Control de elevadores de fruta mediante visión artificial.

MEMORIA PRESENTADA POR:
María Soler León

GRADO EN INGENIERÍA ELÉCTRICA

Convocatoria de defensa: Junio de 2016

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

AGRADECIMIENTOS:

En primer lugar, me gustaría agradecer a mis padres y a mi hermana, el apoyo que siempre me han brindado, y en especial, en este último periodo con el que pongo fin a mis estudios de grado.

En segundo lugar, me gustaría agradecer el asesoramiento y la ayuda recibida por parte del profesor de la Escuela Politécnica Superior de Alcoy y tutor de este trabajo fin de grado, Don Juan Ramón Rufino Valor.
También agradezco al profesor de la EPSA, Don Jaime Masiá, la ayuda prestada con la impresión de una de los componentes de este proyecto.

Por último, me gustaría agradecer a todas y cada una de esas personas, ya sean profesores o no, que de alguna forma han influido en mi desarrollo como graduada en ingeniería y que, gracias a sus consejos y enseñanzas, han hecho posible que hoy esté aquí, defendiendo mi trabajo fin de grado.

¡Gracias a todos!

1. INTRODUCCIÓN

Hoy en día, la tecnología forma parte de nuestra vida de un modo que hace unos años habría parecido inimaginable.
Cada vez es más frecuente que "aparezca" un nuevo y pequeño dispositivo cuya finalidad sea facilitarnos en algún aspecto nuestro día a día.
Pero no solo se queda ahí la intervención de la tecnología en nuestras vidas, actualmente, muchos sectores de la industria están innovando y creciendo gracias a las múltiples y diferentes posibilidades que los avances tecnológicos les brindan.

Uno de los sectores que ha crecido y mejorado gracias a los avances tecnológicos es el sector agroalimentario, gracias, entre otras, a las nuevas técnicas de visión artificial que este sector está empleando. Cada vez más, los controles de calidad y producción que se aplican a los productos, son más precisos y exhaustivos.

Empresas como Unitec, Aweta, Key Technologies o Tomra desarrollan máquinas, para este sector, cuyo fin es seleccionar los productos en buen estado y eliminar los defectuosos, como resultado, observamos que empieza a resultar más y más complicado encontrar, por ejemplo, cerezas en mal estado en una caja que haya sido sometida a estas nuevas técnicas de control de calidad.

2. PRESENTACIÓN DE LA SOLUCIÓN

2.1. NECESIDAD QUE CUBRE

Una de las máquinas, que las empresas dedicadas al desarrollo de líneas de confección de fruta han desarrollado, es el elevador de fruta. Esta máquina tiene como función elevar fruta desde un punto en el que el producto es volcado en agua a su entrada, hasta otro punto en el que el proceso de selección continúa. Esta necesidad de elevación viene dada porque normalmente el transporte del producto en proceso es por gravedad, de ahí que a mayor número de máquinas intervinientes en el proceso, mayor sea la altura del mencionado elevador.

Fue al analizar toda la línea de proceso de fruta en su conjunto cuando se detectó que la fuerte interacción entre todos los elementos que constituyen la línea, hace que todos ellos tengan que estar sincronizados para conseguir un funcionamiento constante y sin interrupciones. Analizando ya, elemento a elemento, se detectó que en el elevador de fruta existía una posible mejora. Esta mejora está relacionada con el volcado de producto a la cuba con agua, desde la que el elevador comienza a transportar fruta. El elevador de fruta no posee en la actualidad ningún dispositivo de control que regule este volcado (manual o automático), con lo que hay ocasiones en las que la banda de transporte del elevador de fruta no está transportando producto y otras ocasiones en las que la cuba está demasiado llena, dependiendo en gran medida de la pericia del operador encargado de la carga, provocando pues un funcionamiento irregular de toda la línea en su conjunto.

Es por ello, que se decidió enfocar este proyecto en el desarrollo de una solución que corrigiera el actual inconveniente que supone la falta de algún medio de control, del volcado de producto, en los elevadores de fruta, con la finalidad de aumentar la carga media del elevador de fruta y por ende, la producción media de toda la instalación en su conjunto.

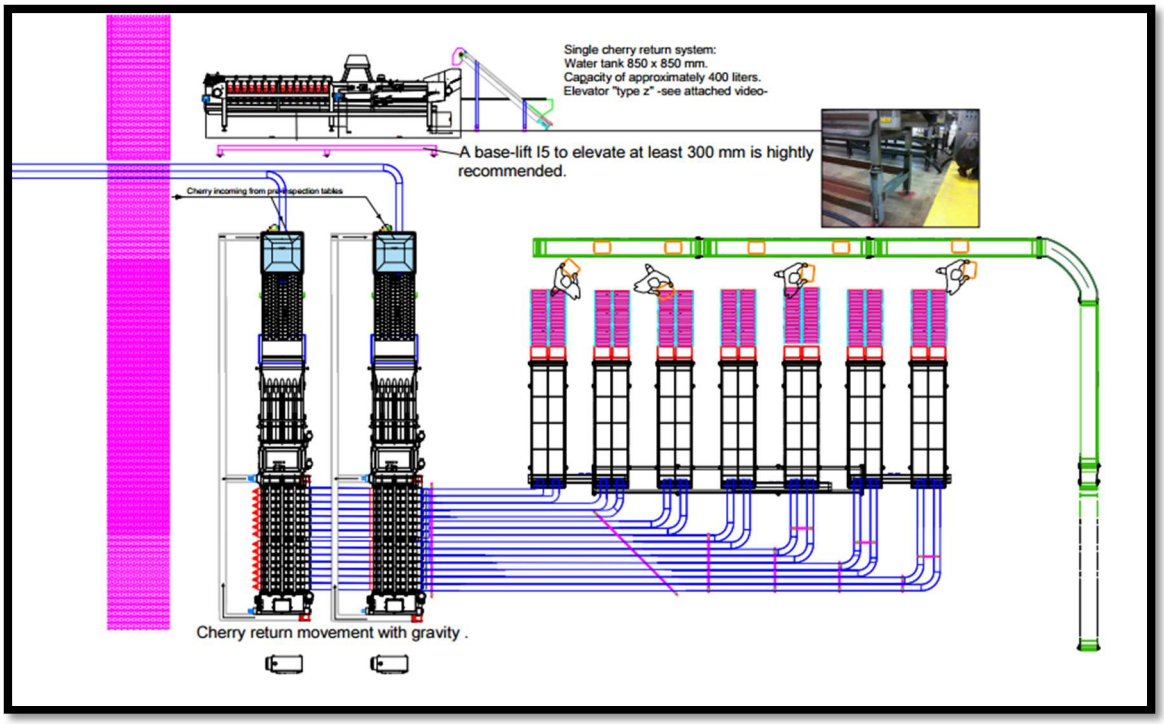


Imagen 1: *Perfil (parte superior de la imagen) y planta (parte inferior de la imagen) de un elevador de fruta.*

2.2. OBJETIVO DEL PROYECTO

El objetivo del presente proyecto es el desarrollo de una solución, tanto a nivel de software como de hardware, capaz de detectar, mediante el uso de visión artificial, la cantidad de banda de transporte vacía en un elevador de fruta y que en función de la misma, active una señal visual hacia el operador de volcado de fruta, en el caso de no existir volcador automático, o una señal de control, en el caso de que sí que se disponga de un volcador automático, para que el volcador de fruta dosifique más producto.

2.3. PLANTEAMIENTO DE LA SOLUCIÓN

Para conseguir nuestro objetivo, la solución que en este proyecto se plantea, ante la necesidad anteriormente expuesta, consiste en el desarrollo de un dispositivo (solución), capaz de controlar el volcado de producto en los elevadores de fruta.

Para lograrlo, se ha desarrollado una solución basada en visión artificial que se encarga de adquirir con una cámara imágenes de la parte de banda de transporte del elevador de fruta a controlar. Tras ello, se van analizando las imágenes adquiridas y en función del resultado del análisis, se envía la orden de volcar producto o de parar de volcarlo.

El análisis es realizado por una aplicación que será ejecutada en un microordenador y sobre la que el usuario podrá fijar unos parámetros de configuración con los que establecer las consignas de trabajo del sistema. Para que el usuario pueda modificar estos parámetros, la solución dispone de un interfaz gráfico que posibilita la comunicación entre ambos.

El software desarrollado (programa), se ha escrito en C++ y es ejecutado por una Raspberry Pi 2 (microordenador) que aparte de permitir analizar las imágenes adquiridas, también va a generar las señales de control que deberán indicar al operador manual o al control del volcador, el estado en que en cada momento deban estar.

Las imágenes a analizar se adquieren mediante una cámara monofocal cuya lente es de enfoque infinito.

Todos ellos, cámara, Raspberry Pi 2, pantalla táctil y diversos componentes auxiliares como cables, van encapsulados en una caja cuyo diseño y fabricación también se desarrolla en el presente proyecto. Finalmente la caja se realiza mediante impresión 3D.

En resumen, la solución planteada en este proyecto consta de dos grandes partes, una parte es el desarrollo de software y la otra parte, el desarrollo de la caja que alberga todos los componentes descritos.

Para probar y demostrar que la solución funciona correctamente, se ha realizado un montaje simple compuesto por tres LED's, tres resistencias, una placa de montaje y cuatro cables de conexión. La idea de este montaje es probar y demostrar que, en efecto, el dispositivo está enviando la señal de arranque o parada, al volcador de fruta, cuando debe hacerlo.
El dispositivo final no poseerá este montaje de prueba. En su lugar habrá dos cables (desde la Raspberry Pi 2) que se conecten al volcador de fruta para hacerlo funcionar cuando deba hacerlo.

3. DESARROLLO DE LA SOLUCIÓN

3.1. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN

Tal y como se ha expuesto en el planteamiento de la solución, ésta está formada por:

1. Elementos físicos (hardware):

    - Una Raspberry Pi 2 (usamos la placa de la misma).
    - Una cámara USB con la que adquirir las imágenes a analizar.
    - Una pantalla táctil con conexión HDMI para la parte de imagen y USB para parte táctil.
    - WIFI USB.
    - Caja que contiene a todos los componentes.
    - Cables de conexión.
    - Montaje para prueba y demostración:

- LED's verde, rojo y azul.
- Tres resistencias de 560 Ω.
- Placa de montaje.
- Cuatro cables de conexión.

2. Software:

   ✓ Sistema Operativo de la Raspberry Pi 2.
   ✓ Aplicación desarrollada para que funcione nuestro dispositivo.

3. Herramientas de desarrollo.

Herramientas de desarrollo:

Para escribir el código de la aplicación que controla a nuestro dispositivo, se ha utilizado el compilador Visual Studio 2015, en su versión Visual Studio Community.
Esta versión del compilador es gratuita y está especialmente enfocada a la creación de aplicaciones para Windows, Android e iOS, aplicaciones web y servicios de nube, todo esto a nivel particular o académica como es nuestro caso.

Software:

El lenguaje de programación escogido para escribir la aplicación que rueda en la Raspberry Pi 2 es el C++. Se decidió utilizar este lenguaje puesto que es muy potente y a la vez no excesivamente complejo.
El Sistema Operativo con el que la Raspberry Pi 2 funciona es Windows IoT en su compilación 10.

Elementos físicos (hardware):

- ***Raspberry Pi 2:***
  El dispositivo seleccionado para que ruede la aplicación que controla a nuestra solución, es la Raspberry Pi 2 Modelo B.

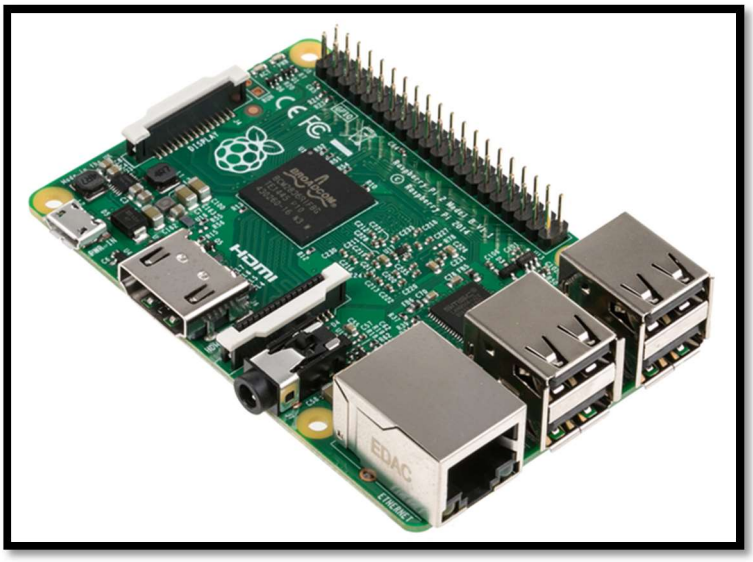


Imagen 2: *Raspberry Pi 2 Modelo B.*

- ***Cámara:***
  La cámara seleccionada para nuestro dispositivo es:
  "ELP 2.1mm Lens 1080p HD Free Driver USB Camera Module for Linux ELP-USBFHD01M-L21"

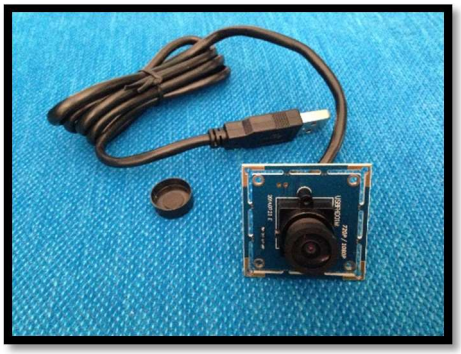


Imagen 3: *Imagen de la cámara y de su cable.*

- ***Pantalla:***
  Como se ha mencionado en el planteamiento, la solución dispone de un interfaz gráfico que permite y facilita la comunicación entre usuario y dispositivo. Este interfaz se muestra en una pantalla.
  La pantalla escogida para tal fin en nuestra solución es:
  "HDMI 5" 800x480 Display Backpack - With Touchscreen"
  Esta pantalla es táctil, compatible con la Raspberry Pi y económica.

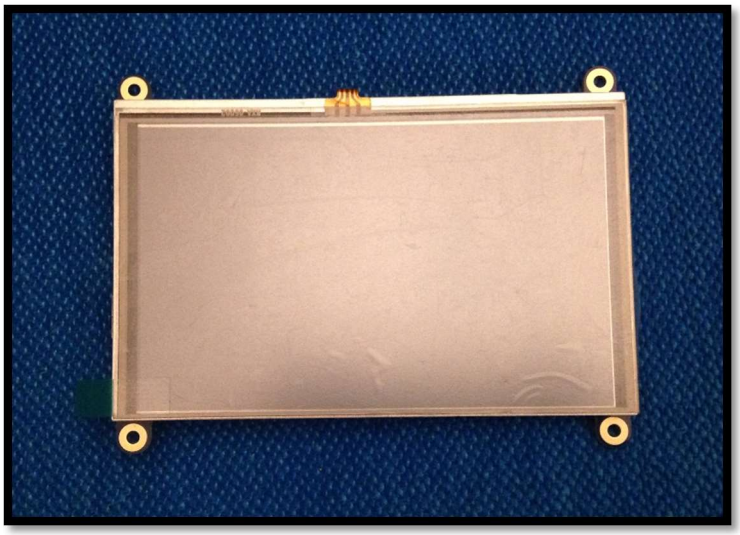


Imagen 4: *Vista de la parte frontal de la pantalla.*

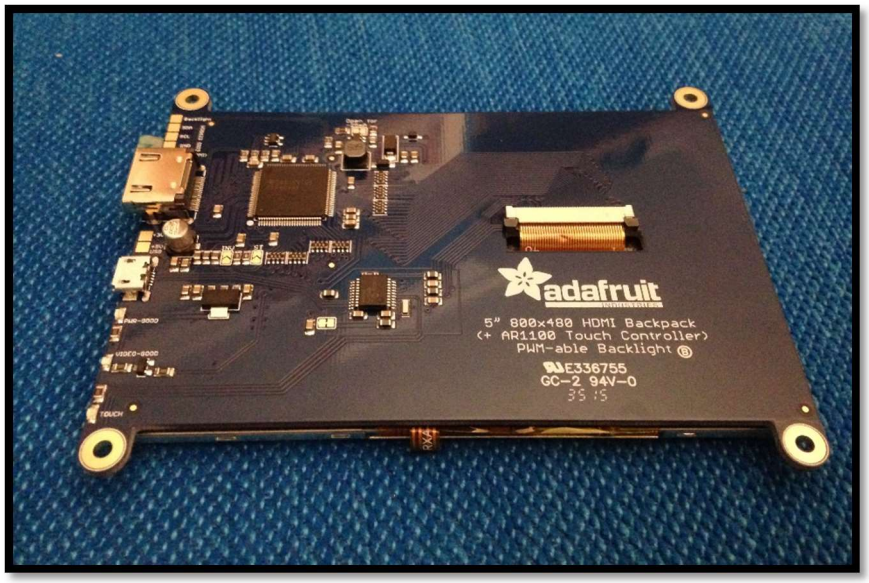


Imagen 5: *Vista de la parte trasera de la pantalla.*

La primera imagen corresponde con la parte delantera de la pantalla. La segunda imagen muestra la parte trasera en la que podemos apreciar, entre otras cosas, un conector HDMI y un micro USB para la alimentación de la pantalla.

- ***WiFi USB:***
  Conectamos a un puerto USB de la Raspberry Pi 2 un pin WiFi para dotar a nuestro dispositivo de conexión a la red.

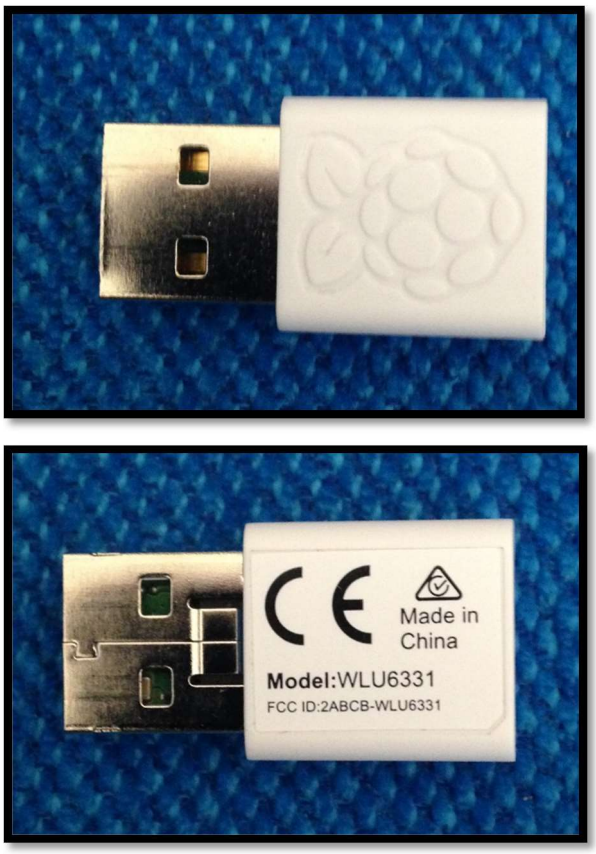


Imagen 6 y 7: *Pin WiFi USB visto por ambas caras.*

- ***Caja contenedora:***
  La caja que alberga a todos los componentes de nuestro dispositivo, se ha diseñado con la herramienta de Autodesk *123D Design* (esta herramienta es gratuita).

Imagen 8: *Caja contenedora con los componentes en su interior antes de cablear.*

- ***Montaje de prueba y demostración:***
  Este montaje se ha realizado sobre una placa de montaje *breadboard*. Consta de tres LED's, uno verde, uno rojo y uno azul. Cada uno de ellos representa un estado:
  - LED verde: el volcado de fruta está activo.
  - LED rojo: el volcado de fruta está inactivo.
  - LED azul: representa el paso a funcionamiento manual del volcador.

  A cada LED conectamos una resistencia de 560 Ω, cada una de ellas a la pata más larga del LED.
  El otro extremo de las resistencias va conectado al pin de la Raspberry Pi 2 "3.3V PWR".
  La pata corta de cada uno de los LED's va conectada, cada una, a un puerto GPIO de la Raspberry Pi 2 (GPIO 5 para el rojo, GPIO 6 para el verde y GPIO 13 para el azul), tal y como hemos definido en el archivo MainPage.xaml.h:

```
const int LED_RED_PIN = 5,
          LED_GREEN_PIN = 6,
          LED_BLUE_PIN = 13;
```

  El control del encendido de los LED's, así como a qué pin GPIO de la Raspberry Pi 2 van asociados, se realiza a través de una de las partes del código que se ha escrito para nuestra aplicación.

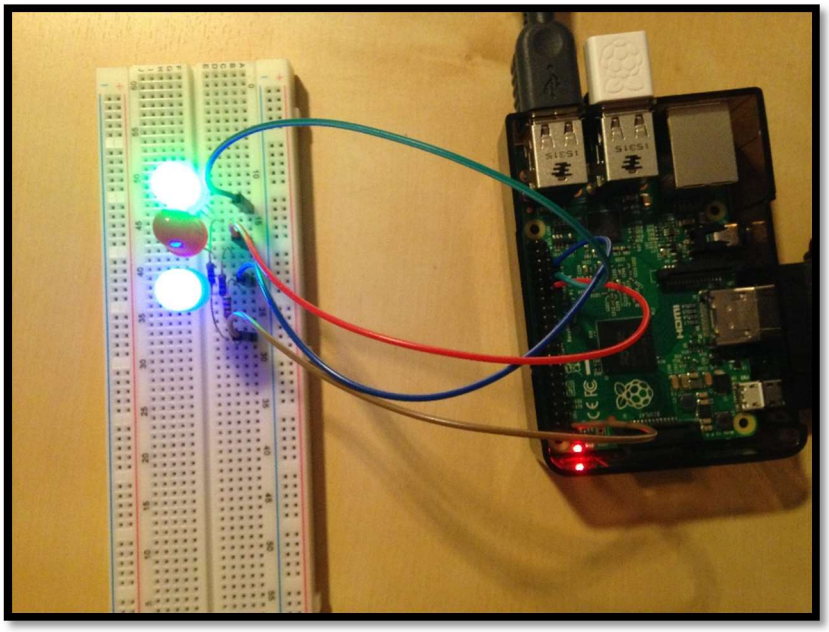


Imagen 9: *Imagen del montaje de prueba y demostración unido a la Raspberry Pi 2.*

  En la foto se puede observar la conexión del montaje completo y a la Raspberry Pi 2.
  Para saber la distribución de pines de la Raspberry Pi 2, se ha seguido el siguiente esquema:

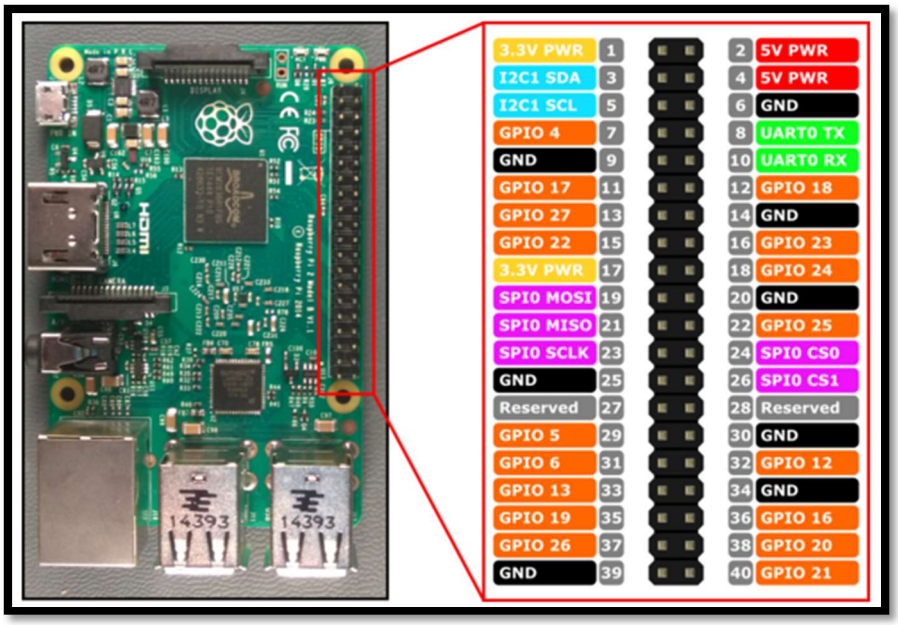


Imagen 10: *Esquema de la distribución de pines de la Raspberry Pi 2 (imagen extraída de Windows Dev Center → Windows IoT).*

- ***Cables:***
  Los cables necesarios para realizar las conexiones entre los diferentes componentes y a la alimentación son:
  - Cable USB a micro USB para la alimentación del dispositivo Raspberry Pi 2.
  - Fuente de alimentación USB de 4 puertos 5V, 7A, 2.4A máx por puerto.
  - Cable HDMI a HDMI para conectar Raspberry Pi 2 y pantalla.
  - Cable USB a micro USB para la alimentación de la pantalla a través de un cable adicional tipo puente datos-alimentación.
  - Cable USB de la cámara a la Raspberry Pi 2. Éste viene como parte de la propia cámara.
  - Cuatro cables de colores macho/hembra para el montaje de prueba.

3.2. JUSTIFICACIÓN DE LOS COMPONENES SELECCIONADOS

3.2.1. Herramienta de desarrollo (Visual Studio 2015)

Tal y como se ha expuesto anteriormente, el entorno de desarrollo escogido para la escritura de nuestra aplicación es la suite de desarrollo Visual Studio 2015, en su versión Visual Studio Community.

Elegimos esta herramienta de desarrollo y no otra porque, en este momento, es la única con la que se puede trabajar con Windows IoT.
En las fases de depuración del programa fue esencial el que el compilador soportase compilación y depuración cruzadas, gracias a ello, se pudo tener el dispositivo en el lugar adecuado para las pruebas, mientras que el ordenador de desarrollo en el que estaba instalado el compilador, se encontraba en un ambiente más de "oficina".

3.2.2. Software de la Raspberry Pi 2 (Windows IoT)

La elección de Windows IoT se ha basado en lo novedoso del enfoque que ha tomado Microsoft, que en clara pugna por el favor de los desarrolladores, ha diseñado un entorno de trabajo en clara competencia con Android Studio, que además de la parte de aplicaciones, también lleva embebida la parte de desarrollo de código para aplicaciones en entorno industrial.

Para una mayor información acerca de la estrategia de diseño de Microsoft en entornos industriales encontramos que el siguiente enlace es sumamente explicativo:
https://opcfoundation.org/wp-content/uploads/2014/08/MS-2_Shewchuk_OPC_Foundation.pdf

3.2.3. Raspberry Pi 2

La principal razón por la que se escogió la Raspberry Pi para nuestra solución, en lugar de haber elegido Arduino, es que la primera es un microordenador, mientras que Arduino es un microcontrolador. Por tanto, la Raspberry Pi era más adecuada para realizar el análisis de imágenes que nuestra solución debe llevar a cabo, puesto que un análisis de imágenes no es algo simple y requiere de mayor potencia de cálculo.
Es por ello por lo que se decidió que la Raspberry Pi era la más adecuada para el propósito de nuestra solución.

Se eligió la Raspberry Pi **2** puesto que cuando esta decisión fue tomada, ésta era la Raspberry más reciente que se encontraba a disposición del público.

3.2.4. Cámara

La elección de cámara vino determinada por:

- Precio asequible.
- Monofocal cuya lente fuera de enfoque infinito.

En un inicio se pensó usar la *Raspberry Pi Camera Board*, pero al final se decidió cambiar esta cámara por otra tipo USB. Esta decisión fue tomada en base a que el cable de la *RaspiCam* es muy corto y susceptible de malas manipulaciones pudiendo provocar problemas futuros y además a la fecha de redacción de esta memoria aún no existe un driver para Windows IoT.

3.2.5. Pantalla

En el momento en el que se decidió qué pantalla utilizar, se tuvo en cuenta que ésta no fuera demasiado pequeña para que así no se tuviese problemas con la visualización del interfaz gráfico. Se vio que con 5" era suficiente, puesto que tampoco se quería una pantalla excesivamente grande.
La pantalla, además, debía ser asequible, en cuanto a precio se refiere, compatible con la Raspberry Pi 2 y táctil.

Se encontró una pantalla que cumplía todos los requisitos anteriores y cuya conexión a la Raspberry Pi 2 era mediante los puertos de entrada/salida de la misma (pines GPIO de la Raspberry).
El problema surgió al conectarla y probarla. Resultó no funcionar debido a que el Sistema Operativo con el que trabajamos en la Raspberry Pi 2 es Windows IoT (tal y como se ha indicado anteriormente).
Al haber sido, este Sistema Operativo, desarrollado por Windows muy recientemente, a fecha de Mayo de 2016, todavía no existen drivers que permitan que esa pantalla, hecha en especial para su uso con Raspberry Pi 2, pueda funcionar con otro Sistema Operativo que no sea el que usualmente se usa en Raspberry (S.O de Linux: Raspbian, Debian, etc).

Frente a este inconveniente, se decidió cambiar la pantalla seleccionada en un inicio, cuya conexión era a los pines GPIO de la Raspberry Pi 2, por otra pantalla de las mismas características que la primera, pero que se conectara a la Raspberry por medio de un cable HDMI.

3.2.6. WiFi USB

El porqué del uso de un WiFi USB en nuestra solución es:

- Es necesario que la Raspberry Pi 2 esté conectada a la red a la que está conectado el ordenador, para poder ir probando mediante cross compilación remota la aplicación desarrollada.
- Se ha pensado incluir el WiFi USB a la solución y no solo usarlo durante el desarrollo de la misma, y así dotarla de la posibilidad de actualizar la aplicación por si más adelante se decide mejorarla en algún aspecto.

Cuando buscamos el pin WiFi USB que íbamos a usar para nuestra solución, tuvimos en cuenta que fuera compatible con la Raspberry Pi 2 y Windows IoT, y que no hubiera que cambiar nada en la configuración de la misma para poder usarlo, es decir, que al conectarlo al puerto USB de la Raspberry Pi 2 proporcionase la opción directa de conectarse a alguna de las redes disponibles en ese momento.

3.2.7. Caja que contiene a todos los componentes

La decisión de usar una caja que contuviera a todos los componentes del dispositivo desarrollado en este proyecto, fue tomada en base a que el hecho de reunir todo el dispositivo en una caja, proporcionaba comodidad de manejo del conjunto solución, así como protección de los componentes frente a agentes externos como agua.

Los elevadores de fruta son máquinas que tras su uso se han de limpiar. Para este fin los trabajadores usan mangueras de agua. Puesto que la caja tiene una alta probabilidad de sufrir salpicaduras de agua o incluso de mojarse entera mientras los operarios limpian el elevador, se decidió diseñarla de manera que fuera lo más estanca posible. De esta forma los componentes del dispositivo estarían resguardados del agua.
Estos componentes son: la placa de la Raspberry Pi 2, la pantalla táctil, la cámara y todos los cables de conexión necesarios.

Se quiso diseñar la caja contenedora porque de esta forma conseguíamos adecuarla a nuestras necesidades particulares.

Para su construcción se decidió usar una impresora 3D en lugar de un molde por cuestiones obvias de precio y plazo de entrega.

Para el diseño de la misma se utilizó un programa llamado *123D Design* de Autodesk, tal y como se ha mencionado anteriormente. Se escogió este programa en concreto ya que es gratuito, fácil de usar y compatible con la mayoría de impresoras 3D.

3.2.8. Montaje para prueba y demostración

La decisión de utilizar una placa de prueba para poder montar los leds ya se expuso en el planteamiento de la solución.
El montaje a usar se basó en uno similar al que aparece en un ejemplo de Adafruit-Microsoft llamado *Blinky sample* al que se puede acceder desde la siguiente dirección:

https://ms-iot.github.io/content/en-US/win10/samples/KitBlinky.htm

3.2.9. Cables de conexión

La selección de cables de nuestra solución se hizo en función de lo que ésta necesitaba y buscando flexibilidad, que no ocuparan mucho espacio para no necesitar que la caja que contiene todas las diferentes partes de la solución fuera más grande de lo necesario.

3.2.10. Otras justificaciones

- ***Entorno de trabajo:***

El entorno en el que se hará uso de la solución diseñada en este proyecto, dispone de luz suficiente como para posibilitar el trabajo humano, con lo que no hemos visto necesario el hecho de agregar a nuestro dispositivo una fuente de luz adicional.

- ***Dispositivo económico:***

Tal y como se ha expuesto ya en algunos de los componentes anteriores, al diseñar la solución, se decidió hacerla económica, por tanto, todos los componentes de la misma se han escogido teniendo en cuenta este factor, aparte de que cumplieran lo que nosotros necesitábamos.

- ***No necesaria la toma de imágenes en tiempo real:***

La velocidad a la que el elevador de fruta y el volcado de producto trabajan, fue decisiva a la hora de determinar si nuestro dispositivo iba a trabajar en tiempo real o si no era necesario.
La decisión final referente a este aspecto fue que no sería necesario el trabajo en tiempo real, puesto que para controlar el volcado de producto al elevador de fruta, no es necesario analizar el estado de la banda de transporte del elevador en todo momento, basta con ir tomando imágenes cada un cierto tiempo.

Al no trabajar en tiempo real, no necesitamos escoger una cámara que tuviera capacidad alta de trabajo (y por tanto también un precio superior), ni tampoco fue necesario realizar un código de programación que se encargara de trabajar en tiempo real.

3.3. DESCRIPCIÓN DETALLADA DE LOS COMPONENES SELECCIONADOS

3.3.1. Herramienta de desarrollo (Visual Studio 2015)

Visual Studio 2015 es un completo entorno de desarrollo integrado (IDE) para crear aplicaciones para Windows, Android e iOS, además de aplicaciones web y servicios de nube innovadores.
Posee herramientas y servicios para proyectos de cualquier tamaño o complejidad.
Los lenguajes que se pueden utilizar en Visual Studio 2015 son:

- ✓ C#
- ✓ Visual Basic
- ✓ F#
- ✓ C++
- ✓ Python
- ✓ Node.js
- ✓ HTML/JavaScript

Además dispone de planificación de sprint, depuración y creación de perfiles avanzados, pruebas automatizadas y manuales y DevOps con implementaciones automatizadas y supervisión continua.

Como ya se ha comentado anteriormente, la versión de Visual Studio 2015 que se ha utilizado en este proyecto es Visual Studio Community. Ésta es una herramienta gratuita, completa y ampliable para desarrolladores que crean aplicaciones que no son empresariales, como es el caso de nuestra solución. Visual Studio Community es gratuito para desarrolladores individuales, proyectos de código abierto, investigación académica, aprendizaje, educación y pequeños equipos profesionales.

Visual Studio 2015 Vs. Android Studio:
Al comparar ambos entornos de desarrollo integrado, encontramos que la mayoría de usuarios coinciden en que Visual Studio es un entorno de desarrollo muy potente para la sencillez de uso que presenta; así como que es un entorno mucho más maduro que Android Studio.
El problema que Android Studio presenta es que resulta más difícil su comprensión de uso. Numerosos usuarios apuntan que para principiantes es mucho mejor el empleo de Visual Studio dada su sencillez de uso y la amplia red de *supporters* que ya existen.
Otra ventaja que presenta Visual Studio es que está mejor documentado. Para poder empezar a usarlo no es necesario leerse ningún tutorial y se dispone de mucha información sobre la herramienta en internet.
Además de todo lo anterior, Visual Studio ofrece la posibilidad de crear aplicaciones universales mientras que Android Studio tan sólo permite la creación de aplicaciones para Android. Podemos observar esto en el hecho de que durante la instalación de Visual Studio, éste permite la posibilidad de instalar el Android SDK (*Software Development Kit*), para desarrollar

aplicaciones Android, además permite la utilización de las librerías Xamarin para poder desarrollar aplicaciones para iOS.

En la página de Visual Studio, encontramos información sobre el emulador de Android que éste posee, así como la posibilidad de descarga del mismo:

https://www.visualstudio.com/en-us/features/msft-android-emulator-vs.aspx

En resumen, se podría decir que Visual Studio es un entorno de desarrollo integrado maduro y potente, cuyo uso es sencillo, ya que está bien documentado y es, en parte, intuitivo. Además de esto, presenta la gran ventaja de la posibilidad de crear aplicaciones universales y multiplataforma (tanto para PC, Raspberry Pi o Smartphome). Todo ello le convierten en un entorno de desarrollo ideal para principiantes, como es nuestro caso.

3.3.2. Software de la Raspberry Pi 2 (Windows IoT)

El Sistema Operativo escogido para nuestra Raspberry Pi 2 es Windows IoT, como ya se ha mencionado anteriormente.
Este Sistema Operativo debe su nombre a *Internet of Things* (el internet de las cosas). Ha sido desarrollado por Microsoft recientemente, liberándose sus primeras versiones de trabajo en Julio de 2015.

Windows IoT es una versión de Windows 10 optimizada para pequeños dispositivos. Funciona con Raspberry Pi 2 y 3, así como con otras placas tales como Arrow DragonBoard 410c & MinnowBoard MAX.

Windows IoT ha sido desarrollado con el fin de ofrecer:

- A nivel doméstico: soluciones simples a problemas simples.
- A nivel industrial: soluciones simples pero robustas.
- Un frente de lucha frente a Google en el que competir en el segmento a las aplicaciones para móviles añadiéndole aplicaciones para dispositivos. Una simple cuenta nos puede arrojar luz sobre este planteamiento, como mucho, cada posible cliente puede tener un número de teléfonos móviles que ahora los expertos cifran en 1,5Uds por usuario, mientras que dispositivos capaces de adoptar internet y unirse a la red, se cifran en más de 5 Uds por usuario, esta cifra a diferencia de la de los teléfonos móviles sigue ascendiendo y por lo tanto generando un impresionante mercado que según los directivos de Microsoft puede llegar a poner en peligro la propia existencia de Google.

3.3.3. Raspberry Pi 2 modelo B

La Raspberry Pi que se seleccionó para nuestra solución, tal y como se ha dicho anteriormente, es la Raspberry Pi 2 modelo B.

La Raspberry Pi 2 modelo B, es la segunda generación Raspberry Pi.
En comparación con la Raspberry Pi 1, tiene:

- Una CPU quad-core de 900 MHZ ARM Cortex-A7
- 1 GB de memoria RAM

La Raspberry Pi 2 es completamente compatible con la Raspberry Pi 1.

Especificaciones técnicas de la Raspberry Pi 2 modelo B:

- Procesador Broadcom BCM2836 ARM7 quad-core funcionando a 900 MHz.
- 1 GB de memoria RAM.
- 4 puertos USB.
- 40 pines GPIO (*general purpose input/output*).
- Un puerto Full HDMI.
- Un puerto Ethernet
- Un puerto micro USB para la alimentación.
- Conector de audio de 3,5 mm combinado y vídeo compuesto.
- Interfaz de cámara (CSI).
- Interfaz de pantalla (DSI).
- Ranura para tarjetas micro SD.
- Núcleo de gráficos 3D VideoCore IV.

Algunos aspectos relevantes de la Raspberry Pi 2 modelo B:

- Al poseer 1 GB de memoria RAM, se pueden ejecutar aplicaciones más grandes y potentes.
- Posibilidad de proporcionar hasta 1.2 A al puerto USB. Esto permite que se conecten más dispositivos, cuya alimentación se proporciona a través del puerto USB, directamente a la Raspberry Pi. Esta característica requiere que la alimentación de la Raspberry Pi, a través de su micro USB, sea de 2 A.

Las medidas de la Raspberry Pi 2 modelo B, son iguales que las de su predecesora, la Raspberry Pi 1 modelo B+.

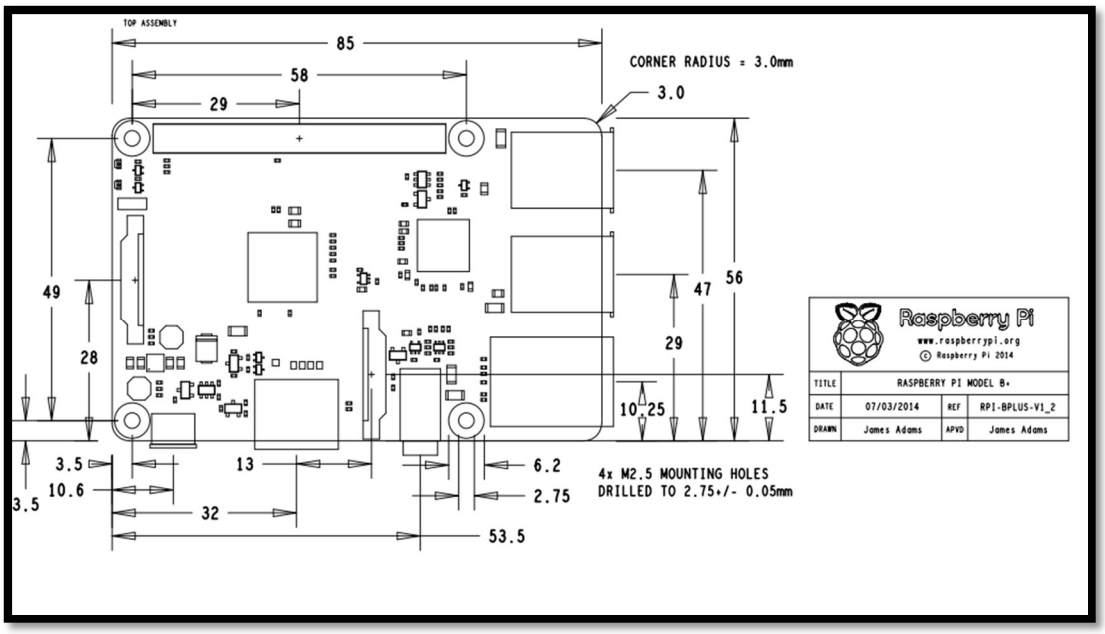


Imagen 11: *Planta acotada de la Raspberry Pi 1 Modelo B+.*

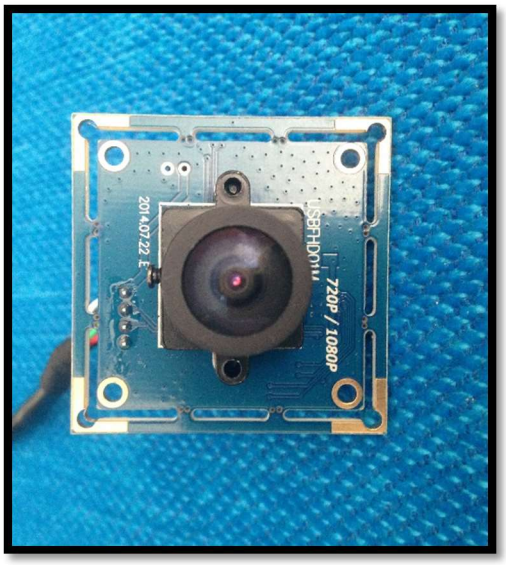
3.3.4. Cámara

La cámara utilizada en nuestro dispositivo, tal y como se ha dicho en la descripción detallada de la solución, es:
"ELP 2.1mm Lens 1080p HD Free Driver USB Camera Module for Linux ELP-USBFHD01M-L21"

Esta cámara posee una lente cuyo ángulo de visión es de 170 grados.

Además dispone de las siguientes características:

- Elevada velocidad de captura de imágenes, MJPEG 60 fps @ 1280 (H) x 720 (V) , 30 fps @ 1920 (H) x 1080 (V).
- Un sensor CMOS 1080P de alta resolución y bajo consumo.
- Alta definición de las imágenes con bajo ruido.
- CMOS muy luminoso lo que lo hace ideal para trabajos en entorno poco iluminados.
- Drivers para su uso en Linux, Windows XP, WIN CE, MAC, SP2.



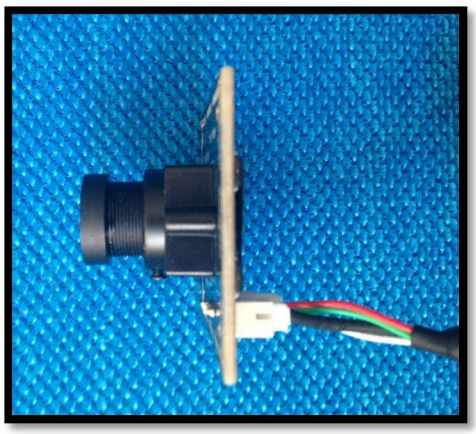


Imagen 12 y 13: *Imágenes frontal y lateral de la cámara.*

Las medidas más relevantes de la cámara son:

- Diámetro del objetivo: Ø = 14 mm
- Distancia entre las perforaciones para el anclaje de la cámara: 28 mm
- Altura de la cámara desde la placa: 22 mm
- Altura del objetivo de la cámara: 3.5 mm

3.3.5. Pantalla

La pantalla que finalmente se ha usado para nuestro dispositivo es:
"HDMI 5" 800x480 Display Backpack - With Touchscreen"

Sus principales características son:

- Pantalla de 5".
- Resolución de 800 x 480.
- Posibilidad de alimentar el dispositivo mediante un micro-B USB.
- Pantalla táctil.
- Conexión HDMI.
- Peso aproximado:
  - 106 g
- Dimensiones:
  - 120.7 x 75.8 x 8 mm

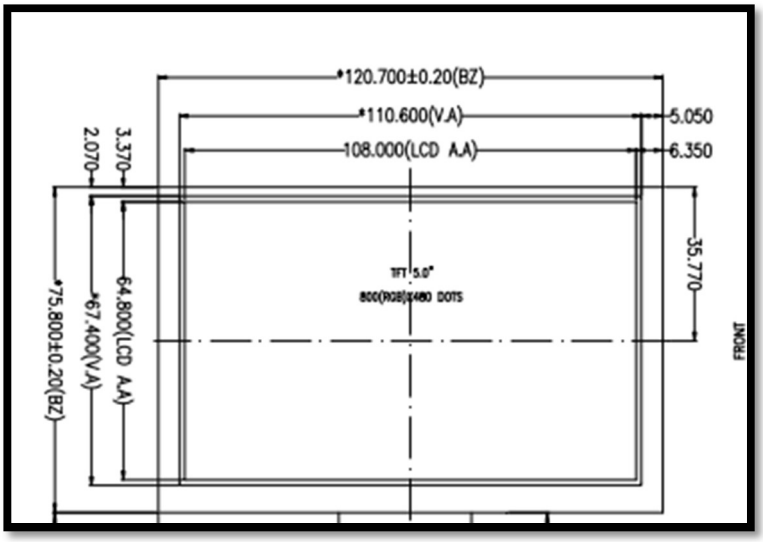


Imagen 14: *Planta acotada de la pantalla (imagen extraída de su DataSheet).*

3.3.6. Montaje para prueba y demostración

Para el montaje para prueba y demostración, que ya se ha explicado anteriormente, se ha hecho uso de:

- LED verde, LED rojo y LED azul.
- Tres resistencias de 560 Ω.
- Cuatro cables de conexión macho/hembra.

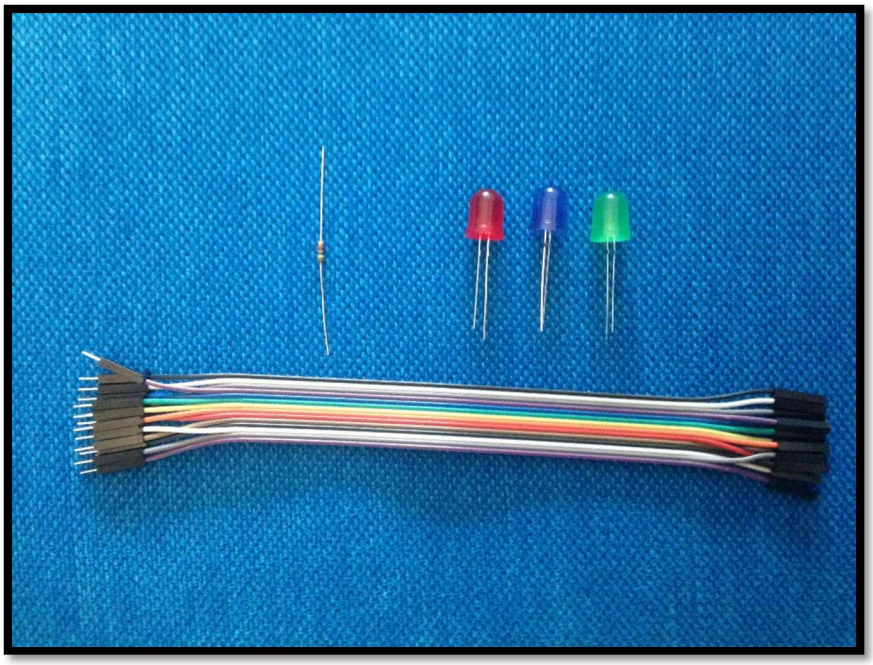


Imagen 15: *Material para la realización del montaje para prueba y demostración.*

El montaje se ha realizado sobre una placa de montaje *breadboard*, como ya se ha indicado en la descripción detallada de la solución.

3.4. ACOPIO DE MATERIALES Y PRESUPUESTO

3.4.1. Compra de los componentes

La compra de los componentes utilizados en nuestra solución se llevó a cabo en las siguientes páginas web:

- Raspberry Pi 2 modelo B: https://www.amazon.es (Amazon)
- Cámara: https://www.amazon.es/ELP-2-1mm-Driver-Camera-Module/dp/B01761VM7K/ref=sr_1_1?ie=UTF8&qid=1460656853&sr=8-1&keywords=ELP+2.1mm+Lens+1080p+HD+Free+Driver+USB+Camera+Module+for+Linux+ELP-USBFHD01M-L21 (Amazon)
- Pantalla táctil: https://www.adafruit.com/products/2260 (Adafruit)
- Resistencias, LED's, placa de montaje universal y cables macho/hembra de colores: https://www.amazon.es (Amazon)
- Adaptador WiFi USB para Raspberry Pi: https://www.amazon.es (Amazon)
- Anker Cargador USB de Pared 36W (7.2A) con 4 Puertos:

https://www.amazon.es/Anker-Cargador-Universal-110-240V-Motorola/dp/B00MVHKTZQ/ref=sr_1_2?ie=UTF8&qid=1462878828&sr=8-2&keywords=anker+7.2+A (Amazon)

- Cable USB a micro USB de 15 cm, negro (StarTech UUSBHAUB6IN): https://www.amazon.es/s/ref=nb_sb_noss?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&url=search-alias%3Daps&field-keywords=uusbhaub6in&rh=i%3Aaps%2Ck%3Auusbhaub6in (Amazon)
- Cable USB 2.0 de datos y alimentación, tipo A/M+A/M-A/H (15 cm) (Nano Cables 10.01.1900): https://www.amazon.es/Nano-Cables-10-01-1900-alimentaci%C3%B3n-cent%C3%ADmetros/dp/B00AKBPH1Q/ref=sr_1_1?ie=UTF8&qid=1462879563&sr=8-1&keywords=10.01.1900 (Amazon).
- Cable USB a micro USB para alimentación Raspberry Pi 2:
  Material propio, valor estimado en presupuesto.

3.4.2. Descarga y preparación del software

Para poner a punto el ordenador con el que se trabajó en la realización de la aplicación de nuestra solución y la Raspberry Pi 2, se siguieron los pasos de un tutorial de *Windows Dev Center*. Este tutorial es una guía sobre los primeros pasos a dar antes de poder desarrollar una aplicación con el Sistema Operativo Windows IoT.

En primer lugar, está la ***preparación del ordenador de trabajo***:

- ***Actualización del Sistema Operativo del ordenador a Windows 10***. La versión de éste debe ser 10.0.10240 o superior.
  https://www.microsoft.com/en-us/software-download/windows10

- ***Descarga de "Windows 10 IoT Core Dashboard"*** para preparar la Raspberry Pi 2 posteriormente.
  https://developer.microsoft.com/en-us/windows/iot/Downloads.htm

- ***Descarga e instalación de "Visual Studio Community 2015"*** desde el siguiente enlace o desde el mismo tutorial.
  https://www.visualstudio.com/es-es/downloads/download-visual-studio-vs.aspx

  Seleccionamos como tipo de instalación "Típico para desarrolladores de Windows 10".

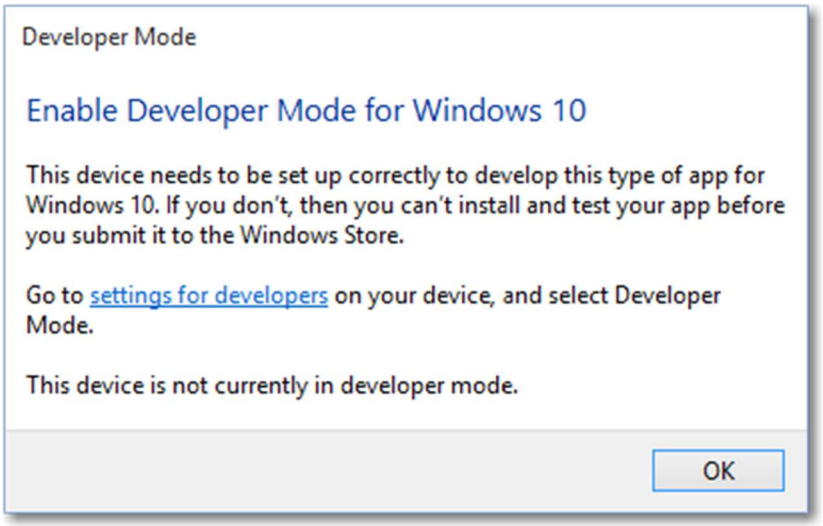
- ***Actualizamos Visual Studio con "Update 1"***, en caso de que no la hayamos instalado desde un principio.
  Durante el desarrollo final de la aplicación se ha liberado el Update 2 pero se ha decidido finalizar y probar completamente la aplicación con el Update 1 a fin de no añadir innecesarios problemas debidos a previsibles inestabilidades iniciales del Update 2.
  https://www.microsoft.com/es-es/download/details.aspx?id=49989

- ***Validación de la instalación de Visual Studio***, puesto que la versión requerida es 14.0.24720.00 *Update 1*.

- ***Instalación de "Windows IoT Core Project Templates".***
  https://visualstudiogallery.msdn.microsoft.com/55b357e1-a533-43ad-82a5-a88ac4b01dec

- ***Habilitación del dispositivo para desarrollar.***
  Para ello seguimos las instrucciones a las que nos redirige el tutorial:

  "Si usas Visual Studio en un dispositivo Windows 10 y abres una solución para una aplicación de Windows 8.1 o Windows 10, se te pedirá que habilites el dispositivo con este cuadro de diálogo. Es necesario habilitar el dispositivo para poder usar los diseñadores y depurar la aplicación.



  Cuando veas este cuadro de diálogo, haz clic en la configuración para desarrolladores para ir directamente a la página Actualización y seguridad, tal como se muestra a continuación."

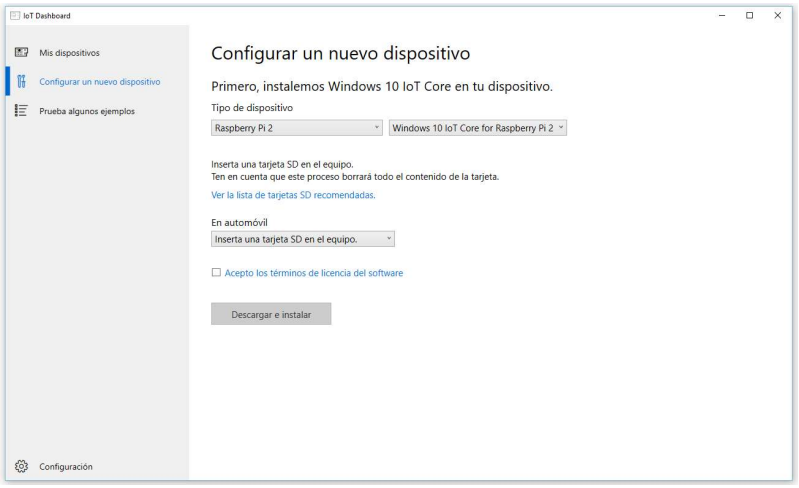
- En Visual Studio 2015:
  Nuevo proyecto → Extensibility → ***Instalación de las "Extensibility tools de Visual Studio Update 1".***

Tras realizar todo lo anterior y siguiendo el tutorial mencionado, la siguiente parte es la ***preparación de la Raspberry Pi 2***:

- Formateo de la tarjeta Micro SD y carga del sistema operativo Windows 10 IoT:

  ✓ Introducimos la tarjeta Micro SD en un adaptador para ordenador.
  ✓ Abrimos Windows 10 IoT Core Dashboard:

- Pulsamos "Configurar un nuevo dispositivo".



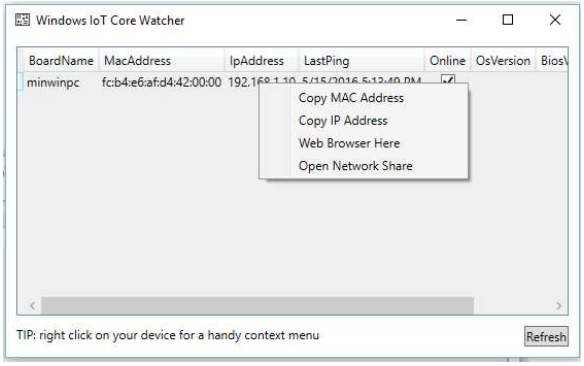
- Conectar la placa de la Raspberry Pi 2:

  ✓ Insertamos la tarjeta Micro SD en la Raspberry Pi 2.
  ✓ Conectamos el adaptador WiFi en uno de los puertos USB de la Raspberry Pi 2.
  ✓ Alimentamos la Raspberry Pi 2 a través de su puerto micro USB.
  ✓ Conectamos la Raspberry Pi 2 a nuestra red seleccionándola desde el interfaz de usuario de la misma Raspberry Pi 2.
  ✓ Tras ello anotaremos la dirección IP que nos habrá devuelto el DHCP (Dynamic Host Configuration Protocol) de la red en la que estemos.

Para acceder a la Raspberry Pi 2, existen varios métodos, el primero de ellos es conectarse al servidor web que la Raspberry Pi 2 lleva incorporado, para ello seguimos estos simples pasos:

- Abrir Google.
- Escribir: http://192.168.1.17:8080
- Escribir usuario y clave.

La segunda es explorar los dispositivos IoT que existen en la red bien con el IoT Core Watcher o con el Windows 10 IoT Core Dashboard. Una vez sobre el dispositivo a configurar o explorar, abrir su correspondiente página web.



3.4.3. Presupuesto

El presupuesto de la solución desarrollada en el presente proyecto es:

ARTÍCULOS	Uds.	PRECIO USD	PRECIO €
Raspberry Pi 2 con tarjeta SD de 8 GB	1	39,99 USD	36,69 €
Cámara: "ELP 2.1mm Lens 1080p HD ELP-USBFHD01M-L21"	1	45,00 USD	41,28 €
HDMI 5" 800x480 Display Backpack - With Touchscreen	1	74,95 USD	68,76 €
Resistencia	3	0,00 USD	0,00 €
LED	3	0,20 USD	0,55 €
Placa de montaje universal "Breadboard"	1	4,00 USD	3,67 €
Adaptador WiFi USB para Raspberry Pi	1	9,99 USD	9,17 €
Cable macho/hembra colores	6	0,08 USD	0,44 €
Anker Cargador USB de Pared 36W (7.2A) con 4 Puertos PowerIQ Cargador de Red Universal 110-240V	1	17,43 USD	15,99 €
Cable USB a micro USB de 15 cm, negro (StarTech UUSBHAUB6IN)	1	1,78 USD	1,63 €
Cable USB 2.0 de datos y alimentación, tipo A/M+A/M-A/H (15 cm) (Nano Cables 10.01.1900)	1	2,33 USD	2,14 €
Cable HDMI a HDMI de alta velocidad, compatible con Ethernet, 3D y retorno de vídeo - 0,9 m	1	5,98 USD	5,49 €
Cable USB a micro USB (propio, valor estimado)	1	1,50 USD	1,38 €
Impresión 3D (cortesía del DIE)	1	- USD	- €
Gastos aduana	1	32,70 USD	30,00 €
TOTAL		235,93 USD	217,19 €

3.5. DESARROLLO DE LA APLICACIÓN SOFTWARE.

3.5.1. Consideraciones preliminares.

Quizás, uno de los aspectos que más ha restringido la legibilidad de los programas a medida que los mismos se han hecho grandes y complejos, es la elección de los nombres a usar para las variables y funciones que componen, tanto las clases, como las funciones contenidas en ellas. Y todavía más importante, cuando un programa alcanza cierto tamaño, es normal fragmentarlo en trozos más pequeños cada uno de los cuales es mantenido por diferentes personas o grupos. Como C sólo tiene un ruedo para lidiar con todos los identificadores y nombres de función, trae como consecuencia que todos los desarrolladores deben tener cuidado de no usar accidentalmente los mismos nombres en situaciones en las que pueden ponerse en conflicto. Esto se convierte en una pérdida de tiempo, se hace tedioso y en último término, es más caro.

El C++ Estándar tiene un mecanismo para impedir estas colisiones: la palabra reservada **namespace** (espacio de nombres). Cada conjunto de definiciones de una librería o programa se «envuelve» en un espacio de nombres, y si otra definición tiene el mismo nombre, pero está en otro espacio de nombres, entonces no se produce colisión.

El espacio de nombres es una herramienta útil y conveniente, pero su presencia implica que debe saberse usar antes de escribir un programa. Si simplemente se escribe un fichero de cabecera y se usan algunas funciones u objetos de esa cabecera, probablemente se reciban extraños mensajes cuando se compile el programa, debido a que el compilador no pueda encontrar las declaraciones de los elementos del fichero de cabecera. Después de ver este mensaje un par de veces se hace familiar su significado (que es: Usted ha incluido el fichero de cabecera pero todas las declaraciones están sin un espacio de nombres y no le dijo al compilador que quería usar las declaraciones en ese espacio de nombres).

Hay una palabra reservada que permite decir «quiero usar las declaraciones y/o definiciones de este espacio de nombres». Esa palabra reservada, es **using**. Todas las librerías de C++ estándar, están incluidas en un único espacio de nombres, que es std (por «standard»). Como ejemplo, si se desea usar la librería estándar, la directiva **using** sería:

```
using namespace std;
```

Esto significa que se quieren usar todos los elementos del espacio de nombres llamado std. Después de esta sentencia, ya no hay que preocuparse de si un componente o librería particular pertenece a un espacio de nombres, porque la directiva **using** hace que el espacio de nombres esté disponible para todo el fichero donde se escribió la directiva **using**.

Exponer todos los elementos de un espacio de nombres después de que alguien se ha molestado en ocultarlos, parece contraproducente, y de hecho, se deberá tener cuidado si se considera hacerlo. Sin embargo, la directiva **using** expone solamente los nombres para el fichero actual, por lo que no es tan drástico como suena al principio. (Pero se debe pensar dos veces antes de usarlo en un fichero cabecera, puesto que eso es temerario).

Existe una relación entre los espacios de nombres y el modo en que se incluyen los ficheros de cabecera. Antes de que se estandarizara la nueva forma de inclusión de los ficheros cabecera (sin el «.h» como en <iostream>), la manera típica de incluir un fichero de cabecera era con el «.h» como en <iostream.h>. En esa época los espacios de nombres tampoco eran parte del lenguaje, por lo que para mantener una compatibilidad hacia atrás con el código existente, si se escribía:

```
#include <iostream.h>
```

En realidad, significaba:

#include <iostream>

using namespace std;

Una vez explicada una de las novedades de programación que ha sido introducida en la compilación 11 de C++, pasamos, ya sin más, a explicar la estructura de la aplicación desarrollada.

Para desarrollar la aplicación, se ha partido de uno de los *templates* que Microsoft ha liberado en la versión Windows IoT.
Para esta aplicación hemos tomado como plantilla una de los múltiples ejemplos en los que se usa una cámara para adquirir imágenes. Una vez estas imágenes son adquiridas, y ya dentro de la aplicación desarrollada, éstas son procesadas para el propósito que nos ocupa, tal y como se explicará más adelante.

Tras numerosas pruebas, la aplicación elegida como base es *CameraGetPreviewFrame*. Esta aplicación en concreto maneja, en un entorno multiplataforma (App que puede rodar en un Smartphone, en un PC o en una Raspberry Pi), una cámara, que en nuestro caso será una cámara USB, adquiere imágenes, las muestra en un *stackpanel* y finalmente llega incluso a aplicarle una transformación de la imagen girando a verde la misma.

Además, se decidió trabajar con programación concurrente (asíncrona). En este tipo de programación puede haber varias tareas ejecutándose a la vez y que cada una de ellas tenga una duración indeterminada.
Una de las soluciones para manejar la concurrencia es la creación de listas de tareas. Normalmente en tareas asíncronas se puede necesitar la finalización de las mismas antes de continuar con la ejecución de alguna otra parte de código de la aplicación; en la nuestra hemos empleado indistintamente la señalización mediante semáforos y la vinculación al estado de la tarea cargada en la lista.
Como ejemplo de ello mostramos una porción de código contenida en la función **OnTick** en la que se crea una cola de tareas cargando en ella la llamada a la función de obtención de la imagen conectada al *frame*, una vez que cargamos la llamada a la función el hilo en el que el código se está ejecutando se suspende hasta la que la llamada a la función ha tenido lugar y ésta se ha ejecutado completamente. Ello es así ya que en el manejador de la cola de tareas hemos optado por usar **.then**, con lo que, y tal y como ya hemos mencionado, el hilo es suspendido. No obstante ello, puede llegar a darse el caso en que debido al gran tamaño del área de análisis, vuelva a ocurrir otro evento **OnTick** que deberemos procesar en su parte de mantenimiento de funcionalidad de los leds y señales adicionales, pero no dejando que se nos anide la llamada a la función de análisis de imagen hasta que la que se esté ejecutando se haya completado en su totalidad.

```
// Obtener la imagen que se ha conectado al frame de visualización
create_task(_mediaCapture->GetPreviewFrameAsync(videoFrame))
        .then([this](VideoFrame^ currentFrame)
{
        // Process the image
```

```
AnalizeImage(currentFrame->SoftwareBitmap);
```

3.5.2. Desarrollo del UI.

A partir de este ejemplo base es donde empieza ya nuestro trabajo de diseño de la aplicación. Éste empieza primero con el diseño del interfaz de usuario o UI (*User interface*). Dado que el mismo debe ser ubicado en una pequeña pantalla de 5", se ha tenido que hacer un cuidadoso diseño de los botones y de su funcionalidad, dotando a muchos de ellos de funcionalidad contextual (son mostrados de distinta manera y actúan de distinta manera en función del estado de la aplicación).
Tras diversos procesos de desarrollo (prueba y error), el resultado que se ha validado finalmente es:

Imagen 16: *Interfaz de usuario (UI) de la aplicación desarrollada en este proyecto.*

3.5.3. Desarrollo del código de la aplicación.

Sobre la base obtenida desde *CameraGetPreviewFrame* se va a añadir la funcionalidad necesaria para soportar los siguientes bloques funcionales:

- Entradas/salidas del bloque I/O de la Raspberry Pi.
- Máquina de estados, la misma es movida por una interrupción generada por el objeto *Timer* que apuntará a la función de gestión de los estados del sistema, de entre los que uno de ellos será el análisis de las imágenes capturadas.
- Atención a los eventos disparados por cada uno de los controles instalados en el UI, de entre los que quizás, el más reseñable es de dimensionamiento del área de imagen a analizar por la aplicación.

Vamos a iniciar la disección del código generado siguiendo la estructura lógica de cualquier programa desarrollado en C++.

Constructor. `MainPage::MainPage()`
En él, fundamentalmente inicializamos los siguientes dispositivos, bien llamado a sus correspondientes constructores si son una clase o a las funciones inicializadoras en el caso de que estén dentro de esta aplicación. Es también responsabilidad del constructor el inicializar todas las variables globales que deban ser manejadas por la aplicación.

- Ventana o *User Form*, ésta es construida por **`InitializeComponent();`**
- Sistema de entradas/salidas I/O, éste es construido por **InitGPIO()**;
- Gestión del *display* en cuanto a orientación, tamaño, etc. Gestión del sistema de vídeo (Media).
- Como nosotros trabajamos en una aplicación y ésta puede ser puesta en primer plano o ser ocultada, deberemos crear los manejadores que recibirán estos eventos, para en esos manejadores, tomar las oportunas medidas de liberación de recursos, memoria, etc. Los manejadores de los eventos *Resuming* y *Suspending* han de ser guardados para que el destructor de la aplicación los devuelta a su estado inicial y puedan apuntar a las direcciones que existían antes de ser creada nuestra aplicación. Es muy importante no omitir estos pasos ya que de hacerlo comprometeríamos seriamente la estabilidad de la plataforma en la que estemos rodando.
- Máquina de estados, ésta descansa sobre un objeto *Timer* encargado de disparar eventos temporizados que a su vez llamarán al manejador de los mismos que hayamos definido y en el que gestionaremos todos los aspectos de gestión de botones y análisis de la imagen. En nuestro caso el manejador de los eventos de tipo *Timer* es apuntado al manejador **MainPage::OnTick** y la temporización la establecemos cada 100ms, tras ello y para arrancar el proceso de *Timer* llamamos a la función *Start* del *Timer* creado **timer_**. Mencionar que tras arrancar el **timer_** se empezará a contar el intervalo que se ha definido y por ello el inicio del **timer_** se colocó al final del constructor.
- Inicializamos el área de imagen que será analizada y todas las variables que definen el análisis de la imagen. Se había pensado escribir un simple trozo de código para crear y mantener un archivo en el que guardar la configuración de color patrón y zona de análisis, pero se ha valorado no hacerlo ya que es simple la configuración de la aplicación y la escritura en la tarjeta micro SD plantea algunos aspectos conflictivos en función de su formateo.
- Ya por último, posicionamos el selector de botones en el primero de ellos **textBox_ImageSet**.

Para una mejor compresión de todo lo explicado, a continuación mostramos el código del constructor:

```
MainPage::MainPage()
    : _mediaCapture(nullptr)
    , _isInitialized(false)
```

```
    , _isPreviewing(false)
    , _externalCamera(false)
    , _mirroringPreview(false)
    , _displayOrientation(DisplayOrientations::Portrait)
    , _displayRequest(ref new Windows::System::Display::DisplayRequest())
    , RotationKey({ 0xC380465D, 0x2271, 0x428C,{ 0x9B, 0x83, 0xEC, 0xEA, 0x3B,
0x4A, 0x85, 0xC1 } })
{
    InitializeComponent();

       // Init the GPIO port to allow write the pin that allow the motor control
       InitGPIO();

    _displayInformation = DisplayInformation::GetForCurrentView();
    _systemMediaControls = SystemMediaTransportControls::GetForCurrentView();

    // Cache the UI to have the checkboxes retain their state, as the
enabled/disabled state of the
    // GetPreviewFrameButton is reset in code when suspending/navigating (see
Start/StopPreviewAsync)
    Page::NavigationCacheMode = Navigation::NavigationCacheMode::Required;

    // Useful to know when to initialize/clean up the camera
    _applicationSuspendingEventToken =
        Application::Current->Suspending += ref new SuspendingEventHandler(this,
&MainPage::Application_Suspending);
    _applicationResumingEventToken =
        Application::Current->Resuming += ref new EventHandler<Object^>(this,
&MainPage::Application_Resuming);

       // Build the dispacth timer to shot the image processing
       timer_ = ref new DispatcherTimer();
       TimeSpan interval;

       // Set tick every 100 ms
       timerCounter = 0;
       timerCounterGetImage = 0;
       timerCounterToggle = 0;
       interval.Duration = 100 * 1000 * 10; //Converting 100ms to TimeSpan
intervals
       timer_->Interval = interval;
       timer_->Tick += ref new EventHandler<Object ^>(this, &MainPage::OnTick);

       // Sets average area at initial set
       analysisArea1.X0 = -1;
       analysisArea1.height = 100;
       analysisArea1.width = 100;
       analysisArea1.incSize = 50;

       // Establecer valor inicial del bitmap de trabajo
       completeImage.height = 0;
       completeImage.width = 0;

       averageAnalysis = 15;
       imageCountFilter = 15;
```

```
        imageAnalysisSet.r = 0;
        imageAnalysisSet.g = 0;
        imageAnalysisSet.b = 0;

        // Establecer los resultados iniciales en los textBox de resultados
        textBox_ImageCount->Text = " 0";
        textBox_ImageResult->Text = " R:  0 G:  0 B:  0";
        textBox_ImageDistance->Text = " 0";

        // Iniciar máquina de estados de control de uso de los botones
        SelPos = 0;
        ThicknessSel = 4,
        ThicknessReleased = 2;

        // Indicar que estamos por defecto parados, el botón pondrá marcha ya que
indica lo que haría si se le presiona
        imageComparation = EQUAL;
        ON_OFF_Status = OFF;

        // Refrescar contenido textBox y otros botones
        TextBoxRefresh();

        // Seleccionar el primer textBox
        textBox_ImageSet->BorderThickness = ThicknessSel;

        // Iniciar cuenta del timer
        timer_->Start();
}
```

Destructor. `MainPage::~MainPage()`
En esta aplicación, el destructor que monta el *template* es muy simple; detener el temporizador (**timer_**) para impedir un llamada al manejador del evento del mismo una vez se desmontan el resto de manejadores, y devolver al sistema los manejadores de *Resuming* y *Suspending*.

El código del destructor de nuestra aplicación es:

```
MainPage::~MainPage()
{
        // Prevent for a one unexpect shot
        timer_->Stop();

        Application::Current->Suspending -= _applicationSuspendingEventToken;
        Application::Current->Resuming -= _applicationResumingEventToken;
        _systemMediaControls->PropertyChanged -=
        _mediaControlPropChangedEventToken;
}
```

Inicialización de la cámara.
Toda esta parte es montada por el *template* y básicamente se basa en la gestión exclusiva de un recurso único del sistema, en este caso la cámara. Cada vez que es llamado el evento *Resuming*, que en nuestro caso es manejado por `void MainPage::Application_Resuming (Object^, Object^)`, se llama a la función **InitializeCameraAsync()** que es ejecutada como una función asíncrona con respecto a la llamada de la función, obteniendo al ser ejecutada la función, la

cámara instalada de forma exclusiva, bien sea la del propio sistema (en el caso de un Smartphone), externa o generando una excepción si ninguna cámara es encontrada, que en el caso de un Smartphone podría deberse a que otra aplicación no hubiese devuelto el recurso cámara al ser llamado el evento *Suspending* de la mencionada aplicación.
Quizás la mejor manera de entender los procesos que se ejecutan en los manejadores de *Suspending* y *Resuming* sea hacerlo simplemente leyendo el código que el *template* ha colocado en ellos.

```
void MainPage::Application_Suspending(Object^,
Windows::ApplicationModel::SuspendingEventArgs^ e)
{
    // Handle global application events only if this page is active
    if (Frame->CurrentSourcePageType.Name ==
Interop::TypeName(MainPage::typeid).Name)
    {
        _displayInformation->OrientationChanged -= _displayInformationEventToken;
        auto deferral = e->SuspendingOperation->GetDeferral();

             // En el proceso de liberación del recurso cámara esperamos a irnos
hasta que esta es efectivamente liberada
        CleanupCameraAsync()
        .then([this, deferral]()
        {
            deferral->Complete();
        });
    }
}

void MainPage::Application_Resuming(Object^, Object^)
{
    // Handle global application events only if this page is active
    if (Frame->CurrentSourcePageType.Name ==
Interop::TypeName(MainPage::typeid).Name)
    {
        // Populate orientation variables with the current state and register for
future changes
        _displayOrientation = _displayInformation->CurrentOrientation;
        _displayInformationEventToken =
            _displayInformation->OrientationChanged += ref new
TypedEventHandler<DisplayInformation^, Object^>(this,
&MainPage::DisplayInformation_OrientationChanged);

        InitializeCameraAsync();
    }
}
```

Analicemos la pareja de funciones que obtienen y liberan el recurso cámara, para ello empezamos con la función **`InitializeCameraAsync();`**.Mediante una llamada a una función de numeración asíncrona, obtenemos los dispositivos de tipo cámara de los que dispone el sistema, esta llamada por motivos obvios necesitamos que se complete antes de analizar los dispositivos obtenidos. En función de la tipología del dispositivo obtenido, actuaremos de una u otra manera.

En el caso de una cámara interna, instalada en el propio dispositivo, comprobaremos si es una cámara trasera o frontal, y en el caso de obtener acceso a la cámara delantera, giraríamos en modo espejo ésta última.
En el caso de encontrar una cámara externa, que será nuestro caso, levantaremos un *flag* para indicar que la cámara es externa. Una vez obtenida la cámara que pueda ser usada, se llamará a **InitializeAsync** de la cámara obtenida y tras ello a **StartPreviewAsync()** para inicializar el proceso de volcado de la imagen de la cámara al panel de visualización situado en nuestro UI.

Si ninguna cámara es encontrada o no podemos inicializarla satisfactoriamente, se dispara una excepción y se escribe una línea de texto a mostrar en la ventana de depuración (*debugging*).

```
/// Initializes the MediaCapture, registers events, gets camera device
information for mirroring and rotating, and starts preview
/// </summary>
/// <returns></returns>
task<void> MainPage::InitializeCameraAsync()
{
    WriteLine("InitializeCameraAsync");

    // Attempt to get the back camera if one is available, but use any camera
device if not
    return
FindCameraDeviceByPanelAsync(Windows::Devices::Enumeration::Panel::Back)
        .then([this](DeviceInformation^ camera)
    {
        if (camera == nullptr)
        {
            WriteLine("No camera device found!");
            return;
        }
        // Figure out where the camera is located
        if (camera->EnclosureLocation == nullptr || camera->EnclosureLocation-
>Panel == Windows::Devices::Enumeration::Panel::Unknown)
        {
            // No information on the location of the camera, assume it's an
external camera, not integrated on the device
            _externalCamera = true; // Esta es nuestra cámara USB
        }
        else
        {
            // Camera is fixed on the device
            _externalCamera = false;

            // Only mirror the preview if the camera is on the front panel
            _mirroringPreview = (camera->EnclosureLocation->Panel ==
Windows::Devices::Enumeration::Panel::Front);
        }

        _mediaCapture = ref new Capture::MediaCapture();

        // Register for a notification when something goes wrong
        _mediaCaptureFailedEventToken =
            _mediaCapture->Failed += ref new
Capture::MediaCaptureFailedEventHandler(this, &MainPage::MediaCapture_Failed);
```

```
        auto settings = ref new Capture::MediaCaptureInitializationSettings();
        settings->VideoDeviceId = camera->Id;

        // Initialize media capture and start the preview
        create_task(_mediaCapture->InitializeAsync(settings)).then([this]()
        {
            _isInitialized = true;

            return StartPreviewAsync();
               // Different return types, must do the error checking here since
we cannot return and send
        // execeptions back up the chain.
        }).then([this](task<void> previousTask)
        {
            try
            {
                previousTask.get();
            }
            catch (AccessDeniedException^)
            {
                WriteLine("The app was denied access to the camera");
            }
        });
    });
}
```

La función que es llamada en el evento *Suspending* es **CleanupCameraAsync()**, esta función está escrita a modo didáctico mediante el uso de un proceso *taskList* en vez de un proceso asíncrono con un *wait* o un *then* para esperar a la ejecución de la liberación. No vale la pena reseñar nada más adicional ya que esta función simplemente esperará a que se libere definitivamente el recurso cámara.

El *template* ha colocado otras funciones encargadas de la gestión de rotación de la cámara, de la localización de los dispositivos cámara mediante la enumeración de los mismos, de la liberación de la cámara en caso de mute y de error de adquisición, de almacenamiento como *bitmap* de la imagen capturada, de escribir *strings* de información y de error, etc. Estas funciones pueden ser utilizadas en el caso de añadir más prestaciones a la aplicación o gestionar de otra forma la liberación de la cámara.

Una vez analizado el arranque de la aplicación basándonos en el constructor de la misma vamos a describir ya las funciones específicas que se han escrito para:

- Inicializar el sistema I/O de la Raspberry.
- Tratar la máquina de estados.
- Atender a los eventos de pulsación sobre los controles del UI.
- Analizar la imagen.

Inicialización de sistema I/O de la Raspberry.

El sistema GPIO de la Raspberry no es imprescindible y por tanto no se ha diseñado la aplicación para que se detenga si no lo encuentra o no lo puede inicializar. Ello quiere decir que si ejecutásemos la aplicación en un Smartphone, esta funcionaría aunque evidentemente no encontrase el GPIO, por ello el diseño del UI contiene una señalización de marcha/paro del volcador para que un operador manual pudiese, mediante un simple Smartphone, analizar el contenido de un elevador de fruta y operar en función de su llenado.

También por la misma razón, no se ha colocado en el UI el led azul, ya que éste es para señalizar en un sistema externo la marcha forzada, cosa que no tendría razón de ser en el caso de no disponer de un volcador automático.

En la función **void MainPage::InitGPIO()**, se define el uso de los pines 5, 6, y 13 como pines de salida. Para ello se obtiene la dirección de cada uno de los pines mediante la llamada a la función **OpenPin,** por ejemplo: **pin_red = gpio->OpenPin(LED_RED_PIN)** y una vez abierto el pin seleccionado, se define si se va a trabajar con él como entrada o como salida mediante la llamada a la función **SetDriveMode(GpioPinDriveMode::Output)**.

Tras ello, se coloca el nivel deseado mediante la llamada a la función **pin_red->Write(GpioPinValue::Low)**.

La función **InitGPIO()** completa es:

```
// Inicialización del GPIO de la Raspberry PI
void MainPage::InitGPIO()
{
	auto gpio = GpioController::GetDefault();

	if (gpio == nullptr)
	{
		pin_red = nullptr;
		pin_green = nullptr;
		pin_blue = nullptr;
		WriteLine("No GPIO found");
		return;
	}

	// Inicializar LED (en la pantalla) y pin en el hardware
	// El sistema empieza parado, Red ON, Green y Blue OFF
	// La lógica es negativa, Low enciende el LED
	pin_red = gpio->OpenPin(LED_RED_PIN);
	pin_red->SetDriveMode(GpioPinDriveMode::Output);
	pin_red->Write(GpioPinValue::Low);

	pin_green = gpio->OpenPin(LED_GREEN_PIN);
	pin_green->SetDriveMode(GpioPinDriveMode::Output);
	pin_green->Write(GpioPinValue::High);

	pin_blue = gpio->OpenPin(LED_BLUE_PIN);
	pin_blue->SetDriveMode(GpioPinDriveMode::Output);
	pin_blue->Write(GpioPinValue::High);

	// Colocar el control LED en rojo
	LED->Fill = redBrush_;
```

```
}
```

Máquina de estados.

La máquina de estados está situada en la función de atención del evento **OnTimer** del temporizador creado en el constructor de la aplicación, en este evento gestionamos dos diferentes bloques de tareas, unas de alta frecuencia en cuanto a su ocurrencia y otra de baja. El nombre del manejador del evento del *Timer* es **void MainPage::OnTick(Object ^sender, Object ^args)**. El bloque de alta frecuencia se encarga de mantener los contadores y analizar el estado de la salida de activación del volcador y de la salida de señalización de marcha forzada. Con el fin de evitar rebotes en el arranque y paro del volcador cada vez que una señal de marcha/paro es activada, ésta se mantiene activa por un tiempo mínimo de 15 segundos. Tras revisar los contadores de temporización de las salidas de marcha/paro y el botón de manual, se ejecuta la función que refresca el contenido del UI **TextBoxRefresh()**.

La última parte del manejador del evento Timer, **OnTick**, se ejecuta cada 3 veces que es invocado el manejador del evento, **OnTick** y es quizás la más compleja, ya que crea una lista de tareas a la que lanza, para su ejecución, la función asíncrona de adquisición de la imagen **GetPreviewFrameAsync(videoFrame)**, cuando ésta termina, entonces se ejecuta el bloque de código incluido en **.then**, donde está la llamada a la función encargada de analizar la imagen adquirida **AnalizeImage(currentFrame->SoftwareBitmap)**.
Tras ello, "empujamos" a la pila de tareas creada, el redibujo del bitmap de trabajo y cuando esta tarea es completada, se llama a la función que ha de redibujar el control del UI encargado de mostrar la imagen resultado del análisis.

El código del manejador del evento Timer **OnTick** es:

```
// Gestión del evento del temporizador
void MainPage::OnTick(Object ^sender, Object ^args)
{
	// Incrementar contadores
	++timerCounterToggle;
	++timerCounter;
	++timerCounterGetImage;

	// Aquí gestionamos la máquina de estados del sistema y de la captura de
	imágenes
	if (ON_OFF_Status == OFF && imageComparation == DIFFERENT)
	{
		// Estamos parados y debiéramos estar en marcha, comprobamos si la
		histéresis ya ha terminado y podemos arrancar
		if (timerCounterToggle > imageCountFilter * 10)
		{
			Set_ON_OFF(ON);

			// Se cambia por toggle, ponemos LED azul apagado, por si
			estaba en marcha
			if (pin_blue != nullptr)
				pin_blue->Write(GpioPinValue::High);
		}
	}
	else if (ON_OFF_Status == ON && imageComparation == EQUAL)
```

```
        {
                // Estamos en marcha y debiéramos estar parados, comprobamos si la
                histéresis ya ha terminado y podemos parar
                if (timerCounterToggle > imageCountFilter * 10)
                {
                        Set_ON_OFF(OFF);

                        // Se cambia por toggle, ponemos LED azul apagado, por si
                        estaba en marcha
                        if (pin_blue != nullptr)
                                pin_blue->Write(GpioPinValue::High);
                }
        }

        // No haremos nada SI:
        // a) Estamos ON_OFF_Status == ON && imageComparation == DIFFERENT
        // b) Estamos ON_OFF_Status == OFF && imageComparation == EQUAL

        // Refrescar botones y controles
        TextBoxRefresh();

        // Comprobamos si hemos de adquirir y procesar una nueva imagen
        if (timerCounterGetImage > 2)
        {
                // Evitar anidamientos debidos a ser llamados mientras aún estamos
                analizando una imagen
                if (analysisInProgress)
                        return;

                analysisInProgress = true;

                // Volver a iniciar el conteo de adquisición de imágenes
                timerCounterGetImage = 0;

                // Get information about the preview
                auto previewProperties =
static_cast<MediaProperties::VideoEncodingProperties^>(_mediaCapture-
>VideoDeviceController-
>GetMediaStreamProperties(Capture::MediaStreamType::VideoPreview));
                unsigned int videoFrameWidth = previewProperties->Width;
                unsigned int videoFrameHeight = previewProperties->Height;

                // Create the video frame to request a SoftwareBitmap preview frame
                auto videoFrame = ref new VideoFrame(BitmapPixelFormat::Bgra8,
videoFrameWidth, videoFrameHeight);

                // Capture the preview frame
                create_task(_mediaCapture->GetPreviewFrameAsync(videoFrame))
                        .then([this](VideoFrame^ currentFrame)
                {
                        // Procesamos la imagen
                        AnalizeImage(currentFrame->SoftwareBitmap);

                        std::vector<task<void>> taskList;

                        // Show the frame (as is, no rotation is being applied)
```

```
                    // Copy it to a WriteableBitmap to display it to the user
                    auto sbSource = ref new
Media::Imaging::SoftwareBitmapSource();

                    taskList.push_back(create_task(sbSource-
>SetBitmapAsync(currentFrame->SoftwareBitmap))
                            .then([this, sbSource]()
                    {
                            // Display it in the Image control
                            PreviewFrameImage->Source = sbSource;

                            // Ya se han finalizado todas las tareas asíncronas,
                            liberamos el semáforo
                            analysisInProgress = false;
                    }));
            });
      }
}
```

Si analizamos detenidamente el código, nos llamará la atención el semáforo **analysisInProgress** que se ha tenido que introducir para que si el análisis de imagen se prolonga, por ejemplo si el cuadro de análisis es grande, no ocurra que se vuelva a crear otra lista de tareas y en ella se vuelva a cargar de nuevo todo el proceso anteriormente descrito para adquirir, procesar y mostrar el resultado del análisis de la imagen adquirida.

Atención a los eventos de pulsación sobre los controles contenidos en el UI.
Los procesos de atención a los eventos que generan los controles contenidos en nuestro UI son simples, únicamente vamos a atender a los eventos **button_Click**. Es el propio Visual Studio el encargado de generar el *template* de atención al evento cuando se selecciona el control en el cuadro de propiedades y se elige crear código para atender el evento **OnClick**.

Alguno de los botones incluidos en el UI son de funcionalidad contextual, dependiendo, tanto su contenido, como las acciones a tomar cuando son pulsados, del estado de la máquina de estados de la aplicación.
Dado lo simple de la gestión de estos eventos, solo vamos a comentar uno de ellos, el de decrementación. El manejador tomado como ejemplo es simple, **void MainPage::button_Click_DEC(Platform::Object^sender,Windows::UI::Xaml::RoutedEventArgs^ e)**, esta función tiene dos parámetros, la identificación del objeto que lo llama (por si se decide agrupar en un único manejador de eventos los eventos disparados por más de un control) y los parámetros del control que ha provocado el evento que pasan al manejador.

En nuestra aplicación manejamos los eventos de 4 controles únicamente, controles de aumento del área de análisis/adquisición de la imagen patrón, control de disminución del área de análisis, control de selección del control que recibirá el nuevo dato calculado y control de marcha/paro forzado. Los manejadores de estos eventos son:

```
void button_Click_INC(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e);
void button_Click_DEC(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e);
void button_Click_SEL(Platform::Object^ sender, Windows::UI::Xaml::RoutedEventArgs^ e);
void button_Click_ON_OFF(Platform::Object^sender,Windows::UI::Xaml::RoutedEventArgs^ e);
```

A nivel de ejemplo sólo mostramos el código del más simple de ellos, el del manejador del evento del control de marcha/paro:

```
// Arrancar o parar forzado el volcador
void MainPage::button_Click_ON_OFF(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e)
{
	// Aquí nos saltamos las protecciones de histéresis y señalizamos que el
sistema pasa a manual
	if (ON_OFF_Status == OFF)
		Set_ON_OFF(ON);
	else
		Set_ON_OFF(OFF);

	if ( pin_blue != nullptr)
		// Señalizamos que el sistema entra en manual hasta nuevo cambio de
estado. LED azul encendido
		pin_blue->Write(GpioPinValue::Low);
}
```

Análisis de la imagen adquirida.

La función de análisis de la imagen adquirida, es la más compleja de toda la aplicación. Vamos a enumerar primero sus bloques funcionales para después explicar más en profundidad el código escrito:

- Obtención del *bitmap* de trabajo y sus características. Se almacenan para comprobación de los tamaños máximos del área de análisis.
- Cálculo de las coordenadas del área de análisis. Esta tarea se hace al arrancar la aplicación y cada vez que mediante los controles de tamaño se varía el área de análisis.
- Señalización del área de análisis mediante un giro de color y un borde rojo.
- Cálculo del color medio y de la distancia de color al color patrón. Este bloque finaliza señalizando si la imagen es igual o distinta.

En el primer bloque, lo más reseñable es la creación del buffer que va a contener la imagen adquirida.

```
// Reinicializar cálculo del color medio
imageAnalysisResult.b = imageAnalysisResult.g = imageAnalysisResult.r = 0;
int total = 0;

// En el formato BGRA8, cada pixel es definido por 4 bytes
const int BYTES_PER_PIXEL = 4;

BitmapBuffer^ buffer = bitmap->LockBuffer(BitmapBufferAccessMode::ReadWrite);
IMemoryBufferReference^ reference = buffer->CreateReference();

Microsoft::WRL::ComPtr<IMemoryBufferByteAccess> byteAccess;
```

```
if (SUCCEEDED(reinterpret_cast<IUnknown*>(reference)-
>QueryInterface(IID_PPV_ARGS(&byteAccess))))
{
        // Obtener el puntero al buffer que contiene la imagen
        byte* data;
        unsigned capacity;
        byteAccess->GetBuffer(&data, &capacity);

        // Obtener toda la información sobre BitmapBuffer
        auto desc = buffer->GetPlaneDescription(0);
```

Lo primero que hacemos es solicitar un bloqueo del buffer que contiene la imagen, este *buffer* se nos ha pasado en la llamada a la función que se ha ejecutado en el evento **OnTick** al solicitar la adquisición de una imagen, realmente una de cada tres veces que **OnTick** es llamado.
Tras ello, se llama a **QueryInterface**, que nos permite obtener el puntero a un objeto del tipo interface.
La llamada a la función **GetBuffer** nos devuelve en **data** (puntero) la dirección del **buffer** que contiene la imagen y en **capacity** el tamaño de la misma. A continuación llamamos a **GetPlaneDescription** que en **desc** obtiene la información de la imagen.

Una vez obtenida la información de la imagen que vamos a analizar, comprobamos si se ha de recalcular de nuevo el área de análisis.

```
// Calcular coordenadas del área de análisis
if (analysisArea1.X0 == -1) {
        analysisArea1.X0 = desc.Width / 2 - analysisArea1.width / 2;
        analysisArea1.X1 = desc.Width / 2 + analysisArea1.width / 2;
        analysisArea1.Y0 = desc.Height / 2 - analysisArea1.height / 2;
        analysisArea1.Y1 = desc.Height / 2 + analysisArea1.height / 2;

        // Si es la primera vez que capturamos una imagen almacenaremos sus atributos
        if (completeImage.height == 0) {
                completeImage.height = desc.Height;
                completeImage.width = desc.Width;
        }
}
```

Tras ello se pasa ya a recorrer la imagen adquirida. En este recorrido hay que destacar que efectuamos dos procesos:

- Destacar y enmarcar la zona de análisis para facilitar al usuario la identificación de que zona de la imagen está siendo analizada.
- Calcular el color medio.

Todo ello tiene lugar dentro de un doble bucle **for** que recorre la imagen línea a línea y dentro de cada línea pixel a pixel.

El bucle es el siguiente:

```
// Recorrer el buffer pixel a pixel
for (int row = 0; row < desc.Height; row++)
{
	for (int col = 0; col < desc.Width; col++)
	{
	// Dirección del pixel a analizar
	auto currPixel = desc.StartIndex + desc.Stride * row + BYTES_PER_PIXEL *
	col;
	// Leo los tres canales b, g, r, dejando fuera el canal alpha
	auto b = data[currPixel + 0]; // Blue
	auto g = data[currPixel + 1]; // Green
	auto r = data[currPixel + 2]; // Red

	// Si estamos dentro del área de análisis…
	if (col >= analysisArea1.X0 && col <= analysisArea1.X1 && row >=
	analysisArea1.Y0 && row <= analysisArea1.Y1)
	{
		// Estamos dentro, comprobamos si estamos sobre el borde y entonces
lo coloreamos de rojo
		if ((col == analysisArea1.X0 || col == analysisArea1.X1) && row >=
		analysisArea1.Y0 && row <= analysisArea1.Y1)
			data[currPixel + 2] = 0xff;
		else if ((row == analysisArea1.Y0 || row == analysisArea1.Y1) &&
		col >= analysisArea1.X0 && col <= analysisArea1.X1)
			data[currPixel + 2] = 0xff;
		else {
			// Intensificar área de análisis para que se vea
			if (data[currPixel + 0] + 100 > 0xff)
				data[currPixel + 0] = 0xff;
			else
				data[currPixel + 0] = b + 100;

			if (data[currPixel + 1] + 100 > 0xff)
				data[currPixel + 1] = 0xff;
			else
				data[currPixel + 1] = g + 100;

			if (data[currPixel + 2] + 100 > 0xff)
				data[currPixel + 2] = 0xff;
			else
				data[currPixel + 2] = r + 100;

			// Calcular el color medio (solo dentro del área de análisis)
			imageAnalysisResult.b += b;
			imageAnalysisResult.g += g;
			imageAnalysisResult.r += r;
			++total;
			}
		}
	}
```

```
}
```

Iniciamos el proceso obteniendo las componentes de color de los canales r, g, b.
Si estamos sobre la línea del borde de la zona de análisis, la coloreamos de rojo mediante la asignación del valor 0xff (255) al pixel en curso. **data[currPixel + 2] = 0xff**; donde 2 es el byte que corresponde al color r (rojo).

Tras haber enmarcado la zona de análisis intensificando el canal r (colocando su valor a 0xff), procedemos a destacar la zona de análisis, para ello incrementamos el valor de cada canal en 100, siempre y cuando al hacerlo no lo saturemos, esta comprobación de saturación la realiza la comparación siguiente (ejemplo en el canal g (verde)):

```
if (data[currPixel + 1] + 100 > 0xff)
        data[currPixel + 1] = 0xff;
else
      data[currPixel + 1] = g + 100;
```

La segunda parte del proceso, calcular el color medio, la realizamos mediante el código siguiente:

```
// Hemos llegado al final del Bitmap, dividimos por el número total de píxeles encontrados
imageAnalysisResult.b /= total;
imageAnalysisResult.g /= total;
imageAnalysisResult.r /= total;

// Mostramos el RGB medio de la imagen capturada
String^ results = " ";
results += "R= " + imageAnalysisResult.r + " G= " + imageAnalysisResult.g + " B= " +
imageAnalysisResult.b;
textBox_ImageResult->Text = results;

// Análisis de la imagen, algoritmo simple
colorDistance = 0;
float average = averageAnalysis / (float)100;
float b, g, r;

b = abs((float)(imageAnalysisSet.b - imageAnalysisResult.b) / 100);
g = abs((float)(imageAnalysisSet.g - imageAnalysisResult.g) / 100);
r = abs((float)(imageAnalysisSet.r - imageAnalysisResult.r) / 100);

if (b > average)
        ++colorDistance;
if (g > average)
        ++colorDistance;
if (r > average)
        colorDistance += 2; // añadimos más peso en el canal rojo ya que la aplicación se
supone diseñada para procesar frutos fundamentalmente rojos
```

```
if (colorDistance >= 2)
        imageComparation = DIFFERENT; // Indica hay diferencia con la imagen, el sistema debería arrancar al volcador si el resto de condiciones lo permite
else
       imageComparation = EQUAL;
```

Lo más destacable de la verdadera zona de análisis de la imagen, es primero el cálculo del color medio dividiendo para ello el color suma total de cada canal, por el número total de píxeles de la zona de imagen analizada:

imageAnalysisResult.b /= total;

Tras ello mostramos en el **textBox** del UI el color R, G, B obtenido y por último calculamos la distancia de color con respecto al color patrón capturado y almacenado en **imageAnalysisSet.g** La distancia es calculada mediante la operación de comparación en porcentaje sobre 100 (%):
g = abs((float)(imageAnalysisSet.g - imageAnalysisResult.g) / 100);

Y si esta distancia es mayor que la definida en el valor que se ha almacenado en **average,** se incrementa el contador de distancia **colorDistance**. En el caso especial del canal r (rojo), la distancia se pondera al doble que la del resto de canales ya que suponemos que la aplicación se ha desarrollado para analizar frutos de color rojo.

```
if (r > average)
        colorDistance += 2;
```

Por último, si la distancia es mayor o igual que dos, definimos a la imagen adquirida como diferente.

La aplicación tiene algunos bloques de código no significativos que únicamente tienen como misión el refresco de los controles del UI, lo mostramos a continuación y no lo comentamos, ya que su simple lectura es autoexplicativa:

```
// Refrescar el contenido de todos los textBox de configuración
void MainPage::TextBoxRefresh()
{
       // Establecer valores en los textBox de configuración
       String^ results;

       results = " ";
       results += "R= " + imageAnalysisSet.r + " G= " + imageAnalysisSet.g + " B= " + imageAnalysisSet.b;
       textBox_ImageSet->Text = results;
       results = " ";
       results += analysisArea1.width;
       textBox_WSet->Text = results;
       results = " ";
       results += analysisArea1.height;
       textBox_HSet->Text = results;
```

```
        results = " ";
        results += averageAnalysis;
        textBox_AverageSet->Text = results;
        results = " ";
        results += imageCountFilter;
        textBox_ImageCountSet->Text = results;
        results = " ";
        results += timerCounterToggle;
        textBox_ImageCount->Text = results;
        results = " ";
        results += colorDistance;
        textBox_ImageDistance->Text = results;

        // Gestión del botón de marcha paro
        if (ON_OFF_Status == ON)
               results = " OFF";
        else
               results = " ON";

        button_ON_OFF->Content = results;
}
```

3.6. DESCRIPCIÓN DE LA CAJA CONTENEDORA DEL DISPOSITIVO

Esta caja, ha sido diseñada persiguiendo los objetivos siguientes:

- Ser lo más estanca posible.
- Tener el mínimo número de piezas móviles.

La necesidad de que la caja cumpla los dos objetivos nombrados anteriormente, viene dada por el hecho de que los elevadores de fruta se limpian diariamente usando agua expulsada por mangueras, con lo cual existe el riesgo de que nuestro dispositivo se moje por salpicaduras o por proyección directa.

Como se ha indicado con anterioridad, la caja debe contener una Raspberry Pi 2, que será el cerebro de nuestro dispositivo, una cámara con la que se adquirirán las imágenes a analizar y una pantalla táctil con conexión HDMI que permita al usuario interactuar con el dispositivo.
La única pieza móvil de la caja es la tapa situada en su parte superior y que además de permitir el acceso al interior de la caja, se encarga de albergar la pantalla táctil.

Entre la caja y la tapa, se ha diseñado una hendidura centrada en su contorno, el objetivo de la misma es que albergue una junta tórica para que al encajar la tapa con el resto de la caja, ésta junta tórica dote de estanqueidad al conjunto tapa, caja.

La tapa se ha diseñado con un agujero para albergar la pantalla táctil, ésta se coloca de abajo hacia arriba y se ancla a la tapa mediante unos pequeños tornillos. Para mantener la estanqueidad de todo el conjunto se ha empleado un pegamento flexible alrededor de la pantalla.
La parte derecha de la tapa está diseñada de forma que los cables de conexión de la pantalla, vayan por debajo hasta el lugar de conexión con la Raspberry Pi 2.

El resto de la caja es una única pieza, cuya parte superior tiene un entrante en el grosor de las paredes. El fin de este entrante es el de que la tapa descanse sobre él cuando se quiera cerrar la caja.

En la parte inferior de la caja va situada la cámara de forma que la pantalla ubicada en su lado opuesto, nos muestre la imagen que está adquiriendo la cámara. De esta forma, la utilización y colocación del dispositivo es más sencilla e intuitiva.
Para que la cámara pueda estar en la parte inferior de la caja, ésta tiene un agujero para que el objetivo de la cámara salga fuera. Para que la caja continúe siendo estanca, también se ha empleado pegamento flexible en la junta.
La cámara utilizada para nuestro dispositivo, tiene cuatro orificios para anclarla con tornillos a una superficie. Es por ello que alrededor del orificio de salida del objetivo de la cámara, se han colocado unos pequeños cilindros en las posiciones donde se ancla la cámara a la caja. La situación de estos cilindros viene determinada por las dimensiones y situación de los orificios para los tornillos de la cámara. La altura de los mismos se determinó teniendo en cuenta la altura de la cámara y la cantidad de ella que debía sobresalir al exterior de la caja.

La Raspberry Pi 2 se decidió colocar en el lateral derecho de la caja con los puertos de conexión hacia arriba. Tanto en la parte superior de este lateral como en la parte derecha del mismo (parte en la que se encuentra el micro USB para la alimentación y el puerto HDMI), se ha dejado espacio suficiente para que los cables que se tienen que conectar a la Raspberry Pi 2 puedan conectarse sin que se fuercen o se doblen excesivamente. Por tanto ambos laterales de la caja tienen las dimensiones de la Raspberry Pi 2 más las partes que se han previsto para los cables.
En este lateral, también se han diseñado cuatro cilindros en el lugar en el que se ancla la placa (Raspberry Pi 2) a la caja. Estos cilindros están colocados en la parte inferior del lateral en el que va situada la Raspberry Pi 2 y la posición de los mismos viene dada por los cuatro orificios que tiene la placa para ser anclada.

Cuando se diseñó la caja, aparte de tener en cuenta las medidas de cada componente (pantalla, cámara y Raspberry Pi 2), también se tuvo en cuenta que el tamaño fuera el adecuado para que los cables no estuviesen demasiado apretados, así como que los componentes tuvieran espacio para permitir una adecuada refrigeración de todos los componentes que ésta alberga.

El cable de alimentación del dispositivo completo, así como los cables que controlan el montaje de LED's, necesitan salir de la caja. Para ello se han diseñado dos orificios en uno de los laterales largos de la caja. Una vez impresa, se ha colocado en uno de los agujeros un prensaestopas y en el otro, un pasamuros, ambos se adhieren a la caja mediante un pegamento flexible.

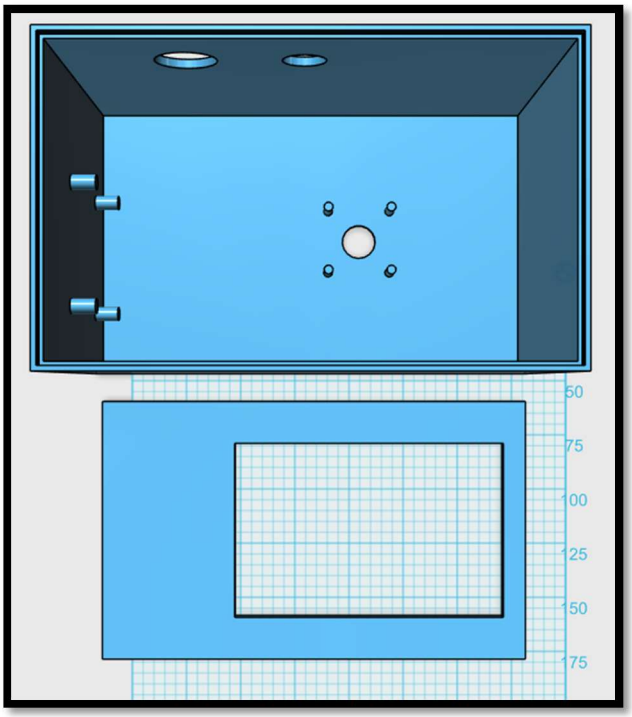


Imagen 17: *Planta de la parte superior e inferior de la caja que contiene al dispositivo completo.*

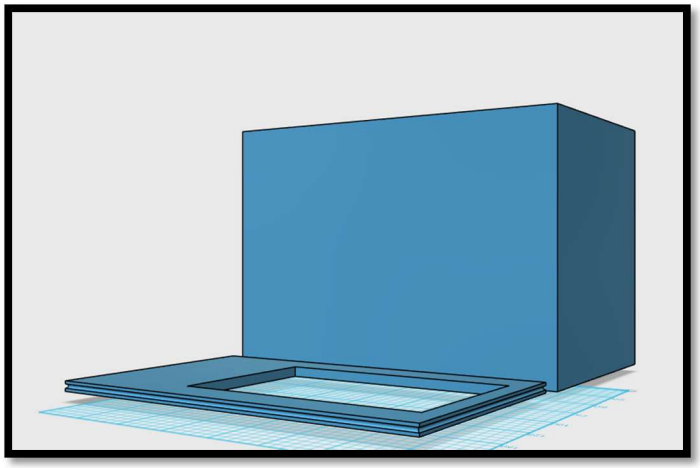


Imagen 18: *Vista de los laterales de la parte superior e inferior de la caja.*

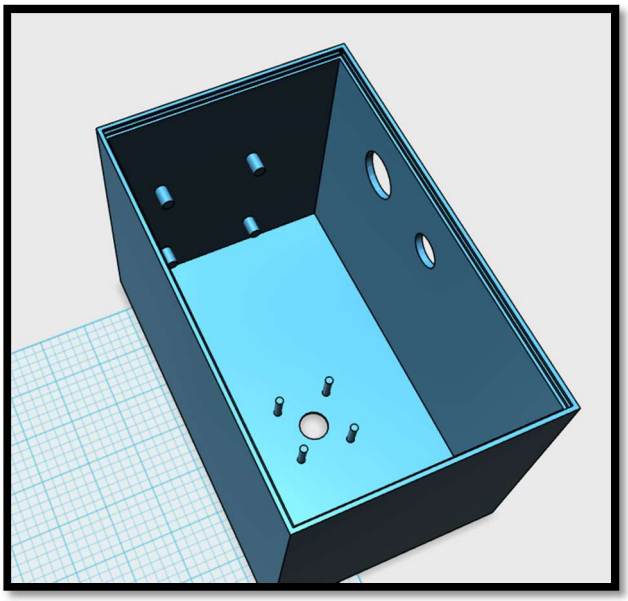


Imagen 19: *Vista del interior de la parte inferior de la caja.*

3.7. DISEÑO E IMPRESIÓN 3D DE LA CAJA CONTENEDORA DEL DISPOSITIVO

El diseño de la caja que contiene a la Raspberry Pi 2, a la cámara, a la pantalla y a los cables de conexión necesarios, se ha llevado a cabo mediante el uso de la herramienta gratuita 123D Design de Autodesk, descargada en:
http://www.123dapp.com/design

Una vez diseñada la caja, se nos plantearon varias opciones a la hora de imprimirla.
La primera de ellas fue acudir a un servicio web de impresión, cuyos precios medios para el volumen de caja obtenido, superaban en cualquier caso los 700 euros, llegando incluso a hacer viable, como segunda opción, la adquisición y montaje de una impresora para realizar la impresión de la caja.
La tercera opción surgió tras ponernos en contacto con el Departamento de Ingeniería Electrónica de la EPSA (DIE) y comentar al tutor de este proyecto, Juan Ramón Rufino, los problemas encontrados para afrontar la impresión de la caja. El tutor propuso, como tercera solución, realizar la impresión de la caja en el departamento anteriormente mencionado.

3.8. MANUAL DE USO DEL DISPOSITIVO

3.8.1. Interfaz de usuario

Imagen 20: *Interfaz de usuario (UI) de nuestra aplicación.*

Como se puede apreciar en la imagen, el interfaz de usuario (UI) de nuestra solución posee varios botones, algunos de ellos con funcionalidad contextual, como ya se ha explicado anteriormente. Para comenzar a usar la solución desarrollada en este proyecto, primero se debe conocer el funcionamiento del UI. Éste permite al usuario:

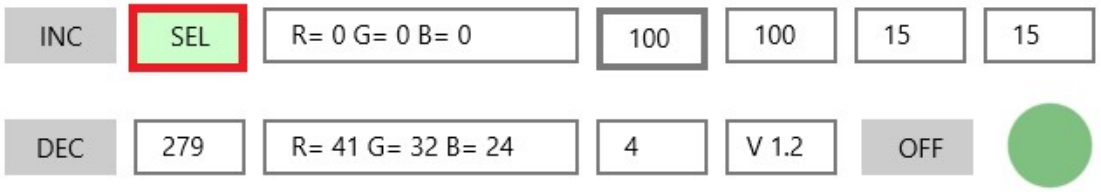
- Establecimiento del área de análisis.
- Captura de la imagen base a partir de la cual el análisis comparará.
- Forzar el arranque o paro del volcador de fruta cuando el operario lo desee.
- Establecimiento de la precisión del análisis.
- Definir el filtro de histéresis o tiempo de ON/OFF mínimo.

A parte de los botones que permiten al usuario modificar el funcionamiento de la solución, el UI de la misma también posee controles con los que se proporciona información al usuario:

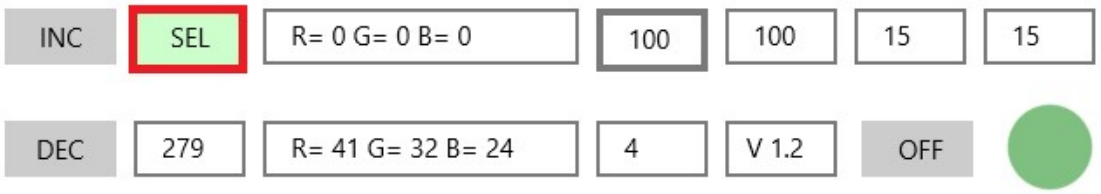
- LED que indica estado ON/OFF del volcador de fruta.
- Control que indica tiempo transcurrido desde el último cambio de estado.
- Control que indica la diferencia o distancia entre la imagen base y la imagen a analizar.
- Control que indica la versión de la aplicación.
- Controles que muestran el RGB de color de la imagen patrón y de la imagen actual.

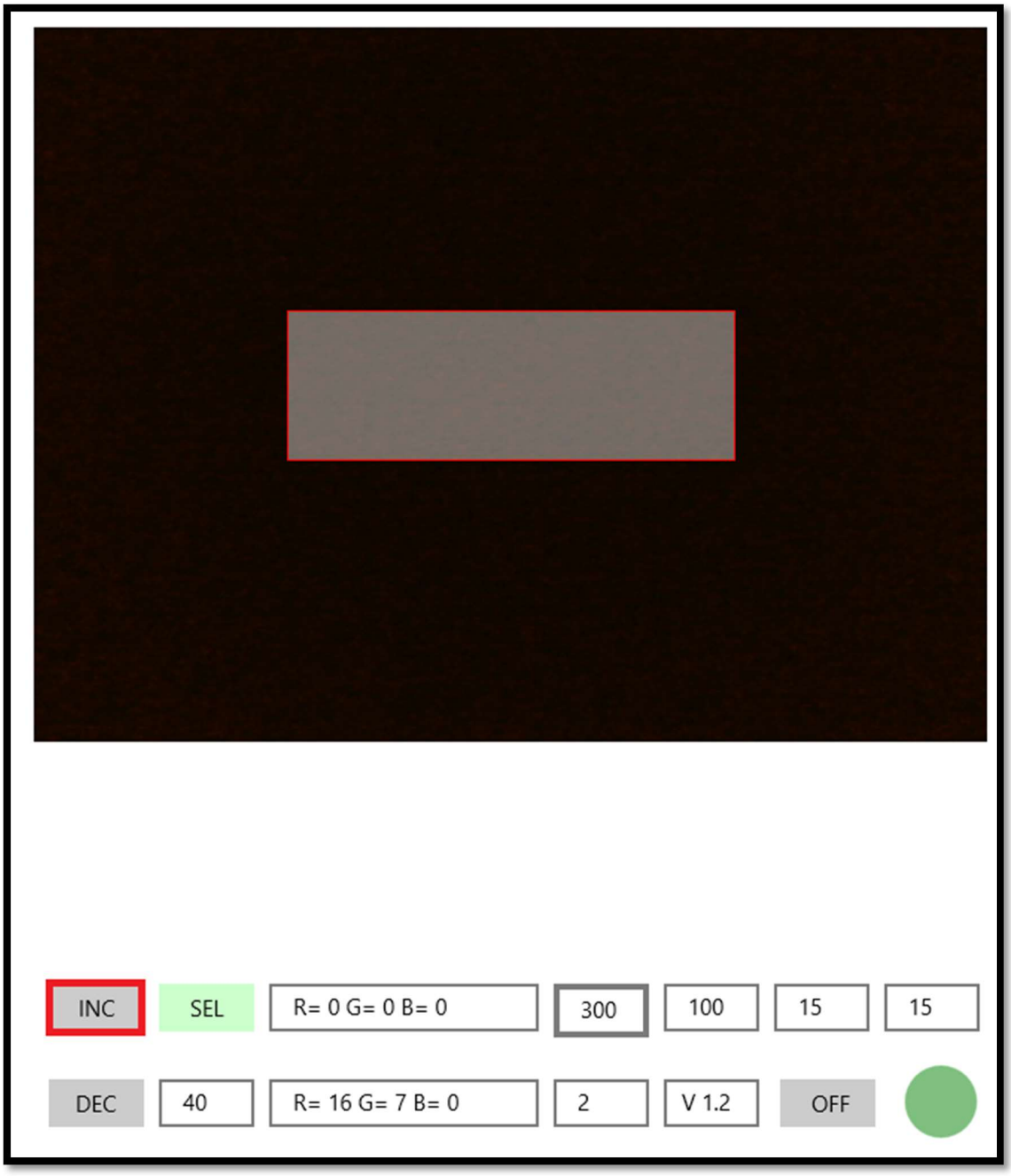
Establecimiento del área de análisis:

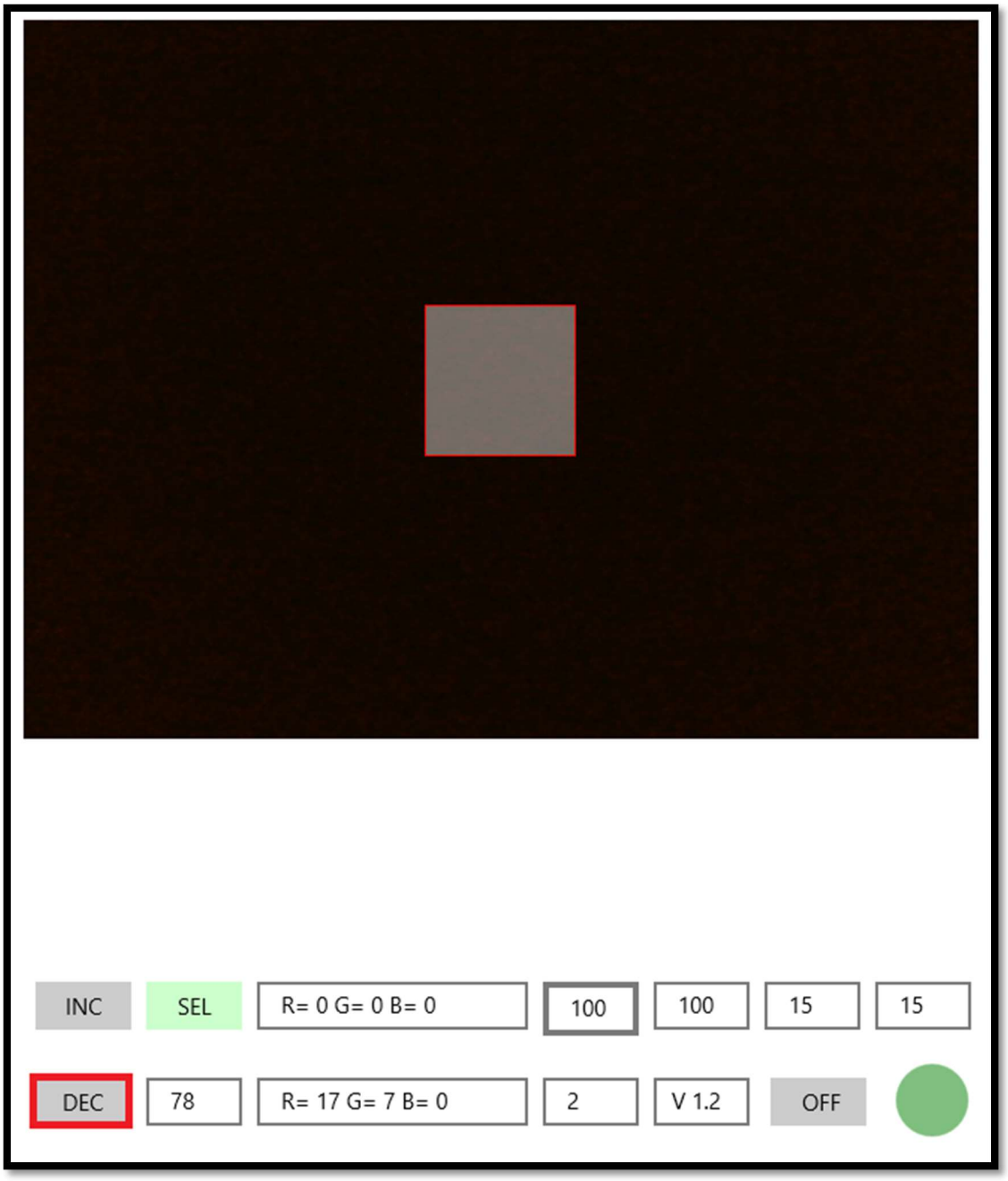
El establecimiento del área de análisis es una de las primeras tareas para configurar la aplicación, para ello pulsaremos SEL hasta que los botones INC DEC se activen y el cuadro de texto de información activo se sitúe sobre el control Width o Heigh, una vez uno de ellos activo, podremos incrementar o decrementar el ancho y alto del área de análisis situada en el centro de la imagen mostrada.



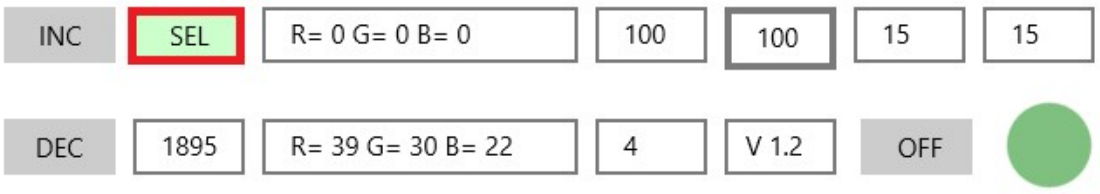
Cuadro de texto de información activo sobre el control Width. Ahora podremos incrementar el ancho del área de análisis pulsando el botón INC y decrementarlo pulsando DEC:

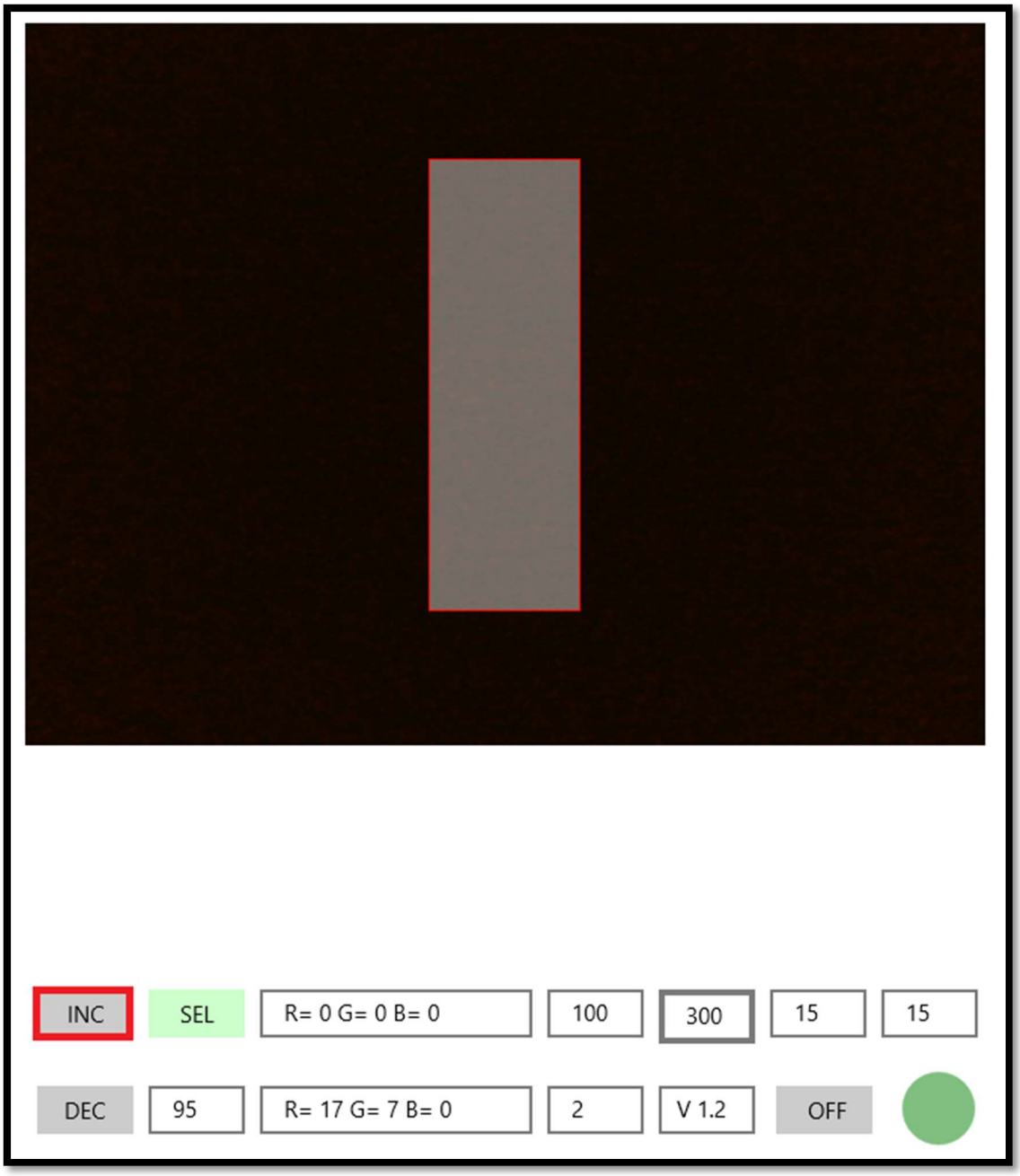


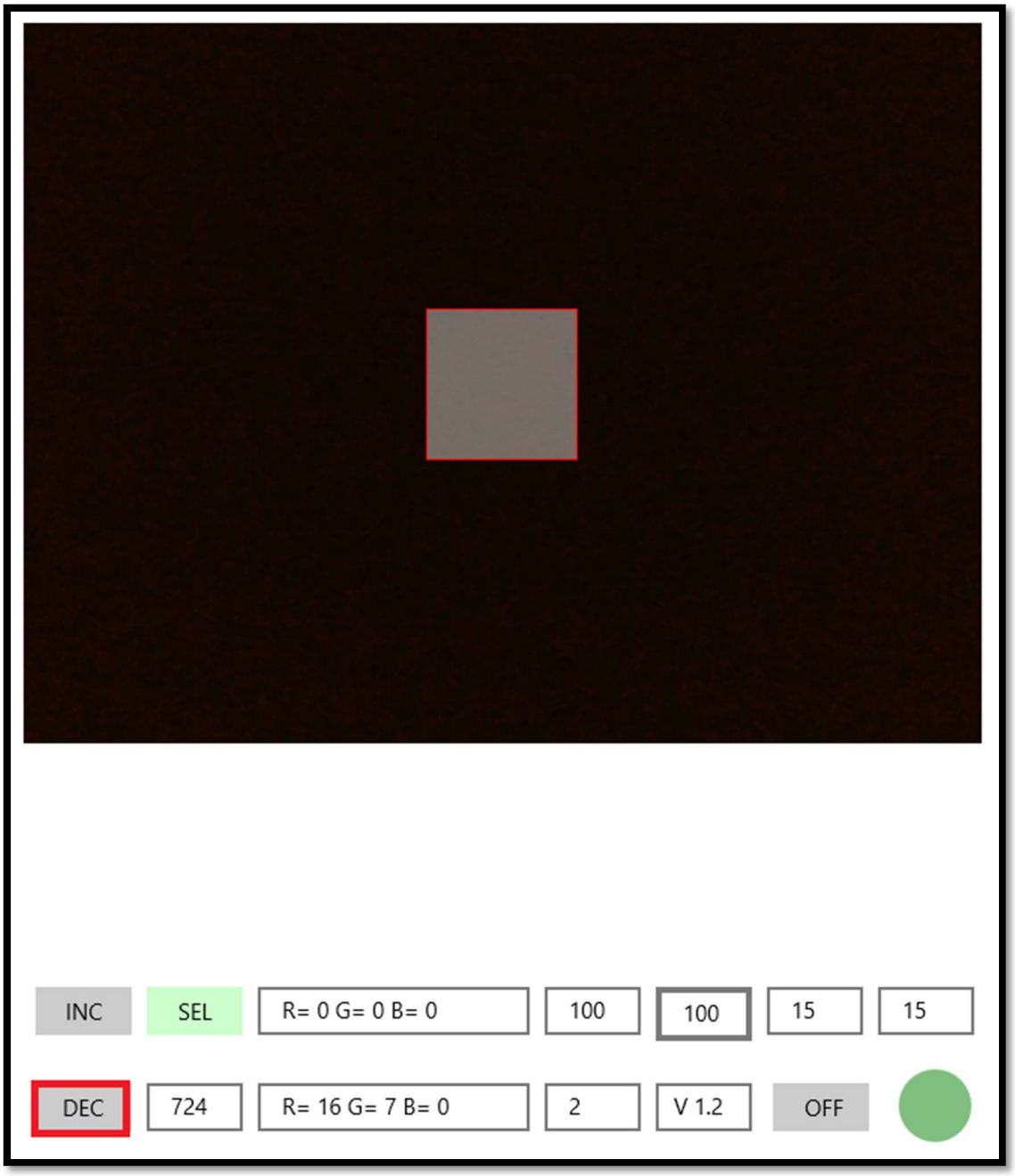

INC
SEL
R= 0 G= 0 B= 0
300
100
15
15
DEC
40
R= 16 G= 7 B= 0
2
V 1.2
OFF



Cuadro de texto de información activo sobre el control Height. Ahora podremos incrementar la altura del área de análisis pulsando el botón INC y decrementarla pulsando DEC:



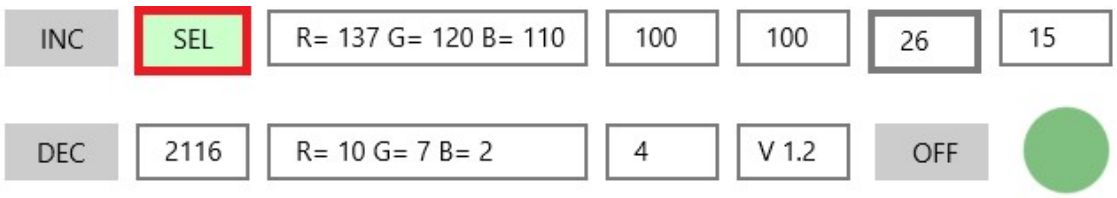

INC
SEL
R= 0 G= 0 B= 0
100
300
15
15
DEC
95
R= 17 G= 7 B= 0
2
V 1.2
OFF



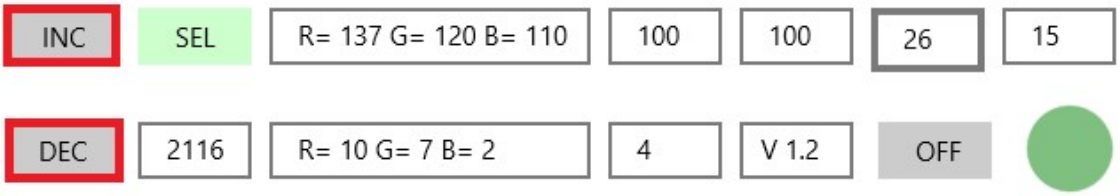
Establecimiento de la precisión del análisis:

Lo que en este control se debe indicar es el porcentaje de desviación sobre la imagen adquirida, o dicho de otro modo, el porcentaje de píxeles distintos, en color medio, con respecto a los de la imagen patrón.

Para establecer la precisión del análisis, pulsamos el botón SEL hasta que el cuadro de texto de información activo se sitúe sobre el control Ave, tal y como se aprecia en la siguiente imagen:



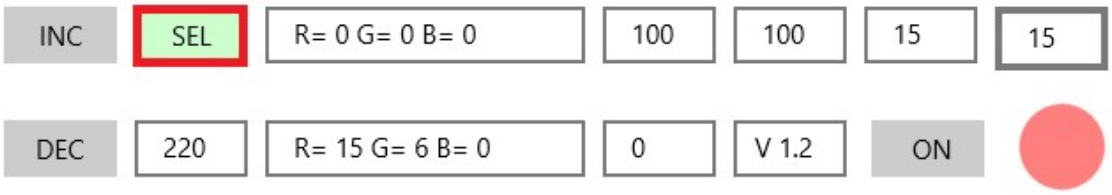
Una vez que el control Ave esté señalado por el cuadro de texto de información activo, podremos aumentar o disminuir su valor pulsando los botones INC o DEC respectivamente. El contador se incrementa de uno en uno.



Definición del filtro de histéresis o tiempo de ON/OFF mínimo:

El tiempo de ON/OFF mínimo (histéresis), es el tiempo que como mínimo deberá funcionar o estar parado el volcador de fruta, una vez que ha pasado a su nuevo estado. Este tiempo viene definido por un filtro que es el que el usuario debe establecer.

Para definir el tiempo, de nuevo pulsaremos SEL hasta que los botones INC DEC se activen y el cuadro de texto de información activo se sitúe sobre el control Filter, como en la imagen siguiente:



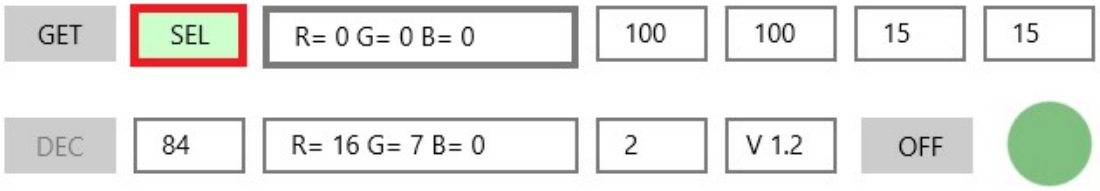
Incrementamos o decrementamos el valor del filtro pulsando los botones INC DEC respectivamente. El contador también se incrementa de uno en uno como ocurre con el de la precisión del análisis. El tiempo mostrado en el control, está expresado en segundos.

La cuenta del tiempo que transcurre desde un cambio de estado se muestra en el control Img C

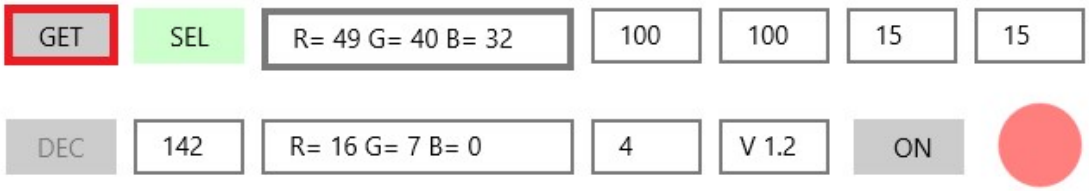
Este control muestra la base de tiempo expresada en segundos/10.

Captura de la imagen base a partir de la cual el análisis compara:

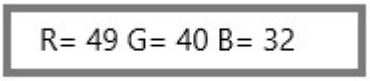
Para capturar la imagen base que el análisis utilizará para comparar con la imagen a analizar, pulsamos SEL hasta que el botón INC cambie por el botón GET .



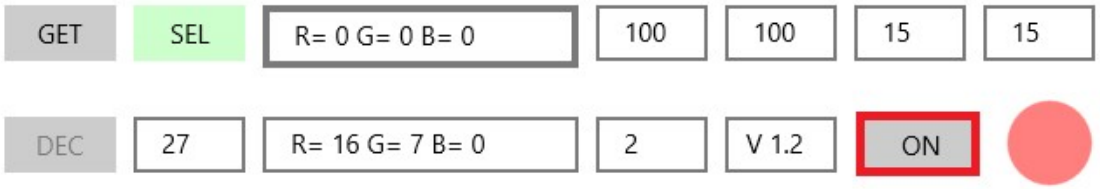
Tras ello, pulsamos el botón GET para capturar como imagen base, la imagen que aparece en el área de análisis.



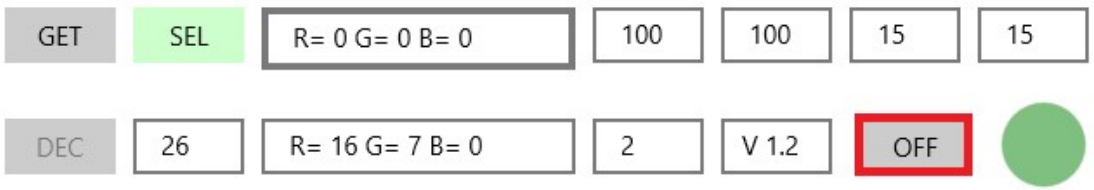
Como podemos observar en la imagen anterior, el UI nos da la información de los valores RGB medios de la imagen base que acabamos de capturar:



Forzar el arranque o paro del volcador de fruta cuando el operario lo desee:
Para forzar al volcador de fruta a arrancar cuando está en paro se debe pulsar el botón:



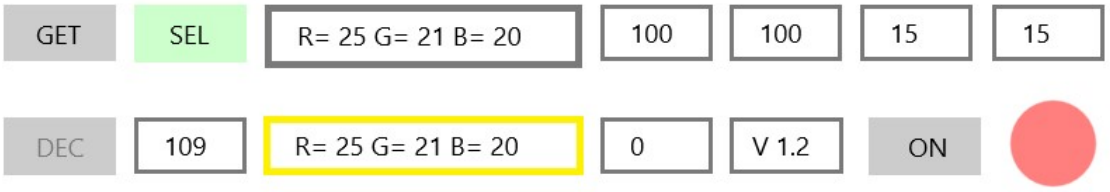
Para forzar al volcador de fruta a parar cuando está en funcionamiento se debe pulsar el botón:



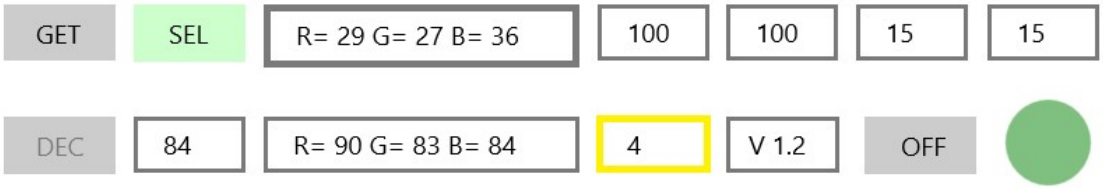
Este botón indica al usuario el estado al que pasará el volcador tras haber sido pulsado.
Tras ser pulsado, siempre se inicia la cuenta del tiempo, mostrándose en el control Img C el tiempo (en segundos/10) transcurrido desde ON o desde OFF:

Al pulsar ON/OFF, el programa sigue de inmediato con su análisis, con lo que si se detectara algún cambio, pasado el tiempo de histéresis, éste adoptará el estado oportuno.

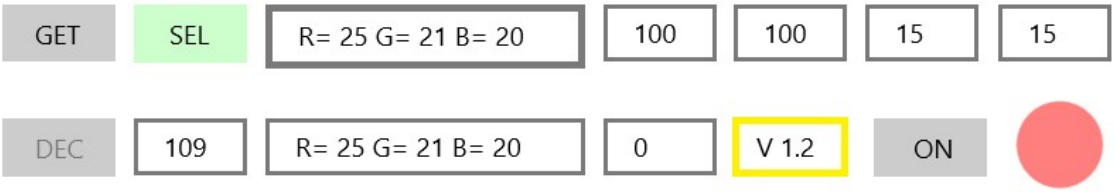
Explicación del resto de controles del UI:
El control remarcado en amarillo, muestra al usuario los valores RGB medios de la imagen en análisis:



Este control (rodeado en amarillo), indica la distancia de color entre la imagen base y la imagen a analizar. Como se ha explicado previamente, la función que se encarga del análisis de imagen, da más peso a la distancia de color en el canal rojo:



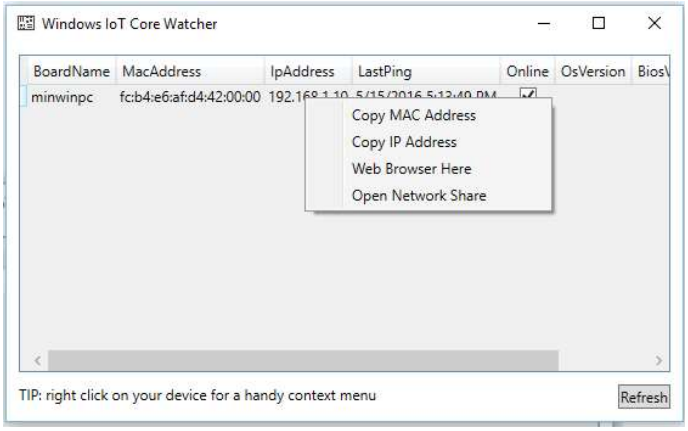
El siguiente control rodeado de amarillo, indica la versión de la aplicación. Se decidió añadir al UI para tener constancia de los cambios que se iban realizando sobre la aplicación y como control, por si en un futuro se realizan actualizaciones de la misma:



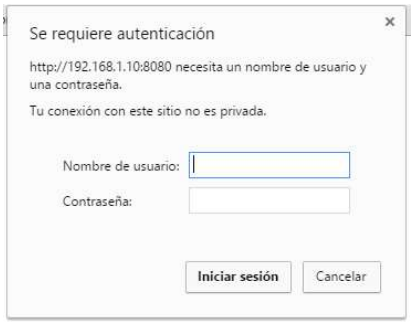
4. MONTAJE

4.1. VOLCADO DE LA APLICACIÓN EN LA RASPBERRY PI 2

Una vez que se ha rodado la aplicación en la Raspberry Pi 2 desde el compilador Visual Studio, ésta mantiene una imagen de la misma en su memoria.
Para volcar el ejecutable deberemos usar el interfaz web de la Raspberry Pi 2. Para ello, desde la herramienta Windows IoT Core Watcher del ordenador, abrimos el interfaz web de la Raspberry Pi 2 pulsando con el botón derecho del ratón sobre nuestro dispositivo Raspberry y seleccionando *Web Browser Here*.

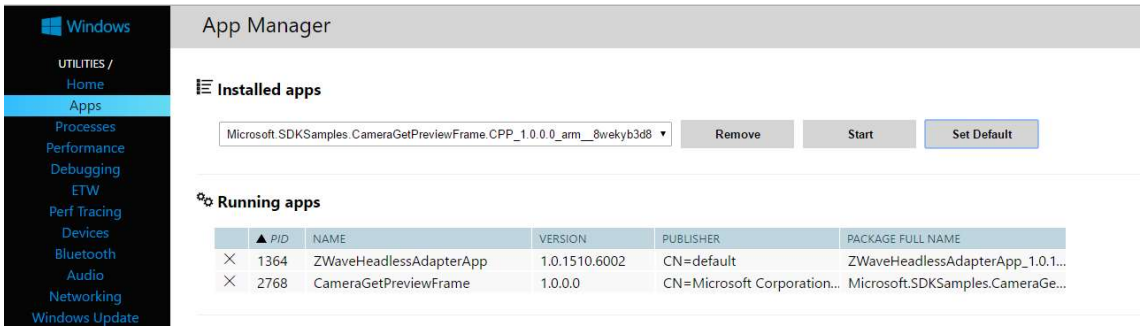


Tras ello, se abre una nueva pestaña en el navegador del ordenador que nos pide el nombre de usuario y la contraseña para poder acceder a nuestra Raspberry.



Una vez introducido nombre de usuario y contraseña, se abre el interfaz de nuestra Raspberry Pi 2. Y una vez en él, vamos a la pestaña Apps y seleccionamos nuestra aplicación en *Installed apps*.
Por último pulsamos *Set Default* y esperamos a que el proceso termine.

A partir de ahora, nuestra Raspberry Pi 2 ya solo contiene la aplicación que se ha desarrollado en este proyecto y ésta se podrá utilizar simplemente poniendo en funcionamiento la Raspberry Pi 2.



4.2. UNIÓN DE TODOS LOS COMPONENTES

Una vez impresa la caja que contiene la solución completa, se procedió al montaje de la misma.

Como ya se explicará en el punto "problemas encontrados", la caja debió de separarse en cada una de sus caras para su impresión, por lo que para comenzar con el montaje, primero se tuvo que unir cada una de las caras, menos la tapa, con un pegamento flexible.

Tras ello, se comenzó a fijar cada componente en su lugar de la caja y se añadieron la junta tórica, el pasamuros y el prensaestopas en sus respectivas localizaciones. Para fijar cada componente a su lugar, se ha utilizado un pegamento flexible.

La tapa, como ya se dijo anteriormente, es la única pieza móvil de la caja, uniéndose al resto con una junta tórica para evitar que entre agua.

Imagen 21: *Caja abierta con componentes en sus situaciones correspondientes y sin cableado.*

5. PRUEBAS DE ESTABILIDAD

Para comprobar la estabilidad de la aplicación desarrollada en este proyecto, se sometió a la misma a la siguiente prueba:

- Se volcó la aplicación en la Raspberry Pi 2 y se dejó rodando en ella durante un periodo aproximado de 72 horas. Durante este periodo de tiempo, se fueron cambiando los parámetros que la aplicación permite al usuario modificar, con el fin de comprobar que todas las funciones de la misma seguían funcionando de forma correcta.

Durante la prueba pudimos observar que la aplicación funcionaba de forma correcta durante todo el tiempo y bajo distintas condiciones de luz, cumpliendo con los objetivos de análisis de imagen y señalización para los que había sido diseñada. También se pudo comprobar que la Raspberry Pi 2 no se calentaba de forma excesiva, tan solo desprendía un ligero calor.

Aunque se podría haber separado los módulos de código principales y ser montados en receptores de código para test de caja negra, dado el pequeño tamaño de la aplicación, no se ha considerado necesario.

6. POSIBLES MEJORAS.

Aunque se ha pretendido en todo momento generar una aplicación y su expresión como producto lo más aproximado al mercado, ésta no deja de ser un prototipo precompetitivo, no cumpliendo pues con las reglas de marcado CЄ. Queda para la fase de industrialización del prototipo precompetitivo, el rediseño mecánico de la caja, junto con la selección del material de la misma y la selección de algunos de los componentes electrónicos como la pantalla e incluso la cámara, de forma que se cumpla con la normativa del marcado CЄ.

Por otro lado, tal y como ya se ha mencionado previamente, los parámetros de configuración de la aplicación no se guardan en la memoria SD de la tarjeta Raspberry. Una posible mejora sería escribir la función encargada de almacenar todo el estado de los parámetros que maneja la aplicación cada vez que uno de ellos cambia.
También se podría elegir como lugar de almacenamiento la nube en lugar de la tarjeta SD, para ello podríamos usar los numerosos ejemplos de uso de la plataforma Azure. Bastaría con añadir alguna de las funciones base que IoT tiene para este propósito, usando para acceder a Azure (Internet), el pin Wifi que tiene conectado la Raspberry Pi 2.

Otra mejora podría ser la detección automática mediante visión artificial de los bordes del elevador de fruta, esta función, aunque intensiva en carga de CPU, solo se ejecuta al arranque de la aplicación, por lo que esta excesiva carga para la Raspberry Pi 2 no es un problema.

7. PROBLEMAS ENCONTRADOS

Tal y como se ha mencionado, uno de los problemas que surgieron, fue a la hora de imprimir, con una impresora 3D, la caja que contiene a la solución completa.

La caja diseñada en este proyecto constaba en un principio de dos partes, una era la tapa de la misma, en la que iba acoplada la pantalla y la otra parte era el resto de la caja, tal y como puede apreciarse en la siguiente imagen.

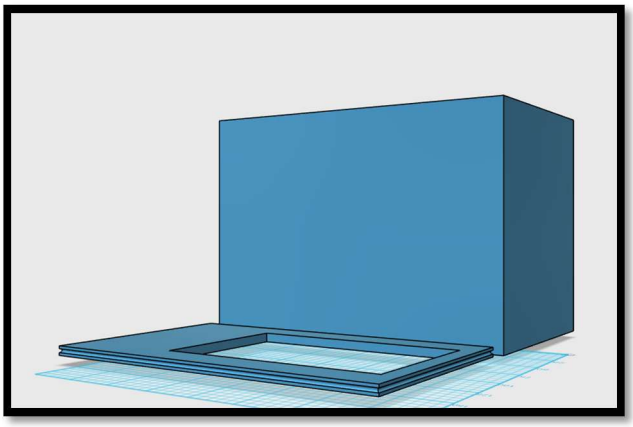


Imagen 22: *Vista de las dos partes, tapa y cuerpo, que componían la caja en un inicio.*

El programa 123D Design de Autodesk, permite enviar el objeto diseñado a cualquiera de estas cuatro compañías para su impresión 3D:

Imagen 23: *Estas son las cuatro compañías que Autodesk 123D recomienda para la impresión 3D.*

Una vez finalizado el diseño de nuestra caja, se solicitó por web, a una de las compañías anteriores, cotización del coste de impresión de la misma. El precio aproximado de impresión rondaba los 700 euros, más que el coste total de todos los componentes de la solución juntos. Por este motivo, se decidió reducir, en la medida de lo posible, el tamaño de la caja, con el propósito de reducir la cantidad de material a utilizar en la impresión de la misma.

Cuando la caja estuvo optimizada al máximo, en lo que a cantidad de material se refiere y siempre respetando las medidas mínimas para que los componentes no se sobrecalentasen, ni los cables de conexión se doblaran de forma excesiva, se intentó imprimir con la impresora 3D que la Escuela Politécnica Superior de Alcoy posee.
Los profesores de la EPSA Don Jaime Masiá y Don Juan Ramón Rufino, tutor de este proyecto, prestaron su ayuda para tal propósito. Fueron ellos los que detectaron que los cilindros que servían de soporte a la Raspberry Pi 2, iban a quedar deformados al ser impresos, debido a que están colocados en una pared vertical y no poseen ningún soporte sobre el que pueda descansar el material que va siendo impreso.
Otro problema que surgió, fue que aun habiéndose reducido el tamaño de la caja, ésta seguía siendo demasiado grande y su impresión duraba aproximadamente 27 horas.
Don Jaime Masiá aconsejó que se dividiera la caja en cada una de sus caras para la impresión y que tras ella se uniesen, también se debía intentar reducir el grosor de las paredes para reducir material.
De esta forma se evitaba que en la impresión de los soportes para la Raspberry Pi 2, éstos se deformasen, puesto que al estar todas las caras de la caja por separado, se podía imprimir cada

una de ellas en horizontal, con lo que en la impresión, siempre habría material sobre el que el nuevo podría depositarse.
Al imprimirse cada pieza por separado y habiéndose reducido su grosor, la cantidad de tiempo y material consumidos en la realización de la caja se verían disminuidos.

Es por todo lo expuesto anteriormente que la caja finalmente quedara para su impresión, de la siguiente forma:

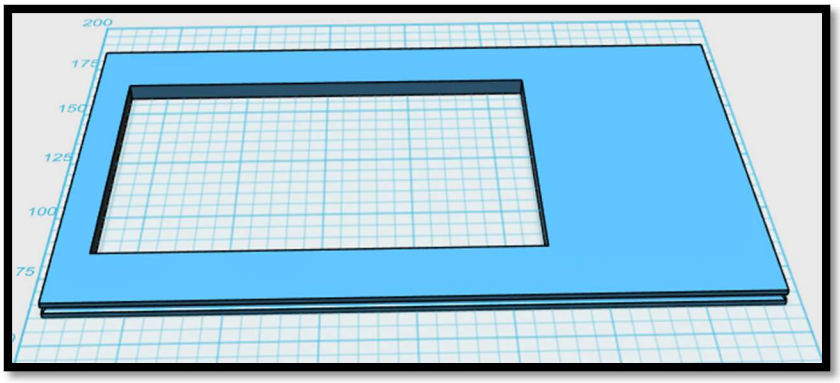


Imagen 24: *Tapa de la caja donde va situada la pantalla.*

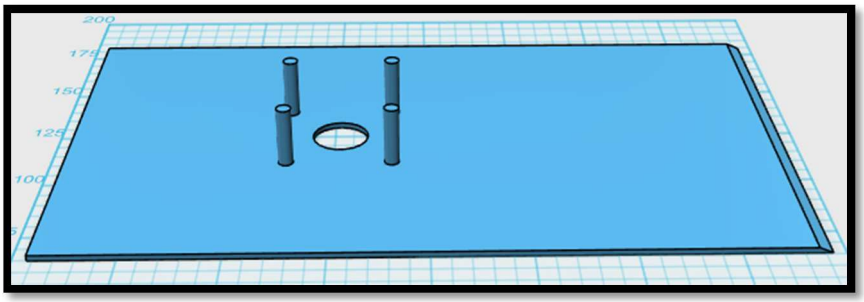


Imagen 25: *Parte inferior de la caja donde va situada la cámara.*

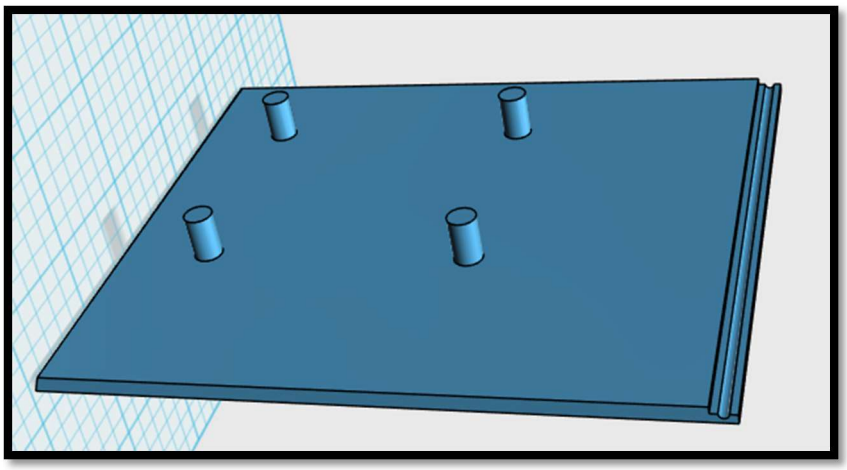


Imagen 26: *Lateral de la caja donde va situada la Raspberry Pi 2.*

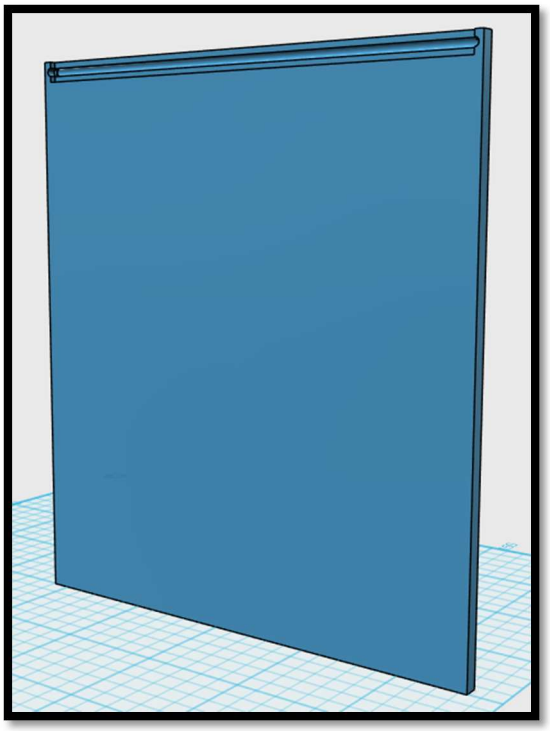


Imagen 27: *Lateral de la caja opuesto al de la Raspberry Pi 2.*

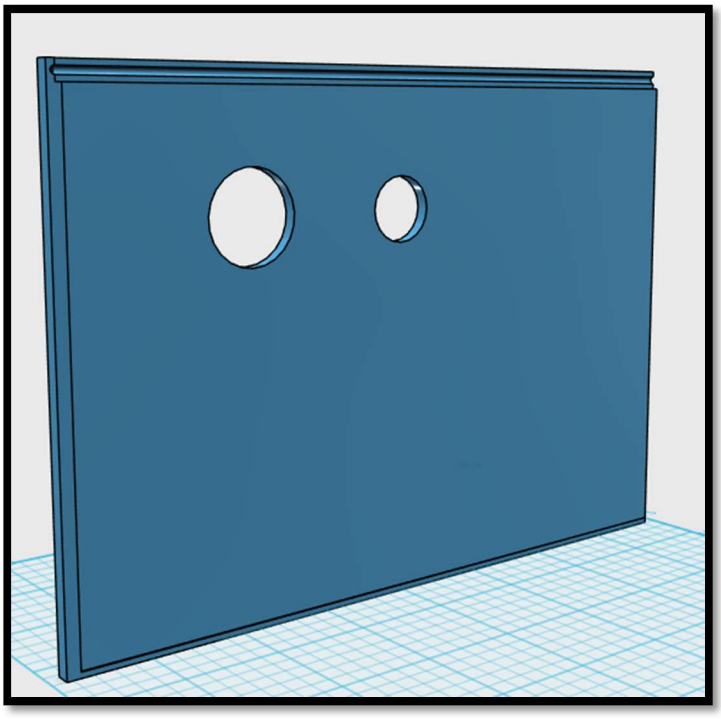


Imagen 28: *Lateral de la caja por el que salen los cables de alimentación y de control del montaje.*

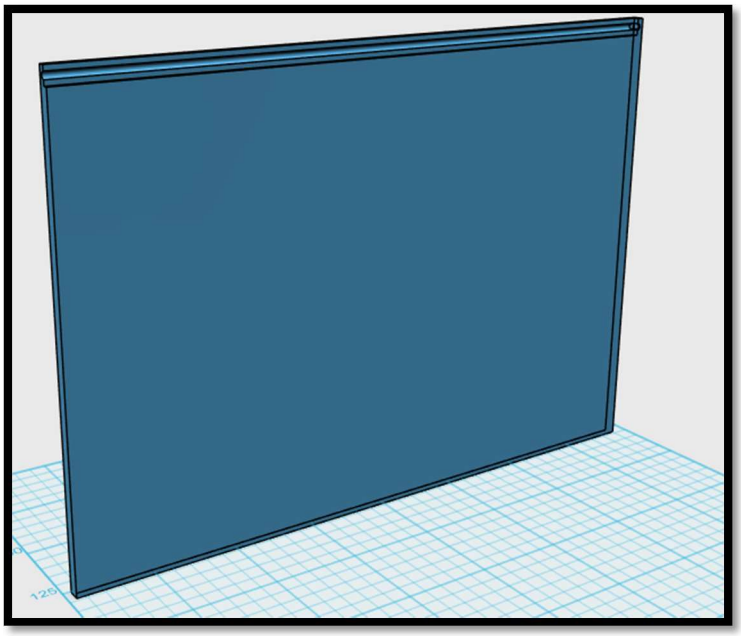


Imagen 29: *Lateral de la caja opuesto al lateral por el que salen los cables de alimentación y de control del montaje.*

Otro de los problemas con los que se tuvo que lidiar ha sido el mal funcionamiento de la cámara seleccionada para la solución.

Ésta en un principio funcionaba cuando se probó con la Raspberry Pi 2, pero de repente, comenzó a fallar. Se comprobó que la cámara no estuviera rota mediante una prueba de la misma con el ordenador, resultando que la cámara funcionaba perfectamente, por lo que se dedujo que el problema lo tenía la cámara al hacerla funcionar con el Sistema Operativo Windows IoT. Este problema surge de la no existencia, a fecha de Mayo de 2016, de *drivers* con los que la cámara pueda ir de forma correcta con este Sistema Operativo.

La solución a este problema, cuyo remedio definitivo vendrá con la escritura futura de esos *drivers,* ha sido la utilización de una cámara web para la defensa de este trabajo fin de grado.

8. CONCLUSIONES

En lo tocante a lo personal, la realización de este proyecto me ha llevado a mejorar mi forma de programación en lenguaje C++. Aprendí a programar con este lenguaje en primero del Grado en Tecnologías Industriales, y durante el Grado en Ingeniería Eléctrica, también se impartió alguna asignatura en la que se programaba con el mencionado lenguaje. Pero sin lugar a dudas, el reto planteado en este proyecto, en cuanto a escritura de código en lenguaje C++ se refiere, y más si cabe en una aplicación basada en visión artificial, me ha hecho evolucionar en este campo y gracias a él, ahora poseo muchas más herramientas con las que defenderme en el mundo de la programación.

A parte de las herramientas, a nivel de programación, que he ido adquiriendo en el desarrollo de este proyecto, también he aprendido a utilizar una herramienta de desarrollo de aplicaciones con la que no había trabajado antes, el Visual Studio. Así como el programa de diseño de Autodesk, 123D Design, con el que se pueden diseñar objetos para posteriormente imprimirlos en 3D.

Cuando comencé con este proyecto, no lo hice con la intención de generar un producto que se fuera a comercializar ya, pero sí enfocado hacia una posible futura comercialización con alguna empresa de ingeniería existente. Es por ello que buscara, no solo el desarrollo de la aplicación que cubre la necesidad a la que se ha enfocado este proyecto, sino, que he buscado el ofrecer una solución completa a esa necesidad. Esta solución completa incluye, aparte de la aplicación, el diseño e impresión del prototipo de la caja que la contiene, así como la búsqueda y adquisición de los componentes que la conforman. Aunque, como ya he mencionado antes, el proyecto no se ha desarrollado para su inmediata comercialización y por tanto, se ha procedido basándose en prototipos y en ideas que deberán modificarse para cumplir con el marcado CЄ si en el futuro se decidiera poner a disposición de alguna empresa.
La redacción del presente proyecto también está orientada al ámbito de la empresa, en el que los proyectos tienden a ser concisos, lo que no quiere decir que contengan menos información.

Resumiendo, puedo afirmar que la realización del presente proyecto, me ha dotado de nuevos y útiles conocimientos en diferentes disciplinas tales como, diseño de objetos para su posterior impresión 3D, programación en lenguaje C++ en un entorno de desarrollo multiplataforma como es el Visual Studio y búsqueda de una solución para una problemática real en el ámbito de la empresa.

Puedo decir, sin miedo a equivocarme, que la realización de éste proyecto ha sido una de las tareas más divertida y enriquecedora que he realizado en mucho tiempo.

9. BIBLIOGRAFÍA

- Página de información general de los productos de Visual Studio 2015: https://www.visualstudio.com/vs-2015-product-editions
- Página de Adafruit con información de la pantalla: https://www.adafruit.com/products/2260
- Información sobre la Raspberry Pi 2:

  https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

  https://www.element14.com/community/docs/DOC-73827/l/raspberry-pi-2-model-b-1gb-technical-specifications#techspecs

  https://www.raspberrypi.org/wp-content/uploads/2014/07/mechanicalspecB+.png

  http://wordnice.net/info/raspberry-pi-2-model-b-caracteristicas-t1044.html

- Nuevas prestaciones de C++ (compilación 11).

  http://arco.esi.uclm.es/~david.villa/pensar_en_C++/vol1/ch02s03s02.html

- Tutorial puesta a punto PC y Raspberry Pi para desarrollar con Windows IoT:

  https://developer.microsoft.com/en-us/windows/iot/win10/kitsetuppcrpi

- Información acerca de la herramienta 123D Design de Autodesk:

  http://www.123dapp.com/3d-printing-services