Universidad Autónoma de Querétaro

Facultad de Ingeniería

Ingeniería en Automatización

Línea terminal en electrónica Industrial

Control de Intensidad Luminosa por Procesamiento de Imágenes

**Tesis**

Que como parte de los requisitos para obtener el Grado de

Ingeniero en Automatización

**Presenta:**

Luis Raúl Castañón Gutiérrez

Dirigido por:

Dr. Jorge Domingo Mendiola Santibáñez

Nombre del Sinodal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Presidente Firma

Nombre del Sinodal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Secretario Firma

Nombre del Sinodal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vocal Firma

Nombre del Sinodal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Suplente Firma

Nombre del Sinodal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Suplente Firma

|  |  |
| --- | --- |
| Nombre y Firma  Director de la Facultad | Director de Investigación y Posgrado |

Centro Universitario

Querétaro, Qro.

Fecha (será el mes y año de aprobación del Consejo Universitario)

*Dedicatoria*

# Abstract

# Resumen

La propuesta se refiere al uso y desarrollo de tecnologías sobre el procesamiento de imagen, esta orientada a mejorar el control de la iluminación y a hacer más eficiente el consumo de energía eléctrica en ambientes automatizados.

Usando una interfaz gráfica y la librería de procesamiento de imágenes: “Open CV”, se implementa un algoritmo para encontrar los niveles de intensidad luminosa en las imágenes, dichos niveles servirán como la retroalimentación de un lazo de control.

El proyecto implementa y desarrolla un controlador difuso. La intensidad luminosa es procesada en función de sus grados de pertenencia en su conjunto correspondiente y en función de la intensidad requerida por el usuario. La señal de control es enviada a un microcontrolador a través de un protocolo de comunicación serial, donde se procesa y traduce en un pulso modulado en amplitud y frecuencia.

El control de la iluminación toma efecto al usar un arreglo electrónico que acopla los pulsos modulados del microcontrolador y con ellos modula la intensidad de iluminación de una lámpara LED.

# Índice

[Abstract 3](#_Toc508711193)

[Resumen 4](#_Toc508711194)

[Índice 5](#_Toc508711195)

[Lista de figuras 8](#_Toc508711196)

[Lista de Tablas 9](#_Toc508711197)

[I. Introducción 10](#_Toc508711198)

[I.I Justificación 11](#_Toc508711199)

[I.II Formulación del problema 11](#_Toc508711200)

[I.III Objetivos 12](#_Toc508711201)

[I.III.I Objetivos generales 12](#_Toc508711202)

[I.III.II Objetivos específicos 12](#_Toc508711203)

[I.IV Estructura de la tesis 13](#_Toc508711204)

[II. Marco teórico 14](#_Toc508711205)

[II.I Antecedentes 14](#_Toc508711206)

[Owlet by Schreder 14](#_Toc508711207)

[Control Centralizado, Bticino 15](#_Toc508711208)

[II.II Luz e Intensidad Luminosa 15](#_Toc508711209)

[II.III Generalidades del procesamiento de Imágenes 16](#_Toc508711210)

[II.IV Microcontroladores 17](#_Toc508711211)

[II.V Protocolo de Comunicación UART 19](#_Toc508711212)

[II.VI PWM 19](#_Toc508711213)

[II.VII OpenCV 20](#_Toc508711214)

[II.VIII Control Automático 20](#_Toc508711215)

[II.IX GitHub 22](#_Toc508711216)

[II.IX.I Instalación y configuración inicial de GitHub 22](#_Toc508711217)

[III. Metodología 26](#_Toc508711218)

[III.I Integración de OpenCV en Windows 7 y Qt Creator 27](#_Toc508711219)

[III.II Desarrollo de la interfaz gráfica 30](#_Toc508711220)

[III.II.I Operación manual: Obtención de la intensidad luminosa 32](#_Toc508711221)

[Botón “Abrir Imagen” 32](#_Toc508711222)

[Botón “Convertir a Grises” 33](#_Toc508711223)

[Botón “Obtención de Puntos Medios” 33](#_Toc508711224)

[Botón “Niveles de Intensidad” 34](#_Toc508711225)

[III.II.II Operación automática: Control de intensidad luminosa 35](#_Toc508711226)

[III.III Desarrollo de la aplicación en el microcontrolador 36](#_Toc508711227)

[III.III.I Configuración de los registros del microcontrolador 38](#_Toc508711228)

[Configuración del reloj 38](#_Toc508711229)

[Configuración de las entradas y salidas digitales 39](#_Toc508711230)

[Configuración del temporalizador 40](#_Toc508711231)

[Configuración de los registros de la interfaz de comunicación UART 42](#_Toc508711232)

[III.III.II Procesamiento de los datos 45](#_Toc508711233)

[IV. Resultados y Discusiones 46](#_Toc508711234)

[V. Conclusiones 47](#_Toc508711235)

[Apéndice I. Código implementado en el microcontrolador 48](#_Toc508711236)

[Apéndice II. Código implementado en la GUI 51](#_Toc508711237)

[Archivo de configuración 51](#_Toc508711238)

[Librerías 52](#_Toc508711239)

[methods.h 52](#_Toc508711240)

[mainwindow.h 52](#_Toc508711241)

[Archivos fuente 55](#_Toc508711242)

[methods.cpp 55](#_Toc508711243)

[mainwindow.cpp 61](#_Toc508711244)

[Diseño de la interfaz 69](#_Toc508711245)

[GUI 69](#_Toc508711246)

[XML 69](#_Toc508711247)

[Referencias 82](#_Toc508711248)

# Lista de figuras

[Figura 1. Espectro electromagnético y la luz visible. 16](#_Toc508711249)

[Figura 2. Arquitectura del microcontrolador MSP430G2553. 17](#_Toc508711250)

[Figura 3. Estructura de un mensaje del protocolo UART. 19](#_Toc508711251)

[Figura 4. Modulación de una señal PWM. 20](#_Toc508711252)

[Figura 5. Diagrama a bloques de un sistema controlado. 21](#_Toc508711253)

[Figura 6. Ventana "acerca de" en QT Creator. 26](#_Toc508711254)

[Figura 7. Ventana "acerca de" de CCS. 27](#_Toc508711255)

[Figura 8. Ventana de edición de variables de entorno. 28](#_Toc508711256)

[Figura 9. Integración de librerías y archivos de acceso en Qt Creator. 29](#_Toc508711257)

[Figura 10. Diagrama de flujo de la obtención de la intensidad luminosa. 30](#_Toc508711258)

[Figura 11. Proceso de control de iluminación (GUI). 31](#_Toc508711259)

[Figura 12. Sección: "Proceso Manual" en la interfaz gráfica. 32](#_Toc508711260)

[Figura 13. Agrupación: "Puntos Medios". 33](#_Toc508711261)

[Figura 14. Conjunto visual: "Tamaño del área". 34](#_Toc508711262)

[Figura 15. Conjunto visual: "Niveles de Intensidad". 34](#_Toc508711263)

[Figura 16. Secuencia de programación del microcontrolador. 37](#_Toc508711264)

[Figura 17. Módulo del reloj en microcontrolador MSP430G2553. 39](#_Toc508711265)

[Figura 18. Diagrama de configuración de salidas del MSP430G2553. 39](#_Toc508711266)

[Figura 19. Diagrama a bloques del temporalizador. 41](#_Toc508711267)

[Figura 20. Modos de salida usando el temporalizador. 42](#_Toc508711268)

[Figura 21. Diagrama a bloques de la interfaz del MCU en modo UART. 44](#_Toc508711269)

# Lista de Tablas

# Introducción

En las siguientes líneas se propone el desarrollo de un sistema de control de intensidad luminosa, donde se implementa una interfaz gráfica para el procesamiento de imágenes con la finalidad de obtener información para alimentar a un sistema de control con retroalimentación. Se usa un dispositivo digital para embeber los algoritmos el cual utiliza un protocolo de comunicación UART. De un control fuzzy, se obtendrá una señal que se acoplará a un driver tipo Boost para lámparas LED.

Derivado del constante crecimiento poblacional y del impacto ambiental que este conlleva, uno de los rubros de la ingeniería versa en torno a mejorar la eficiencia de los sistemas que hacen uso de los servicios básicos del ser humano, como el agua potable y la energía eléctrica. (Lobao, Devezas, & Catalao, 2015). Se han desarrollado tecnologías que mejoran los sistemas de iluminación, como podemos evidenciarlo con los reguladores tipo “deemers”, el uso de celdas solares o con la tecnología LED. También otras áreas de la ingeniería buscan solventar esta necesidad, como es por ejemplo la arquitectura moderna, aprovechando la luz solar en las construcciones.

Otro ámbito popularizado en los últimos años versa en torno al procesamiento de imágenes, que como podemos notar, cada vez se van encontrando mayores aplicaciones al uso de sistemas de visión en la industria e incluso en nuestros hogares, así que el proyecto tiene la tendencia de ser innovador al usar un sistema de visión para cerrar un lazo de control. Una base de nuestra formación se enfoca al desarrollo y uso de sistemas digitales en donde se implementen funcionalidades como protocolos de comunicación y configuración de señales de control que puedan ser utilizadas en acoplamientos de potencia; en conjunto, las cualidades de este proyecto, engloban la innovación tecnológica, la vinculación con un problema social y el uso de nuestra formación académica para solventarlo.

## I.I Justificación

En la actualidad los sistemas de visión son cada día más populares en las aplicaciones de innovación tecnológicas. Se pueden encontrar cámaras en un sin número de edificios, oficinas, salones e incluso en los hogares, muchas de las cuales tienen la función primaria de ser usados como elementos de seguridad y protección contra la delincuencia; también podemos notar la creciente selección y preferencia por los sistemas de iluminación tipo LED, puesto que permiten un mayor ahorro en los consumibles, como el servicio de luz, así como el impacto ambiental que este conlleva.

Bajo la premisa del reúso de la infraestructura, se deriva la idea de adecuar un sistema que nos permita automatizar la iluminación en ambientes con variaciones en ella, tal como espacios que contengan ventanas o domos, lo cual derivará en un aumento al ahorro de los consumibles y por ende a una disminución al impacto ambiental.

Es de vital importancia, dada la tasa de crecimiento poblacional y sus consecuencias, que los servicios de consumo humano sean utilizados de una manera mucho más eficiente. Por otra parte, como ingenieros y miembros de grupos sociales, nos compete proponer soluciones a esta clase de problemas haciendo uso de la tecnología y los conocimientos generados durante nuestra formación.

## I.II Formulación del problema

## I.III Objetivos

Bajo el contexto social y tecnológico, así como las necesidades de innovación en nuestro rubro, se han delimitado los siguientes objetivos para desarrollar un prototipo de un sistema de control de la intensidad luminosa haciendo uso del procesamiento de imagen.

### I.III.I Objetivos generales

* Desarrollar un sistema que nos permita controlar la intensidad luminosa usando como actuador un convertidor tipo boost de corriente directa para usarse con lámparas LED y como sensor una cámara que permita inhabilitar su característica de auto-compensación de luminosidad embebido en una interfaz gráfica de usuario.

### I.III.II Objetivos específicos

* Integrar la librería OpenCV en un software de distribución libre que permita el desarrollo de GUIs.
* Procesar imágenes usando los métodos de OpenCV.
* Diseñar un controlador digital apropiado para el sistema.
* Acoplar la señal de control en un sistema digital que permita integrar un convertidor tipo boost en el diseño.
* Diseñar una interfaz gráfica donde se controle y se asigne el valor más apropiado de intensidad luminosa.

## I.IV Estructura de la tesis

# Marco teórico

## II.I Antecedentes

La energía eléctrica es uno de los servicios más importantes para el ser humano en la actualidad. Su uso es primordial en las actividades del día a día y es imprescindible en la industria. Generar energía eléctrica representa uno de los problemas ambientales en nuestra actualidad, pues es a través de los recursos naturales y dadas las demandas que se ven reflejadas en estudios del SIE (Sistema de Información Energética), estos se van agotando. (SENER, 2017) & (Liu & Ramirez, 2017).

Con base a lo anterior, nuestro propósito versa en torno al ahorro de tal recurso energético, para esto se desarrolla un sistema con mayor eficiencia, al iluminar entornos de manera adecuada y promoviendo el ahorro de energía usando controles automatizados.

Muchas empresas han considerado el objetivo anterior y se han desarrollado numerosos sistemas para el ahorro de energía eléctrica y a su vez automatizando luminarias y entornos. En este apartado se mencionan algunos sistemas y soluciones desarrolladas por diferentes empresas.

### Owlet by Schreder

Schreder es una empresa con sede en Bélgica y representación internacional, es referenciada como una compañía que brinda soluciones en iluminación con sus productos. Owlet es un proyecto de iluminación que actualmente se encuentran en funcionamiento en ciudades como: Ajman, Braga, Bra, Houthalen-Helchteren, Lisbon, entre otros países. El proyecto implementa sensores de movimiento, luz, velocidad y dirección interconectados a una red inalámbrica, donde un sistema de control se encarga de configurar la iluminación en las luminarias de acuerdo a parámetros configurables. (Schreder, 2017).

La implementación de sensores de alto costo, así como los múltiples algoritmos funcionando al mismo tiempo generan una solución sustentable en países europeos; nuestro proyecto está configurado para no ocupar sensores de iluminación, puesto que el procesamiento de imagen realizaría la función de detección mediante una cámara.

### Control Centralizado, Bticino

Bticino ofrece una solución integral para poder solventar la necesidad de hacer más eficientes los sistemas de iluminación, hace uso de un protocolo de comunicación creando una red donde varios sensores pueden ser integrados a la misma, la información obtenida es procesada y se puede determinar bajo determinados patrones, que luminaria debe ser encendida y con qué intensidad; la solución ofrece una interfaz que también permite monitorear los estados de los sensores y modular la iluminación de manera individual (bticino, 2017).

El uso de sensores de tecnología de innovación y alto costo, así como el pago de licencias de uso de software son algunos de los elementos que marcarían la diferencia entre este producto y el proyecto que estamos planteando.

## II.II Luz e Intensidad Luminosa

La luz es una forma de radiación electromagnética que fluye en forma de ondas en cualquier medio con una dirección determinada y solo es perceptible cuando interactúa con la materia. (Sirlin, 2017).

Algunas características que la luz posee son: Amplitud, Longitud de onda, Velocidad y Frecuencia.

La luz visible forma parte del llamado “espectro electromagnético” que es el modo en que se ordena la energía radiante según su longitud de onda, como podemos ver en la Figura 1, la luz es visible o la percibida por el ojo humano está comprendida en las longitudes de onda de los 380 nanómetros hasta los 780 nanómetros.

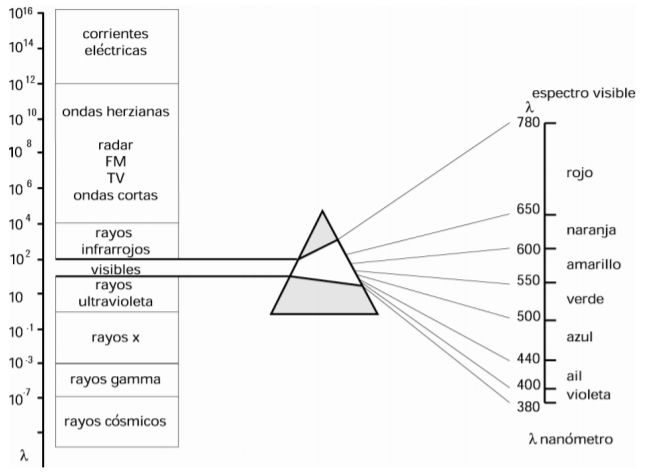


Figura 1. Espectro electromagnético y la luz visible.

La intensidad luminosa está definida como la cantidad de luz emitida por una fuente que incide sobre una superficie en un ángulo especifico y cuya magnitud es expresada en candelas.

## II.III Generalidades del procesamiento de Imágenes

Una imagen digital está concebida como un arreglo bidimensional, cuyo valor proviene de un proceso de muestreo, existen varios tipos de muestreo como: Densidad de muestreo, muestreo por medidas de área, muestreo por medidas de distancia entre otros métodos.

La intensidad luminosa puede ser expresada como el valor promedio de la sumatoria de los pixeles que contiene la imagen en escala de grises (Young , Gerbrands, & van Vliet, 2017).

Dado que la imagen es concebida como un arreglo bidimensional, las operaciones matriciales de algebra lineal nos permitirán realizar operaciones y procesamientos para obtener datos de las imágenes.

## II.IV Microcontroladores

Texas Instruments (TI) es una empresa estadounidense que desarrolla y comercializa semiconductores y tecnología digital, siendo el tercer mayor fabricante de semiconductores a nivel mundial. Dentro de los procesadores digitales que ofertan se encuentra la familia de los MSP430, son microcontroladores de dieciséis bits que han sido diseñados para trabajar en aplicaciones de bajo consumo de energía (3.3 V y 230 uA), dependiendo de las características de su arquitectura son agrupadas en familias que distinguen su distribución funcional.

De forma básica un microcontrolador puede determinarse en cinco etapas o bloques funcionales, la Figura 2 muestra un diagrama a bloques de la constitución o arquitectura que obedece todo sistema llamado microcontrolador. Las cinco etapas son: la unidad de procesamiento central (CPU), la memoria primaria, dispositivos de entrada salida, buses (direcciones, datos y control) y una unidad de reloj.

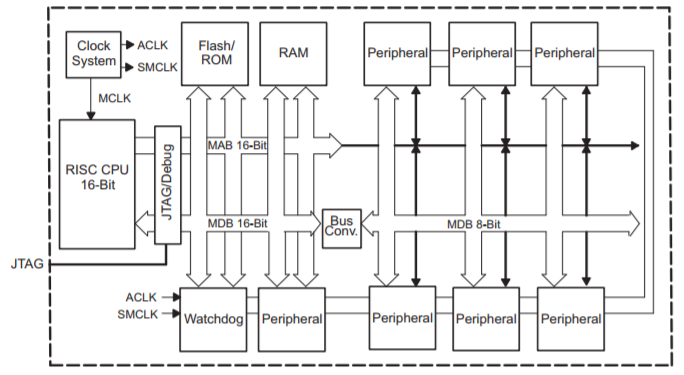


Figura 2. Arquitectura del microcontrolador MSP430G2553.

La primer etapa es la unidad que aporta la capacidad de cómputo del sistema; las memorias pueden determinarse como los registros, memorias de corto plazo, una memoria de datos, que es la que almacena información mientras el microcontrolador funciona y la memoria del programa que tiene la característica de poder permanecer la información guardada en el sistema aunque no esté encendido. Los dispositivos de entradas y salidas se refieren a las conexiones exteriores de los microcontroladores, son conocidas como GPIO (General Purpose Input/Output) y por lo general se agrupan en puertos de ocho terminales. Los buses son los conductores que transmiten diferente tipo de información como las direcciones, los datos de control, las instrucciones y los valores de variables. Finalmente, la unidad de reloj es variable en cada sistema, algunos pueden tener osciladores controlados digitalmente independientes de cristales externos, otros requieren de un cristal para multiplicar la oscilación y generar una señal de reloj, entre más sistemas generadores de señal.

Los microcontroladores de TI utilizan un software que lleva a cabo la compilación y programación en sus microcontroladores, este software se llama Code Composer Studio y su programación utiliza un lenguaje en C básico. Una ventaja para el uso de estos microcontroladores es que no requieren de una licencia para el uso del software ni para la programación de los mismos. (Texas Instruments, 2017).

La familia de los microcontroladores MSP430G2553 cuentan con las herramientas y la arquitectura capaces para soportar las necesidades de nuestro proyecto, aunado a su bajo costo y soporte. (Texas Instruments, 2017).

## II.V Protocolo de Comunicación UART

El protocolo UART (Universal Asynchronous Receiver-Transmitter) es un estándar de comunicaciones que se implementa en una amplia gama de aplicaciones, muchos dispositivos electrónicos tienen el protocolo embebido en su arquitectura. Tiene un marco de datos específico para transmitir la información de un lugar a otro. La Figura 3 muestra la estructura básica de los mensajes bajo este protocolo.

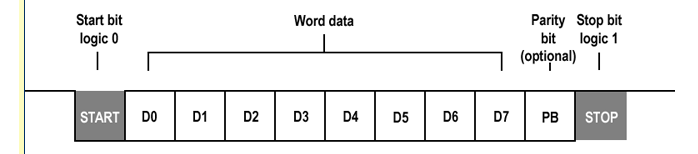


Figura 3. Estructura de un mensaje del protocolo UART.

## II.VI PWM

El PWM (Pulse Wide Modulation) por sus siglas en inglés, se refiere a una modulación del ancho de pulso de una señal. Esta señal cuadrada unipolar y cuya frecuencia se define desde el inicio de su diseño. La propiedad interesante del empleo de una señal de este tipo, es que la señal cuadrada de salida puede ser variada en cuanto la amplitud de la onda cuadrada, en el ciclo de trabajo completo, como puede notarse en la Figura 4. Este tipo de señales suelen ser utilizadas como salidas de control y precisamente es lo que usaremos en nuestro proyecto, el ancho del pulso se verá afectado en función a un algoritmo de control.



Figura 4. Modulación de una señal PWM.

## II.VII OpenCV

OpenCV es una librería de visión artificial, en ella se encuentran implementados diversos algoritmos de procesamiento de imagen que son utilizados en infinidad de aplicaciones con intensiones comerciales y de investigación, las cuales son permitidas por su licencia de libre propósito (OpenCV, 2017).

La librería se puede obtener en distintas versiones a través de su página oficial: opencv.org. Los algoritmos pueden ser utilizados en desarrollos con diversos lenguajes de programación como C++, C#, java, Python, entre otros; es compatible con muchas plataformas como NetBeams, QtCreator, Visual Studio, etc. Lo anterior, hace de OpenCV una librería adecuada para la implementación de nuestro proyecto.

## II.VIII Controladores

Uno de los objetivos de nuestro proyecto es el de implementar un controlador que sea capaz de regular una señal a modo que una lámpara pueda iluminar a cierto nivel esperado, para esto es necesario implementar un sistema de control. La Figura 5, nos muestra un diagrama a bloques de un sistema controlado. Como entrada existe un valor de referencia o un valor deseado de una variable física, este valor se compara con el estado actual de la variable y la diferencia es procesada por un bloque de control el cual determina la magnitud de una señal que actúa sobre una planta o sistema controlado, perturbando la condición de la variable física que es medida por algún sensor.

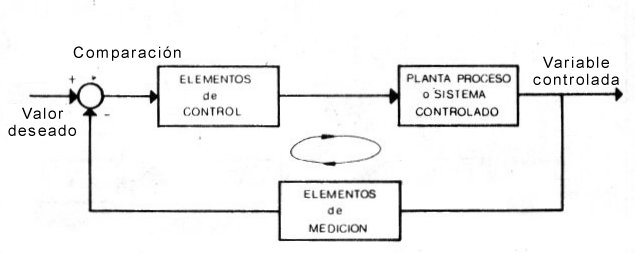


Figura 5. Diagrama a bloques de un sistema controlado retroalimentado.

Existen diversos tipos de controladores en la actualidad y es necesario verificar las condiciones que se encuentran inmersas en todo sistema, ya que es en función de lo anterior que un tipo de controlador responderá de mejor manera que otro, así bien, es necesario primero construir nuestro prototipo para poder delimitar las características del sistema y con ello inferir el tipo de controlador adecuado.

Tradicionalmente los controladores del tipo PID (Proporcional Integral Derivativo) suelen ser los más seleccionados de los sistemas controlados, sin embargo, en muchas ocasiones el control PI (Proporcional Integral) suele ser suficiente para obtener una respuesta deseada en el sistema. En las siguientes líneas se mostrará la teoría sobre un controlador PID que engloba las combinaciones que nos permitirían obtener un controlador adecuado para nuestro sistema.

### II.VIII.I Controlador PID

En la teoría clásica del control, un elemento clave es el modelado matemático del sistema a controlar. Cuando se adquiere el modelo del sistema existen diversas metodologías para diseñar y construir un controlador para dicho sistema.

Un controlador del tipo Proporcional-Integral-Derivativo multiplica el error o diferencia entre el estado actual de la variable física y el estado deseado, por una constante (constante proporcional) más la multiplicación entre una constante integral por la integral de la diferencia, más la multiplicación de la derivada del error por una constante derivativa. (Hernandez, Silva, & Carrillo, 2013).

La función de transferencia de un controlador PID esta descrita en la ecuación 1.1

 1.1

En tiempo continuo la expresión 1.1 puede denotarse como 1.2.

 1.2

La ecuación 1.1 expresada en tiempo discreto se muestra en 1.3.

 1.3

### II.VIII.II Controlador Difuso

En contraste con la teoría del control clásica, un control difuso puede diseñarse sin modelar matemáticamente un sistema, se enfoca en obtener una comprensión intuitiva de cómo controlar el proceso de la mejor manera, usando la heurística y reglas para controlar la planta. (Passino & Yurkovich, 1998).

Visualmente la arquitectura de un controlador difuso se describe en la Figura 6.



Figura 6. Arquitectura de un controlador difuso.

Un controlador difuso está compuesto básicamente de cuatro bloques:

1. Reglas. Las reglas guardan la información para controlar de la mejor manera el proceso.
2. Mecanismo de Inferencia. Evalúa cuales son las reglas aplicables para las circunstancias actuales en el proceso.
3. Interface de Fuzzificacion. Modifica la entrada para que pueda ser interpretada y comparada con las reglas de control.
4. Interface de Defuzzificacion. Convierte los resultados obtenidos en el mecanismo de inferencia a la entrada de la planta.

Para el diseño de un control difuso, es necesario juntar información de cómo debe actuar el controlador en un sistema de lazo cerrado. La información puede deducirse desde una índole humana al observar los movimientos de un operador o bien entendiendo la dinámica del sistema y deduciendo así la serie de reglas que se deben de cumplir en determinadas circunstancias.

Las reglas toman como referencia la salida de la planta y la referencia de entrada, en función de estas condiciones se describe que entrada debe aplicarse a la planta para modificar o mantener las condiciones actuales.

La Tabla 1 nos muestra un ejemplo de cómo describir un bloque de reglas para un mezclador de agua.



Tabla 1. Reglas difusas para un control térmico para una mezcladora de agua.

Existen muchos métodos de Fuzzificación o bien métodos para crear conjuntos difusos. (Zimmermann, 2001), en esta sección y con fines descriptivos se muestra un ejemplo para crear conjuntos difusos con funciones triangulares de pertenencia, para un mezclador de agua.

Las funciones de pertenencia representan la información sobre las condiciones o variables que yacen a la entrada del sistema, la fuzzyficación transforma el valor a un “grado de pertenencia” para ser procesado en el mecanismo de inferencia.

La Figura 7 nos muestra una clasificación de temperaturas orientadas al control de un mezclador de agua determinado de manera subjetiva.



Figura 7. Clasificación de temperaturas.

En la Figura 8 se observan los conjuntos difusos que forman parte del sistema, el cual recibe el nombre de función de membresía, este proceso es denominado *fuzzificación.*

Los elementos apreciados son los siguientes:

* El eje X es la temperatura en grados Celsius.
* El eje Y representa el grado de membresía, describe cuantitativamente la función de membresía
* La forma de la función de membresía es triangular. En los métodos de fuzzificación la forma de la función muestra variaciones.
* Existen puntos que pertenecen a más de un conjunto difuso.



Figura 8. Funciones de pertenencia para un mezclador de agua.

El grado de membresía asociado a cada conjunto difuso es calculado usando interpolación lineal, llamado *grado de pertenencia.* Para este ejemplo, el grado de pertenencia se calcula con las siguientes ecuaciones:

1.4

1.5

1.6

Con las reglas asociadas y los grados de pertenencia en función de la entrada, el mecanismo de inferencia generará una salida en cuanto al nivel de apertura que deberán tener las válvulas. Para ello se generan conjuntos difusos en función a la apertura de las válvulas.



Figura 9. Conjuntos difusos para la válvula de agua caliente.



Figura 10. Conjuntos difusos para la válvula de agua fría.

La Defuzzificacion es el método por el cual un conjunto difuso es convertido en un número real y también existen muchos métodos para ello (Passino & Yurkovich, 1998), en continuidad con nuestro ejemplo, usamos el método del centro de gravedad y el criterio del máximo para denotar el proceso.

El método del centro de gravedad, es un término derivado por la teoría de la mecánica clásica, dicho método permite encontrar aquel punto de un centroide en donde todas las fuerzas que actúan sobre centroide producen un momento resultante nulo. (Beer, Johnston, Eisenberg, & Mazurek, 2007).

En lógica difusa, el método del centro de gravedad busca encontrar aquel punto de equilibrio del centroide formado por el grado de pertenencia de las entradas del sistema, siendo este punto la salida de control.

1.7

La ecuación 1.7 es la ecuación característica para encontrar el valor del centro de gravedad. ‘y’ es nuestra variable de control, µ(y) es la función de pertenencia para el conjunto de salida Y, R es el número de reglas que están inmersas en nuestro sistema.

## II.IX GitHub

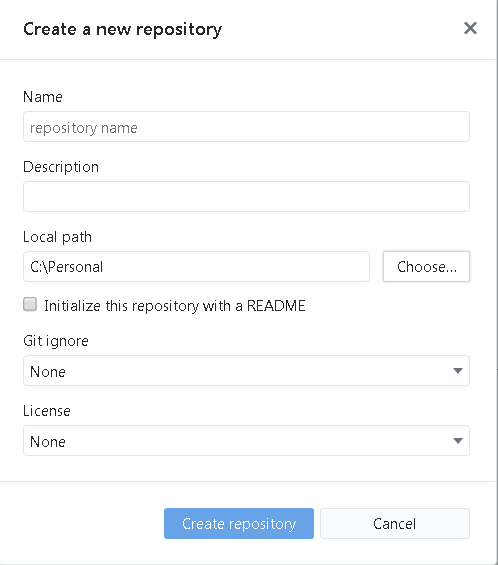
GitHub es una plataforma que permite llevar un control de versiones en un proyecto en desarrollo, permite hacer uso de una plataforma en línea con cualquier explorador web o bien usando una interfaz gráfica de uso libre. Utiliza como base Git (The Git Project, 2018), por lo cual las funcionalidades son las mismas, entre las características más importantes de este software podemos encontrar:

* El proyecto está guardado en un espacio virtual, el cual puede ser accedido por múltiples desarrolladores a la vez sin afectarse el contenido.
* Pueden crearse varias ramificaciones de desarrollo.
* El historial de cambios y avances se guarda en el espacio virtual, puede accederse a diversos puntos en el desarrollo con el fin de recuperar cambios.
* En cualquier momento los cambios realizados se pueden subir a la ramificación principal y todos los desarrolladores pueden trabajar sobre la misma rama.

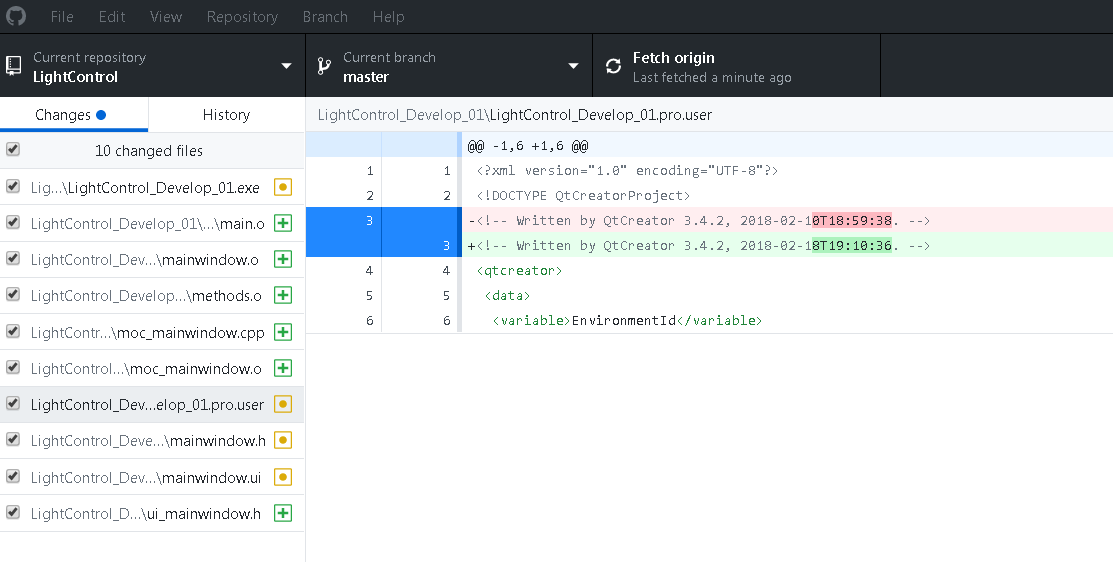
### II.IX.I Instalación y configuración inicial de GitHub

Los pasos a seguir para instalar la aplicación y hacer uso de ella para controlar las versiones de un desarrollo son las siguientes:

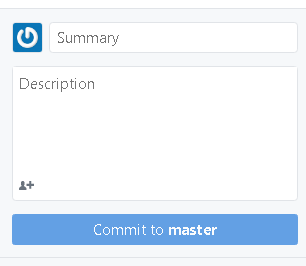
* Navegar a la página oficial de GitHub en el siguiente link: <https://github.com/>
* Crear una cuenta usando un correo electrónico y una contraseña.
* Bajar el software a través del siguiente link: <https://desktop.github.com/>
* Dejar que el instalador seleccione las opciones por default, aceptar licencia y finalizar la instalación.
* Abrir el programa y acceder con la cuenta de correo y la contraseña antes creada.
* Crear un repositorio nuevo:
  + File -> New repository
  + Se abrirá la siguiente ventana:



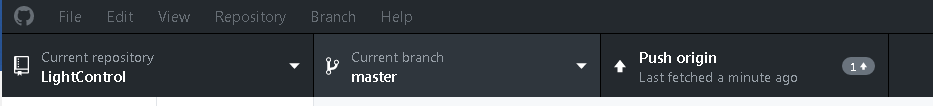
* + Asignar un nombre, el lugar donde se guardara el desarrollo y alguna descripción del mismo.
  + Dar clic en crear repositorio.
  + A partir de este momento cualquier cambio que se lleve a cabo en el directorio se verá reflejado en la interfaz, si se agrega un archivo la interfaz lo identificara, si un archivo es modificado, la interfaz mostrará los cambios realizados en el archivo.
  + En la siguiente imagen se muestra el ejemplo de un cambio en un archivo:



* + El archivo seleccionado se muestra sombreado en el navegador, los cambios son mostrados y son señalados, en rojo se muestra el contenido antes de ser modificado y en verde el nuevo contenido.
  + Los archivos con el símbolo de adición en verde son los archivos nuevos agregados al directorio de control.
  + Para agregar los archivos al repositorio, primero es necesario dar clic en el botón de “commit to master” que se encuentra en la parte inferior de la ventana de cambios y se muestra en la siguiente imagen:



* + Se recomienda hacer un resumen de los cambios realizados antes de pulsar el botón, estos campos aparecerán en el historial de cambios del desarrollo.
  + Finalmente para que los cambios estén disponibles en el repositorio se debe dar clic en el botón de: “Push origin” ubicado en la parte superior de la interfaz a un lado del nombre de la rama y se muestra en la siguiente imagen:



GitHub tiene una página de soporte que puede consultarse en cualquier momento, además cuenta con muchos foros informativos y con especificaciones y guías de uso, lo anterior puede consultarse en el siguiente link: <https://services.github.com/>.

# Metodología

El desarrollo de este proyecto fue controlado en todas las secciones a través de GitHub. GitHub es una interfaz de control de versiones que usa como base Git. Las características del control de versiones han sido expuestas en la sección del marco teórico de este documento, no obstante se expone el link de GitHub que contiene información adecuada para poder hacer uso de la aplicación: <https://help.github.com/>.

El sistema operativo en donde se desarrolló el código fue Windows 7.

La interfaz gráfica fue desarrollada en la plataforma QT Creator versión 3.4.2 como lo muestra la Figura 11:

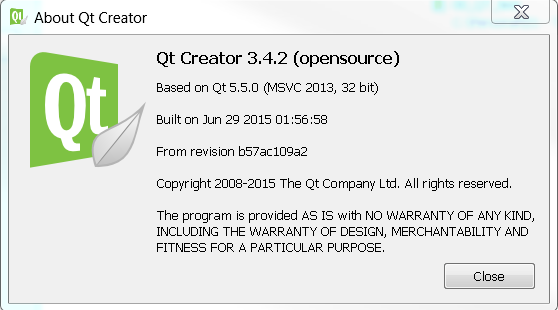


Figura 11. Ventana "acerca de" en QT Creator.

Qt Creator es un entorno de desarrollo integrado IDE (Integrated Development Enviroment) de distribución gratuita bajo ciertos estándares de desarrollo. Permite el desarrollo de Interfaces graficas de usuarios GUI (Graphical User Interface), es un depurador de código, ofrece soporte para múltiples plataformas, entre ellas Windows y mucho desarrollo puede ser encontrado en páginas web.

La programación del microcontrolador MSP430G2553 se desarrolló en la interfaz Code Composer Studio (CCS) versión 7.1.0.00015 que se muestra en la Figura 7.

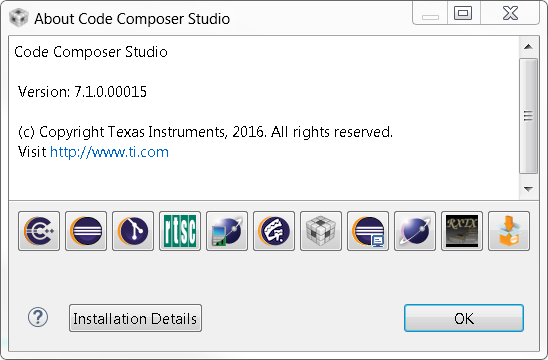


Figura 12. Ventana "acerca de" de CCS.

Texas Instruments ofrece la distribución gratuita de esta interfaz de programación para sus dispositivos electrónicos, el sistema cuenta con varias herramientas específicas que permiten la edición de texto, debuggeo de los sistemas y código integrado, terminal de comunicaciones, mapeo de registros y memorias, entre otras.

## III.I Integración de OpenCV en Windows 7 y Qt Creator

La integración de la librería en la IDE Qt Creator se reduce a unos cuantos pasos. Se parte de la premisa de que se ha instalado adecuadamente la versión especificada de Qt Creator.

1. Descargar la librería OpenCV. (OpenCV, 2018).
2. Descomprimir la librería en un directorio sin caracteres especiales (símbolos o guiones) ni espacios, se recomienda C:\OpenCV.
3. Agregar al path del sistema la carpeta donde se encuentran las librerías dinámicas (DLL) como lo muestra la Figura 8:

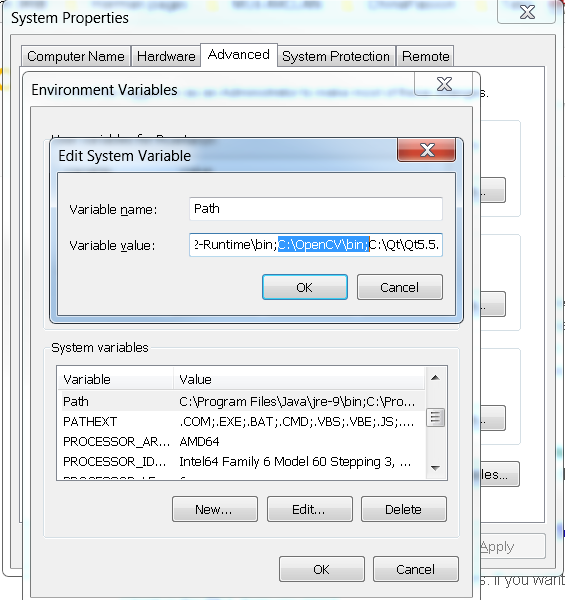


Figura 13. Ventana de edición de variables de entorno.

1. Crear un proyecto nuevo en Qt Creator.
   * File -> New
   * Application -> Qt Widgets Application
   * Asignar nombre y directorio donde se guardara el Proyecto.
   * Seleccionar el compilador por default (MinGW).
   * Asignar nombre de los archivos fuente.
   * Finalizar el Wizard.
2. Al crear un nuevo proyecto y seleccionar las opciones por default, Qt Creator genera cinco archivos base:
   * Nombre\_del\_proyecto.pro
   * mainwindow.h
   * main.cpp
   * mainwindow.cpp
   * mainwindow.ui
3. Antes de incluir las librerías de OpenCV en el código, es necesario agregar el path de las librerías y los archivos de acceso a ellas en el archivo de configuración (Nombre\_del\_proyecto.pro), la Figura 9 nos muestra los archivos que fueron usados en nuestro proyecto.

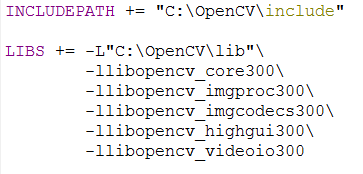


Figura 14. Integración de librerías y archivos de acceso en Qt Creator.

1. Finalmente, en los archivos del código donde se requiera usar las funciones de OpenCV se debe agregar la librería:

#include <opencv2/opencv.hpp>

## III.II Desarrollo de la interfaz gráfica

La interfaz gráfica (GUI) fue diseñada para operar en base a dos propósitos, el primero fue demostrar la operación de la obtención de la intensidad luminosa mediante la segmentación de la imagen en áreas específicas de la misma, este proceso se modela en el diagrama de flujo contenido en la Figura 15.

  
Figura 15. Diagrama de flujo de la obtención de la intensidad luminosa.

Al recibir el comando para comenzar el algoritmo, el software esta diseñado para pedir una imagen de entrada. El usuario seleccionará la imagen dentro de un directorio. El software evaluará si la imagen seleccionada es compatible en cuanto al tamaño y formato.[[1]](#footnote-1) En caso de que la imagen no sea compatible, el software mostrara un mensaje de error, aludiendo al formato de la imagen. Si la imagen es compatible, la imagen será procesada, cambiando su espacio de color a escalas de grises para luego implementar el algoritmo para deteccion de intensidad luminosa.

El segundo y principal propósito de la interfaz gráfica es el de automatizar el control de iluminación, por lo cual, algunas características son agregadas. La Figura 16 nos muestra el diagrama de flujo que se desarrolla para controlar la iluminación de forma automática.

Se agrega la capacidad de soportar instancias de comunicación serial. Dos puertos de comunicación son creados, uno para mandar la señal de control a un microcontrolador (MCU), otro puerto de comunicación serial es usado para leer las imágenes utilizando una cámara de video externa.

Se agrega el uso de un temporalizador, este servicio nos permitirá llamar la ejecución del algoritmo cada cierto intervalo de tiempo. Se agrega también el controlador Fuzzy quien usara la intensidad luminosa como variable de entrada.



Figura 16. Proceso de control de iluminación (GUI).

El proceso de control de iluminación automático crea un temporalizador el cual invoca una secuencia cada cierto intervalo de tiempo.

La secuencia, primeramente, lee y verifica la imagen recibida por la cámara externa, cambia el espacio de color y obtiene la intensidad luminosa por segmentación de la imagen, verifica que este valor de intensidad sea un parámetro valido así como el valor de intensidad luminosa deseado (parámetro que debe ser asignado en la interfaz, de no asignarse, valores por default son asignados) y procesa este valor en el controlador fuzzy. Finalmente, la salida de control es enviada al microcontrolador mediante un puerto de comunicación serial. Si el puerto de comunicación serial del microcontrolador presenta alguna clase de error, el proceso se detiene enviando un mensaje de error.

### III.II.I Operación manual: Obtención de la intensidad luminosa

La interfaz gráfica cuenta con una sección donde se puede obtener los niveles de intensidad de una imagen de forma manual, el proceso se completa presionando cuatro botones. Como es un proceso secuencial, los botones son habilitados y deshabilitados de acuerdo con el paso actual, es decir, no se podrá presionar ningún otro botón más que el subsecuente. La Figura 12 nos muestra la sección del proceso manual para la obtención de la intensidad luminosa en la GUI.

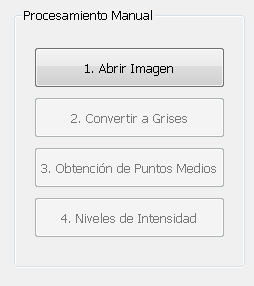


Figura 17. Sección: "Proceso Manual" en la interfaz gráfica.

#### Botón “Abrir Imagen”

El botón “Abrir Imagen” tiene la función de obtener y mostrar una imagen en algún directorio especificado por el usuario, la imagen puede ser de cualquiera de los siguientes tipos o extensiones de archivos:

\*.bmp, \*.pbm, \*.pgm, \*.ppm, \*.sr, \*.ras, \*.jpg, \*.jpeg, \*.jpe, \*.jp2, \*.tiff, \*.tif, \*.png

Si la imagen seleccionada es compatible esta será mostrada en una ventana anexa con el nombre “Imagen Seleccionada”.

#### Botón “Convertir a Grises”

El objetivo del botón es de cambiar el espacio de color de la imagen seleccionada a escala de grises, la conversión de la imagen a este espacio de color nos permite encontrar los niveles de intensidad luminosa en la misma.

#### Botón “Obtención de Puntos Medios”

La propuesta de mejora en nuestro proyecto se encuentra en la selección de zonas en la imagen para encontrar un índice de intensidad luminosa, el objetivo de la propuesta es encontrar un valor que nos permita caracterizar la intensidad luminosa de la imagen, sin analizar todos los pixeles que comprenden la imagen, lo cual nos ahorrara tiempo de procesamiento.

El botón de “Puntos Medios” nos permite encontrar en la imagen, los puntos medios de las zonas a analizar en ella, y se visualizan en la agrupación “Obtención de Puntos Medios” como lo muestra la Figura 13.

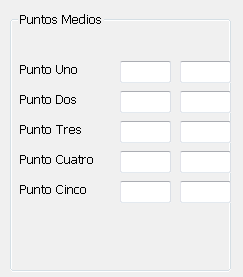


Figura 18. Agrupación: "Puntos Medios".

#### Botón “Niveles de Intensidad”

La interfaz nos permite asignar el tamaño del área y como primer elemento antes de obtener los niveles de intensidad adquiere los valores del grupo visual: “Tamaño del área” que se muestra en la Figura 14, si no existe un valor en los elementos de texto asigna valores predeterminados en código.

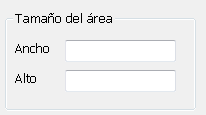


Figura 19. Conjunto visual: "Tamaño del área".

Finalmente, se lleva a cabo el análisis de las zonas características y se obtiene el valor promedio de la intensidad luminosa en la imagen, los valores son expuestos en la interfaz con el grupo visual “Niveles de Intensidad” como lo muestra la Figura 15.

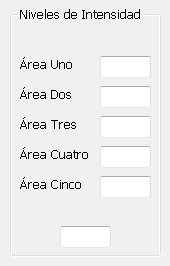


Figura 20. Conjunto visual: "Niveles de Intensidad".

### III.II.II Operación automática: Control de intensidad luminosa

La operación automática se basa en el proceso expuesto en la Figura 11. Los pasos que son ejecutados se resumen a continuación:

Al presionar el botón “Habilitar Temporalizador”:

* Habilitar un temporalizador que llame una secuencia cada determinado tiempo.
  + Configurar y abrir un puerto de comunicación serial.

Secuencia:

* Obtener una imagen de una cámara.
* Verificar características y validar que la información obtenida de la cámara pertenece a una imagen; si existe algún error mandar un mensaje característico y finalizar el ciclo.
* Llamar la secuencia manual descrita en III.II.I para procesar la imagen y obtener el índice de intensidad luminosa.
* Obtener o asignar el valor predeterminado de la intensidad luminosa deseada.
* Calcular la señal de control enviando el valor deseado de intensidad luminosa y el valor obtenido en III.II.I.
* Verificar que la conexión entre el microcontrolador y la interfaz se encuentre habilitada, en caso de haber algún error, finalizar el ciclo.
* Mandar la señal de control al micro a través del puerto de comunicación serial.
* Esperar el siguiente ciclo.

## III.III Desarrollo de la aplicación en el microcontrolador

El desarrollo del código del microcontrolador MSP430G2553 se implementó en la interfaz de usuario Code Composer Studio, distribuida por TI de forma gratuita.   
El sistema tiene como propósito:

* Recibir y procesar información vía UART.
* Generar una señal digital de control (PWM).

El protocolo de UART fue configurado para habilitar una comunicación a 9600 baudios con ocho bits de datos de transmisión y un bit de paro.

La señal de control fue configurada para generar un PWM de 1 KHz usando un pin de salida del microcontrolador regido por un temporalizador del mismo dispositivo (TA0).

En la Figura 16 podemos observar la secuencia de programación desarrollada en el microcontrolador.



Figura 21. Secuencia de programación del microcontrolador.

### III.III.I Configuración de los registros del microcontrolador

En la primera sección del diagrama de la figura dieciséis podemos notar la configuración de los registros del microcontrolador para cada módulo que fue habilitado. El acceso y nombre de los registros, así como la descripción de sus funcionalidades se encuentra detallado en el manual de usuario de la familia del microcontrolador. (Texas Instruments Incorporated, 2013).

#### Configuración del reloj

El microcontrolador MSP430G2553 soporta diferentes tipos de osciladores como fuente, entre ellos se cuenta con un llamado: oscilador interno controlado digitalmente (DCO por sus siglas en ingles), el cual es usado y configurado en nuestro dispositivo, el modulo provee tres salidas: el reloj principal (MCLK), el reloj sub-principal (SMCLK) y el reloj auxiliar (ACLK) las cuales fueron configuradas para generar una frecuencia de 1 MHz. El diagrama a bloques que caracteriza el funcionamiento del módulo de reloj se muestra en la Figura 17.

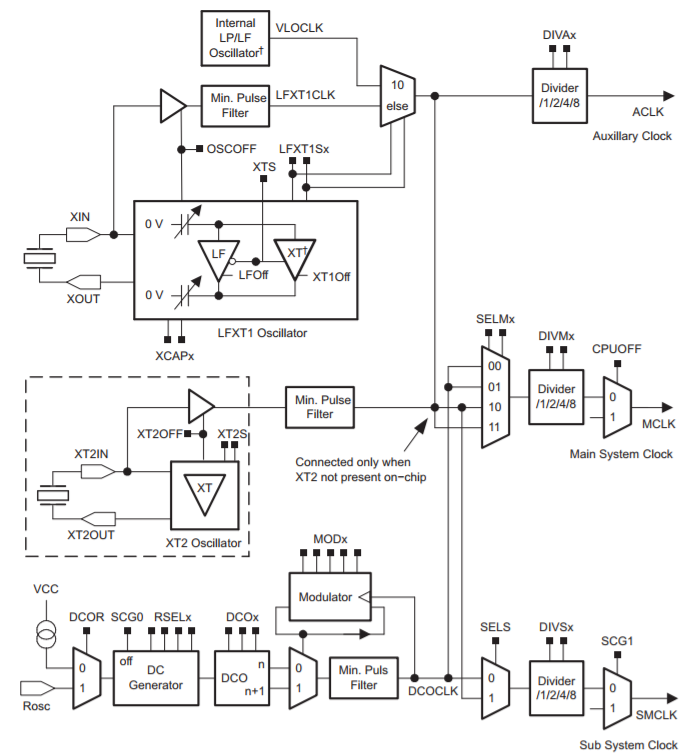


Figura 22. Módulo del reloj en microcontrolador MSP430G2553.

#### Configuración de las entradas y salidas digitales

El propósito de nuestro desarrollo requiere configurar un par de terminales de salida para establecer una comunicación serial con la interfaz gráfica, para ello se hace uso del diagrama de configuración de salidas del microcontrolador en cuestión que se encuentra expuesto en la Figura 18.

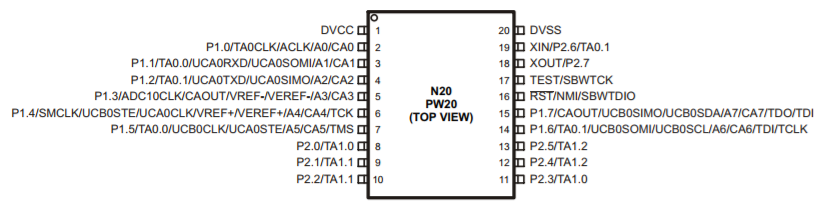


Figura 23. Diagrama de configuración de salidas del MSP430G2553.

Las terminales 1 y 2 del puerto 1, son las que están eléctricamente configuradas para soportar la configuración serial en el dispositivo, en función de lo anterior se habilitan los registros correspondientes en el puerto 1.

Aunado a la configuración necesaria en el puerto para la comunicación serial, se utiliza una terminal digital del mismo puerto para generar una señal de control la cual será manipulada por un temporalizador del mismo microcontrolador, de acuerdo con el diagrama de configuración de salida se elige la terminal 6 para poder generar la señal de control usando el temporalizador 0.

#### Configuración del temporalizador

El temporalizador en el MSP430G2553 es un contador de 16-bits, tiene tres registros de captura y comparación (CCR) que puede soportar simultáneamente y pueden configurarse para generar salidas de control tipo PWM. El funcionamiento de este módulo se modela en el diagrama a bloques de la Figura 19.

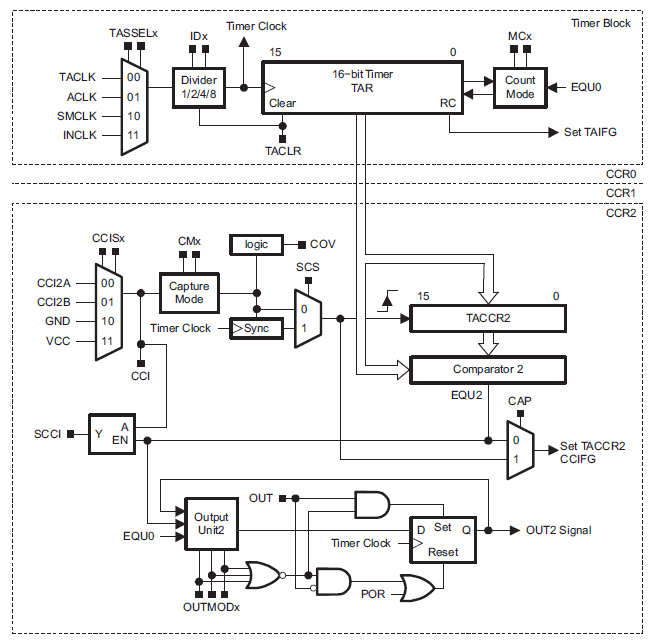


Figura 24. Diagrama a bloques del temporalizador.

El temporalizador recibe la entrada del SMCLK el cual fue configurado para generar una señal de 1 MHz y el registro de captura y comparación 0 (CCR0) se habilitó para modular una frecuencia de 1 KHz. Los cálculos para la obtención del valor de los registros se basan en la fórmula 1.4 que se expresa a continuación:

 1.4

El control habilitado de este temporalizador con los registros CCR0 y CCR1 se muestra en la Figura 20 con el modo 2 (Toogle/Reset).

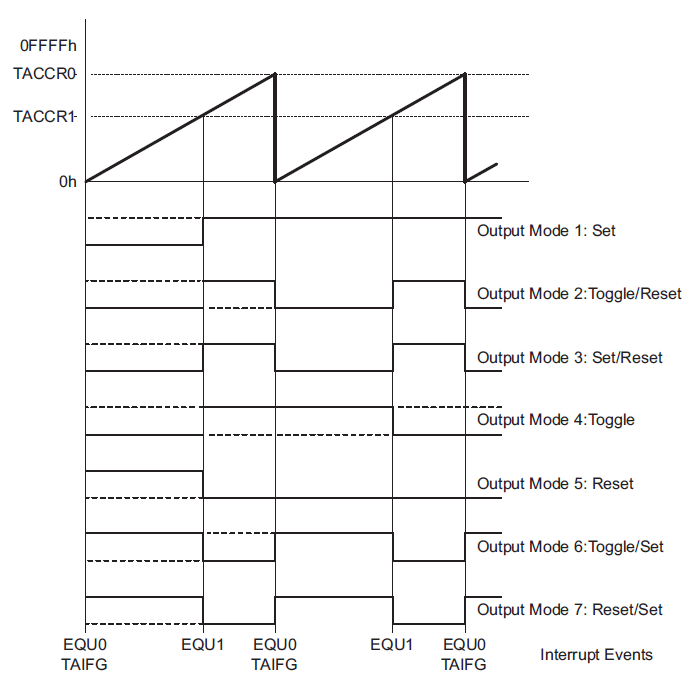


Figura 25. Modos de salida usando el temporalizador.

#### Configuración de los registros de la interfaz de comunicación UART

El microcontrolador MSP430xGxx posee una interfaz de comunicación serial la cual puede configurarse para soportar protocolos de comunicación como: SPI, I2C y UART.

La Figura 21 muestra el diagrama a bloques de la interfaz en modo UART, la interfaz transmite y recibe caracteres a una velocidad asíncrona a otro dispositivo, la velocidad de comunicación es la misma en la recepción y la transmisión.

La configuración realizada para el uso del protocolo UART en nuestro dispositivo es:

* Modo asíncrono
* Sin paridad
* Primer dato en llegar es el menos significativo
* Datos de 8 bits
* Un bit de paro
* Selección del SMCLK como fuente de reloj
* Velocidad de 9600 baudios

El cálculo del registro para generar una velocidad de 9600 baudios se muestra en la ecuación 1.5.

 1.5

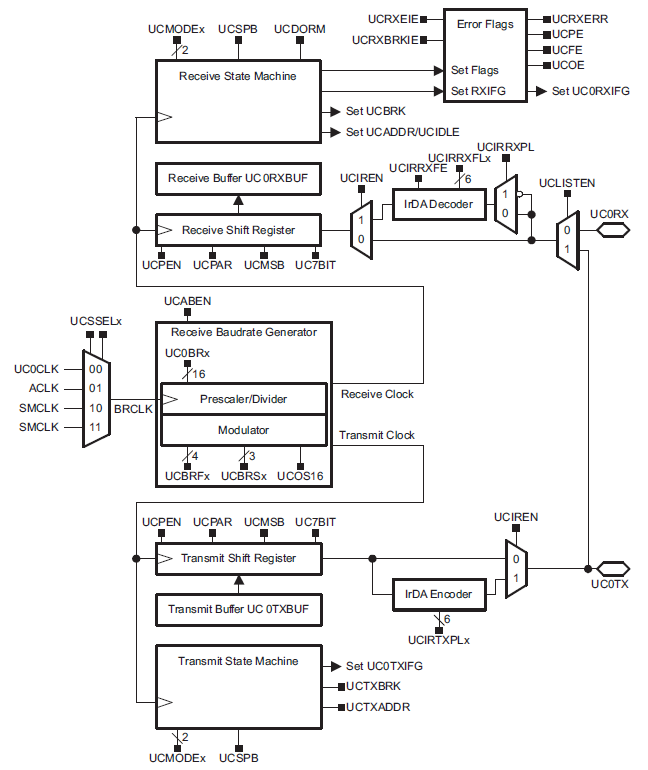


Figura 26. Diagrama a bloques de la interfaz del MCU en modo UART.

### III.III.II Procesamiento de los datos

El procesamiento de los datos se basa en el diagrama a bloques descrito en la Figura 16, en la interrupción generada por la entrada de datos seriales se guarda el valor recibido y se manda a una función que validará y procesará el tipo de dato.

La función primero valida si el carácter recibido nos da la indicación de procesar la señal de control o guardar el dato.

En caso de guardar el dato, la función verifica que el valor se encuentre dentro de los limites usados, de no ser así invalida la trama e inicializa nuevamente; si el valor está dentro de los limites, guarda el dato.

En caso de procesar la señal de control, la función transforma el valor recibido en cuentas del temporalizador, valida y actualiza la cuenta del registro de captura y comparación CCR1 provocando un cambio en el PWM de la terminal de salida.

# Resultados y Discusiones

# Conclusiones

# Apéndice I. Código implementado en el microcontrolador

**#include** <msp430g2553.h>

**#include** <stdlib.h>

**#include** <string.h>

/\*

\* Defines section

\*/

**#define** FREQ 125

**#define** MINIMUM\_RATE 5

**#define** MAXIMUM\_RATE FREQ-5

**#define** MAXIMUM\_ARRAY\_DATA 2

**#define** ASCII\_MIN\_NUMBER 47

**#define** ASCII\_MAX\_NUMBER 58

/\*

\* Function declaration

\*/

**void** **processData**(**char**);

/\*

\* Struct declarations

\*/

**typedef** **struct** data{

**char** data\_input[2];

**unsigned** **short** **int** data\_count;

} data;

/\*

\* Global variables

\*/

**static** data input\_data;

**void** **main**(**void**) {

//Watchdog configuration

WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

//Clock configuration

DCOCTL = CALDCO\_1MHZ;

BCSCTL1 = CALBC1\_1MHZ;

BCSCTL2 = SELM0 + DIVS\_0;

BCSCTL3 = LFXT1S\_2;

//IO configuration

P1DIR = BIT6;

P1SEL = BIT6;

P1OUT &= ~BIT6;

/\* Configure Pin Muxing P1.1 RXD and P1.2 TXD \*/

P1SEL |= BIT1 + BIT2 ;

P1SEL2 = BIT1 + BIT2;

P1DIR |= BIT1;

UCA0CTL0 = 0x00;

/\* Configure \*/

UCA0CTL1 |= UCSSEL\_2; // SMCLK

UCA0BR0 = 104; // 1MHz 9600

UCA0BR1 = 0; // 1MHz 9600

UCA0MCTL = UCBRS\_1; // Modulation UCBRSx = 1

/\* Take UCA0 out of reset \*/

UCA0CTL1 &= ~UCSWRST;

//Timer A0 configuration

TA0CTL = TACLR;

TA0R = 0;

TA0CCR0 = FREQ;

TA0CCR1 = MAXIMUM\_RATE;

TA0CCTL0 = CCIE;

TA0CCTL1 = OUTMOD\_2;

TA0CTL = TASSEL\_2 + MC\_1 + ID\_3;

//Interrupt configuration

IE2 |= UCA0RXIE; //Enable USCI\_A0 RX interrupt

\_BIS\_SR(GIE);

//Global variables initialization

**memset**(input\_data.data\_input, '\0', **sizeof**(input\_data.data\_input));

input\_data.data\_count = 0;

//Infinite loop

**for**(;;){

}

}

**#pragma** vector=TIMER0\_A0\_VECTOR

**\_\_interrupt** **void** **Timer0\_A0** (**void**)

{

// Clean interruption flag

TA0CCTL0 &= ~CCIFG;

}

**#pragma** vector=USCIAB0RX\_VECTOR

**\_\_interrupt** **void** **USCI0RX\_ISR**(**void**)

{

**static** **char** buff;

**while** (!(IFG2&UCA0TXIFG)); // USCI\_A0 TX buffer ready?

buff = UCA0RXBUF;

processData(buff);

}

/\* Function name: processData(char)

\* Developer: Raul Castañon

\* Details: Function used to process incoming data

\*/

**void** **processData**(**char** buff\_rx)

{

// Local variable declarations

**static** **int** pwm\_val;

// Verify if we have to process data or save data

// To process data we have to receive '\n'

**if**(buff\_rx != '\n')

{

// Validate incoming data

**if**((MAXIMUM\_ARRAY\_DATA > input\_data.data\_count) &&

(ASCII\_MIN\_NUMBER < buff\_rx) && (ASCII\_MAX\_NUMBER > buff\_rx))

{

// Add data to array

input\_data.data\_input[input\_data.data\_count] = buff\_rx;

// Increment count from data received

input\_data.data\_count++;

}

**else**

{

// Clean variables

input\_data.data\_count = 0;

**memset**(input\_data.data\_input, '\0', **sizeof**(input\_data.data\_input));

}

}

**else**

{

// Validate count and data

**if**((MAXIMUM\_ARRAY\_DATA >= input\_data.data\_count) && ('\0' != input\_data.data\_input[0]))

{

// Convert received data (duty cycle) in timer counts

pwm\_val = (**int**)((**atoi**(input\_data.data\_input))\*128)/100;

// Check limits

**if**(pwm\_val < MINIMUM\_RATE)

{

pwm\_val = MINIMUM\_RATE;

}

**else** **if**(pwm\_val > MAXIMUM\_RATE)

{

pwm\_val = MAXIMUM\_RATE;

}

**else**

{

// Do nothing

}

// Set pwm duty cycle

TA0CCR1 = pwm\_val;

// Clean variables

input\_data.data\_count = 0;

**memset**(input\_data.data\_input, '\0', **sizeof**(input\_data.data\_input));

}

}

}

# Apéndice II. Código implementado en la GUI

## Archivo de configuración

#-------------------------------------------------

#

# Project created by QtCreator 2017-12-14T10:24:36

#

#-------------------------------------------------

QT += core gui

QT += serialport

QT += core

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = LightControl\_Develop\_01

TEMPLATE = app

SOURCES += main.cpp\

mainwindow.cpp \

methods.cpp

HEADERS += mainwindow.h \

methods.h

FORMS += mainwindow.ui

INCLUDEPATH += "C:\OpenCV\include"

LIBS += -L"C:\OpenCV\lib"\

-llibopencv\_core300\

-llibopencv\_imgproc300\

-llibopencv\_imgcodecs300\

-llibopencv\_highgui300\

-llibopencv\_videoio300

## Librerías

### methods.h

#ifndef METHODS

#define METHODS

// Included libraries

#include <QDebug> // TODO verify if we need this at the end of the project

#include <stdlib.h>

#include <math.h>

// Defines declarations

#define VERY\_NEGATIVE 0.0

#define NEGATIVE 65.0

#define ZERO 125.0

#define POSITIVE 190.0

#define VERY\_POSITIVE 255.0

// Function declarations

void setMiddlePoints(unsigned int, unsigned int);

unsigned int FuzzyCtl(unsigned int, unsigned int);

void memDegreeP(int);

void memDegreeI(int);

unsigned int fuzzyRules(void);

// Exported variables

extern unsigned int middlePoints[5][2];

#endif // METHODS

### mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

//Included libraries

#include <QMainWindow>

#include <opencv2/opencv.hpp>

#include <QFileDialog>

#include <QDebug>

#include <QtSerialPort/QSerialPort>

#include <QTimer>

#include <qstring.h>

#include "methods.h"

// Definitions

#define DEFAULT\_TEST\_POINT "180"

#define DEFAULT\_COM\_PORT "COM10"

#define DEFAULT\_PWM\_VALUE "50\n"

#define DEFAULT\_SIZE\_INTENSITY 50

#define DEFAULT\_TIMER\_ON 2000

#define DEFAULT\_VIDEO\_NAME "Video window"

#define DEFAULT\_WAITKEY\_TIME 100

#define DEFAULT\_VIDEO\_CAM\_PORT 0

#define DEFAULT\_SETPOINT 120

#define DEFAULT\_SET\_POINT "120"

#define TIMER\_ENABLED "Timer one is enabled"

#define TIMER\_DISABLED "Timer one is disabled"

#define VIDEO\_ENABLED "Video camera open"

#define VIDEO\_DISABLED "Video camera closed"

#define CONNECTION\_SUCESSFUL "Serial communication sucessful"

#define SERIAL\_CLOSED "Serial port closed"

#define NAMED\_WINDOW "Selected Image"

#define SERIAL\_COMM\_ERROR "Error 0: could not open serial port"

#define ERROR\_PORT\_OPEN "Error 1: Port already open"

#define ERROR\_TIMER\_OPEN "Error 2: Timer one is enabled"

#define ERROR\_TIMER\_TOCLOSE "Error 3: Timer one is not active"

#define ERROR\_SET\_POINT "Error 4: Default setPoint assigned"

#define ERROR\_SERIAL\_CLOSE "Error 5: SerialPort is not open"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void on\_OpenImageButton\_clicked();

void on\_closeApp\_clicked();

void on\_grayScale\_clicked();

void on\_middlePoints\_clicked();

void on\_getIntensity\_clicked();

void on\_openSerialPort\_clicked();

void on\_closeSerialPort\_clicked();

void on\_setTimer\_clicked();

void on\_stopTimer\_clicked();

void onTimeOut();

void on\_openVideoCam\_clicked();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

## Archivos fuente

### methods.cpp

#include "methods.h"

// Global variables declarations

unsigned int middlePoints[5][2];

// Controller variables

int p\_err = 0;

int i\_err = 0;

double mu\_e\_vn = 0.0;

double mu\_e\_n = 0.0;

double mu\_e\_z = 0.0;

double mu\_e\_p = 0.0;

double mu\_e\_vp = 0.0;

double mu\_ei\_vn = 0.0;

double mu\_ei\_n = 0.0;

double mu\_ei\_z = 0.0;

double mu\_ei\_p = 0.0;

double mu\_ei\_vp = 0.0;

double mu\_u\_voff = 0.0;

double mu\_u\_off = 0.0;

double mu\_u\_on = 0.0;

double mu\_u\_von = 0.0;

/\* Function name: FuzzyCtl(uint, uint)

\* Developer: Raul Castañon

\* Details: Get control signal

\*/

unsigned int FuzzyCtl(unsigned int intensityLevel, unsigned int setPoint)

{

// Local variable declarations

unsigned int ut = 0; // This will be the return control variable

// Get the proportional error

p\_err = setPoint - intensityLevel;

// Verify limits of the proportional error

if(p\_err >= 255)

{

p\_err = 250;

}

else if (p\_err <= 0)

{

p\_err = 5;

}

else

{

// do nothing

}

// Get the integral error

i\_err += p\_err;

// Verify limits of the integral error

if(i\_err >= 255)

{

i\_err = 250;

}

else if (i\_err <= 0)

{

i\_err = 5;

}

else

{

// do nothing

}

// Set the membership degree variables for p\_err

memDegreeP(p\_err);

// Set the membership degree variables for i\_err

memDegreeI(i\_err);

// Apply the fuzzy rules

ut = fuzzyRules();

// Get the output from the controler

return ut;

}

/\* Function name: fuzzyRules(void)

\* Developer: Raul Castañon

\* Details: Apply fuzzy rules

\*/

unsigned int fuzzyRules(void)

{

// Local variable declarations

double u\_voff[6], u\_off[6], u\_on[6], u\_von[6];

double ut\_voff = 0.0;

double ut\_off = 0.0;

double ut\_on = 0.0;

double ut\_von = 0.0;

double ut = 0.0;

// Local variable initialization

memset(u\_voff, 0x00, sizeof(u\_voff));

memset(u\_off, 0x00, sizeof(u\_off));

memset(u\_on, 0x00, sizeof(u\_on));

memset(u\_von, 0x00, sizeof(u\_von));

// Rule one

u\_voff[0] = std::min(mu\_e\_vn, mu\_ei\_vn);

// Rule two

u\_voff[1] = std::min(mu\_e\_vn, mu\_ei\_n);

// Rule three

u\_voff[2] = std::min(mu\_e\_vn, mu\_ei\_z);

// Rule four

u\_voff[3] = std::min(mu\_e\_vn, mu\_ei\_p);

// Rule five

u\_voff[4] = std::min(mu\_e\_vn, mu\_ei\_vp);

// Rule six

u\_voff[5] = std::min(mu\_e\_n, mu\_ei\_vn);

// Rule seven

u\_off[0] = std::min(mu\_e\_n, mu\_ei\_n);

// Rule eight

u\_off[1] = std::min(mu\_e\_n, mu\_ei\_z);

// Rule nine

u\_off[2] = std::min(mu\_e\_n, mu\_ei\_p);

// Rule ten

u\_off[3] = std::min(mu\_e\_n, mu\_ei\_vp);

// Rule eleven

u\_off[4] = std::min(mu\_e\_z, mu\_ei\_vn);

// Rule twelve

u\_off[5] = std::min(mu\_e\_z, mu\_ei\_n);

// Rule thirteen

u\_on[0] = std::min(mu\_e\_z, mu\_ei\_z);

// Rule fourteen

u\_on[1] = std::min(mu\_e\_z, mu\_ei\_p);

// Rule fiveteen

u\_on[2] = std::min(mu\_e\_z, mu\_ei\_vp);

// Rule sixteen

u\_on[3] = std::min(mu\_e\_p, mu\_ei\_vn);

// Rule seventeen

u\_on[4] = std::min(mu\_e\_p, mu\_ei\_n);

// Rule eighteen

u\_on[5] = std::min(mu\_e\_p, mu\_ei\_z);

// Rule nineteen

u\_von[0] = std::min(mu\_e\_p, mu\_ei\_p);

// Rule twenty

u\_von[1] = std::min(mu\_e\_p, mu\_ei\_vp);

// Rule twentyone

u\_von[2] = std::min(mu\_e\_vp, mu\_ei\_vn);

// Rule twentytwo

u\_von[3] = std::min(mu\_e\_vp, mu\_ei\_n);

// Rule twentythree

u\_von[4] = std::min(mu\_e\_vp, mu\_ei\_z);

// Rule twentyfour

u\_von[5] = std::min(mu\_e\_vp, mu\_ei\_p);

// Get the module of each output membership

ut\_voff = sqrt(pow(u\_voff[0], 2.0) + pow(u\_voff[1], 2.0)+ pow(u\_voff[2], 2.0)+ pow(u\_voff[3], 2.0)+ pow(u\_voff[4], 2.0)+ pow(u\_voff[5], 2.0));

ut\_off = sqrt(pow(u\_off[0], 2.0) + pow(u\_off[1], 2.0)+ pow(u\_off[2], 2.0)+ pow(u\_off[3], 2.0)+ pow(u\_off[4], 2.0)+ pow(u\_off[5], 2.0));

ut\_on = sqrt(pow(u\_on[0], 2.0) + pow(u\_on[1], 2.0)+ pow(u\_on[2], 2.0)+ pow(u\_on[3], 2.0)+ pow(u\_on[4], 2.0)+ pow(u\_on[5], 2.0));

ut\_von = sqrt(pow(u\_von[0], 2.0) + pow(u\_von[1], 2.0)+ pow(u\_von[2], 2.0)+ pow(u\_von[3], 2.0)+ pow(u\_von[4], 2.0)+ pow(u\_von[5], 2.0));

ut = (unsigned int) ((ut\_voff \* NEGATIVE) +(ut\_off \* ZERO) + (ut\_on \* POSITIVE) + (ut\_von \* VERY\_POSITIVE));

return ut;

}

/\* Function name: memDegreeI(int)

\* Developer: Raul Castañon

\* Details: Get membership degree

\*/

void memDegreeI(int i\_err)

{

// Function from the VERY\_NEGATIVE area

if((VERY\_NEGATIVE <= i\_err) && (NEGATIVE > i\_err))

{

mu\_ei\_vn = (NEGATIVE - i\_err)/(NEGATIVE - VERY\_NEGATIVE);

}

else

{

mu\_ei\_vn = 0.0;

}

// Funtion of the NEGATIVE area

if((VERY\_NEGATIVE <= i\_err) && (NEGATIVE > i\_err))

{

mu\_ei\_n = (i\_err - VERY\_NEGATIVE)/(NEGATIVE - VERY\_NEGATIVE);

}

else if((NEGATIVE <= i\_err) && (ZERO > i\_err))

{

mu\_ei\_n = (ZERO - i\_err)/(ZERO - NEGATIVE);

}

else

{

mu\_ei\_n = 0.0;

}

// Function of the ZERO area

if((NEGATIVE <= i\_err) && (ZERO> i\_err))

{

mu\_ei\_z = (i\_err - NEGATIVE)/(ZERO - NEGATIVE);

}

else if((ZERO <= i\_err) && (POSITIVE > i\_err))

{

mu\_ei\_z = (POSITIVE - i\_err)/(POSITIVE - ZERO);

}

else

{

mu\_ei\_z = 0.0;

}

// Function of the POSITIVE area

if((ZERO <= i\_err) && (POSITIVE> i\_err))

{

mu\_ei\_p = (i\_err - ZERO)/(POSITIVE - ZERO);

}

else if((POSITIVE <= i\_err) && (VERY\_POSITIVE > i\_err))

{

mu\_ei\_p = (VERY\_POSITIVE - i\_err)/(VERY\_POSITIVE - POSITIVE);

}

else

{

mu\_ei\_p = 0.0;

}

// Function of the VERY\_POSITIVE area

if((POSITIVE <= i\_err) && (VERY\_POSITIVE> i\_err))

{

mu\_ei\_p = (i\_err - POSITIVE)/(VERY\_POSITIVE - POSITIVE);

}

else

{

mu\_ei\_p = 0.0;

}

}

/\* Function name: memDegreeP(int)

\* Developer: Raul Castañon

\* Details: Get membership degree

\*/

void memDegreeP(int p\_err)

{

// Function from the VERY\_NEGATIVE area

if((VERY\_NEGATIVE <= p\_err) && (NEGATIVE > p\_err))

{

mu\_e\_vn = (NEGATIVE - p\_err)/(NEGATIVE - VERY\_NEGATIVE);

}

else

{

mu\_e\_vn = 0.0;

}

// Funtion of the NEGATIVE area

if((VERY\_NEGATIVE <= p\_err) && (NEGATIVE > p\_err))

{

mu\_e\_n = (p\_err - VERY\_NEGATIVE)/(NEGATIVE - VERY\_NEGATIVE);

}

else if((NEGATIVE <= p\_err) && (ZERO > p\_err))

{

mu\_e\_n = (ZERO - p\_err)/(ZERO - NEGATIVE);

}

else

{

mu\_e\_n = 0.0;

}

// Function of the ZERO area

if((NEGATIVE <= p\_err) && (ZERO> p\_err))

{

mu\_e\_z = (p\_err - NEGATIVE)/(ZERO - NEGATIVE);

}

else if((ZERO <= p\_err) && (POSITIVE > p\_err))

{

mu\_e\_z = (POSITIVE - p\_err)/(POSITIVE - ZERO);

}

else

{

mu\_e\_z = 0.0;

}

// Function of the POSITIVE area

if((ZERO <= p\_err) && (POSITIVE> p\_err))

{

mu\_e\_p = (p\_err - ZERO)/(POSITIVE - ZERO);

}

else if((POSITIVE <= p\_err) && (VERY\_POSITIVE > p\_err))

{

mu\_e\_p = (VERY\_POSITIVE - p\_err)/(VERY\_POSITIVE - POSITIVE);

}

else

{

mu\_e\_p = 0.0;

}

// Function of the VERY\_POSITIVE area

if((POSITIVE <= p\_err) && (VERY\_POSITIVE> p\_err))

{

mu\_e\_p = (p\_err - POSITIVE)/(VERY\_POSITIVE - POSITIVE);

}

else

{

mu\_e\_p = 0.0;

}

}

/\* Function name: setMiddlePoints(uint, uint)

\* Developer: Raul Castañon

\* Details: Get middle points from the data of the image

\*/

void setMiddlePoints(unsigned int cols, unsigned int rows)

{

// Initialize global variable

memset(middlePoints, 0, sizeof(middlePoints));

// Get middle points

middlePoints[0][0] = (unsigned int)(cols/2.0);

middlePoints[0][1] = (unsigned int)(rows/2.0);

middlePoints[1][0] = (unsigned int)(cols/4.0);

middlePoints[1][1] = (unsigned int)(rows/4.0);

middlePoints[2][0] = (unsigned int)(3\*(cols/4.0));

middlePoints[2][1] = (unsigned int)(rows/4.0);

middlePoints[3][0] = (unsigned int)(cols/4.0);

middlePoints[3][1] = (unsigned int)(3\*(rows/4.0));

middlePoints[4][0] = (unsigned int)(3\*(cols/4.0));

middlePoints[4][1] = (unsigned int)(3\*(rows/4.0));

}

### mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

using namespace cv;

// Global variables

QSerialPort serial;

QTimer \*timerOne;

Mat inputImage;

Mat inputVideo;

unsigned int intensityLevel = 0;

bool serialFlag = false;

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

// Configure timer one

timerOne = new QTimer(this);

connect(timerOne, SIGNAL(timeout()), this, SLOT(onTimeOut()));

// Disable not required buttons

ui->grayScale->setDisabled(true);

ui->middlePoints->setDisabled(true);

ui->getIntensity->setDisabled(true);

ui->closeSerialPort->setEnabled(false);

ui->openSerialPort->setEnabled(true);

ui->setTimer->setEnabled(true);

ui->stopTimer->setEnabled(false);

}

MainWindow::~*MainWindow*()

{

delete ui;

}

/\* Function name: OpenImageButton()

\* Developer: Raul Castañon

\* Details: Function used to open and show an image

\*/

void MainWindow::on\_OpenImageButton\_clicked()

{

QString inputImg = QFileDialog::getOpenFileName();

inputImage = imread(inputImg.toLatin1().data());

// Verify if there is an image

if(!(inputImage.empty()))

{

// Show image

namedWindow(NAMED\_WINDOW, WINDOW\_NORMAL);

imshow(NAMED\_WINDOW, inputImage);

// Enable grayscale button

ui->grayScale->setEnabled(true);

// Disable OpenImage button

ui->OpenImageButton->setEnabled(false);

}

}

/\* Function name: closeApp()

\* Developer: Raul Castañon

\* Details: Close application

\*/

void MainWindow::on\_closeApp\_clicked()

{

// Check if we need to stop timer

if(true == timerOne->isActive())

{

timerOne->stop();

}

// Check if we need to close SerialPort

if(true == serialFlag)

{

MainWindow::on\_closeSerialPort\_clicked();

}

close();

}

/\* Function name: on\_grayScale\_clicked()

\* Developer: Raul Castañon

\* Details: Convert image in grayscale format

\*/

void MainWindow::on\_grayScale\_clicked()

{

if(false == inputImage.empty())

{

// Set image in grayscale format

cvtColor(inputImage, inputImage, CV\_RGB2GRAY);

// Show result in same window

imshow(NAMED\_WINDOW, inputImage);

ui->middlePoints->setEnabled(true);

ui->grayScale->setEnabled(false);

}

}

/\* Function name: on\_middlePoints\_clicked()

\* Developer: Raul Castañon

\* Details: Get the middle points to process the calculation of intensity

\*/

void MainWindow::on\_middlePoints\_clicked()

{

// Get middle points of the img

setMiddlePoints(inputImage.cols, inputImage.rows);

// Put values on the GUI labels

ui->P\_one\_cols->setText(QString::number(middlePoints[0][0]));

ui->P\_one\_rows->setText(QString::number(middlePoints[0][1]));

ui->P\_two\_cols->setText(QString::number(middlePoints[1][0]));

ui->P\_two\_rows->setText(QString::number(middlePoints[1][1]));

ui->P\_three\_cols->setText(QString::number(middlePoints[2][0]));

ui->P\_three\_rows->setText(QString::number(middlePoints[2][1]));

ui->P\_four\_cols->setText(QString::number(middlePoints[3][0]));

ui->P\_four\_rows->setText(QString::number(middlePoints[3][1]));

ui->P\_five\_cols->setText(QString::number(middlePoints[4][0]));

ui->P\_five\_rows->setText(QString::number(middlePoints[4][1]));

// Enable visualization of getIntensity button

ui->getIntensity->setEnabled(true);

// Disable current button

ui->middlePoints->setEnabled(false);

}

/\* Function name: on\_getIntensity\_clicked()

\* Developer: Raul Castañon

\* Details: Get intensity levels from the image

\*/

void MainWindow::on\_getIntensity\_clicked()

{

// Local variables

Scalar promAreaOne, promAreaTwo, promAreaThree, promAreaFour, promAreaFive;

Mat areaOne, areaTwo, areaThree, areaFour, areaFive;

// Verify size of the area

if((ui->widthArea->text().isEmpty())||(ui->hightArea->text().isEmpty()))

{

qDebug("Default size asigned");

ui->widthArea->setText(QString::number(DEFAULT\_SIZE\_INTENSITY));

ui->hightArea->setText(QString::number(DEFAULT\_SIZE\_INTENSITY));

}

// Get areas

areaOne = inputImage(Rect(middlePoints[0][0], middlePoints[0][1], ui->widthArea->text().toInt(), ui->hightArea->text().toInt()));

areaTwo = inputImage(Rect(middlePoints[1][0], middlePoints[1][1], ui->hightArea->text().toInt(), ui->hightArea->text().toInt()));

areaThree = inputImage(Rect(middlePoints[2][0], middlePoints[2][1], ui->widthArea->text().toInt(), ui->hightArea->text().toInt()));

areaFour = inputImage(Rect(middlePoints[3][0], middlePoints[3][1], ui->widthArea->text().toInt(), ui->hightArea->text().toInt()));

areaFive = inputImage(Rect(middlePoints[4][0], middlePoints[4][1], ui->widthArea->text().toInt(), ui->hightArea->text().toInt()));

// Get intensity level per area

promAreaOne = mean(areaOne);

promAreaTwo = mean(areaTwo);

promAreaThree = mean(areaThree);

promAreaFour = mean(areaFour);

promAreaFive = mean(areaFive);

// Show results

ui->intensityLevel\_1->setText(QString::number(promAreaOne[0]));

ui->intensityLevel\_2->setText(QString::number(promAreaTwo[0]));

ui->intensityLevel\_3->setText(QString::number(promAreaThree[0]));

ui->intensityLevel\_4->setText(QString::number(promAreaFour[0]));

ui->intensityLevel\_5->setText(QString::number(promAreaFive[0]));

// Get intensity level mean

intensityLevel = (unsigned int)((promAreaOne[0] + promAreaTwo[0] + promAreaThree[0] + promAreaFour[0] + promAreaFive[0]) / 5.0);

ui->intensityLevel->setText(QString::number(intensityLevel));

// Enable open Image button and disable getIntensity

ui->OpenImageButton->setEnabled(true);

ui->getIntensity->setEnabled(false);

}

/\* Function name: openSerialPort()

\* Developer: Raul Castañon

\* Details: Configure and open the serialPort

\*/

void MainWindow::on\_openSerialPort\_clicked()

{

if(false == serialFlag)

{

// Set configuration conditions

serial.setPortName(DEFAULT\_COM\_PORT);

serial.setBaudRate(QSerialPort::Baud9600);

serial.setDataBits(QSerialPort::Data8);

serial.setParity(QSerialPort::NoParity);

serial.setStopBits(QSerialPort::OneStop);

serial.setFlowControl(QSerialPort::NoFlowControl);

// Check if the port has been open

if(serial.*open*(QIODevice::ReadWrite))

{

qDebug(CONNECTION\_SUCESSFUL);

serial.write(DEFAULT\_PWM\_VALUE);

serialFlag = true;

ui->openSerialPort->setEnabled(false);

ui->closeSerialPort->setEnabled(true);

}

//if not send error message

else

{

qDebug(SERIAL\_COMM\_ERROR);

ui->closeSerialPort->setEnabled(false);

ui->openSerialPort->setEnabled(true);

}

}

else

{

qDebug(ERROR\_PORT\_OPEN);

}

}

/\* Function name: closeSerialPort()

\* Developer: Raul Castañon

\* Details: Close serial port

\*/

void MainWindow::on\_closeSerialPort\_clicked()

{

// Verify serial port condition

if(serial.isOpen())

{

serial.*close*();

serialFlag = false;

qDebug(SERIAL\_CLOSED);

ui->closeSerialPort->setEnabled(false);

ui->openSerialPort->setEnabled(true);

}

else

{

// Do nothing

qDebug(ERROR\_PORT\_OPEN);

}

}

/\* Function name: setTimer()

\* Developer: Raul Castañon

\* Details: Start timer one

\*/

void MainWindow::on\_setTimer\_clicked()

{

// Verify if timer one is not active

if(!(timerOne->isActive()))

{

qDebug(TIMER\_ENABLED);

timerOne->start(DEFAULT\_TIMER\_ON);

ui->setTimer->setEnabled(false);

ui->stopTimer->setEnabled(true);

ui->openSerialPort->click();

}

else

{

qDebug(ERROR\_TIMER\_OPEN);

}

}

/\* Function name: stopTimer()

\* Developer: Raul Castañon

\* Details: Stop timer one

\*/

void MainWindow::on\_stopTimer\_clicked()

{

// Verify timer one is active

if(timerOne->isActive())

{

qDebug(TIMER\_DISABLED);

timerOne->stop();

ui->setTimer->setEnabled(true);

ui->stopTimer->setEnabled(false);

ui->closeSerialPort->click();

destroyWindow(NAMED\_WINDOW);

}

else

{

qDebug(ERROR\_TIMER\_TOCLOSE);

}

}

/\* Function name: on\_openVideoCam\_clicked()

\* Developer: Raul Castañon

\* Details: Open video camera and show continuosly

\*/

void MainWindow::on\_openVideoCam\_clicked()

{

VideoCapture videoCam;

videoCam.*open*(DEFAULT\_VIDEO\_CAM\_PORT);

if(videoCam.*isOpened*())

{

qDebug(VIDEO\_ENABLED);

do

{

videoCam >> inputVideo;

imshow(DEFAULT\_VIDEO\_NAME, inputVideo);

waitKey(DEFAULT\_WAITKEY\_TIME);

ui->openVideoCam->setEnabled(false);

if(waitKey(30) > 0)

{

videoCam.*release*();

destroyWindow(DEFAULT\_VIDEO\_NAME);

qDebug(VIDEO\_DISABLED);

ui->openVideoCam->setEnabled(true);

break;

}

}while(true);

}

}

/\* Function name: onTimeOut()

\* Developer: Raul Castañon

\* Details: Linker to the timeOut signal from timerOne

\*/

void MainWindow::onTimeOut()

{

// Local variables

VideoCapture videoCam;

unsigned int setPoint = 0;

unsigned int ut = 0;

char u\_t[3];

// Open videoCamera

videoCam.*open*(0);

if(videoCam.*isOpened*())

{

// Get image

videoCam >> inputImage;

// Process Image

ui->grayScale->clicked();

ui->middlePoints->clicked();

ui->getIntensity->clicked();

// Verify setPoint

if((ui->setPoint->text().toInt() > 0) && (ui->setPoint->text().toInt() <= 255))

{

// Get set point

setPoint = (unsigned int) ui->setPoint->text().toInt();

}

else

{

qDebug(ERROR\_SET\_POINT);

setPoint = DEFAULT\_SETPOINT;

ui->setPoint->setText(DEFAULT\_SET\_POINT);

}

// Send params to the controller

ut = FuzzyCtl(intensityLevel, setPoint);

/\* ut is the control signal, it has been calculated in the limits of a byte

\* need to normalize to the mcu needs

\* maximum value that mcu will receive will be 100 (this will be the 100% of the PWM cycle

\* minimum value will be 5.

\* 255 -> 100

\* ut -> x

\*/

ut = 100 - (unsigned int)((ut \* 100) / 255);

itoa(ut, u\_t, 10);

u\_t[2] = '\n';

// Verify if serial port is open before writting

if(serial.isOpen())

{

serial.write(u\_t);

qDebug(u\_t);

}

else

{

qDebug(ERROR\_SERIAL\_CLOSE);

ui->stopTimer->clicked();

}

}

}

main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

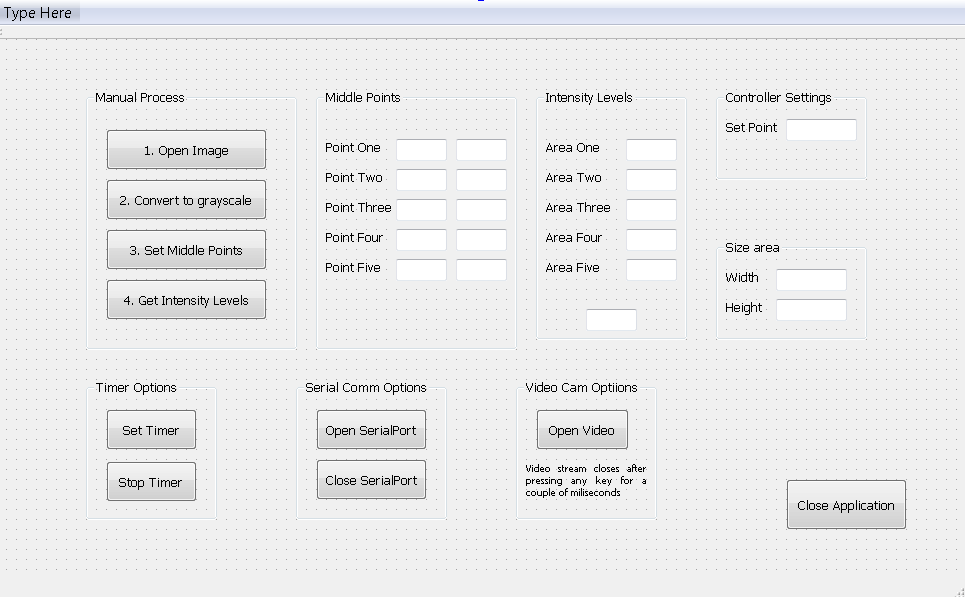
w.show();

return a.exec();

}

## Diseño de la interfaz

### GUI



### XML

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>971</width>

<height>602</height>

</rect>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="centralWidget">

<widget class="QGroupBox" name="groupBox">

<property name="geometry">

<rect>

<x>90</x>

<y>340</y>

<width>131</width>

<height>141</height>

</rect>

</property>

<property name="title">

<string>Timer Options</string>

</property>

<widget class="QPushButton" name="stopTimer">

<property name="geometry">

<rect>

<x>20</x>

<y>82</y>

<width>91</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Stop Timer</string>

</property>

</widget>

<widget class="QPushButton" name="setTimer">

<property name="geometry">

<rect>

<x>20</x>

<y>30</y>

<width>91</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Set Timer</string>

</property>

</widget>

</widget>

<widget class="QGroupBox" name="groupBox\_4">

<property name="geometry">

<rect>

<x>90</x>

<y>50</y>

<width>211</width>

<height>261</height>

</rect>

</property>

<property name="title">

<string>Manual Process</string>

</property>

<widget class="QPushButton" name="OpenImageButton">

<property name="geometry">

<rect>

<x>20</x>

<y>40</y>

<width>161</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>1. Open Image</string>

</property>

</widget>

<widget class="QPushButton" name="grayScale">

<property name="geometry">

<rect>

<x>20</x>

<y>90</y>

<width>161</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>2. Convert to grayscale</string>

</property>

</widget>

<widget class="QPushButton" name="middlePoints">

<property name="geometry">

<rect>

<x>20</x>

<y>140</y>

<width>161</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>3. Set Middle Points</string>

</property>

</widget>

<widget class="QPushButton" name="getIntensity">

<property name="geometry">

<rect>

<x>20</x>

<y>190</y>

<width>161</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>4. Get Intensity Levels</string>

</property>

</widget>

</widget>

<widget class="QGroupBox" name="groupBox\_7">

<property name="geometry">

<rect>

<x>720</x>

<y>200</y>

<width>151</width>

<height>101</height>

</rect>

</property>

<property name="title">

<string>Size area</string>

</property>

<widget class="QLineEdit" name="widthArea">

<property name="geometry">

<rect>

<x>60</x>

<y>30</y>

<width>71</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="hightArea">

<property name="geometry">

<rect>

<x>60</x>

<y>60</y>

<width>71</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLabel" name="label\_12">

<property name="geometry">

<rect>

<x>10</x>

<y>30</y>

<width>55</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Width</string>

</property>

</widget>

<widget class="QLabel" name="label\_13">

<property name="geometry">

<rect>

<x>10</x>

<y>60</y>

<width>55</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Height</string>

</property>

</widget>

</widget>

<widget class="QGroupBox" name="groupBox\_5">

<property name="geometry">

<rect>

<x>320</x>

<y>50</y>

<width>201</width>

<height>261</height>

</rect>

</property>

<property name="title">

<string>Middle Points</string>

</property>

<widget class="QLabel" name="label\_2">

<property name="geometry">

<rect>

<x>10</x>

<y>50</y>

<width>61</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Point One</string>

</property>

</widget>

<widget class="QLabel" name="label\_3">

<property name="geometry">

<rect>

<x>10</x>

<y>80</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Point Two</string>

</property>

</widget>

<widget class="QLabel" name="label\_4">

<property name="geometry">

<rect>

<x>10</x>

<y>110</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Point Three</string>

</property>

</widget>

<widget class="QLabel" name="label\_5">

<property name="geometry">

<rect>

<x>10</x>

<y>140</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Point Four</string>

</property>

</widget>

<widget class="QLabel" name="label\_6">

<property name="geometry">

<rect>

<x>10</x>

<y>170</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Point Five</string>

</property>

</widget>

<widget class="QLineEdit" name="P\_one\_cols">

<property name="geometry">

<rect>

<x>80</x>

<y>50</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_two\_cols">

<property name="geometry">

<rect>

<x>80</x>

<y>80</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_three\_cols">

<property name="geometry">

<rect>

<x>80</x>

<y>110</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_four\_cols">

<property name="geometry">

<rect>

<x>80</x>

<y>140</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_five\_cols">

<property name="geometry">

<rect>

<x>80</x>

<y>170</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_four\_rows">

<property name="geometry">

<rect>

<x>140</x>

<y>140</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_two\_rows">

<property name="geometry">

<rect>

<x>140</x>

<y>80</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_five\_rows">

<property name="geometry">

<rect>

<x>140</x>

<y>170</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_one\_rows">

<property name="geometry">

<rect>

<x>140</x>

<y>50</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="P\_three\_rows">

<property name="geometry">

<rect>

<x>140</x>

<y>110</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

</widget>

<widget class="QGroupBox" name="groupBox\_3">

<property name="geometry">

<rect>

<x>520</x>

<y>340</y>

<width>141</width>

<height>141</height>

</rect>

</property>

<property name="title">

<string>Video Cam Optiions</string>

</property>

<widget class="QPushButton" name="openVideoCam">

<property name="geometry">

<rect>

<x>20</x>

<y>30</y>

<width>93</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Open Video</string>

</property>

</widget>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>10</x>

<y>70</y>

<width>121</width>

<height>61</height>

</rect>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Preferred" vsizetype="Preferred">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="font">

<font>

<pointsize>6</pointsize>

<kerning>true</kerning>

</font>

</property>

<property name="autoFillBackground">

<bool>false</bool>

</property>

<property name="text">

<string>Video stream closes after pressing any key for a couple of miliseconds</string>

</property>

<property name="textFormat">

<enum>Qt::AutoText</enum>

</property>

<property name="alignment">

<set>Qt::AlignJustify|Qt::AlignVCenter</set>

</property>

<property name="wordWrap">

<bool>true</bool>

</property>

</widget>

</widget>

<widget class="QGroupBox" name="groupBox\_8">

<property name="geometry">

<rect>

<x>720</x>

<y>50</y>

<width>151</width>

<height>91</height>

</rect>

</property>

<property name="title">

<string>Controller Settings</string>

</property>

<widget class="QLabel" name="label\_14">

<property name="geometry">

<rect>

<x>10</x>

<y>30</y>

<width>61</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Set Point</string>

</property>

</widget>

<widget class="QLineEdit" name="setPoint">

<property name="geometry">

<rect>

<x>70</x>

<y>30</y>

<width>71</width>

<height>22</height>

</rect>

</property>

</widget>

</widget>

<widget class="QPushButton" name="closeApp">

<property name="geometry">

<rect>

<x>790</x>

<y>440</y>

<width>121</width>

<height>51</height>

</rect>

</property>

<property name="text">

<string>Close Application</string>

</property>

</widget>

<widget class="QGroupBox" name="groupBox\_6">

<property name="geometry">

<rect>

<x>540</x>

<y>50</y>

<width>151</width>

<height>251</height>

</rect>

</property>

<property name="title">

<string>Intensity Levels</string>

</property>

<widget class="QLabel" name="label\_11">

<property name="geometry">

<rect>

<x>10</x>

<y>170</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Area Five</string>

</property>

</widget>

<widget class="QLabel" name="label\_7">

<property name="geometry">

<rect>

<x>10</x>

<y>80</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Area Two</string>

</property>

</widget>

<widget class="QLabel" name="label\_10">

<property name="geometry">

<rect>

<x>10</x>

<y>110</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Area Three</string>

</property>

</widget>

<widget class="QLabel" name="label\_9">

<property name="geometry">

<rect>

<x>10</x>

<y>50</y>

<width>61</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Area One</string>

</property>

</widget>

<widget class="QLabel" name="label\_8">

<property name="geometry">

<rect>

<x>10</x>

<y>140</y>

<width>71</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Area Four</string>

</property>

</widget>

<widget class="QLineEdit" name="intensityLevel\_1">

<property name="geometry">

<rect>

<x>90</x>

<y>50</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="intensityLevel\_2">

<property name="geometry">

<rect>

<x>90</x>

<y>80</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="intensityLevel\_3">

<property name="geometry">

<rect>

<x>90</x>

<y>110</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="intensityLevel\_4">

<property name="geometry">

<rect>

<x>90</x>

<y>140</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="intensityLevel\_5">

<property name="geometry">

<rect>

<x>90</x>

<y>170</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QLineEdit" name="intensityLevel">

<property name="geometry">

<rect>

<x>50</x>

<y>220</y>

<width>51</width>

<height>22</height>

</rect>

</property>

</widget>

</widget>

<widget class="QGroupBox" name="groupBox\_2">

<property name="geometry">

<rect>

<x>300</x>

<y>340</y>

<width>151</width>

<height>141</height>

</rect>

</property>

<property name="title">

<string>Serial Comm Options</string>

</property>

<widget class="QPushButton" name="closeSerialPort">

<property name="geometry">

<rect>

<x>20</x>

<y>80</y>

<width>111</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Close SerialPort</string>

</property>

</widget>

<widget class="QPushButton" name="openSerialPort">

<property name="geometry">

<rect>

<x>20</x>

<y>30</y>

<width>111</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Open SerialPort</string>

</property>

</widget>

</widget>

</widget>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>971</width>

<height>26</height>

</rect>

</property>

</widget>

<widget class="QToolBar" name="mainToolBar">

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

<widget class="QStatusBar" name="statusBar"/>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources/>

<connections/>

</ui>

# Referencias

bticino. (15 de Mayo de 2017). *bticino México*. Obtenido de Control Centralizado: http://www.bticino.com.mx/index.php?id=1912

Hernandez, V. M., Silva, R., & Carrillo, R. V. (2013). *Control Automático: Teoría de diseño, contrucción de prototipos, modelado, identificación y pruebas experimentales.* México DF: Colección CIDETEC del Instituto Politécnico Nacional.

Liu, W., & Ramirez, A. (2017). State of the art review of the enviromental assessment and risks of underground geo-energy resources exploitation. *ELSEVIER*, 628-644.

Lobao, J. A., Devezas, T., & Catalao, J. P. (2015). Energy efficiency of lighting installations: Software application and experimental validation. *ELSEVIER*, 110-115.

OpenCV. (15 de Mayo de 2017). *OpenCV*. Obtenido de Home Page: http://opencv.org/

OpenCV. (2 de January de 2018). *OpenCV releases*. Obtenido de OpenCV: https://opencv.org/releases.html

Schreder. (15 de Mayo de 2017). *Schreder Owlet Sistemas de Control.* Obtenido de Control Inteligente para una Iluminación Eficiente: http://www.schreder.com/globalassets/sitecollectiondocuments/additional-content/schreder-owlet-sistemas-de-control.pdf

SENER. (14 de Mayo de 2017). *Secretaría de Energía*. Obtenido de Sistema de Información Energética: http://sie.energia.gob.mx/bdiController.do?action=cuadro&cvecua=IE7C02

Sirlin, E. (15 de Mayo de 2017). *Física de la Luz.* Obtenido de elisirlin: http://www.elisirlin.com.ar/11\_fisica%20de%20la%20luz.pdf

Texas Instruments. (15 de Mayo de 2017). *Technical documents.* Obtenido de Datasheet: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf

Texas Instruments. (15 de Mayo de 2017). *Technical documents.* Obtenido de Users manual: http://www.ti.com/lit/ug/slau144j/slau144j.pdf

Texas Instruments Incorporated. (2013, July). *Texas Instruments*. Retrieved from MSP430x2xx User's Guide: http://www.ti.com/lit/ug/slau144j/slau144j.pdf

Young , I. T., Gerbrands, J. J., & van Vliet, L. J. (15 de Mayo de 2017). Fundamentals of Image Processing. Universidad Técnica de Delft, Delft, Países Bajos.

1. Los formatos y tamaños de imagen se encuentran precedidos por la documentación de OpenCV. [↑](#footnote-ref-1)