UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Ingeniería en Electrónica y Telecomunicaciones

Implementación de una estación prototipo con visión artificial, aplicado a la agricultura de precisión

Trabajo de titulación previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones

Autor:

Pablo Esteban Villota Neira

CI: 0401585401

Director:

Ing. Santiago Renán González Martínez, PhD

CI: 0103895934

Cuenca-Ecuador

14/10/2019

Resumen:

La agricultura de precisión es una actividad que mediante la observación de variables ambientales permite actuar de forma precisa y oportuna sobre las parcelas agrícolas para así lograr una mayor y más eficiente producción. En Ecuador, esta es una actividad poco explotada por pequeños y medianos productores, tanto por falta de conocimiento de las tecnologías que pueden utilizarse para este propósito, como por falta de los recursos económicos necesarios para llevarla a cabo.

Si bien existen varias tecnologías que están ayudando al despliegue de la agricultura de precisión, en el presente trabajo de tesis se hará énfasis en tres de ellas. En primer lugar, se utilizan técnicas de procesamiento de imágenes para la detección de frutos de una parcela. Posteriormente se emplean dos tecnologías que llevarán este proyecto a un ámbito actual de investigación, como es el desarrollo de aplicaciones basada en el *Internet of Things* (IoT). Específicamente, computación en la nube para la gestión de la información generada por sensores y el protocolo *Message Queue Telemetry Transport* (MQTT) para la mensajería de los datos.

Es por esto que se propone el diseño y construcción de una estación prototipo, con capacidad para monitorizar variables ambientales, desarrollar tareas de video-vigilancia así como capturar y procesar imágenes para la detección de frutos. Esta estación enviará los datos recolectados a través de Internet a una aplicación web alojada en la nube de IBM; finalmente los resultados podrán ser monitorizados por un usuario a través del navegador.

Palabras claves: Agricultura de precisión. Internet of Things. Thresholding. Modelos de Mezclas Gaussianas. Node-Red. IBM cloud, MQTT.

Abstract:

Precision agriculture is an activity that, through the observation of environmental variables, allows precise and timely action on agricultural plots in order to achieve greater and more efficient production. In Ecuador, this is an activity little exploited by small and medium producers, both for lack of knowledge of the technologies that can be used for this purpose and for lack of the necessary economic resources to carry it out.

Although there are several technologies that are helping to deploy precision agriculture, in the present thesis work, emphasis will be placed on three of them. First, image processing techniques are used for the detection of fruits of a plot. Subsequently, two technologies are used that will take this project to a current field of research, such as the development of applications based on the Internet of Things (IoT). Specifically, cloud computing for the management of information generated by sensors and the Message Queue Telemetry Transport (MQTT) protocol for data messaging.

Thus, the design and construction of a prototype station is proposed, with the ability to monitor environmental variables, develop video surveillance tasks as well as capture and process images for the detection of fruits. This station will send the data collected through the Internet to a web application hosted in the IBM Cloud; finally, the results can be monitored by a user through the browser.

Keywords: Precision agriculture. Internet of Things. Thresholding. Gaussian Mixture Models. Node-Red. IBM Cloud, MQTT.

Índice del Trabajo

Cláusula de Propiedad Intelectual

Pablo Esteban Villota Neira, autor del trabajo de titulación "Implementación de una estación prototipo con visión artificial, aplicado a la agricultura de precisión", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 03 de octubre de 2019

Pablo Esteban Villota Neira

C.I: 0401585401

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Pablo Esteban Villota Neira en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Implementación de una estación prototipo con visión artificial, aplicado a la agricultura de precisión", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 03 de octubre de 2019

Pablo Esteban Villota Neira

C.I: 0401585401

Agradecimientos

A mis padres, por estar pendientes de mis progresos como profesional y como persona.

A mi hermano, cuya curiosidad me ha ayudado a siempre preguntarme el porqué de las cosas.

A mi tutor Santiago, por su paciencia y disponibilidad siempre que necesité algún consejo.

A mis amigos, cuyas palabras a veces difíciles de digerir pero siempre bien intencionadas, me han ayudado a reflexionar las decisiones de mi vida.

A mis compañeros, hayan sido o no afines a mí, todos me ayudaron a conocer las dificultades y ventajas del trabajo en equipo.

A todos mis maestros que tuvieron la vocación de la docencia y pasión por la profesión, porque ellos fueron la motivación que muchas veces necesité.

A Dios, el universo o aquel sentimiento superior que hace que todo cobre sentido.

I. Índice General

II. Índice de Figuras

III. Índice de Tablas

IV. Listado de Acrónimos

ANN	Artificial Neural Network
API	Application Programming Interface
BGR	Blue-Green-Red
BIC	Bayesian Information Criterion
CCL	Connected-Components Labeling
CSS	Cascading Style Sheets
EM	Expectation-Maximization
FAO	Food and Agriculture Organization
FPS	Fotogramas por segundo
GMM	Gaussian Mixture Models
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HLS	Hue-Saturation-Lightness
HSV	Hue-Saturation-Value
HTML	Hyper Text Markup Language
I2C	Inter Integrated Circuits
IoT	Internet of Things
LAB	Lightness, color channel a, color channel b
M2M	Machine to Machine
MQTT	Message Queue Telemetry Transport
PaaS	Platform as a service
QoS	Quality of Service
RGB	Red-Green-Blue
SBC	Single Board Computer
SSH	Secure Shell
SVM	Support Vector Machine
TLS	Transport Layer Security
UAV	Unmanned Aerial Vehicle
UV	Ultravioleta

1. INTRODUCCIÓN Y OBJETIVOS DE LA TESIS

En este capítulo se presenta una introducción al trabajo de tesis, en particular se describe la problemática de la limitada aplicación tecnología en el sector agrícola de nuestro país, a partir de lo cual se propone un dispositivo tecnológico de bajo coste el cual contribuirá en los procesos de análisis de los cultivos. Además, se describe el alcance y los objetivos del presente trabajo de tesis

1.1. Definición del Problema

En Ecuador, la agricultura representó el 7% del producto interno bruto del año 2017, siendo además la actividad que produce más empleos en el país, como se resaltan en [1],[2]. Sin embargo, pese a los grandes avances de la tecnología, la producción de los campos en Ecuador es considerablemente baja. Para citar un ejemplo, según la base de datos de la Organización de Alimentos y Agricultura de las Naciones Unidas (FAOSTAT, 2016), [3], la producción de tomates en Ecuador en 2016 fue de 31,4 ton/hec (tonelada/hectárea) que está muy por debajo de países desarrollados como es el caso de Estados Unidos, donde se producen 90,2 ton/hec e incluso es inferior a la media mundial (37 ton/hec).

Las causas de esta baja producción son varias, como por ejemplo, el limitado acceso a fertilizantes o el bajo despliegue de sistemas de riego. Sin embargo, uno de los mayores inconvenientes es la limitada aplicación de tecnología en los cultivos[4]. Ciertamente en Ecuador es reducida la aplicación de tecnologías que permiten optimizar y mejorar los procesos agrícolas, especialmente en el ámbito de la pequeña producción. Cabe resaltar, que en nuestro país 3 de cada 4 productores tienen superficies de producción inferiores a 5 hec [5]. Bajo tal condición, los pequeños productores se encuentran en desventaja y resulta complejo asumir los costes de la tecnificación en los cultivos.

Sin ir muy lejos, la situación es muy distinta en otros países de la región, por ejemplo, en Argentina existen varias empresas dedicadas a la implementación de tecnología aplicada a la agricultura. Específicamente, se han construido estaciones de monitoreo que permiten hacer estudios de la productividad de una parcela, mediante el uso de sensores y actuadores, plataformas SIG (Sistema de Información Geográfica), dispositivos GPS (*Global Positioning System*) así como *software* para la monitorización remota de cultivos [6].

La tecnificación de los cultivos con la ayuda de tecnologías emergentes recibe el nombre de *agricultura de precisión*. Actualmente, en la industria agrícola, se emplean robots tanto para la recopilación de datos para elementos actuadores, cámaras fotográficas como sensores de imagen, además de diversas estructuras de redes inalámbricas con la finalidad de conectar sensores y actuadores y de brindar conectividad hacia Internet. Este último punto es muy importante ya que permite integrar la agricultura de precisión junto con soluciones tipo IoT (*Internet of Things*). En particular, las arquitecturas IoT son cada vez más empleadas en diversas áreas como la industria, la domótica e incluso con fines de entretenimiento. En cuanto a la agricultura de precisión, las soluciones IoT permiten monitorizar ciertos parámetros de interés en los cultivos como humedad, temperatura, imágenes de los cultivos, gases, entre

otros. De ésta forma es posible realizar tareas de control y vigilancia remota, planificación y previsión de cosechas, comunicación M2M (Machine to Machine) para automatización e incluso análisis de big data [7].

Bajo tales condiciones, resulta evidente la necesidad de mejorar la tecnificación agrícola en nuestro país, priorizando a los pequeños productores, con soluciones de bajo costo que permitan incrementar la producción de cultivos por hectárea.

1.2. Justificación y Alcance

De acuerdo a la FAO (*Food and Agriculture Organization*), la producción agrícola deberá crecer en un 70% para el 2050 para poder alimentar a una población mundial de ocho mil millones de personas que se estima habrá para ese entonces. Para alcanzar tal objetivo considerando problemas tales como el cambio climático y el limitado acceso al agua, es imprescindible que todos los sectores agrícolas y ganaderos estén equipados con tecnologías digitales [7]. Este organismo también resalta, que la inversión en el sector agrícola es la forma más efectiva de reducir el hambre y la pobreza y proveer sostenibilidad en un país [8]. En tal sentido, se prevé que gran parte de la industria agrícola empleará soluciones basadas en tecnologías IoT[8] y redes de sensores[9]. Ante tal escenario, nuestro país no puede ser la excepción. Las ventajas y posibilidades que ofrece la tecnología aplicada a los cultivos, ha sido una de las principales motivaciones para el planteamiento y desarrollo del presente trabajo de tesis.

En particular, una tecnología que está siendo cada vez más utilizada para el análisis remoto de cultivos es la visión artificial. Las aplicaciones del procesamiento de imágenes en agricultura son numerosas, tal como, la identificación de los efectos de los insectos en los cultivos [10], la medición del estrés de las plantas según la temperatura del suelo [11], tareas de video vigilancia [12] así como la identificación del estado de los cultivos [13], entre otros. Esta tecnología ofrece mucha flexibilidad en sus aplicaciones, y dependiendo del *hardware* usado, un mismo equipo puede cubrir áreas extensas.

Con base en esto, la propuesta de este trabajo de tesis se enfoca en el desarrollo de una solución aplicada a la agricultura de precisión, la misma que integra tecnologías tipo IoT junto con mecanismos de visión artificial. Puntualmente, se ha planteado crear una estación prototipo que realice operaciones de visión artificial tales como extracción de características de color y tamaño de los frutos, parámetros que a futuro pueden emplearse para determinar la madurez o salud de un cultivo [14],[15]. Además, se contempla la capacidad de video vigilancia, posibilitando así una mejor planificación de los cultivos.

Por otra parte, la estación contará con un módulo GPS que podrá ser usado para ubicar la estación, por ejemplo, en un escenario conformado por un conjunto muy grande de estaciones de monitoreo, dotando de escalabilidad al proyecto. La cámara estará conectada a una plataforma de desarrollo tipo SBC (*Single Board Computer*) para tener la capacidad de realizar el procesamiento de la imagen. El GPS estará conectado al SBC mediante sus puertos de entrada digitales. En cuanto, a la gestión de la información capturada, se desarrollará una aplicación web, que permita visualizar en tiempo real los datos obtenidos de los sensores

disponibles en la estación. El prototipo contará con una interfaz de comunicación para su conexión a internet, actuando como *gateway* para el envío de los datos de los sensores.

Para mejorar la autonomía energética del prototipo, se realizarán pruebas para caracterizar el consumo eléctrico de las diferentes tareas que realizará la estación y así estimar el tiempo de autonomía con la que esta contará.

Siendo el factor económico un limitante para el acceso a la tecnología en los campos agrícolas del Ecuador, la elaboración de este prototipo tomará como base *hardware abierto* junto soluciones en *software* libre con el objetivo de reducir su costo al mínimo posible. Concretamente se usarán tarjetas *SBC Raspberry Pi* como hardware, y la librería Open CV de Python para el procesamiento de las imágenes como base de software. Finalmente, el prototipo contará con una batería, lo que permitirá su portabilidad y fácil instalación.

1.3. Objetivo General

Diseñar y desarrollar una estación prototipo que use visión artificial enfocada a la agricultura de precisión

1.4. Objetivos Específicos

1. Implementar en la estación prototipo un módulo GPS que permita la ubicación de la parcela a analizar.

2. Caracterizar el consumo de energía del prototipo implementado.

3. Implementar un sistema de visión artificial para la identificación del tamaño y color de los frutos.

4. Configurar interfaces de comunicación que permitan el acceso del prototipo a Internet, así como su integración futura a una red de sensores inalámbrica

5. Desarrollar una aplicación web que permita la gestión y monitorización remota de la estación prototipo

1.5. Estructura del Documento

El documento está estructurado de la siguiente forma. En el Capítulo 2 se presenta el marco teórico; en particular se resaltan conceptos fundamentales acerca de la agricultura de precisión, Internet of Things, redes de sensores, así como conceptos básicos del procesamiento de imágenes y video; finalmente, se describen brevemente las herramientas empleadas para el desarrollo web. A continuación, en el Capítulo 3, se presentan los principales trabajos relacionados, disponibles en la literatura. Posteriormente, en el Capítulo 4, se detalla la implementación de la estación prototipo. Las pruebas realizadas en campo y los resultados obtenidos, se describen en el Capítulo 5. Finalmente, en el Capítulo 6, se exponen las conclusiones y recomendaciones generales del presente trabajo de tesis.

2. MARCO TEÓRICO

2.1. Introducción

El trabajo de tesis propuesto tiene como propósito el diseño e implementación de una estación prototipo para la monitorización de variables de interés en el ámbito de la agricultura de precisión. Con tal finalidad, en este capítulo se exponen los conceptos fundamentales para entender el trabajo en su conjunto, así como una introducción a las tecnologías empleadas.

2.2. Agricultura de Precisión

La agricultura de precisión es el término que se le da a la administración de lotes agrarios en los que se ejecutan tareas de medición y actuación sobre los cultivos, utilizando tecnología moderna para desarrollar un plan de acción. La agricultura de precisión tiene principalmente tres objetivos, incrementar la producción de las plantas, reducir el impacto ambiental y obtener mayores ingresos económicos en las prácticas agrícolas.

Para la captura de datos y toma de medidas, las tareas habituales consisten en levantamientos topográficos, emplazamiento de sensores y toma de imágenes fotográficas con cámaras fijas, móviles o satelitales. Mientras que las tareas destinadas a los actuadores se enfocan principalmente en la fertilización, fumigación, cosecha y riego.

Las tecnologías usadas para llevar a cabo estas tareas son muy variadas, desde infraestructuras complejas para los sistemas de riego, redes de sensores, sistemas robóticos para fertilización y fumigación, sistemas de visión e inteligencia artificial [16], análisis de datos, entre otros. Además, puesto que cada parcela presenta condiciones distintas, los sistemas de posicionamiento son de gran importancia para identificar el lugar exacto en el cual se desarrolla un evento. En esta área está muy difundido el uso de sistemas GPS (*Global Positioning System*) y GNSS (*Global Navigation Satellite System)* con los que es posible generar mapas de variabilidad espacial y actuar sobre los cultivos con propósitos varios.

En el Ecuador, la agricultura de precisión se encuentra principalmente en los sistemas automatizados de riego. Por otra parte, existen algunos proyectos en etapas iniciales que emplean imágenes satelitales y ortofotos captadas por drones, que permiten determinar niveles de clorofila, humedad y otros factores en las plantas. Sin embargo, debido a su alto costo, tal sistema está limitado a los productos de alta exportación y consumo como por ejemplo, el banano, el cacao o la cebada, mientras que para los pequeños productores resulta casi imposible entrar en la era de la denominada agricultura 3.0 [17].

2.3. Tecnología IoT

El término *Internet of Things* (IoT), apareció en los años 90 para referirse a la conexión de objetos cotidianos hacia Internet. Si bien en esa época eran muy limitados los objetos conectados a Internet, con el paso de los años y la reducción en costo y tamaño de los microprocesadores, este número se ha incrementado exponencialmente. Como un ejemplo en el año 2010 existían más objetos conectados que personas en el mundo [18] y según los

pronósticos del portal Forbes [19] para 2025 la cifra de dispositivos conectados a Internet será de 75 mil millones.

Los factores que han favorecido este enorme crecimiento han sido principalmente la ubicuidad de Internet, la caída en los precios de miniordenadores, la estandarización de los protocolos de comunicaciones, los avances en los análisis de datos y el crecimiento de la computación en la nube [20].

El impacto de esta tecnología ha sido tal que no solo ha cambiado nuestra relación con los objetos, sino también ha motivado el desarrollo de nuevas arquitecturas de comunicación. Un claro ejemplo son las conexiones M2M (*Machine to Machine*) que pasaron de 106 millones en 2012 a 360 millones en 2018 [21]. Además, existen otros modelos de comunicación propios del *Internet of Things*, tales como las conexiones dispositivo-nube, dispositivo-puerta de enlace y el intercambio de datos a través del *back-end* [20].

En cuanto a los ámbitos de aplicación, se extienden a casi cualquier aspecto de la vida cotidiana, desde dispositivos conectados al cuerpo humano, pasando por aplicaciones industriales hasta dispositivos destinados al entretenimiento, y por supuesto también está presente en la agricultura. Entre las aplicaciones típicas en las que se emplea tecnología IoT en la agricultura, se encuentra el envío de la información recolectada por sensores hacia la nube para posteriormente realizar un análisis y ejercer algún tipo de control, por ejemplo, mediante sistemas robóticos como tractores o vehículos aéreos no tripulados. En el presente trabajo de tesis se emplea una estrategia similar, capturando y enviando en este caso imágenes de un cultivo, hacia un servidor para que estas puedan ser procesadas y analizadas.

2.4.Procesamiento de Imágenes

Al hablar de procesamiento de imágenes se cuenta con un amplio abanico de algoritmos y procedimientos para llegar a una gran cantidad de resultados posibles. Así, es posible realizar tareas como el filtrado, la segmentación, la clasificación, cambios de espacio de color, entre otros mecanismos de procesamiento [18], [19]. En cuanto, al nivel de análisis, los algoritmos pueden actuar a nivel de pixeles, por lotes o global, en el dominio del espacio o en el dominio de la frecuencia. Adicionalmente, existen métodos de mayor complejidad como por ejemplo, las redes neuronales [22],[23] o los algoritmos de aprendizaje supervisado tipo *Support Vector Machine* (SVM) [24]. La aplicación de un mecanismo u otro depende en gran medida del problema al que se enfrente. Estas tareas y algoritmos de procesamiento de imágenes por lo general se usan conjuntamente formando marcos de trabajo por los cuales las imágenes a procesarse pasan secuencialmente.

El problema de procesamiento de imágenes al cual se hará frente en este trabajo de tesis, consiste en la detección de frutos sobre fotografías capturadas a una distancia entre cinco y diez metros. En concreto la tarea fundamental se enfoca en identificar y separar los frutos del fondo de la imagen. Por tanto, la complejidad del procesamiento se debe a la existencia de elementos adicionales en las fotografías, tales como hojas, lotes de tierras, ramas, cielo, nubes o montañas.

Como solución al problema planteado, se propone la aplicación de dos algoritmos descritos en la literatura. El primer mecanismo consiste en realizar un filtrado espectral de los pixeles, dicho esquema se denomina *Thresholding* [25]. El otro método que se evaluará está basado en el agrupamiento o *clustering* de los píxeles de la imagen, en el cual cada pixel se coloca dentro de un grupo o *cluster* acorde a un criterio previamente definido [26].

Se contará con un marco de trabajo para cada algoritmo; estos serán desarrollados mediante una serie de experimentos a partir de los datos provistos por un conjunto de imágenes tomadas en campo. El esquema del marco de trabajo basado en *Thresholding* se muestra en la Figura 1. Este marco de trabajo comienza con una conversión al espacio de color HSV para posibilitar una segmentación por color más sencilla. Luego se implementa el algoritmo de *Thresholding* para filtrar los pixeles con los matices deseados, segmentando la región de interés del fondo de la imagen. Posteriormente se genera la imagen binaria que permitirá realizar las operaciones subsecuentes. La binarización de la imagen consiste en colocar los pixeles de la región de interés en color negro, dándole un valor binario de 1, mientras que el fondo de la imagen pasará a estar de color blanco con un valor binario de 0 (este proceso también podría ser inverso).

Las operaciones morfológicas permiten eliminar o agregar pixeles a la imagen para reducir el ruido o rellenar espacios y así mejorar los resultados. Posteriormente se utiliza el algoritmo de etiquetado de componentes conectados para colocar una etiqueta en cada una de las manchas de la imagen binaria. Después de esto se realiza el conteo de objetos que servirá para estimar el número de frutos presentes. Luego se realiza un encuadre de las manchas encontradas en la imagen original para visualizar los frutos encontrados. Finalmente se calculan estadísticas de la imagen tales como el número de manchas encontrado, número de frutos estimado, tamaño mínimo, máximo y promedio de las manchas y color promedio de los frutos.

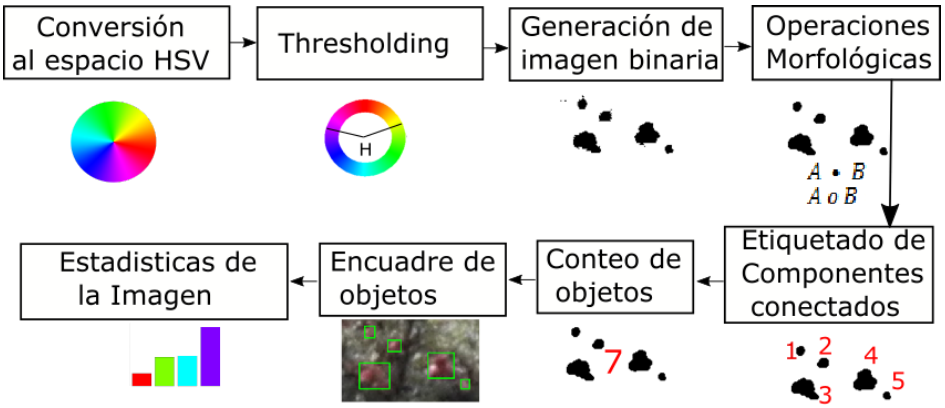

Figura 1 Marco de trabajo del procesamiento de imágenes con Thresholding

El marco de trabajo basado en *clustering* se muestra en la Figura 2. En primer lugar se realiza un cambio de espacio de color. En el Capítulo 4 se describe como se encontró el espacio de color usado finalmente. Luego se realiza la agrupación de píxeles (*clustering*) mediante el algoritmo de Esperanza-Maximización (EM, por las siglas en inglés de *Expectation-Maximization*). Luego es necesario identificar que *cluster es* el que agrupa a los frutos.

Posteriormente se genera la imagen binaria y se procede de manera similar que en el marco de trabaja basado en Thresholding.

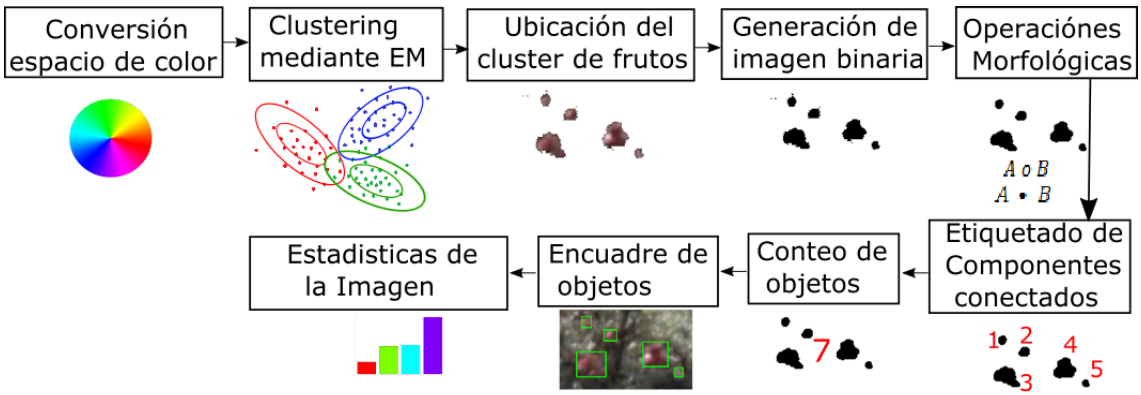

Figura 2 Marco de trabajo del procesamiento de imágenes con GMM

2.4.1. Espacios de Color

Los colores en una imagen están definidos por una tupla de tres números. El significado de cada uno de estos números y su mapeo en la gama de colores es lo que define un espacio de color [27]. El espacio de color más usado en la mayoría de sistemas digitales es el espacio RGB (*Red-Green-Blue*), sin embargo, existen otras alternativas, cada una con diferentes características que las hacen atractivas según el problema que se intente abordar. En cuanto a los sistemas digitales, la forma más común de representar los espacios de color es mediante 24 bits, 8 por cada canal. Como resultado se tiene 256 niveles por cada canal y un total de 16.7 millones de tonalidades posibles. En el presente trabajo se experimentó con algunos de ellos, cuyas características se presentarán a continuación.

El espacio RGB es un espacio de color aditivo formado por una combinación lineal de los valores de color rojo (*Red*), verde (*Green*) y azul (*Blue*). La información de crominancia (información del color) y luminancia se encuentra en la mezcla de los tres componentes, por lo que ante cambios de luminosidad del ambiente los tres canales sufren cambios notables, siendo este un problema al intentar segmentar imágenes con diferente iluminación[28]. Otra desventaja de este espacio, es que no es perceptivamente lineal, es decir, que un cambio percibido como pequeño por la vista no siempre es así numéricamente.

El espacio HSV tiene como componentes el matiz de la longitud de onda dominante (Hue), la saturación (S) y el valor (Value) que es la magnitud de intensidad lumínica del pixel. La principal ventaja de este espacio es que tiene un solo canal para representar el color (el canal *Hue*), a diferencia de RGB que tiene tres. Por esta razón este espacio de color es muy utilizado cuando se desea filtrar una gama específica de colores [29]. Otra ventaja de este espacio es que ante cambios de luz en el ambiente el único canal que se verá notablemente afectado es el canal *Value*. Como principal desventaja se tiene que es dependiente del dispositivo en el que se visualiza.

El espacio LAB tiene como componentes la luminosidad (*Lightness*), el canal de color A (componente de color desde el verde al magenta) y canal de color B (componente de color desde el azul hasta el amarillo). En este caso, la información de color está codificada en dos

canales, mientras que el canal restante provee información de la intensidad lumínica. La principal ventaja del espacio LAB, es que es independiente del dispositivo en el que se visualiza, además de ser perceptualmente lineal.

2.4.2. Thresholding

Thresholding consiste en un algoritmo muy sencillo que logra segmentar imágenes al pasar sus pixeles por un umbral o serie de umbrales con valores específicos. Se distinguen dos tipos de Thresholding, aquel realizado sobre la imagen en escala de grises y aquel que trabaja sobre la imagen a color. En el presente trabajo de tesis se empleará el segundo tipo.

El *Thresholding* a color comúnmente trabaja sobre el espacio HSV, el cual como se mencionó previamente tiene la ventaja de que solo uno de sus canales muestra el color. Adicionalmente, en la literatura se describe que es posible obtener buenos resultados combinando el algoritmo de *Thresholding* y el espacio de color RGB [30].

La principal ventaja del Thresholding es su sencillez y bajo costo de procesamiento. No obstante, tiene como desventaja, la dificultad para segmentar objetos con matices semejantes al fondo de la imagen. Además, que al aumentar el tamaño del umbral para ampliar la cantidad de objetos segmentados, el número de falsos positivos se incrementa de forma significativa. Por otra parte, el principal desafío con este método, consiste en determinar los valores óptimos para el umbral, tarea que depende mucho de las características de la imagen como por ejemplo la iluminación o la resolución.

2.4.3. Clustering

El *Clustering* o agrupación es un proceso de aprendizaje no supervisado, es decir que a priori no se conoce una salida para los datos de entrada, con el cual se separan los elementos del conjunto de entrada en diferentes grupos o *clusters*. En el procesamiento de imágenes, dicho mecanismo es útil para realizar la tarea de segmentación de la imagen, es decir para separar del resto de elementos, la parte de la imagen que es de interés para el problema.

Existen varias formas de lograr esta agrupación, una de ellas es el uso de modelos de mezclas de distribuciones gaussianas (GMM por las siglas en inglés de *Gaussian Mixture Models*). Estos modelos parten de la premisa de que los datos del conjunto de entrada fueron generados por la mezcla de una cantidad finita y conocida de distribuciones gaussianas con parámetros desconocidos [31].

Los parámetros a determinar que definen un modelo GMM con n *clusters* y un conjunto de entrada $\boldsymbol{X}$, donde cada una de sus entradas $\boldsymbol{x_i}$ es un vector de dimensión m, son la ubicación y forma de cada una de las distribuciones gaussianas multivariables, que para un *cluster* c se definen mediante la ecuación (1):

$$\mathbf{N}(\mathbf{x_i}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}) = \frac{1}{(2\pi)^{\frac{n}{2}}! \, |\boldsymbol{\Sigma_c}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x_i} - \boldsymbol{\mu_c})^{\mathrm{T}} \boldsymbol{\Sigma_c}^{-1}(\mathbf{x_i} - \boldsymbol{\mu_c})\right) \qquad (1)$$

Donde $\mathbf{N}(\mathbf{x_i}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})$ es la distribución gaussiana multivariable para el *cluster* c, $\boldsymbol{\mu}$ es el vector de medias con n entradas (una por cada *cluster* c) de dimensión m y $\boldsymbol{\Sigma}$ es el vector de covarianzas con n entradas de dimensión mxm. Además, es necesario determinar el vector $\boldsymbol{\pi}$ de dimensión n; este vector define la probabilidad de que un punto de dato $\boldsymbol{x_i}$ se encuentre en cada *cluster*.

Para la estimación de estos parámetros se utiliza el algoritmo de *Expectation-Maximization* (EM) que iterativamente ejecuta una etapa de esperanza (E), en la cual se crea una función para el cálculo de la esperanza de la verosimilitud logarítmica usando la estimación actual de los parámetros, y una etapa de maximización (M) que maximiza la esperanza de la verosimilitud logarítmica para obtener parámetros de máxima verosimilitud [32].

Para empezar, el algoritmo inicializa los parámetros ($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ y $\boldsymbol{\pi}$) de manera aleatoria para un número de *clusters* dado. Luego de esto se procede con la etapa E en la que para cada dato $\boldsymbol{x_i}$ se calcula la probabilidad r_{ic} de que dicho punto pertenezca al *cluster* c, mediante la ecuación (2).

$$r_{ic} = \frac{\pi_c \boldsymbol{N}(\boldsymbol{x_i}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})}{\sum_{k=1}^{K} \pi_k \boldsymbol{N}(\boldsymbol{x_i}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})} \tag{2}$$

En la etapa M se calcula para cada *cluster* c el peso total m_c que es la porción de probabilidad de cada *cluster* calculada sobre todos los puntos del conjunto de datos de entrada, como se muestra en la ecuación (3):

$$m_c = \sum_i r_{ic} \tag{3}$$

Luego se normaliza sobre la suma total de m_c de todos los *clusters* para formar el vector $\boldsymbol{\pi}$, como se aprecia en la ecuación (4):

$$\pi_c = \frac{m_c}{m} \tag{4}$$

Dónde:

$$m = \sum_c m_c \tag{5}$$

Y a continuación se actualiza también $\boldsymbol{\mu_c}$ y $\boldsymbol{\Sigma_c}$ como se muestra en las ecuaciones (6) y (7) respectivamente:

$$\boldsymbol{\mu_c} = \frac{1}{m_c} \sum_i r_{ic} \boldsymbol{x_i} \tag{6}$$

$$\boldsymbol{\Sigma_c} = \frac{1}{m_c}\sum_i r_{ic}(\mathbf{x_i} - \boldsymbol{\mu_c})^{\mathrm{T}}(\mathbf{x_i} - \boldsymbol{\mu_c}) \quad (7)$$

Estas se etapa se las repite iterativamente hasta que la función de verosimilitud logarítmica del modelo converja. Esta función se la calcula como se muestra en la ecuación (8):

$$\ln p(\boldsymbol{X}|\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k \boldsymbol{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)) \quad (8)$$

Los resultados de aplicar el modelo GMM en conjunto con el algoritmo EM, depende en gran medida de la cantidad de *clusters* que se seleccione para realizar los cálculos. En particular, un pequeño número de *clusters* podría agrupar en un solo grupo tanto características deseadas como no deseadas de la imagen, lo que se traduce en una segmentación deficiente que presenta la región de interés rodeada de pixeles que no le corresponden. Por otro lado, un número grande de *clusters* puede producir un sobre-ajuste del modelo, ocasionando regiones de interés incompletas, además de que esto conlleva un coste de procesamiento mayor.

Consecuentemente, existen varios criterios con los cuales estimar un número adecuado de *clusters*, entre ellos el criterio de Akaike o la Validación Cruzada [33]. Sin embargo, uno de los criterios más utilizados actualmente, mismo que se empleó en el presente trabajo, es el Criterio de Información Bayesiano (BIC por las siglas en inglés de *Bayesian Information Criterion*) [32],[34]. Este criterio se calcula como se muestra en la ecuación (9):

$$BIC \approx \frac{1}{2} * k_j * \ln(n) - \ln L(\widehat{\boldsymbol{\theta}}) \quad (9)$$

Dónde:

$$L(\widehat{\boldsymbol{\theta}}) = p(\boldsymbol{x}|\widehat{\boldsymbol{\theta}}, M) \quad (10)$$

$L(\widehat{\theta})$ es la medida de la verosimilitud maximizada del modelo M con el conjunto de datos de entrada $\boldsymbol{x}$. $\widehat{\boldsymbol{\theta}}$ es el valor de los parámetros que maximizan la función de verosimilitud. La constante k_j es el número de parámetros libres que se estiman en el modelo y n es el número de observaciones, es decir la cantidad de entradas en $\boldsymbol{x}$. En los modelos GMM los parámetros libres son los valores de las medias y las matrices de covarianzas, por lo que el número de parámetros libres para un modelo con i *clusters* y con datos de entrada de dimensión m se puede calcular con la ecuación (11):

$$k_j = i * (1 + m + (m * (m + 1)/2)) \quad (11)$$

El criterio BIC se aplica sobre el modelo una vez sus parámetros hayan sido ajustados con el algoritmo EM, por lo que el proceso de encontrar el número adecuado de *cluster* se torna iterativo. El proceso inicia ajustando el modelo con $i = 2$, se evalúa BIC y luego se incrementa el número de *clusters* en 1. El procedimiento se repite hasta alcanzar el número límite de

clusters impuesto. Una vez se cuente con la evaluación de BIC en todos los modelos ajustados, algunos autores recomiendan elegir el modelo con el menor valor de BIC, mientras que otros recomiendan elegir el primer mínimo local encontrado, como se resalta en [35]. En la práctica, elegir el modelo con el menor valor de BIC suele añadir un costo de procesamiento a veces innecesario, por lo que, el segundo enfoque suele ser el más indicado. Además en ciertas fuentes se suele definir BIC como el negativo de la ecuación (9), en este caso se elige el modelo con el máximo local[13].

Por otra parte, para que el modelo sea útil para la segmentación de imágenes, el conjunto de entrada debe consistir en los pixeles de una imagen que contengan una considerable cantidad de los elementos que se quieren segmentar. Naturalmente incluyendo el fondo característico que tenga las imágenes a ser segmentadas. El procedimiento descrito se puede comprender con mayor claridad mediante el diagrama de flujo mostrado en la Figura 3.

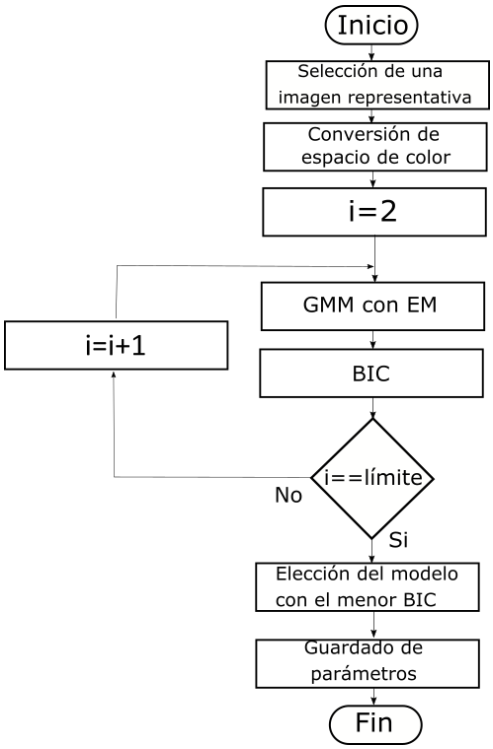

Figura 3 Diagrama de flujo del ajuste del modelo

Finalmente, una vez que el modelo ha sido ajustado y se han obtenido los parámetros específicos de cada *cluster*, el siguiente paso consiste en determinar el *cluster* en el cual los frutos han sido agrupados. Sin embargo, puesto que se está trabajando con un modelo de aprendizaje no supervisado, el modelo no contendrá esta información. En tal sentido, la solución implementada consistió en analizar las características espectrales que debe tener un pixel para ser parte del *cluster* de frutos y realizar una predicción de tal pixel con el modelo ajustado. Por ejemplo, si los frutos son de color rojo, se predecirá el *cluster* al que pertenece un pixel rojizo.

2.4.4. Operaciones Morfológicas

Una vez que se cuenta con la imagen segmentada y sus elementos han sido binarizados, es posible mejorar los resultados obtenidos mediante la aplicación de Operaciones Morfológicas.

Las Operaciones Morfológicas permiten realizar un procesamiento sobre estructuras geométricas disponibles en una imagen. Los operandos para dicho procedimiento, por lo general consisten en una imagen binaria de entrada y un elemento estructural. En cuanto a los elementos estructurales, son formas con las cuales la imagen de entrada interactuará según la operación morfológica lo defina. Estos elementos suelen ser estructuras binarias, siendo las formas más comunes, círculos, cuadrados y óvalos, aunque también es posible definir estructuras más complejas.

Las dos operaciones morfológicas básicas son la dilatación y la erosión. La dilatación, representada por el signo $\oplus$, expande la imagen en sus bordes al interactuar con un elemento estructural. Para una imagen A y un elemento estructural B, la dilatación se define como se muestra en la ecuación (12):

$$A \oplus B = \bigcup_{b \in B} A_b \tag{12}$$

Donde A_b es la traslación de A por b. Así, se puede entender la dilatación como el emplazamiento de los puntos de B en A, cuando el centro de B se encuentra en algún punto de A [36].

La erosión, representada por el signo $\ominus$, contrae los bordes de la imagen de acuerdo a su interacción con el elemento estructural. Para una imagen A y un elemento estructural B, la erosión se define como se observa en la ecuación (13):

$$A \ominus B = \bigcap_{b \in B} A_{-b} \tag{13}$$

Donde A_{-b}, es la traslación de A por $-b$. La erosión puede ser entendida como la translación del centro de B sobre cada punto de A, en la cual, si todos los puntos de B están contenidos en A, el punto permanece, caso contrario dicho punto se elimina.

De la dilatación y la erosión se derivan dos operaciones morfológicas muy usadas en el procesamiento de imágenes, que son la Apertura (*Opening*) y la Cerradura (*Closing*). La operación *Opening*, tiene la capacidad de rellenar espacios en blanco dentro de una imagen binaria. En cuanto a su representación, se emplea el signo o. Además, matemáticamente se define como la dilatación de la erosión de una imagen A por un elemento estructural B, tal como se muestra en la ecuación (14).

$$A \, o \, B = (A \ominus B) \oplus B \tag{14}$$

La operación *Closing*, tiene la capacidad de eliminar elementos semejantes al elemento estructural, como pueden ser manchas de ruido en la imagen. Está representada por el signo •, y está matemáticamente definida como la erosión de la dilatación de una imagen A por un elemento estructural B, como se lo puede observar en la ecuación (15).

$$A \bullet B = (A \oplus B) \ominus B \tag{15}$$

2.4.5. Etiquetado de Componentes Conectados

El algoritmo de etiquetado de componentes conectados (CCL, por las siglas en inglés de *Connected-Components Labeling*) actúa sobre imágenes binarias, asignando una etiqueta única a cada mancha (conjunto de píxeles conectados) de la imagen [37]. Este procesamiento resulta indispensable en tareas como conteo de objetos, reconocimiento de patrones y extracción de características. La funcionalidad básica de este algoritmo consiste en buscar los pixeles con un valor binario igual a 1 y verificar en toda su vecindad si hay más pixeles con valor igual a 1, como un indicador de que existe conexión. Para aquellos casos donde se tenga un conjunto aislado de pixeles se asigna una etiqueta a dicho grupo. En la práctica, implementar el algoritmo en la forma descrita, no es la opción más adecuada, por lo que en la literatura se proponen algunas alternativas para su optimización [38], [39], [40]. En el presente trabajo de tesis, se utilizará la librería OpenCV [41] que cuenta con un método que implementa dos versiones del algoritmo, una que verifica los pixeles vecinos en las líneas horizontal y vertical, y otro que verifica los 8 píxeles circundantes.

2.4.6. Conteo de Objetos

Para realizar un conteo de objetos existen dos enfoques muy empleados. El más simple de ellos consiste en contar la cantidad de manchas. A su vez, dicha tarea resulta más simple cuando se ha realizado un etiquetado de los componentes conectados, ya que el proceso se reduce a tomar el número de etiquetas colocadas. El problema de emplear este enfoque para el conteo de frutos, es que podrían existir manchas que agrupan múltiples frutos, que en este caso serían contados como solo uno. El otro enfoque propone tomar el número total de pixeles y dividirlo para el área esperada de un fruto. En este caso, el inconveniente es que podría darse el caso de muchos frutos aislados con un área reducida por estar solapados por la vegetación que se contarían como una cantidad menor a la real.

Como solución alternativa en el presente trabajo de tesis se emplea una combinación de estos dos enfoques, es decir, se cuenta el número de manchas, y en el caso de tener manchas muy grandes que indique un posible agrupamiento de frutos, se las divide por el área esperada de un fruto.

2.5. Procesamiento de Video

El procesamiento de video involucra una amplia cantidad de tareas posibles de acuerdo al resultado buscado. En el presente trabajo, se hará mención solo a las tareas necesarias para su transmisión y almacenamiento. Como punto de partida, es necesario considerar que en un video los fotogramas sin procesamiento, es decir en formato *raw* presentan una cantidad

significativa de información que en gran parte son inapreciables o redundantes. En tal sentido, es necesario aplicar mecanismo de compresión para que su transmisión y almacenamiento sean más sencillos y eficientes. Con tal objetivo se emplean algoritmos especializados denominados CODECs, encargados de codificar y decodificar archivos de video. Existen varios estándares de compresión o codificación de video, entre los más usados en la actualidad, se encuentran, H.265, H264, VP8 y VP9.

En tal contexto resulta útil el *software* FFmpeg [42], que es un programa de manejo de contenido multimedia que agrupa un conjunto de herramientas para codificar, decodificar, multiplexar, hacer *streaming*, filtrar y reproducir video.

2.6. Tecnologías para el Desarrollo de Aplicaciones Web

A diferencia de las aplicaciones de escritorio que son programas que se ejecutan sobre el sistema operativo de una plataforma nativa, las aplicaciones web están alojadas y se ejecutan en el servidor del desarrollador para ser visualizadas en un navegador. Este tipo de soluciones presenta grandes ventajas tales como, un ahorro de espacio en disco, la facilidad de actualización, la compatibilidad entre distintas plataformas, portabilidad y seguridad. Además, existen aplicaciones híbridas, en las cuales la plataforma del usuario aloja un programa pequeño que se complementa con las funcionalidades del servidor, tal es el caso de las aplicaciones móviles de Facebook o Twitter.

En el trabajo de tesis, se desarrollará una aplicación web para visualizar la información capturada por la cámara fotográfica y los sensores, para que pueda ser visualizada desde cualquier lugar con la ayuda de un navegador web.

Específicamente, el desarrollo de la aplicación web se la realizó sobre la plataforma IBM Cloud. Esta plataforma provee una gran variedad de servicios sobre la nube que permiten diseñar y desarrollar aplicaciones, incluso es posible configurar gran parte de la infraestructura tecnológica que una empresa clásicamente implementaba dentro de sus *Data Centers*. Estos servicios tienen la ventaja de ser ubicuos, además de escalables, es decir, se puede incorporar mayor cantidad de servicios o recursos según el cliente lo necesite y solo se pagará por los recursos utilizados. En el presente trabajo se hará uso de dos servicios prestados por la IBM Cloud, el servicio de *Node-Red Starter* y el servicio de *Internet of Things Platform.* En el Apéndice A se describe el entorno de trabajo de la IBM Cloud.

Para el caso de la comunicación entre la estación y la aplicación web desarrollada en la nube, se utilizará el protocolo MQTT, el cual está especialmente diseñado para el ámbito de soluciones IoT.

2.6.1. Node-Red Starter

Node-Red starter es un servicio en la nube del tipo *Platform as a Service* (Plataforma como servicio, PaaS por sus siglas en inglés), en el cual la empresa dueña de la nube proporciona una plataforma a los desarrolladores de aplicaciones para crear y desplegar aplicaciones. Así estas plataformas incluyen distintos lenguajes de programación además de servicios de red para que los usuarios accedan a las aplicaciones. En este servicio en particular, las aplicaciones se desarrollan en Node-Red [43], una herramienta de programación gráfica construida sobre el entorno de ejecución Node.js. Dicho entorno se encuentra optimizado para trabajar sobre aplicaciones de red, además cuenta con una gran cantidad de librerías, entre ellas algunas que permiten desplegar páginas web de manera rápida sin necesidad de ocuparse de la programación de las etiquetas HTML o del estilo en CSS. El editor de Node-Red, se

ejecuta en el navegador web, lo cual permite la compatibilidad con muchas plataformas. En el Apéndice B se describe como crear una nueva instancia de este servicio así como los detalles de su entorno de trabajo.

2.6.2. Internet of Things Platform

Este servicio también llamado *IBM Watson IoT Platform,* está destinado a la fácil gestión de dispositivos tipo IoT. Para ello proporciona los servicios de registro de dispositivo, conexión, almacenamiento, visualización de datos además de simulación de dispositivos y conectividad con la plataforma IBM Cloud. Adicionalmente, es posible configurar distintos niveles de seguridad en los dispositivos, y crear grupos para su administración. Internamente este servicio implementa un *broker* MQTT al que se puede acceder desde el servicio de Node-Red para conectar la aplicación web con los dispositivos IoT. El tamaño máximo de los paquetes MQTT que este broker permite es de 128kB. En el Apéndice C se describen los pasos para crear una nueva instancia de este servicio, así como los detalles de su entorno de trabajo.

2.6.3. Protocolo MQTT

MQTT es un protocolo para la transmisión de mensajes, diseñado especialmente para arquitecturas tipo IoT. Su popularidad sobre protocolos similares, es consecuencia de su fácil implementación y especialmente por ser liviano y usar pocos recursos, lo cual es indispensable para dispositivos IoT, que por lo general tienen limitados recursos energéticos y de ancho de banda.

MQTT está basado en la pila TCP/IP y es un protocolo con un modelo publicación/suscripción, es decir, los clientes se conectan mediante TCP/IP con un servidor denominado *broker* localizado en la nube o en una red local. A partir de la conexión establecida, el cliente puede enviar mensajes con cierto tópico para que luego estos puedan ser alcanzados por los dispositivos suscritos al mismo. Este esquema convierte a MQTT en un protocolo asíncrono, es decir que los extremos de la comunicación no deben tener conexión directa, lo cual es muy conveniente cuando se cuenta con una gran cantidad de dispositivos interesados en recibir mensajes [44].

MQTT utiliza por defecto el puerto 1883, y en el caso de funcionar sobre TLS (*Transport Layer Security*) utiliza el puerto 8883. El protocolo puede manejar paquetes de hasta 256MB. Sin embargo, en la práctica este tamaño depende de la implementación en el *broker*, por lo que suele ser considerablemente menor.

Un aspecto muy importante en MQTT es la calidad de servicio. La calidad de servicio (QoS por las siglas en inglés de *Quality of service)* es un mecanismo para afrontar las posibles fallas de conexión debidas al medio u otros factores. MQTT implementa 3 niveles de QoS. En el nivel 0, los mensajes se envían solo una vez, por lo que ante un fallo de conexión los mensajes no llegarán al receptor y no se tendrá constancia del error. El nivel 1, se envía los mensajes hasta garantizar la entrega, por lo que el suscriptor podría recibir mensajes duplicados. En el nivel 2, se garantiza la entrega de los mensajes a los suscriptores una única vez [45].

En cuanto al *broke*r, es posible crearlo instalando el servidor Eclipse MQTT en un ordenador tanto para una red local como para su acceso a Internet, en caso de que se cuente con una dirección IP pública. Sin embargo, lo más sencillo es usar un servicio de *broker* MQTT público o privado.

El esquema de funcionamiento de MQTT, se muestra en la Figura 4. En particular, el cliente que desea publicar contenido tiene que registrarse en el *broker* mediante un mensaje CONNECT que lleva información del nombre de usuario y contraseña, lo cual es respondido por el servidor con un mensaje CONNACK. A continuación, los clientes que quieran suscribirse a la información que se publique deben enviar un mensaje SUBSCRIBE al broker con la información del tópico de los mensajes que desea recibir. El *broker* responde a esta solicitud con un mensaje SUBACK. Cuando el cliente publicista desea enviar contenido al *broker*, lo hace mediante un mensaje PUBLISH que lleva información del tópico del mensaje. Finalmente, el *broker* se encarga de distribuir este mensaje a todos los clientes que estén suscritos al tópico.

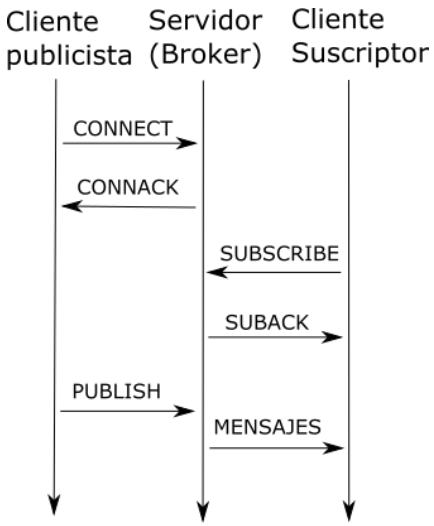

Figura 4 Esquema de funcionamiento de MQTT

2.7. Conclusiones

En este capítulo se ha revisado los principales conceptos relacionados con el trabajo de tesis, desde aquellos que tienen que ver con su contexto tales como la definición de IoT y la agricultura de precisión, hasta los detalles técnicos requeridos para comprender el funcionamiento de la aplicación, como por ejemplo las tecnologías web y los algoritmos usados para el procesamiento de imágenes. Además, de manera particularmente importante se expuso las diferentes etapas requeridas para el tratamiento y análisis de las imágenes, como son la binarización, segmentación, operaciones morfológicas y conteo. Dichos procesos definen el marco de trabajo a seguir en el Capítulo 4.

3. TRABAJOS RELACIONADOS

En este capítulo se describen brevemente los principales trabajos disponibles en la literatura relacionados con el tema de tesis. Conforme a esto, se reseñan trabajos en torno a la creación de estaciones de monitoreo, el procesamiento de imágenes, comunicación inalámbrica entre dispositivos y tecnologías para el desarrollo de aplicaciones web.

En la actualidad existen una gran variedad de plataformas de desarrollo, las mismas que pueden emplearse para el despliegue de redes se sensores y sistemas de monitorización. En particular, se destaca el trabajo descrito en [46], donde se emplea una plataforma Raspberry Pi, como estación móvil. Para ello, dicha plataforma fue emplazada en un dron con el objetivo de gestionar los datos que recibe de los sensores, la conexión a Internet, así como la adquisición de imágenes y video. Por otra parte, en cuanto a la arquitectura de comunicación, se emplea la red celular para la conexión con la estación base. Esto debido al tráfico significativo generado por la cámara de video que restringe otro tipo de soluciones tales como redes basadas en ZigBee o LoRa. Además, que por las características de su aplicación es necesario contar con un rango de cobertura bastante amplio. En cuanto, a la adquisición y transmisión de video, para que la información de la cámara sea útil, los fotogramas deben tener la suficiente calidad para ser evaluadas por un experto, y a la vez mantener una tasa de datos baja para no producir latencia en el video. Con esta premisa, a partir de las pruebas realizadas, los autores proponen una resolución de 240x120 pixeles a una tasa de 15 fps (fotogramas por segundo). Finalmente, otro aspecto a destacar del mencionado trabajo, es el ahorro de energía, para lo cual se plantea el uso de un protocolo que reduzca al mínimo la permanencia en estado activo de los nodos, incluyendo al nodo móvil.

Por otra parte, en cuanto al rol de la agricultura de precisión en la actualidad, en [47], se presenta un análisis sobre la importancia de las nuevas tecnologías como medio para optimizar la producción y los recursos dentro del entorno agrícola. Específicamente, se hace énfasis en los retos de implementación y de seguridad a los que se enfrentan este tipo de soluciones. Además, se exponen las principales aplicaciones y beneficios que se podrían obtener al integrar la agricultura con el mundo del *Internet of Things*.

En cuanto al ámbito de las técnicas de procesamientos de imágenes enfocados en la agricultura de precisión, en [13], se describe un trabajo donde se realiza una detección de tomates a partir de imágenes de alta resolución obtenidas por un vehículo aéreo no tripulado o UAV (*Unmanned Aerial Vehicle).* Con tal objetivo, se emplea un *framework* de clasificación espectral-espacial. Específicamente, se propone el uso de tres métodos de clasificación espectral, *Expectation-Maximization* (EM), Mapas Auto-organizados y K-medias. Como punto importante de este trabajo se destaca que en lugar de usar solo dos *clusters* de clasificación (tomates y no tomates), se recurre al uso del criterio de información Bayesiano (BIC) para obtener el número óptimo de *clusters*. Posteriormente realizan una clasificación espacial con el fin de eliminar falsos positivos y remover ruido. Como resultado de todo el proceso se concluye que el uso de un número adecuado de *clusters*, genera mayor precisión en la clasificación en cada uno de los tres métodos utilizados. Particularmente el método que obtuvo mayor precisión en la clasificación fue EM.

En [48], se describe un trabajo donde se realiza la clasificación entre plantas y maleza en un cultivo de remolacha azucarera, para lo cual se realiza tareas de segmentación y operaciones morfológicas para comparar las formas de las plantas. De manera similar, en [49], se detalla el uso de operaciones morfológicas para mejorar los resultados de la segmentación. La extracción de las características tiene por objetivo alimentar una red neuronal para la posterior clasificación de frutos con enfermedades. En [47], se realiza un procedimiento similar al trabajo descrito previamente, en este caso para la extracción de características útiles para la detección de enfermedades en naranjas. Además, se desarrollan experimentos empleando varios espacios de color, eligiendo finalmente el espacio RGB para el procesamiento de las imágenes. Por otra parte, en [30], se realiza una segmentación de imágenes de plantaciones de manzanas mediante *Thresholding*, a partir de lo cual se implementa un algoritmo para manzanas rojas y otro para manzanas verdes.

En cuanto al análisis de los espacios de color, en [50],[51],[52], se describen estudios donde realizan segmentación de imágenes mediante EM experimentando con los espacios de color YUV, HSV y LAB respectivamente, consiguiendo resultados sobresalientes y mostrando que la elección del espacio de color depende del escenario en el que se aplique.

Un aspecto a evaluar durante el procesamiento de imágenes, sin duda es la selección de los mecanismos o algoritmos adecuados. En tal sentido, en [53], se presenta una revisión detallada de las técnicas usadas en el procesamiento de imágenes. En particular, se mencionan algoritmos para pre-procesamiento, segmentación, extracción de características y clasificación. Aunque el artículo se enfoca en la evaluación de frutas y vegetales analizadas bajo un entorno controlado, con fotografías cercanas y con elementos aislados, lo cual dista de lo que se busca en el presente trabajo de tesis, sin embargo, ofrece comparativas muy útiles sobre los distintos algoritmos usados, ampliando el panorama de la visión artificial.

En cuanto al despliegue de aplicaciones web y su integración con el ámbito del *Internet of Things*, en [54], se hace uso de la herramienta de desarrollo Node-Red para desplegar rápidamente una aplicación que gestione los datos de una estación de monitoreo de la calidad del aire. Además, se destaca el uso del protocolo MQTT para el intercambio de datos entre la estación y los sensores. En [55], se describe otro ejemplo que emplea Node-Red, en este caso con el objetivo de implementar una aplicación que interactúa con el servicio de asistente de voz Alexa y un conjunto de sensores. En cuanto al intercambio de datos se utiliza el protocolo MQTT, y como *broker* los servicios web de Amazon. Por otra parte, el trabajo descrito en [56], resulta particularmente atractivo, puesto que se propone un sistema que interconecta una estación de monitoreo de variables ambientales dentro de una Raspberry Pi con la nube de IBM, ambos extremos usando Node-Red para desplegar sus aplicaciones. En [57], se muestra el uso de Node-Red para el despliegue de aplicaciones en un campo industrial, destacando la importancia del protocolo MQTT como un protocolo de comunicación asíncrono. Finalmente, en [58], se presenta una propuesta que hace uso del *broker* MQTT Mosquitto que es de código abierto, además de detallar algunas características de MQTT como el tamaño máximo de mensajes y el parámetro de calidad de servicio, que son vitales en cierto tipo de aplicaciones para asegurar la recepción de los mensajes.

4. IMPLEMENTACIÓN DEL SISTEMA DE MONITOREO

4.1.Introducción

En el presente capítulo se detallará la implementación del sistema de monitoreo planteado. Este sistema tiene tres partes principales, la estación prototipo encargada de la adquisición y procesado de datos, la aplicación web encargada del control y visualización de los datos, y el sistema de comunicación para el intercambio de datos de sensores e información de control. El esquema de este sistema se muestra en la Figura 5.

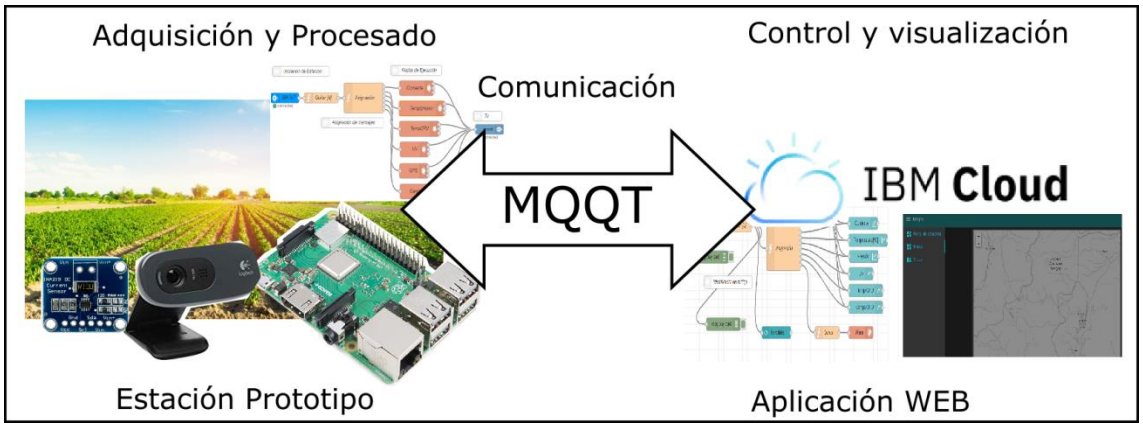

Figura 5 Esquema del sistema implementado

El capítulo está organizado de la siguiente forma. En la sección 4.2 se detalla el hardware utilizado para la implementación de la estación. En la sección 4.3 se describen las librerías utilizadas para el procesamiento de imágenes. En la sección 4.4 se describen los sistemas que hacen posible la comunicación entre la estación y la aplicación web. En la sección 4.5 se detalla el funcionamiento de los sensores y el procesamiento de imágenes y video, ejecutados sobre la estación prototipo. Finalmente, en la sección 4.6 se presenta la implementación de la aplicación web en la nube de IBM (*IBM Cloud*).

4.1.1. Descripción del Equipamiento

El prototipo diseñado en este trabajo de tesis, tiene la capacidad de adquirir señales ambientales tales como temperatura, presión, humedad e índice UV (Ultravioleta). Además de ello, incluye la capacidad de adquirir imágenes fotográficas y señales de video por lo que la estación está provista de una cámara. Además, puesto, que se busca tener precisión en la ubicación de la parcela en la que se emplace el prototipo, se cuenta con un sensor de geo-localización. Todas las señales adquiridas son procesadas por la estación y enviadas hacia Internet, por lo que todos los sensores y la cámara deberán estar concentrados en un mini-computador con capacidades de red. Adicionalmente, el prototipo cuenta con un periodo de autonomía energética para poder operar en lugares remotos, es decir, lleva incorporado una batería. Para agrupar todos estos elementos se construyó un encapsulado, cuya construcción y ensamblaje se muestran en el Apéndice G. A continuación, se detallará las características de los dispositivos usados.

4.1.2. Mini Computador Raspberry Pi

El hardware principal a ser usado para la implementación de la estación es el miniordenador Raspberry Pi modelo 2 B, un microordenador que cuenta con una interfaz para conexión Ethernet, memoria RAM de 1Gb, procesador Broadcom BCM2836 ARM Cortex-A7 a 900MHz, tarjeta gráfica Broadcom VideoCore IV 250 MHz.

OpenGL ES 2.0, 4 puertos USB y un precio de fábrica de USD 35. En [46], se hace una comparación entre varias plataformas: Raspberry Pi, Minnow Board, Minnow Board Max, BeagleBone Low-spec, BeagleBone High-Spec, Humming Board y Banana Pi, comparando su CPU, RAM, conectividad, comunidad de usuarios y costo. A partir del análisis se concluye que la plataforma Raspberry Pi, conjuga buenas características técnicas junto con un bajo costo y una amplia comunidad de usuarios.

Para su funcionamiento es necesario agregar una tarjeta de memoria microSD con un mínimo recomendable de 4GB de espacio, sobre la cual se instalará el sistema operativo. El sistema operativo más utilizado en la Raspberry Pi y que se hará uso en el presente trabajo es Raspbian, que es una distribución de Linux basada en Debian.

4.1.3. Sensores

Con el fin de medir el consumo energético de la estación se utilizó el sensor de corriente Adafruit INA219 [59] que tiene la capacidad de medir hasta 3.2A con una resolución de 0.8mA e incluso aumentar la precisión aceptando menores rangos de corriente. Las principales características de este sensor se muestran en la Tabla 1**.**

Tabla 1 Características técnicas del sensor de corriente Adafruit INA219

Voltaje de la fuente	3V - 5.5V
Temperatura de operación	-40°C - 125°C
Comunicación	I^2C
ADC	12bits
Resistencia de medición	0.1Ω, 1%

Para medir el índice UV se utiliza el sensor Adafruit SI1145 [60], el cual a partir de un análisis del espectro visible e infrarrojo, consigue estimaciones muy precisas del índice UV. Las principales características de este sensor se muestran en la Tabla 2.

Tabla 2 Características técnicas del sensor UV Adafruit SI1145

Voltaje de la fuente	3V - 5V
Temperatura de	-40°C - 85°C

operación	
Comunicación	I^2C
Espectro IR	550nm - 1000nm
Espectro visible	400nm - 0nm

La medición de temperatura, humedad y presión se agrupa en un solo sensor, en este caso se empleó el sensor Adafruit BME280 [61]. Cabe destacar, que pertenece a la nueva línea de sensores que reemplazan a los de la línea BMP. Las principales características de este dispositivo se muestran en la Tabla 3.

Tabla 3 Características técnicas del sensor de temperatura, humedad y presión Adafruit BME280

Voltaje de la fuente	3V - 5V
Temperatura de operación	-40°C - 85°C
Comunicación	I^2C, SPI
Precisión sensor humedad	±3%
Precisión sensor temperatura	±1°C
Precisión sensor presión	±1hPa

El GPS utilizado para la ubicación de la estación es el Adafruit Ultimate GPS v3 [62]. Este módulo tiene una alta sensibilidad de rastreo de -165dBm, puede rastrear señal de hasta 22 satélites en 66 canales, tiene una antena interna y la posibilidad de instalar una antena externa para mejorar la recepción. Las principales características de este módulo se muestran en la Tabla 4.

Tabla 4 Características técnicas del sensor de geo-localización Adafruit Ultimate GPS

Voltaje de la fuente	3V - 5.5V
Tamaño antena de parche	15mm x 15mm x 4mm
Comunicación	USB
Precisión de posición	1.8metros

Velocidad máxima	515m/s

La cámara utilizada para la adquisición de imágenes es la webcam Logitech c270 [63], con capacidad de adquirir video en alta definición e imágenes con calidad mejorada por software de tres megapixeles. Las principales características de esta cámara se muestran en la Tabla 5.

Tabla 5 Características técnicas de la cámara web Logitech c270

Resolución máxima	720p
Enfoque	Foco fijo
Comunicación	USB
Fps máximo	30fps
Campo visual	60°

Los módulos de sensores de BME280, INA219 y SI1145 se integraron en una sola placa electrónica junto con tres botones y tres indicadores leds. El diseño de esta placa se muestra en el Apéndice F.

4.1.4. Batería

Para dotar de autonomía energética, la estación cuenta con un paquete de batería externo. Esta batería es la *Ravpower External Battery Pack Element Series* [64], que cuenta con 2 salidas USB. Sus principales características se muestran en la Tabla 6.

Tabla 6 Características técnicas del paquete de batería externo Ravpower Element Series

Capacidad	10400mAh
Entrada	5V/2A max
Salida 1	5V/2A max
Salida 2	5V/1.5A max

4.2.Descripción del Software

Para el desarrollo de los scripts se usó en su mayoría el lenguaje de programación Python, y en menor medida Bash. Las librerías usadas para el funcionamiento de los sensores se describen en el Apéndice E. Para el procesamiento de imágenes se utilizaron tres librerías Numpy, OpenCV y Sklearn.

Numpy, es un paquete que extiende las capacidades matemáticas de Python al proporcionar un gran número de funciones y facilidades para el uso de arreglos N-dimensionales [65]. Dicha característica es fundamental para el procesamiento ya que las imágenes se tratan como

arreglos tridimensionales si son imágenes a color y bidimensionales si están en escala de grises.

OpenCV, es una librería de código abierto creada para el desarrollo de la visión artificial y aplicaciones tipo *machine learning*. Contiene una amplia cantidad de algoritmos y herramientas optimizadas para el procesamiento de imágenes [41]. En el presente trabajo se hará uso de sus herramientas para el manejo de las imágenes capturadas, así como del algoritmo para realizar etiquetado de componentes conectados, y obtener sus estadísticas.

Sklearn, es una librería diseñada para realizar tareas de minería y análisis de datos [66]. Cuenta con una implementación optimizada del algoritmo GMM mediante EM. Dicho algoritmo ha sido empleado en el presente trabajo de tesis, además de la herramienta para el análisis del número de *clusters* (BIC).

4.3.Sistemas de Comunicación

El sistema desarrollado cuenta con tres interfaces de comunicación, la interfaz para la comunicación de la estación a la *IBM Cloud*, la interfaz de comunicación desde la *IBM Cloud* a la estación y aquella que permite a la estación su conexión hacia Internet.

En cuanto al intercambio de datos entre la plataforma Node-Red en IBM Cloud y la estación de monitoreo, es necesario recurrir a la plataforma IBM Watson IoT. En esta plataforma se debe registrar los dispositivos que se usarán y generar una clave de API, este proceso se explica con detalle en el Apéndice D.

Cabe resaltar que, para posibilitar esta comunicación, se requiere previamente instalar las librerías node-red-contrib-ibm-watson-iot y node-red-contrib-scx-ibmiotapp. Dicha acción debe ser efectuada en el editor de Node-Red, tanto en el lado del dispositivo como de la aplicación web.

En cuanto a la interfaz que permite el acceso a Internet, dependiendo del lugar de emplazamiento de la estación, es posible emplear una conexión de tipo Ethernet, Wifi o incluso mediante un módulo 3G o 4G. En cualquier caso, se requiere configurar una dirección IP fija. En tal sentido, con el propósito que la conexión se realice de forma automática, dicha configuración se realizó mediante la edición del archivo *interfaces* ubicado en el directorio /etc/network/ del sistema operativo Raspbian.

4.3.1. Comunicación Estación – IBM Cloud

Una vez que se ha registrado el dispositivo, se requiere seguir un conjunto de pasos para enviar datos desde la estación hacia la aplicación web en *IBM Cloud*. En concreto, en el editor de Node-RED se tendrán que configurar bloques tanto del lado del dispositivo como de la aplicación web, como se detalla a continuación

4.3.1.1. Configuración de los Bloques en el Dispositivo

En el lado del dispositivo, se requiere trabajar con el nodo *wiotp out*, el cual permite enviar mensajes de eventos a la plataforma *IBM Watson IoT* mediante el protocolo MQTT. En la Figura 6, se muestra el nodo *wiotp out*, disponible en el entorno de Node-Red.

Figura 6 Nodo wiotp out

En cuanto a la configuración, se requiere ingresar a las propiedades del nodo y especificar el tipo de conexión como Dispositivo Registrado. En la Figura 7, se presenta una captura de la configuración realizada sobre dicho nodo.

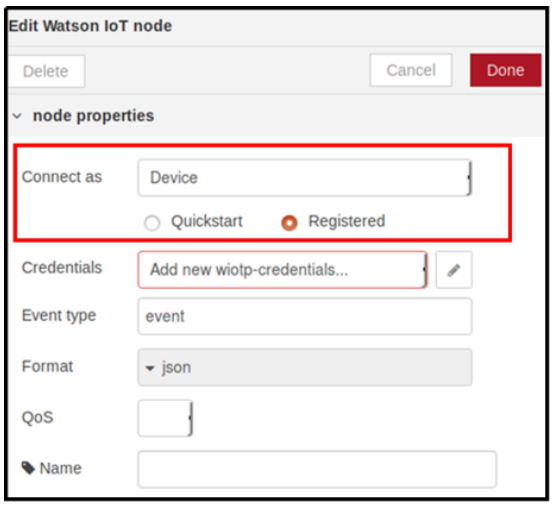

Figura 7 Propiedades del nodo wiotp out

Por otra parte, en la Figura 8, se presenta la edición de las credenciales para el dispositivo. Como se puede apreciar, el nombre del servidor puede omitirse ya que su valor por defecto es *[organization id].messaging.internetofthings.ibmcloud.com*. En cuanto al tipo de dispositivo, el ID de dispositivo y el token de autenticación son los colocados durante el registro del dispositivo. Es necesario aclarar que el *token* de autorización no se trata del obtenido al generar la clave de la API. Una vez que se ha completado los pasos anteriores, el nodo de envío estará completamente configurado por lo que resta configurar el nodo de recepción en el lado de la aplicación web.

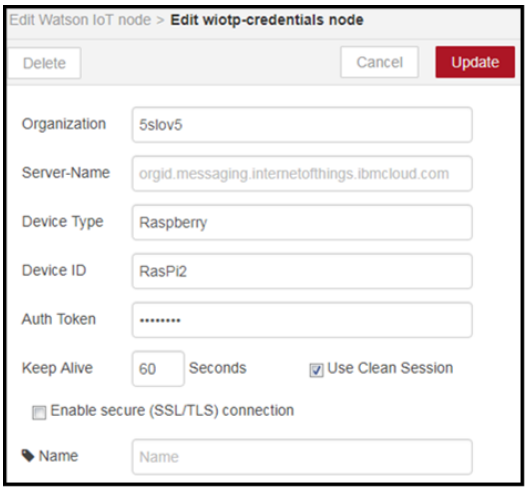

Figura 8 Credenciales nodo wiotp out

4.3.2. Configuración de los Bloques en la Nube

En cuanto al entorno Node-Red de lado de la nube IBM, se requiere trabajar con el nodo *ibmiot in*. En la Figura 9, se presenta una captura del mencionado nodo.

Figura 9 Nodo ibmiot in

La funcionalidad del nodo *ibmiot in*, consiste en que permite recibir mensajes de la plataforma *IBM Watson IoT*, provenientes de dispositivos y aplicaciones. Además, emplea el protocolo MQTT así como el formato json por defecto. Para su configuración, en primer lugar se debe modificar la opción de autenticación a "*API Key*", el tipo de entrada a "*Device Event*" y se ingresa la ID del dispositivo en el campo correspondiente como se muestra en la Figura 10.

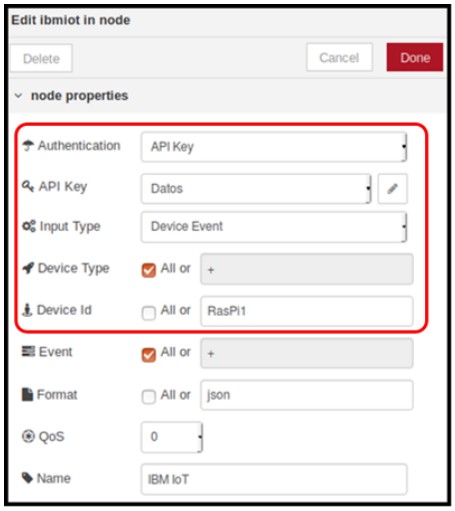

Figura 10 Propiedades nodo ibmiot in

A continuación, se procede a editar el campo *API Key* dando clic en su botón de edición. Específicamente, se debe ingresar un nombre, y los datos obtenidos en la API que fue generada previamente. En el campo *API Key* se ingresa la clave de API y en *API Token* la señal de autenticación de la API. De igual forma que en el lado del dispositivo, no es necesario colocar el nombre del servidor puesto que su valor por defecto es *[organization id].messaging.internetofthings.ibmcloud.com*. Una vez finalizado el procedimiento, se deben guardar los cambios y el nodo quedará configurado. En la Figura 11, se presenta una captura con los parámetros de edición del nodo.

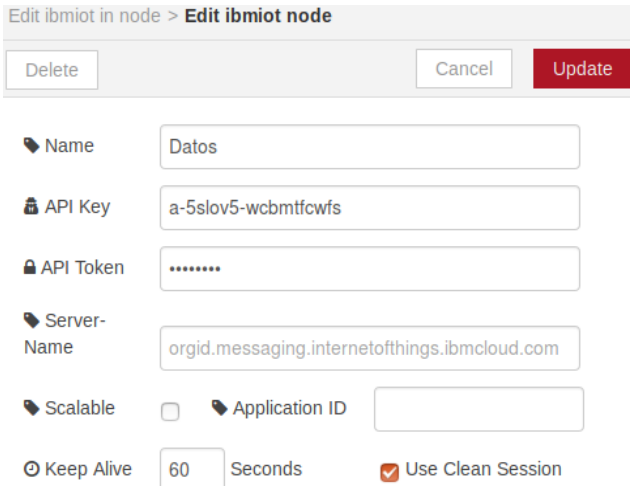

Figura 11 Formulario para ingresar credenciales de API

4.3.2.1. Prueba de Conexión

Para verificar el funcionamiento de la comunicación Estación-IBM Cloud, es posible emplear los nodos *inject* y *debug*, como se describe en el diagrama de la Figura 12.

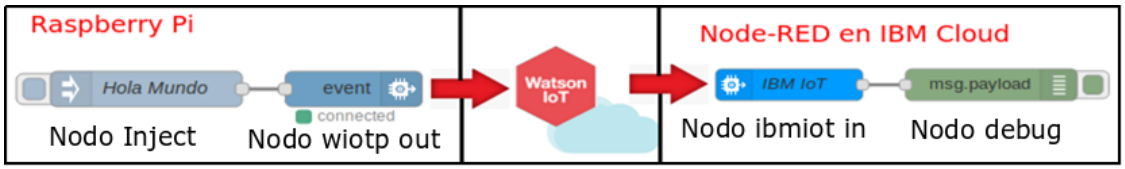

Figura 12 Comunicación Dispositivo - Nube

Al ingresar datos mediante el bloque de inyección (nodo inject), éstos serán recibidos por la aplicación web y serán visualizados en el panel *debug* como se demuestra en la Figura 13.

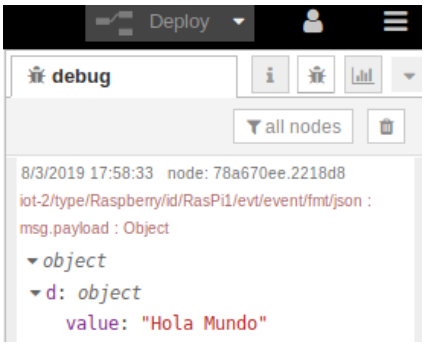

Figura 13 Panel debug en el lado IBM Cloud

4.3.3. Comunicación IBM Cloud – Estación

Hasta el momento se ha creado una comunicación unidireccional desde la estación hacia la aplicación web. A continuación, para crear la comunicación en el otro sentido es necesario registrar otro dispositivo, el cual se ha denominado como *"RasPi2"* y estará ligado al mismo dispositivo físico. Cabe indicar que es posible usar el mismo dispositivo registrado; sin embargo, suelen existir problemas de disponibilidad. En la Figura 14, se muestra una captura con los registros de los dispositivos creados.

Figura 14 Lista de dispositivos registrados

4.3.3.1. Configuración de los Bloques en el Dispositivo

En el lado de la estación, se desplegará un bloque *ibmiot in* y se lo configurará de manera casi idéntica a la efectuada para la recepción de mensajes en el lado de la aplicación web, con la única diferencia de que el ID del dispositivo apuntado será en este caso *"RasPi2"*, como se resalta en la captura de la Figura 15.

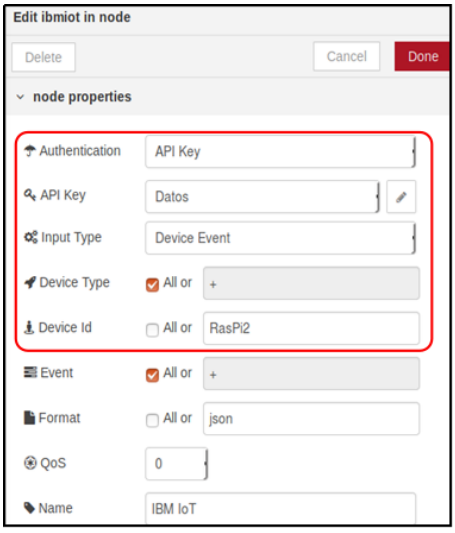

Figura 15 Propiedades nodo ibmiot in en dispositivo

Los datos de la clave de API serán los de la clave de API creada anteriormente. Nótese que ahora, para la comunicación bidireccional se tiene en el dispositivo un nodo *wiot* y otro *ibmiot,* lo mismo sucederá en el lado de la aplicación web.

4.3.3.2. Configuración de los Bloques en la Nube

En este lado se usará el bloque *wiotp out*. Se editan sus propiedades de forma similar a como ya se lo hizo en el lado del dispositivo para el envío de mensajes, pero ahora el dispositivo al que se apuntará será *RasPi2* en lugar de *RasPi1,* colocando sus respectivas credenciales como se muestra en la Figura 16.

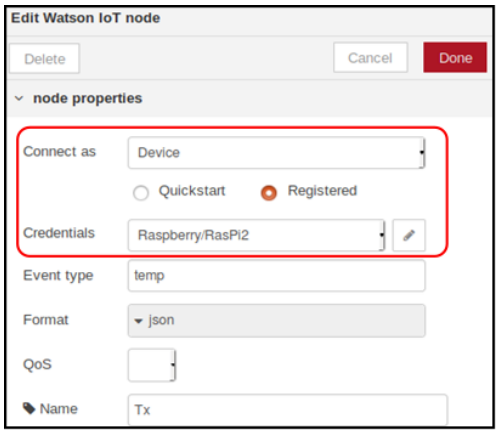

Figura 16 Propiedades nodo wiotp out en la nube

4.3.3.3. Prueba de Conexión

De igual forma es posible realizar una prueba de funcionamiento de la comunicación, mediante una conexión de bloques como se muestra en la Figura 17.

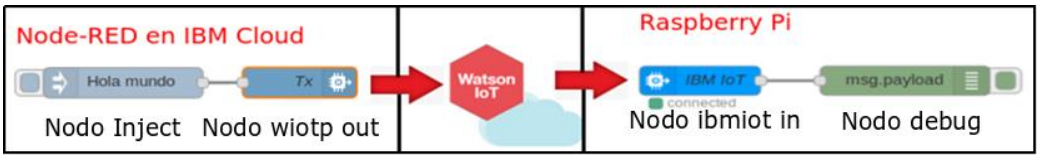

Figura 17 Comunicación Nube - Dispositivo

Al inyectar datos, en este caso se los recibe en el dispositivo y se visualizan en el panel de *debug* como se muestra en la **Figura 18**.

Figura 18 Panel debug en el lado del dispositivo

4.3.4. Estructura de los Mensajes

En el trabajo de tesis, los mensajes a ser transmitidos tienen como finalidad el transporte de datos de los sensores o de información de control. Al llegar, estos mensajes deben ser asignados a un nodo específico para su procesamiento. Con tal motivo los mensajes deben ser enviados con cierta estructura, incluyendo en su cabecera información necesaria para conocer a que nodo de procesamiento corresponde.

Los mensajes en Node-Red son enviados por defecto en formato JSON, de tal manera que la estructura de los mensajes que se ha diseñado es la siguiente:

{"codigo":"[código del sensor]","valor":"[]","topic":"treal"}

Como se puede apreciar, los mensajes constan de tres campos, *codigo*, *valor* y *topic*. En el campo *codigo* se coloca un código único para cada sensor o comando. Los códigos usados se detallan en la Tabla 7.

Tabla 7 Código de sensores/comandos

Sensor/Comando	**Código**
Corriente	corr
Temperatura	temp
Presión	pres
Índice uv	uvix
GPS	gps
Temperatura del cpu	tcpu
Carga del cpu	cpuload
Estadísticas	stat
Captura de fotos	Cam
Captura de video	vid

En el campo *valor* se coloca el valor de la medición. Por otra parte, si se trata de un comando para encender o apagar el sensor, los valores serán “true” o “false” respectivamente. El campo *topic* es una denominación necesaria para colocar correctamente los datos en los cuadros.

Para el caso de los mensajes generados por el módulo GPS, se adicionan dos campos adicionales, *“lat”* y *“lon”* con la información de las coordenadas de la estación.

Por otro lado, es necesario tomar en cuenta que, al enviar un mensaje, Node-Red empaqueta todo bajo una etiqueta *“d”*, por lo que los mensajes tendrán una apariencia como la indicada a continuación.

{“d”:” {"codigo":"[codigo del sensor]","valor":"[]","topic":"treal"}”}

Dicha característica, debe considerarse para retirar este envolvente cuando el mensaje es recibido. Finalmente, es importante mencionar que los mensajes de los sensores se configuraron con el nivel de QoS más bajo, puesto que la pérdida de uno de estos mensajes no tiene una relevancia mayor (aunque esto podría depender de la aplicación). Mientras que los mensajes de comandos están configurados con el mayor nivel de QoS para asegurar su llegada al receptor, puesto que la pérdida de uno de estos mensajes puede ocasionar problemas en el funcionamiento del sistema.

4.4. Implementación de la Estación Prototipo

En esta sección se detallarán todos los procesos que realiza la estación prototipo como la integración de la cámara y los sensores, el desarrollo de los algoritmos de procesamiento de imágenes y la forma en la que se transmite las imágenes y el video a través de MQTT. Todos estos procesos están integrados por una aplicación en Node-Red de un solo flujo, la cual se puede observar en la figura 19. El funcionamiento de sus nodos se explicará a lo largo de esta sección.

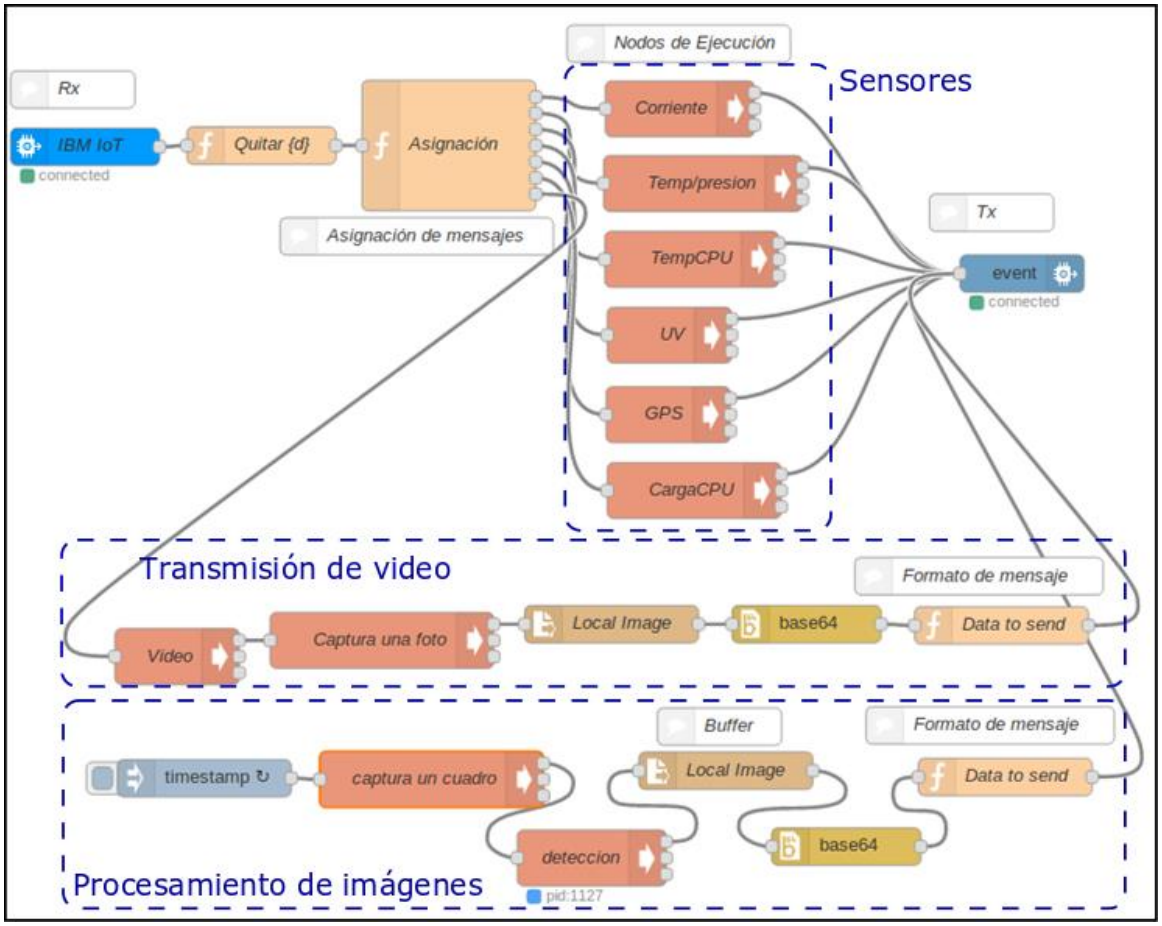

Figura 19 Aplicación Node-Red en la estación prototipo

4.4.1. Integración de la Web Cam

Para el control y uso de la *web cam* desde la aplicación en Node-RED, se utiliza un nodo de ejecución (*exec*) que realiza una llamada a la aplicación *fswebcam*. Dicha aplicación tiene la

capacidad de capturar fotografías y video especificando sus parámetros en la misma línea de comandos. En la Figura 20, se muestra el despliegue del nodo de ejecución que realiza la captura de una fotografía con una resolución de 544x280 pixeles y almacena el resultado en el fichero ./imagen.jpg.

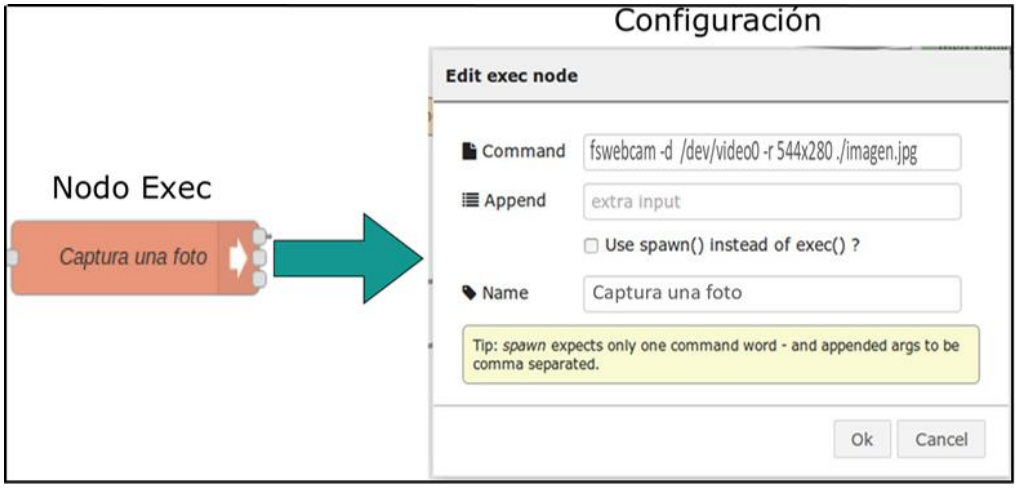

Figura 20 Nodo de ejecución para la cámara web

4.4.2. Procesamiento de Imágenes

El procesamiento de imágenes desarrollado en el presente trabajo de tesis, tiene como objetivo identificar frutos en fotografías de plantaciones en campo abierto y obtener información estadística de los mismos. Para cumplir con dicho objetivo, se emplearon los procedimientos expuestos en el apartado 2.4. Durante esta etapa, con el objetivo de analizar con mayor rapidez las diferentes variables involucradas en el procesamiento de las imágenes, los experimentos se realizaron mediante una computadora de escritorio. En cuanto a los experimentos con la estación prototipo, estos se presentarán posteriormente en el Capítulo 5. Cabe indicar, que en este caso la cámara utilizada para la adquisición de las imágenes, sobre las cuales se probaron los algoritmos, es ligeramente superior a la usada en la estación. Adicionalmente, también se emplearon fotografías obtenidas de bancos de imágenes libres y disponibles en [67], juntando en total un conjunto de diez fotografías con frutos rojos.

Para el caso del procesamiento de imágenes basado en GMM con EM, en primera instancia se experimentó el ajuste del modelo usando tres tipos de espacios de color, RGB, HSV y LAB. Para estos experimentos se utilizó una cantidad de 5 *clusters*, cantidad escogida sin ningún criterio previo.

En cuanto a la imagen seleccionada para ajustar el modelo, se empleó una fotografía con una resolución de 480 x 380 *pixeles*, la misma que se presenta en la Figura 21.

Figura 21 Imagen de muestra empleada para el ajuste del modelo mediante el esquema GMM

En la Figura 22(a), (b) y (c), se muestran los resultados obtenidos de la segmentación empleando 5 *clusters*, para los espacios de color, RGB, HSV y LAB respectivamente.

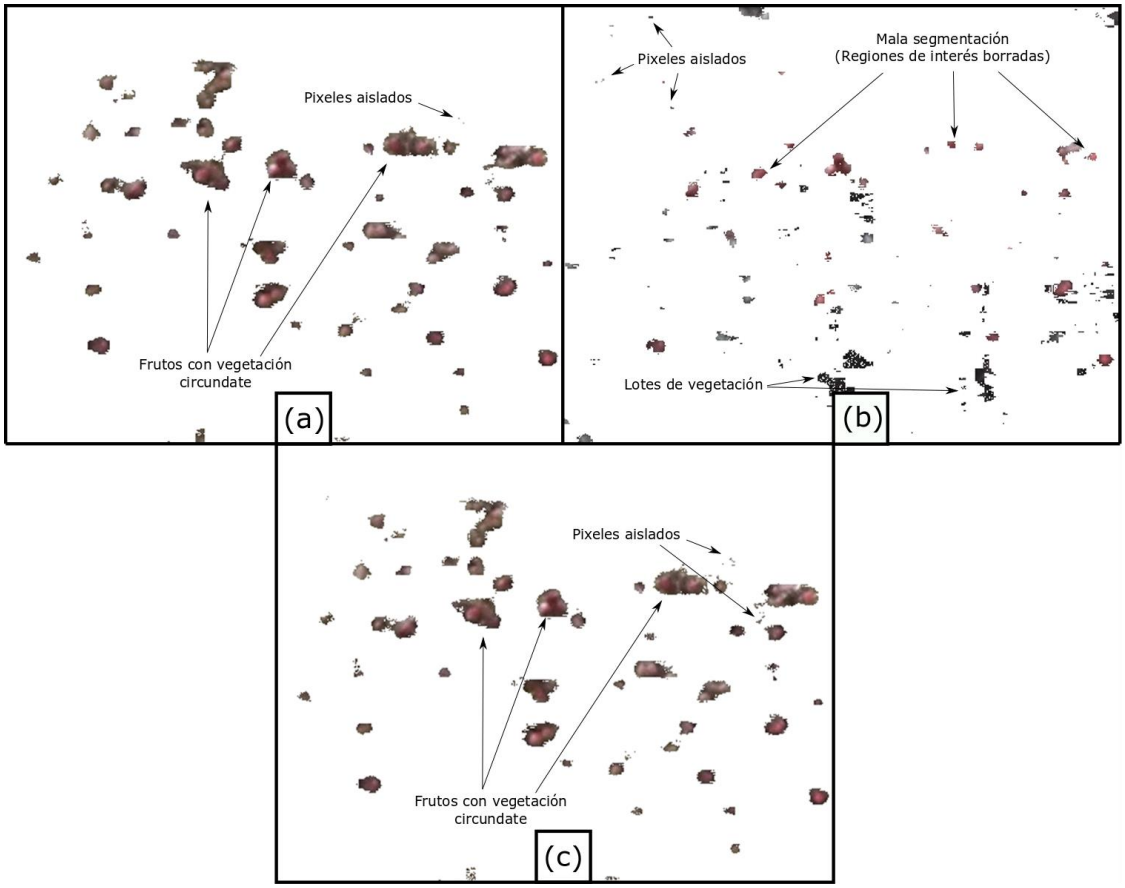

Figura 22 Resultados de la segmentación empleando el modelo GMM, con 5 *clusters* y tres tipos de espacios de color. (a) RGB. (b) HSV. (c) LAB

De estos experimentos se puede concluir que los resultados en RGB y LAB son bastante parecidos, y muy eficientes para la tarea de segmentación, mientras que el resultado en HSV dista mucho de los elementos que debieron ser detectados como frutos.

En tal sentido, en un intento de proveer al modelo información no solo de cada pixel, si no de lo que sucede en su contexto, se realizó dos experimentos adicionales, en los que además de la información provista por los tres canales del espacio de color, se añadió otras tres dimensiones al modelo GMM. En el primero de estos experimentos estas dimensiones fueron el promedio de los canales de los espacios de color de los píxeles circundantes. En el segundo experimento en lugar de hacerlo con el promedio, se añadió información de la varianza de los pixeles circundantes. Los resultados obtenidos de estos experimentos se muestran en la Figura 23 y 23 (b) respectivamente.

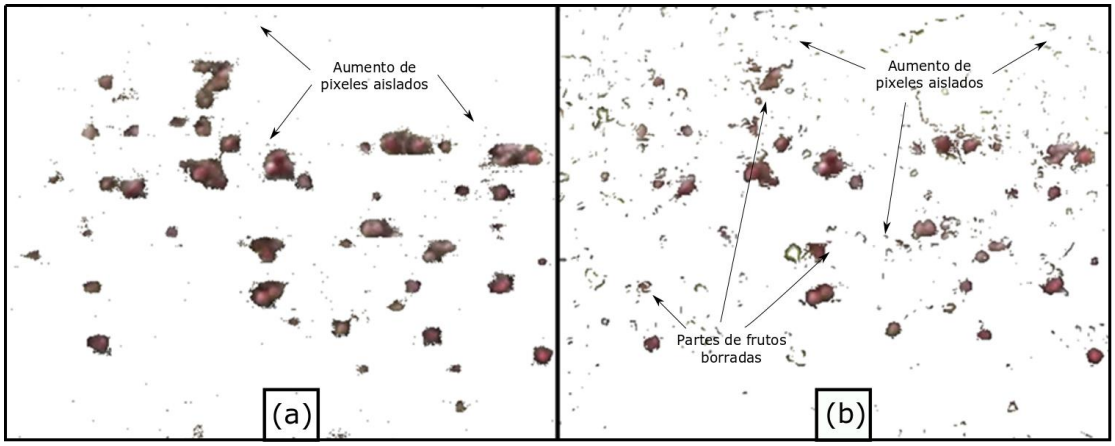

Figura 23 Resultados de la segmentación incluyendo información adicional en el modelo GMM. (a) Promediado de pixeles. (b) Varianza de pixeles

Como se puede apreciar en la Figura 23 (a), para el caso del experimento usando el promedio de los pixeles, se observa que, si bien la segmentación es buena, no mejora los resultados obtenidos al emplear solo los canales de color. Además, de que se añade una considerable cantidad de pixeles aislados. En cuanto a los resultados del segundo experimento indicados en la Figura 23 (b), la segmentación se torna muy deficiente. Estos dos modelos fueron descartados.

Hasta el momento se cuenta dos modelos candidatos, sin embargo, al analizar nuevamente los resultados de las Figuras 22(a) y 22 (c) correspondiente a los modelos en RGB y LAB con 5 *clusters* se detectan problemas semejantes. Es decir, en estos modelos los frutos aparecen rodeados de pixeles que no les pertenecen y que posiblemente están ahí por reflectancia de las hojas que los rodean. Además, existen ciertos píxeles aislados que no deberían ser parte del *cluster* de frutos. Estos problemas podrían deberse a que no se han creado la suficiente cantidad de *clusters* para separar a estos píxeles. Por consiguiente, se procedió a emplear el criterio BIC para estimar la cantidad de *clusters* adecuada que deberían utilizarse para estos modelos. En la Figura 24(a) y 24 (b), se presentan los resultados de aplicar el criterio BIC para los modelos en RGB y LAB respectivamente.

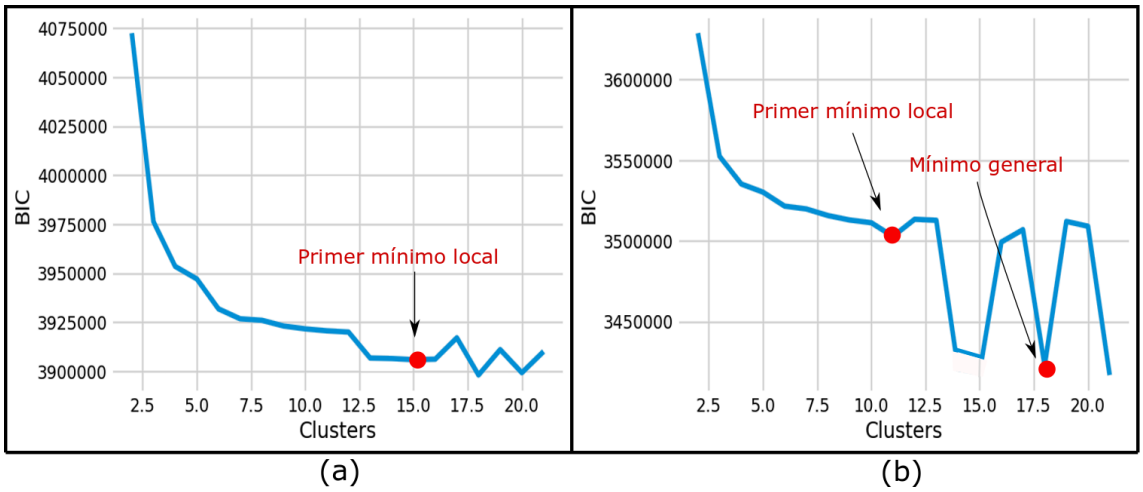

Figura 24 Resultados del criterio BIC para estimar el número de clusters. (a) RGB. (b) LAB

Como se puede apreciar, para el caso del espacio RGB, el primer mínimo local se encuentra en 15 *clusters* y el mínimo de todos los modelos analizados en 18 *clusters* con un valor muy parecido al modelo de 15 *clusters*, por lo que se lo descartará a este último. En el espacio LAB se tiene un primer mínimo local en el modelo de 11 *clusters* y un mínimo general en el modelo de 18 *clusters* con un valor de BIC considerablemente menor.

A continuación, se realizaron experimentos con estos 3 modelos considerados y sus resultados se exponen en las Figuras 25(a), 25(b) y 25(c) respectivamente.

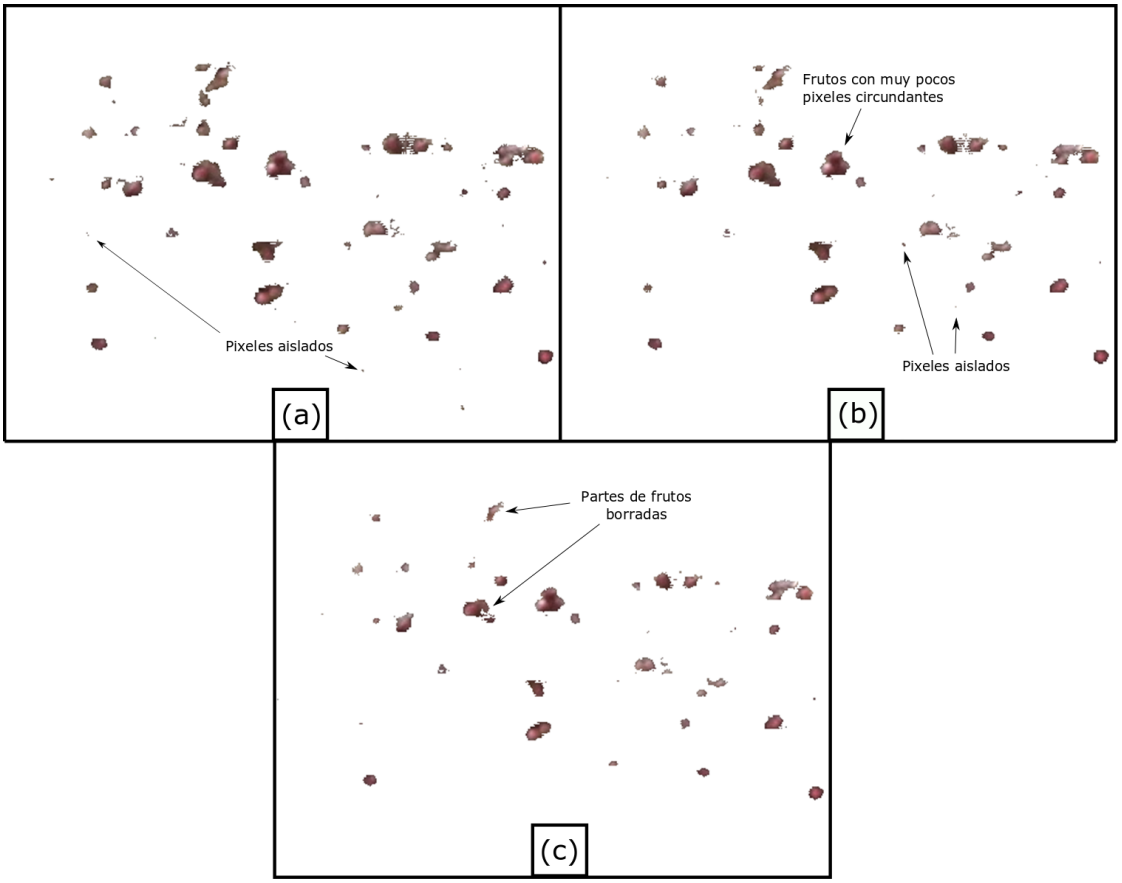

Figura 25 Resultados de la segmentación. (a) RGB, 15 clusters. (b) LAB 11 clusters. (c) LAB 18 clusters

De estos tres modelos, se observa que el modelo en espacio LAB con 11 *clusters*, Figura 25 (b), se ajusta mejor a los resultados esperados, puesto que tiene menor cantidad de píxeles no-frutos que rodean a los frutos (como sucede en el modelo RGB de 15 *clusters*) y no elimina pixeles pertenecientes a los frutos (como sucede en el modelo LAB de 18 *clusters*, lo cual podría indicar un sobre ajuste del modelo). Este modelo también tiene la ventaja de que requiere el menor número de *clusters* de los tres analizados, lo cual lo es una característica deseada en términos de carga de procesamiento. Una última ventaja de este modelo es que al encontrarse en espacio LAB será más inmune a cambios de luz en el ambiente.

A partir de los experimentos y los resultados obtenidos, se empleará el modelo LAB con 11 *clusters* para realizar la segmentación. En cuanto, a los inconvenientes que aún se observan, como pequeños píxeles aislados y puentes entre algunos frutos, se requiere un tratamiento adicional para eliminarlos. Específicamente, para eliminar estos píxeles se realizó una operación de *closing* sobre la imagen binaria empleando un elemento estructural con forma de disco de radio 3 pixeles. En la Figura 26, se muestran una comparación de los resultados antes y después de la operación de *closing*.

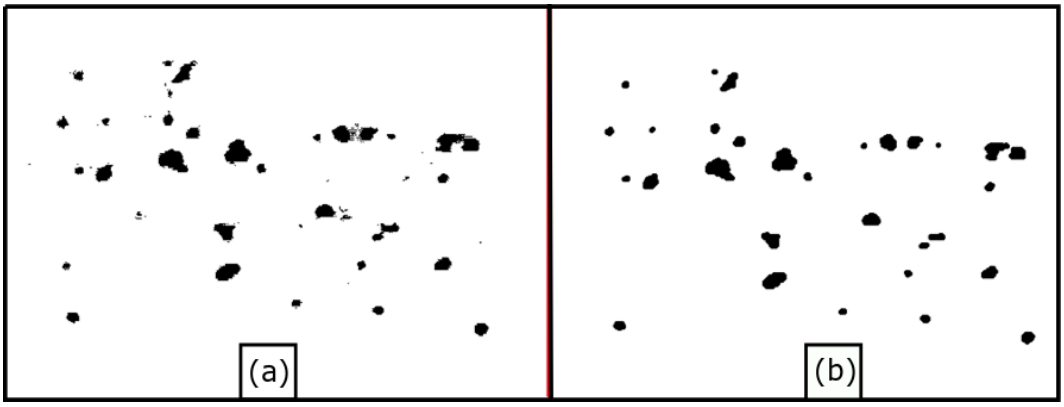

Figura 26 Imagen binaria. (a) Antes de la operación closing. (b) Después de la operación closing

Como se puede apreciar, los resultados son satisfactorios. Un efecto secundario de la operación *closing*, con el elemento estructural escogido, es el pulido de las manchas en formas más redondas, lo cual no aporta mayor problema. Otro inconveniente que tiene la aplicación de esta operación es el hecho de que puede eliminar frutos que tengan un tamaño similar al elemento estructural, provocando que inevitablemente los frutos muy pequeños no sean detectados.

Por último, se realizó un encuadre de los elementos ubicados para poder identificarlos con mayor claridad en su contexto. Los resultados de aplicar dicha operación se muestran en la Figura 27:

Figura 27 Imagen de muestra con elementos ubicados

A continuación, mediante un *script* implementado en Python se obtuvieron las estadísticas de los objetos encontrados en esta imagen. En la Figura 28, se presenta los resultados obtenidos

```
#######REPORTE DE ESTADÍSTICAS######
Número de manchas:                  26
Número estimado de frutos:          28
Color promedio de los frutos(BGR): [ 93  93 131]
Tamaño objeto mínimo(pixeles):      21
Tamaño objeto máximo(pixeles):      340
Tamaño objeto promedio(pixeles):    109
####################################
```

Figura 28 Estadísticas de la imagen de muestra

Los resultados obtenidos luego de la secuencia de experimentos desarrollados, son satisfactorios. En particular, se han encontrado prácticamente la totalidad de los frutos que se pueden detectar con la visión humana. Además, se destaca que no ha sido seleccionada ninguna zona que no corresponda a un fruto, por lo que ahora se puede actualizar el marco de trabajo por uno más específico, modificando en este caso el primer y quinto bloque, que será usado para la predicción de las imágenes que validarán el modelo. Este marco de trabajo actualizado se muestra en la Figura 29. En cuanto a la implementación en código, en el Apéndice H, se presenta el *script* desarrollado en Python.

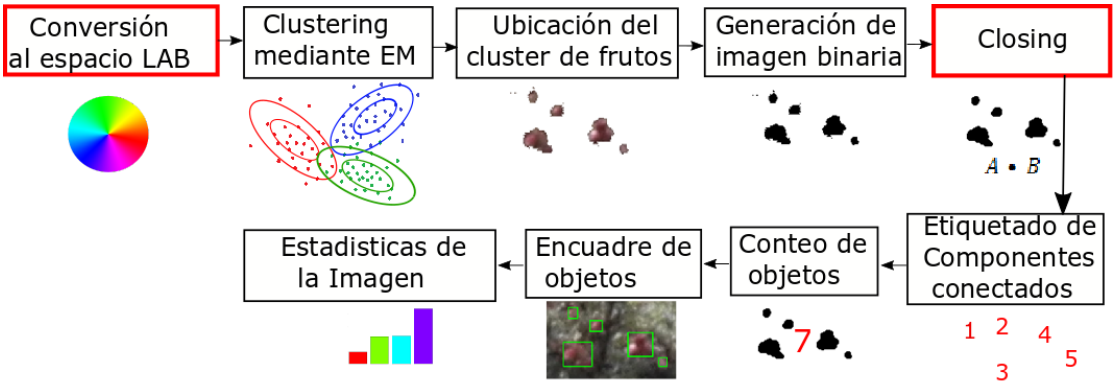

Figura 29 Marco de trabajo actualizado, incluyendo el espacio de color LAB y la operación closing

Por otra parte, para verificar si el modelo definido, servirá para segmentar otras imágenes diferentes a la empleada para el ajuste, es necesario realizar una tarea de validación, probando dicho modelo con imágenes distintas, pero dentro del mismo contexto, es decir mismos frutos, fondo similar y niveles similares de iluminación.

Para llevar a cabo dicha validación, se utilizó dos imágenes tomadas cerca de la ubicación donde fue capturada la imagen empleada en la etapa de ajuste, asegurando de esta forma, que tendrán el mismo contexto. Además, se incluyeron dos imágenes adicionales en contextos distintos, esto con el objetivo de forzar las capacidades del modelo. Estas imágenes se muestran en la Figura 30.

Figura 30 Imágenes empleadas para la validación del modelo. (a) y (b) Contexto similar. (c) y (d) Contexto distinto

La Figura 30(a) y la Figura 30(b) muestran las dos imágenes en un contexto similar a la imagen de ajuste. En particular, los frutos son los mismos (manzanas) y la vegetación e iluminación son similares. Por otra parte, en la Figura 30(c), se muestra una imagen con otros frutos (tomate de árbol) y diferente nivel de iluminación y vegetación. Sin embargo, los frutos son de un color semejante a los frutos de la imagen de ajuste. Finalmente, la Figura 30(d), de igual forma consiste en una imagen en otro contexto, con frutos diferentes (tomates) y con un color semejante a los frutos de la imagen de ajuste aunque con un mayor brillo. Al ejecutar el procesamiento indicado por el marco de trabajo desarrollado (Figura 29), se obtienen los resultados mostrados en la Figura 31.

La Figura 31(a), muestra excelentes resultados puesto que han sido localizados casi la totalidad de los frutos, sin contabilizar ningún falso positivo. En la Figura 31(b), se han localizado todos los frutos, pero también cuenta con algunos falsos positivos que deberán tomarse en cuenta al probar la estación en el campo. La Figura 31(c), aunque está en otro contexto ha localizado casi todos los frutos presentes, pero el número de falsos positivos también es considerable.

Finalmente, la Figura 31(d), aun pese a estar en otro contexto, muestra un gran número de elementos ubicados, sin contabilizar falsos positivos.

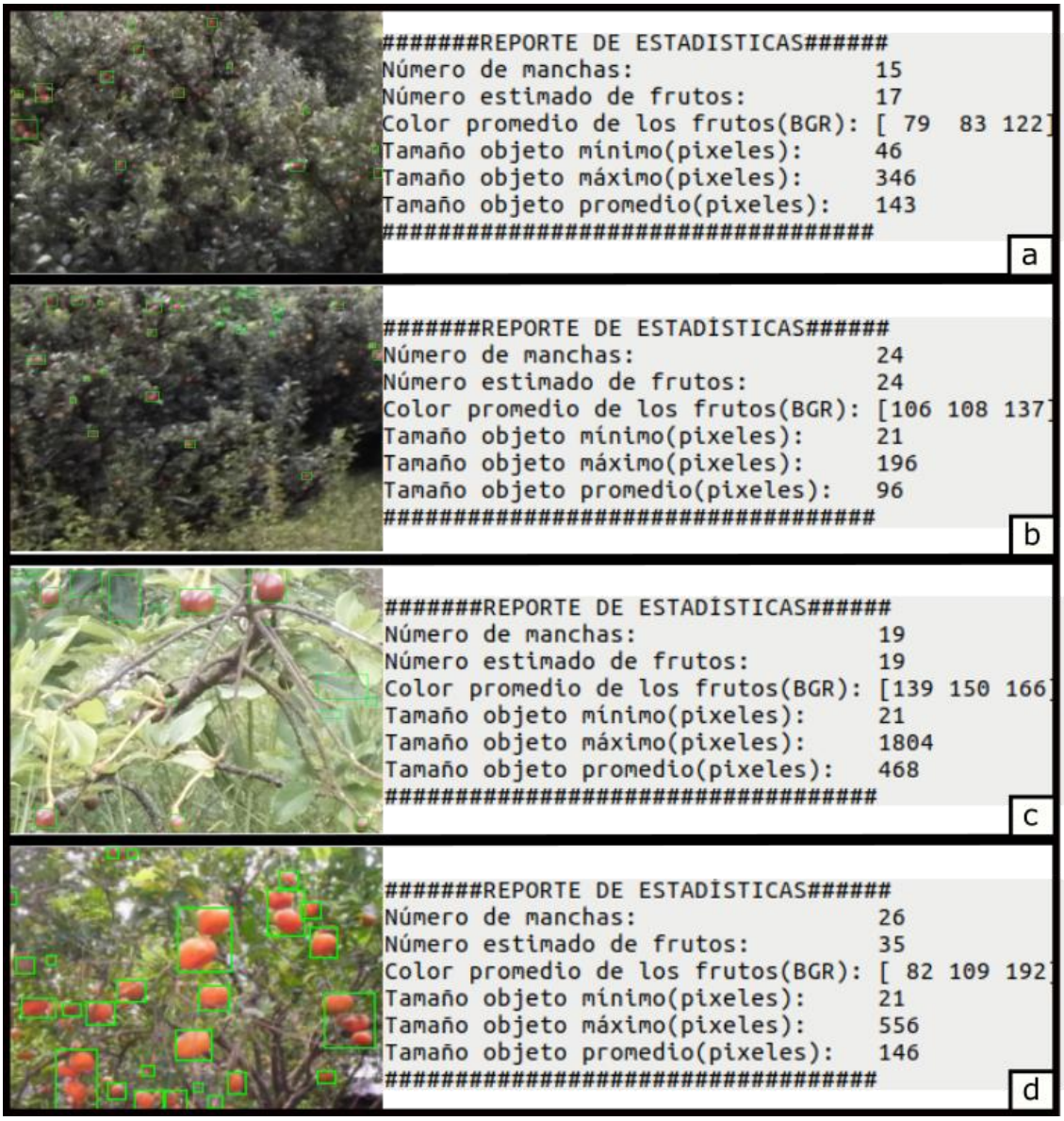

Figura 31 Imágenes de validación procesadas. (a) y (b) Contexto similar. (c) y (d) Contexto distinto

Los resultados obtenidos luego de la validación del modelo son muy satisfactorios por lo que se podrá utilizar este esquema de visión artificial en campo, cuyos resultados se mostrarán en el Capítulo 5.

Para el caso del marco de trabajo basado en *Thresholding* el procedimiento fue bastante sencillo, en primera instancia se decidió crear un umbral que abarque el área total de todos los frutos de las imágenes del conjunto usado. En cuanto a las operaciones morfológicas se decidió usar solamente la operación de *closing* con un disco de radio igual a 3 pixeles, como elemento estructural. Los valores de umbral usado en este algoritmo se especifican en la Tabla 8, mientras que los resultados de este algoritmo con un *Thresholding* amplio, se muestran en la Figura 32.

Tabla 8 Valores de Thresholding amplio

Valores HSV de Umbral Thresholding Amplio		
Canal	**Mínimo**	**Máximo**
HUE	-100°	60°
Saturation	35	255
Value	70	255

Figura 32 Resultados obtenidos con un Thresholding amplio en varias imágenes.

En la Figura 32(a) el resultado es muy bueno puesto que se han encuadrado todos los frutos presentes en la imagen y la cantidad de falsos positivos es baja. Por otro lado en la Figura 32(b) y en la Figura 32 (c) la cantidad de falsos positivos es significativa, motivo por el cual no se realizaron más pruebas y se decidió descartar este modelo.

A continuación, se realizó otro experimento con un umbral más reducido descartando matices rojizos tenues para así reducir la cantidad de falsos positivos. Los valores de umbral en este

algoritmo de *Thresholding* reducido se muestran en la Tabla 9 y los resultados obtenidos se presentan en la Figura 33:

Tabla 9 Valores Thresholding reducido

Valores HSV de umbral thresholding reducido		
Canal	**Mínimo**	**Máximo**
HUE	-80°	20°
Saturation	65	255
Value	70	255

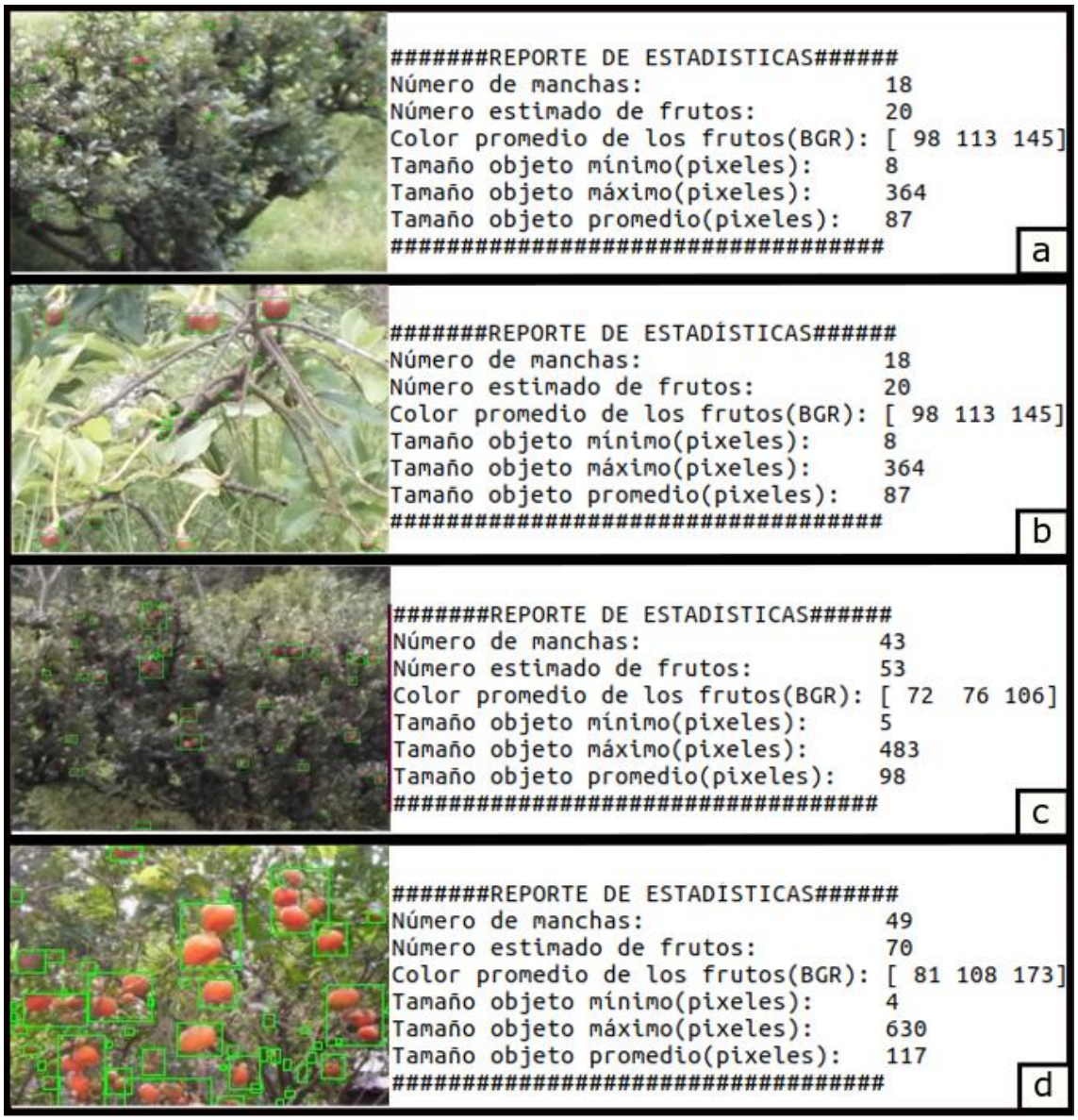

Figura 33 Resultados obtenidos mediante Thresholding reducido en varias imágenes.

Los resultados con este algoritmo con un umbral reducido muestran una gran reducción en la cantidad de falsos positivos, aunque aún existe una considerable cantidad de ramas que son contadas como frutos tal como se observa en las Figuras 33(b) y 33(d). Otro problema que tiene este modelo se lo puede observar realizando un acercamiento a la Figura 33(b) y extrayendo la parte segmentada como se lo observa en la Figura 34. Como se puede apreciar, una gran parte del fruto no ha sido segmentada, lo cual podría ocasionar que ciertos frutos no sean detectados en determinadas fotografías.

Figura 34 Acercamiento Figura 21(b) con extracción de segmentación

Pese a estos problemas, al tener en cuenta las ventajas de bajo coste de procesamiento de este algoritmo, los resultados son satisfactorios y podría ser considerado en la implementación de la estación. El código fuente del script en Python que se utilizó para esta segmentación se encuentra en el Apéndice G.

4.4.3. Transmisión de Imágenes y Video en Tiempo Real

La transmisión de video en tiempo real a través del protocolo MQTT está fuertemente limitada por el máximo tamaño de los paquetes que admite el *broker* utilizado, que en este caso es de 128kB, dificultando el envío de los segmentos de video. Además, el hecho de que MQTT sea un protocolo asíncrono añade retraso al video transmitido. Pese a estas restricciones se implementó un algoritmo básico para la transmisión de video mediante MQTT.

De manera general este algoritmo al recibir una señal de la aplicación web, captura un *frame* a la vez, para luego enviarlo por MQTT a la nube de IBM. Para esto, en primer lugar, se utilizó una cantidad muy baja de cuadros por segundo, siendo esta de 1 fps. Luego se configuró el tamaño de los cuadros en 272x140 usando el formato jpg. Posteriormente, se realizó una verificación del tamaño de los cuadros, esto es muy importante puesto que cuando se intenta enviar un mensaje mayor a 128kB, el nodo de envío de mensajes MQTT se colapsa, incrementando el retraso en la transmisión de video y ocasionando problemas en el sistema en general, ya que todos los mensajes se envían por el mismo nodo.

Una vez capturada la fotografía es necesario guardarla en un *buffer*, convertirla a base 64 y finalmente colocar el encabezado con el código correspondiente. Se configuró el QoS en su nivel más bajo, ya que, al tratarse de video en vivo, los cuadros que lleguen con desorden no son de utilidad. El diagrama de flujo correspondiente a la programación indicada, se presenta en la Figura 35.

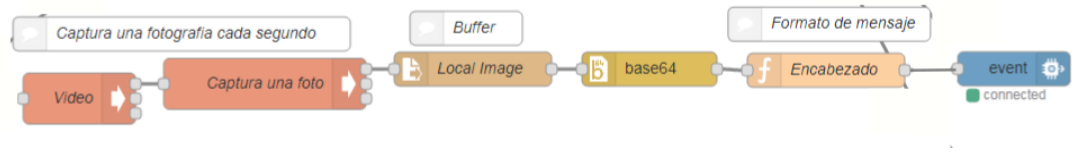

Figura 35 Diagrama de flujo en Node-Red, para la transmisión de video

En cuanto a la transmisión de imágenes, el procedimiento es muy similar al del video, con la diferencia de que no se espera una señal desde la aplicación web para iniciar su funcionamiento. En este caso se realizó la configuración para la captura automática de una fotografía al día, la misma que posteriormente es procesada para la detección de frutos y obtención de las estadísticas. Luego de esto se procede de igual manera que en la transmisión de video, tal como se muestra en la Figura 36.

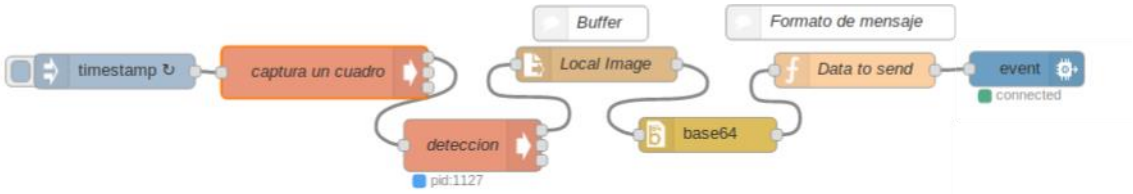

Figura 36 Diagrama de flujo en Node-red para la transmisión de imágenes

4.4.4. Funcionamiento de los Sensores

Además de los sensores indicados en el apartado 4.1.3, (Temperatura, presión, corriente, índice UV y GPS), se incluyeron las lecturas de dos sensores internos de la Raspberry Pi, la temperatura del procesador y la carga del procesador. Esto se realizó con la finalidad de verificar el estado de la plataforma durante el funcionamiento de la estación.

Todos los sensores funcionarán de manera similar, esto es, cuando se reciba un mensaje desde la *IBM Cloud* destinado a prender o apagar un sensor, la aplicación en Node-Red evaluará el mensaje leyendo el campo "*codigo*" y los asignará al nodo de ejecución del sensor que corresponda para encenderlo o apagarlo según el campo "*valor*" lo indique. Si el mensaje es de apagado (*false*) se envía una señal *kill* al script que controla al sensor para detener su ejecución. Si, por el contrario, el mensaje es de encendido (*true*), el nodo de ejecución en Node-Red dará inicio a un *script* en Python o Bash. Dicho *script* tiene como finalidad realizar la lectura del sensor, escribir dicha lectura en un archivo de texto y finalmente conformar un mensaje JSON para que posteriormente sea enviado a la aplicación web para su visualización.

Este funcionamiento se lo puede observar analizando el flujo de Node-Red presentado en la Figura 37 y en el diagrama de flujo presentado en la Figura 38.

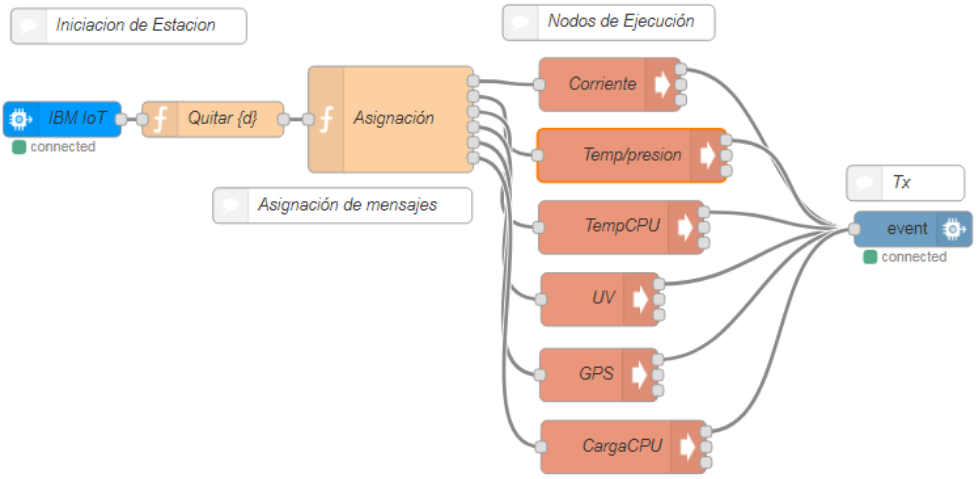

Figura 37 Flujo Node-Red para el funcionamiento de los sensores

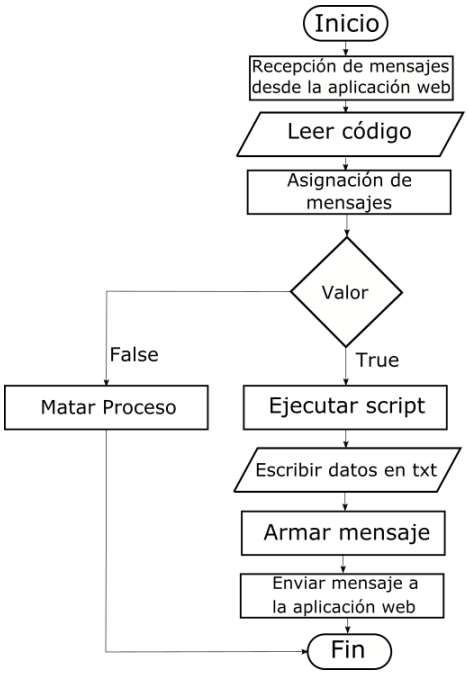

Figura 38 Diagrama de flujo correspondiente al funcionamiento de los sensores

La instalación de las librerías de los sensores de adquisición de datos y geo-localización, así como algunos detalles necesarios para su funcionamiento e implementación los *scripts* se presentan en el Apéndice E.

4.5. Implementación de la Aplicación Web

En la presente sección se detallará el funcionamiento de la aplicación web, con una explicación de los nodos usados en la programación y mostrando el diseño y configuración de la interfaz gráfica.

4.5.1. Programación de la Aplicación

La programación en Node-Red de la aplicación web se la realizó en dos flujos, uno destinado a la presentación de los datos provenientes de la estación y otro que contiene los controles que

envían comandos a la estación para realizar peticiones de datos a los sensores. El flujo de petición de datos se muestra en la Figura 39.

Este flujo se encarga en primer lugar de recibir los datos de los sensores, la cámara y las estadísticas resultado del procesamiento de imágenes. Luego, mediante un nodo de función se asigna cada uno de estos datos, a los cuadros de visualización que correspondan. Estos cuadros de visualización para el caso de los datos de los sensores se tratan de gráficos cartesianos. Para el mapa es una plantilla web que muestra un mapa proveniente de OpenStreetMap. Para el caso de las imágenes y video son plantillas web capaces de visualizar imágenes y para las estadísticas son cuadros de texto. Además de esto, se cuenta con el nodo del botón borrar que permite limpiar los gráficos en los cuadros de datos de los sensores. Finalmente, la región con el comentario "Despliegue del Mapa en Dashboard" realiza la configuración de la plantilla web en la que se visualiza el mapa.

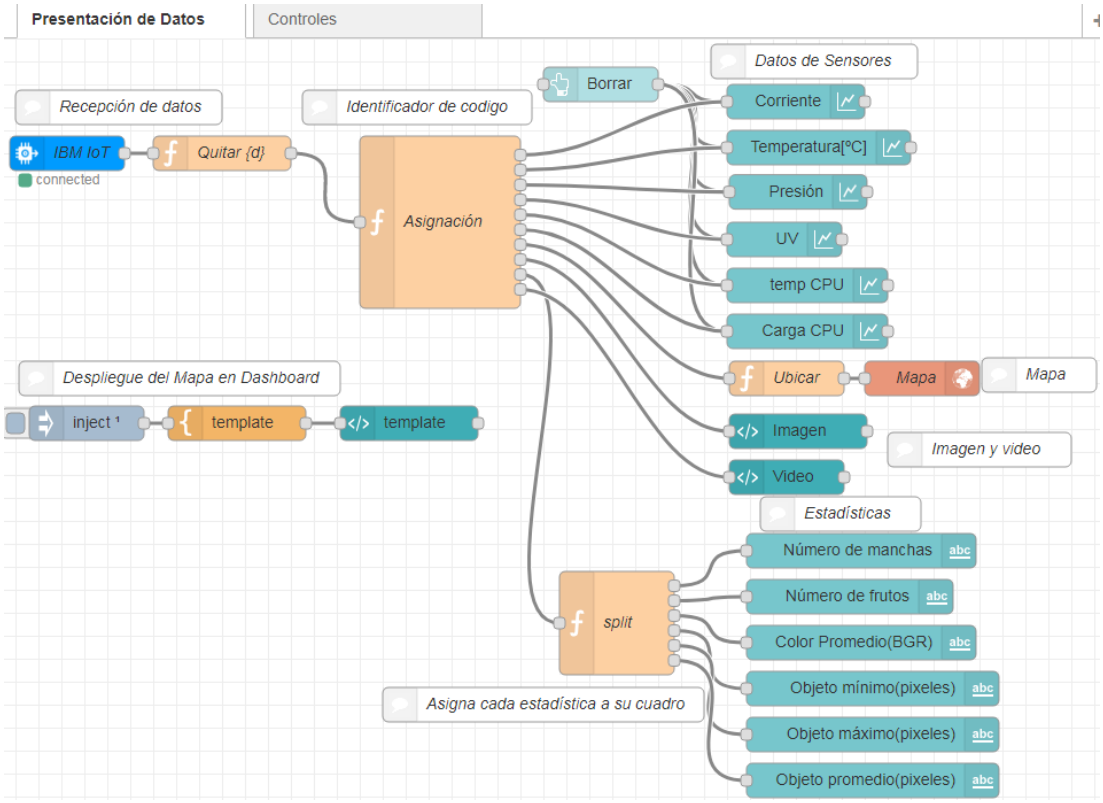

Figura 39 Flujo en Node-Red para la Presentación de Datos

El flujo de los Controles se muestra en la Figura 40. En este flujo se implementa los interruptores que permiten activar/desactivar los sensores, el botón que envía una petición al GPS para ubicar la estación, así como el interruptor que activa/desactiva la transmisión de video. Todos estos nodos se encuentran conectados a un nodo *wiotp out* que envía esta información a la estación mediante el protocolo MQTT.

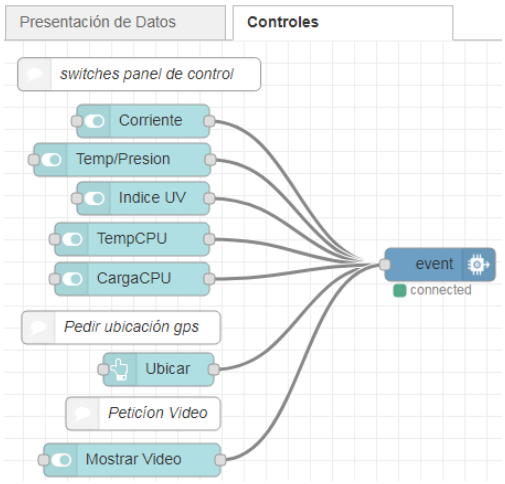

Figura 40 Flujo de Controles implementado en Node-Red

4.5.2. Configuración y Diseño de la Interfaz Gráfica

Para poder crear la interfaz gráfica de la aplicación web es necesario instalar la librería *node-red-dashboard*. Dicha librería, provee de un conjunto de nodos para implementar un panel de control dentro de una página web. El URL por defecto de la página es *[URL del editor Node-RED]/ui*. Para el caso de tener múltiples pestañas sus URLs *serán [URL del editor Node-RED]/ui/#!/n* donde *n=0,1,2…* es el número de pestaña.

La configuración de la interfaz gráfica se la realiza sobre la pestaña *dashboard*, ubicada en el panel derecho del editor. Aquí se pueden crear pestañas que aparecerán en el menú de la interfaz. Además, la información de cada pestaña puede ser organizada en grupos y se puede modificar el tamaño y la disposición de cada uno de los elementos. En la Figura 41 se muestra las pestañas creadas con sus respectivos grupos.

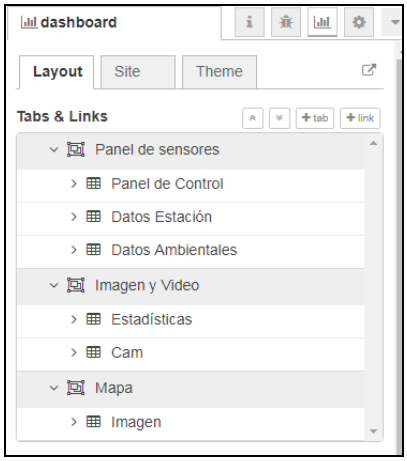

Figura 41 Panel de configuración *dashboard*

Se crearon tres pestañas, una para mostrar un panel con los datos provenientes de los sensores en la estación, otro para mostrar las imágenes provenientes de la cámara en la estación y la última para mostrar la ubicación de la estación en un mapa con la información del GPS emplazado en la estación.

En la Figura 42, se muestra el diseño de la pestaña del panel de sensores. En la Figura 43 se observa el diseño de la pestaña de imagen y video y en la Figura 44 se muestra la pestaña para la visualización del mapa.

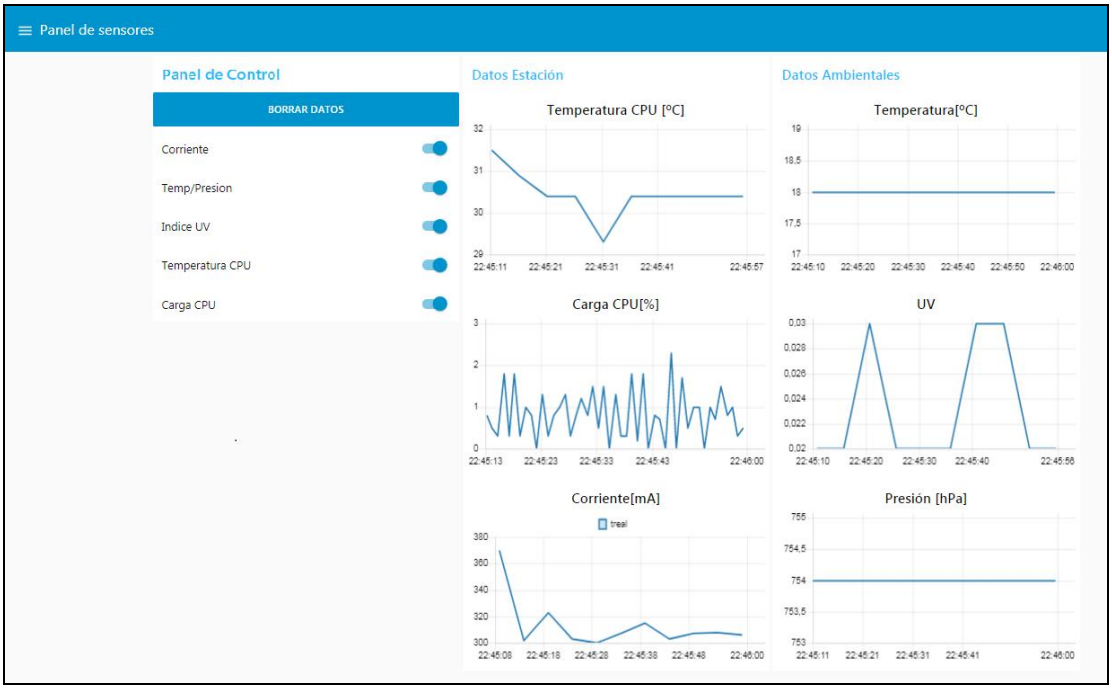

Figura 42 Pestaña Panel de sensores

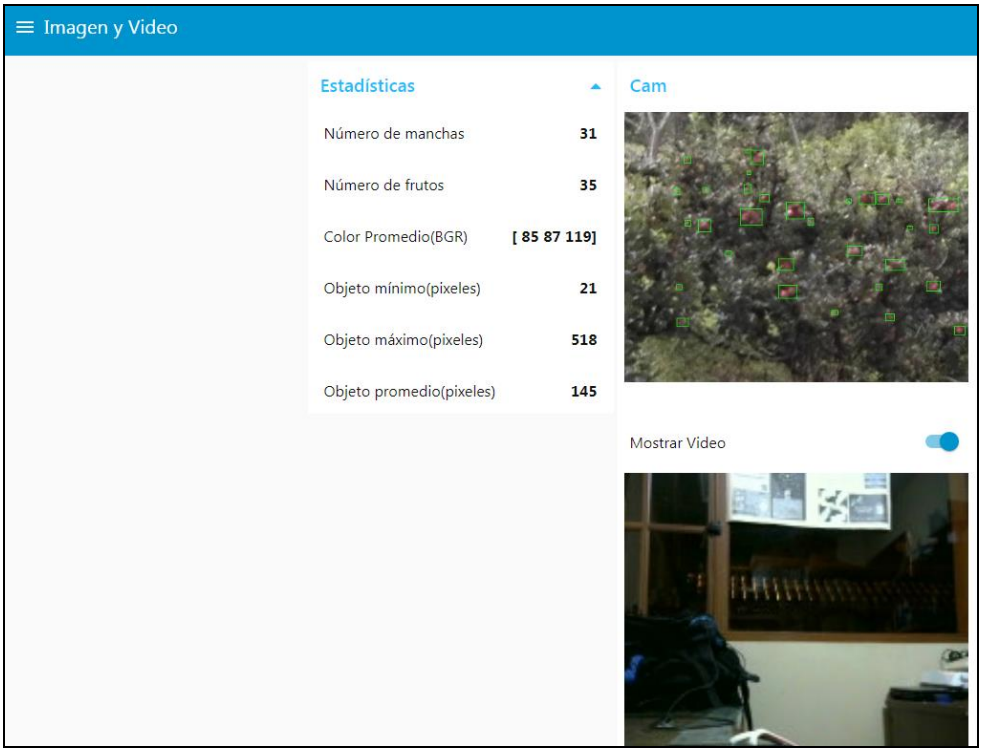

Figura 43 Pestaña Imagen y Video

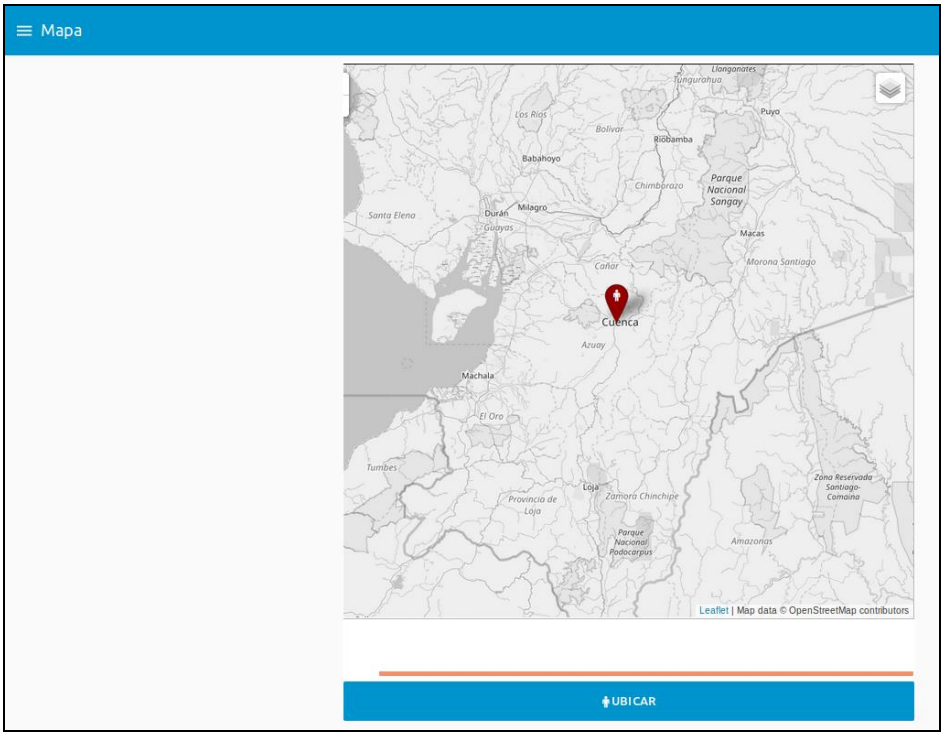

Figura 44 Pestaña para la visualización del Mapa

La pestaña del panel de sensores tiene tres grupos: el panel de control, los datos de la estación y los datos ambientales. El panel de control tiene la función de activar o desactivar los sensores, además de borrar los datos mostrados en los cuadros de los sensores. Los datos de la estación muestran los datos provenientes de los sensores de corriente, temperatura del procesador y carga del procesador. El grupo de datos ambientales muestra los datos provenientes de los sensores de temperatura, índice UV y presión atmosférica.

La pestaña Imagen y Video contiene dos grupos, Estadísticas y cámara (Cam). El grupo de estadísticas contiene los datos estadísticos provenientes del procesamiento de imágenes de campos frutales. Dichas estadísticas consisten en el número de manchas, el número estimado de frutos, el color promedio, y las dimensiones de los objetos con los valores mínimo, promedio y máximo, especificados en pixeles. El grupo cámara, contiene la imagen procesada con un encuadre de los frutos detectados, un control para activar/desactivar la transmisión de video y un cuadro que muestra el video transmitido en tiempo real.

La pestaña mapa muestra un solo grupo que contiene una plantilla que muestra un mapa y un botón que realiza una petición al sensor de geo-localización para mostrar la ubicación de la estación en el mapa.

4.6. Conclusiones

En este capítulo se expusieron todos los detalles respecto a la implementación de la estación y la aplicación web, comenzando con una descripción del hardware y el software utilizado junto con las configuraciones necesarias para la comunicación entre los dos extremos del sistema, para posteriormente desarrollar dos algoritmos de procesamiento de imágenes para el

problema objetivo de la tesis. Por último, se detalló la programación tanto de la estación como de la aplicación web.

5. EVALUACIÓN Y PRUEBAS EXPERIMENTALES

5.1. Introducción

En el presente capítulo se mostrarán los resultados de las pruebas realizadas empleando la estación prototipo. Estas pruebas consisten en el procesamiento de imágenes capturadas con la cámara web de la estación y la transmisión de los datos de los sensores y video a través de Internet con el protocolo MQTT. Además, se caracterizará el consumo energético de la estación considerando las diferentes funcionalidades disponibles.

5.2. Escenarios de Pruebas

Para las pruebas de procesamiento de imágenes se capturaron fotografías con la cámara web instalada en la estación. Dicha cámara permite capturar imágenes con una calidad ligeramente menor a la empleada para el desarrollo de los algoritmos en el Capítulo 4. En este caso, las fotografías tienen un tamaño de 544x288 pixeles y fueron tomadas en la granja "El Romeral", propiedad de la Universidad de Cuenca, ubicada en el kilómetro 10 de la vía Paute-Guachapala, que cuenta con gran cantidad de plantaciones frutales y de hortalizas. Las fotografías fueron tomadas a las tres de la tarde con un cielo parcialmente nublado, datos importantes para tener una idea de la iluminación presente en ese instante. Se decidió restringir las fotografías a un solo tipo de frutos que fueron manzanas, por tratarse de un fruto que resalta del fondo en el que se encuentra y que al momento de tomarse las fotografías estaban en etapa de cosecha por lo cual eran además bastante abundantes.

La transmisión de los datos de los sensores y la cámara no se la realizó en el mismo sitio por dificultades del acceso a Internet. Estas pruebas se las realizó en un ambiente controlado con la estación conectada a Internet mediante Wifi usando la batería de la misma estación como fuente de energía.

5.3.Procesamiento y Análisis de Imágenes

Para el procesamiento de las imágenes capturadas se usó tanto el algoritmo de *Thresholding* reducido como el algoritmo de GMM con EM. En primer lugar, se expondrán los resultados con el uso del algoritmo de *Thresholding* reducido. Este algoritmo es casi idéntico al desarrollado en el Capítulo 4, con la única diferencia que el elemento estructural utilizado para la operación de *closing* es un disco de radio 2 pixeles en lugar de 3. Los resultados obtenidos se detallan en la Figura 45.

En general los resultados son buenos puesto que la mayoría de los frutos han sido detectados, sin embargo en la Figura 45(b) y 45(d) se observan ramas o lotes de suelo que han sido identificadas como frutos, y en la Figura 45(e) existen ciertos frutos con coloración más tenue que no han sido detectados.

Además de esto, si se realiza un acercamiento a ciertas zonas de las imágenes extrayendo la parte segmentada como frutos, tal como se lo observa en la Figura 46, se aprecia que no toda el área de los frutos ha sido segmentada, evitando una correcta extracción de las características de los frutos. Aun así, se recalca que este algoritmo tiene la ventaja de un bajo

coste de procesamiento, notándose que el tiempo en procesar cada imagen en la Raspberry Pi fue de aproximadamente 30 segundos.

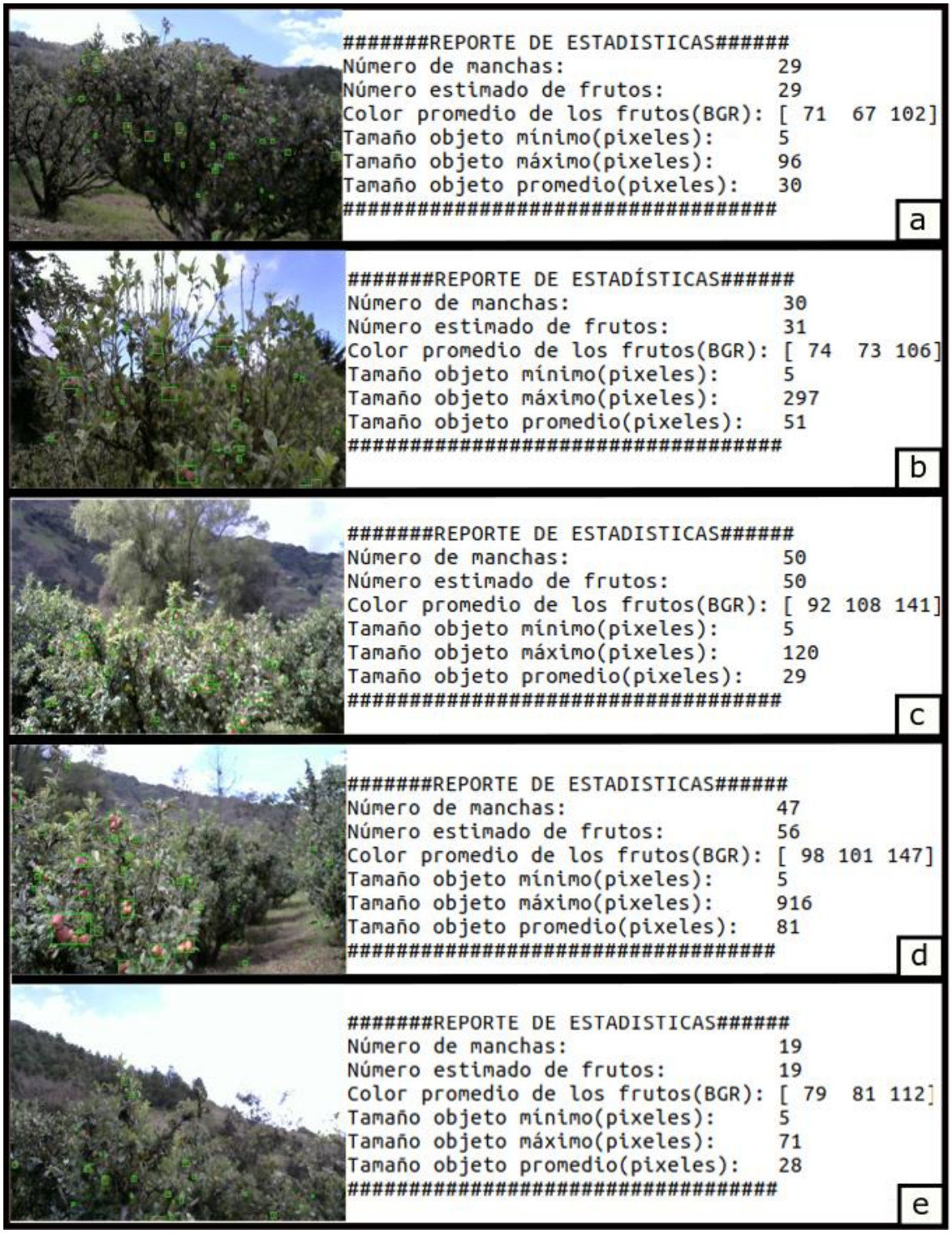

Figura 45 Resultados de detección de frutos con Thresholding reducido en varias imágenes

Figura 46 Acercamiento con segmentación extraída

En cuanto al algoritmo de GMM con EM, se aplicó la misma solución de desarrollada en el capítulo anterior, es decir con 11 *clusters* para la segmentación, aunque reduciendo en este caso el elemento estructural de la operación *closing* a un disco de radio 2 pixeles.

Se aplicó el esquema clásico para el uso de estos modelos, en el cual se realiza el ajuste con una imagen de prueba para posteriormente usarlo con todas las demás fotografías. No obstante, se presentó la dificultad de encontrar una imagen que dé buenos resultados para la detección de frutos en el resto de fotografías, tal como se lo observa en la Figura 47, en la cual se ven 2 imágenes de ajuste junto a 2 imágenes de prueba con resultados muy deficientes puesto que se pueden observar lotes de cielo y suelo identificados como frutos.

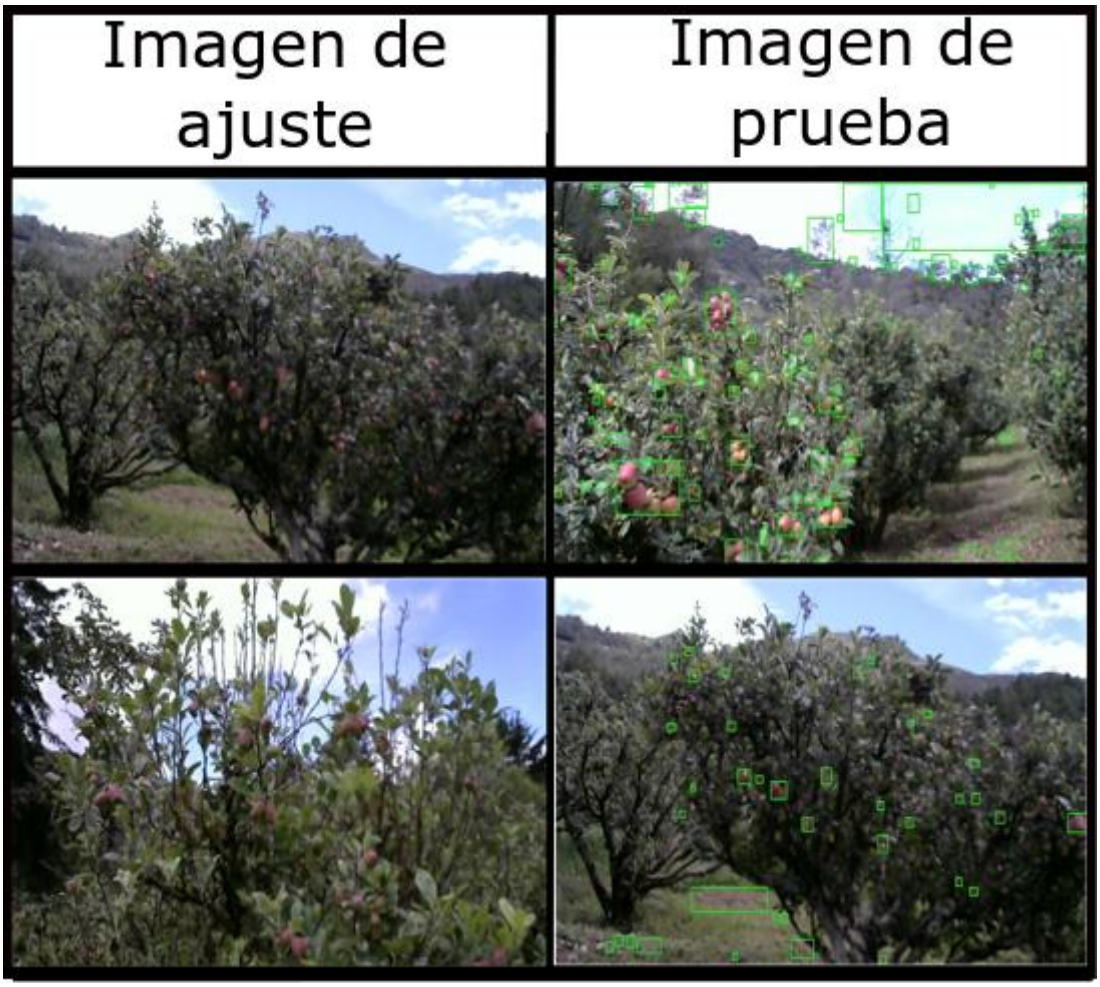

Figura 47 Resultados fallidos de GMM con EM

Finalmente, se realizó un experimento usando la Figura 48 como imagen de ajuste obteniéndose en este caso buenos resultados, los cuales se los puede observar en la Figura 49. Dichos resultados, son mejores a los obtenidos con el algoritmo de *Thresholding* puesto que se observa menor cantidad de falsos positivos junto con una mejor segmentación. Sin embargo, el coste de procesamiento es mucho más alto, teniendo un tiempo de aproximadamente 21 minutos en procesar una imagen en la Raspberry Pi.

Figura 48 Imagen de ajuste seleccionada para la segmentación con el modelo GMM-EM

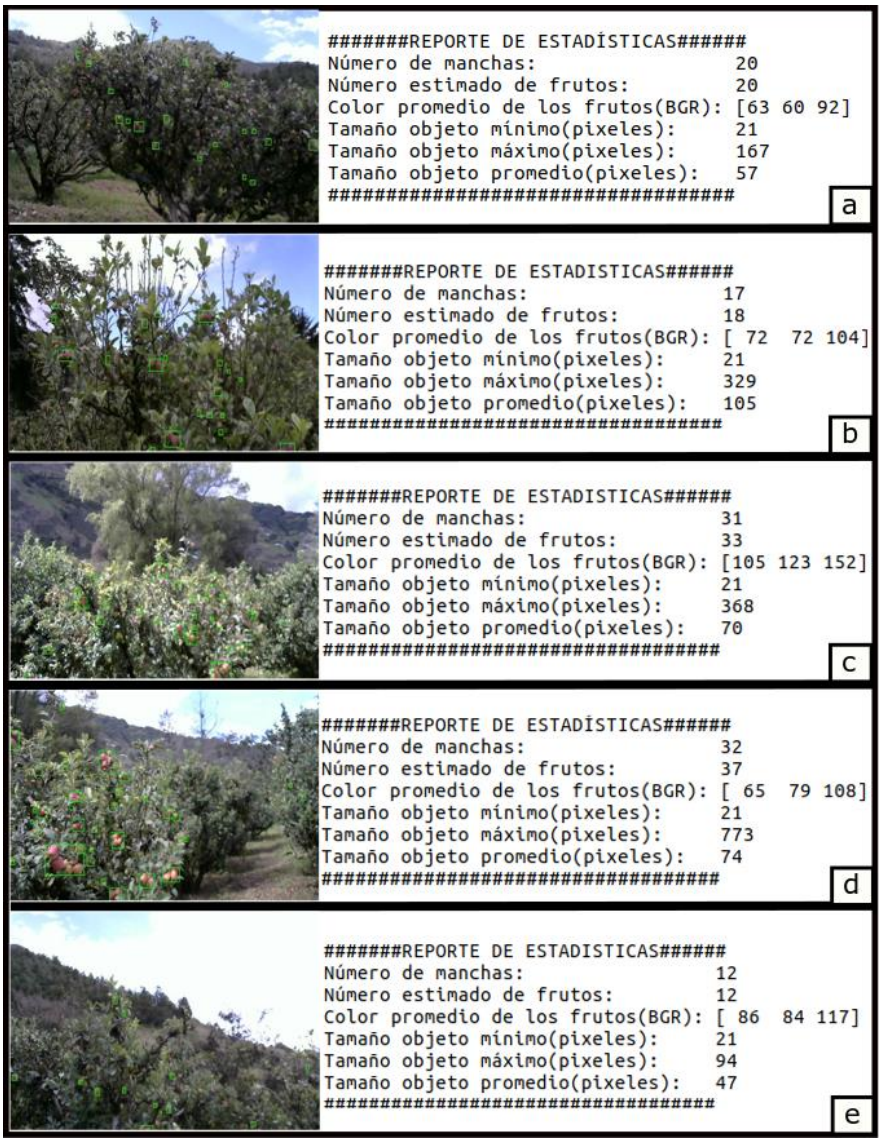

Figura 49 Resultados de detección de frutos con GMM mediante EM en varias imágenes

5.4. Transmisión de Datos de Sensores y Video

Para probar la efectividad de la transmisión de mensajes, mediante el protocolo MQTT, se diseñó un experimento que simula el envío simultáneo de datos desde seis nodos en intervalos de 30 segundos, empleando el nivel más bajo de QoS. El experimento se desarrolló hasta completar 100 mensajes por nodo, es decir un total de 600 mensajes en toda la simulación. Del lado del receptor en la aplicación web, se implementó un contador de mensajes. En la Figura 50 se muestra la programación realizada en Node-Red para dicha simulación.

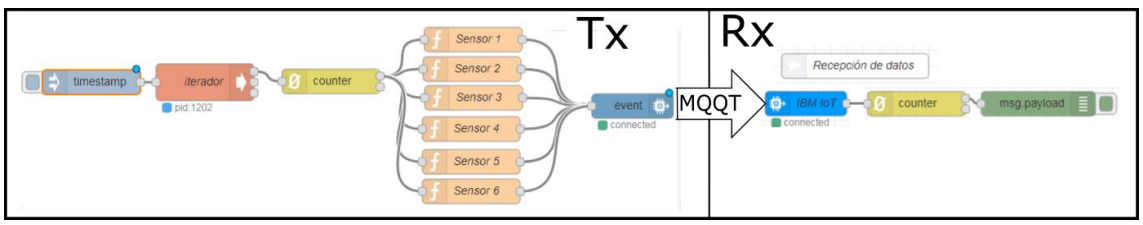

Figura 50 Simulación de transmisión de datos de sensores

Aunque el nivel más bajo de QoS en MQTT no garantiza la transmisión de los mensajes, al terminar la simulación se obtuvo que la totalidad de los mensajes fueron recibidos con éxito, lo cual prueba la efectividad de este protocolo y de la plataforma IoT de IBM como *broker* MQTT. En la Figura 51 se muestra el panel *debug* con los últimos mensajes que llegaron en la simulación.

Figura 51 Panel debug con los resultados de la simulación

En cuanto a la transmisión de video, pese a las limitaciones del protocolo MQTT analizadas en el apartado 4.4.3, durante los experimentos se consiguió una transmisión de video satisfactoria, con retrasos no muy marcados de entre 1 a 4 segundos. Dicho retardo, consideramos resulta admisible para la funcionalidad de video vigilancia que se desea incluir en la estación prototipo.

5.5. Caracterización del Consumo Energético

Mediante el uso del sensor de corriente, se realizaron experimentos para caracterizar el consumo de energía que demanda la estación prototipo, según la funcionalidad que se encuentre activa. Con tal objetivo, se realizó un muestreo de la corriente cada 5 segundos, durante distintos intervalos de tiempo. Finalmente, los resultados de dichos intervalos fueron promediados para obtener una estimación. En primer lugar, se midió la corriente que consume la estación al ejecutar únicamente la aplicación de Node-Red, sin ningún proceso extra durante 15 minutos. A continuación, se activaron todos los sensores incluyendo los sensores internos de la Raspberry Pi (temperatura y carga de la CPU) y tomando mediciones durante 15 minutos. Posteriormente, se obtuvieron las mediciones de corriente al realizar peticiones de ubicación con el GPS cada 5 segundos, de igual forma durante 15 minutos. En cuanto al consumo de corriente durante el procesamiento de imágenes se midió este parámetro durante los 22 minutos que duró el procesamiento. Finalmente, durante otros 15 minutos se midió la corriente mientras se realizaba transmisión de video en vivo. Los resultados promedios de estas mediciones se presentan en la Tabla 10. Adicionalmente se presenta una estimación de la corriente que demandan las diferentes tareas de forma individual restando el consumo correspondiente a la aplicación de Node-Red.

Tabla 10 Consumo de corriente de las tareas de la estación

Tarea	**Medición Promedio(mA)**	**Estimación Consumo por Tarea (mA)**
Plataforma Node-Red	301	301
Sensores	308	7
GPS	323	22
Procesamiento de imágenes	398	97
Transmisión de video	456	155

Cabe indicar que la tarea "Plataforma de Node-Red" incluye el consumo de todos los procesos del sistema operativo. Por otra parte, en la Tabla 11, se ha definido un tiempo estimado de activación para cada una de las tareas a lo largo de un día. A partir, de dichos valores se calcula el consumo total de energía que requiere la estación. Además, considerando que la batería instalada en la estación tiene una capacidad de 10400mAH y el consumo total estimado es de 7337mAH, la estación tendrá una autonomía estimada de 34 horas.

Tabla 11 Consumo de energía estimado por tarea

Tarea	**Tiempo estimado(H)**	**Tiempo estimado(H) * (Promedio(mA) - consumo Plataforma Node-Red(mA))**
Plataforma Node-Red	24	7224 mAH
Sensores	0,00923077	0,06461538 mAH
GPS	0,00555556	0,12222222 mAH
Procesamiento de imágenes	0,36666667	35,5666667 mAH
Transmisión de video	0,5	77,5 mAH
Consumo de energía total		7337,2535mAH

5.6.Conclusiones

En este capítulo se expuso los resultados de las pruebas realizadas sobre la estación para validar los algoritmos de procesamiento de imágenes usando fotografías tomadas con la cámara de la propia estación prototipo. Los resultados obtenidos con los dos algoritmos son satisfactorios. Sin embargo, cabe resaltar que el algoritmo de GMM con EM produce una mejor segmentación, aunque con un comportamiento mucho más lento que el algoritmo de *Thresholding*, el cual se podría implementar si se desea ahorrar una pequeña cantidad de energía. Además, se verificó la efectividad del protocolo MQTT para la transmisión de datos y video con tiempos de retardo admisibles para la aplicación diseñada. Por último, se analizó la autonomía energética de la estación, proceso necesario puesto que la estación podría emplazarse en un lugar remoto sin una fuente externa de energía.

6. CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan las conclusiones finales del trabajo de tesis, revisando brevemente sus principales características y funciones, junto con un conjunto de recomendaciones que podrían ser útiles para investigaciones futuras o para llevar la estación diseñada a operar en campos agrícolas.

6.1.Conclusiones

El trabajo realizado es un importante acercamiento entre la agricultura de precisión y el internet de las cosas, necesario para incrementar la tecnificación del campo. El conocer las variables ambientales de una parcela agrícola permite tomar acciones para optimizar recursos y aumentar la producción, y las fotografías de los frutos permiten estimar la fecha y magnitud de la cosecha, e incluso en trabajos futuros la información extraída podría ser usada para detectar enfermedades en la planta o problemas en la parcela. La estación diseñada es conveniente para nuestro medio en el cual el principal impedimento para esta tecnificación es el factor económico.

Para su creación se revisó una gran cantidad de trabajos relacionados, estudiando los principales algoritmos usados para la segmentación de imágenes, las tecnologías usadas en el Internet de las cosas y las ventajas de usar servicios en la nube.

El uso de los servicios de la nube de IBM como PaaS junto con el protocolo MQTT coloca a este trabajo en la escena actual de la emergente tecnología IoT y amplía la visión de las aplicaciones futuras que se pueden crear con ella.

El problema de procesamiento de imágenes al que se hizo frente, mostró la complejidad de la segmentación de imágenes como método para la detección de objetos dentro de un fondo con texturas y tonalidades muy distintas en fotografías de resolución moderada.

Los resultados del procesamiento de imágenes son satisfactorios, cumpliendo el objetivo de detectar efectivamente una gran cantidad de frutos y extraer información que puede ser útil en trabajos futuros.

La transmisión de datos por MQTT mostró las ventajas de este protocolo, y por qué este se está convirtiendo en el más popular para aplicaciones IoT. Además, la transmisión de video mostró una aplicación no común sobre este protocolo con buenos resultados.

La aplicación web creada permite el monitoreo de las variables de forma muy sencilla y desde cualquier lugar del mundo, además de permitir ubicar la estación gracias al sensor de geo-localización.

Por último, la caracterización de la energía de la estación permitió estimar el tiempo de autonomía energética, parámetro importante a considerar para mantener la estación operativa.

6.2.Recomendaciones

Entre las principales recomendaciones para futuros trabajos, en primer lugar, se resalta la conveniencia de mejorar las capacidades del protocolo MQTT, para lo cual se podría crear un *broker* propio, y así evitar las limitaciones del tamaño máximo de los paquetes.

En cuanto al procesamiento de imágenes, aunque los resultados obtenidos son en general muy buenos, se sugiere realizar un mayor número de experimentos, probando más algoritmos de segmentación, pre-procesamiento o post-procesamiento. No obstante, encontrar una solución definitiva resulta de gran complejidad, puesto que este es un problema muy dependiente de las condiciones lumínicas del lugar e incluso de la calidad de las imágenes que captura la cámara, razón por la cual tanto en el presente trabajo de tesis como en los trabajos relacionados que han sido analizados, se llega a una solución para un problema específico en forma de aproximación.

Otra mejora que se podría dar al sistema, especialmente con un enfoque comercial, consiste en reducir la energía que consumen las tareas del sistema operativo, para así llegar a tener mayor tiempo de independencia. Esto se lo podría lograr desinstalando muchos de los programas que Raspbian trae instalados por defecto, o incluso, aunque mucho más demandante, se podría crear una distribución de Linux que solo tenga instalados los programas absolutamente necesarios.

Una recomendación importante si se desea emplazar esta estación en el campo es construir un mejor encapsulado que la proteja de todas las inclemencias del clima. En particular, si se desea instalar la estación en un invernadero, el sistema debe estar muy bien refrigerado, debido a las altas temperaturas que se generan en estos ambientes, lo cual podría hacer difícil que la estación funcione sin una fuente de energía externa.

REFERENCIAS

[1] Universidad Técnica del Norte, “La Importancia de la Agricultura en nuestro país,” 2017. [Online]. Available: http://www.utn.edu.ec/ficaya/carreras/agropecuaria/?p=1091.

[2] A. Monteros, E. Sumba, and S. Salvador, “Productividad Agrícola en el Ecuador,” *Magap*, p. 12, 2014.

[3] Food and Agriculture Organization of the United Nations, “FAOSTAT,” *Crops Production*, 2016. [Online]. Available: http://www.fao.org/faostat/en/#data/QC.

[4] V. M. Andrade, “La tecnificación agrícola,” *El Telégrafo*, p. 1, 26-Apr-2016.

[5] Diario EL Telégrafo, “Desarrollo y Tecnificación,” p. 1, 28-Jan-2014.

[6] E. García and F. Flego, “Agricultura de Precisión,” *Cienc. Tecnol. Univ. Palermo*, pp. 99–116, 2018.

[7] Beecham, “Towards Smart Farming: Agriculture Embracing the IoT Vision,” *Beechham Res.*, vol. 44, no. 0, p. 6, 2014.

[8] P. Gennari, *FAO Statistical Pocketbook 2015*. Rome, 2015.

[9] M. Oxnam *et al.*, “State of the Market THE INTERNET OF THINGS 2015,” *Info*, vol. 54, no. 1, pp. 1–31, 2014.

[10] V. Alchanatis, A. Navon, I. Glazer, and S. Levski, “An Image Analysis System for measuring Insect Feeding Effects caused by Biopesticides,” *J. Agric. Eng. Res.*, vol. 77, no. 3, pp. 289–296, 2000.

[11] S. Khanal, J. Fulton, and S. Shearer, “An overview of current and potential applications of thermal remote sensing in precision agriculture,” *Comput. Electron. Agric.*, vol. 139, pp. 22–32, 2017.

[12] A. J. G. Sanchez, F. G. Sanchez, and J. G. Haro, “Wireless sensor network deployment for integrating video-surveillance and data-monitoring in precision agriculture over distributed crops,” *Comput. Electron. Agric.*, vol. 75, no. 2, pp. 288–303, 2011.

[13] J. Senthilnath, A. Dokania, M. Kandukuri, R. K.N., G. Anand, and S. N. Omkar, “Detection of tomatoes using spectral-spatial methods in remotely sensed RGB images captured by UAV,” *Biosyst. Eng.*, vol. 146, pp. 16–32, 2016.

[14] S. Lal, S. K. Behera, P. K. Sethy, and A. K. Rath, “Identification and counting of mature apple fruit based on BP feed forward neural network,” *Proc. 2017 3rd IEEE Int. Conf. Sensing, Signal Process. Secur. ICSSS 2017*, pp. 361–368, 2017.

[15] S. Abirami and M. Thilagavathi, “Classification of fruit diseases using feed forward back propagation neural network,” *Proc. 2019 IEEE Int. Conf. Commun. Signal Process. ICCSP 2019*, pp. 765–768, 2019.

[16] P. Ogallar, “Nuevas tecnologías: la potencia de la inteligencia artificial,” *La Nacion*, 2017. [Online]. Available: https://www.lanacion.com.ar/2050218-nuevas-tecnologias-la-potencia-de-la-inteligencia-artificial.

[17] V. Mendoza, “Asimetría agrícola,” *El Telégrafo*, 04-Nov-2014.

[18] D. Evans, “Internet de las cosas Internet de las cosas Cómo la próxima evolución de Internet lo cambia todo,” *Inf. técnico Cisco*, 2011.

[19] L. Columbus, “2017 Roundup Of Internet Of Things Forecasts,” *Forbes*, 2017. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#dfed2061480e.

[20] K. Rose, S. Eldridge, and L. Chapin, “La Internet De Las Cosas — Una Breve Reseña,” *Internet Soc.*, p. 83, 2015.

[21] Statista.com, “Number of machine-to-machine (M2M) connections worldwide from 2012 to 2018 (in millions)*,” 2018.

[22] A. dos Santos Ferreira, D. Matte Freitas, G. Gonçalves da Silva, H. Pistori, and M. Theophilo Folhes, “Weed detection in soybean crops using ConvNets,” *Comput. Electron. Agric.*, vol. 143, no. November, pp. 314–324, 2017.

[23] M. Dyrmann, H. Karstoft, and H. S. Midtiby, “Plant species classification using deep convolutional neural network,” *Biosyst. Eng.*, vol. 151, no. 2005, pp. 72–80, 2016.

[24] Y. Lu and R. Lu, *Quality Evaluation of Apples*. 2016.

[25] N. Kulkarni, “Color Thresholding Method for Image Segmentation of Natural Images,” *Int. J. Image, Graph. Signal Process.*, vol. 4, no. 1, pp. 28–34, 2012.

[26] S. Wazarkar and B. N. Keshavamurthy, “A survey on image data analysis through clustering techniques for real world applications,” *J. Vis. Commun. Image Represent.*, vol. 55, pp. 596–626, 2018.

[27] R. Lagani, *OpenCV 2 Computer Vision Application Programming Cookbook*, vol. 14, no. 4. 2011.

[28] V. Gupta, “Color spaces in OpenCV (C++ / Python),” *Learn OpenCV*, 2017. [Online]. Available: https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/.

[29] J. Howse, *OpenCV Computer Vision with Python Table of Contents*. Birmingham, 2013.

[30] R. Zhou, L. Damerow, Y. Sun, and M. M. Blanke, “Using colour features of cv. ‘Gala’ apple fruits in an orchard in image processing to predict yield,” *Precis. Agric.*, vol. 13, no. 5, pp. 568–580, 2012.

[31] Scikit Learn, “Gaussian mixture models.” [Online]. Available: https://scikit-learn.org/stable/modules/mixture.html.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” *J. R. Stat. Soc. Ser. B*, vol. 39, no. 3, pp. 293–297, 1977.

[33] I.J.Myung, “Computational Approaches to Model Evaluation,” *Int. Encycl. Soc. Behav. Sci.*, vol. 339, no. 16, pp. 2453–2457, 2001.

[34] Scikit Learn, “Gaussian Mixture Model Selection.”

[35] C. Fraley, “How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis,” *Comput. J.*, vol. 41, no. 8, pp. 578–588, 2005.

[36] R. Gonzalez and R. Woods, *Digital Image Processing*. 2002.

[37] L. Di Stefano and A. Bulgarelli, “A simple and efficient connected components labeling algorithm,” *Proc. - Int. Conf. Image Anal. Process. ICIAP 1999*, pp. 322–327, 1999.

[38] Y. S. Halabi, “New Algorithm - Simulation Connected Components Labeling for Binary Images .,” vol. 3, no. 12, 2013.

[39] T. Asano and H. Tanaka, “In-Place Algorithm for Connected Components Labeling,” *J. Pattern Recognit. Res.*, vol. 5, no. 1, pp. 10–22, 2013.

[40] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, “The connected-component labeling problem: A review of state-of-the-art algorithms,” *Pattern Recognit.*, vol. 70, pp. 25–43, 2017.

[41] OpenCV.org, “About OpenCV.” [Online]. Available: https://opencv.org/about/.

[42] FFMEG, “About FFmpeg.” [Online]. Available: https://www.ffmpeg.org/about.html.

[43] OpenJS Foundation, “Node-Red.” [Online]. Available: https://nodered.org.

[44] G. C. Hillar, *MQTT Essentials - A Lightweight IoT Protocol*. Birmingham: Packt Publishing Ltd., 2017.

[45] MQTT.org, “MQTT Documentation.” [Online]. Available: mqtt.org/documentation.

[46] J. Polo, G. Hornero, C. Duijneveld, A. García, and O. Casas, “Design of a low-cost Wireless Sensor Network with UAV mobile node for agricultural applications,” *Comput. Electron. Agric.*, vol. 119, pp. 19–32, 2015.

[47] S. Sabzi, Y. Abbaspour-Gilandeh, and G. García-Mateos, “A new approach for visual identification of orange varieties using neural networks and metaheuristic algorithms,” *Inf. Process. Agric.*, vol. 5, no. 1, pp. 162–172, 2018.

[48] A. Bakhshipour and A. Jafari, “Evaluation of support vector machine and artificial neural networks in weed detection using shape features,” *Comput. Electron. Agric.*, vol. 145, no. December 2017, pp. 153–160, 2018.

[49] M. Sharif, M. A. Khan, Z. Iqbal, M. F. Azam, M. I. U. Lali, and M. Y. Javed, “Detection and classification of citrus diseases in agriculture based on optimized weighted segmentation and feature selection,” *Comput. Electron. Agric.*, vol. 150, no. April, pp. 220–234, 2018.

[50] T. J. R. Rozo, J. C. G. Alvarez, and C. G. C. Dominguez, “Infrared thermal image segmentation using expectation-maximization-based clustering,” *STSIVA 2012 - 17th Symp. Image, Signal Process. Artif. Vis.*, pp. 223–226, 2012.

[51] Yudong Guan, Qi Zhang, Xutao Zhang, Youhua Jia, and Shen Wang, “A Study of Color Image Segmentation Base on Stochastic Expectation Maximization

Algorithm in HSV Model," pp. 1198–1200, 2006.

[52] E. Fida, J. Baber, M. Bakhtyar, R. Fida, and M. J. Iqbal, "Unsupervised image segmentation using lab color space," *2017 Intell. Syst. Conf. IntelliSys 2017*, vol. 2018-Janua, no. September, pp. 774–778, 2018.

[53] A. Bhargava and A. Bansal, "Fruits and vegetables quality evaluation using computer vision: A review," *J. King Saud Univ. - Comput. Inf. Sci.*, 2018.

[54] S. Chanthakit and C. Rattanapoka, "Mqtt based air quality monitoring system using node MCU and node-red," *Proceeding 2018 7th ICT Int. Student Proj. Conf. ICT-ISPC 2018*, pp. 1–5, 2018.

[55] A. Rajalakshmi and H. Shahnasser, "Internet of things using node-red and alexa," *2017 17th Int. Symp. Commun. Inf. Technol. Isc. 2017*, vol. 2018-Janua, pp. 1–4, 2018.

[56] M. Lekić and G. Gardašević, "IoT sensor integration to Node-RED platform," *2018 17th Int. Symp. INFOTEH-JAHORINA, INFOTEH 2018 - Proc.*, vol. 2018-Janua, no. March, pp. 1–5, 2018.

[57] M. Tabaa, B. Chouri, S. Saadaoui, and K. Alami, "Industrial Communication based on Modbus and Node-RED," *Procedia Comput. Sci.*, vol. 130, pp. 583–588, 2018.

[58] J. Skovranek, M. Pies, and R. Hajovsky, "Use of the IQRF and Node-RED technology for control and visualization in an IQMESH network," *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 295–300, 2018.

[59] Texas Instruments, "INA219: CURRENT / POWER MONITOR with I2C™ Interface," no. September, pp. 1–29, 2011.

[60] Silicon Labs, "PROXIMITY/UV/AMBIENT LIGHT SENSOR IC WITH I2C INTERFACE," 2013.

[61] Adafruit Industries, "Adafruit BME280 Humidity + Barometric Pressure + Temperature Sensor Breakout," pp. 1–19, 2016.

[62] Lady Ada, "Adafruit Ultimate GPS," *Adafruit Ind.*, pp. 1–38, 2014.

[63] Logitech, "LOGITECH ® HD WEBCAM c270," 2011.

[64] RavPower, "External Battery Pack Element Model RP-PB07." p. 6.

[65] Numpy.org, "Numpy." [Online]. Available: https://www.numpy.org.

[66] Scikit-learn.org, "Scikit-learn Machine learning in Python." [Online]. Available: https://scikit-learn.org/stable/.

[67] "ImageNet." [Online]. Available: http://image-net.org/explore.

APENDICES

A. Entorno de Trabajo IBM Cloud

Para la fecha de realización de este trabajo la empresa IBM se encuentra en una etapa de transición de sus servicios en la nube, lo cual incluye, un cambio de nombre, de la dirección web y de la interfaz gráfica de su plataforma, aunque esencialmente sus servicios son los mismos. El nombre de dominio empleado inicialmente por la plataforma IBM Cloud fue Bluemix con su dirección asociada *Bluemix.net*, por tal razón la gran mayoría de documentación sobre este servicio está bajo ese nombre, e incluso dentro de la plataforma aún existe mucha referencia a dicho nombre, lo cual con esta aclaración no debería causar confusiones. Como se mencionó la interfaz gráfica también se encuentra en transición, pero por el momento aún se puede trabajar sobre ella en el dominio *Bluemix.net*. Sin embargo, considerando que esta interfaz se dará de baja a mediano plazo, no se trabajará sobre ella. Para trabajar sobre la plataforma actualizada se debe ingresar a la dirección *cloud.ibm.com*.

Al ingresar en esta dirección se presenta la página de inicio de sesión como se muestra en la Figura 52. Para poder iniciar una sesión, es necesario contar con una cuenta de IBM, la misma que puede ser obtenida de forma gratuita, registrándose en su página.

Figura 52 Página de inicio IBM Cloud

Al iniciar sesión se presenta el panel de control de la plataforma IBM Cloud, en el cual se muestran ventanas sobre el resumen de recursos, las aplicaciones creadas, los costes por uso, manuales de usuario entre otras herramientas, como se muestra en la Figura 53.

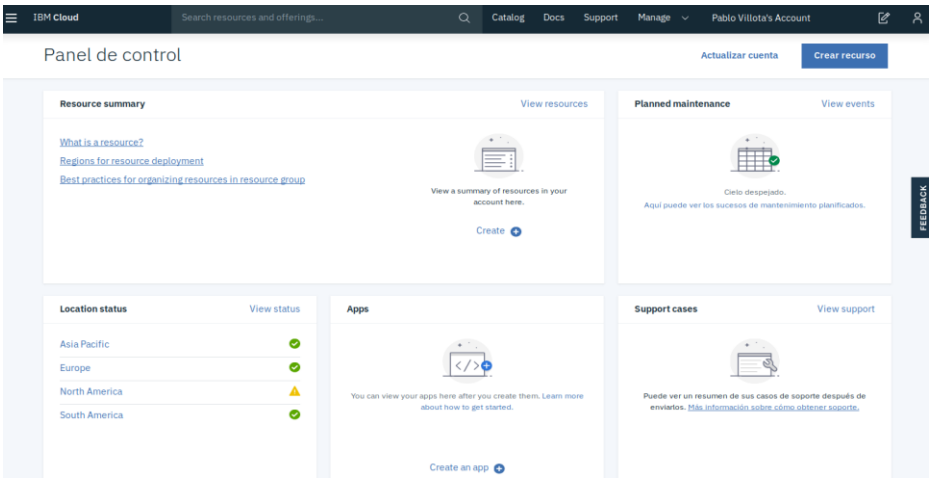

Figura 53 Panel de control IBM Cloud

En la esquina superior izquierda se muestra el menú de Navegación en el cual se encuentran los principales recursos disponibles, tal como se lo observa en la Figura 54.

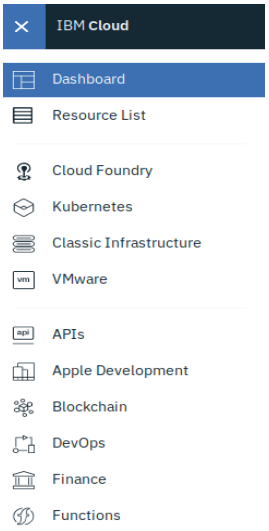

Figura 54 Menú de Navegación IBM Cloud

Otro elemento importante a destacar, es el catálogo, cuyo botón de acceso se muestra en la parte superior de la interfaz de la plataforma. En el catálogo se puede visualizar todos los servicios a los que se puede acceder como herramientas de desarrollo, contenedores, inteligencia artificial, bases de datos, entre otros. En la Figura 55 se muestra dicho catálogo.

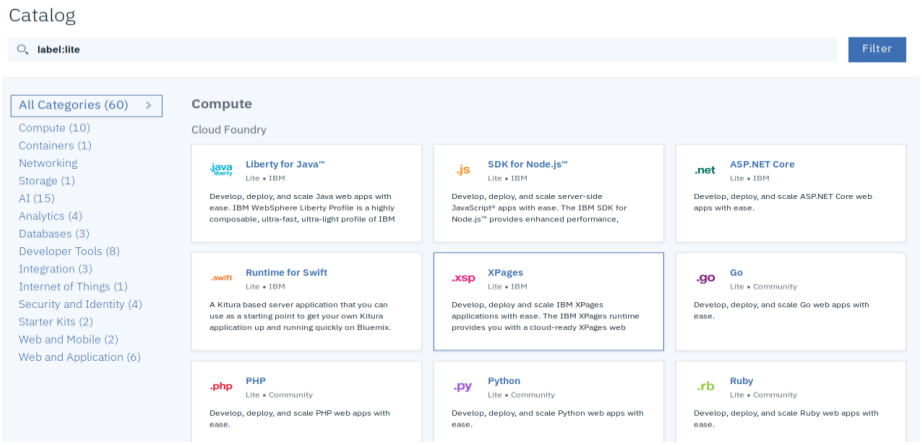

Figura 55 Catálogo IBM Cloud

Para el caso del trabajo de tesis, los servicios empleados de este catálogo son *Internet of Things Plataform* y *Node-Red Starter* como se lo observa en la Figura 56.

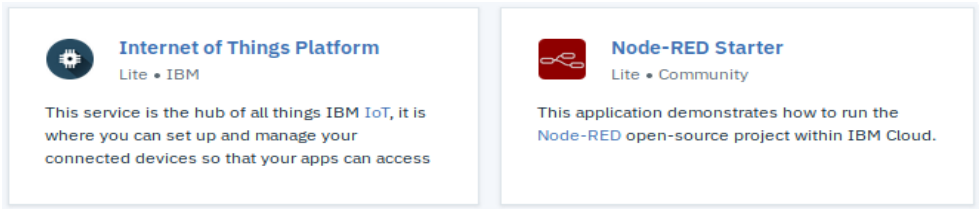

Figura 56 Servicios de IBM Cloud que se usarán

B. Entorno de Trabajo Node-Red Starter

Para iniciar la plataforma de desarrollo con Node-Red se busca en el catálogo la opción *Node-Red Starter* que desplegará el formulario mostrado en la Figura 57:

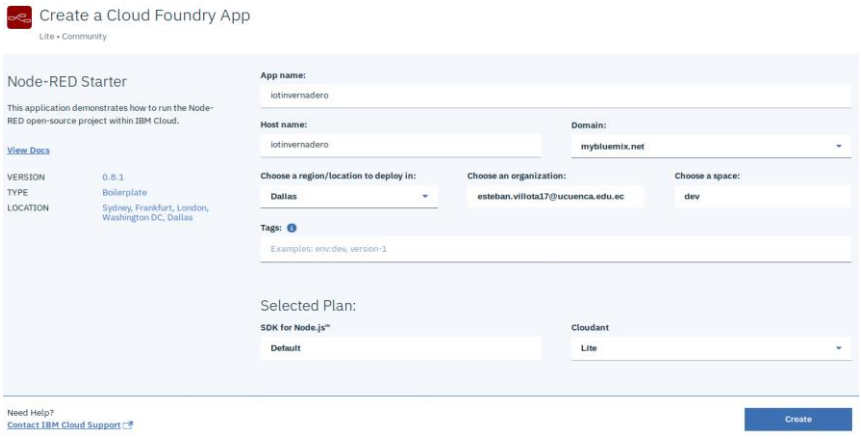

Figura 57 Instanciación de servicio Node-RED Starter

Como se observa, se debe otorgar un nombre de la aplicación y un nombre de host. El resto de campos pueden mantenerse con sus valores por defecto siempre y cuando sea conveniente. En la sección de *Selected Plan* (Plan seleccionado) es importante colocar el servicio *SDK for Node.js* como *"Default"* y el servicio *Cloudant* como *"Lite"*. Tal configuración, se realiza para que el uso de la plataforma sea gratuito. Finalmente, es posible confirmar la opción "Create", a continuación de lo cual se mostrará la ventana de Figura 58:

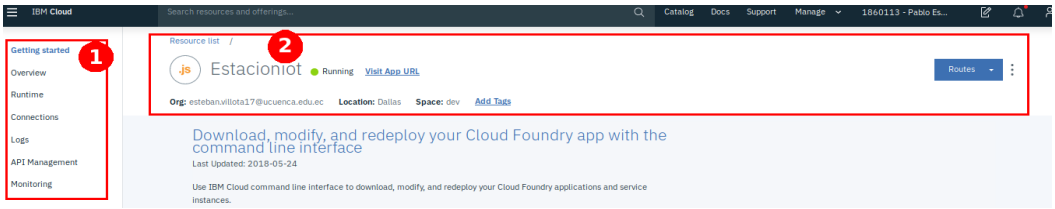

Figura 58 Ventana de inicio Node-RED Starter

La captura de la Figura 58, corresponden a la interfaz gráfica de la plataforma. En particular, se distinguen dos secciones importantes. Por una parte, 1) el menú de la plataforma, en el cual se puede acceder tutoriales, detalles de uso, consola mediante conexión ssh (*Secure Shell*), registro de uso, entre otros. Además, se encuentra 2) la lista de recursos, en la cual se puede ver si la aplicación se está ejecutando, acceder a su URL y realizar acciones de gestión como por ejemplo iniciar, detener o reiniciar la aplicación.

El uso de esta plataforma es gratuito si se trabaja bajo ciertas condiciones detalladas en los términos y condiciones de uso de la aplicación. Una de estas condiciones es que el tamaño de la aplicación no puede exceder los 256MB, lo cual para el presente trabajo resulta escaso. Para ampliar la capacidad hay que dirigirse a la opción *Overview* (Visión General). Aquí, en la opción *MB Memory per instance* hacemos clic en el signo más hasta llegar a 512, tal como se muestra en la Figura 59:

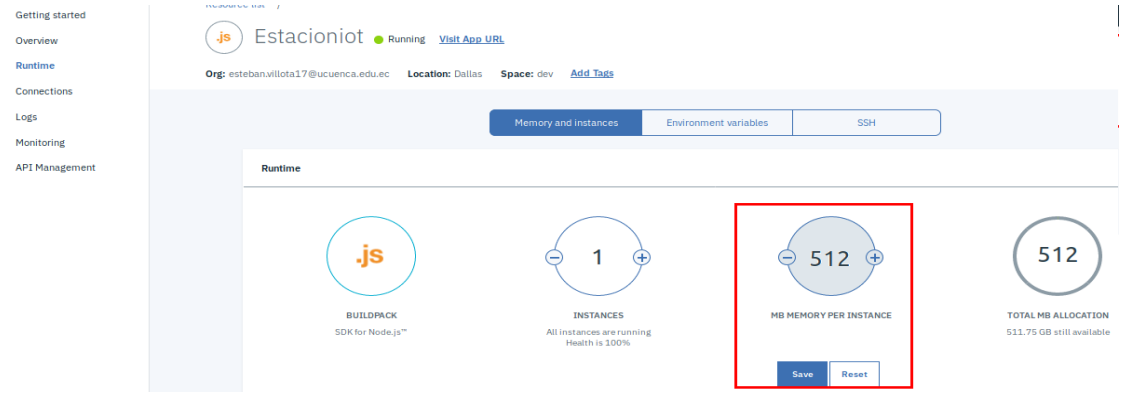

Figura 59 Incremento de memoria de la instancia

Una vez hecho esto y dar clic en guardar, se mostrará un mensaje que advierte que debemos cambiar de plan, lo que a su vez llevará a un formulario para ingresar la información de una tarjeta de crédito y datos adicionales. Luego de realizar este trámite se mostrará un mensaje confirmando que con el ingreso de los datos de la tarjeta de crédito ahora se puede usar hasta 512MB sin costo adicional, por lo que el uso de la aplicación seguirá siendo gratuito.

Finalmente hay que dirigirse al entorno de trabajo de Node-Red. Para esto, en la sección de lista de recursos se da un clic en *"Visitar URL de la aplicación",* lo que conducirá a la ventana de inicio que se muestra en la Figura 60.

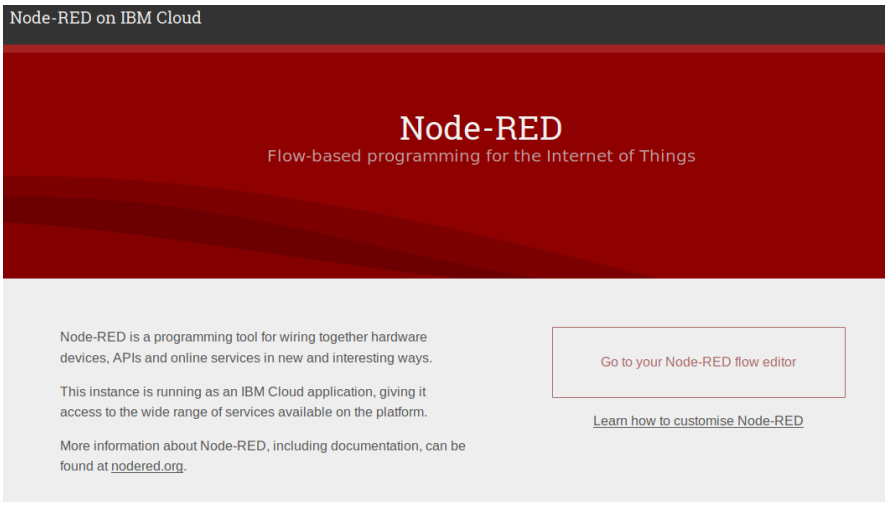

Figura 60 Página de inicio del editor Node-RED

Al dar clic en la opción *Go to your Node-RED flow editor,* se redirigirá a un formulario en el cual se debe ingresar los datos de un usuario que podrá modificar la aplicación, y luego de esto finalmente se mostrará el editor de Node-RED como se lo aprecia en la Figura 61:

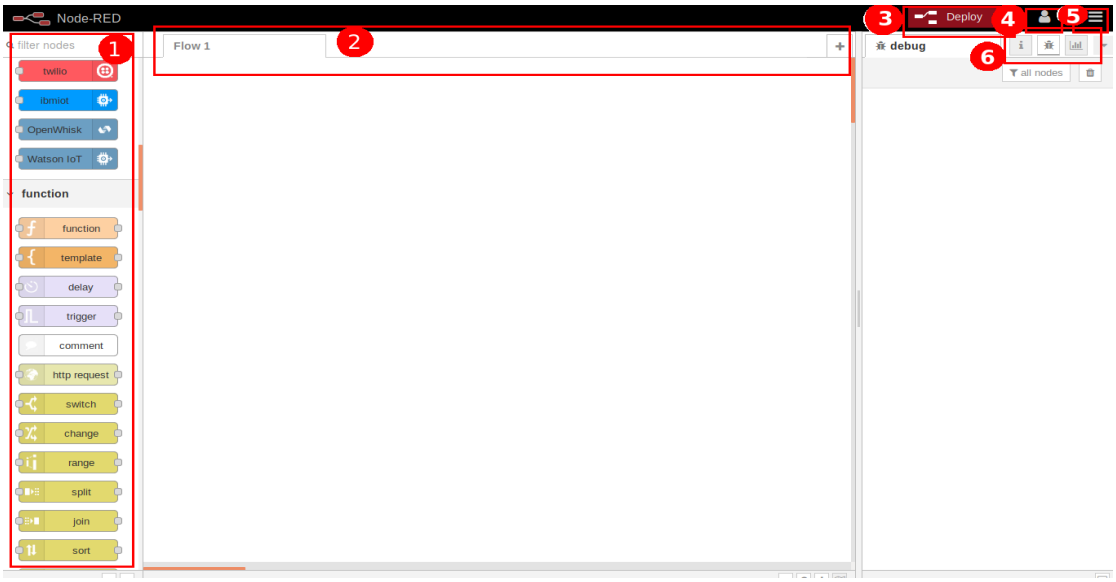

Figura 61 Editor Node-RED

En este editor se distinguen seis partes importantes. 1) La paleta de nodos, donde se ubican todos los nodos o bloques con los que se programará la aplicación. 2) Pestañas de flujos de trabajo, es aquí donde se desplegarán los nodos de programación, cabe resaltar que los distintos flujos se ejecutan paralelamente. 3) Botón desplegar, este botón guarda los cambios realizados sobre los flujos de trabajo e inicia su ejecución. 4) Botón de registro, es necesario estar registrado para poder hacer cambios al programa. 5) Menú del editor, posteriormente se describirán algunas de sus opciones. 6) Pestañas de información, las mismas que alojan múltiple información, dependiendo de las librerías instaladas y de los nodos usados. Entre los datos de mayor relevancia se encuentra, la información sobre los nodos, el panel de *debug*, la configuración del panel de control y la configuración de los bloques.

En cuanto al menú del editor, observado en la Figura 62, las opciones más importantes son: *Importar,* que permite ingresar código en formato json para desplegar los nodos correspondientes, además se puede encontrar ejemplos de las librerías instaladas. *Exportar,* con el cual se puede respaldar la programación de los nodos en formato json. *Manage palette,* esta opción desplegará una ventana en la cual es posible verificar las librerías instaladas, así como instalar nuevas librerías. En la Figura 63 se muestra una captura de esta ventana.

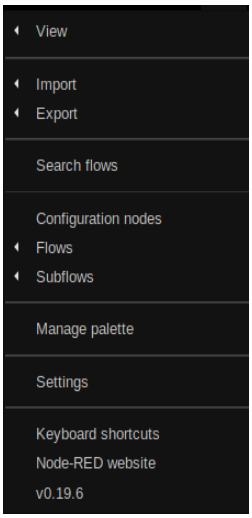

Figura 62 Menú del editor Node-RED

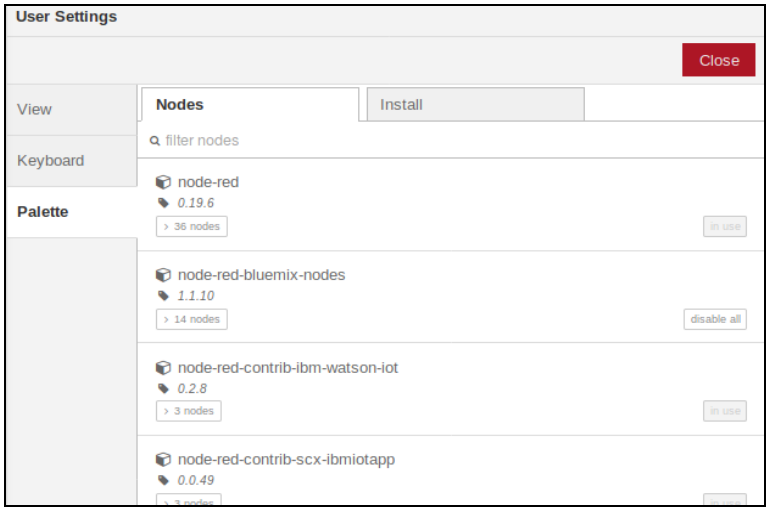

Figura 63 Ventana de "Manage Palette"

Este editor estará disponible en la URL *[Nombre de host].mybluemix.net*. Para el caso del editor en los dispositivos, se puede acceder con cualquier explorador web en la URL *[ip privada del host]:1880.*

C. Entorno de Trabajo IBM Watson IoT

Para usar la plataforma IBM Watson IoT, se debe iniciar una nueva instancia de la plataforma buscándola en el catálogo como *Internet of Things Platform*. Se solicitará asignar un nombre al servicio, y en cuanto al resto de opciones, se pueden emplear los valores por defecto. El formulario para ingresar estos datos se muestra en la Figura 64.

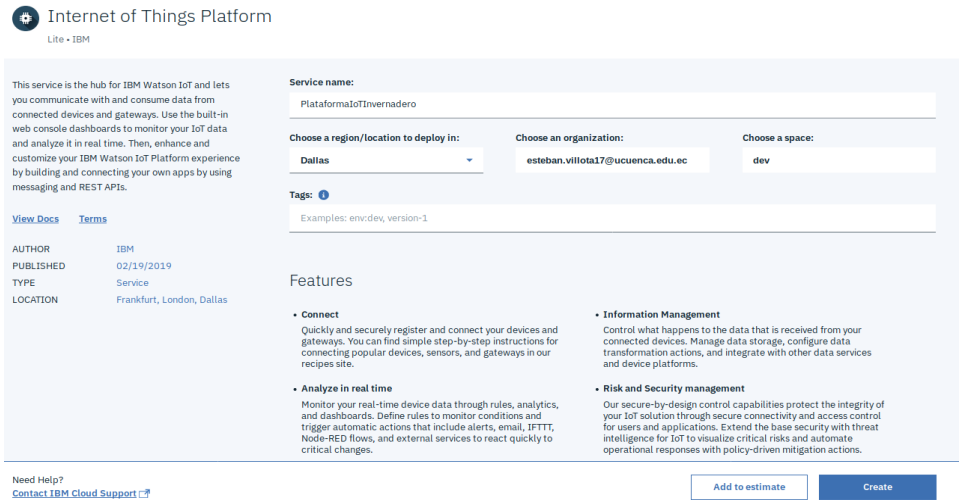

Figura 64 Formulario de instanciación IBM Watson IoT

Al aceptar la opción *"Create",* se mostrará la ventana indicada en la Figura 65, en la cual se detalla la cantidad de datos intercambiados y disponibles. También se presenta la opción para modificar el plan y vincular más servicios con la plataforma. En la sección, *Manage* se encuentra el botón *Lanzar,* el cual conduce al entorno de trabajo de IBM Watson IoT.

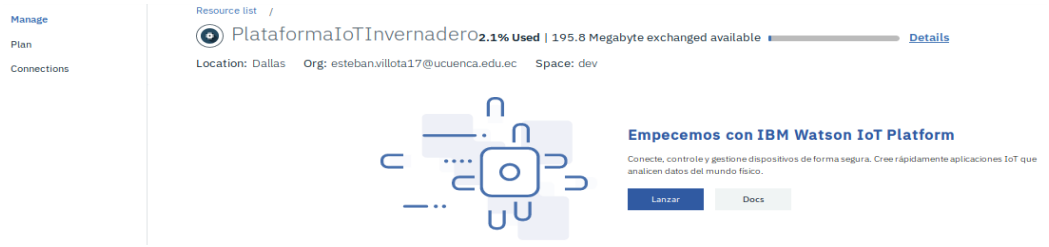

Figura 65 Administrador de la instancia IBM Watson IoT

En la plataforma se distinguen dos partes importantes. 1) El Menú de la plataforma, en la cual se puede observar botones con los cuales acceder a la administración de los dispositivos, los usuarios, las aplicaciones, las reglas de seguridad, entre otros. Las dos opciones que se emplearon en el presente trabajo fueron los dispositivos y las aplicaciones. 2) Información de la cuenta, donde se muestra el usuario que está empleando la plataforma y el ID de la organización, el cual será necesario al momento de configurar la comunicación entre el dispositivo y la aplicación web. La ventana de esta plataforma se la observa en la Figura 66.

Figura 66 Plataforma IBM Watson IoT

D. Registro de Dispositivos en la Plataforma IBM Watson IoT

Dentro de la plataforma IBM Watson IoT en la opción de *Dispositivos*, como primer paso se debe crear un tipo de dispositivo. Los tipos de dispositivos sirven para englobar un conjunto de características comunes entre un grupo de dispositivos. En este caso, se creará un tipo de dispositivo al que se ha denominado *"Raspberry"*. Además del nombre, para crear un tipo de dispositivo también se piden datos como el número de serie, modelo, descripción, versión del hardware, fabricante, clase de dispositivo, versión del firmware y ubicación descriptiva. Sin embargo, estos datos son opcionales y se los puede dejar en blanco. Los formularios para ingresar esta información se presentan en la Figura 67 y Figura 68.

Figura 67 Formulario tipo de dispositivo

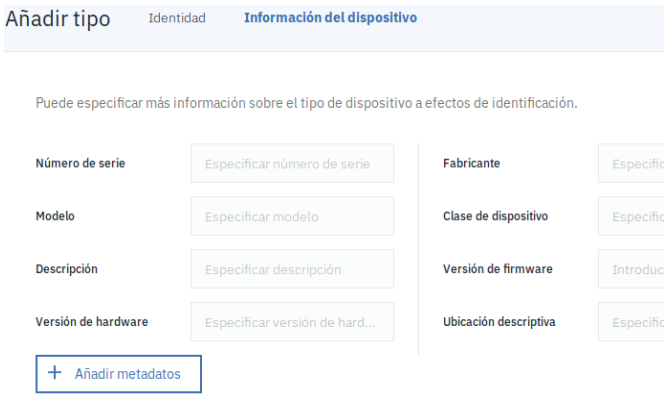

Figura 68 Datos adicionales tipo de dispositivo

Una vez definido el tipo, a continuación se procede a crear el dispositivo, para ello se selecciona el tipo de dispositivo y se le coloca un ID única que en este caso será *"RasPi1"*, como se puede apreciar en la Figura 69.

Figura 69 Ventana nuevo dispositivo

Posteriormente, se solicitan datos adicionales, los cuales en su mayoría son de tipo opcional y pueden omitirse. Además, se pueden añadir metadatos en formato JSON. Dicho formulario se muestra en la Figura 70.

Figura 70 Datos adicionales del dispositivo

Por último, se tendrá que proporcionar una señal de autenticación para el dispositivo, tal como se lo observa en la Figura 71. En caso de no especificarse una, ésta será generada de forma automática.

Figura 71 Formulario para la señal de autenticación del dispositivo

Una vez creado el dispositivo se visualizará un resumen de sus datos junto con la señal de autenticación, la cual deberá ser recordada, puesto que no se podrá acceder a ella nuevamente.

Las credenciales del dispositivo habilitan la conexión del lado del dispositivo, pero para habilitar la conexión del lado de la aplicación web es necesario generar una clave de API y sus correspondientes *tokens* de aplicación. Para ello, hay que dirigirse a la pestaña de *"Aplicaciones"* ubicada en el menú principal de la plataforma Watson y posteriormente dar clic en el botón *"Generar clave de API"*. Como resultado, se desplegará el formulario mostrado en la Figura 72, para agregar una descripción y una fecha de caducidad de la clave, los cuales son opcionales.

Figura 72 Datos adicionales para la clave de API

Al presionar siguiente, se mostrará un formulario para ingresar el rol de la aplicación. En este campo se selecciona la opción *"Aplicación estándar"*. Finalmente se genera la clave, visualizándose en pantalla junto con la señal de autenticación. Nuevamente se debe tomar en cuenta, que esta información debe ser anotada, puesto que no volverá a mostrarse. En la Figura 73, se presentan estos datos

Figura 73 Datos de la clave de API

E. Instalación de los Sensores de Recolección de Datos

¬ Instalación Sensor INA219

Para poder usar el sensor INA219 es necesario instalar la librería que abstrae el uso de sus registros. Al momento existen dos librerías, *pi-ina219* y *Adafruit_CircuitPython_INA219*. Esta última se ejecuta sobre *CicuitPython*. Se optó por la primera alternativa, la cual se puede instalar usando el siguiente comando desde la terminal

$sudo pip uninstall pi-ina219

El programa desarrollado en Python realiza lecturas periódicas del sensor, empleando para ello el método *getCurrent_mA()*.

Cuando el nodo de ejecución en Node-RED corre este, o cualquier otro de los programas implementados, recoge los datos impresos en pantalla y los envía al siguiente nodo, que en este caso es el *wiotp out,* el cual se encarga de enviar los datos a la nube de IBM.

¬ Instalación Sensor BME280

Para manejar el sensor BME280, se usa la librería adafruit_bmp280, escrita originalmente para el intérprete *Circuitpython*. Para usar dicha librería, previamente se debe instalar la librería *adafruit-blinka* que funciona como una capa intermedia entre el intérprete Python 3.4 y la API de *hardware* en *Circuitpython*. Esta librería se instala usando el Gestor de Paquetes de Python (pip) ingresando los siguientes comandos en el terminal.

$ sudo pip3 install --upgrade setuptools

$ pip3 install RPI.GPIO

$ sudo pip3 install adafruit-blinka

A continuación, se instala la librería adafruit_bmp280 con el siguiente comando.

$ sudo pip3 install adafruit-circuitpython-bmp280

Hecho esto, es necesario adicionar un último paso. La librería adafruit_bmp280 está creada para el sensor BMP280. El sensor BME280 es la nueva generación de los sensores de temperatura y presión, y aunque tiene muchas semejanzas con el sensor BMP280, difiere en ciertos aspectos, uno de ellos es su número de identificación de chip. En el BMP280 es 0x58 y en el BME280 es 0x60. Dicha identificación se encuentra alojada en el registro 0xD0 de la memoria del sensor. Así, para poder usar la librería es

necesario modificarla para que reconozca este nuevo número de identificación. Para ello se ingresa en el código de la librería con el comando.

$ sudo nano /usr/local/lib/python3.4/dist-packages/adafruit_bmp280.py

Aquí se buscará la línea 44 correspondiente al comando: _CHIP_ID = const(0x58) y se la cambiará por _CHIP_ID = const(0x60).

Una vez completado estos pasos, es posible realizar lecturas de los datos de temperatura y presión con los métodos *temperature* y *pressure* de la librería. Es importante resaltar que este programa correrá solamente en Python3

¬ Instalación Sensor SI1145

El sensor SI1145 utiliza la librería disponible en *Github* bajo el nombre *Python_SI1145***.** Una vez descargada se la descomprime y dentro de su carpeta se ejecuta el siguiente comando desde la terminal.

$sudo python setup.py install

Una vez instalado, se puede leer los datos del sensor con el método *readUV()*.

¬ Sensores de Temperatura y Carga del Procesador

Para la recolección de los datos de la temperatura del procesador no se requieren librerías extra, ya que basta ejecutar el comando *"vcgencmd measure_temp"*, el cual imprime la temperatura en pantalla. En este caso, se implementó un script en bash, para realizar periódicamente esta lectura y colocar los datos en formato JSON.

Para obtener la carga del CPU es necesario instalar el módulo *psutils*, con los comandos:

$sudo apt-get install build-essential python-dev python-pip

$sudo pip install psutil

Con esto se puede realizar un script en Python obteniendo la carga del procesador con el método *get_cpuload()*.

¬ Instalación Sensor de Geo-localización

Para el funcionamiento del sensor de geo-localización es preciso instalar la librería *gpsd* con el comando.

$sudo apt-get install gpsd gpsd-clients

Luego es necesario desactivar la instancia de *gpsd,* la cual es inicializada durante la instalación, para esto se ejecutan los siguientes comandos.

$sudo systemctl stop gpsd.socket

$sudo systemctl disable gpsd.socket

Después se debe iniciar el servicio de gpsd con el siguiente comando.

$sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock

Es importante que este comando sea ejecutado con el dispositivo USB correspondiente al GPS, en este caso fue ttyUSB0. Este comando debe ser ingresado manualmente en cada ocasión luego de encender el dispositivo.

Luego de realizados estos pasos se requiere instalar la librería de Python, para lo cual se usa el gestor de librerías pip3 con el siguiente comando.

```
$pip3 install gps
```

Con la librería ya instalada se puede escribir un script en Python para obtener las coordenadas del GPS. En el script, luego de importar la librería, se pueden leer los reportes creados por el dispositivo. El dispositivo crea gran cantidad de datos dispuestos en reportes expresados en formato JSON. Para el caso de las coordenadas, se debe encontrar el reporte cuya clase sea 'TPV' y leer los campos "lat" y "lon". Luego estos datos se los coloca en formato JSON y se los imprime en pantalla para que puedan ser recogidos por el nodo de ejecución de Node-RED.

F. Diseño de las Placas Electrónicas

El diseño de la placa electrónica para integrar los sensores ambientales y de corriente se lo realizó sobre una placa de fibra de vidrio de doble cara. Además de las conexiones de estos 3 módulos de sensores, también se incluyó 3 indicadores leds, una salida para otra placa con botones y pines para la conexión I2C (Inter Integrated Circuits) de un sensor extra que se podría conectar eventualmente. El diseño electrónico de la placa se muestra en la Figura 74.

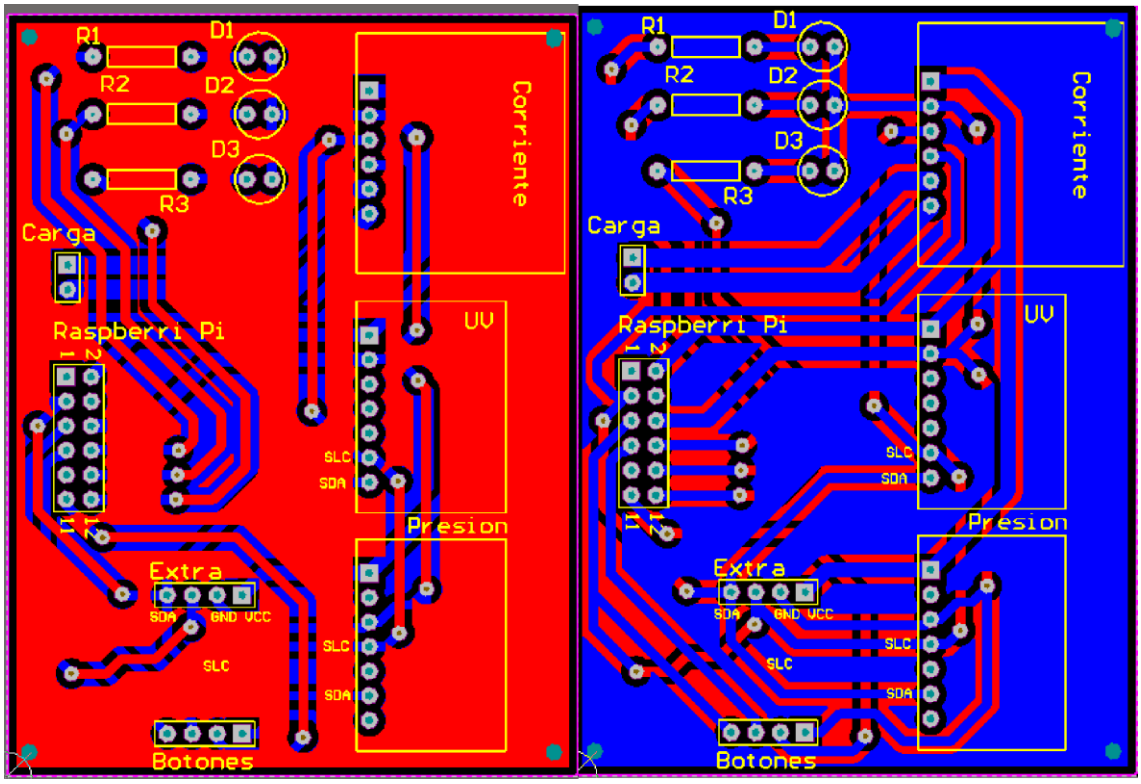

Figura 74 Placa electrónica de sensores (izq.) Cara superior (der.) Cara inferior

La placa de botones es más sencilla y de menor dimensión, tiene solo una cara con espacios para conectar 3 botones. Estos botones tienen la finalidad de realizar acciones sobre la estación sin tener que escribirlos sobre la línea de comandos, por ejemplo, para tomar fotografías. El diseño de esta placa se muestra en la Figura 75.

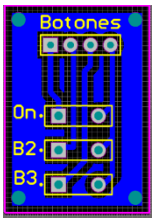

Figura 75 Placa de botones

G. Encapsulado de la Estación

Con el fin de agrupar todos los elementos de la estación (Raspberry Pi, módulos de sensores, placas electrónicas, cámara, batería, módulo GPS y cables), se construyó un encapsulado en acrílico de 3mm como se lo observa en la Figura 76. En la Figura 77 se muestran los planos de este encapsulado.

Por motivos de visualización se colaron solo las principales cotas, las cuales se encuentran en milímetros.

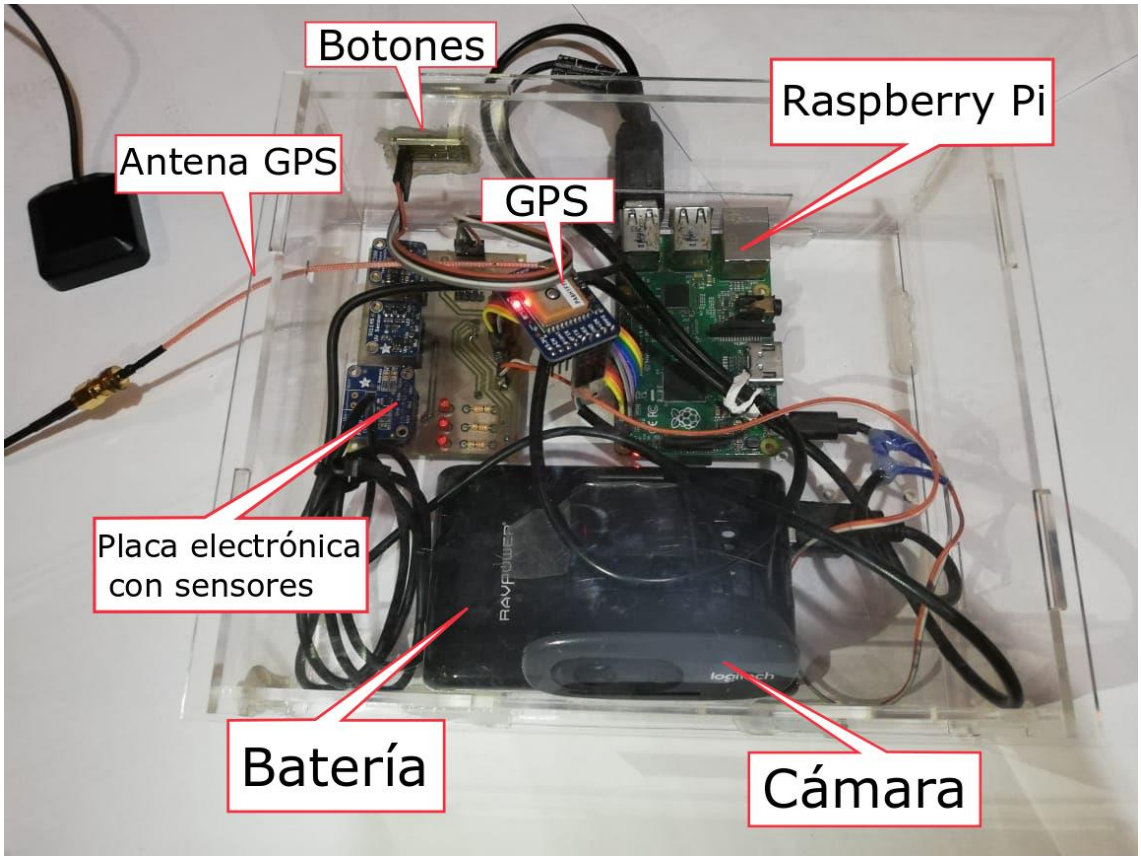

Figura 76 Ensamble de la estación

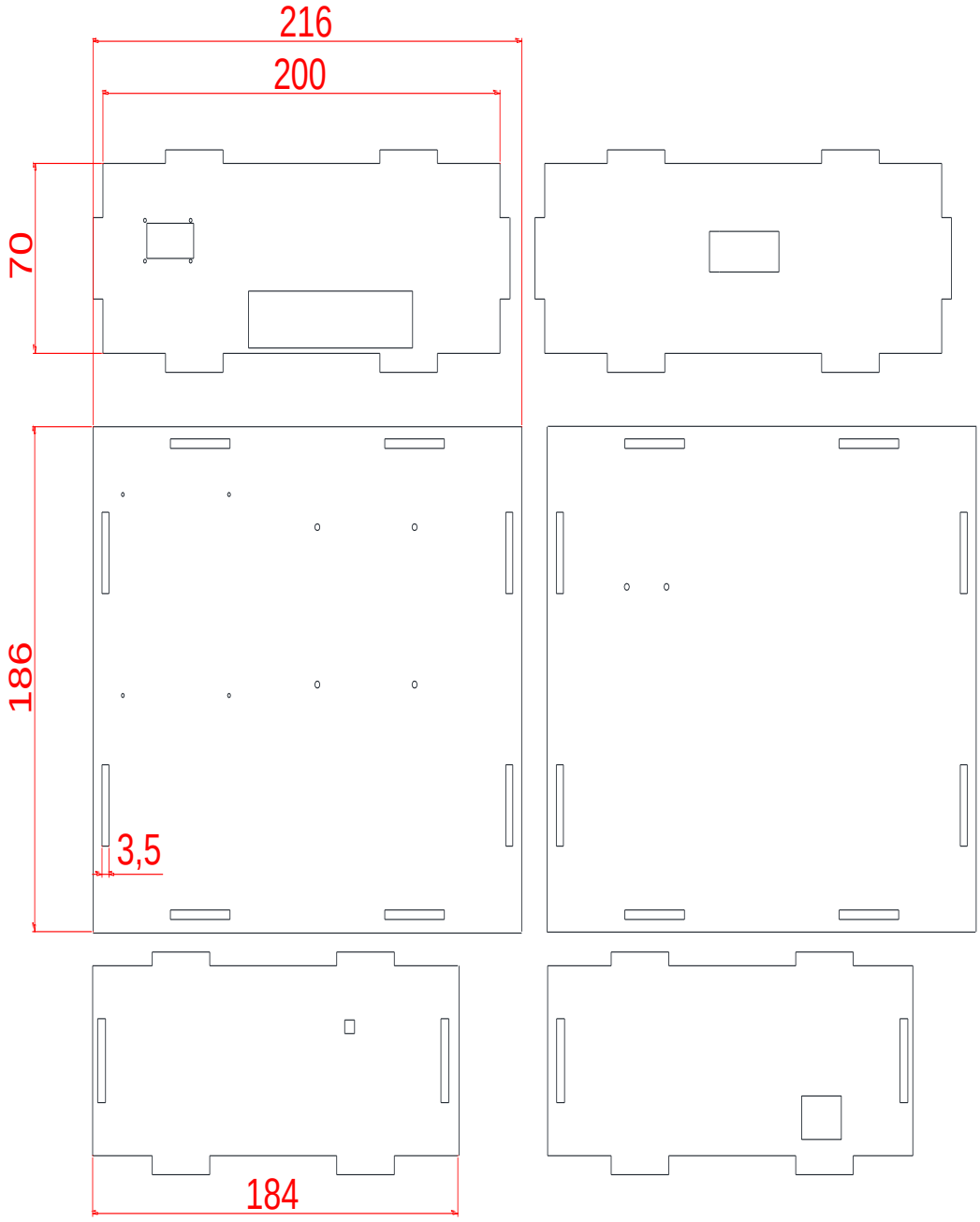

Figura 77 Encapsulado de la estación

H. Código de Segmentación con GMM

A continuación, se detalla el código desarrollado para la segmentación de las imágenes mediante el algoritmo GMM

```
#Librerías necesarias
import numpy as np
from sklearn.mixture import GaussianMixture
import cv2
import time

#####DATOS DE ENTRADA###########
n_clusters=11
```

```
#Definimos la imagen de entrada para ajustar el modelo
X = cv2.imread('ajuste.jpg')
#imagen en la que se realizará la predicción
X2= cv2.imread('test.jpg')
#Capturamos tiempo de ejecución
tiempo=time.strftime("%H%M%S")
################################################
################################################
#cambiamos el espacio de color a LAB
lab = cv2.cvtColor(X, cv2.COLOR_BGR2LAB)
#Se colocan los pixeles de forma plana en un array
#Cada columna será un color(dimensión)
a,b,c=X.shape#a+b es el número de pixeles, c las
dimensiones(componentes LAB)
ent=lab.reshape(a*b,c)

#####AJUSTE DEL MODELO##########
#Se ajusta los datos de entrada a los clusters deseados
GMM = GaussianMixture(n_components=n_clusters).fit(ent)
########################################
#### Predicción de la imagen#######
########################################
lab = cv2.cvtColor(X2, cv2.COLOR_BGR2LAB)
a,b,c=X2.shape
ent=lab.reshape(a*b,c)
#en clusters se guarda el número del cluster en el cual
#es más probable que pertenezca el pixel
clusters=np.array([],dtype=int)

#Se hace una predicción de cluster con cada pixel
for i in range(a*b):
#prediction devuelve un arreglo con la probabilidad de que el pixel
este en cada cluster
    prediction=GMM.predict_proba(ent[i].reshape(1,3))
#Se elige el cluster con mayor probabilidad
    pos=np.argmax(prediction)
    clusters=np.append(clusters,pos)

########################################
########################################
########################################
#Para saber en que cluster está el fruto
#muestra de un punto rojizo en LAB
muestraR=np.array([ 75, 151, 136])
pred=GMM.predict_proba(muestraR.reshape(1,3))
frutos=np.argmax(pred)


#Selec es el número de cluster seleccionado
selec=frutos
#binaria guardará los pixeles en binario
binaria = np.copy(X2[:,:,2].flatten())
#pinta de negro los elementos del cluster
for i in range(a*b):
    if clusters[i]==selec:
        binaria[i]=0
        #binaria=np.append(binaria,0)
    else:
```

```
        #binaria=np.append(binaria,254)
        binaria[i]=255#blanco


#Se guarda la imagen
cv2.imwrite('binaria.png',binaria)

######Elementos#####
#Se guardara los elementos con los colores normales
selec=frutos
#elementos = np.copy(X[:,:,2].flatten())
original=X2.reshape(a*b,c)
elementos = np.copy(original)
#pinta de negro los elementos del cluster
for i in range(a*b):
    if clusters[i]==selec:
        elementos[i]=original[i]
        #binaria=np.append(binaria,0)
    else:
        #binaria=np.append(binaria,254)
        elementos[i]=np.array([255,255,255])
        #elementos[i]=muestraR


# Se guarda la imagen
elementos=elementos.reshape(a,b,c)
cv2.imwrite("elementos"+tiempo+".png",elementos)
################################################

#Operaciones Morfológicas

#####Entrada de datos##########
img =binaria
#Definicion del kernel
#kernel=np.array([[0,1,0],[1,1,1],[0,1,0]],np.uint8)
kernel=np.array([[0,1,1,1,0],[1,1,1,1,1],[1,1,1,1,1],[1,1,1,1,1],[0
,1,1,1,0]],np.uint8)

##Closing###########
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
cv2.imwrite("closing"+tiempo+".png",closing)


##########ETIQUETAS EN LOS OBJETOS####################
#convertimos en binaria(0 o 1)
#Se usa THRESH_BINARY_INV para que el fondo sea cero y los objetos
255
entrada = cv2.threshold(closing, 127, 255,
cv2.THRESH_BINARY_INV)[1]
# La conectividad puede ser 4 u 8
conectividad = 4
# Algoritmo de Labeling Connected components con estadisticas
salida = cv2.connectedComponentsWithStats(entrada,conectividad,
cv2.CV_32S)
# La salida se divide en 4 parametros:
# Numero de etiquetas(labels)
num_labels = salida[0]
# Imagen con etiquetas numeradas
```

```
labels = salida[1]
# Estadisticas(leftmost, topmost, weight, height, area)
stats = salida[2]
# centroides
centroids = salida[3]


#Conteo de frutos: Conteo de todas las manchas, y aquellas muy
grandes
#se las divide sobre el tamaño promedio del objeto y se toma el
entero
#El fondo de la imagen será el que más pixeles tenga
fondo=np.amax(stats[:,4])
#tamaño de un objeto promedio
prom=120
cont=0
for i in range(num_labels):
      if stats[i][4]<fondo:
            if stats[i][4]>=prom:
                  temp=int(stats[i][4]/prom)
                  cont=cont+temp
            else:
                  cont=cont+1

#-----------------------------------------------------------------
####Calculo del color promedio###########3
ImgColor = cv2.imread("elementos"+tiempo+".png")
vectorColor=np.array([])
####Color Promedio####
a,b=closing.shape
for i in range(a):
      for j in range(b):
            if closing[i][j]==0:
                  vectorColor=np.append(vectorColor,ImgColor[i][j])

c=vectorColor.shape
vectorColor=vectorColor.reshape(int(c[0]/3),3)
ColorPromedio=np.mean(vectorColor,0)
ColorPromedio=ColorPromedio.astype(int)

##############################
#######Estadisticas#########
#Se obtiene el valor de area del mayor objeto, el menor y el
promedio
fondoIndex=np.argmax(stats[:,4])
statsSinFondo = np.delete(stats,fondoIndex,0)
objetoMin=np.amin(statsSinFondo[:,4])
objetoMax=np.amax(statsSinFondo[:,4])
objetoProm=int(np.mean(statsSinFondo[:,4]))

#####REPORTE#########
print("#######REPORTE DE ESTADÍSTICAS######")
print("Número de manchas:                 "+str(num_labels-1))
print("Número estimado de frutos:         "+str(cont))
print("Color promedio de los frutos(BGR): "+str(ColorPromedio))
print("Tamaño objeto mínimo(pixeles):     "+str(objetoMin))
print("Tamaño objeto máximo(pixeles):     "+str(objetoMax))
print("Tamaño objeto promedio(pixeles):   "+str(objetoProm))
```

```
print("####################################")
################################
#######Cuadrados sobre objetos
#original = cv2.imread('ImagTest.png')
#aqui se modifica el vector X2
for i in range(num_labels-1):
      cv2.rectangle(X2, (statsSinFondo[i][0], statsSinFondo[i][1]),
(statsSinFondo[i][0]+statsSinFondo[i][2],
statsSinFondo[i][1]+statsSinFondo[i][3]), (0, 255, 0), 1)

cv2.imwrite("cuadrados"+tiempo+".jpg",X2)
#cv2.imshow('cuadrados',original)
#cv2.waitKey(0)
```

I. Código de Segmentación con Thresholding

A continuación, se detalla el código desarrollado para la segmentación de las imágenes mediante el algoritmo *Thresholding.*

```
import numpy as np
from sklearn.mixture import GaussianMixture
import cv2
import time

#####DATOS DE ENTRADA###########
#Definimos la imagen de entrada
X = cv2.imread('test.jpg')
tiempo=time.strftime("%H%M%S")
################################################
################################################
#cambiamos el espacio de color a HSV
hsv = cv2.cvtColor(X, cv2.COLOR_BGR2HSV)

#Definimos la máscara que se usará
#Esta máscara está compuesta por 2 sub-máscaras
#sub máscara 1

lower_red = np.array([0,65,70])
upper_red = np.array([20,255,255])
mask1 = cv2.inRange(hsv, lower_red, upper_red)

##sub máscara 2
lower_red = np.array([140,65,70])
upper_red = np.array([180,255,255])
mask2 = cv2.inRange(hsv,lower_red,upper_red)
#Máscara final
mask1 = mask1+mask2

#Invertimos las máscara
mask2 = cv2.bitwise_not(mask1)
res1 = cv2.bitwise_and(X,X,mask=mask1)

bina=np.copy(res1)

a,b,c=res1.shape
for i in range(a):
      for j in range(b):
```

```
            if (res1[i][j][0]==0) and (res1[i][j][1]==0) and
(res1[i][j][2]==0):
                    res1[i][j][0]=255
                    res1[i][j][1]=255
                    res1[i][j][2]=255


cv2.imwrite("elementos"+tiempo+".png",res1)

binaria = cv2.bitwise_not(bina,mask=mask2)
binaria=binaria[:,:,0]
#-----------------------------------------------------------------
#Operaciones Morfológicas
#####Entrada de datos##########
#img = cv2.imread('binaria.png',0)
img =binaria
#Definicion del kernel
kernel=np.array([[0,1,0],[1,1,1],[0,1,0]],np.uint8)
#kernel=np.array([[0,1,1,1,0],[1,1,1,1,1],[1,1,1,1,1],[1,1,1,1,1],[
0,1,1,1,0]],np.uint8)

##Closing###########
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
cv2.imwrite("closing"+tiempo+".png",closing)

##########ETIQUETAS EN LOS OBJETOS####################
#convertimos en binaria(0 o 1)
#Se usa THRESH_BINARY_INV para que el fondo sea cero y los objetos
255
entrada = cv2.threshold(closing, 127, 255,
cv2.THRESH_BINARY_INV)[1]
# La conectividad puede ser 4 u 8
conectividad = 4
# Algoritmo de Labeling Connected components con estadisticas
salida = cv2.connectedComponentsWithStats(entrada,conectividad,
cv2.CV_32S)
# La salida se divide en 4 parámetros:
# Numero de etiquetas(labels)
num_labels = salida[0]
# Imagen con etiquetas numeradas
labels = salida[1]
# Estadisticas(leftmost, topmost, weight, height, area)
stats = salida[2]
# centroides
centroids = salida[3]


#Conteo de frutos: Conteo de todas las manchas, y aquellas muy
grandes
#se las divide sobre el tamaño promedio del objeto y se toma el
entero
#El fondo de la imagen sera el que mas pixeles tenga
fondo=np.amax(stats[:,4])
#tamaño de un objeto promedio
prom=120
cont=0
for i in range(num_labels):
      if stats[i][4]<fondo:
```

```
            if stats[i][4]>=prom:
                  temp=int(stats[i][4]/prom)
                  cont=cont+temp
            else:
                  cont=cont+1

#-----------------------------------------------------------------
####Calculo del color promedio###########3
ImgColor = cv2.imread("elementos"+tiempo+".png")
vectorColor=np.array([])
####Color Promedio####
a,b=closing.shape
for i in range(a):
      for j in range(b):
            if closing[i][j]==0:
                  vectorColor=np.append(vectorColor,ImgColor[i][j])

c=vectorColor.shape
vectorColor=vectorColor.reshape(int(c[0]/3),3)
ColorPromedio=np.mean(vectorColor,0)
ColorPromedio=ColorPromedio.astype(int)

##############################
#######Estadisticas#########
#Se obtiene el valor de área del mayor objeto, el menor y el
promedio
fondoIndex=np.argmax(stats[:,4])
statsSinFondo = np.delete(stats,fondoIndex,0)
objetoMin=np.amin(statsSinFondo[:,4])
objetoMax=np.amax(statsSinFondo[:,4])
objetoProm=int(np.mean(statsSinFondo[:,4]))

#####REPORTE#########
print("#######REPORTE DE ESTADÍSTICAS######")
print("Número de manchas:                  "+str(num_labels-1))
print("Número estimado de frutos:          "+str(cont))
print("Color promedio de los frutos(BGR): "+str(ColorPromedio))
print("Tamaño objeto mínimo(pixeles):      "+str(objetoMin))
print("Tamaño objeto máximo(pixeles):      "+str(objetoMax))
print("Tamaño objeto promedio(pixeles):    "+str(objetoProm))
print("####################################")
##############################
#######Cuadrados sobre objetos
#original = cv2.imread('ImagTest.png')
#aqui se modifica el vector X2
for i in range(num_labels-1):
      cv2.rectangle(X, (statsSinFondo[i][0], statsSinFondo[i][1]),
(statsSinFondo[i][0]+statsSinFondo[i][2],
statsSinFondo[i][1]+statsSinFondo[i][3]), (0, 255, 0), 1)

cv2.imwrite("cuadrados"+tiempo+".jpg",X)
```