidad de información, y por tanto, para esta aplicación, resulta indeseable
- la salida digital permite evitar problemas de ruido y diferencias en las tensiones de referencia del conversor A/D

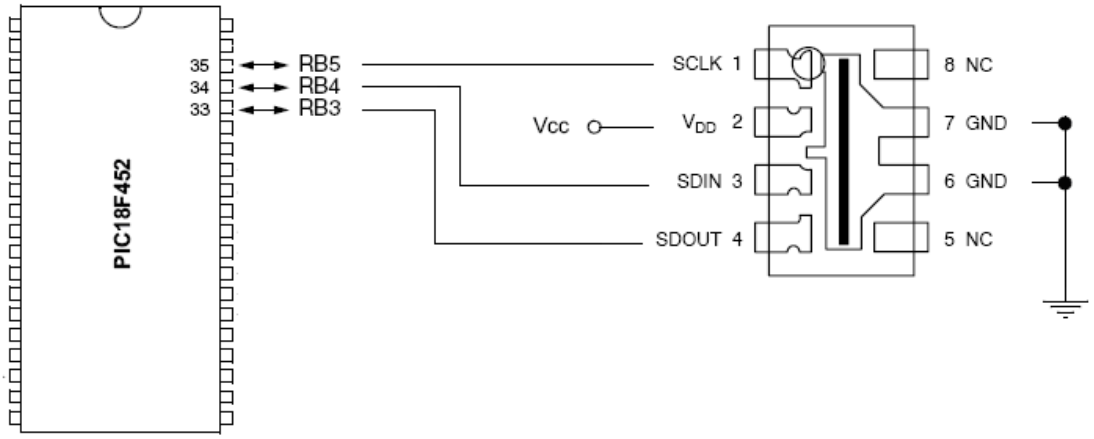
Es en vistas de esto que se considera que el TSL3301-LF ofrece una solución eficiente y de alto rendimiento, sin transmitir más información de la necesaria y ofreciendo la posibilidad de conexión directa al microcontrolador.

2.2.2 Conexión con el microcontrolador

Si bien el TSL3301-LF acepta velocidades de clock de hasta 10MHz, fue necesario considerar ciertos aspectos de la interfaz serie del integrado, así como la disponibilidad de módulos que se tendrá en el microcontrolador elegido, antes de conectarlos.

La interfaz serie del TSL3301 permite controlarlo completamente con 3 señales digitales, compatibles con una USART *isosíncrona*. Este tipo de interfaz utiliza una señal de clock para controlar la entrada y salida de las señales al integrado, y al mismo tiempo utiliza bits de Start y Stop para sincronizar los bytes transmitidos. Este modo de utilización de la USART del integrado no está directamente disponible en el microcontrolador elegido. Por otro lado, se decidió que la interfaz serie entre el microcontrolador y la computadora se realice a través del módulo USART del segundo, y éste posee un sólo módulo disponible, de modo que la posibilidad de utilizar este módulo estaba descartada de antemano.

La solución alternativa elegida para el manejo de este integrado, con un compromiso pequeño de la velocidad de lectura, fue utilizar un puerto de propósito general del microcontrolador para tratar las tres señales que lo controlan con el método de “bit banging”, dejando el módulo USART de hardware disponible para la comunicación entre el microcontrolador y la computadora. En este método de operación, la señal de clock no está activa permanentemente, sino en momento de realizar transmisiones de comandos y lecturas, y también cuando el sensor necesite esta señal para su operación interna. La información específica para lograr este modo de operación se obtuvo con ayuda de una nota de aplicación provista por el fabricante que trata la particularidad mencionada de la interfaz serie del sensor, así como analizando la hoja de datos del integrado.



Conexión entre microcontrolador y sensor óptico lineal
NC – sin conexión

Si se acepta que la comunicación se realice en modo half-duplex, y que el microcontrolador esté dedicado a la comunicación con el sensor mientras se lee (cosas que para este proyecto no traen mayores dificultades), se puede aplicar este método, creando funciones diseñadas para la comunicación como parte adicional del programa.

2.2.3 Velocidad de lectura

El microcontrolador elegido es capaz de ejecutar 10MIPS, de modo que no podrá sostener la tasa de clock máxima aceptable del sensor, de 10MHz (porque sólo para pulsar el clock se necesitarán 2 instrucciones, y como mínimo se necesitan 2 instrucciones más para procesar la entrada del bit y desplazar el byte recibido). Si para cada pixel se transmiten 10bits (Start – Byte Pixel – Stop), se tiene una tasa máxima de transmisión de pixels de 1MHz en general para el sensor. La implementación elegida logra una tasa máxima de transmisión de pixels de 250kHz, mejorable, pero aún aceptable para esta aplicación.

Es necesario tener en cuenta que no sólo será necesario leer los pixels, sino que también se deberá procesar la información recibida para poder trabajar con ella. En la implementación final se ha conseguido una velocidad total de aproximadamente 500 lecturas completas por segundo (medición, procesamiento y almacenamiento en memoria EEPROM). Para lograr esto fue necesario optimizar la función de recepción y evitar llamados de subrutinas, lazos y cálculos innecesarios en el código. Con esta velocidad de lectura se puede explorar, por ejemplo, una línea de 15cm en un segundo, tomando 33 muestras por cada centímetro, o una línea de 20cm en un segundo, tomando 25 muestras por cada centímetro.

2.2.4 Registros programables

El sensor posee 7 registros:

Registro	***Dirección***
Offset Izquierdo	0x00
Ganancia Izquierda	0x01
Offset Central	0x02
Ganancia Central	0x03
Offset Derecho	0x04
Ganancia Derecha	0x05
Modo de operación	0x1F

TSL3301 – Tabla de Registros accesibles

Modo de operación

Este registro posee 8 bits, de los cuales uno determina si el dispositivo entra o sale del modo Sleep. Los demás bits de este registro no tienen sentido para el usuario y deben valer 0. Útil para aplicaciones alimentadas con batería, no se lo utiliza en esta aplicación.

Offset y Ganancia

Como se puede observar en el diagrama en bloques del sensor, es posible corregir la ganancia y el offset (corriente oscura) para mejorar la calidad de imagen y optimizar el rango de las mediciones, así como adaptarlo al sistema en el que se lo utilice.

El sensor posee 102 pixels, que están divididos en 3 secciones de 34, y cada una de estas secciones tiene registros individuales programables que determinan el nivel de offset y ganancia para cada sección.

- *Offset* - Son tres registros de 8 bits codificados en magnitud-signo, con valores posibles entre -128 y 127
- *Ganancia* – Son tres registros de 5 bits de longitud, que aceptan valores entre 0 y 31

Estos seis registros sirven como entrada para 6 conversores D/A, cuyo valor analógico de salida se utiliza para realizar las correcciones sobre los niveles de carga medidos. Sus valores no están determinados luego del encendido del sensor, de modo que es requisito programar su valor por lo menos una vez. Al mismo tiempo, es posible cambiar el valor de estos registros entre mediciones y dar distintos valores para cada sección.

Las correcciones de ganancia y corriente oscura son realizadas en el bloque de conversión de carga a tensión, antes del muestreo de la tensión correspondiente a la medición. La corrección de ganancia se aplica luego de aplicarse la corrección de offset sobre la carga. Es por esto que las modificaciones sobre la ganancia de este bloque pueden variar el nivel óptimo de offset, y puede ser necesario corregir el segundo luego de un cambio en el primero.

2.2.5 Comandos de control del integrado

Para controlar al sensor, se le transmiten comandos vía la interfaz serie, que varían de acuerdo a la función que se busca que ejecute. De los 10 comandos disponibles para el control del integrado, la aplicación realizada utiliza 6, que se enumeran a continuación:

Comando	***Código***	***Descripción***
RESET	0x1B	Reiniciar bloques de integración y lectura
STARTInt	0x08	Comenzar período de integración
SAMPLEInt	0x10	Terminar de integrar y muestrear valores
READPixel	0x02	Iniciar transmisión de pixels leídos
ABORTPixel	0x19	Finalizar transmisión de pixels abruptamente
REGWrite	0x40 + Dirección del registro	Realizar escritura en un registro programable

TSL3301 – Tabla de Comandos

El comando REGWrite debe ser seguido por el valor a asignar al registro seleccionado.

2.3 ASPECTOS DE DISEÑO Y CONSTRUCCIÓN DEL LECTOR

Una vez lograda la interfaz entre el sensor y el microcontrolador, es necesario aplicarlo como parte de un lector utilizable. Para este proyecto se decidió construir un prototipo que asemeje la punta de un resaltador, dentro de la cual se ubica el sensor y el conjunto necesario para que éste pueda tomar imágenes legibles. Se busca que este prototipo sea de tamaño reducido y de manejo intuitivo para el usuario.

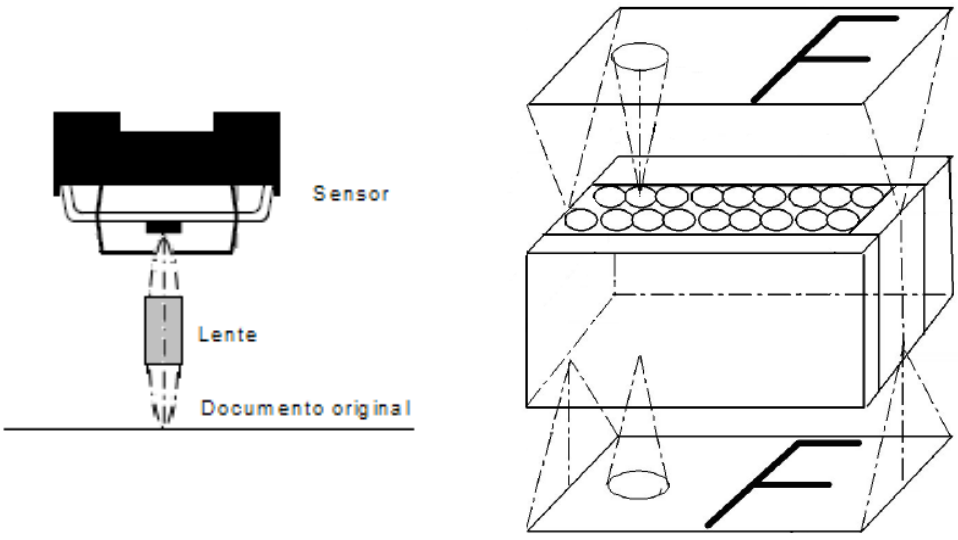
Para la construcción de este lector se tuvieron en cuenta los siguientes aspectos:

2.3.1 Necesidad de Lente

A pesar de poseer características atractivas en cuanto a su capacidad de tomar imágenes, en general el sensor no alcanza por sí sólo para lograr transmitir líneas legibles. Es necesario apoyarse en un sistema de iluminación controlado, y, dependiendo de las características de la implementación, puede ser necesario recurrir a lentes para enfocar la imagen que se desee tomar.

Para utilizar el sensor en una aplicación de reconocimiento óptico de caracteres, es necesario transferir la imagen que se desea analizar directamente al sensor, sin magnificación ni deformaciones. Para lograr una transferencia uniforme de una imagen de una superficie plana, el fabricante recomienda utilizar una matriz de lentes SELFOC®. El principio de este tipo de lentes consiste en utilizar una matriz de microlentes GRIN cilíndricas (con una variación gradual del índice de refracción). El perfil de índice de refracción de estas microlentes es tal que:

- Los rayos incidentes se refractan en una lente de modo que convergen a un punto focal **(a)**
- La disposición geométrica de las microlentes es tal que, entre distintas lentes, los rayos refractados se superponen de modo que se forme una imagen continua, copia de la imagen de origen, sin deformaciones **(b)**



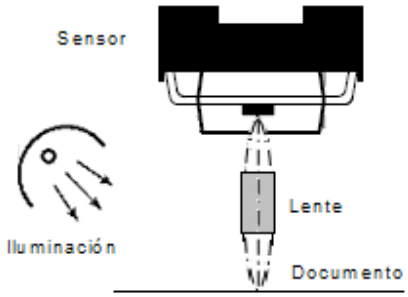
(a) ***(b)***

Características de matriz de lentes GRIN

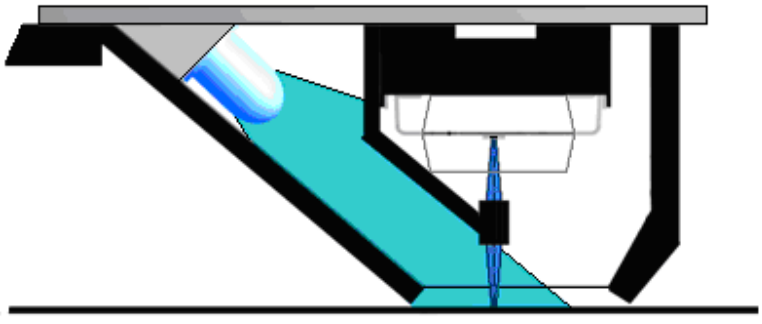
Si bien el fabricante del sensor no ofrece este tipo de lente a la venta, salvo como parte de un módulo de evaluación para el sensor, éste remite a los interesados a su proveedor en Estados Unidos, y también ofrece una nota de diseño donde se indica la forma de calcular los parámetros para el pedido que sean acordes con el sensor lineal, la aplicación deseada y los códigos propios del fabricante. Sin embargo, por cuestiones de tiempo y disponibilidad de recursos, se recurrió a una lente de similares características que se tenía disponible como parte de un fax desechado. Obviamente, esta lente no está diseñada específicamente para trabajar con este sensor lineal en particular, pero su función en ambas aplicaciones es la misma, y una adaptación mecánica para adecuarse a la distancia focal de la nueva lente resulta en una imagen de calidad aceptable y enfocada. Más adelante se mencionará una corrección en software para adaptar definitivamente la lectura a esta lente con un compromiso del rango dinámico del sistema.

2.3.2 Fuente de iluminación interna

La alta sensibilidad del sensor es a su vez un factor que debe ser tenido en cuenta a la hora del diseño. Si consideramos que la implementación que se busca realizar tiene que ejecutar la mayor cantidad de lecturas por segundo posible, un reducido tiempo de integración para la medición de intensidad de luz es deseable, de modo que la alta sensibilidad de sensor resulta apropiada. Será necesario entonces determinar la fuente de iluminación adecuada para que la medición, para el tiempo de integración que se seleccione, resulte en una imagen de contraste y foco razonables.



a)



b)

Implementación de sensor con lente y fuente de iluminación a) Teórica b) Prototipo

Para probar el prototipo se realizaron varias pruebas, teniendo en cuenta que el sistema tenía múltiples variables controlables cuyo valor había que determinar: era necesario determinar el tiempo de integración, el offset y la ganancia del sensor, la disposición física de la lente entre el sensor y la superficie de lectura que optimice el foco, y la fuente de iluminación, que debía tener suficiente intensidad para trabajar en el tiempo de integración deseado, y debía estar dispuesta de modo que la luz a lo largo de la superficie de lectura fuera lo suficientemente uniforme. Durante esas pruebas se encontraron los mejores resultados utilizando LED azul de 3mm.

La característica principal de este LED era que su intensidad era suficientemente buena para que se logre distinguir texto en un tiempo de integración reducido. Se utilizó un LED con longitud de onda visible para que el usuario pueda observar que el lector se encuentra activado. Para lograr que el haz de luz del LED sea menos concentrado, se cubrió el LED con un material translúcido, de modo de dispersar más uniformemente la luz de la fuente. En un futuro es posible controlar la intensidad de esta fuente con una señal PWM, para asegurar la intensidad apropiada, o ubicar dos fuentes de luz de menor intensidad en paralelo para proveer una línea de iluminación más uniforme.

2.3.3 Estructura externa del prototipo

Como ya se mencionó, la sensibilidad del sensor es un factor positivo a la hora de permitir tiempos de integración reducidos, si se aplica una fuente lumínica de intensidad adecuada. Al mismo tiempo, como consecuencia de esta sensibilidad, cualquier otra fuente de luz que alcance al sensor o a la superficie que se quiere leer afectará el resultado de la medición del sensor y hasta podría saturarlo fácilmente. Como construir un prototipo que sea sensible a la luz ambiente parece inaceptable para esta aplicación, se decidió tratar de lograr una estructura externa que encapsule y proteja al sensor de cualquier fuente de luz que no sea la designada.

2.3.4 Botón de operación

Para que el sistema resulte más amigable para el usuario, se incorporó un botón que permite que éste decida cuándo ejecutar la lectura del texto. Este botón se ubica a un costado del sensor, de modo que el lector terminado resulte funcional y relativamente compacto. Cuando el usuario termina de pasar el sensor sobre la línea, suelta el botón, y comienza la segmentación y transmisión de los datos guardados hacia la computadora. El botón se integra con una resistencia de pull-up en la placa del microcontrolador, y lleva la señal a tierra cuando el usuario lo oprime.

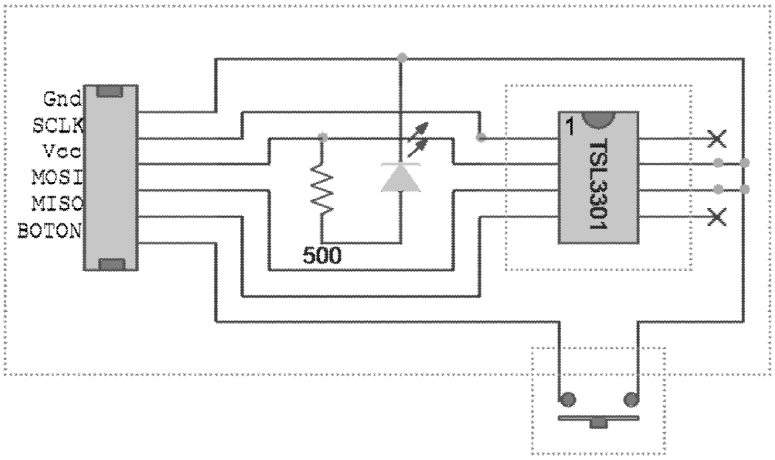
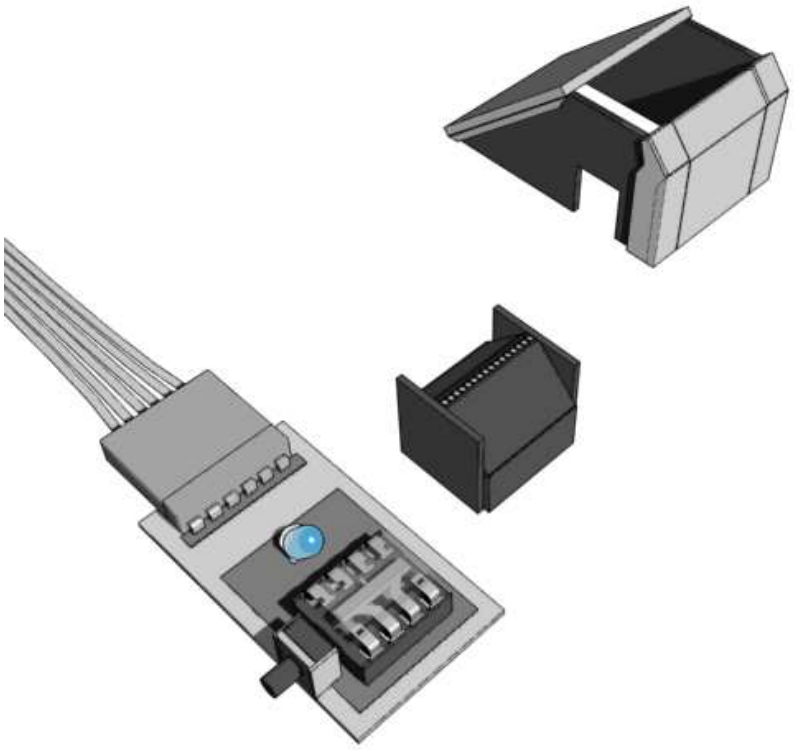


Diagrama de circuito de lector con fuente de iluminación y botón de operación

2.3.5 Construcción del lector - Región final para lectura

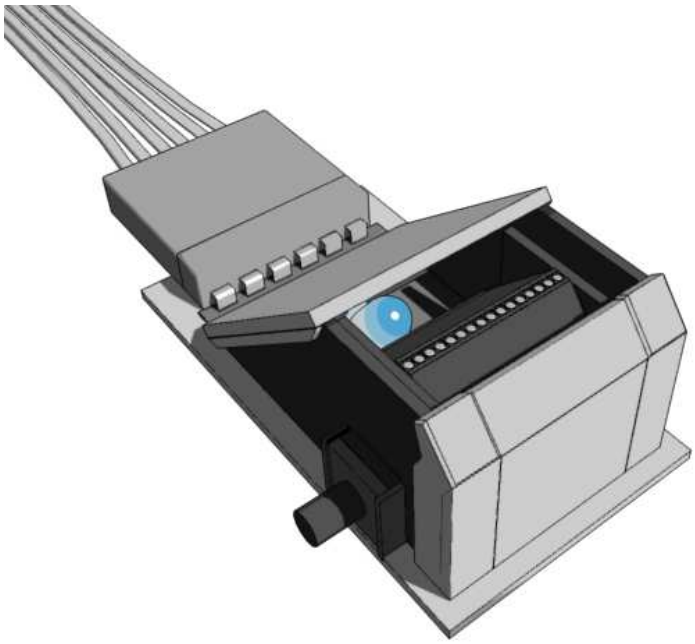
El resultado de la construcción del lector se simula en las imágenes a continuación. Se realizaron pruebas, controlando la distancia entre la lente, el sensor, y la superficie de lectura, para que el lector tenga un nivel adecuado de foco, dadas las condiciones de iluminación y medición establecidas. Como consecuencia de la disposición final de los elementos en el lector, y por el método de fijación de la lente sobre el sensor, se obtuvo una región de lectura final que abarca 96 de los 102 pixels disponibles, con características de iluminación lo suficientemente uniformes para esta aplicación.



Estructura de lector prototipo

El área de lectura final resulta lo suficientemente grande para abarcar caracteres en un amplio rango de tamaños, como se buscaba. Es por esto que resulta posible que el sensor abarque más de una línea de texto simultáneamente, cuando en verdad se busca explorar una sola línea a la vez. Esta posibilidad deberá ser contemplada en el algoritmo final de segmentación, para obtener una aplicación tan robusta como sea posible, y hace que sea necesario un aumento en la complejidad de la función que cumple ese propósito.

El posible incorporar mejoras al dispositivo que aporten más funcionalidades, como sensores que midan su desplazamiento lineal, o ruedas que faciliten el deslizamiento del lector sobre la superficie de lectura. De cualquier forma, tales mejoras se agregarían al sistema que se desarrolló, sin alterar su estructura interna.



Prototipo de lector para aplicación OCR

2.4 MEMORIA EEPROM 24LC256

2.4.1 Motivo de utilización de este tipo de memoria

La memoria EEPROM 24LC256 de Microchip es una memoria no volátil de alta densidad, compatible con el protocolo I^2C. Si bien, por el modo en que se utiliza esta memoria, se podría haber elegido otra volátil con el mismo propósito, no se encontró una memoria RAM *serie* de la misma capacidad en un integrado de tamaño comparable. De todas formas, visto que el integrado acepta un millón de ciclos de lectura-escritura, y ofrece una gran cantidad de memoria disponible con interfaz serie, se entiende que este tipo de memoria es apropiado para esta aplicación.

El principal motivo para la utilización de una memoria externa fue la intención de lograr una buena velocidad de lectura, para que el usuario pueda utilizar el lector de forma más cómoda. Si en un mismo ciclo se realizara la lectura desde el sensor, el análisis de la información recibida, la segmentación de caracteres y la transmisión de la región para el OCR, todo este tiempo sumado al tiempo de ejecución del OCR en la computadora cuando se completa un símbolo concluiría en una velocidad de lectura de pocas líneas por segundo. En las primeras pruebas se operaba de esta forma, y se tenía una velocidad de lectura y transmisión de líneas en escala de grises de aproximadamente 60 líneas por segundo, cosa que obligaba al usuario a desplazar el lector lentamente sobre el texto para recibir líneas legibles. Teniendo en cuenta esto, se tomaron acciones para mejorar la velocidad de lectura:

- *Operar en blanco y negro:* Se ahorra memoria y se optimizan los tiempos de análisis y detección de límites de caracteres. La función de OCR funciona en blanco y negro sin mayores diferencias, mientras se observen espacios que separen los trazos (condición límite para una detección óptica no intuitiva).
- *Separar el proceso en dos:*
  - Al oprimir el botón, realizar lecturas con mínimo análisis y guardarlas en la memoria EEPROM tan rápido como sea posible, sin transmitir nada a la computadora.
  - Al soltar el botón, leer los valores guardados en la memoria, analizarlos y transmitirlos a la computadora, detectando límites de caracteres.

  El usuario sólo tiene que oprimir el botón y pasar el lector sobre el texto, luego soltarlo y esperar que se procese la imagen guardada.

Por otro lado, en esta aplicación se busca realizar una acción conjunta entre el microcontrolador y la computadora, para que, aprovechando las propiedades de cada uno, se pueda realizar una operación de reconocimiento más eficiente. No se busca que el microcontrolador actúe sólo como un repetidor de la lectura del sensor. Su modo de operación es tal que permite optimizar fácilmente partes de los recursos, y permite salvar dificultades propias del módulo de reconocimiento, como se mencionará más adelante.

2.4.2 Operación I^2C

El bus I^2C (Inter-Integrated Circuit) permite comunicar microcontroladores y múltiples periféricos por una interfaz serie de dos cables (sin contar la tierra). El módulo MSSP del microcontrolador puede funcionar como módulo I^2C maestro, cosa que será aprovechada para conectar al microcontrolador con la memoria externa.

El dispositivo que actúa como maestro en I^2C entrega la señal de clock (SCL), y los datos se transmiten en la señal SDA. Ambas líneas se diseñan en modo open-drain, de modo que se necesitan resistencias de pull-up para asegurar el valor de la tensión para el estado lógico ´1´ (cuando se suelta la línea). Llevar la línea a tierra se considera un ´0´ lógico.

El diagrama de conexión para la memoria EEPROM se muestra a continuación:

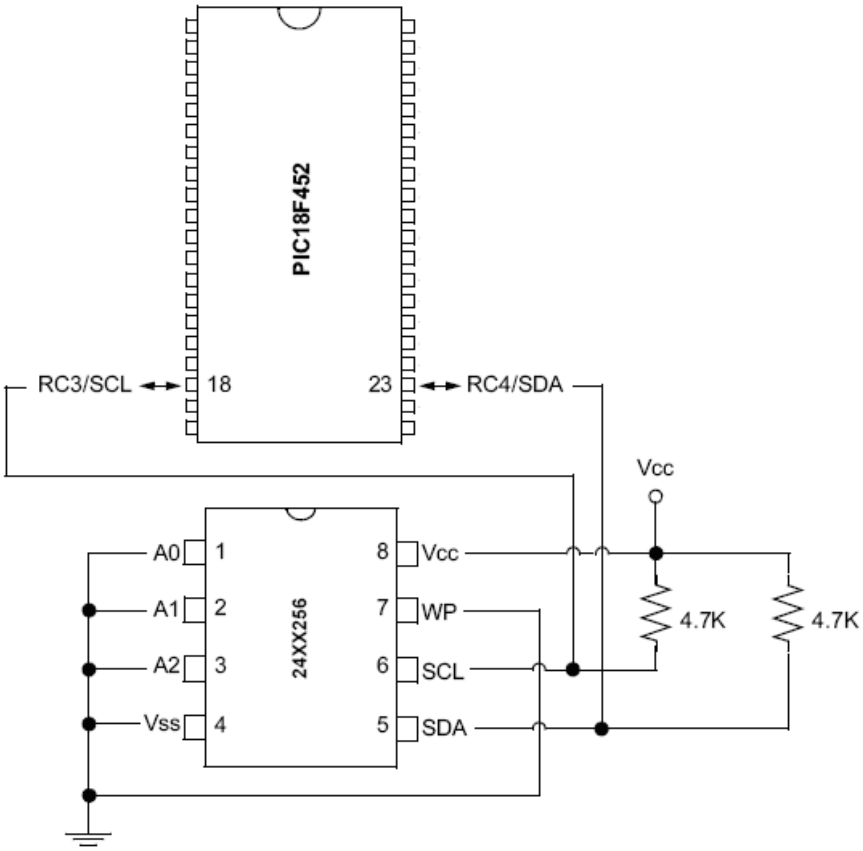


Diagrama de conexión 24LC256 con PIC18F452
Fuente: Nota de aplicación AN991, Microchip

El módulo I^2C del microcontrolador se configura en modo Master I^2C, y la memoria sólo responde a los comandos que se le soliciten. Los valores de las resistencias son recomendados en la hoja de datos de la memoria, y en la mencionada nota de aplicación del fabricante.

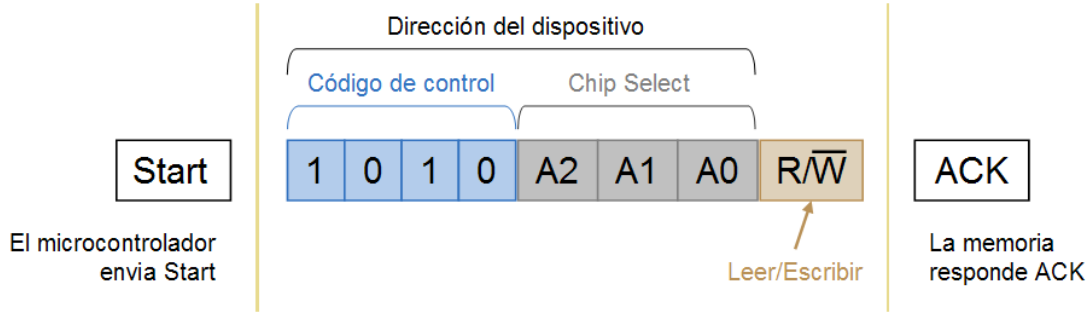
La velocidad de reloj serie (SCL) elegida (impuesta por el microcontrolador), de 400kHz, es la más alta aceptada por esta memoria.

2.4.3 Protocolo

Todos los mensajes en I^2C comienzan en el maestro con una señal de Start (transición alto-bajo de SDA con SCL alto), y terminan con una señal de Stop (transición bajo-alto de SDA con SCL alto), también transmitida por el maestro.

Byte de control

El byte de control es el primer byte enviado por el dispositivo maestro.



- Los primeros 4 bits transmitidos son el código de control, que indican la clase de dispositivo a la que se envía el comando y es 1010 para esta memoria.
- Los 3 bits siguientes permiten diferenciar 8 memorias 24xx256, son bits de chip select que sirven para realizar expansiones de memoria.
- El último bit indica el tipo de operación que se solicita a la memoria. 1 para lectura y 0 para escritura

La memoria responde al byte de control con un bit de Acknowledge (señal activa baja).

Bytes de dirección

Como la memoria 24LC256 posee 32768 bytes, se necesitan 15 bits para identificar cada byte disponible. Es por esto que esta memoria requiere la transmisión de 2 bytes de dirección, luego del byte de control, para indicar sobre qué dirección se va a ejecutar el comando transmitido –existen funciones que no requieren la selección de la posición de memoria, como la opción de leer sobre la dirección actual, y el algoritmo de acknowledge polling.

Page Write

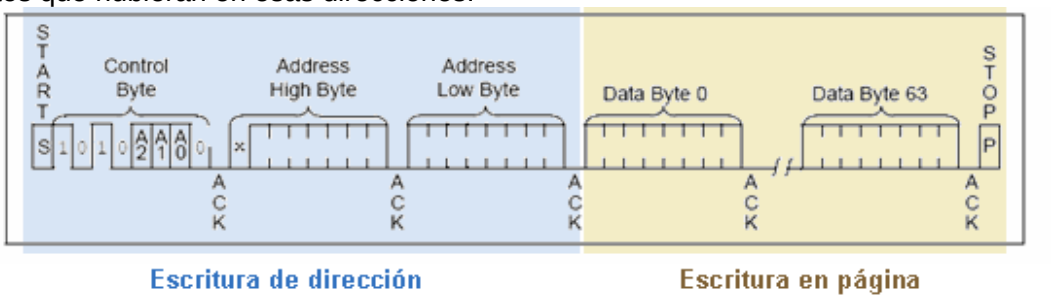
Luego del envío de la señal Start, el byte de control indicando que se desea escribir, y de recibir el acknowledge por parte de la memoria, el siguiente byte transmitido es el byte de mayor orden de la dirección donde se quiere escribir. Se recibe otro acknowledge, y se transmite el byte menos significativo de la dirección.

Al confirmarse recepción (ACK), se transmite el primer byte a grabar en la posición de memoria indicada, y se confirma por parte de la memoria. El maestro transmite el siguiente byte a grabar en la página, y este proceso continua hasta que se transmite el byte número 64 como máximo, y se completa así la escritura de la página.

Luego de recibirse el último acknowledge, el maestro genera la condición de Stop, y se termina el comando. Es en este punto que se inicia el ciclo de escritura de los bytes transmitidos.

La escritura de páginas se limita a los extremos de la página (por ejemplo, entre la dirección 0 y 63 de la memoria). Si se intentara realizar una escritura a través del

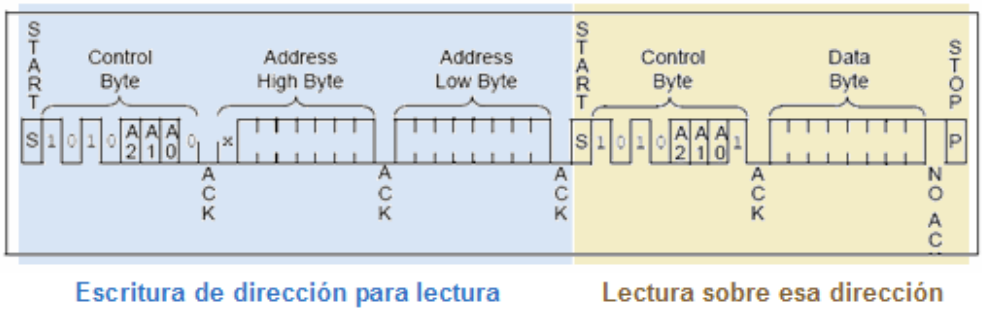
límite de una página, el contador interno volverá a su inicio, y se escribirá sobre los datos que hubieran en esas direcciones.



Esquema de ejecución de escritura de página sobre 24LC256
Fuente: Hoja de datos 24LC256, Microchip

Lectura Aleatoria / Lectura Secuencial

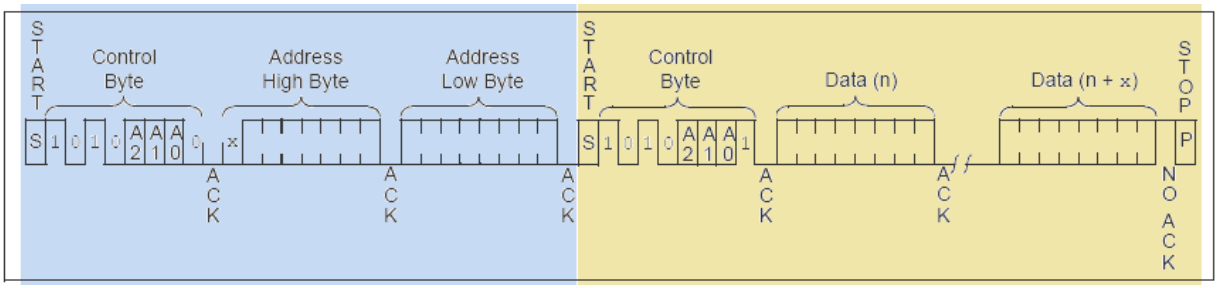
El proceso de lectura de una dirección específica comienza escribiendo la dirección donde se quiere leer, como los 3 primeros bytes de la escritura de página. Luego de recibir el ACK de la memoria, el maestro envía otro Start, y otro byte de control, indicando que se quiere leer (R/W = 1). Se confirma la recepción del segundo byte de control, la memoria transmite el byte, y el maestro no genera un ACK (NO ACK), y genera una condición de Stop.



Esquema de ejecución de lectura simple para 24LC256
Fuente: Hoja de datos 24LC256, Microchip

Para una lectura secuencial se hace lo mismo, pero el maestro envía un ACK en vez del NO ACK, y la memoria sigue transmitiendo valores de las direcciones siguientes a la escrita al principio. Este proceso puede seguir hasta que el maestro no genera la condición de ACK y manda un Stop.

La lectura secuencial no tiene límites como la escritura de página, se puede leer la totalidad de la memoria en una sola operación.



Esquema de ejecución de lectura secuencial para 24LC256
Fuente: Hoja de datos 24LC256, Microchip

2.4.4 Ciclo de escritura

La hoja de datos de la memoria EEPROM indica que, para cada proceso de escritura de un byte o de una página, se debe cumplir un ciclo de escritura máximo de 5ms. El tiempo de escritura se suma al tiempo de transmisión de la información en el bus I^2C, y debe ser tenido en cuenta a la hora de decidir si su implementación es viable.

La memoria posee un propiedad que hace posible detectar cuándo se cumple el tiempo de escritura, para optimizar el número de escrituras por segundo. El dispositivo no genera una condición de acknowledge durante el ciclo de escritura, de modo que, después de un comando de escritura, se pueden enviar bytes de control hasta que el dispositivo responda con un acknowledge. Cuando la memoria genera esta condición, quiere decir que está lista para una nueva escritura.

Diagrama de flujo para acknowledge polling
Fuente: Hoja de datos 24LC256

Con esta utilidad no es necesario esperar siempre 5ms para asegurarse de que las escrituras en memoria se realicen correctamente.

2.5 MAX232

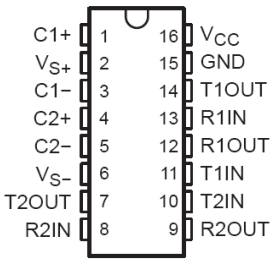
2.5.1 Motivo de utilización de este integrado

Como ya se mencionó, se decidió que la comunicación entre el microcontrolador y la computadora se realice aprovechando el módulo USART del primero, para conectarlo con un puerto serie en una computadora. Para lograr esto, es necesario realizar una adaptación a nivel de capa física, debido a la diferencia entre los valores de tensión correspondientes a los niveles lógicos ´1´ y ´0´ en el microcontrolador y en el puerto serie de la computadora.

- Comúnmente, un ´1´ lógico en el standard RS232 se manifiesta con una tensión negativa entre -3V y -15V, y un ´0´ lógico se manifiesta con una tensión positiva entre +3V y +15V. Los valores de tensión cercanos a 0V no tienen significado en este standard.

- En el microcontrolador, alimentado con una fuente de 5V, un nivel bajo corresponde a tensiones cercanas a 0V, y un nivel alto corresponde a tensiones cercanas a la alimentación (la lógica que se asigne a esos niveles depende del criterio del programador). El umbral de decisión para estos niveles varía levemente con las tecnologías utilizadas para el hardware de los distintos puertos.

La conversión de niveles de tensión se logra con el integrado MAX232, que sirve como driver y conversor para 2 entradas y salidas, y por tanto permite traducir las señales lógicas de entrada a los niveles de tensión apropiados para el tipo de tecnología a cada lado de la interfaz según corresponda.



Integrado MAX232, vista superior
Fuente: Hoja de datos MAX232, Texas Instruments

Este integrado incluye un generador de tensión capacitivo, que traduce niveles de tensión de entrada de tecnologías TTL/CMOS (del lado del microcontrolador) a niveles dentro del standard RS232 (del lado de la computadora). También convierte tensiones correctamente en el sentido contrario.

2.5.2 Velocidad de transmisión para la interfaz

Como cada línea significativa que se reciba en el microcontrolador se transmitirá en cierto punto a la computadora (sea en escala de grises o en blanco y negro), se buscará para la comunicación entre ambos la velocidad más alta alcanzable, que se encuentre dentro de las posibilidades del microcontrolador y de los límites de compatibilidad entre éste y la computadora. Esta velocidad de transmisión se obtuvo en los 230400 bauds.

2.5.3 Conexión

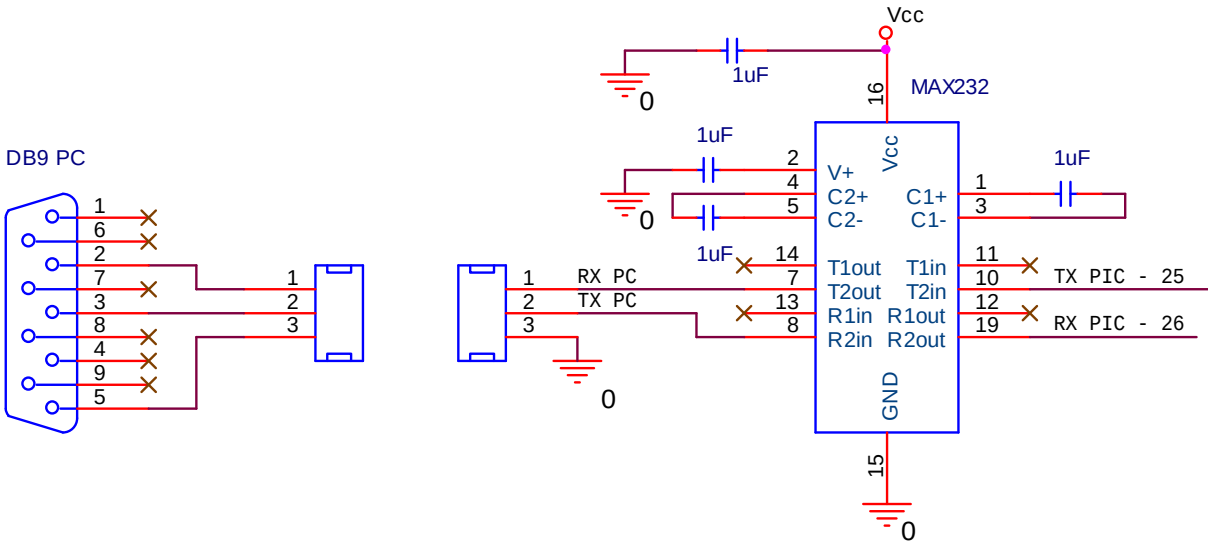


Diagrama de conexión PC - MAX232 – Microcontrolador PIC18F452

En esta aplicación en particular, se utiliza la interfaz indicada en la imagen superior, junto con un adaptador RS232-USB que permite una conexión directa con computadoras que no dispongan de un puerto RS232 compatible. Existen varias soluciones que logran este propósito, de costo variable, y que van desde cables adaptadores a la venta hasta proyectos comprobados que permiten construir el adaptador individualmente.

2.6 CONTROL DE DESPLAZAMIENTO *

La pregunta en este punto consistiría en cómo determinar la forma en que se decide cuándo tomar imágenes del lector, para que las lecturas representen trazos de imágenes similarmente espaciadas. Una alternativa es tomar un intervalo de tiempo constante para realizar las mediciones. Como se está tratando con un prototipo que es operado por un humano, es posible que el desplazamiento del lector a lo largo de un renglón no sea perfectamente recto, y que la velocidad del desplazamiento del lector no sea constante (aunque si el sistema permite operar a alta velocidad, normalmente las variaciones de velocidad serán reducidas). Existen sensores que permiten medir el desplazamiento del lector sobre la superficie, y pueden ayudar a corregir errores de desplazamiento, como por ejemplo una rueda codificada (encoder rotacional) de alta resolución, que transmita pulsos cada vez que la rueda se gira un cierto ángulo, o un acelerómetro, que permita calcular la aceleración instantánea del lector.

Para el caso del encoder rotacional, se transmitiría información al microcontrolador que indicaría directamente cuando se debe medir la siguiente línea, pero con esta alternativa se necesita contacto con la superficie a analizar, para que pueda girar la rueda.

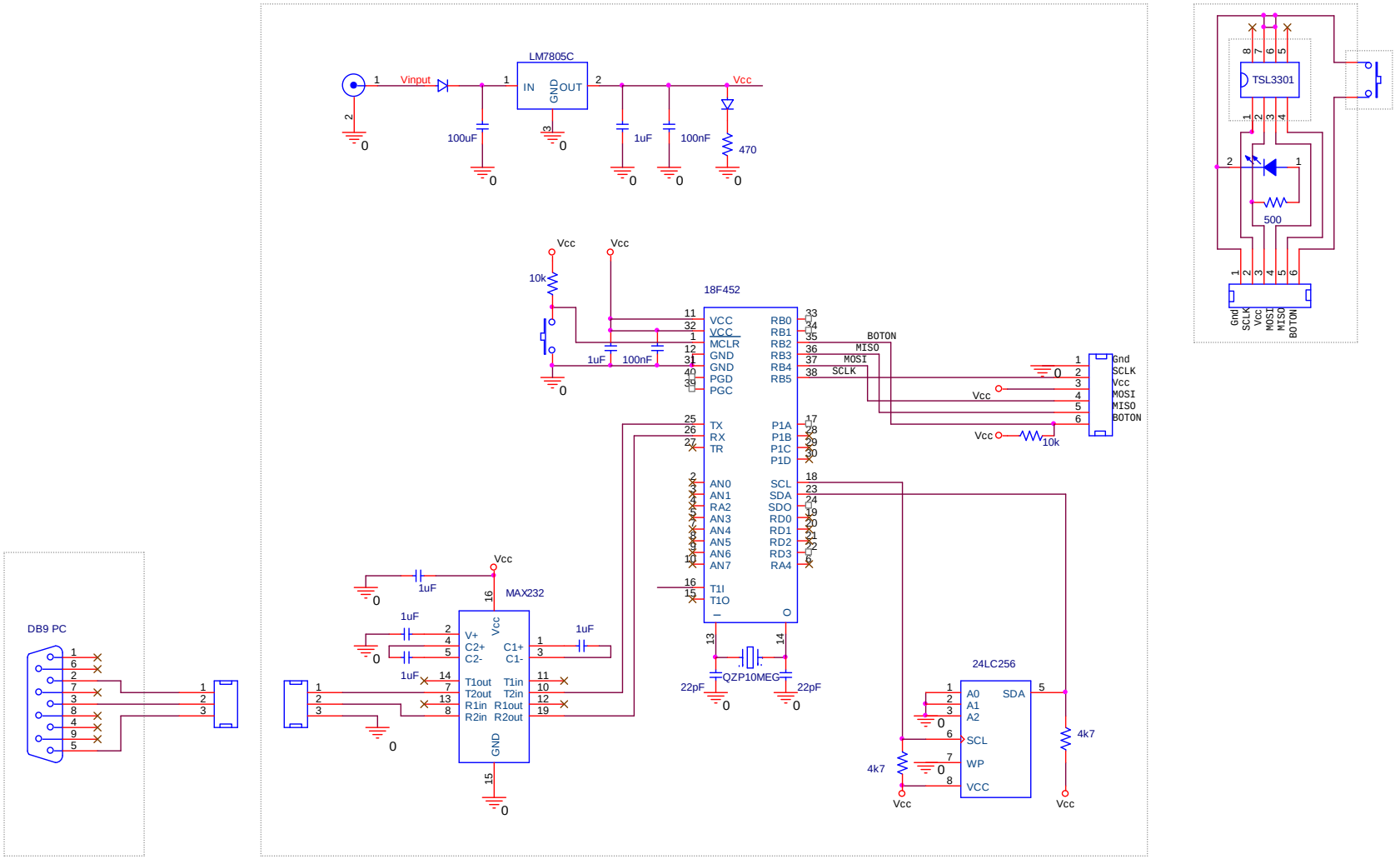
El caso del acelerómetro es más complejo: para convertir información de aceleración a posición es preciso integrarla, cosa que en un microcontrolador se traduce a realizar mediciones sucesivas y sumarlas (multiplicándolas por el intervalo de tiempo entre mediciones). Dependiendo de la forma en que se utilice la información, podrá ser necesario integrar la aceleración dos veces, para convertirla a posición. Para que la información resultante de esta integración tenga el menor error posible, es necesario muestrear la aceleración tan rápido como sea posible. Este requerimiento impide incorporar la lógica de control del acelerómetro en el microcontrolador principal del sistema, que estará ocupado ejecutando la función principal del programa.

La alternativa de solución para control de desplazamiento con acelerómetro consiste en incorporar otro microcontrolador, de menores capacidades, que sea el encargado de muestrear el acelerómetro tan rápido como pueda. Cuando la cuenta resultante del procesamiento de la información indique que se ha recorrido la suficiente distancia, será momento de que este controlador secundario indique al primario que debe realizar la siguiente lectura del sensor. La cuenta de la posición volverá a cero (aunque deberá guardarse la velocidad previa), y se volverá a contar hasta que se recorra el mismo trayecto otra vez.

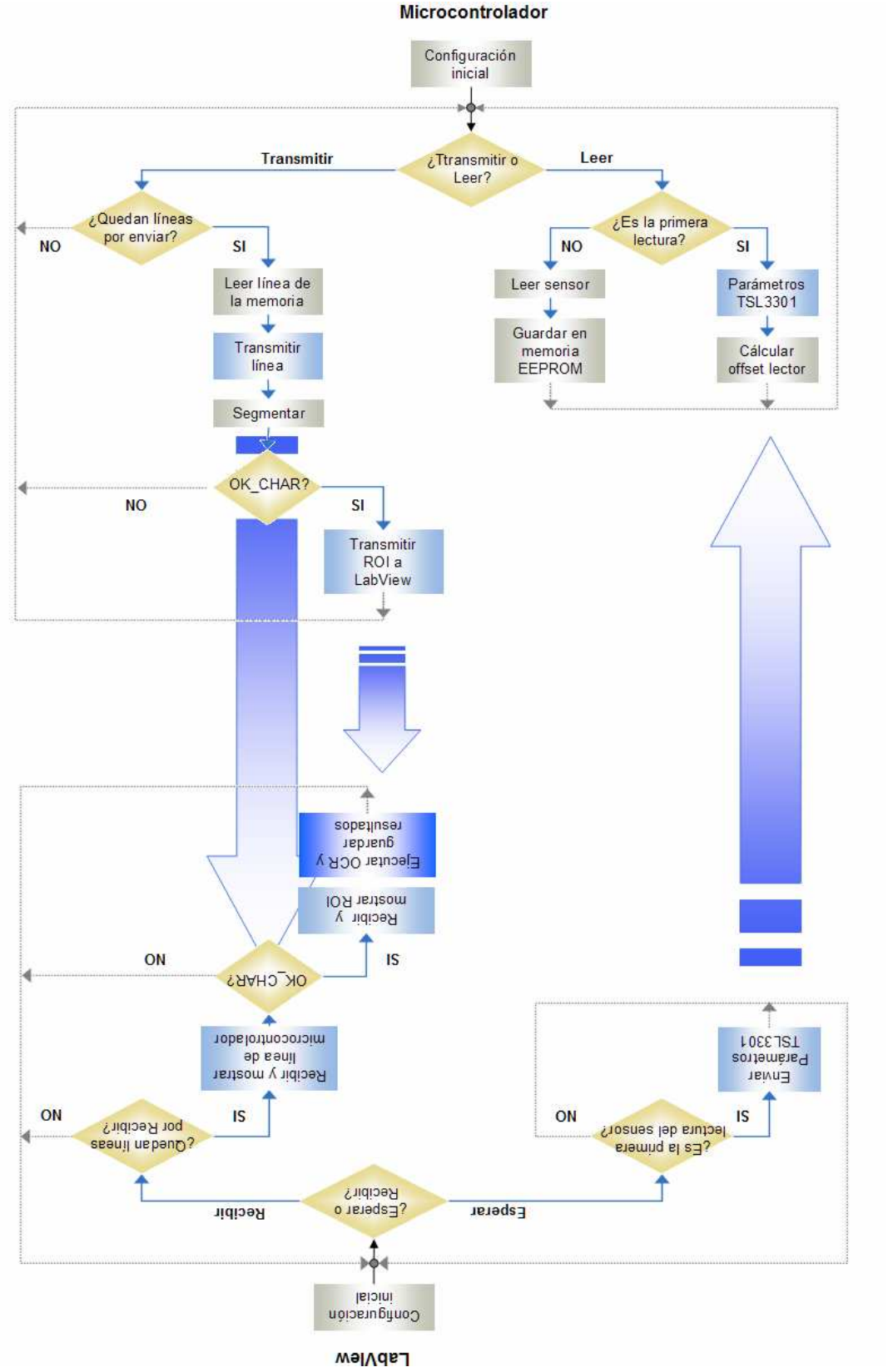
En el caso del acelerómetro, las lecturas solamente serían válidas si la superficie de lectura no fuera curva, ya que cualquier acelerómetro mide inevitablemente la aceleración de la gravedad. Es posible anular esta aceleración sumando un sensor de inclinación al sistema, pero esto suma complejidad para un control que necesita tener alta velocidad de muestreo por los motivos que ya se mencionaron.

* Esta funcionalidad no forma parte del prototipo desarrollado

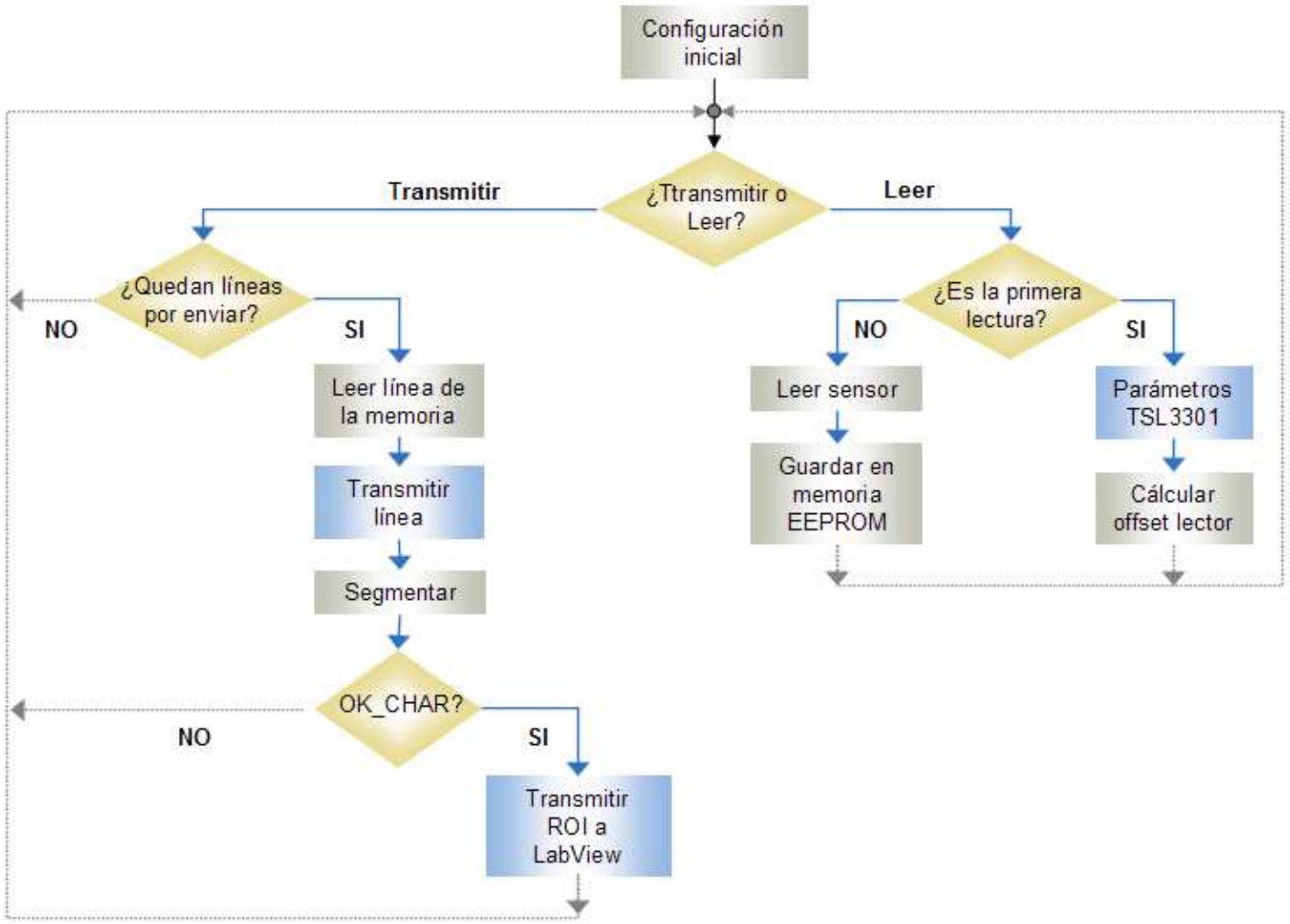
2.7 DIAGRAMA COMPLETO DEL SISTEMA



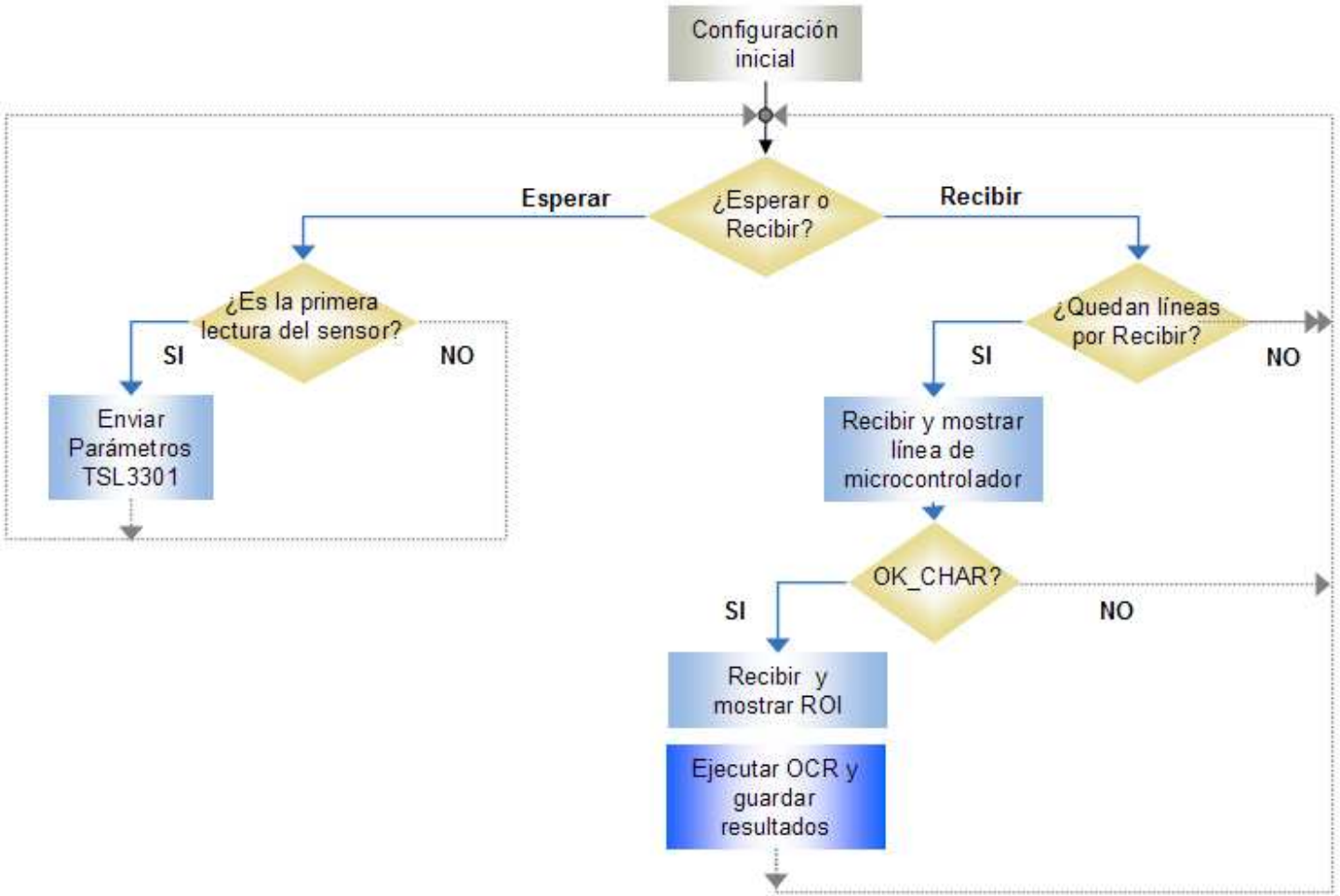
2.8 OPERACIÓN BÁSICA DEL PROGRAMA CONJUNTO – VERSIÓN FINAL



Operación básica de la aplicación
Microcontrolador



Operación básica de la aplicación LabVIew



2.8.1 Funcionamiento general

La idea básica detrás de la aplicación desarrollada se muestra en los diagramas de las páginas anteriores.

o **Microcontrolador PIC**

El microcontrolador se ocupa de leer imágenes desde el sensor lineal, guardarlas en la memoria, transmitirlas a la aplicación en LabView y segmentar caracteres, de acuerdo con la siguiente operación:

- Usuario presiona el botón: **Leer**
  - Recibir parámetros del sensor desde el LabView
  - Guardar líneas en memoria EEPROM
- Usuario suelta el botón **Transmitir**
  - Transmitir las líneas guardadas al LabView
  - Segmentar a medida que se transmite (Reconocer fin de un símbolo)
  - Indicar fin de símbolo a LabView
  - Enviar ROI (Región de Interés, posición del símbolo) al LabView

o **LabView**

La aplicación en LabView se ocupa de enviar los parámetros para configurar el uso del sensor, recibir las líneas medidas y mostrarlas en pantalla, ejecutar OCR sobre la región de interés que se le indique y guardar los resultados en un archivo de texto.

- Usuario presiona el botón: **Esperar**
  - Transmitir parámetros del sensor al PIC
- Usuario suelta el botón **Recibir**
  - Recibir y mostrar líneas en pantalla
  - Cuando el PIC indica fin de símbolo (OK_CHAR), ejecutar OCR en ROI
  - Guardar caracteres en archivo de texto

Para mantener un esquema sencillo, los diagramas anteriores no especifican los pormenores necesarios en ambas partes para permitir que esta aplicación funcione, como la función de sincronización entre el microcontrolador y el LabView, y la forma en que se indica al LabVIew el estado del botón de operación, o si se completó o no un símbolo.

2.8.2 Sincronización

La solución para la sincronización del sistema se logró con dos herramientas:

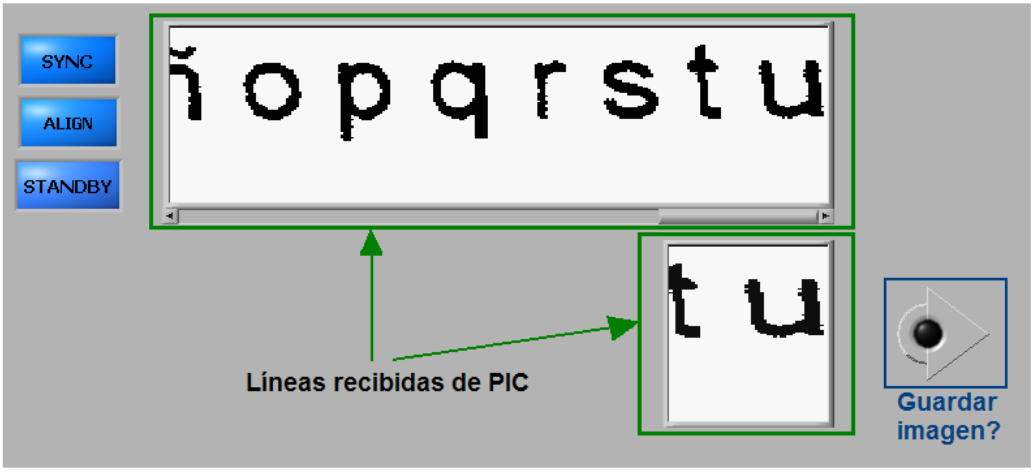
- Una función de sincronización entre ambos equipos, que se ejecuta como inicio del lazo principal
- La transmisión de flags en momentos específicos desde el microcontrolador, que indican el estado de la lectura.

  Por ejemplo, si el usuario está presionando el botón, se transmite una “R” (“leyendo”) al LabView, y si no lo esta presionando, se transmite una “P” (“procesando”). Dependiendo de lo que se reciba en el programa del LabView, se decidirá qué hacer en ese lado de la aplicación.

2.8.3 Fin de símbolo, análisis de imagen y ROI

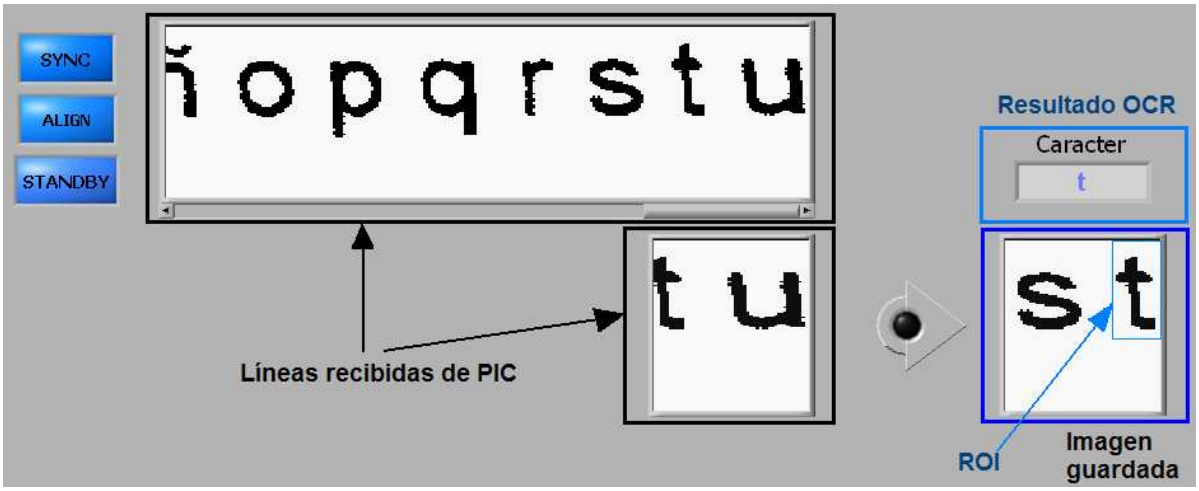
Cuando el PIC transmite las líneas al LabView, éste las muestra en pantalla. Mientras transmite, el PIC, analiza las líneas y ejecuta una función de segmentación, de modo que **el microcontrolador sabrá cuándo se termina un símbolo en el momento en que ese símbolo se hace completamente visible en pantalla**.

Teniendo esto en cuenta, se hace que el microcontrolador mande otro flag luego de transmitir cada línea, que indica si con esta línea se termina o no un símbolo. Cuando el LabView recibe de parte el PIC el indicador de que se terminó un símbolo, saca un "foto" de lo que tiene en pantalla (que incluye al símbolo completo) y la guarda. La imagen incluye el contenido del gráfico y sus bordes, que son ignorados.



El programa en LabView guarda el gráfico inferior cuando el PIC le indica que se terminó un símbolo (cuando la luz se prende)

Lo siguiente es recibir la ROI del PIC. La ROI indica qué parte de la foto que sacó el LabView tiene el símbolo que se quiere reconocer, y se calcula en el PIC. Cabe destacar que el **PIC deja pasar algunas líneas luego de que sabe que se terminó un símbolo antes de avisar** al LabView, para evitar problemas con ciertos caracteres.



*La imagen inferior derecha es la superposición de la imagen que se guardó cuando se terminó la letra **t** y la ROI recibida*

Para calcular la ROI en el microcontrolador fue necesario hacer una conversión de tamaños: El gráfico inferior de la imagen de arriba abarca 60 lecturas.

- El PIC transmite 60 líneas de 96 pixels de alto.
- La imagen que se guarda en la computadora (la inferior derecha) tiene una resolución de 144x144 pixels, e incluye bordes de 4 pixels de ancho.

Esta conversión se cumple en el PIC, antes de transmitir la ROI. Es importante considerar que **los trazos que toquen el límite de la ROI serán ignorados**. Con la imagen recibida y la ROI adecuada, se ejecuta la función de OCR en el LabView, y el resultado se guarda en un archivo de texto.

2.8.4 Espacios, puntos y comas

El programa que se utiliza para hacer OCR no reconoce espacios en blanco. Los puntos y las comas son tipos especiales de caracteres, debido a su reducido tamaño relativo y su simplicidad gráfica. En este proyecto se aplican métodos alternativos para la detección de espacios en blanco, puntos y comas. En estos casos, el microcontrolador analiza la información que tiene a su disposición, y directamente ordena al LabView guardar el símbolo que detecte en el archivo de texto.

2.9 PROGRAMA MICROCONTROLADOR

Esta sección trata el grupo de funcionalidades que fue necesario incorporar al microcontrolador para poder lograr el funcionamiento general del sistema que se trató en la sección anterior.

2.9.1 Módulo USART

- *Descripción*

La comunicación con la computadora se realiza utilizando el módulo USART del microcontrolador en modo asíncrono. Para esta comunicación se busca una velocidad alta, por lo que resulta apropiado haber elegido un cristal de alta velocidad con el multiplicador de frecuencia (PLL) activado (el microcontrolador está trabajando en el límite superior de su velocidad de funcionamiento, 40MHz, 10MIPS).

Fue necesario encontrar una velocidad para la cual exista compatibilidad entre los valores admisibles por el módulo serie de la computadora, y los valores alcanzables por el microcontrolador, determinado por el valor de SPBRG. La velocidad más alta para la que se logró compatibilidad fue 230400 bauds. Para lograr esta velocidad fue necesario calcular el valor del byte SPBRG, para BRGH = 1 (indicador de alta velocidad asíncrona). La ecuación utilizada fue obtenida de la hoja de datos del microcontrolador:

$$\textbf{Baud Rate} = F_{osc} / (16 (\textbf{SPBRG} + 1))$$

Con **BRGH = 1**, F_{osc} **= 40MHz**, y **Baud Rate = 230400**, se tiene **SPBRG = 9,85** (se elige 10).

Aprovechando la librería para manejo del módulo USART del compilador C18 (accesible al incluir el archivo de cabecera *usart.h*), el control de este módulo se volvió más sencillo.

- *Configuración*

La configuración del módulo se realiza en la instrucción de apertura del mismo. Se trabaja sin interrupciones en la comunicación, en modo asíncrono. Se transmiten 8 bits de información y se acepta recepción continua.

```
//Inicialización USART
//Interrupciones  OFF     //Recepción Continua        //Modo 8 bits
//MODO ASINCRONO 230400 Bauds (spbrg 10), BRGH = 1
OpenUSART( USART_TX_INT_OFF & USART_RX_INT_OFF
& USART_ASYNCH_MODE & USART_EIGHT_BIT
& USART_CONT_RX, 10);
```

- *Recepción*

Para la recepción de bytes desde la USART se utiliza el siguiente código (a modo de ejemplo):

```
while (!DataRdyUSART());    //Esperar recepción de byte completa
//DataRdyUSART devuelve el valor de RCIF en PIR, flag de recepción
Recibido=ReadUSART();       //Leer byte recibido y guardarlo en Recibido
```

- *Transmisión*

Para la transmisión de bytes, se utiliza el siguiente código a modo de ejemplo:

```
while (BusyUSART());        //Esperar a que el transmisor esté libre
putcUSART(Valor);           //Transmitir el byte Valor
```

2.9.2 Sincronización con LabView

Del lado del microcontrolador, la función syncro() permite asegurar el sincronismo para la comunicación con la aplicación sobre LabView. Esta función se ejecuta como primera instrucción del lazo principal, y consiste en la transmisión de 4 bytes. Los primeros 2 bytes se transmiten desde la aplicación en LabVIew al microcontrolador (se transmite “ST”). Una vez que el microcontrolador recibe este string, responde con otros 2 bytes (enviando “OK”).

```
void syncro (void){
volatile unsigned char sync;

        //El PIC recibe “ST” del LabView
START:
        sync=' ';
        while(sync != 'S'){                     //Se espera en este estado hasta recibir ´S´
                while(!DataRdyUSART());         //Esperar recepción de byte completa
                sync=ReadUSART();               //Guardar el byte en sync

                if(sync!='S')
                        goto START;             //Si no se recibio lo correcto, volver
        }
        sync=' ';
                while(!DataRdyUSART());         //Esperar recepción de byte completa
                sync=ReadUSART();               //Guardar el byte en sync

                if(sync!='T')
                        goto START;             //Si no se recibio lo correcto, volver

        //El PIC envia “OK” para que el LabView lo interprete

        while(BusyUSART());             //Esperar a que el transmisor esté libre
        putcUSART('O');                 //Enviar “O”

        while(BusyUSART());             //Esperar a que el transmisor esté libre
        putcUSART('K');                 //Enviar “K”

}       //Las aplicaciones estan sincronizadas
```

Si bien no se utilizan funciones específicas para esto, se incluye como parte del sistema de sincronización al mencionado envio de flags (que funcionan como comandos) desde el microcontrolador al LabView.

El microcontrolador envia bytes en ciertos puntos del programa, que indican el estado del sistema. El programa en LabView recibe estos indicadores, y responde en forma coherente. El proceso de envio de flags y su significado se explica con el siguiente pseudocódigo simplificado de lazo principal:

```
while(1){
	syncro();		//Sincronizar PIC y LabView

	if(Se está presionando el botón){
		while(BusyUSART());
		putcUSART('R');

		if(Primera pasada desde que se oprime el botón){
			while(BusyUSART());
			putcUSART('1');

			//Recibir parámetros y calcular offset (ver sección 2.8.5)
		}
		else{
			while(BusyUSART());
			putcUSART('2');

			//Recibir líneas y guardarlas en EEPROM
		}
	}
	else{			//Si el usuario soltó el botón
		while(BusyUSART());
		putcUSART('P');

		if(Termino de enviar trazos){
			while(BusyUSART());
			putcUSART('1');
		}
		else{
			while(BusyUSART());
			putcUSART('2');

			//Leer desde EEPROM, transmitir línea y procesar
			if(Termino símbolo){
				while(BusyUSART());
				putcUSART('R');

				//Enviar ROI
			}
			else{
				while(BusyUSART());
				putcUSART('O');
			}
		}

	}
}
```

2.9.3 Botón de operación

El programa no incluye mecanismos de protección ante posibles fallas mecánicas transitorias que afecten la señal proveniente del botón (anti-rebote), simplemente porque no se considera necesario incorporarlo:

En el lazo principal del programa se empieza preguntando si el botón está activado o desactivado, y en función de eso se decide lo que se va a hacer. En cualquier caso, el tiempo de ejecución del lazo principal siempre toma un lapso en el orden de los milisegundos. Es el mismo código del lazo principal el que cumple la función de mecanismo anti-rebote para atenuar el efecto de fallas mecánicas transitorias que afecten la señal del botón.

2.9.4 Comunicación con TSL3301-LF

Para lograr la comunicación del microcontrolador con el sensor, es necesario crear funciones que transmitan los comandos y cumplan los requerimientos de clock según las especificaciones del sensor. Esto resultó en la creación de las siguientes funciones en el código del microcontrolador:

- *Reset_TSL*

  Función aplicada para la inicialización de la lógica interna del sensor. Envia la secuencia apropiada de ciclos de reloj para asegurar la correcta detección del bit de Start, y para asegurar el estado lógico correcto del pin SDOUT (MISO).

```
void Reset_TSL (void){
        TSCLK = 0;

        MOSI = 0;              //Pulsar SCLK 30 veces con MOSI en 0
        Send_Clocks(30);       //para garantizar estado de SDOUT (MISO)

        MOSI = 1;              //Pulsar SCLK 10 veces con MOSI = 1 para
        Send_Clocks(10);       //garantizar detección de start

        Send_TSL (0x1B);       //Enviar comando RESET
        Send_Clocks(5);        //Pulsar SCLK 5 veces
        Send_TSL (0x5F);       //Comando escribir registro de modo
        Send_TSL (0x00);       //Registro de modo = 0x00 (modo normal)
}
```

- *Send_Clocks*

  Utilidad para transmisión sucesiva de pulsos de reloj, utilizada en otras funciones de interfaz con sensor.

```
void Send_Clocks (char veces){
        while(veces>0){
                TSCLK = 1;
                TSCLK = 0;
                veces--;
        }
}
```

- *Send_TSL*
  Envia el byte deseado al sensor, agregando los bits de Start y Stop requeridos y controlando el clock como requiere la interfaz isosíncrona.

```
void Send_TSL (unsigned char byte){
        unsigned char varb;
        varb = byte;
        MOSI = 0;                         //Enviar start bit
        TSCLK = 1;      TSCLK = 0;        //Pulsar SCLK
        //Enviar 8 bits
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        if((varb%2))    MOSI  = 1;
        else            MOSI = 0;
        TSCLK = 1;      TSCLK = 0;        varb = varb/2;
        MOSI = 1;                         //Enviar stop bit
        TSCLK = 1;      TSCLK = 0;        //Pulsar SCLK
}
```

- *Rcv_TSL*
  Recibe un byte del sensor, descartando los bits de Start y Stop, y controlando el clock de acuerdo con los requerimientos del sensor.

```
unsigned char Rcv_TSL(void){
        unsigned char value;
        value = 0;
        TSCLK = 1;              TSCLK = 0;      //Pasar start bit - pulsar SCLK
        //Recibir 8 bits
        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;

        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;

        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;
        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;
        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;
        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;
        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;              value/=2;
        if(MISO)
                value|=128;
        TSCLK = 1;              TSCLK = 0;
        TSCLK = 1;              TSCLK = 0;      //Pasar Stop bit - pulsar SCLK
        return value;
}
```

Cabe destacar que, para optimizar el tiempo de lectura de la línea de 96 pixels, se evitó entre otras cosas invocar esta función 96 veces. En lugar de eso, se incluyó su código con ligeras modificaciones dentro de la rutina de interrupción. De todas formas se hace uso de esta función en otras partes del código, para el seteo de la ganancia y offset, y dentro de la función Reset_TSL().

2.9.5 Timer 0 – Muestreo uniforme

El sistema está programado de modo que el microcontrolador pueda muestrear las imágenes desde el sensor a una tasa máxima de 500 veces por segundo. Con esto en mente, y asumiendo que el usuario va a desplazar el lector a una velocidad relativamente constante, se decide muestrear el sensor a una velocidad constante, de 500 muestras por segundo. Esto se logra mediante la utilización de uno de los timers del PIC, el timer0. Este es un timer de 16 bits capaz de funcionar con interrupciones (salir del funcionamiento del lazo principal del sistema para muestrear el sensor cada vez que termine de contar, para después regresar al lazo principal). Dispone de un prescaler programable para reducir la frecuencia del reloj que controla el timer, y llevar la cuenta a los valores deseados.

El timer 0 se programó en esta aplicación para trabajar en modo 16 bits, con una velocidad de entrada de $F_{osc}/4$ (en modo PLL enabled, y con un cristal externo de 10MHz, $F_{osc}/4$ = 10MHz) y con el prescaler programado en dividir por 8.

El timer trabaja con 2 registros de 8 bits, formando combinados un registro de 16 bits, cuyo valor se incrementa en uno cada vez que la fuente de reloj de entrada (a través del prescaler y sincronizada con la lógica interna del timer) lo indica. Cuando el registro pasa de la cuenta 0xFFFF (65535) a 0x0000, el timer genera la condición de interrupción que se mencionó anteriormente.

Para que el timer tarde 2ms en terminar de contar, es necesario que cuente 2500 veces, de acuerdo con la siguiente fórmula:

$((F_{osc}/4) / 8) / \text{cuenta} = 1 / 2\text{ms} =$ **500Hz**
((10MHz / 4) / 8) / cuenta = 500Hz
cuenta = 2500

$F_{osc}/4$	10MHz
8	Prescaler
500 Hz	Frecuencia de muestreo buscada

Entonces se deberá cargar el valor (65536 – 2500) en el registro del timer antes de activarlo, para que tarde 2ms en en interrumpir. El valor en hexadecimal (como se usa en el código) es 0xF63C: Se carga TMR0H = 0xF6
TMR0L = 0x3C

Cuando el usuario presiona el botón, se leerá como mínimo cada 2ms una muestra desde el sensor como parte de la rutina de interrupción. En el tiempo que queda para correr el lazo principal, el PIC transmite y guarda las muestras previas en la memoria EEPROM, como se enuncia más adelante.

Como resultado de pruebas, se obtuvo un tiempo mínimo de cuenta del timer de 2ms en el cual se pueda ejecutar el conjunto de las operaciones necesarias para el sistema, sin que se pierdan muestras .

2.9.6 Corrección de offset

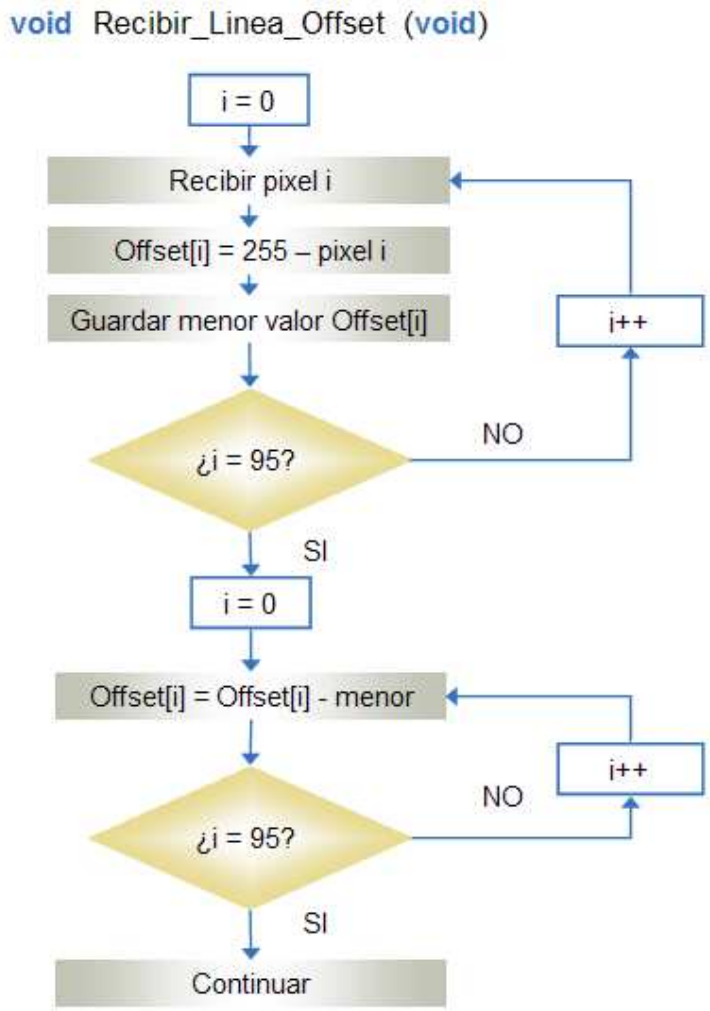
La estructura lograda para el lector tiene las condiciones para ofrecer imágenes con foco y contraste razonables. Si bien estos parámetros se mantienen uniformes a lo largo de la línea medida, existe una variación en la intensidad con la que la luz alcanza al receptor en distintos puntos, causada por la estructura de la lente utilizada, que no fue exactamente diseñada para ser utilizada como parte de este lector (se entiende que la separación entre las “microlentes” es el factor que causa este efecto).

El efecto principal de esta incompatibilidad consiste en una variación del nivel de offset que tiene la imagen a lo largo de la extensión de la línea, como se muestra a continuación:

Resultado de medición sin corrección de offset, como se observa en programa de prueba sobre LabView

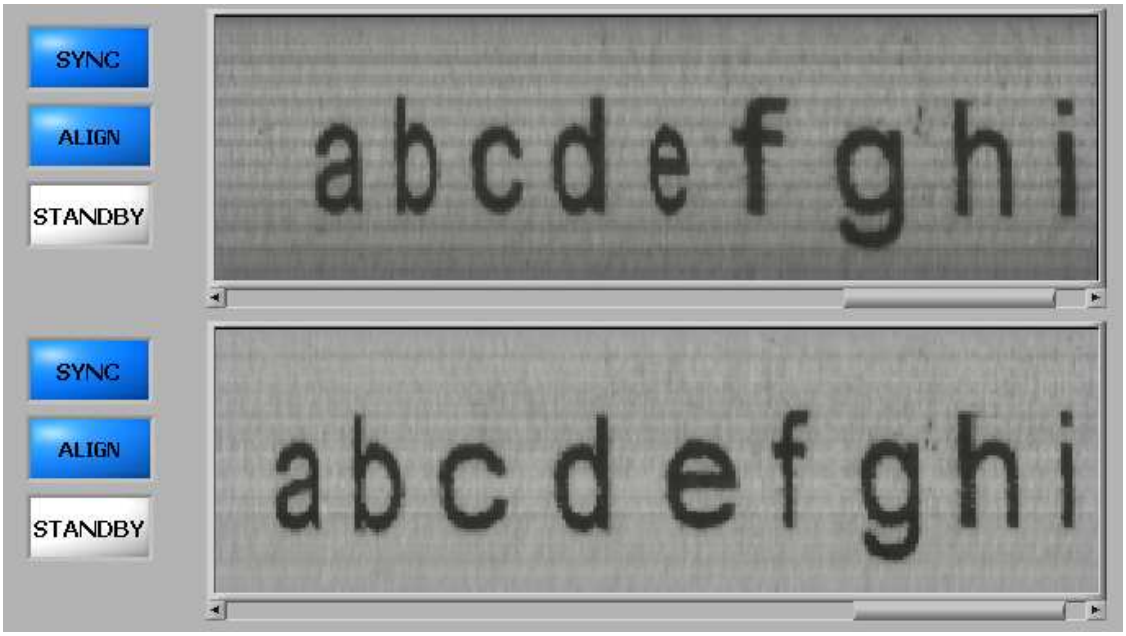
Como se entiende que las condiciones de foco y contraste se mantienen relativamente uniformes a lo largo de la línea medida, basta con hacer una corrección de offset que se aplique individualmente para cada pixel, para superar esta dificultad. Para lograr esto se creó una función que calcula el nivel de offset para cada pixel luego del encendido del dispositivo, al ubicar al lector sobre una línea en blanco. Este nivel se resta en las siguientes mediciones, para corregir el efecto de esta variación en la iluminación.

En este algoritmo, **menor_off** guarda el valor del pixel más iluminado de la línea, de modo que **offset[i]** guarda la diferencia con este valor en una línea blanca. La estructura lógica de la función de corrección es la siguiente:



A partir del cálculo de los valores de offset, basta con restarlos al valor de pixel que le corresponda para corregir la variación en el nivel de luz. Teniendo en cuenta que el sistema está hecho para medir imágenes en blanco y negro, se agregó una condición para hacer esta corrección de offset, que determina que sólo se reste el valor del offset a los pixels que sean lo suficientemente claros. De este modo, cuando un pixel es lo suficientemente oscuro, se asume que pertenece a un trazo, y por tanto no se le resta el offset, con lo que se mejora el contraste final de los caracteres.

El resultado de la aplicación de este algoritmo se muestra en las siguientes imágenes:



Efecto de corrección de offset sobre los datos enviados por el sensor lineal

En el prototipo, esta función se ejecuta sólo la primera vez que el usuario aprieta el botón, por lo que se requiere que esta primera lectura comience en una región en blanco de la superficie que se quiere explorar, si se quiere como una prueba rápida inicial. El código C de la función que se muestra en el diagrama en bloques se muestra a continuación:

```
void Recibir_linea_offset(void){
        //Recibir 96 pixels y analizar
        for(i=0;i<96;i++){
                pixels = 255-Rcv_TSL();        //0 - blanco / 255 - negro
                offset_ln[i] = pixels;
                if(pixels<menor_off)
                        menor_off = pixels;
        }
        //Descartar ultimos pixels
        Send_TSL(0x19);                //Enviar comando ABORTPixel
        //Determinar offset
        for(i=0; i<96; i++)
                offset_ln[i] -= menor_off;
}
```

Las variables que se utilizan en esta función son globales, y se declaran en el programa completo

2.9.7 Funciones de interfaz I^2C – Módulo MSSP I^2C Master

La memoria EEPROM utilizada (24LC256, de Microchip) es una memoria de 256kbits (32768 bytes), de modo que se requieren 15 bits para direccionar cada byte en la memoria. Las funciones de C18 para control de memorias EEPROM están diseñadas para operar con memorias con campos de dirección de 8 bits, de modo que no sirven para controlar la memoria seleccionada vía I^2C.

Como consecuencia de esta dificultad, y para facilitar la implementación por hardware de memorias EEPROM de este tipo, el fabricante ofrece una nota de aplicación que trata el modo de operación de la memoria seleccionada. Junto con esta nota, se ofrece una librería con funciones para usar en lugar de las originales, y que pueden operar con memorias EEPROM con hasta 2 bytes de dirección.

En este proyecto se recurrió a estas funciones para permitir utilizar el módulo I^2C del microcontrolador. Estas funciones operan sin recurrir a las interrupciones del microcontrolador, cosa que evita conflictos y las deja libres de ser utilizarlas para otros propósitos (por ejemplo, indicar cuándo leer el sensor). La librería especial incorporada hace uso a su vez de las funciones de la librería básica de manejo de i2c del C18.

Para este programa se utilizaron las funciones de *page write*, *acknowledge polling* y *lectura secuencial* desde la memoria.

Control de la memoria en el programa

El módulo MSSP del microcontrolador se configura para operar en modo I^2C maestro (la memoria nunca envia comandos). Se activa el control de slew rate para operación en 400kHz (I^2C High Speed Mode), y se designa el valor del registro SSPADD para operar a esa velocidad, de acuerdo con la ecuación:

$$\textbf{SCL Rate} = F_{osc} / (4 (\textbf{SSPADD} + 1))$$

Con $\mathbf{F_{osc} = 40MHz}$, y **SCL Rate = 400kHz**, se tiene **SSPADD = 24 = 0x18**

```
//Inicializacion I2C
OpenI2C(MASTER, SLEW_ON);     //Modulo I2C Master slew rate para 400kHz
SSPADD = 0x18;                //SPADD = (FOSC/Brate)/4 – 1
```

Como las páginas de la memoria tienen 64 bytes, y la lectura de una línea de 96 pixels en blanco y negro ocupa 12 bytes (96 bits), se pueden guardar fácilmente 5 líneas en cada página. El umbral para determinar si el valor del bit es blanco o negro es definido por el usuario en la interfaz del programa final.

- Cuando se explora texto, el programa lee 5 líneas y las guarda en la memoria con un **Page Write**. Esta función guarda el vector **PageString[]** en la página de la memoria que comienza en la posición **Address**:

  ```
  HDPageWriteI2C ( ControlByte, Address/256, Address%256, PageString );
  ```

- Mientras se escribe, se reciben las siguientes 5 líneas como parte de la rutina de interrupción invocada cuando el timer termina de contar, de modo de aprovechar el tiempo y lograr lecturas uniformemente espaciadas.
- Con la función de **Acknowledge Polling** se evita que se inicien ciclos de escritura antes de lo debido:

  ```
  EEAckPolling ( ControlByte );
  ```

- Una vez terminado el ciclo de escritura, se está listo para escribir las siguientes 5 líneas en la memoria y leer otras 5 líneas.

Cuando se termina de explorar texto (se suelta el botón), se ejecuta una **lectura secuencial** de 12 bytes (una lectura del sensor). Cada línea recibida se transmite a la computadora, y se procesa en el microcontrolador para detectar los límites de los caracteres.

```
HDSequentialReadI2C (ControlByte, Address_Read/256, Address_Read%256, MemRead, 12);
```

Con este comando se guarda en el vector **MemRead[]** los 12 bytes que se leen desde la memoria EEPROM, comenzando en la posición **Address_Read**.

En todas estas funciones el byte de control determina la dirección de la memoria (ControlByte = **b'10100000'**). Su último bit (R/W) se modifica dentro de la función de lectura según sea necesario.

2.9.8 Funciones de segmentación de caracteres

En este documento se entiende a la segmentación de caracteres como el proceso por el que se encuentra y delimita cada símbolo que aparece en la imagen registrada.

Si bien la utilidad de reconocimiento óptico de caracteres del LabView es capaz de realizar la segmentación de caracteres en forma independiente, se eligió incorporar una función con este propósito en manos del microcontrolador por motivos que se explicarán más adelante.

Los principios que se asumieron para desarrollar un algoritmo de segmentación para esta aplicación fueron los siguientes:

- En una imagen no degradada, los caracteres impresos se pueden distinguir porque existe una separación física entre ellos
- Cada símbolo se carcteriza por la continuidad de los trazos que lo conforman, salvo en el caso de la ñ, la i y la ü y las vocales con tilde
- Para los casos excluidos en el punto anterior, los trazos que no forman parte del cuerpo principal del característica se encuentran directamente por encima de éste

En base a estos principios, se decidió realizar un algoritmo que detecte los límites de los cuerpos principales de los caracteres, buscando la separación visible entre estos.

Una vez distinguido el cuerpo del sìmbolo, bastará con extender ligeramente el límite superior por sobre el límite calculado para que la región designada incluya el trazo separado para los caracteres discontinuos. Como el módulo de reconocimiento del LabView es capaz de reconocer estos caracteres discontinuos junto con los otros, este método soluciona casi todos los casos de discontinuidad, de modo que se podrá tratar a todos los caracteres de igual forma (bajo el mismo algoritmo).

Ejemplo de solución para caracteres discontinuos

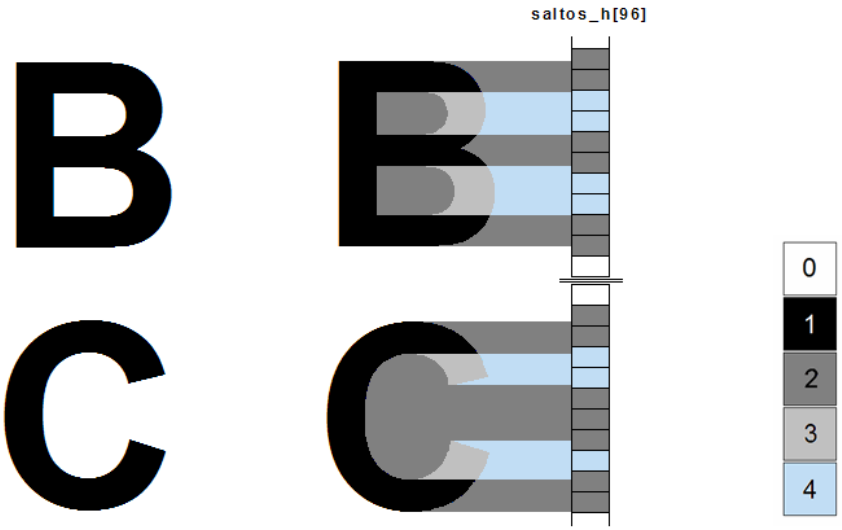
Las dos versiones de la función de segmentación que se mencionarán se relacionan con la forma de lectura que se tiene con el sensor lineal. En ambas se analiza cada línea vertical en el orden en que llega, y se actualiza el estado de la segmentación con cada línea recibida.

Cuando se detecta el fin de un símbolo, el estado de la segmentación pasa a ser OK_CHAR. En este punto, el microcontrolador tiene que calcular y transmitir a la computadora la **región de interés (ROI)**, que consiste en los 4 límites que delimitan la región de la imagen sobre la cual ejecutar OCR.

2.9.8.1 Función inicial de segmentación – Limitaciones

La primera idea que se implementó para la segmentación de caracteres consistía en contar el número de "saltos" de blanco a negro o de negro a blanco a lo largo de las líneas verticales registradas. Para 96 pixels, se usa un vector de 96 bytes **saltos_h[96]**, que registra el número de saltos de color por los que pasó cada fila de la imagen.

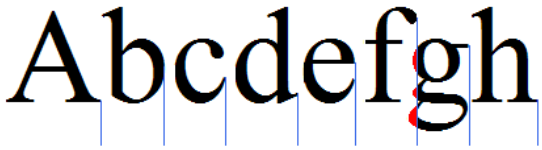
La lógica con la que se completa ese vector se muestra en la siguiente imagen:



Lógica para función inicial de segmentación

En esta imagen puede observarse que, cuando se termina un símbolo, *todos los valores del vector de saltos son pares dentro de sus límites y valen cero en sus bordes*, donde no se pasó por ningún trazo Es por esto que se pensó un algoritmo que detecte un bloque par rodeado por ceros para identificar el fin de un símbolo.

Este algoritmo se probó inicialmente, cuando se desarrollaba el programa en LabView, para evaluar el desempeño del módulo de reconocimiento en la computadora. Sin embargo, esta función no está preparada para trabajar con caracteres que se superponen, como se muestra en la imagen a continuación:



Ejemplos de superposición de caracteres

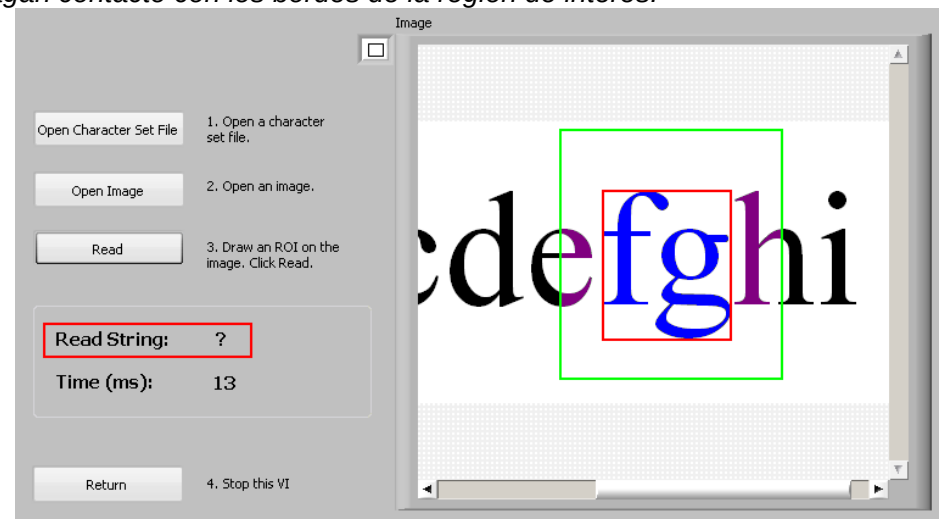
Este fenómeno se observaría ocasionalmente si se piensa en fuentes con estilo normal, pero se vuelve más probable si se piensa que el usuario no siempre va a ubicar el lector en ángulo recto respecto de la línea del texto, y se vuelve aún más común cuando la fuente está en *itálica*.

Por otro lado, deberá tenerse en cuenta que, para cualquier función de segmentación que se utilice, se necesitará realizar modificaciones adicionales para operar cuando se observan partes de dos o más renglones al mismo tiempo.

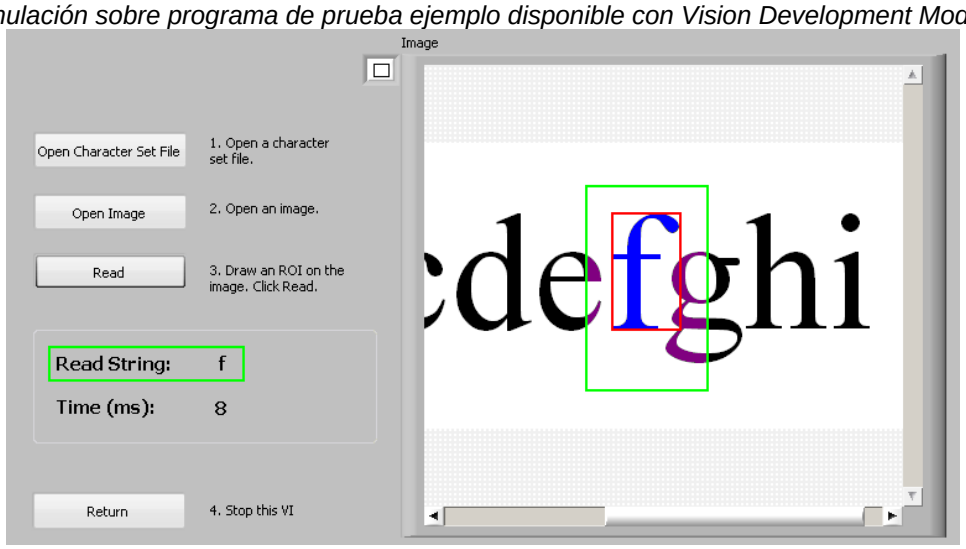
2.9.8.2 Segmentación en LabView - Función conjunta

Como ya se mencionó, el módulo de OCR del LabView es capaz de reconocer igualmente los caracteres continuos y discontinuos, mientras la parte discontinua del símbolo se encuentre por encima de este.

Por otro lado, si bien el módulo no puede separar caracteres si no existe una separación vertical entre ellos, es valiosa la opción que existe para *ignorar los trazos que hagan contacto con los bordes de la región de interés:*



a) La utilidad de OCR no es capaz de distinguir caracteres que se superponen
Simulación sobre programa de prueba ejemplo disponible con Vision Development Module



b) La utilidad de OCR puede ignorar los trazos que tocan la ROI (cuadro verde)
Simulación sobre programa de prueba ejemplo disponible con Vision Development Module

En estas imágenes se observa que el módulo de OCR del LabView (utilizado de esta forma) tiene la misma limitación que el algoritmo de segmentación que se propuso para el microcontrolador. Si se utilizara ese algoritmo de segmentación, el sistema conjunto tendría esa limitación, y no se podría reconocer caracteres superpuestos.

Sin embargo, en este punto se sabe que:

- El módulo de OCR en la PC puede hacer procesamiento de imágenes de mayor nivel que el microcontrolador, y a más alta velocidad (puede ignorar trazos que no interesan dentro de la ROI)
- El microcontrolador podría hacer lo que el módulo en la computadora no hace (identificar límites de caracteres, aún cuando se superponen)

Es por esto que se decidió utilizar un **modo de operación conjunta** como el antes mencionado, con el microcontrolador adaptado al módulo de OCR, de modo que ambos sirvan para cumplir partes de una segmentación en conjunto más robusta. Para esto es necesario desarrollar un nuevo algoritmo de segmentación en el microcontrolador, más robusto y complejo.

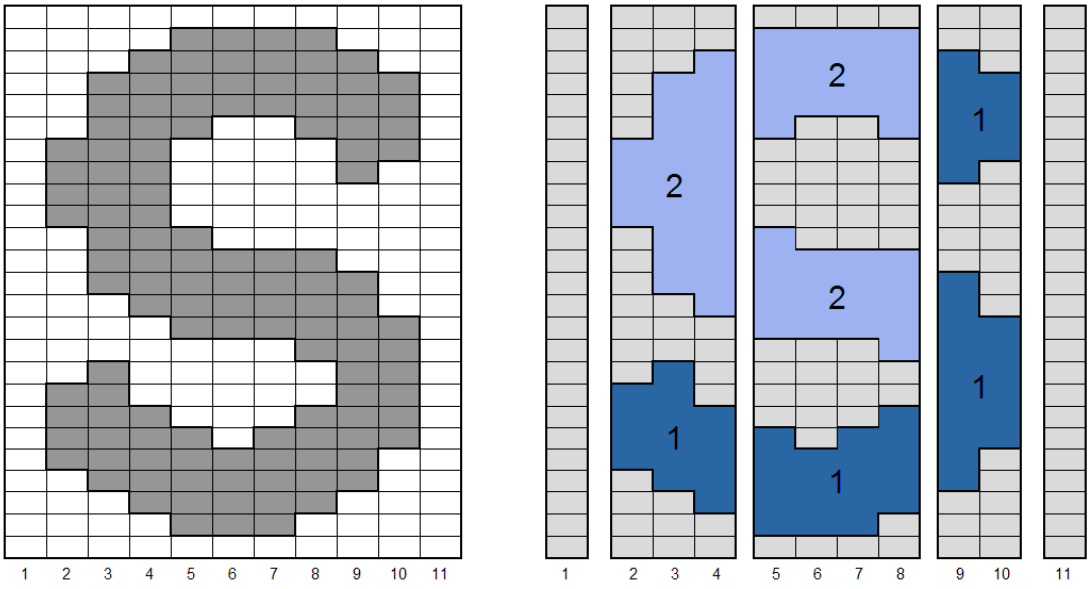
2.9.8.3 Función final de segmentación

El segundo algoritmo de segmentación que se diseñó para esta aplicación maneja la misma información que el anterior: se analiza cada línea de entrada a medida que lee desde el sensor (o que se recibe de la memoria). Lo que hace con esa información recibida es lo que cambia. Se busca que este algoritmo detecte y guarde cada trazo según aparece en pantalla, y que asigne a cada trazo un identificador que indique a qué símbolo corresponde. En esta aplicación, se considera como trazo a una sección de bits adyacentes de valor 1 (negro).

Será necesario que este algoritmo sea capaz de detectar cuatro casos posibles:

- Aparece un trazo nuevo
- Desaparece un trazo
- Un trazo se separa en dos o más
- Dos o más trazos de juntan

Para poder decidir con qué caso se está tratando, se debe analizar tanto la línea de bits actual como la inmediata anterior. Comparando la línea actual con la anterior se puede decidir cuál de los casos anteriores ocurre, como se muestra en la siguiente imagen, que ejemplifica todos los casos posibles:



Línea 1	No se detecta nada
Línea 2 – 4	Se detectan dos trazos separados, se asigna a cada uno un ID de símbolo distinto
Línea 5 – 8	El trazo con ID 2 se separa en dos, se asigna a cada trazo separado el ID correspondiente. El trazo con ID 1 sigue presente
Líneas 9 – 10	El trazo con ID 1 se junta con uno de los trazos con ID 2, y por lo tanto ambos pertenecen al mismo sñimbolo. Se asigna a todos los trazos el ID más bajo (ID 1)
Línea 11	Desaparecen todos los trazos de un símbolo – OK_CHAR

Lógica para algoritmo de segmentación de caracteres con detección de trazos

Si bien la lógica del algoritmo está definida, es necesario determinar la forma de llevar este algoritmo básico al código del microcontrolador.

En el microcontrolador se analiza la línea de 96 bits actual y la previa, y a partir de estas se determina un conjunto de variables sobre las que se trabaja para cumplir con la lógica que se mencionó. De todas estas, las más importantes son las variables que representan los trazos presentes en cada línea.

Trazos_Actuales[25][5]

Se incluye en este vector una entrada para cada trazo detectado en la línea actual
Cada entrada incluye 5 bytes:

Inicio	Orden del bit en el que inicia el trazo (0 a 95)
Fin	Orden del bit en el que termina el trazo (0 a 95)
ID	Identifica el símbolo al que pertenece el trazo
MIN	Valor mínimo que alcanzó el símbolo ID
MAX	Valor máximo que alcanzó el símbolo ID

Trazos_Previos[25][5]

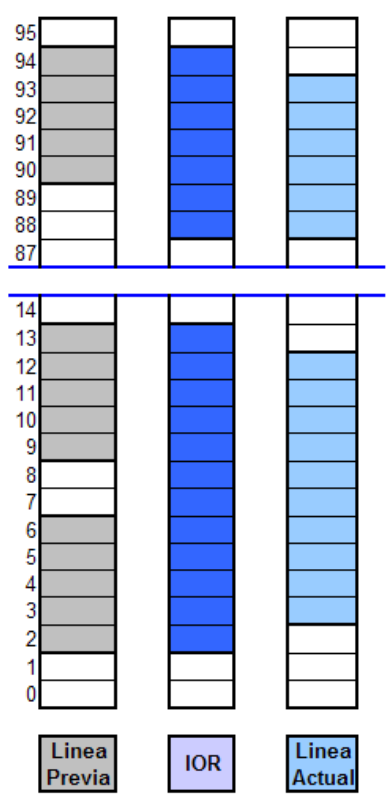
Es el vector Tazos_Actuales[25][5] de la pasada anterior
Cada entrada incluye 5 bytes:

Inicio	Orden del bit en el que inicia el trazo (0 a 95)
Fin	Orden del bit en el que termina el trazo (0 a 95)
ID	Identifica el símbolo al que pertenece el trazo
MIN	Valor mínimo que alcanzó el símbolo ID
MAX	Valor máximo que alcanzó el símbolo ID

Trazos_IOR[50][2]

Hay una entrada para cada trazo detectado en la combinación de la línea actual y la previa (IOR lógico entre fila anterior y actual) – *Ver diagrama*
Cada entrada incluye 2 bytes:

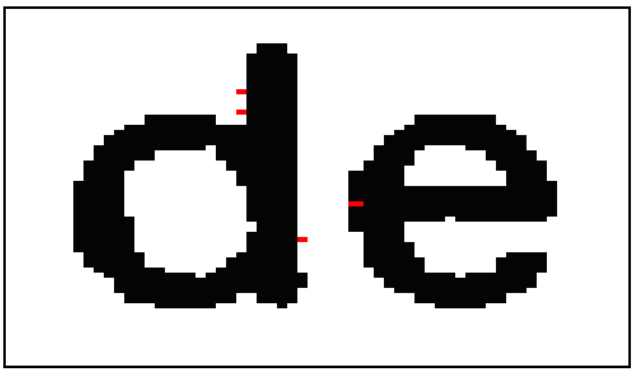
Inicio	Orden del bit en el que inicia el trazo (0 a 95)
Fin	Orden del bit en el que termina el trazo (0 a 95)



Variables para detección de trazos

Los vectores de trazos actuales y previos están preparados para registrar un máximo de 25 trazos, y, por coherencia Trazos_IOR está preparado para recibir la suma de los trazos de los otros dos vectores (50 trazos). Se definió este valor máximo de trazos a detectar pensando en las condiciones del sistema en general, y de la implementación lograda:

- En general, la sección vertical de un símbolo tiene un máximo de 4 trazos (**g**)
- Sin embargo, la medición con el sensor lineal no es perfecta, como se muestra en la imagen, por lo que a modo de contingencia se aumentó el número máximo de trazos para contemplar “ramificaciones” de los trazos



La degradación de la imagen causa aparición ocasional de más trazos

La lógica básica de la nueva función de segmentación, como se implementa en el microcontrolador, se muestra en el siguiente diagrama en bloques:

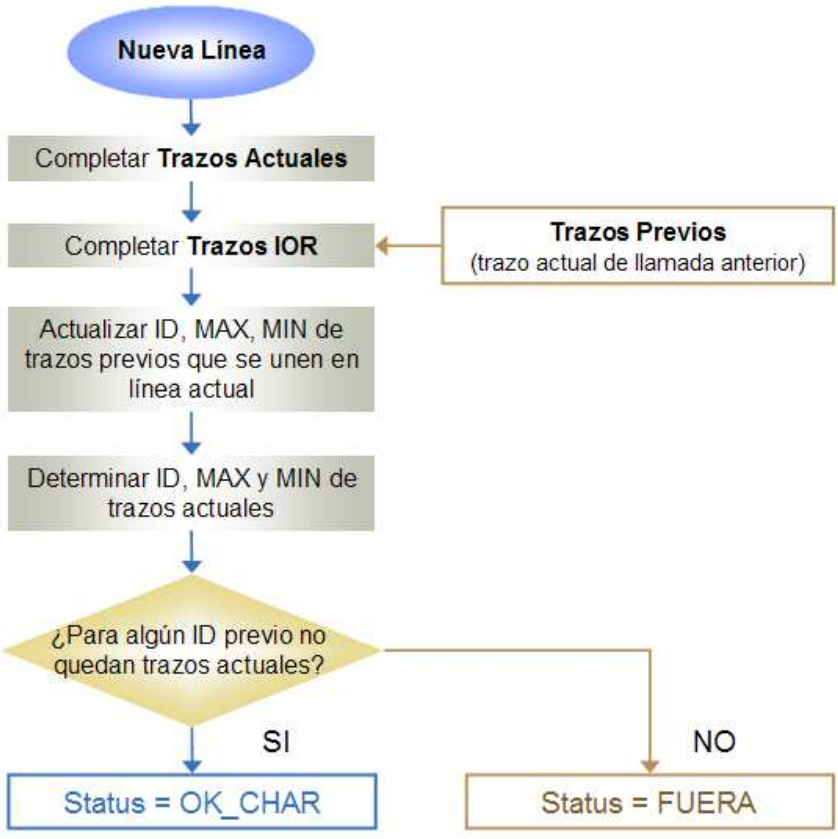


Diagrama en bloques para nueva versión de función de segmentación

A continuación se incluye el código para el microcontrolador de la función de segmentación con reconocimiento de trazos.

```
void Set_status_v(void){                //Segmentación de caracteres
volatile unsigned char Trazos_IOR[50][2];
volatile unsigned char ID_IOR_MIN, ID_IOR_MAX;
volatile unsigned char IOR_MAX_P, IOR_MIN_P;
volatile unsigned char inicio_temp, fin_temp, status_v_IOR, status_v_ACTUAL;

status_v = status_v_ACTUAL = status_v_IOR = FUERA;
max_trazos_P = max_trazos_A;            //Desplazar numero de trazos anteriores
max_trazos_A = max_trazos_IOR = 0;   //Reiniciar numero de trazos actuales

//Crear Trazos Actuales
for(i=0;i<97;i++){
        if((propix[i]&0x01)==1){        //Si el pixel actual es 1
                if(status_v_ACTUAL==FUERA){
                //Si empieza a ver un trazo o esta terminando la zona de interés
                        status_v_ACTUAL=DENTRO;
                        inicio_temp=i;
                }
        }
        else{                           //Si el pixel actual es 0
                if(status_v_ACTUAL==DENTRO){
                //Si termino de ver un trazo, registrarlo
                        status_v_ACTUAL=FUERA;
                        fin_temp=i-1;
                        Trazos[actual][max_trazos_A][INICIO] = inicio_temp;
                        Trazos[actual][max_trazos_A][FIN] = fin_temp;
                        max_trazos_A++;
                }
        }
}
//Crear Trazos IOR = (Actual[i])IOR(Previo[i])
for(i=0;i<97;i++){
        if(propix[i]!=0){               //Si el estado actual o el previo es 1
                if(status_v_IOR==FUERA){
                //Si empieza a ver un trazo o esta terminando la zona de interés
                        status_v_IOR=DENTRO;
                        inicio_temp=i;
                }
        }
        else{                           //Si el estado actual y el previo son 0
                if(status_v_IOR==DENTRO){
                //Si termino de ver un trazo, registrarlo
                        status_v_IOR=FUERA;
                        fin_temp=i-1;
                        Trazos_IOR[max_trazos_IOR][INICIO] = inicio_temp;
                        Trazos_IOR[max_trazos_IOR][FIN] = fin_temp;
                        max_trazos_IOR++;
                }
        }
}
```

```
//Para cada trazo IOR, analizar trazos Previos e identificar ID conjunto
//(si se juntan trazos, juntar ID previo)
for(i=0; i<max_trazos_IOR; i++){
        inicio_temp = Trazos_IOR[i][INICIO];
        fin_temp = Trazos_IOR[i][FIN];
        for(m=0;m<30;m++)
                ID_stat[m]=0;
        ID_IOR_MIN=255;
        coincidió = 0;
        for(m=0;(m<max_trazos_P); m++){
                if((Trazos[previo][m][INICIO]<=fin_temp)
                   && (Trazos[previo][m][FIN]>=inicio_temp)){
                //Si el trazo previo coincide en algún punto con el IOR
                        coincidió++;
                        //Actualizar ID_IOR_MIN y ID_IOR_MAX
                        ID_stat[(Trazos[previo][m][ID])]=1;
                        if((Trazos[previo][m][ID])<ID_IOR_MIN)
                                ID_IOR_MIN = Trazos[previo][m][ID];
                        //Actualizar limites maximo y minimo previo + IOR para
                        ////cohesion de característica
                        if(Trazos[previo][m][MIN]>inicio_temp)
                                Trazos[previo][m][MIN]=inicio_temp;
                        if(Trazos[previo][m][MAX]<fin_temp)
                                Trazos[previo][m][MAX]=fin_temp;
                }
        }

        if(coincidió>1){
                IOR_MIN_P = Trazos_IOR[i][INICIO];
                IOR_MAX_P = Trazos_IOR[i][FIN];
                //Para los trazos que coincidieron, actualizar ID al minimo valor
                for(m=0; m<max_trazos_P; m++){
                        if((ID_stat[(Trazos[previo][m][ID])])==1){
                        //Si el trazo coincidió con el trazo IOR
                        if(Len_ID[ID_IOR_MIN]<Len_ID[Trazos[previo][m][ID]])
                                Len_ID[ID_IOR_MIN] = Len_ID[Trazos[previo][m][ID]];
                                //Actualizar ID a ID_IOR_MIN
                                (Trazos[previo][m][ID])=ID_IOR_MIN;
                                //Medir IOR_MIN_P e IOR_MAX_P
                                if(Trazos[previo][m][MIN]<IOR_MIN_P)
                                        IOR_MIN_P = Trazos[previo][m][MIN];
                                if(Trazos[previo][m][MAX]>IOR_MAX_P)
                                        IOR_MAX_P = Trazos[previo][m][MAX];
                        }
                }
```

```
                //Asignar los valores MIN y MAX a los Trazos_P que coincidieron
                for(m=0; m<max_trazos_P; m++){
                        if(Trazos[previo][m][ID]==ID_IOR_MIN){
                        //Si el trazo coincidió con el trazo IOR
                                //Actualizar MIN y MAX
                                Trazos[previo][m][MIN]=IOR_MIN_P;
                                Trazos[previo][m][MAX]=IOR_MAX_P;
                        }
                }
        }
}

//Reiniciar valores de cuanta_ID
for(i=1;i<30;i++)
        cuenta_ID[i]=0;

//Comparar trazos actuales con trazos previos
//Asignar ID nuevos y cuenta_ID
for(i=0; i<max_trazos_A; i++){
        fin_temp = Trazos[actual][i][FIN];
        inicio_temp = Trazos[actual][i][INICIO];
        coincidió=0;

        for(m=0;(m<max_trazos_P) && (coincidió==0); m++){
                //Si el trazo coincide en elgun punto con el actual
                if((Trazos[previo][m][INICIO]<=fin_temp)
                    && (Trazos[previo][m][FIN]>=inicio_temp)){
                        coincidió=1;
                        Trazos[actual][i][ID] = Trazos[previo][m][ID];
                        Trazos[actual][i][MIN] = Trazos[previo][m][MIN];
                        Trazos[actual][i][MAX] = Trazos[previo][m][MAX];
                        cuenta_ID[(Trazos[actual][i][ID])]++;
                }
        }

        //si no coincidió con ninguno, nuevo trazo con nuevo ID, MAX y MIN
        if(coincidió == 0){
                Trazos[actual][i][ID] = nuevo_ID;
                Trazos[actual][i][MIN] = Trazos[actual][i][INICIO];
                Trazos[actual][i][MAX] = Trazos[actual][i][FIN];
                cuenta_ID[nuevo_ID]=1;
                Len_ID[nuevo_ID]=0;
                nuevo_ID++;
        }
}
```

```
//Evaluar fin de símbolo - si para algun ID previo queda cuenta_ID, entonces
for(i=0; (i<max_trazos_P) && (status_v!=OK_CHAR); i++){
if((cuenta_ID[(Trazos[previo][i][ID])])==0){
        upper_limit=Trazos[previo][i][MAX];
        lower_limit=Trazos[previo][i][MIN];
        toph64= Len_ID[Trazos[previo][i][ID]]+2;
        if(     (((upper_limit+lower_limit)/2)>=renglon_inf)
                &&(((upper_limit+lower_limit)/2)<=renglon_sup)    ){
                //si el simbolo esta en el mismo renglon que el anterior
            if(     ((upper_limit-lower_limit)>=MIN_SIZE_CHAR)
                &&((upper_limit-lower_limit)<95)){
                //y es lo suficientemente grande
                //y no hubo condiciòn de error inicial
                status_v = OK_CHAR;     //Entonces se termino un caracter

                //Calcular ROI
                  //Limite superior
                if((upper_limit+0.6*(upper_limit-lower_limit))<=96)
              ROI1.temp = 5 + (96-(upper_limit+0.6*(upper_limit-lower_limit)))*scale_height;
                else
                        ROI1.temp = 5;
                ROI1.Top = ROI1.temp;
                        renglon_sup = upper_limit;
                  //Limite inferior
                ROI1.temp = 5 + (96-(lower_limit-3))*scale_height;
                ROI1.Bottom = ROI1.temp;
                if(ROI1.Bottom>=140)
                        ROI1.Bottom = 140;
                renglon_inf = lower_limit;
                  //Limite izquierdo
                if(toph64>56)
                        toph64=56;
                ROI1.temp = (Len_IMG +4)- (toph64+4)*scale_len;
                ROI1.Left = ROI1.temp;
                if(ROI1.Left<=5)
                        ROI1.Left = 5;
                  //Limite derecho siempre es el borde de la imagen

                //Averiguar si hubo espacio en blanco
                font_height = renglon_sup-renglon_inf;
                imprimir_caracter=0;
                if(totalh>toph64){
                        espacio = totalh-(toph64);
                //La diferencia es el espacio entre el caracter anterior y el actual
                        space_ratio = font_height/espacio;
                        if(((space_ratio)<3.5)) { //Si se midiò un espacio, imprimirlo
                                imprimir_caracter=1;
                                caracter = ' ';
                        }
                }
        }
}
```

```
		//Reconocer puntos y comas
		if(((upper_limit-lower_limit)<MIN_SIZE_CHAR)
			&&(lower_limit!=0)
			&&(status_v!=OK_CHAR)
			&&((upper_limit-lower_limit)>3)
			&&((upper_limit)>=renglon_inf)
			&&((lower_limit)<=renglon_sup)){
			//Si se vio un caracter chico en el rango del ultimo
			imprimir_caracter=0;
			if(upper_limit < (renglon_inf*1.3)){	//Si el punto està en la parte inferior
				imprimir_caracter=1;
				font_height = upper_limit-lower_limit;
				if((font_height/toph64<=0.7))
					//si el alto es parecido al ancho, es un punto
					caracter = '.';
				else	//Si es màs alto que ancho es una coma
					caracter = ',';
			}
		}

}
}
//Retraer IDs en 0
for(i=1; i<nuevo_ID; i++){		//Para cada ID
		if(cuenta_ID[i]==0){		//Si no quedan trazos con ese ID

			for(m=0;m<max_trazos_A;m++){
				if(Trazos[actual][m][ID]>i)
					Trazos[actual][m][ID]--;
			}
			for(m=i;m<=nuevo_ID;m++){
			//Retraer cuenta_ID para la proxima
				cuenta_ID[m]=cuenta_ID[m+1];
				Len_ID[m]=Len_ID[m+1];
			}
			cuenta_ID[m-1]=cuenta_ID[m]=0;	//Borrar la ultima
			Len_ID[m-1]=Len_ID[m]=0;		//Borrar la ultima
			nuevo_ID--;
			i--;					//Para que no saltee uno
		}
		else
			Len_ID[i]++;		//Si quedan trazos con ese ID, aumentar Longitud
}

//Toggle actual y previo
i = actual;
actual = previo;
previo = i;
}
```

Las variables que no se declaran en este código son variables globales, inicializadas en el cuerpo principal del programa, que se muestra más adelante.

Dado que se tiene que conservar el valor de los trazos actuales en una lectura para que sean los trazos previos de la siguiente llamada, se tendría que pasar para esto el conjunto de 125 bytes que hacen a la matriz de trazos actuales a la matriz de trazos previos. Este proceso consumiría tiempo, y en verdad es innecesario:

Para ahorrar tiempo, código y facilitar la lógica, se unió el vector de trazos actuales y el de trazos previos en la matriz Trazos **[2]** [25] [5], donde la primera dimensión distingue los trazos previos de los actuales. Basta con alternar el valor del índice “actual” (de 0 a 1, por ejemplo) por el “previo” (de 1 a 0, por ejemplo) para que, en la siguiente llamada, el bloque de memoria que antes representara a los trazos actuales ahora represente a los trazos previos. En la nueva llamada se llenará la memoria que corresponde a los trazos actuales con información de la nueva línea.

Esta función es capaz de reconocer texto cuando se observan múltiples renglones. Ante la posibilidad de que aparezcan caracteres sobre más de una línea, se optó por utilizar el siguiente método para diferenciarlos:

- Inicialmente, se aceptará el primer símbolo que se complete y que abarque la parte inferior del rango del visor (y que cumpla los otros requisitos de tamaño y longitud)
- A partir de ahí, los demás caracteres se aceptarán si se encuentran en una posición cercana a la del último símbolo aceptado

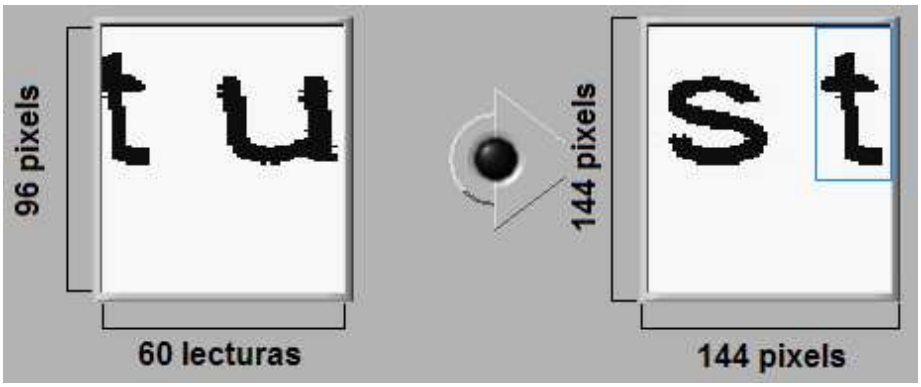
Esta condición se expresa al determinar los valores iniciales de **renglon_sup** y **renglon_inf** en 47 y 0 respectivamente, en el cuerpo principal del programa. Con este método, basta con que el usuario se ocupe de tratar de que la línea que le interesa reconocer se encuentre inicialmente en la parte inferior del visor para lograr que se reconozca esa línea deseada.

Esta condición es en verdad algo más flexible, ya que sólo se tiene que cumplir para el primer símbolo que se quiera leer, y pueden aparecer trazos por debajo de la línea de interés, mientras que no sean caracteres completos. Mientras no se encuentre demasiado elevada respecto del rango del visor, la línea deseada podrá ser identificada por el algoritmo.

2.9.9 Envio de ROI

Cuando encuentra el fin de un símbolo, el microcontrolador tiene que indicar dónde se lo puede encontrar en el LabView, para que se lo pueda interpretar. En la aplicación en LabView, las líneas recibidas del PIC se muestran en pantalla en gráficos ,y se toman imágenes de los gráficos cuando el PIC indica que terminó un símbolo.

Como se mencionó en la sección 2.7.3, es necesario hacer una conversión de unidades en el microcontrolador para que el conjunto funcione:



- El gráfico (izquierda) se forma con 60 líneas de 96 pixels de alto enviadas, esta es la información que se maneja en el PIC
- La imagen que se guarda en la computadora (derecha) tiene una resolución de 144x144 pixels (136x136 sin considerar los bordes)

La conversión se hace con una ecuación lineal simple, usando un multiplicador vertical y otro horizontal, y márgenes de seguridad para que los trazos del símbolo que interesa no toquen los limites de la ROI.

El límite superior se ubica aproximadamente un 70% por encima del límite del símbolo, para abarcar un posible trazo discontinuo para vocales con acento y la ñ, como se mencionó en la sección 2.8.7.

El límite derecho de la ROI siempre es el límite derecho de la imagen (sin contar la parte del borde del gráfico), porque siempre que se termina de ver un símbolo en el microcontrolador, éste está tocando el borde derecho de la imagen.

Los otros límites de la ROI se calculan de la siguiente forma:

Limite superior
5 + (96-(upper_limit+0.7*(upper_limit-lower_limit)))*scale_height;

Limite inferior
5 + (96-(lower_limit-3))*scale_height;

Limite izquierdo
(Len_IMG +4)- (toph+4)*scale_len;

scale_height es el factor de corrección vertical
upper_limit es el valor más alto entre 0 y 96 que alcanzó el símbol
lower_limit es el valor más bajo entre 0 y 96 que alcanzó el símbolo
toph es la longitud del símbolo
Len_IMG es el pixel donde termina la imagen copiada (sin contar el borde)

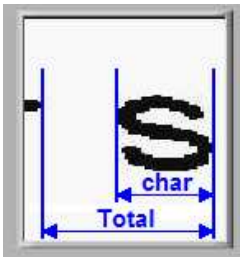
Recordar que se eligió esperar algunas líneas después de detectar el fin del símbolo antes de indicarlo al LabView, por eso se ve blanco antes del límite derecho de la ROI. Esto sirve como margen de seguridad.

2.9.10 Detección de espacios en blanco, puntos y comas[*]

Incorporar al sistema la capacidad de detectar espacios en blanco es imprescindible para que el texto reconocido pueda ser legible. Sumar a esto una detección de puntos y comas vuelve al sistema capaz de reproducir texto de forma coherente con la estructura del texto de origen.

La detección de los espacios que separan las palabras no puede realizarse directamente con el módulo de reconocimiento del LabView como se lo utiliza en esta aplicación. Sin embargo, el microcontrolador puede manejar la suficiente información para lograr una detección confiable de estos caracteres.

En el PIC se cuenta el número de muestras que pasaron desde que se terminó el último símbolo aceptado (**Total**), y para cada símbolo se cuenta su longitud en muestras (**char**).



Con esta información se puede inferir, cuando se termina un símbolo, si antes de este había o no un espacio en blanco. La longitud del espacio entre la letra anterior y la presente se puede obtener tomando la diferencia entre estos dos valores. Si el espacio es lo suficientemente grande, se asume que hubo un espacio en blanco, y en caso contrario no se hace nada.

Como se pretende que el sistema pueda trabajar con fuentes de tamaño variable, el umbral que se utiliza para decidir si se está tratando con un espacio o no no puede ser constante. Se debe utilizar para el umbral valor que esté vinculado con el tamaño de la fuente.

Si bien la altura de los caracteres no es constante para un tamaño de fuente (comparar la letra o con la O), su valor cambia con el tamaño de la fuente que se lee. Por otro lado, no se puede usar la longitud del símbolo como parámetro para el umbral, porque ese parámetro es aún más variable (comparar la letra i con la B). Para esta aplicación, se considera que la altura de los caracteres tiene un rango de variación aceptable y, luego de hacer pruebas, se decidió que el umbral para la decisión acerca de la presencia de espacios en blanco se vinculara con esta, y tuviera el valor:

$$\mathbf{Umbral_{espacios} = (Altura\ del\ característica)/3.5}$$

El caso de la detección de puntos y comas es un caso que no se contempla en el programa de lectura de caracteres, como protección contra errores en las lecturas y degradaciones en las imágenes obtenidas (**en el algoritmo de segmentación se ignoran los caracteres de tamaño reducido**). Es por esto que los puntos y las comas son ignorados, y debe tratarse con ellos de forma diferente.

Cuando el programa reconoce un símbolo completo de tamaño reducido (el punto o la coma), la función normal de segmentación lo ignora. Cuando ocurre esto, si

[*] Se puede utilizar para reconocer otros tipos de caracteres, como guiones, comillas, etc.

el símbolo pequeño se encuentra en la parte inferior del renglón, podrá tratarse de un punto o una coma.

El punto y la coma se encuentran en la parte inferior del renglón

Una vez que se reconoce que se está tratando con un punto o una coma, se procede a distinguirlos entre sí. El método utilizado hace uso de la relación tamaño-altura de esos caracteres. En grandes rasgos, una coma es más alta que ancha, y en un punto el alto y el ancho son similares. Se calcula en el PIC la relación entre altura de símbolo y ancho del mismo (parámetros que están disponibles, para la ROI y la detección de espacios en blanco), y en función del resultado se define si se trata de un punto o una coma.

2.9.11 Código de programa OCR.c

Como anexo del informe se adjunta el código completo del programa instalado en el microcontrolador

2.10 APLICACIÓN EN LABVIEW

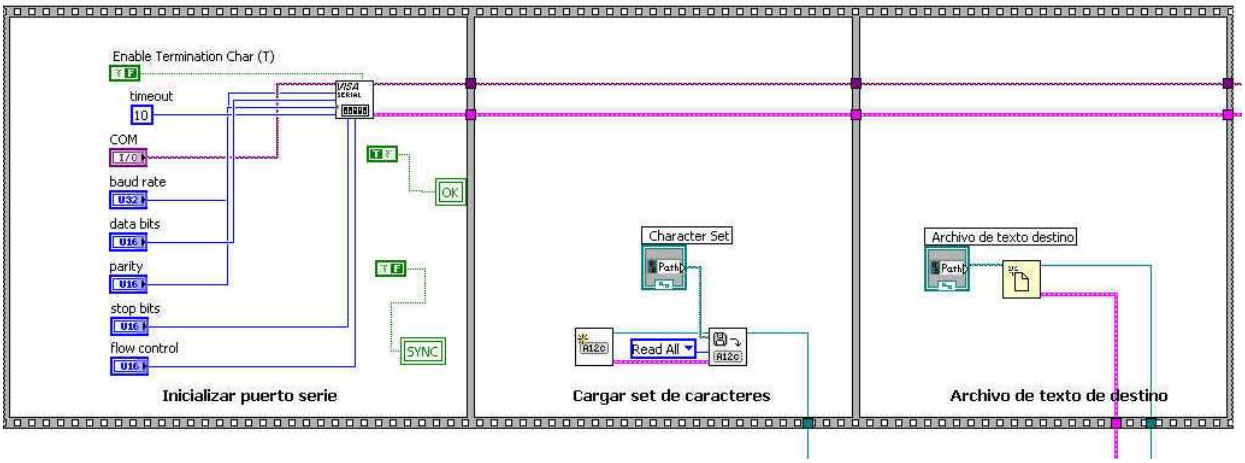
Versión Final – Aplicación conjunta de OCR

La aplicación en el microcontrolador se comunica con una interfaz programada en lenguaje gráfico, sobre LabView. Como ya se indicó en secciones previas, el programa en LabView consiste en una secuencia de configuración inicial, seguida de un lazo infinito, y se encarga de:

- Transmitir los parámetros para el control del sensor TSL3301
- Recibir las imágenes tomadas y mostrarlas en pantalla
- Ejecutar OCR sobre la región de interés deseada para las imágenes tomadas
- Guardar el texto reconocido en un archivo

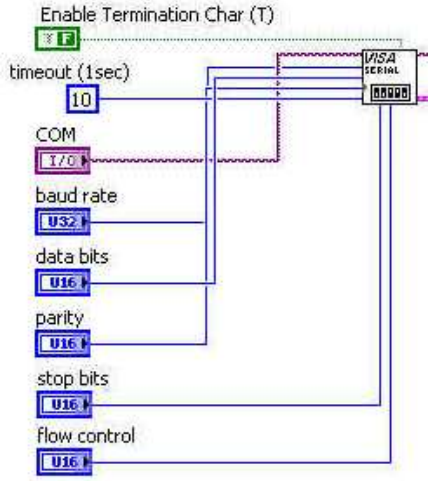
2.10.1 Configuración inicial

El bloque de configuración inicial para el programa desarrollado se muestra a continuación:



Esta secuencia cumple las siguientes funciones:

- *Abrir y configurar puerto serie para comunicaciones (RS232)*

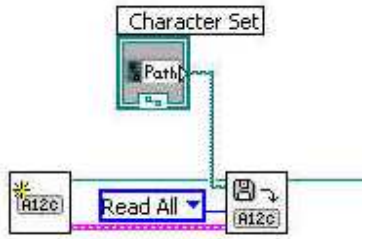


El bloque VISA Configure Serial Port inicializa el puerto serie designado **(puerto COM**), de acuerdo con los parámetros definidos por el usuario (en este caso coinciden con la configuración de la USART en el PIC):

Timeout	Tiempo de espera máximo para lecturas y escrituras
Baud Rate	Velocidad para la comunicación (**230400 bauds**)
Data bits	Número de bits de datos (**8 bits** para este proyecto)
Paridad	Especificar forma de detección de errores por paridad **No se utiliza en este proyecto**
Stop Bits	Número de bits que indican el fin de una trama (**1 bit de stop**)
Flow Control	Método de control de flujo **No se hace control de flujo** para esta aplicación: no se hizo necesario para una conexión directa con un cable corto y con sincronización por software, con pocas probabilidades de error
Enable Termination Char	Reconocer un símbolo que indica la terminación de una trama? **Falso para esta aplicación**

Las salidas de este módulo sirven para vincularlo con las futuras transmisiones y recepciones, indicando el nombre de puerto que se abrió y configuró, e información de errores en su control.

- *Iniciar sesión de OCR y levantar Character Set*



En este punto se inicia una sesión de OCR sobre la que se trabajará, y se cargan las propiedades para el reconocimiento, desde un archivo de Character Set que se creó en la utilidad de entrenamiento. La ubicación de este archivo es especificada por el usuario, y modificable.

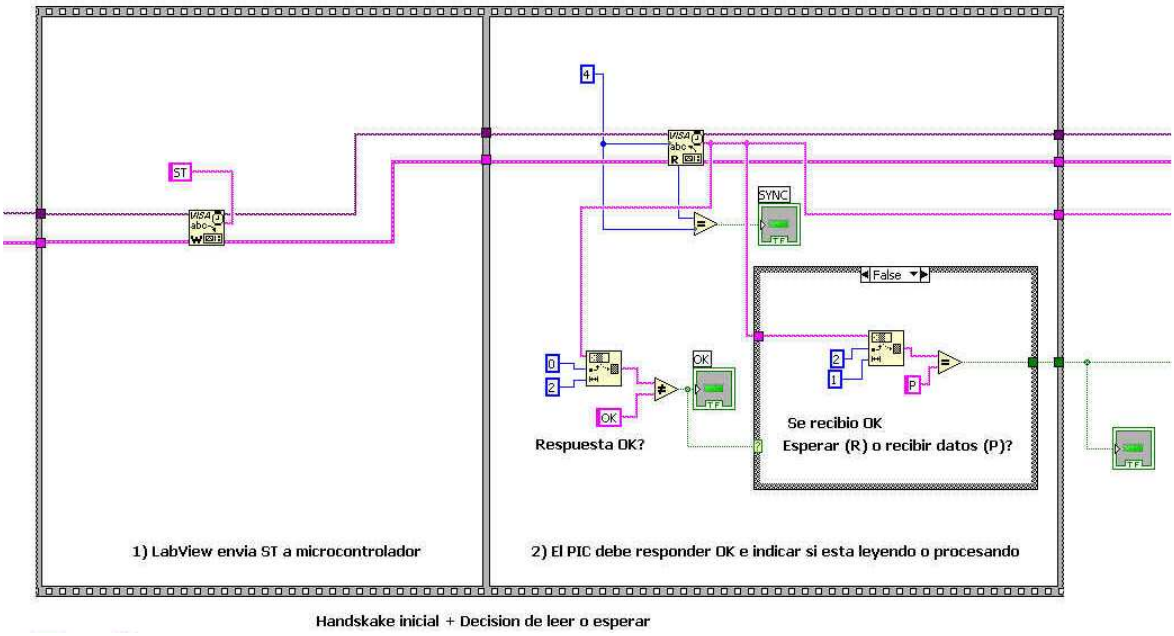
- *Cargar archivo de texto destino para los caracteres reconocidos*

Se abre un archivo de texto (o se lo crea, si no existía). En este archivo se guardarán los caracteres reconocidos en la sesión de OCR.

2.10.2 Lazo principal

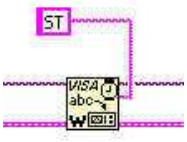
En el lazo principal se ejecuta la secuencia de sincronización con el PIC, y luego se opera de forma distinta según las indicaciones que se reciben del microcontrolador. Es por esto que no se puede mostrar el lazo completo en una sola imagen.

La secuencia de sincronización es la contraparte de la secuencia que se explicó previamente para el PIC, y se muestra en la siguiente imagen.



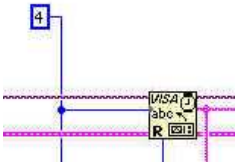
Handskake inicial + Decision de leer o esperar

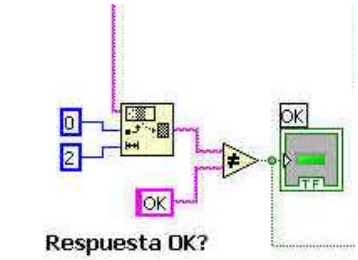
En función de los bytes recibidos desde el PIC luego del OK, se decide qué hacer en el LabView:



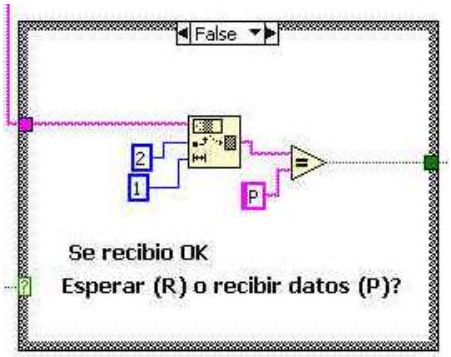
El **LabView envia "ST"**

El PIC responde con 4 bytes, de estructura "**OK**xx", con lo que indica qué hacer después



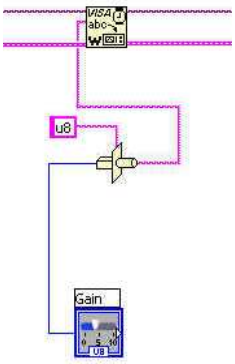


Se analizan los primeros 2 bytes recibidos, para verificar que se recibió un **"OK"** inicial. Para esto se saca un subconjunto del string de 4 bytes, indicando el **offset** desde donde iniciar (**0** porque son los primeros 2), y el **número de bytes** a sacar (**2** para el OK)
Se compara los primeros 2 bytes recibidos con "OK", y si es distinto la salida es "True". Si son iguales, la salida es "False"

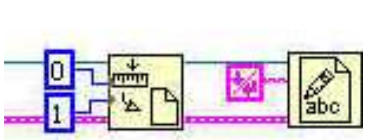


Si se recibió un OK, se analiza el siguiente byte recibido. En el string de 4 bytes, se extrae un subconjunto de **offset 2** (porque el OK ya se verificó) y **extensión de 1 byte**.
Si se recibe una P se recibirán líneas, y si se recibe otra cosa se espera a que el PIC lea.

1) Esperar a que el microcontrolador explore la línea
1)a) Transmisión de parámetros para control del sensor
Inicialmente, es necesario transmitir los parámetros de configuración del sensor al microcontrolador. Esta operación sólo se ejecuta una vez, después de que el usuario aprieta el botón.

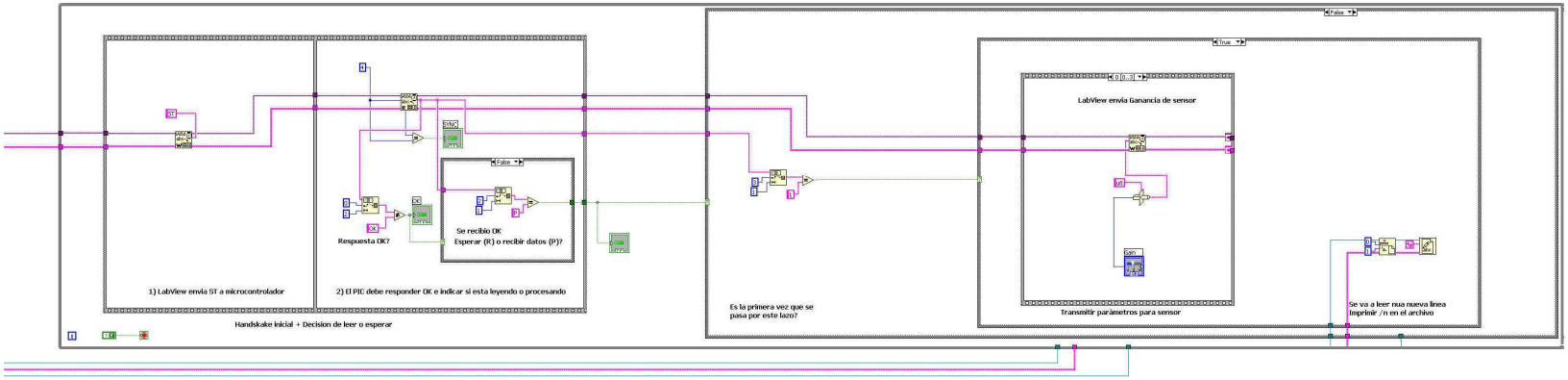


Con este bloque de programa se envia un valor entre 0 y 255, controlable por el usuario y visible en pantalla, al microcontrolador.
Este proceso se repite 4 veces para indicar la ganancia, offset y el timpo de integración para el sensor. También se determina por este método el umbral de decisión entre 0 y 1.

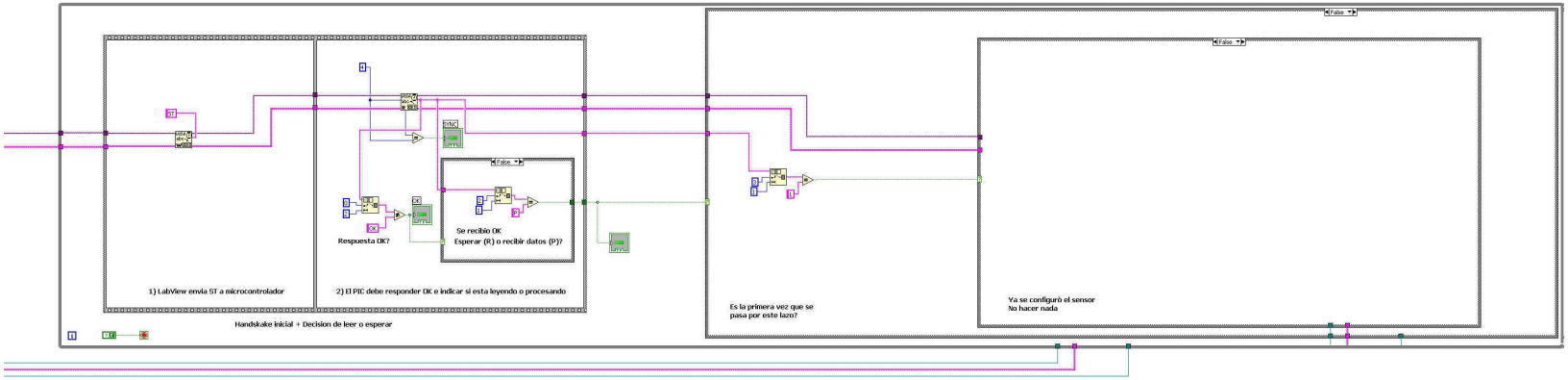


Como esta parte del programa se ejecuta una vez, cuando el usuario quiere leer una nueva línea, se aprovecha para insertar un **salto de línea** en el archivo de texto que se cargó en la inicialización.
Se utiliza el bloque **SetFilePosition**. Con las constantes de entrada indicadas, se ubica el cursor al final del archivo. Con esto se evita sobreescribir lo que se grabó. Luego se escribe el sñimbolo de salto de línea en la posición indicada.

1)b) Esperar a que se termine de explorar
Una vez se transmitieron los parámetros, no hay nada por hacer para el LabView mientras se está explorando.



1)a) Transmisión de parámetros para control del sensor



1)b) Esperar a que se termine de explorar

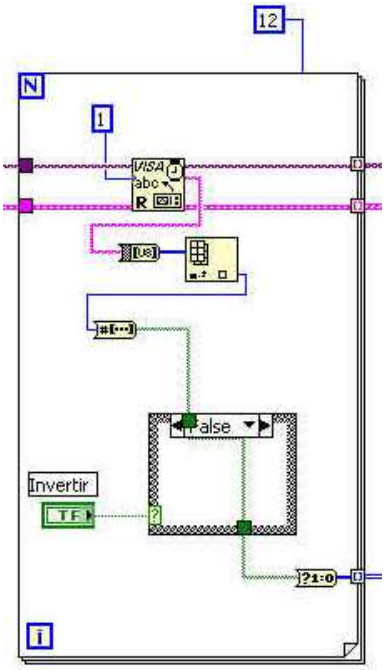
2) Recibir información del microcontrolador y ejecutar OCR cuando se deba

2)a) Recibir línea del microcontrolador

En este punto, se reciben líneas de 96 pixels en blanco y negro (12 bytes) y se las muestra en pantalla. Se da la opción de invertir colores, para poder tratar texto más claro que el fondo sobre el que se lo imprime. La imagen se muestra en pantalla en 2 gráficos:

- Un gráfico muestra una imagen de mayor longitud, que permite ver de forma más global el texto que se está leyendo
- La otra versión abarca un rango menor de líneas. Es esta imagen, más chica (para abarcar uno o dos caracteres), la que se guarda y sobre la que luego se ejecuta el OCR. De esta forma se evita que el procesamiento tome más tiempo del necesario.

Luego de recibirse cada línea, el microcontrolador indica si se terminó de completar un símbolo o no, y el LabView opera en consecuencia:

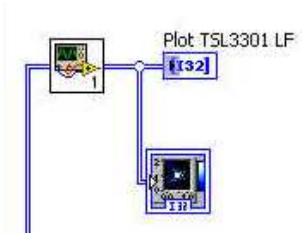


Una línea está formada de 12 bytes. Se recibe cada byte

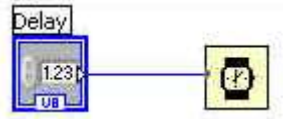
y se separa a cada uno en un arreglo de 8 bits

En este punto se da la opción con un indicador en pantalla de invertir el valor de los bits, para **poder operar con texto blanco sobre un fondo negro**

Los arreglos de 8 bits se convierten a arreglos de 8 números (1 o 0), en formato aceptable para el gráfico.



Los valores obtenidos en el punto anterior se ordenan (izquierda) y se grafican en pantalla en dos gráficos (uno más grande que el otro).



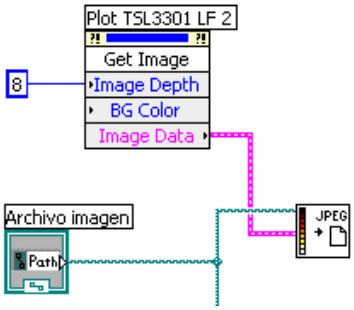
Se da la opción en pantalla de retrasar el tiempo de lazo, para poder observar lo que sucede con claridad. Esto se utilizó para realizar pruebas.

2)a)i) No se completó un símbolo, no ejecutar OCR

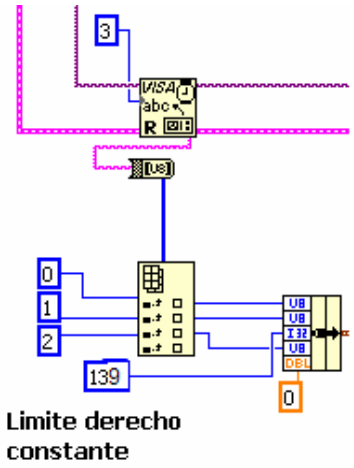
Si el PIC indica que falta seguir leyendo para completar una letra, no se ejecuta nada luego de mostrar las líneas, y se vuelve al inicio.

2)a)ii) Se completó un símbolo, ejecutar OCR y guardar en archivo

Si el PIC indica que se completó una letra, el programa en la PC guarda la imagen chica en pantalla, recibe la ROI del microcontrolador y la muestra en pantalla, ejecuta OCR sobre la ROI en esa imagen, y guarda el resultado del análisis en el archivo de texto abierto. *Ver sección 2.7.3 para ejemplo.*

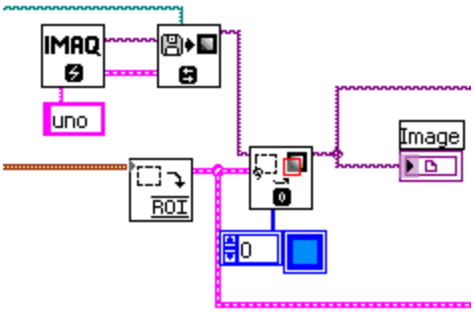


Cuando el PIC indica que terminó una letra, este bloque toma una imagen del gráfico que aparece en pantalla (GetImage) con una resolución determinada por la constante (8 representa una imagen en escala de grises de 256 colores, 4 representa imagen de 16 colores).
La imagen tomada de Plot TSL3301 LF 2 se guarda en formao JPEG en la dirección que el usuario indica (bloque Archivo imagen).



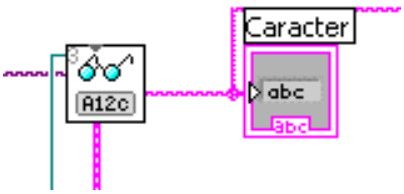
El siguiente paso es recibir la ROI del microcontrolador. Se reciben 3 bytes, para el límite izquierdo, superior e inferior. El límite derecho es constante, y representa el borde de la imagen.

Con los 3 bytes recibidos, y la constante del borde derecho, se crea un bloque de datos que representa la ROI. El otro valor que se suma al bloque permite variar el girar la ROI respecto de la horizontal

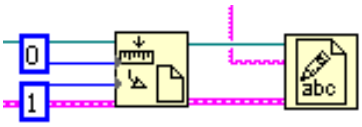


Luego de recibir la ROI, se abre una sesión de las utilidades de visión, y se carga la imagen que se había guardado (bloques superiores).

Con el bloque de datos que se creó en el punto anterior, se utiliza un bloque que lo convierte a formato de ROI. Esta ROI es superpuesta a la imagen que se cargó, y se muestra en pantalla.



Con la imagen cargada, la ROI recibida, y el set de caracteres que se cargó en la inicialización, se ejecuta OCR (izquierda), y el característica de salida se muestra en pantalla. Si la letra no puede ser reconocida, se muestra "**–**"



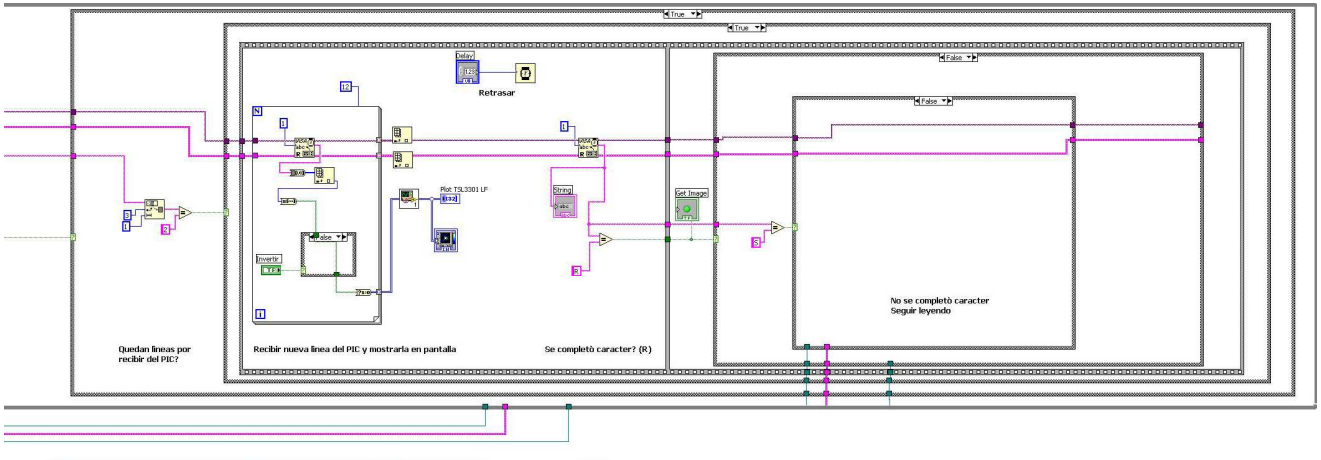
Una vez se obtuvo el símbolo, se lo guarda en el archivo de texto que se cargó en la inicialización. Para esto es necesario ubicarse al final del archivo, para no sobreescribirlo (izquierda). Luego se guarda el símbolo que resultó del punto anterior.

2)a)iii) Se completó un símbolo no detectable por OCR

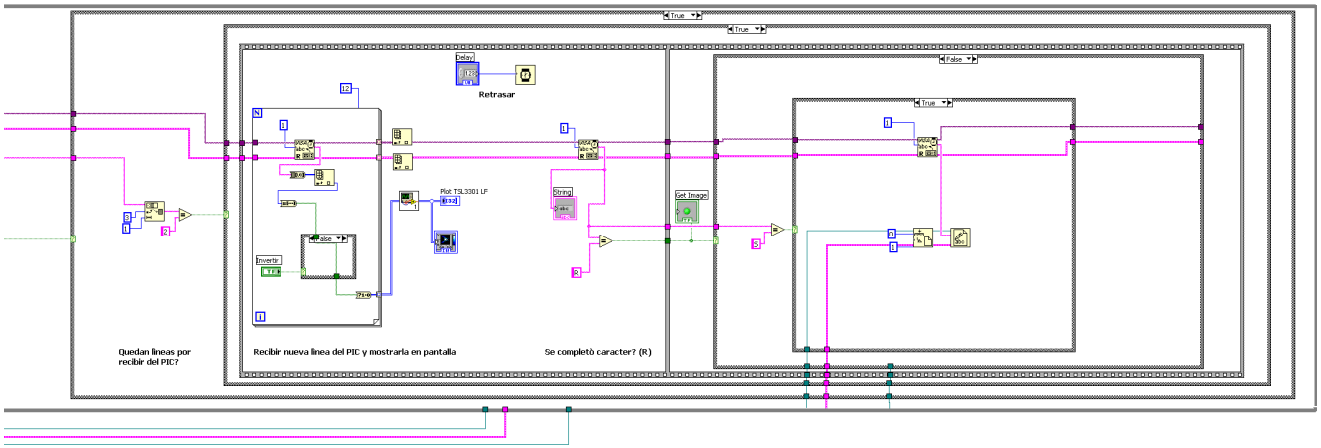
Esta operación se cumple cuando el microcontrolador encuentra un espacio, punto, coma, o algún otro símbolo que reconozca sin la ayuda del módulo de reconocimiento del LabView. Cuando el PIC indica que se cumplió esta condición, se recibe el nuevo símbolo y se lo guarda en el archivo de texto.

2)b) Se terminó de leer líneas, no hacer nada

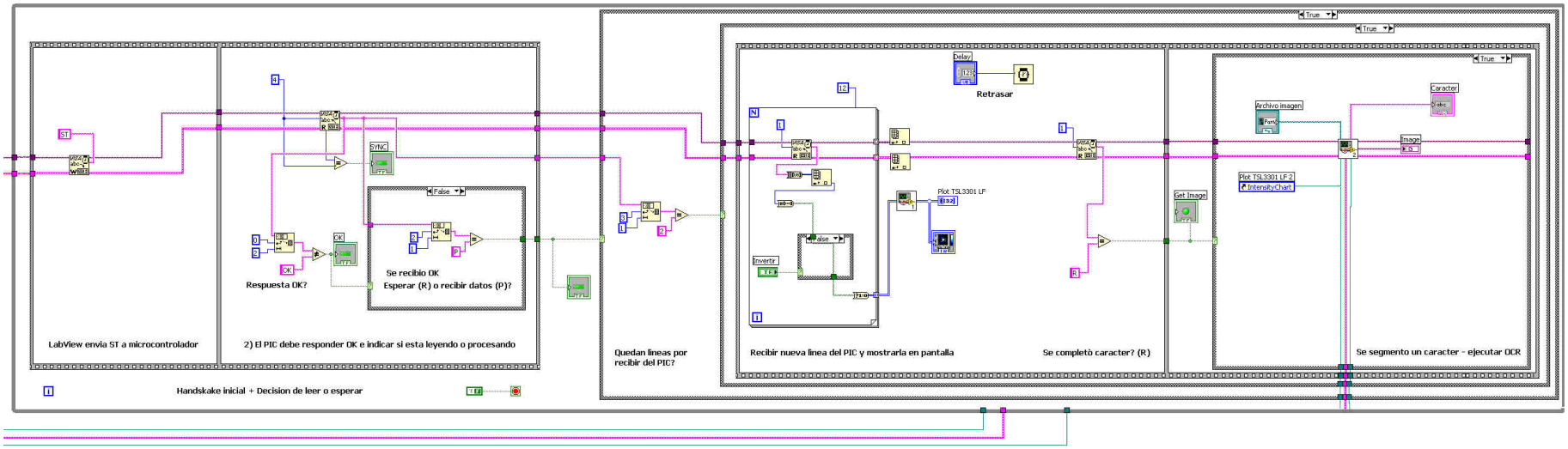
Si ya no quedan más líneas por procesar, no queda nada más por hacer hasta que se vuelva a explorar una línea. L sincronización con el PIC se sostiene, pero no se hace nada más.



2)a)i)No se completó un símbolo, no ejecutar OCR

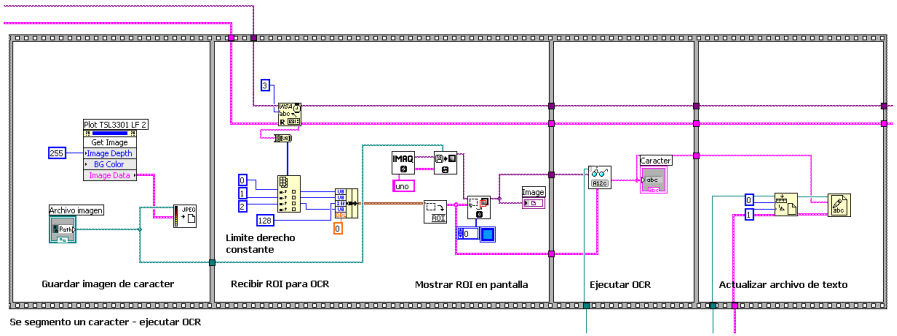


2)a)iii)Se completó un símbolo no detectable por OCR

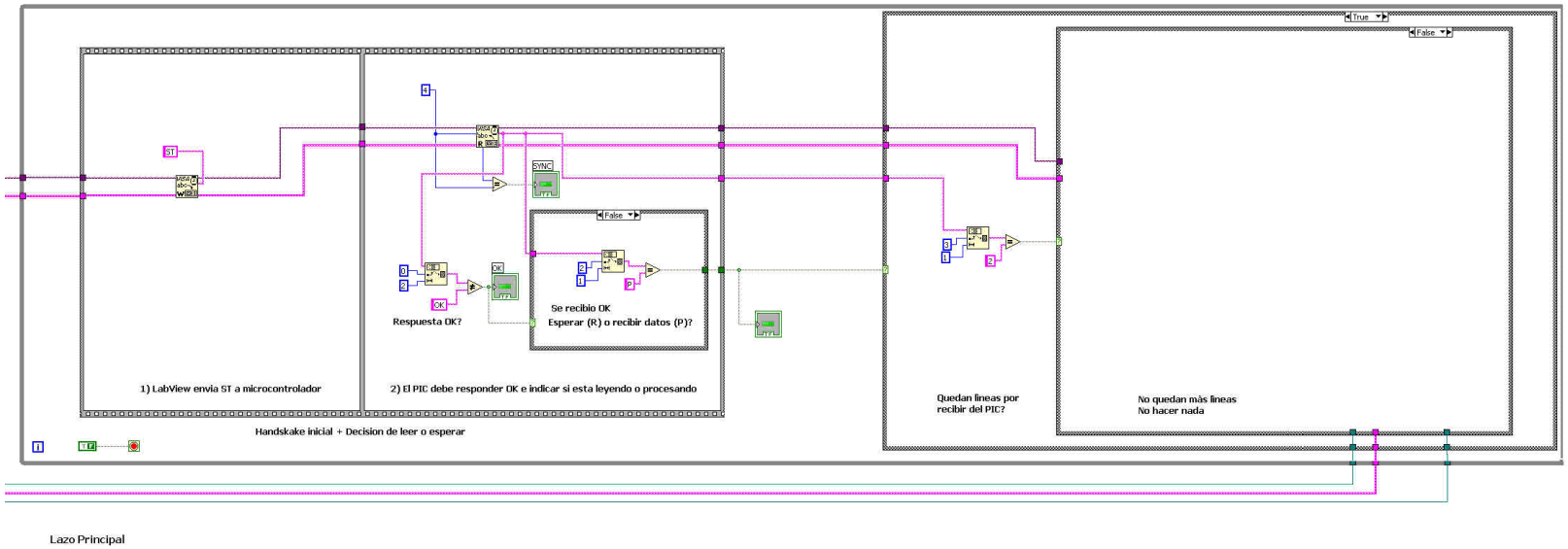


Lazo Principal

2)a)ii)Se completó un caracter, ejecutar OCR y guardar en archivo



Sub VI de reconocimiento de caracteres



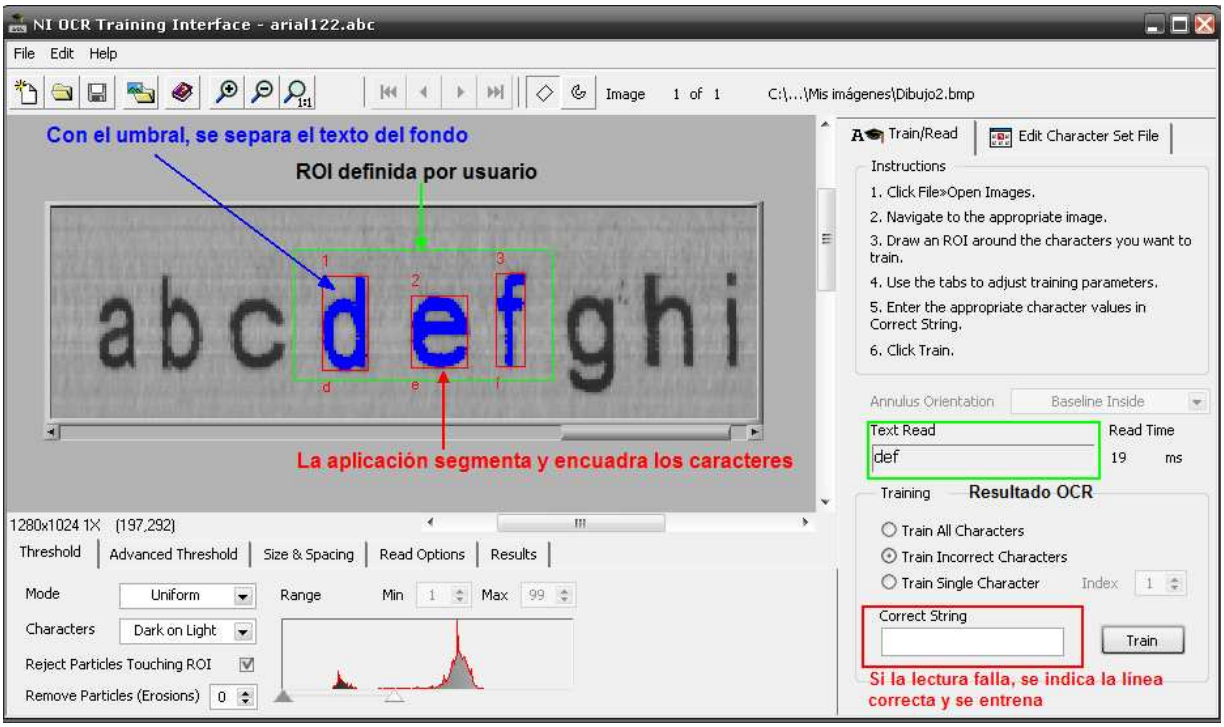
2)b)Se terminó de leer líneas, no hacer nada

2.11 ENTRENAMIENTO DE SET DE CARACTERES

Para que el sistema pueda reconocer los caracteres que se segmenten, es necesario crear un juego de parámetros que permita identificar los caracteres y distinguirlos. Este conjunto de parámetros y su significado conforma el set de caracteres.

Para este proyecto se utilizó una utilidad de entrenamiento para OCR ya disponible, la utilidad OCR Training, que viene como parte del Vision Development Module. El set de caracteres creado con esta utilidad puede ser invocado en un programa en LabView, para hacer un programa automático de OCR.

En esta utilidad, se carga una imagen con caracteres, se selecciona una ROI sobre esa imagen, y se observa cómo el programa segmenta los caracteres (cada símbolo aparece enmarcado). Si esta separación fue correcta (recordar la limitación para la segmentación que se mencionó), el usuario indica en una entrada cuál es la secuencia de caracteres que se está observando y, a partir de ahí, el programa guarda una imagen y parámetros específicos para cada símbolo.



Utilidad de entrenamiento para OCR
Disponible como parte del National Instruments Vision Development Module

De ahí en adelante se puede observar que el programa puede reconocer los caracteres que ya fueron entrenados en otras imágenes. Para mejorar los resultados, se puede entrenar más de una imagen para un símbolo, aunque un set de caracteres con demasiada información puede resultar en un tiempo de reconocimiento mayor.

Antes de guardar el set de caracteres, se debe definir un juego de parámetros que hace a la forma, complejidad y velocidad del reconocimiento. Algunos de estos parámetros (los que resultaron importantes para esta aplicación) se mencionan a continuación:

Umbral Hay varios métodos disponibles para distinguir a los caracteres de su fondo. Desde un umbral definido por el usuario a otras lógicas automáticas. En esta aplicación no es algo muy importante, porque se define un umbral en el microcontrolador y se convierte las imágenes a blanco y negro. Se eligió el método de umbral uniforme.

Estrategia de lectura Es el criterio utilizado a la hora de determinar si un símbolo coincide con alguno entrenado. Los modos posibles son Agresivo y Conservador. En el modo agresivo se utilizan menos criterios juntos para decidir si un símbolo coincide o no.

Resolución de lectura Es el nivel de detalle que utiliza el sistema a la hora de determinar si hay coincidencias. Puede ser bajo, medio o alto. A menor resolución, menor detalle, y menor tiempo de ejecución de OCR.

Independencia respecto de la relación de aspecto del símbolo Permite leer caracteres con distinto tamaño o relación alto/ancho que aquella con la que se entrenó el símbolo. Esta propiedad es valiosa para esta aplicación, para casos en que la relación de aspecto de los caracteres se vea modificada.

Remover partículas que tocan límites de ROI Esta utilidad ya fue explicada, se la activa en esta aplicación.

Separación entre elementos Si bien en este proyecto se asume que los caracteres son continuos y conexos, este módulo es capaz de trabajar con caracteres discontinuos. Se puede determinar una separación máxima entre los elementos que conforman un símbolo y, si no hay fallas, el algoritmo puede segmentar correctamente esos caracteres. Como se busca tratar con caracteres de tamaño variable, no se quiere utilizar este parámetro para superar la discontinuidad mencionada.

3 RESULTADOS

Calidad de imagen

La calidad de imagen que se logró con el lector prototipo es suficiente para distinguir fácilmente los caracteres del fondo con buena tolerancia, y programable para tratar con caracteres de trazos delgados. La conversión de la medición a blanco y negro puede dificultar la correcta segmentación para caracteres de trazos muy delgados, pero debe recordarse que las condiciones ópticas del sistema son mejorables con una lente perfectamente adaptada al sensor que se utiliza. En ese caso, no se tendría ese tipo de problemas.

Muestra de rango de sensibilidad para caracteres impresos (los defectos en la impresión influyen en la medición)

Velocidad de operación resultados

El conjunto de medidas tomadas para lograr una velocidad de operación apreciable cumplieron su propósito:

Velocidad de lectura

Es posible para el prototipo recorrer una línea completa en hoja A4 en un tiempo de entre 4 y 6 segundos, obteniendo imágenes con buena cantidad de muestras para cada letra. Recordar que se muestrea a aproximadamente 500 muestras por segundo, como se mencionó anteriormente, y que la lectura no debe extenderse por más tiempo del debido para no llegar al fin de la capacidad de memoria disponible.

Como se puede desplazar el lector a una velocidad relativamente alta, es más difícil que se presenten variaciones grandes de velocidad de exploración a lo largo de una línea (es más difícil mantener una velocidad constante cuando esta velocidad es reducida).

Velocidad de conversion

La transmisión de cada línea guardada desde el PIC a la aplicación sobre LabView se produce a una velocidad suficiente para convertir alrededor de 6 caracteres por segundo. Es necesario tener en consideración que la velocidad de operación del sistema varía con las condiciones de funcionamiento de la computadora sobre la que opera la aplicación de OCR.

Segmentación

El algoritmo de segmentación desarrollado funciona apropiadamente, de acuerdo con sus especificaciones. Se segmentan caracteres correctamente, aún cuando se opera con texto por encima y por debajo de la línea que interesa convertir. El reconocimiento de espacios en blanco funciona correctamente en general, aún considerando las diferencias en la altura posible de los caracteres.

El caso de la **í** es especial, porque esta es la única letra para la cual la parte separada del cuerpo principal del símbolo puede extenderse varias muestras más allá de donde el cuerpo de la letra termina. Es para contener esta posibilidad que se aplicó la mencionada técnica de esperar unas líneas después del fin del símbolo, y esta técnica funciona en la mayoría de los casos.

La única condición para el usuario que asegura una buena selección de renglón consiste en que el primer símbolo del renglón que se quiere leer abarque la parte “inferior” del lector (la más cercana al usuario). A partir de la primera letra reconocida, las siguientes serán aceptadas cuando abarquen una posición similar a la anterior a éstas, de modo que, si el lector no se desvia abruptamente, se podrá reconocer texto entre múltiples renglones aunque el desplazamiento del lector no sea perfectamente paralelo a la línea del renglón.

Si bien se entiende que es necesario mejorar la capacidad del lector con una lente adaptada para garantizar la correcta segmentación de caracteres de trazos de ancho variable y con serifas y adornos, debe reconocerse que se pudo poner en funcionamiento la lógica de segmentación que se deseaba.

Utilidad de reconocimiento óptico de caracteres

Una vez conseguido un algoritmo de segmentación que funciona en un amplio rango de situaciones, la capacidad de reconocimiento del sistema queda en manos del módulo de reconocimiento elegido para la aplicación. En pruebas iniciales, se cargó un set de caracteres de pocas muestras (en general consistía en una muestra por cada símbolo). El rendimiento del sistema mejoró mucho al hacer un entrenamiento más fuerte antes de las mediciones, con más variantes.

La mayor dificultad del sistema completo en estas condiciones es la de distinguir una misma letra en mayúscula y minúscula. Si bien con entrenamiento se mejora mucho la respuesta e identificación de esos caracteres, se está trabajando al mismo tiempo con un reconocimiento independiente de la relación de aspecto de los caracteres. Es por esto que será difícil distinguir entre las dos variantes de un mismo símbolo. De cualquier forma, esto no ocurre para caracteres en los que la versión en mayúscula sea distinta de su versión en minúscula (como la G y la g), y existen métodos sencillos de corrección para este cuando ocurre esta falla, disponible en procesadores de texto ordinarios.

Naturalmete, la degradación del texto de origen repercute en los resultados de efectividad de la conversión y puede degradarlos. Dependiendo de la calidad de la impresión, las combinaciones de letras como la rt, ff, etc, pueden estar unidas y verse como un símbolo simple. Para superar estos errores se puede tomar una solución alternativa: entrenar un caracter especial que, por ejemplo, represente al par rt. Sin embargo, este defecto no es responsabilidad del usuario ni de la aplicación.

Luego de entrenar el sistema para trabajar con caracteres en letra Arial de varios tamaños, se procedió a medir bloques de texto y analizar los errores en algunos resultados. En mediciones aisladas, y para una muestra de 2300 caracteres en letra de tamaño 11 y 12, se obtuvo un porcentaje promedio de efectividad del 96%, sin considerar las fallas que se produjeron por defectos de iluminación y por caracteres conectados, como se mencionó en el párrafo anterior. En mayor medida el sistema confunde la letra i con la letra l, algunas minúsculas con sus versiones mayúsculas y viceversa, y puede ocurrir que aparezcan símbolos registrados que resulten más grandes que la ventana de 60 muestras y por eso no sean reconocidos (generalmente la “m” y la “W”). Éste último problema se soluciona con ventanas más grandes, (aunque se paga con un mayor tiempo de procesamiento de imagen).

Si bien en este punto se considera probada la efectividad del sistema desarrollado, se entiende que su rendimiento puede mejorarse más con agregados y mejoras que se mencionarán más adelante.

4 CONCLUSIONES

Los objetivos planteados para este proyecto fueron alcanzados, lográndose el desarrollo de una aplicación multiplataforma capaz de reconocer caracteres del alfabeto latino, y la construcción de un prototipo de lector compatible con esta aplicación y con las características y funcionalidades que ésta le exige.

Para lograr dichos objetivos fue necesario lograr una operación cooperativa entre todos los dispositivos que hacen al sistema, para lo cual se debió agregar módulos y hacer adaptaciones que permitieran interconectarlos y utilizarlos en una forma que optimizara el aprovechamiento de las herramientas disponibles.

Durante el desarrollo del sistema se hizo necesario resolver problemas y complicaciones que se presentaron, mayormente vinculados con las características ópticas necesarias para la construcción del lector prototipo. Fue preciso realizar muchas pruebas sobre el conjunto del sistema de iluminación y de transporte óptico de imágenes, hasta lograr una disposición de componentes que se tradujera en imágenes de buena calidad. Superada esta dificultad, el otro desafío de mayor importancia que se presentó fue el de crear un algoritmo de segmentación robusto y confiable para la mayor parte de los caracteres que se pudieran encontrar, que pudiera trabajar con más de un renglón de texto en pantalla.

Como ya se mencionara, la funcionalidad de un control de desplazamiento incorporado al sistema permitiría reducir sus limitaciones, y por tanto mejorar la tasa de efectividad máxima del sistema, con entrenamiento de los suficientes caracteres en el utilizado módulo de OCR o en algún otro.

Los posibles avances sobre la presente versión de la aplicación quedan por ser analizados en detalle, pero se los explora en la siguiente sección. En este punto, se entiende que las potenciales evoluciones y usos de esta aplicación indicarían que la propuesta puede encontrar aplicaciones comerciales y de uso cotidiano de distinto tipo.

5 USOS POTENCIALES

El prototipo, con el nivel de desarrollo propuesto en el presente, permite utilizarlo como traductor de pixels a caracteres multiplataforma, que interpreta textos del alfabeto latino para los diferentes idiomas que lo utilizan.

La versatilidad del desarrollo permite en principio reconocer las evoluciones posibles que a continuación se detallan, indicándose en cada caso los avances, adaptaciones y/o modificaciones necesarias, requeridas para su logro.

5.1 Adaptaciones y modificaciones – mejoras

Las siguientes son mejoras aplicables al sistema desarrollado en una futura etapa. No todas las alternativas son aplicables simultáneamente, pero se considera que vale la pena mencionarlas para resaltar el potencial de crecimiento del proyecto:

a. *Posible aplicación con otros programas de OCR gratuitos*

Es posible implementar una variante de este sistema, que pueda operar con otros programas de reconocimiento de caracteres. Después de todo, el microcontrolador tiene en su poder la imagen de cada línea, y sabe cuándo termina y comienza cada característica. Como mínimo, el microcontrolador podría mandar la imagen de cada símbolo segmentado a la computadora, y un programa ejecutado sobre ésta podría invocar el reconocimiento óptico sobre esa imagen (para evitar los problemas de segmentación mencionados). Otras variantes son posibles.

b. *Diccionarios para mejora de rendimiento*

Para mejorar el resultado del reconocimiento se puede pasar el resultado del mismo por una aplicación con funciones de diccionario. Como mínimo, se puede llevar el archivo de texto guardado en esta misma aplicación a un procesador de texto y hacer corrección de errores sobre ese texto cargado.

c. *Lente diseñada para el sensor*

Es claro que si se utiliza una lente adaptada completamente al sensor con el que se trabaja, se podrá aprovechar por completo su capacidad de tomar imágenes, y se evitaría tener que resolver por software los defectos propios del sistema óptico que se mencionaron.

d. *Aumentar resolución*

En esta aplicación se trabaja con imágenes en blanco y negro para mejorar la velocidad y no consumir grandes bloques de memoria en cada medición desde el sensor. Si bien trabajar con imágenes en escala de grises no parece algo necesario al tratar con texto impreso, hacerlo podría mejorar la resolución y el detalle de las imágenes que se reconocen en la computadora, y al mismo tiempo dar la posibilidad de que el lector guarde imágenes pequeñas con el lector, además de texto.

Hay que tener en cuenta que, si se guarda más información para las imágenes, también será necesario lograr transmitirla a mayor velocidad a la computadora para no comprometer la velocidad del sistema. Una alternativa para lograr esto sería utilizar una interfaz USB de alta velocidad para la comunicación entre la computadora y un microcontrolador (que tenga módulo USB . Existen implementaciones más directas, que simulan una USART sobre una interfaz física USB, y pueden alcanzar velocidades de hasta 640kbps (casi 3 veces la velocidad lograda para este sistema).

e Aplicación libre – memoria SD

Una posible implementación con una memoria SD de alta capacidad se puede lograr con mínimas modificaciones en la lógica del programa (las modificaciones de hardware necesarias para implementar esta ampliación deberán analizadas).

Con una memoria de alta capacidad y velocidad no sólo se podrá guardar imágenes en escala de grises, sino que se podría guardar más de una línea explorada antes de transmitirla (debería diseñarse un protocolo para reconocer dónde comienzan y terminan las líneas). Con esta funcionalidad adicional, se podría crear un prototipo que pueda funcionar *separado de la computadora*:

- cuando está desconectado de la computadora, el sistema guarda imágenes de muchas líneas en la memoria.
- cuando se conecta el sistema a la computadora, se ejecuta OCR sobre las líneas guardadas, como se hace en esta aplicación.

Si una lectura del sensor en alta resolución ocupa 96 bytes, una memoria de 1 gigabyte podría guardar alrededor de 10.000.000 de lecturas sucesivas. A 2.000 lecturas para un renglón, se tendría capacidad para guardar 5.000 renglones en la memoria, antes de tener que descargarlos en la computadora.

f. Posible implementación de algoritmo OCR en DSP

Si bien ya se mencionó que, por la naturaleza de este sistema, sería mejor utilizar un programa de reconocimiento de patrones de alta capacidad, sería posible implementar el algoritmo de OCR dentro de un microcontrolador o un DSP (Digital Signal Processor) y observar cuál es el rendimiento que se alcanza.

Existen varios algoritmos que pueden implementarse con este propósito, desde cálculos estadísticos acerca de qué característica se corresponde mejor con una secuencia de estados recibida (Hidden Markov Model), a cálculos de valores que describen características únicas de cada símbolo (por ejemplo, los descriptores de Fourier, una serie de valores que representa la forma del contorno externo de los caracteres leídos) y permiten identificarlos.

g Interfaz con software intérprete de voz

Incorpora la tecnología que lee el texto reconocido al usuario, a través de dispositivos de conexión alámbricos/inalámbricos (auriculares, parlantes).

5.2 Evoluciones

o *“Lápiz” óptico autónomo*

Este tipo de lector es la versión portátil del prototipo que se desarrolló. Es capaz de almacenar gran cantidad de líneas antes de necesitar descargarlas en una computadora, y con una interfaz de comunicación de alta velocidad podría tratar texto en escala de grises.

*Requierimientos: puntos **b, c** y **e** de la sección anterior*

o *Scanner portátil*

La corrección de offset mencionada en la sección 2.8.5 es útil para mejorar la calidad de imagen para el reconocimiento de texto, pero reduce ligeramente el rango dinámico del sistema y dificulta la visualización de imágenes en escala de grises (las líneas de variación de luz persisten en imágenes con muchos tonos distntos). Aunque este no fue el propósito principal del sistema, si se supera esta dificultad (con una lente adaptada a la estructura del lector), se podrá observar imágenes en escala de grises

con una resolución vertical de 300dpi, y esta información podría ser grabada en una memoria, del mismo modo que las líneas de texto que se graban en blanco y negro en esta aplicación.

*Requierimientos: puntos **c, d** y **e** de la sección anterior*

o *Lector Braille*

La estructura que se eligió para el lector en este proyecto utiliza una fuente de iluminación no vertical, lo que permite ver variaciones en el relieve de la superficie sobre la que se lo pase. Esta característica hace posible utilizar este lector para reconocer texto en Braille. Para esto, el algoritmo de segmentación debería ser modificado, ya que los caracteres en Braille son esencialmente discontinuos. Por otro lado, la estructura de los caracteres en este formato es más uniforme, así que, una vez separados será más fácil identificarlos. Un lector Braille de estas características puede vincularse con una utilidad de conversión de texto a voz para servir como traductor portátil.

*Requierimientos: puntos **b, c, e, g**, opcional **f**.*

o *Conectividad del lector con otros dispositivos electrónicos*

Es posible generar una interfaz, alámbrica o inalámbrica que permita el almacenamiento de los datos leídos en dispositivos electrónicos de otro tipo o con otras interfaces, utilizando las capacidades del equipo receptor para dar más funcionalidades al sistema.

6 AGRADECIMIENTOS

Profesor Ingeniero Ricardo Vecchio, por su colaboración desde los desarrollos iniciales de la interfaz de conexión entre el microcontrolador y la aplicación sobre LabView y por el esfuerzo y la dedicación en la respuesta a las consultas que surgieron en el transcurso del proyecto.

Profesor Ingeniero Norberto Heyaca, por la dedicación al responder consultas relativas a la dirección del proyecto y sus posibles aplicaciones desde los principios del desarrollo.

A mis compañeros de facultad, por sus observaciones y recomendaciones para mejorar el proyecto.

A mi familia, por el apoyo brindado y la ayuda durante la elaboración de este informe y a lo largo del proyecto en general.

7 REFERENCIAS BIBLIOGRÁFICAS

- PIC18F452 Datasheet, Microchip Technology Inc., 2006
- LM7805 Datasheet, Fairchild Semiconductor Corporation, 2001
- TSL3301-LF Datasheet, Texas Advanced Optoelectronic Solutions Inc., 2007
- Designer's Notebook 4: Using the TSL3301 with a Microcontroller, Texas Advanced Optoelectronic Solutions Inc., 2001
- Designer's Notebook 15: Linear Sensor Array Optical Path Design, Texas Advanced Optoelectronic Solutions Inc., 2004
- Designer's Notebook 2: SELFOC® Lens Array for Line Scanning Applications, Texas Advanced Optoelectronic Solutions Inc.
- Design Considerations for an Absolute Steering Angle Encoder Demo, TAOS Inc. - Microchip Technology Inc., 2004
- 24LC256 Datasheet, Microchip Technology Inc., 2007
- AN1195, Using C to Interface 8051 MCUs with I^2C™ Serial EEPROMs, Microchip Technology Inc., 2008
- AN991, Using the C18 Compiler and the MSSP to Interface I^2C™ EEPROMs with PIC18 Devices, Microchip Technology Inc., 2005
- MAX232 Datasheet, Texas Instruments, 2004
- Getting Started With LabView™, National Instruments, 2005
- LabView™ Fundamentals, National Instruments, 2005
- Nationa Instruments Vision Concepts Manual, National Instruments Corporation, 2007
- Microchip MPLAB
- MPLAB® C18 C Compiler Getting Started, Microchip Technology Inc., 2005
- MPLAB® C18 C Compiler Libraries, Microchip Technology Inc., 2005
- MPLAB® C18 C Compiler User's Guide, Microchip Technology Inc., 2005
- www.taosinc.com
- http://www.nsgamerica.com/index.php?lang=english&page=lense_arrays
- en.wikipedia.org
- www.ni.com

8 ANEXO – CÓDIGO DE PROGRAMA PARA MICROCONTROLADOR