TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Nuevo León

División de Estudios de Posgrado e Investigación

TESIS

Inspección óptica automatizada de tarjetas electrónicas y conteo de tornillos por visión artificial

que presenta

Ing. Alejandro Martínez Lara

NC. G18481536

para obtener el grado de

MAESTRÍA EN INGENIERÍA

Director de tesis

Dr. Miguel Ángel Ochoa Villegas

Ciudad Guadalupe, Nuevo León. Enero 2021

"2021, Año de la Independencia"

Aceptación de documento de Tesis

Guadalupe Nuevo León, 11/enero/2021

DR. MARIO CÉSAR OSORIO ABRAHAM
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
PRESENTE:

El Comité Revisor de Tesis nos es grato comunicarle que, conforme a los lineamientos para la obtención del grado de Maestría en Ingeniería de este Instituto, y después de haber sometido a revisión académica el proyecto de Tesis titulado: **"INSPECCIÓN ÓPTICA AUTOMATIZADA DE TARJETAS ELECTRÓNICAS Y CONTEO DE TORNILLOS POR VISIÓN ARTIFICIAL",** realizada por la **ING. ALEJANDRO MARTÍNEZ LARA,** No. De Control: **G18481536,** dirigida por el **DR. JUAN ANTONIO ROJAS ESTRADA (*In memoriam*) y DR. MIGUEL ANGEL OCHOA VILLEGAS,** y habiendo realizado las correcciones que le fueron indicadas, acordamos **ACEPTAR** el documento final de proyecto de Tesis, así mismo le solicitamos tenga a bien extender el correspondiente oficio de autorización de impresión.

Sin otro particular, agradecemos la atención.

A T E N T A M E N T E
Excelencia en Educación Tecnológica-
"CIENCIA Y TECNOLOGÍA AL SERVICIO DEL HOMBRE"

DIRECTOR DE TESIS

DR. MIGUEL ANGEL OCHOA VILLEGAS
DOCTOR EN DOCTORADO EN TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES
CÉDULA: 9386150

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA — TECNÓLOGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE NUEVO LEÓN
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

REVISOR

DR. RENE SANJUAN GALINDO
DOCTOR EN INGENIERÍA
CÉDULA: 7685678

REVISOR

DR. MARIO CESAR OSORIO ABRAHAM
DOCTOR EN DOCTORADO EN INGENIERÍA FÍSICA INDUSTRIAL
CÉDULA: 7576591

C.c.p.- Expediente

"2021, Año de la Independencia"

Aceptación de impresión de Tesis

Guadalupe Nuevo León, 11/enero/2021

ING. ALEJANDRO MARTÍNEZ LARA,
NO. DE CONTROL: G18481536
ESTUDIANTE DE LA MAESTRÍA EN INGENIERÍA
PRESENTE:

Después de haber atendido las recomendaciones sugeridas por la Comisión del Consejo de Posgrado de Ingeniería, en relación a su trabajo de Proyecto de Tesis, cuyo título es: **"INSPECCIÓN ÓPTICA AUTOMATIZADA DE TARJETAS ELECTRÓNICAS Y CONTEO DE TORNILLOS POR VISIÓN ARTIFICIAL"**, me permito comunicarle que, conforme a los Lineamientos para la Operación de los Estudios de Posgrado, se le concede la autorización para que proceda con la impresión de su proyecto de Tesis.

Sin otro particular, agradecemos la atención.

A T E N T A M E N T E
Excelencia en Educación Tecnológica·
"CIENCIA Y TECNOLOGÍA AL SERVICIO DEL HOMBRE"

DR. MARIO CÉSAR OSORIO ABRAHAM
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO.

C.c.p.- Expediente

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE NUEVO LEÓN
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"2021, Año de la Independencia"

AUTORIZACIÓN PARA PRESENTAR EXAMEN DE GRADO DE MAESTRÍA

Guadalupe Nuevo León, 11/enero/2021

ING. ALEJANDRO MARTÍNEZ LARA,
NO. DE CONTROL: G18481536
ESTUDIANTE DE LA MAESTRÍA EN INGENIERÍA
PRESENTE:

Por medio del presente me es grato comunicarle que SE AUTORIZA la presentación del examen de Maestría en Ingeniería, toda vez que ha cubiertos los requisitos necesarios de Proyecto de Tesis, cuyo título es: **"INSPECCIÓN ÓPTICA AUTOMATIZADA DE TARJETAS ELECTRÓNICAS Y CONTEO DE TORNILLOS POR VISIÓN ARTIFICIAL".**

Aprovecho la ocasión para desearle el mejor de los éxitos en su examen, así como en su vida profesional, y agradecerle la confianza depositada en nuestra institución para la realización de sus estudios de Posgrado.

A T E N T A M E N T E
Excelencia en Educación Tecnológica
"CIENCIA Y TECNOLOGÍA AL SERVICIO DEL HOMBRE"

DR. MIGUEL ÁNGEL OCHOA VILLEGAS
SUBDIRECTOR ACADÉMICO

EDUCACIÓN TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE NUEVO LEÓN
SUBDIRECCIÓN ACADÉMICA

C.c.p.- Expediente

turnitin

Tesis - Alejandro_Martinez.pdf
Dec 19, 2020
20960 palabras/104503 caracteres

Tesis - Alejandro_Martinez.pdf

Resumen de fuentes

8%

SIMILITUD GENERAL

1	Escuela Politecnica Nacional on 2020-03-10 TRABAJOS ENTREGADOS	<1%
2	Curtin University of Technology on 2020-10-07 TRABAJOS ENTREGADOS	<1%
3	Universidad Internacional de la Rioja on 2020-10-13 TRABAJOS ENTREGADOS	<1%
4	www.readbag.com INTERNET	<1%
5	www.springerprofessional.de INTERNET	<1%
6	Escuela Politecnica Nacional on 2016-10-04 TRABAJOS ENTREGADOS	<1%
7	Johannes Richter, Detlef Streitferdt. "Modern Architecture for Deep Learning-Based Automatic Optical Inspection", 2019 IEEE 43rd Ann... CROSSREF	<1%
8	sedici.unlp.edu.ar INTERNET	<1%
9	www.ustabuca.edu.co INTERNET	<1%
10	Politécnico Colombiano Jaime Isaza Cadavid on 2017-06-07 TRABAJOS ENTREGADOS	<1%
11	Universidad Santo Tomas on 2015-10-22 TRABAJOS ENTREGADOS	<1%
12	d-nb.info INTERNET	<1%
13	oa.upm.es INTERNET	<1%
14	Hangyu Qi, Tianhua Xu, Guang Wang, Yu Cheng, Cong Chen. "MYOLOv3-Tiny: A new convolutional neural network architecture for real-t... CROSSREF	<1%
15	euler.math.nara-wu.ac.jp INTERNET	<1%
16	Technical University of Cluj-Napoca on 2020-07-13 TRABAJOS ENTREGADOS	<1%
17	link.springer.com INTERNET	<1%

Inspección óptica automatizada de tarjetas electrónicas y conteo de tornillos por visión artificial

Presenta
Alejandro Martínez Lara

Tesis para el título de
Maestro en Ingeniería

División de Estudios de Posgrado e Investigación
Instituto Tecnológico de Nuevo León
México
12 Diciembre 2020

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Nuevo León

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Guadalupe Nuevo León, 31/enero/2021
Asunto: Cesión de derechos.

DR. MARIO ALBERTO MARTÍNEZ HERNÁNDEZ
INSTITUTO TECNOLÓGICO DE NUEVO LEÓN
DIRECTOR
PRESENTE

Hago manifesta lo

CESIÓN DE DERECHOS

De la tesis **INSPECCIÓN ÓPTICA AUTOMATIZADA DE TARJETAS ELECTRÓNICAS Y CONTEO DE TORNILLOS POR VISIÓN ARTIFICIAL** que fue dirigida por el **Dr. Juan Antonio Rojas Estrada (In Memoriam)** y el Codirector **Dr. Miguel Ángel Ochoa Villegas**.

En cumplimiento a los requisitos de obtención del grado que señala el numeral 2.15.3 de los Lineamientos para la Operación de los Estudios de Posgrado en el Tecnológico Nacional de México.

ATENTAMENTE
Excelencia en Educación Tecnológica®

Ing. Alejandro Martínez Lara
CANDIDATO AL GRADO DE MAESTRO DEL PROGRAMA MAESTRÍA EN INGENIERÍA

c.c.p. Archivo

CATERPILLAR®

Tecnología Modificada, S.A. de C.V.
Ave. Transformación # 545
Entre Calle Industrias y Ave. De la República
Parque Industrial FINSA
Nuevo Laredo, Tamps. C.P. 88275
México

Mailing address: P.O. Box 3599
Laredo, Tx. 78044-3599
USA

Nuevo Laredo, Tamaulipas a 4 de enero de 2021

Asunto: Carta de Impacto / beneficio / utilización

A quien corresponda

Por este medio se hace constar que el proyecto **INSPECCIÓN ÓPTICA AUTOMATIZADA DE TARJETAS ELECTRÓNICAS Y CONTEO DE TORNILLOS POR VISIÓN ARTIFICIAL** desarrollado por **ALEJANDRO MARTÍNEZ LARA** del Instituto Tecnológico de Nuevo León, programa educativo Maestría en Ingeniería, en el periodo agosto 2018 a diciembre 2020 tuvo la aportación dentro de Tecnología Modificada SA de CV al implementar un sistema automatizado de inspección óptica para las tarjetas electrónicas en el negocio de ECU. Este sistema es capaz de detectar hasta el 97% de defectos como soldadura insuficiente, puentes de soldadura, componentes ausentes/presentes y alineación de circuitos integrados. Además, también se desarrolló un prototipo de bajo costo para resolver la problemática de las unidades que se envían a campo sin todos los tornillos instalados. Este prototipo se desarrolló utilizando técnicas de visión artificial, logrando un porcentaje de detección de 98%, lo cual ayudará a eliminar este tipo de defectos, además de que es un sistema adaptable a características que se requieran inspeccionar en este o en otros productos del catálogo de la empresa. Ambas soluciones se atacaron debido a las oportunidades detectadas en el pago de garantías.

Para los efectos que tenga lugar.

TECNOLOGÍA MODIFICADA
DEPTO. RECURSOS HUMANOS

Claudia Contreras Faz
Gerente de Recursos Humanos

A Sara y a César

Índice general

Índice de figuras

Índice de cuadros

Siglas

AOI Inspección Óptica Automatizada.

CPU Unidad Central de Procesamiento.

DWT Transformada Discreta de Ondícula.

ECM Engine Control Module.

ECU Engine Control Unit.

FGPA Matriz de Puertas Lógicas Programable en Campo.

GPU Unidad de Procesamiento Gráfico.

IDE Entorno de Desarrollo Integrado.

NPI New Product Introduction.

NPU Uso de Partes Nuevas.

OCR Reconocimiento Óptico de Caracteres.

OCV Verificación Óptica de Caracteres.

OpenCV Open Source Computer Vision Library.

PCB Tarjetas de Circuito Impreso.

SDM Dispositivos de Montaje Superficial.

Agradecimientos

Este trabajo representa el esfuerzo y tiempo dedicado al seguimiento de mi formación como ingeniero y profesionista. No pudo haber sido posible sin el apoyo de personas clave, a quienes quiero agradecer. A Alberto Reyna y Sicilia Ribota por el continuo soporte y motivación para seguir mejorando y ampliando mis habilidades. Al Dr. Juan A. Rojas (QEPD), Dr. Miguel Ochoa y Dr. Rene Sanjuan por su tiempo, disposición y dedicación a lo largo de estos años, tanto en las clases impartidas, como en los seminarios de tesis y en la gestión del programa de Posgrado.

Resumen

El ensamble manual de las tarjetas electrónicas (PCB) para unidades de control electrónico (ECU) re-manufacturadas carece de algún control para el conteo de los tornillos instalados, lo cual ha ocasionado defectos que no son detectados antes de que el producto final sea enviado al cliente. Adicionalmente, se cuenta con una inspección visual de los componentes de la tarjeta realizada por un operador con un microscopio digital, sin algún control automático y con aceptación que depende del criterio del operador. Este proyecto propone un sistema de inspección óptica del total de tornillos en las tarjetas después de su ensamble, mediante la implementación de un sistema de visión artificial desarrollado en Python, utilizando la librería OpenCV. La ejecución del código se realiza en un dispositivo Raspberry Pi. Con el sistema implementado se logra un porcentaje de reconocimiento de 99.09 % de tornillos instalados en las tarjetas. También se propone la implementación de un sistema de Inspección Óptica Automatizada (AOI) para la detección de defectos en los componentes de las tarjetas electrónicas. Este sistema muestra un porcentaje de detección de 96.77 %.

Abstract

The manual assembly of the printed circuit boards (PCB) for the re-manufactured Electronic Control Units (ECU) lacks a controlled installed screw count. This has caused defects which are not detected before the product is shipped to the final customer. Additionally, there is a visual inspection for the components on the boards that is performed by an operator using only a digital microscope, without any control and open to the operator's acceptance criteria. This project proposes an optical inspection system to count the total of installed screws after the board assembly, by implementing a computer vision system developed on Python, using the OpenCV library. The code for this application will be executed by a Raspberry Pi device. An installed screw recognition rate of 99.09% is achieved with this system. Also proposed is an Automated Optical Inspection (AOI) system to detect component defects on the boards. This system showed a detection rate of 96.77%.

Capítulo 1

Introducción

La visión artificial se ha utilizado en muchas aplicaciones. Un caso particular es la inspección y clasificación de características de objetos, especialmente para reemplazar la visión humana, ya que una máquina puede ser más confiable y consistente a lo largo de un día (en un ambiente de manufactura).

Actualmente, la mayoría de los sistemas de inspección óptica se basan en redes neuronales que no sólo requieren de una gran cantidad de imágenes para ser entrenadas, sino que además son de alto costo [1]. Estos sistemas requieren gran poder de procesamiento para alcanzar tiempos de ciclo aceptables, e incluso en aplicaciones que cuentan con computadoras sobrecargadas, los resultados se obtienen de manera lenta y el tiempo de desarrollo es largo.

Con mayor rapidez la tecnología se vuelve más accesible y en el caso del módulo Raspberry Pi, las comunidades de desarrolladores del mundo se apoyan para acelerar los tiempos de creación de nuevas aplicaciones, así como para explotar el máximo poder que tiene esta plataforma. Raspberry Pi es una plataforma que presenta muchas ventajas, entre ellas la seguridad (al estar construido sobre un sistema operativo basado en UNIX), el bajo costo y el poder moderado que presenta.

La remanufactura de tarjetas electrónicas para Módulos de Control Electrónicos (ECM) es un proceso manual. Inicialmente son des-ensambladas y reparadas a lo largo de una serie de operaciones. Sin embargo, dos de los procesos más críticos para la operación son la actualización o reparación y el ensamble de las tarjetas electrónicas, ya que es la última oportunidad para poder detectar defectos en ellas.

Existen dos problemáticas que se requieren atacar para reducir la cantidad de defectos. Primeramente, una actualización o reparación incorrecta de las tarjetas electrónicas. Esto se manifiesta en componentes electrónicos ausentes o presentes, puentes de soldadura, soldadura insuficiente o fabricante equivocado. En segundo lugar, la omisión de algún tornillo durante el ensamble final de la tarjeta en la carcasa del producto. El cliente final ha recibido unidades que presentan alguna o ambas condiciones, las cuales, al ser expuestas al calor y vibraciones de motores, presentan fallas funcionales debido a que la integridad y estabilidad de la tarjeta electrónica se encuentra comprometida, generando reclamos de garantía por parte del cliente final y generando un impacto negativo en la reputación de la marca.

La operación de inspección de componentes de la tarjeta es un proceso realizado de manera manual, utilizando un microscopio digital que es enfocado y desplazado sobre el área de la PCB (Tarjeta de Circuito Impreso). Actualmente sólo se inspecciona menos del 1 % de la cantidad total de componentes de la tarjeta debido al tiempo

que se requiere para que una persona inspeccione un porcentaje mayor. Para resolver este problema, se propone la implementación de una máquina de *Automated Optical Inspection* (AOI) de la marca Nordson Yestech para inspeccionar los componentes instalados en la tarjeta electrónica. Esta inspección abarcará tanto los componentes actualizados o reparados, como el resto de los componentes de la tarjeta. Los defectos que tienen más relevancia son: componentes ausentes, puentes de soldadura, falta de soldadura y componentes presentes donde deberían de ser ausentes.

El ensamble de estos módulos de control también es un proceso manual y no se cuenta con un sistema de control de ensamble para monitorear diversos pasos críticos como la instalación del número correcto de tornillos. Se propone la implementación de un sistema de inspección con visión artificial para el conteo automatizado de tornillos instalados en una unidad al finalizar el ensamble de la misma. Este sistema será desarrollado en la plataforma Raspberry Pi usando el lenguaje de programación Python y una implementación de la librería OpenCV (*Open Source Computer Vision Library*). Las ventajas esperadas son que el sistema autónomo sea de bajo costo, con mantenimiento mínimo y que también sea escalable y adaptable. El negocio de remanufactura de Unidades de Control Electrónico (ECU) tiene un volumen de producción considerablemente bajo en comparación con la manufactura de producto nuevo. Es por esto, que el tiempo de ejecución del código en una Raspberry Pi es suficiente para la aplicación.

Considerando estas dos problemáticas, se revisó la literatura para identificar opciones tecnológicas en cuanto a sistemas de AOI, así como de inspección automatizada por visión artificial para componentes mecánicos como tornillos, además de implementaciones ya realizadas con la librería OpenCV en Python.

Zhao, Cheng y Jin [2] proponen un sistema de AOI para componentes SMD (Dispositivos de Montaje Superficial) basado en hardware y software de National Instruments. En su diseño integran, además, un sistema de movimiento de las tarjetas electrónicas usando servomotores con movimiento en dos ejes; esto para tener la capacidad de alinear correctamente la unidad bajo inspección. Utilizan la suite de desarrollo para aplicaciones de visión NI Vision Development Module (VDM) y el lenguaje de programación C/C++ en la IDE (Entorno de Desarrollo Integrado) LabWindows/CVI. Logran la detección de errores en la posición y rotación de los componentes SMD mediante el uso de algoritmos de ecualización, remuestreo, coincidencias geométricas y de patrones, sin embargo, el diseño depende del software y hardware propietario de National Instruments y está limitado a componentes SMD, así como a las características de posición y rotación.

Pramudita y Hariadi [3] desarrollan técnicas para mitigar errores por cambios de rotación de tarjetas electrónicas en un sistema de AOI. Este sistema es basado en una Placa Computadora (SBC, *Single Board Computer*) Raspberry Pi y utiliza la librería OpenCV para la implementación de algoritmos basados en la determinación de la distancia Euclidiana entre objetos. Esta implementación es únicamente para detectar estos cambios de rotación en las tarjetas, sin embargo, demuestra el poder y utilidad para este tipo de aplicaciones de Raspberry Pi y OpenCV.

Lin, Chiang y Hsu [4] realizan un sistema para la detección de capacitores en PCB basado en el algoritmo YOLO, el cual se basa en un conjunto de técnicas de redes neuronales que se encargar de identificar las probabilidades de la presencia de un objeto en una imagen. Este algoritmo, como su nombre lo explica (You Only Look Once, Sólo Mirar una Vez), tiene como objetivo acelerar la clasificación de

objetos al solo analizar la imagen una vez. Requiere de entrenamiento y y requiere de poder de cómputo moderado para entregar resultados en tiempos de ejecución aceptables — sin embargo, presenta oportunidades para la identificación correcta de objetos pequeños y de grupos de objetos en una sola imagen [5]. La implementación de YOLO para la detección de capacitores es exitosa y logra tiempos de ejecución en el rango de 208 a 711 ms. Sin embargo, esta aplicación es específica para la detección de capacitores DIP (Dual Inline Package) y está limitada a este tipo de componente.

Raihan y Ce [6] trabajaron en la detección de defectos en PCB utilizando OpenCV y el método de substracción de imágenes. En su trabajo describen el preprocesamiento que aplicaron a sus imágenes antes de iniciar el proceso de identificación de defectos, como la aplicación de filtros y thresholding. En esta aplicación de inspecciona una cantidad más amplia de defectos que en las mencionadas anteriormente, sin embargo, la inspección es realizada a la tarjeta electrónica sin componentes instalados. Hacen pruebas con imágenes de baja y alta resolución, logrando mejores resultados con las últimas — 80 % de precisión en 2.68 segundos.

Para la aplicación de detección de tornillos en el ensamble de las PCB, también se revisó literatura. Galan [7] propone un sistema de visión para identificar y clasificar defectos en superficies de carcasas metálicas. Sin embargo, el hardware utilizado para esta aplicación es caro y elaborado. Esto es debido a que la aplicación busca identificar características de tamaños reducidos en tiempos bajos de procesamiento, los cuales no son requerimientos críticos para el propósito del presente trabajo. El autor reportó el uso de OpenCV, la cual es una librería muy popular y completa para desarrollo de aplicaciones de visión artificial, pero finalmente fue desarrollada en Ubuntu Linux. El sistema binariza las imágenes, es decir, las convierte a escala de grises y aplica filtros y algoritmos para convertir las imágenes a sólo dos tonos: blanco o negro. Los resultados fueron aceptables, con tiempos de procesamiento bajos al utilizar un GPU (Unidad de Procesamiento Gráfico) dedicado en lugar de un CPU (Unidad Central de Procesamiento) y así mantener un error menor al 12 %. Jaffery et al. [8], desarrollaron un sistema para reemplazar las inspecciones realizadas por un operador en vías de ferrocarril para detectar tuercas y tornillos ausentes. En este caso, se utilizó una técnica de reconocimiento de patrones y la DWT (*Discrete Wavelet Transform*). El sistema de procesamiento no fue significativamente poderoso, sin embargo, sí fue mucho más que un sistema de Raspberry Pi. Un enfoque más robusto es propuesto por Johan y Prabuwono [9], el cual utiliza una red neuronal artificial para reconocer tornillos y tuercas. Este sistema utilizó MATLAB para procesar las imágenes. Sin embargo, la inspección en línea requiere que los tiempos de cómputo sean rápidos, ya que los objetos a inspeccionar son transportados mediante una banda. Este sistema no sólo inspeccionaba tornillos y tuercas, sino que además los clasificaba. Los autores lograron una tasa de clasificación correcta del 92 %. En el trabajo de Ruvo *et. al.*, [10] desarrollaron una inspección en tiempo real para detección de tornillos con cabeza hexagonal, utilizando un sistema FPGA (Matriz de Puertas Lógicas Programable en Campo) para realizar el procesamiento de las imágenes, además de un algoritmo predictivo.

Existen ya implementaciones [11] de *template matching* con múltiples patrones de referencia, sin embargo, en general se corren en equipos con grandes capacidades computacionales. En este caso, siendo una aplicación sencilla, se pueden aprovechar las especificaciones de la Raspberry Pi para la implementación de sistemas de bajo costo y que cambian la perspectiva que se tiene actualmente de este tipo de dispositivos,

generalmente asociados a proyectos estudiantiles o de aficionados a la electrónica o informática. También se aprovecha el poder del lenguaje de programación Python. El número de usuarios de este lenguaje de programación ha incrementado en los últimos años debido a su flexibilidad y poder en el desarrollo de librerías específicas que son *open source* [12]. De igual manera, OpenCV ha sido aceptado globalmente como un componente esencial en aplicaciones de desarrollo con visión artificial, no sólo para Python, sino también para otros lenguajes de programación.

La técnica de *template matching* puede hacer operaciones más complejas, adicionales a la detección de objetos en imágenes; también se pueden implementar sistemas de clasificación e incluso generar conjuntos de entrenamiento dinámicos.

Todas estas implementaciones hacen uso de diversos métodos de procesamiento de imágenes, así como de diferentes rangos y tipos de software y hardware. Sin embargo, históricamente, la AOI ha sido desarrollada específicamente para la manufactura de PCB nuevas en líneas automatizadas de ensamble [13]. En el presente trabajo se tratará el tema de la implementación de un sistema de AOI para tarjetas remanufacturadas o retrabajadas de manera manual, lo cual implica variaciones en el proceso, ya que cada persona puede realizar su trabajo de manera diferente, sin embargo, el estándar y parámetros de aceptación son iguales para todas las tarjetas. Se propone además, un sistema de detección de tornillos presentes en el ensamble final de estas tarjetas electrónicas, basado en Raspberry Pi y en la librería de OpenCV.

Una vez realizada la investigación del estado del arte en las áreas de visión artificial, se notó que la mayoría de las aplicaciones desarrolladas se enfocan en un problema muy específico, además de que, en el caso de la AOI, sólo se menciona la manufactura de PCB nuevas, sin tomar en cuenta el retrabajo o reparación de las mismas. Es por esto que se busca la implementación y adecuación de un sistema de AOI para las tarjetas electrónicas remanufacturadas, ya que este sector de la industria tiene cada vez más relevancia para apoyar los esfuerzos intencionales de cuidado del medio ambiente. En cuanto al uso de la Raspberry Pi para la detección de tornillos en procesos de ensamble, se espera darle un giro más industrial al uso de estos dispositivos para explotar al máximo su potencial y demostrar que tienen el poder suficiente para aplicaciones específicas, además de que su bajo costo los hace atractivos. Además, mejorar la calidad del producto y mantener la reputación del producto con el cliente final de estos ECU, ya que se contabilizan 80 unidades en promedio con este tipo de defectos detectados en campo. Internamente, sin embargo, se encuentran 850 defectos anualmente que se detectan antes de que el producto final sea enviado al cliente.

1.1 Objetivos

Estudiando el problema y realizando el trabajo de medición para identificar los problemas, enunciamos los objetivos de la investigación en los siguientes apartados.

1.1.1 Objetivo general

Implementar un sistema de Inspección Óptica Automatizada (AOI) capaz de validar la actualización correcta de la tarjeta electrónica, de acuerdo a los requerimientos de la ingeniería del producto, enfocado en defectos por puente de soldadura, exceso de

soldadura, presencia o ausencia de componentes y ensamble de tornillos en la familia Cat A4:E4.

1.1.2 Objetivos específicos

- Diseñar un proceso automático de carga manual para sustituir la operación actual.
- Identificar los componentes electrónicos críticos en las tarjetas electrónicas.
- Medir el número de componentes electrónicos actualmente inspeccionados.
- Identificar la tecnología disponible para realizar esta inspección visual en tarjetas retrabajadas.
- Desarrollar sistema de visión artificial para la detección de fallas en el retrabajo de los componentes electrónicos y en el ensamble de las tarjetas electrónicas.
- Implementar el sistema de visión artificial en el proceso de remanufactura de tarjetas electrónicas.

1.2 Hipótesis

Tomando en cuenta las preguntas de la investigación y teniendo definidos los objetivos, podemos enunciar la hipótesis como sigue:

> La automatización de la inspección visual en el proceso de remanufactura de tarjetas electrónicas eliminará en un 80 % los defectos por tornillos ausentes, puentes de soldadura, exceso de soldadura y presencia o ausencia de componentes.

Capítulo 2

Marco Teórico

2.1 Visión computacional

Como humanos, interactuamos con el mundo a través de nuestros cinco sentidos. Somos capaces de distinguir el peso o forma de un objeto a través de nuestras manos con el tacto. Con nuestros oídos podemos separar sonidos, tales como de instrumentos musicales, de voces de acuerdo al timbre, tono o intensidad. Mediante el gusto podemos saborear la comida y con nuestra nariz podemos experimentar los diferentes olores. Con nuestros ojos detectamos la luz que nuestro cerebro descifra y conocemos los colores, la opacidad de algún objeto, así como su forma, su tamaño, de qué tipo es, las características físicas del mismo y constantemente tomamos decisiones de acuerdo a lo que vemos.

La visión computacional se refiere a la implementación artificial, a través de algoritmos y cálculos matemáticos, para darle a una máquina o robot la capacidad de observar el mundo a su alrededor, así como tomar decisiones de acuerdo a lo que ve. En corto, de imitar lo más cercanamente posible la visión humana. [14] Esto para diferentes aplicaciones como el rastreo de objetos a lo largo de un plano, la identificación de rostros para métodos de seguridad o la detección de características deseadas en procesos de manufactura, entre un sinfín de aplicaciones que actualmente utilizan la visión computacional.

Al igual que los elementos que influyen en la visión humana, existen elementos análogos en la visión artificial o computacional, los cuales se pueden comparar de la siguiente manera:

- **Ojos - cámara:** el funcionamiento del ojo, artificialmente, se implementa mediante el uso de cámaras, las cuales a su vez hacen uso de diversos tipos de lentes, cada unas con características apropiadas para la aplicación a tratar. Mediante las cámaras es posible darle ojos a una máquina o a un robot.

- **Cerebro - procesador:** la función que realiza el cerebro al interpretar las imágenes que obtiene mediante los ojos, en la visión computacional la realizan los procesadores mediante la implementación de métodos y modelos matemáticos ya estudiados desde hace décadas y que han ido evolucionando para las diversas aplicaciones requeridas, por ejemplo, el Reconocimiento Óptico de Caracteres (OCR), de colores o de sombras.

Al hablar de un sistema de visión artificial, se habla de un conjunto de elementos (generalmente de hardware y software) que se ocupan, no sólo de capturar las imágenes

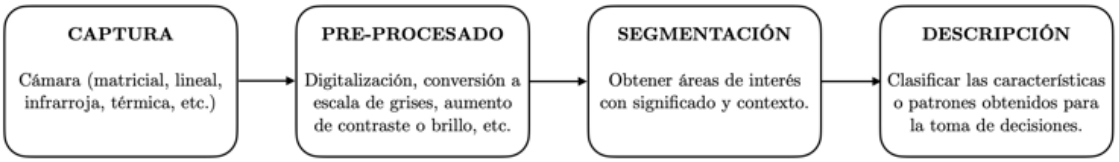


Figura 2.1: Esquema básico de un sistema de visión artificial

mediante sensores y cámaras, sino además de procesarlas y darles significado en el entorno en que se encuentran. Dicho esto, se puede describir la secuencia de procesamiento típica de un sistema de este tipo como sigue: se captura una imagen de un objeto o escenario, se convierte a un formato digital que puede ser entendido y manipulado por el procesador, se realiza el procesamiento de la imagen de acuerdo a lo requerido por la aplicación, se obtienen los resultados requeridos y, yendo un paso más allá, se toman decisiones basadas en dichos resultados. En la Figura 2.1 se muestra un esquema de visión básico con los elementos mencionados.

Los elementos que conforman un sistema de visión artificial varían de acuerdo a la aplicación, sin embargo, en general se pueden mencionar los siguientes:

1. **Iluminación:** la cantidad de luz con las características requeridas por la aplicación. De acuerdo a lo que se busca detectar o inspeccionar es el tipo de iluminación que se utilizará. Esta puede ser de diversos tipos y fuentes.

2. **Lente y sensor:** son los que se encargan de capturar la luz para poder transformarla en imágenes. La lente captura la luz y la envía al sensor, el cual se encarga de convertirla en una imagen digital que puede ser utilizada dentro del sistema.

3. **Procesamiento digital de imágenes:** es la parte de software en la que se realiza la extracción de característica, la segmentación de la imagen, la detección de objetos o de caracteres. Es decir, es la parte en la que se obtiene la información que se busca partiendo de la imagen.

Dentro de un sistema de visión siempre es crucial la selección de la lente, sensor o bien cámara que se utilizará para la aplicación deseada, ya que de esto depende que podamos obtener las características más pequeñas o más grandes de la imagen según lo que busquemos.

2.2 Características de la cámara

Las características de la cámara a utilizar para la aplicación de detección de tornillos en una tarjeta electrónica ensamblada es importante mencionar que la característica que queremos medir es de 8 mm. Las características importantes de la cámara son:

- **Resolución:** se refiere al tamaño de la imagen que será capturada por la cámara y es expresada en un múltiplo de píxeles por columnas y filas. Por ejemplo, la cámara seleccionada tiene un sensor de 8 megapixeles (donde el tamaño de la imagen será de 3,266 x 2450 píxeles).

- **Campo de visión:** es la porción del espacio que puede captar la cámara. [15] Este variará de acuerdo a la lente y la distancia focal.
- **Distancia focal:** es la distancia entre el centro de la lente y el plano donde los objetos se enfocan. [15]
- **Distancia mínima:** es la distancia mínima a la cual un objeto se puede enfocar por la cámara.
- **Profundidad de campo:** es el campo de visión en el que los objetos que se encuentre estarán enfocados.
- **Razón de adquisición de imágenes:** es el número de imágenes por segundo que pueden ser capturadas por el sistema de visión.

2.3 Imagen digital

Para entender el concepto de imagen digital es necesario hablar de representación. Un ejemplo muy simple de representación son los sistemas numéricos – todos son una forma diferente de representar los números. Conforme se vayan estudiando más sistemas numéricos, veremos que existen múltiples representaciones para el mismo fenómeno.[16]

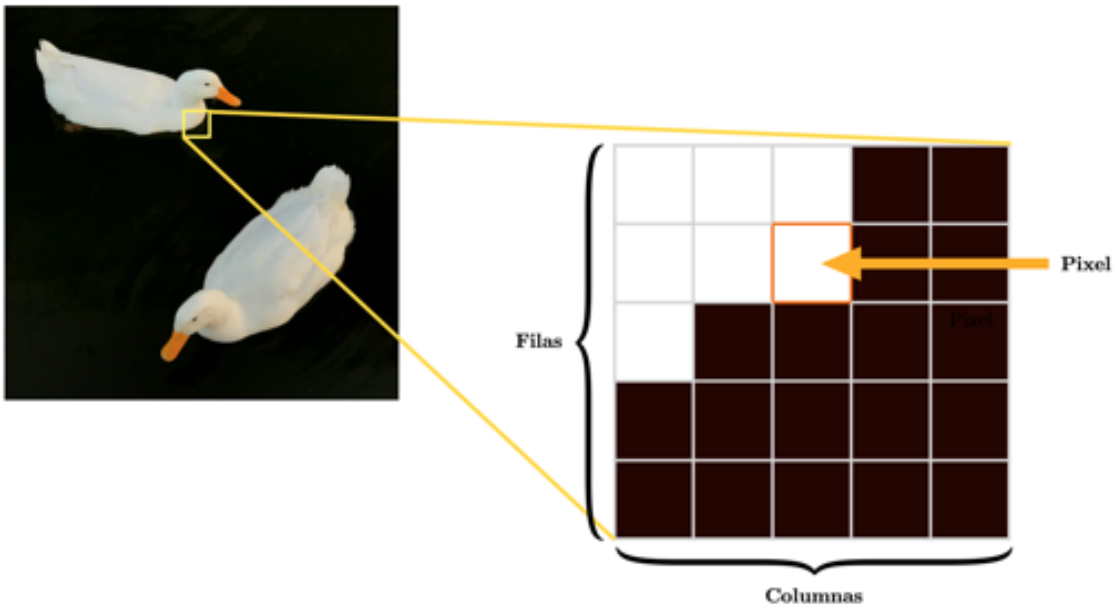


Figura 2.2: Representación de imagen digital en una matriz de nxm pixeles.

En el caso de la representación de una imagen, es la forma en la que una cámara (recordando que el propósito de una cámara es simular el funcionamiento del ojo humano) representa la información que está "viendo", de manera que los sitemas informáticos puedan entenderla y procesarla.

Para el presente estudio se están tomando imágenes 2D, las cuales son representadas por una matriz de puntos, en donde a cada punto se le llama pixel y tiene un valor discreto en la escala RGB que puede ir de 0 a 255. En la Figura 2.2 podemos observar cómo se forma una imagen digital.

2.4 Procesamiento digital de imágenes

El procesamiento digital de imágenes se refiere a los diversos algoritmos diseñados y desarrollados para procesar las imágenes obtenidas por un sistema de visión. Una imagen por sí sola es sólo la representación de lo que está viendo el sistema, mediante el procesamiento de esta, es posible que los sistemas sean capaces de tomar decisiones como moverse o no moverse, identificar alguna característica crítica en un producto, activar algún actuador, enviar alertas al usuario, etc.

2.5 Binarización de imágenes

La binarización de una imagen es la conversión de color o escala de grises de una imagen a una imagen de dos niveles o dos colores, blanco y negro. Por lo general una imagen consta de una parte que es el primer plano y otra que es el fondo. Este proceso conlleva a una pérdida de información inherente, sin embargo, la binarización es un paso crucial en la detección de características como texto o bordes en una imagen.[17]

Existen diversos algoritmos para lograr la binarización de una imagen. Uno de los más populares es el método de Otsu.

2.6 Segmentación de imágenes

El campo del procesamiento digital de imágenes es vasto y complejo, por lo que el enfoque serán las técnicas y los elementos utilizados para el desarrollo del proyecto.

Como se ha mencionado en secciones anteriores, el procesamiento digital de imágenes es el qué hacer con las imágenes una vez que han sido interpretadas y almacenadas por el sensor de la cámara utilizada en la aplicación. La segmentación de imágenes se refiere a encontrar un grupo de píxeles que van juntos [14]. Es un análisis estadístico de una imagen para el cual existen múltiples algoritmos que se han ido perfeccionando y mejorando conforme ha avanzado la tecnología. Algoritmos populares que se utilizan en la segmentación de imágenes son la detección de contornos, separar y unir, cortes normalizados, etc. En esta sección se tratará el tema de filtros, los cuales son útiles para la detección de características en imágenes digitales.

2.7 Filtros

En el procesamiento digital de señales o visión artificial, el uso de filtros es una técnica común y sin la cual muchas aplicaciones del campo no son posibles. Generalmente, uno de los primeros pasos en la manipulación de las imágenes es aplicar filtros, ya sea para reducir ruido, para separar colores o para obtener alguna característica deseable de la imagen obtenida. Los filtros digitales se pueden aplicar a una imagen en el dominio del espacio (operando directamente sobre los pixeles) o de la frecuencia, haciendo uso de técnicas de transformación de Fourier.

Matemáticamente, se puede decir que existen filtros lineales y filtros no lineales. Las operaciones que se realizan a las matrices correspondientes a los valores de los pixeles de las imágenes en los filtros en general son descritas como operaciones locales

o por çolonias"(*neighbourhoods* en inglés). Esto es, cada pixel en la imagen filtrada es el resultado de la suma ponderada de los pixeles de cada colonia contra una máscara definida. En el caso de los filtros no lineales, el tema de interés es el de operaciones morfológicas. Estas operaciones son utilizadas ampliamente en imágenes binarizadas, definiendo un valor de umbral sobre el cual se basa la decisión del valor de un pixel. Algunas operaciones morfológicas comunes son dilatación, erosión, apertura y cerramiento. [18]

2.7.1 Filtro Gaussiano

El filtro Gaussiano es uno de los filtros más comunes y utilizado en el procesamiento digital de imágenes para suavizar o eliminar ruido. Para suavizar, la técnica utilizada es que se reduce la variación en la intensidad de pixeles con respecto a sus vecinos. Para eliminar ruido, el algoritmo se encarga de cambiar el valor de intensidad de pixeles individuales que son muy diferentes a sus pixeles vecinos. En la Figura 2.3 se muestra la comparación de una imagen antes y después de aplicar un filtro Gaussiano.

Este filtro es tipo pasa-bajos, lo cual quiere decir que detalles de altas frecuencias y ruidos son filtrados efectivamente. Para un píxel en la imagen, cuyo valor es el promedio ponderado de los valores de los píxeles circundantes en la imagen. En la Ecuación 2.7.1 se encuentra la representación matemática de este tipo de filtro.

$$G(\boldsymbol{I}_p) = \frac{1}{W_g} \sum_{q \in \Omega} g_{\sigma_s}(p-q)\boldsymbol{I_q} \tag{2.1}$$

Donde $W_g = \sum_{q \in \Omega} g_{\sigma_s}(p-q)$ es el coeficiente normalizado, σ_s es la desviación estándar de la función Gaussiana correspondiente. [19]

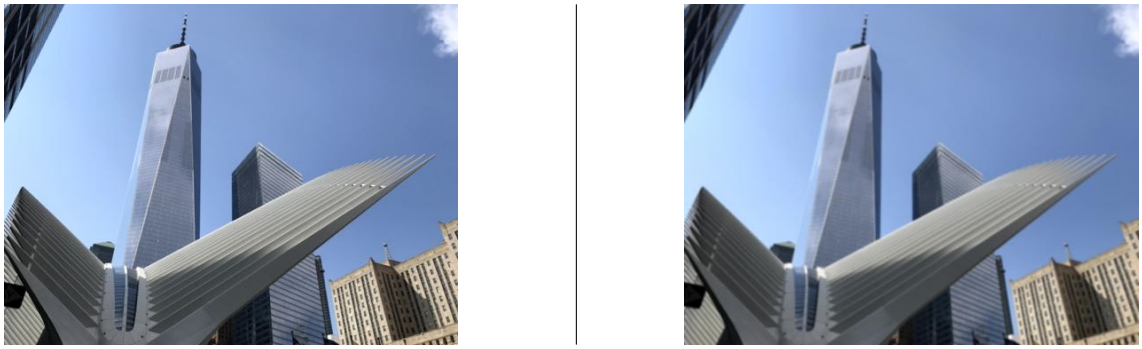


Figura 2.3: Comparación de una imagen original y la misma imagen después de aplicarle un filtro Gaussiano.

2.7.2 Filtro Sobel

Para la implementación del filtro Sobel se requiere el uso de dos matrices o *kernels*, por su nombre en inglés. Estos *kernels* se utilizan en una operación de convolución con cada pixel de la imagen. Un *kernel* tiene una respuesta máxima a los bordes verticales y otro Un *kernel* tiene la respuesta al borde horizontal. El valor mayor de las dos convoluciones son el nuevo valor del pixel bajo prueba. [20] En la Figura 2.4 es presenta una comparación de una imagen antes y después de aplicarle el operador Sobel.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

El operador Sobel se utiliza para la detección de bordes en imágenes. Durante el algoritmo se obtienen los valores ponderados de grises de los pixeles superiores, inferiores, derechos e izquierdos de cada uno de los pixeles, lo cual resulta en que también se reduzca el ruido.

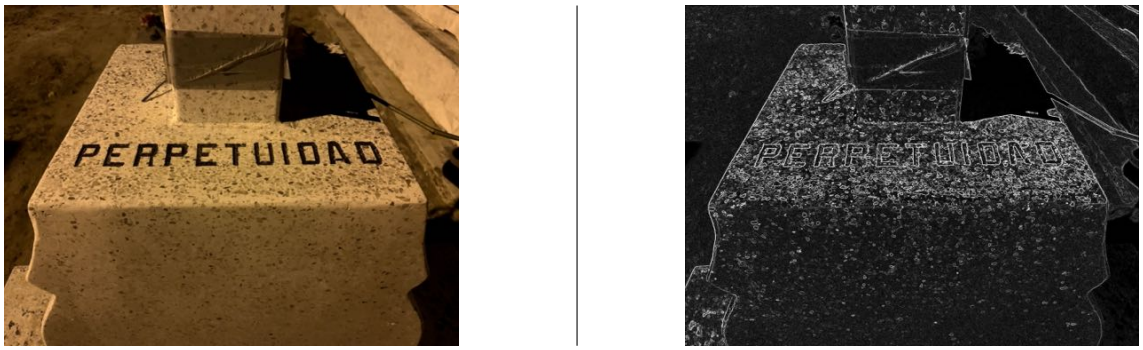


Figura 2.4: Comparación de una imagen original y la misma imagen después de procesarla con un operador Sobel para detección de bordes.

2.7.3 Filtro Canny

Otro operador que es utilizado en aplicaciones en las que se requiere la detección de bordes es el de Canny. Este algoritmo fue desarrollado por John F. Canny en 1986, el cual tiene como objetivo obtener la mejor aproximación de la detección de bordes en una imagen [21]. Este algoritmo es constantemente retomado para mejorar y optimizar. OpenCV incluso tiene una función predeterminada para la detección de bordes con el método de Canny [6].

Para implementar la detección de bordes con el método de Canny, el primer paso es la conversión de la imagen a escala de grises, para después aplicar un filtro Gaussiano para eliminar ruido y suavizar la imagen. Para esto se utiliza la ecuación:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

Donde x y y representan la ubicación del pixel actual dentro de la imagen y el valor de la desviación estándar σ se utiliza de manera arbitraria. Posteriormente se aplica un operador Sobel para obtener la dirección y la amplitud de los bordes detectados en la imagen, lo cual resulta en una imagen en la cual se detectan bordes, sin embargo, no completa ni claramente.

Durante el siguiente paso se obtienen los valores de la amplitud y ángulos del gradiente de la imagen. Para esto se utiliza un tabulador en el que dependiendo del valor del ángulo del borde, se clasifican en ángulos de 0, 45, 90 y 135 grados. Las ecuaciones 2.3 y 2.4 se utilizan para obtener las amplitudes G y los valores del gradiente θ.

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.3}$$

$$\theta = \arctan \frac{G_y}{G_x} \tag{2.4}$$

Donde G_x y G_y son los valores después del filtro Gaussiano en x y y de la imagen filtrada y θ es la dirección y orientación determinada de cada pixel.

Se aplica, a continuación, la supresión no-máxima, en la cual se realiza el algoritmo para recorrer las matrices de amplitud y gradiente. Se analizan los pixeles vecinos al pixel de la posición actual (se itera pixel por pixel por toda la imagen) en dirección del gradiente de dicho pixel. Esta parte del algoritmo se encarga de eliminar los pixeles que no sean detectados como parte de un borde.

Figura 2.5: Comparación de una imagen original y la misma imagen después de aplicarle un filtro Canny para detección de bordes.

Por último, se aplica un algoritmo de selección de umbral de histéresis. Durante este paso se recorre la matriz de la imagen y se evalúa pixel por pixel (por lo general evaluando contra sub-matrices de 3x3) para determinar si es un pixel fuerte o uno débil. Si un pixel es detectado como fuerte, se le asigna la intensidad de 1 (o blanco), si es débil, se le asigna la intensidad de 0 (o negro). Siendo así, al final se tiene una imagen binarizada (con sólo dos intensidades, blanco y negro), en las cuales todos los bordes se representan en blanco, y todo el fondo de la imagen se representa con negro, como se muestra en la Figura 2.5.

2.7.4 Reconocimiento Óptico de Caracteres (OCR)

El Reconocimiento Óptico de Caracteres es una tecnología que se ha implementado y utilizado desde la década de 1970, y su objetivo principal es la extracción y reconocimiento de caracteres tipográficos, o de texto, de imágenes como fotografías, señalamientos de tráfico, libros y prácticamente cualquier imagen que contenga texto. Desde la década del 2010, durante la cual el uso y acceso a los móviles inteligentes se movió de ser un nicho del mercado a ser de consumo general, la tecnología del OCR (por sus siglas en inglés) ha avanzado al punto de que se puede realizar en tiempo real.

Un sistema de OCR depende de un paquete de tipografías, ya sea con una sola tipografía o con múltiples. Conforme se vayan considerando más tipos de fuentes, el sistema se hace más complejo. En sistemas avanzados, incluso se puede definir

Figura 2.6: Reconocimiento de texto en una imagen.

un conjunto de caracteres especiales y tipografías personalizadas de la aplicación en la que se están empleando. [22] Los algoritmos para la detección de caracteres son variados, sin embargo, las técnicas utilizadas son generalmente las mismas (filtrado para detección de imágenes, análisis estadísticos para espaciamiento, etc.), sólo con modificaciones para optimizar los tiempos de procesamiento y los resultados obtenidos [23]. En la Figura 2.6 se observa un ejemplo en el que se extrae y se muestra el texto de una imagen.

2.7.5 Análisis de BLOB

El análisis de binario de objetos grandes (BLOB por sus siglas en inglés) es una técnica que se emplea en el procesamiento digital de imágenes para detectar objetos grandes en fotografías o video. El funcionamiento del algoritmo se basa en detectar la conectividad de pixeles contiguos en imágenes. [24] Esto se realiza para extraer características de un objeto en una imagen como su geometría, su ubicación, etc. Las aplicaciones de estos algoritmos varían desde la detección de cuerpos en movimiento, hasta el exceso de soldadura en un pin de un circuito integrado.

2.8 OpenCV

OpenCV (*Open Source Computer Vision Library*) es una librería de software de código abierto para visión artificial y aprendizaje automático. OpenCV fue construida para proveer una infraestructura común para aplicaciones de visión artificial y para acelerar el uso de percepción automática en productos comerciales. Al contar con una licencia de BSD (*Berkeley Software Distribution*), OpenCV facilita al usuario la utilización y modificación del código. La librería tiene más de 2,500 algoritmos optimizados, los cuales se pueden utilizar para reconocer caras, identificar y clasificar objetos, extraer modelos 3D de objetos, entre muchas más aplicaciones. OpenCV cuenta con interfaces para múltiples lenguajes de programación, entre los más usados: C++, Python y Java; además de estar disponible para diversos sistemas operativos como Windows, Linux, Android y macOS [25].

2.9 Template matching

El algoritmo de *template matching* se utiliza para buscar y encontrar incidencias de una imagen patrón o de entrenamiento dentro de otra imagen. En el caso de la implementación de *template matching* en OpenCV, se utiliza un algoritmo de convolución en 2D y compara la imagen de referencia proporcionada como argumento de entrada de la función contra la imagen en la que se desea buscar y encontrar la incidencia de la misma. La función `matchTemplate()` en OpenCV tiene implementados varios métodos de comparación de imágenes. La función de OpenCV para *template matching* tiene la siguiente definición:

```
cv.MatchTemplate(I, T, R, metodo);
```

donde `I` es la imagen en la que se buscará `T`, el patrón que se quiere encontrar. `R` es la imagen con los resultados de las coincidencias y `metodo` es el método que indica el algoritmo a utilizar dentro de la función para encontrar las incidencias del patrón en la imagen prueba [26].

El argumento de `metodo` en la función de OpenCV cuenta con seis opciones diferentes para la detección de coincidencias:

- `CV_TM_SQDIFF`: este método calcula la diferencia cuadrada entre la imagen patron y la imagen de prueba. La mejor coincidencia en este caso tiene un valor de 0. La Ecuación 2.5 describe este método.

$$R_{\text{sdiff}} = \sum_{x',y'} [T'(x',y') - I(x+x',y+y')]^2 \tag{2.5}$$

- `CV_TM_CCORR`: este método obtiene la correlación entre la imagen patrón y la imagen de prueba en cada posición. La mejor coincidencia en este caso tiene un valor grande (el cual no es necesariamente el máximo, ya que depende del nivel de ruido de la imagen). La Ecuación 2.6 describe este método:

$$R_{\text{ccorr}} = \sum_{x',y'} [T'(x',y') \cdot I(x+x',y+y')]^2 \tag{2.6}$$

- `CV_TM_CCOEFF`: este método calcula el coeficiente de correlación de coincidencia y describe la correlación entre la imagen patrón restada de su media y la imagen de prueba restada de su media en cada posición, considerando sólo el tamaño de la imagen patrón. Cabe notar que la correlación puede dar resultados pobres cuando la energía de la imagen, $\sum I^2(x,y)$, varía entre posiciones. Las Ecuaciones 2.7, 2.8 y 2.9 describen este método y una coincidencia también es un valor grande.

$$R_{\text{ccoeff}} = \sum_{x',y'} [T'(x',y') \cdot I(x+x',y+y')]^2 \tag{2.7}$$

donde

$$T'(x',y') = T(x',y') - \frac{\sum_{x'',y''} T(x'',y'')}{w-h} \tag{2.8}$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{\sum_{x'',y''} I(x'',y'')}{w-h} \tag{2.9}$$

y donde w y h son el ancho y alto de la imagen respectivamente.

Figura 2.7: Ejemplo de una imagen patrón encontrada en una imagen de prueba mediante el uso de *template matching*.

Los métodos adicionales implementados para la función `MatchTemplate` son versiones normalizadas de los algoritmos ya descritos. Estos métodos se llaman: `CV_TM_SQDIFF_NORMED`, `CV_TM_CCORR_NORMED` y `CV_TM_CCOEFF_NORMED`. La normalización es realizada dividiendo el método por el factor de nomalización descrito por la Ecuación 2.10 [27].

$$\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2} \tag{2.10}$$

En la Figura 2.7 se observa una imagen patrón `T` que fue encontrada en la imagen de prueba `I` utilizando el método `CV_TM_CCOEFF_NORMED`.

2.10 Automatic Optical Inspection

La Inspección Óptica Automática es una inspección visual que se utiliza en la fabricación de PCB para detectar defectos de pasta de soldadura o de soldadura en componentes de diversos tipos. Es un proceso que consiste en la captura de imágenes de la tarjeta electrónica, ya sea en su totalidad o en regiones de interés de acuerdo a lo que se requiera inspeccionar. El hardware y los algoritmos utilizados en el sistema de AOI también dependerán de las características a inspeccionar, pero en general consta de una unidad óptica (iluminación, cámaras), mecanismo de posicionamiento y un sistema de control (hardware, software) [13] .

La AOI como proceso es un control de calidad que se utiliza generalmente después de la fabricación de las tarjetas electrónicas para detectar defectos antes de que las tarjetas sean enviadas al cliente final. La necesidad de este proceso nace del incremento en la densidad de componentes de las PCB conforme los avances tecnológicos fueron requiriendo tarjetas y componentes cada vez más pequeños. El tiempo de inspección de un sistema de AOI es menor al que le tomaría a una persona por tarjeta, tomando en cuenta también que este tipo de sistemas se utilizan en ambientes de producción masiva.

Los sistemas de AOI utilizan algoritmos de visión artificial, como filtros, ecualización de histogramas, binarización, OCR, detección de bordes, análisis de BLOB y *template matching*. Estos algoritmos se utilizan dependiendo de la característica que se requiera inspeccionar. Por ejemplo, para inspeccionar que un circuito integrado sea el correcto basándose en su número de parte impreso sobre él, se entrenaría y se utilizaría la técnica de OCR; de igual manera, si se requiere determinar si existen puentes de soldadura, se podría binarizar la imagen y evaluar la existencia de material reflejante entre terminales para definir que hay soldadura presente.

Dentro de los defectos de interés para los sistemas de AOI se encuentran los defectos de componente, los defectos de terminales (en dispositivos SMD, *through-hole*, DIP, etc.) o defectos de soldadura. Estos se describen en las secciones 2.10.1, 2.10.2 y 2.10.3.

2.10.1 Defectos de componente

- **Ausente:** haciendo uso del algoritmo de medición de cuerpo en 3D, la máquina es capaz de calcular el volumen de la región de interés indicado sobre la tarjeta. Al tener valores fijos de dimensiones en x, y, z esperados, el sistema es capaz de medirlos en cada una de las tarjetas. Siendo así, una tarjeta que en el componente a inspeccionar no cumpla con el volumen esperado, se detecta como componente ausente.

- **Incorrecto:** para determinar que el componente es incorrecto, en este sentido, se utiliza un algoritmo de OCR para leer el número de parte. Esto es utilizado específicamente en circuitos integrados, los cuales cuentan con la leyenda del número de parte y fabricante en la parte superior.

- **Polaridad:** en componentes que presentan una muesca (circuitos integrados) o alguna otra identificación de la polaridad del mismo, se utilizan algoritmos de coincidencia de patrones, los cuales requieren una serie de imágenes de referencia o entrenamiento para aprender a diferenciar lo aceptable de lo no aceptable.

2.10.2 Defectos de terminales

- **Posición incorrecta:** al seleccionar el algoritmo de medición 3D de terminales, es posible, mediante la medición de las posiciones x, y, z en la zona de interés, determinar si una terminal individual o una serie de terminales del mismo componente se encuentran bien posicionadas de acuerdo a lo que se haya utilizado como referencia de entrenamiento. Esto es, una terminal que se encuentra doblada o ausente, es detectada gracias a este algoritmo.

- **Levantadas:** para la detección de esta clase de defectos se utiliza el mismo algoritmo que para detectar posición incorrecta, sólo que se enfoca principalmente en la altura de las terminales contra el plano de referencia (la PCB).

2.10.3 Defectos de soldadura

Todos estos defectos hacen uso del algoritmo disponible en la máquina llamado Perfil de Medición de Soldadura en 3D. Este algoritmo se concentra en medir volúmenes

de soldadura en la región de interés y determina, de acuerdo al tipo de componente y terminal, distintos parámetros requeridos para tomar decisiones, como alturas, ángulos y volúmenes de soldadura en porcentaje. Siendo así, este algoritmo sirve para detectar defectos de soldadura.

- **Abierta:** se refiere a la ausencia completa de soldadura en una terminal. Es decir, el volumen de soldadura es casi nulo de acuerdo a los parámetros establecidos.
- **Insuficiente:** es cuando las terminales cuentan con soldadura, sin embargo, no es suficiente para cubrir el volumen requerido de acuerdo a lo establecido como aceptable.
- **Esferas de soldadura:** es cuando se encuentra un exceso de soldadura en una terminal, mucho más del límite de volumen superior.
- **Corto:** al proporcionar la distancia entre terminales aceptable y característica de un componente, el sistema es capaz de analizar también el espacio entre dicha separación. Es así como se logra medir el número de terminales identificadas, así como la presencia de objetos o soldadura entre ellas.

2.11 Aplicaciones

La teoría revisada en este capítulo respalda las herramientas y funciones utilizadas para el desarrollo del presente trabajo y proyecto. Ambas aplicaciones, AOI y la detección de tornillos, utilizan estos fundamentos teóricos de la visión artificial para lograr los objetivos declarados. Es importante conocer los conceptos y las funciones que se describen en este capítulo para comprender de mejor manera las aplicaciones que se le dan en el desarrollo e implementación de este proyecto.

En el siguiente capítulo se describirán los materiales y métodos utilizados para el desarrollo de ambas aplicaciones, así como la manera en la que se validará el funcionamiento de las mismas.

Capítulo 3

Materiales y Métodos

3.1 Materiales

3.1.1 AOI

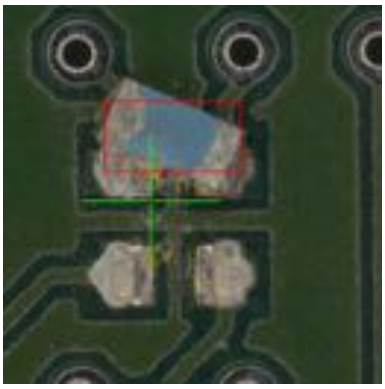
El sistema de visión que se utilizará es el FX-940 Ultra 3D AOI de la marca Nordson Yestech, el cual es capaz de realizar inspecciones en 2D y en 3D. El sistema cuenta con cinco cámaras: una orientada de arriba a abajo, y las otras cuatro cámaras en los lados para realizar las inspecciones y la generación de modelos en 3D. Las cámaras tienen la capacidad de inspeccionar componentes electrónicos con tamaño mínimo de 3x6 mm. Utiliza un sistema de iluminación desarrollado por la compañía, denominado *Fusion Lightning*, basado en LED multi-ángulo. El tamaño máximo de PCB que puede ser introducido a la máquina es de 500x525 mm, con altura de hasta 50 mm.

La computadora utilizada en el sistema de AOI cuenta con un procesador de cuatro núcleos Intel Xeon E5-1630 v3 a una velocidad de 3.7 GHz, 24 GB de RAM y corre el sitema operativo de Windows 10 a 64 bits. El GPU instalado es el GeForce GTX 1080 de NVIDIA.

Los algoritmos implementados en el software propietario de Nordson son: manipulaciones de color, Verificación Óptica de Caracteres (OCV), OCR, reconocimiento de códigos de barras, algoritmos basados en imagen o en reglas, así como análisis de componentes en 3D.

Los defectos que pueden ser detectados por la máquina haciendo uso de estos algoritmos se clasifican en:

- **Defectos de parte:** posición, ausente, incorrecto, polaridad, alineación, elevación.



(a) Posición y alineación incorrecta; componente ausente.

(b) Componente incorrecto.

Figura 3.1: Ejemplos de defectos de parte.

- **Defectos de terminal:** doblada, levantada, puentes.

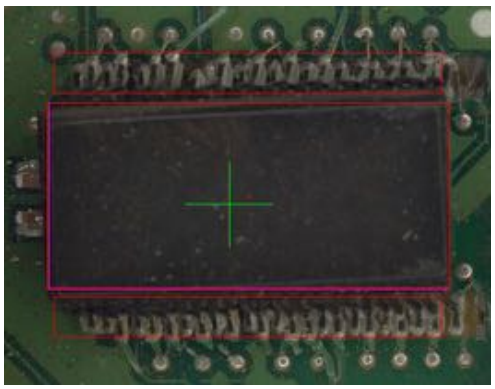
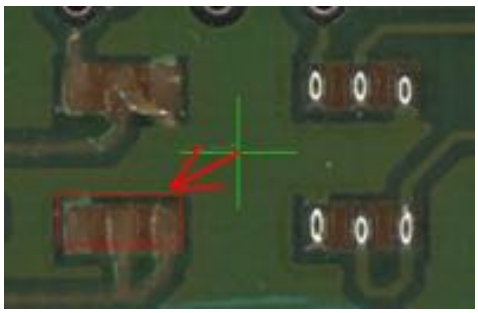


Figura 3.2: Ejemplo de componente con terminales dobladas y levantadas.

- **Defectos de soldadura:** abierta, insuficiente, corto, esferas de soldadura.

(a) Exceso de soldadura, puentes.



(b) Terminal abierta, sin soldadura.

3.1.2 Detección de tornillos ausentes

CPU y software

El sistema propuesto para la detección de tornillos está basado en una Raspberry Pi 3 Modelo B (Figura 3.4) con 1 GB de RAM y un procesador Quad Core 1.2GHz Broadcom BCM2837 64bit. El primer prototipo del software se desarrolló en MATLAB. La implementación del código en la Raspberry Pi se hará usando el lenguaje de programación Python 2.7 con la librería open-source OpenCV 3 que permitirá contar con un prototipo de bajo costo, respecto a las opciones disponibles en el mercado.

Figura 3.4: Raspberry Pi 3 Model B.

Captura de imágenes

La captura de las imágenes de las PCB se realizó con una Raspberry Pi Camera v2.1 (Figura 3.5), la cual tiene un sensor de 8 megapíxeles con resolución verdadera de 3280x2464 y distancia focal de 3.04 mm. La distancia entre la cámara y las PCB es de 35.5 cm.

Figura 3.5: Módulo Raspberry Pi Camera v2.1.

3.2 Métodos

3.2.1 AOI

Actualmente, la inspección de las tarjetas electrónicas actualizadas o reparadas se lleva a cabo con un microscopio digital y una pantalla. El microscopio es manipulado

manualmente - tanto la posición como el enfoque de la cámara - y se inspeccionan únicamente los componentes señalados con marcador como actualizados o reparados. Se realiza una inspección general de la tarjeta, sin embargo, el nivel de inspección varía de operador a operador.

Los componentes que serán inspeccionados con el sistema AOI son los que se clasifican como actualizaciones mandatorias. Una actualización mandatoria es un cambio que se hace en la tarjeta por definición del área de Ingeniería de Producto para que el producto siga funcionando en un segundo ciclo de vida útil. Estas actualizaciones pueden ser desde el reflujo de soldadura en las terminales de un componente, hasta el reemplazo o conversión de un componente. Estas listas de actualizaciones mandatorias se tienen publicadas en el sistema de calidad de la empresa y son seguidas por los operadores técnicos en cada unidad. Además, se tiene una base de datos interna en la que se almacenan los defectos encontrados en la operación de inspección de las tarjetas. Se tomaron las áreas de las tarjetas donde se han detectado más defectos y se consideraron los componentes en ellas para el desarrollo de las recetas o programas del sistema AOI.

Para la validación de la hipótesis, se realizará un estudio de R&R (Repetitividad y Reproducibilidad) de atributos para determinar el porcentaje confiabilidad del sistema de AOI para la detección de defectos en las PCB. Cuando hablamos de repetitividad, hablamos de la variación en el proceso a causa del sistema de medición (en este caso, la máquina AOI); y cuando hablamos de reproducibilidad, nos referimos a la variación en el proceso a causa de la persona realizando la operación (en este caso, los operadores que utilizarán la máquina).

Para realizar este estudio se obtuvieron 30 PCB del proceso de remanufactura. Se generaron defectos conocidos en 15 de las tarjetas y se aseguró que el resto de las unidades eran aceptables. Después de generar las recetas o programas para la inspección de los componentes de las tarjetas, se realizará un experimento en el que 3 operadores distintos inspeccionarán las 30 tarjetas en el sistema AOI y se registrará el resultado final de la máquina para posteriormente llevar a cabo el análisis estadístico de estos datos y determinar el porcentaje de repetitividad y reproducibilidad del proceso.

Los defectos generados en las unidades fueron definidos de manera que se cubriera la totalidad de las fallas observadas en la línea de producción y en los componentes que se han detectado en reclamos de garantías. En el Cuadro 3.1 se muestran los componentes y los defectos generados. Para las unidades consideradas como aceptables, se aseguró que no se encontraran defectos en todos los componentes inspeccionados por el programa de la máquina. Los parámetros para la definición de los valores aceptables se obtuvieron del estándar IPC-A-610F para ensambles de tarjetas electrónicas.

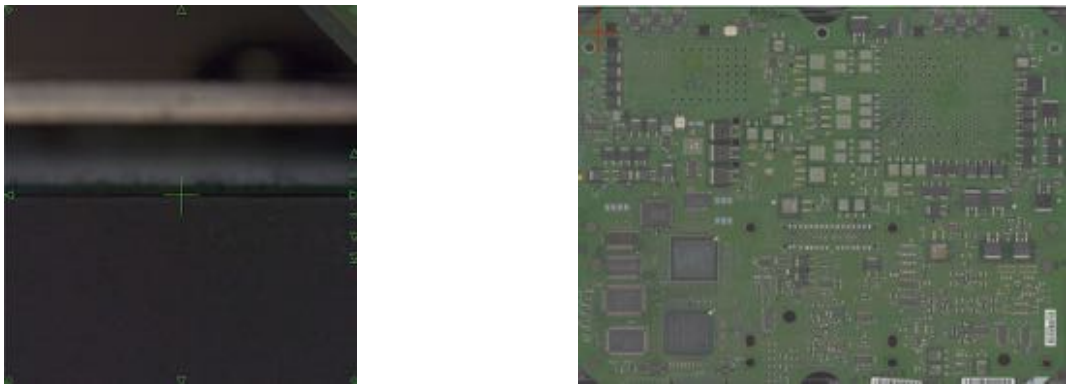


Figura 3.6: Esquina seleccionada para alineación y tarjeta escaneada.

Referencia de Componente	Defecto	Algoritmo Utilizado
C160, C354, C213, C38, C250, C55	Componente ausente	*3D Body Measurement*
U57	Componente incorrecto	OCR
U3	Polaridad	*Template Matching*
U18	Posición incorrecta de terminal	*Lead Solder*
Q91	Terminal levantada	*Lead Positioning*
U59	Poca soldadura o abierta	*Lead Solder*
U26	Puente o exceso de soldadura	*Lead Solder*

Cuadro 3.1: Defectos conocidos generados en tarjetas para experimento.

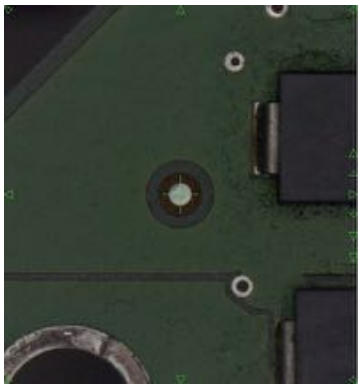
En esta sección se describirá la programación de cada uno de los algoritmos utilizados en la máquina para la detección de estos defectos, iniciando por la alineación y primer escaneo de la tarjeta para la generación de los planos de referencia que se utilizan para los *renders* en 3D de componentes.

Generación de receta y alineación de tarjeta

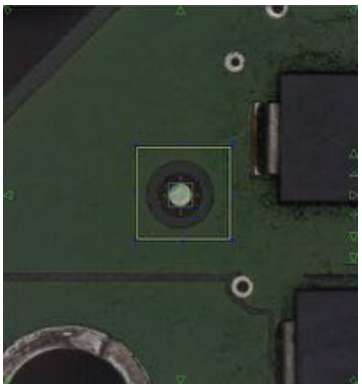
Al generar una receta nueva en el programa NYTVision, se requiere ajustar el ancho de la banda transportadora de las tarjetas. Las dimensiones de las tarjetas del producto a inspeccionar son 33.27 cm de largo por 23.36 cm de ancho, por lo que se ajusta la banda a 235 mm y se almacena el valor para que el programa ajuste la banda a ese ancho siempre que se cargue la receta. Cabe mencionar que la máquina cuenta con la capacidad de cargar archivos CAD para la generación de recetas, sin embargo, no se cuenta con estos archivos de diseño de la tarjeta, por lo que la totalidad de la programación se realizará de manera manual.

Se requiere una tarjeta de referencia para realizar la alineación de la tarjeta en la máquina. El algoritmo de alineación requiere que se seleccionen dos puntos del perímetro de la tarjeta: la esquina inferior izquierda y la esquina superior derecha. Las tarjetas a inspeccionar no son de forma rectangular, por lo que se seleccionan puntos simulando una tarjeta de esta forma. Una vez que se tienen los puntos seleccionados, automáticamente se realiza un escaneo de toda el área seleccionada. Este escaneo consiste en la captura de múltiples fotografías de la tarjeta para poder generar una imagen de la totalidad de la tarjeta. En la Figura 3.6 se observa la tarjeta escaneada y alineada en el rectángulo seleccionado, así como un ejemplo del punto de la parte superior derecha.

También se requiere la selección de tres marcadores de referencia o fiduciarios,



(a) Un fiduciario seleccionado

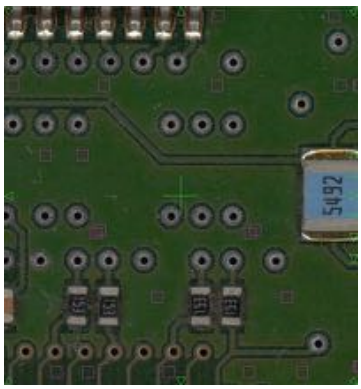


(b) Entrenamiento de fiduciario

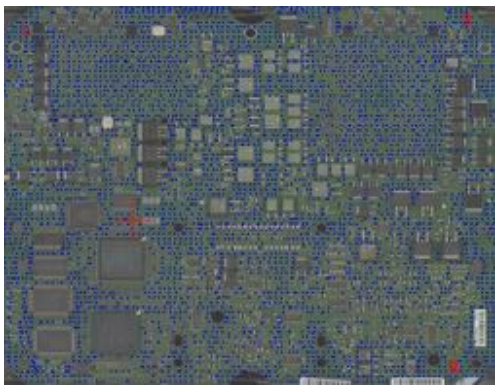
Figura 3.7: Ejemplo de entrenamiento de fiduciario.

los cuales se utilizarán en cada ejecución de la receta para alinear las cámaras con la tarjeta y eliminar las variaciones mecánicas que se puedan tener en el movimiento de la banda transportadora de las tarjetas. Los dibujos de ingeniería del diseño de la tarjeta presenta puntos que son utilizados como fiduciarios en la fabricación de producto nuevo, por lo que se seleccionan tres puntos cercanos al borde de la tarjeta. Estos puntos se seleccionan debido a que son distantes entre sí y entre ellos se encuentra la mayoría del área de trabajo que tendrá la máquina. En la Figura 3.7a se presenta el ejemplo de un punto fiduciario. Estos puntos se deben entrenar proporcionando un área en la que se debe encontrar el fiduciario, así como una imagen muestra de cómo es el punto. En la Figura 3.7b se observa la manera en la que se especifican estas dos características al programa de inspección.

La generación de imágenes en 3D de la tarjeta y sus componentes requiere un plano de referencia x, por lo que se proporciona una muestra de la superficie de dicho plano; en este caso, la superficie de la tarjeta donde no se encuentra ningún componente ni pista. En la Figura 3.8a se observa el punto seleccionado como referencia del plano y el resultado al terminar el algoritmo de generación de puntos de referencia. Todos los puntos azules sobre la tarjeta (Figura 3.8b) representan los puntos de referencia del plano SF_REF.



(a) Zoom de puntos de plano SF_REF



(b) Plano SF_REF en la tarjeta

Figura 3.8: Representación de los puntos del plano SF_REF en la tarjeta.

Hasta este momento del proceso de programación, se tienen todas las referencias

de alineación para proceder incluir los componentes que se inspeccionarán con cada uno de los algoritmos mencionados en el Cuadro 3.1.

El sistema de AOI cuenta con 22 algoritmos diferentes para la detección de características en las tarjetas electrónicas:

1. *Pattern Matching*
2. *Barcode Reading*
3. *Color Inspection*
4. *Resistor Color Band*
5. OCR
6. *Edge Locator*
7. *Conformal Coat Inspection*
8. *Auto Focus*
9. *3D Coplanarity*
10. *Laser Height Gauge*
11. *Wire Bond Measurement*
12. *3D Height Measurement*
13. *Film Thickness Measure*
14. *3D Body Measurement*
15. *3D Polarity Measurement*
16. *Solder Blob Analysis*
17. *Lead Bank Blob Analysis*
18. *Histogram Analysis*
19. *Image Subtraction*
20. *3D Lead Measurement*
21. *3D Solder Profile Measurement*
22. *Geometric Pattern Matching*

Para el caso de los defectos que se inspeccionará en esta aplicación los algoritmos de *3D Body Measurement*, *Template Matching* y *Lead Solder* cubren el 100 % de las características que se evaluarán. Se presentará la programación de estos algoritmos utilizando el componente U3. Se seleccionó este componente debido a que en él se utilizan todos los algoritmos mencionados.

Posicionando las cámaras en la parte de la tarjeta donde se encuentra el componente U3, se utiliza la opción de Agregar Nuevo Componente y se introduce la

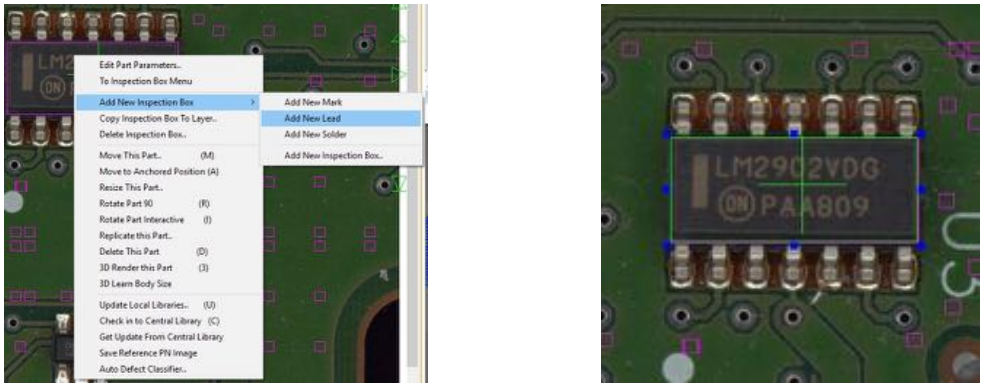
referencia del componente, en este caso, U3. Se especifica, además, la orientación del componente. En este caso es 0° y, debido a que el componente no se encuentra agregado aún a la base de datos del programa, no se especifica número de parte ni paquete. En la pantalla aparece un rectángulo de color fucsia de tamaño ajustable, el cual se ha de colocar en el contorno del componente, excluyendo las terminales, debido a que con esto sólo se está especificando la ubicación y el tamaño del componente.

Una de las grandes ventajas de la programación de las recetas en la máquina AOI es que cada inspección y cada algoritmo se programan por separado. El sistema proporciona 8 capas, las cuales se pueden designar para un tipo de inspecciones o de características de los componentes. Para esta receta, se define de manera arbitraria que la capa 0 será para todas las posiciones y tamaños de los componentes, mientras que la capa 1 será utilizada para todas las inspecciones de terminales o soldadura.

Programación de algoritmo: *3D Body Measurement*

El algoritmo de *3D Body Measurement* se utiliza para la medición y detección del volumen de un componente cualquiera en la tarjeta. El valor de volumen esperado de un componente en una posición determinada se utiliza para determinar si un componente se encuentra presente o no. Para ello es necesario proporcionar a la aplicación las medidas del componente, cargarlas de la librería de la receta o utilizar la funcionalidad de auto-enseñar de la máquina para determinarlas.

El sistema permite la adición de cada inspección mediante cajas de inspección. Estas pueden ser de tres tipos: marcas (características dimensionales de componente), terminal o soldadura. Para utilizar el algoritmo de *3D Body Measurement*, es necesario especificar el tipo de caja de inspección como Marca.

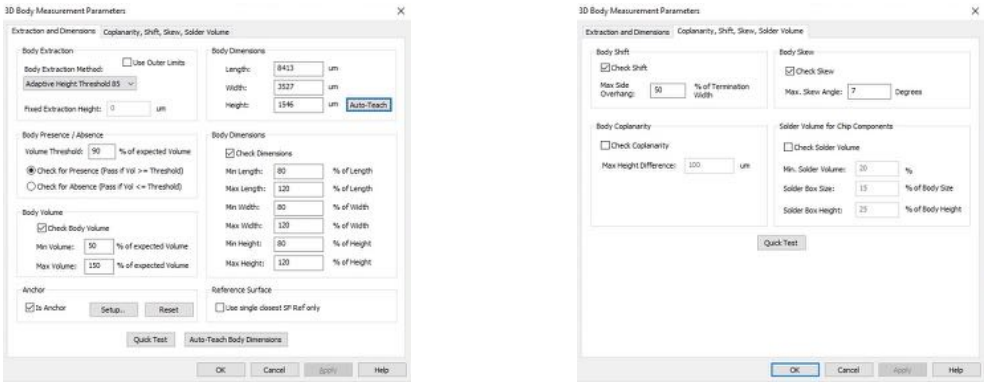


(a) Los tres tipos de cajas de inspección. (b) U3 con caja de inspección tipo Marca.

Figura 3.9: Colocación de caja de inspección para *3D Body Measurement*.

Al seleccionar esta opción, el programa coloca un rectángulo de color verde sobre el área de trabajo actual, el cual ha de ser ajustado al tamaño requerido para la inspección. En este caso, lo que se requiere es determinar el volumen del cuerpo del componente, por lo que se coloca en la misma posición y del mismo tamaño que el recuadro fucsia, que indica el cuerpo del componente. Esto se ilustra en la Figura 3.9).

En los parámetros de la Caja de Inspección, se selecciona el algoritmo de *3D Body Measurement* para indicar el tipo de decisión que se tomará en esta marca. Posteriormente, es necesario indicar los parámetros para la inspección del algoritmo.



(a) Extracción y Dimensiones. (b) Posición, Sesgo y Volumen de Soldadura.

Figura 3.10: Parámetros para algoritmo de *3D Body Measurement*.

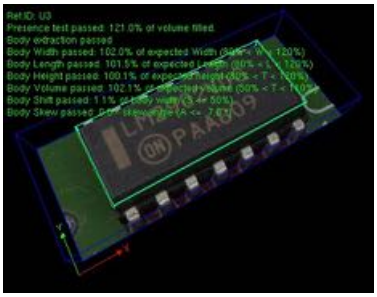


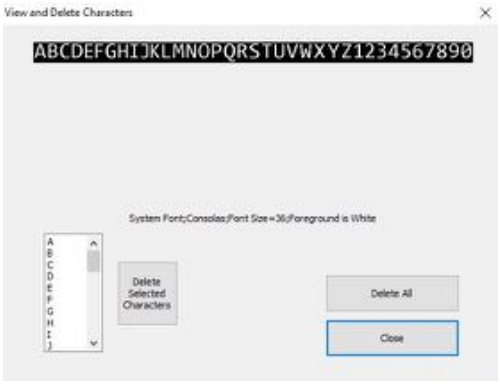
Figura 3.11: Representación del componente inspeccionado en 3D.

En la Figura 3.10 se presentan las pantallas en las que se configura el algoritmo. Los valores de anchura y longitud se determinan automáticamente a partir del recuadro verde colocado sobre el componente, sin embargo, también hay que especificar la altura. Este valor puede ser introducido manualmente o utilizar la opción de Auto-Enseñar. Esta opción hace uso de las cámaras de visión para determinar la altura correspondiente. Otro de los parámetros que hay que especificar es el tipo de detección que se desea, ya sea de componente presente o de componente ausente, y el volumen umbral (en porcentaje) para la toma de decisión de pasa o falla. En este caso, el valor utilizado es 90 % debido a que buscamos que el componente se encuentre presente y siempre se utiliza el mismo tipo de componente.

Otros parámetros que requieren configuración son los cambios en la orientación del componente y el sesgo (*body shift* y *skew*), los cuales se programan de acuerdo al estándar de IPC. En el caso del U3, la opción de revisar soldadura para componentes de chip no se utiliza, debido al tipo de componente que es. Sin embargo, esta opción sí se utiliza para los componentes como el C160, y esto permite que el programa automáticamente incluya la inspección de la soldadura presente en ellos.

Realizando todas estas configuraciones, el programa es capaz de inspeccionar la presencia o ausencia de un componente, así como su posición, tamaño, volumen, sesgo y cambios en la orientación del componente. En la Figura 3.11 se muestra el *render* en 3D generado cuando se capturan las fotografías con las cámaras 3D del sistema, así como un resumen de los resultados de las mediciones realizadas del componente.

(a) Parámetros para detección de texto en imagen.



(b) Caracteres entrenados.

Figura 3.12: Configuración del algoritmo OCR para la detección de cadenas de texto en imagen de prueba.

Programación de algoritmo: OCR

Para la detección del número de parte o valor de un componente, se programó el algoritmo de OCR en los componentes requeridos. La programación de este algoritmo se enfocó en el número de parte de circuitos integrados.

En el caso del U3, el número de parte es LM2902VDG. Se realizó la programación del algoritmo de OCR para asegurar que todos los U3 en todas las tarjetas utilizan este componente. Para ello, se agrega otra Caja de Inspección tipo Marca y se indica que se utilizará OCR como método de decisión.

Este algoritmo requiere una cadena de texto específica para buscar en el área de inspección seleccionada, así como la longitud de caracteres de la misma. Hace falta, además, incluir la tipografía que se utilizará para encontrar las coincidencias de los caracteres. Se especificó la fuente Consolas, ya que es mono-espaciada y es similar a la fuente utilizada por el fabricante del componente. También es necesario indicarle al algoritmo qué caracteres buscará en la imagen. Para esta inspección, se agregó el alfabeto en letras mayúsculas y los números del 0 al 9. La configuración del algoritmo se presenta en la Figura 3.12.

La decisión de algoritmo de OCR se toma en base a un umbral seleccionado que puede ir desde 0 a 100. En este caso, de manera experimental se utilizó el valor de 60 para evitar el rechazo de unidades buenas. Una vez que todos estos parámetros son configurados, la máquina será capaz de buscar y evaluar que el texto impreso en el componente es el que se espera leer, por lo tanto, definir si es el componente correcto o no.

En la Figura 3.13 se muestra el resultado de una unidad probada después de configurar el algoritmo, mostrando una coincidencia con una calificación de 88.51. Se observa en la imagen que el caracter 0 lo interpreta como una O, lo cual introduce el error.

(a) Caja de inspección para OCR. (b) Resultado de inspección por OCR.

Figura 3.13: Caja de inspección del componente U3 y resultado de una inspección.

Programación de algoritmo: *Template Matching*

La máquina cuenta con su propio algoritmo para *Template Matching* el cual se puede utilizar para diversas necesidades, como inspeccionar el fabricante de un componente, caminos en la tarjeta, fiduciarios, etc. En este caso, el algoritmo se programó para determinar la polaridad correcta de un circuito integrado.

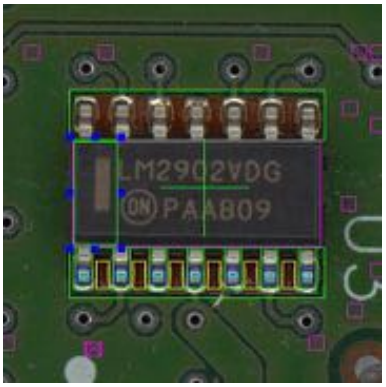
En la máquina, el nombre especificado para este algoritmo es *Pattern Matching*. Funcionalmente, el método estadístico para determinar la similitud entre la imagen patrón y la imagen de prueba es el de correlación normalizada. La decisión es tomada a partir de la calificación que el algoritmo asigna a la imagen de prueba. Esta calificación es un valor de 0 a 100 y también requiere que se especifique un valor de umbral para ello.

Al seleccionar el tipo de decisión de *Pattern Matching*, el sistema espera una imagen de entrenamiento para saber qué es lo que se desea buscar en cada imagen de prueba. Para esto basta con seleccionar con la caja de inspección de la marca y seleccionar la opción *Train Template*. Con esto, la imagen seleccionada se almacena como patrón y todas las tarjetas electrónicas serán inspeccionadas comparando contra él.

En la Figura 3.14a se observa el patrón seleccionado para identificar la orientación del componente U3. Esta línea siempre se debe encontrar en esta posición. Si la línea no se encuentra, esto indicará que el componente se encuentra instalado incorrectamente o que podría tener otro componente instalado. En la Figura 3.14c se observa un resultado de una prueba después de haber entrenado el algoritmo.

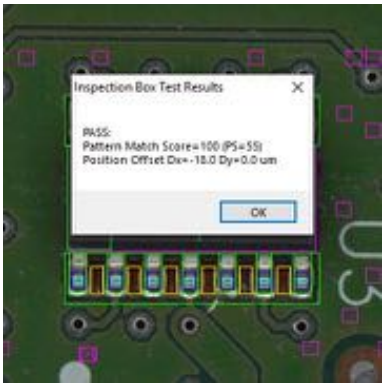
Programación de algoritmo: *Lead Solder*

Con el algoritmo *Lead Solder*, es posible inspeccionar todo lo referente a la soladura de las terminales de circuitos integrados. Dentro del mismo algoritmo se especifican diversos valores como el volumen de soldadura aceptable, el espacio entre terminales (en el caso de que sea un banco de terminales), posición de las terminales, etc. Todos estos parámetros nos ayudan a detectar puentes de soldadura, soldadura insuficiente, bolas de soldadura e instalación incorrecta de un circuito integrado.



(a) Caja de inspección para *Template Matching*.

(b) Opción para entrenar algoritmo.



(c) Resultado de inspección por *Template Matching*.

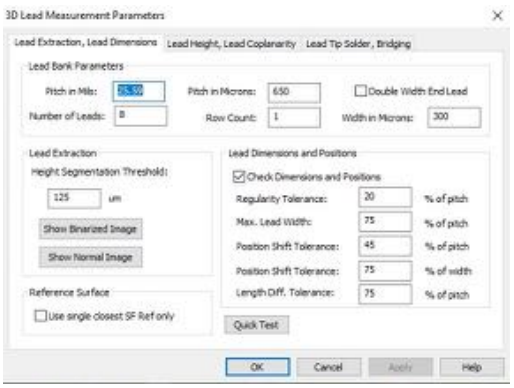
Figura 3.14: Configuración y resultado de programación de algoritmo *Template Matching*.

(a) Caja de inspección para *Lead Solder*. (b) Resultado de inspección.

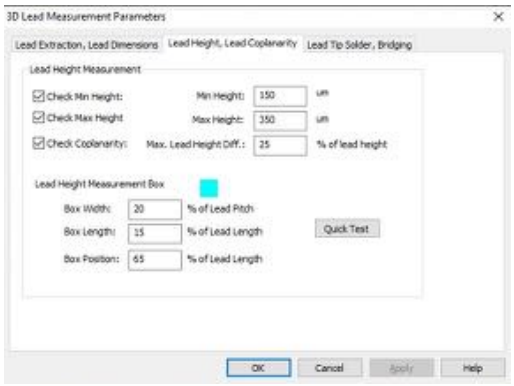
Figura 3.15: Ejemplo de inspección de banco de terminales para U3. En (b) se observan los colores de cada uno de los parámetros del algoritmo.

Este tipo de caja de inspección se debe seleccionar como *Lead* y se utiliza, donde aplique, como un banco de terminales (Figura 3.15a); es decir, no es necesario agregar una caja de inspección por cada terminal, ya que el sistema puede reconocer el número de terminales encontradas en la imagen utilizada para el entrenamiento del algoritmo. De igual manera, al sistema también se le puede especificar el número de terminales de manera manual.

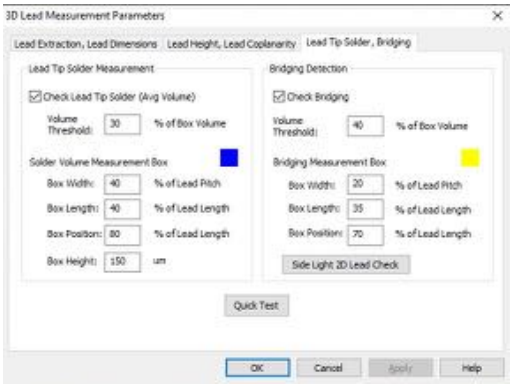
La programación de este algoritmo se hace en tres partes: extracción y dimensiones, altura y coplanaridad, y soldadura o puentes. En la Figura 3.16 se presentan las tres pantallas de configuración de este algoritmo. En la primera parte, se especifican los parámetros del banco de terminales, como la separación entre las terminales, el número de terminales, las tolerancias en las dimensiones de las terminales, etc. El sistema es capaz, con la función de *auto-teach* de detectar los tamaños de las terminales, sin embargo, también es posible introducirlos de manera manual. En la segunda parte, se especifican los parámetros de altura de las terminales y de coplanaridad de las mismas y las tolerancias aceptables. En la tercera sección se introducen todos los valores referentes a la ubicación y volumen aceptable de la soldadura, así como la activación de la opción para detectar puentes entre terminales. Todos los valores de los parámetros utilizados en la programación de este algoritmo se definieron de acuerdo al estándar IPC mencionado anteriormente. Sin embargo, algunos valores fueron definidos mediante experimentación de acuerdo a los resultados obtenidos con las funciones de *auto-teach*.



(a) Banco de terminales.



(b) Altura y Coplanaridad.



(c) Soldadura de terminales y puentes.

Figura 3.16: Configuración del algoritmo *3D Lead Solder*.

En cada inspección que utilice este algoritmo, los diferentes parámetros que se evalúan se identifican de un color diferente para reconocerlos fácilmente. Los parámetros de altura y coplanaridad se pintan de color cian, mientras que las características de soldadura en la punta de las terminales se muestra en azul, y los puentes entre terminales de color amarillo (Figura 3.15b). Cuando un banco de terminales queda programado correctamente, se utiliza una opción llamada Alinear Opuesto, lo cual genera una copia de esta caja de inspección del lado opuesto a la distancia especificada en las dimensiones del componente. En caso de que no quede exactamente donde se encuentra el otro banco de terminales, cabe solo con moverla al lugar deseado. Esto reduce el tiempo de programación de cada componente.

Población de receta de inspección

Utilizando todos estos algoritmos se procede a la programación del resto de los componentes. En total, se programaron 819 componentes en el modelo completo de la tarjeta. Estos componentes se seleccionaron utilizando la base de datos interna de la empresa. Se identificaron las áreas de la tarjeta en las que se han registrado la mayor cantidad de defectos. En la figura se observa la densidad de los componentes agregados a la receta para inspección. Todos los componentes que se inspeccionan se encuentran rodeados de azul. A modo de comparación, se proporciona también la imagen de la tarjeta cuando aún no se ha programado ningún componente.

Implementación de sistema AOI

La implementación del sistema AOI en las instalaciones de la empresa concluyó en Diciembre de 2019. Se siguió la metodología 6 Sigma para determinar la causa raíz del problema planteado y para la selección de la mejor opción en cuanto a equipo para resolverlo.

Se siguió la metodología en las fases DMAIC: definir, medir, analizar, mejorar y controlar. En cada una de las fases se desarrollan diferentes actividades claves:

- **Definir:** planteamiento de problema, alcance del proyecto, oportunidad de negocio, identificación de clientes, etc.
- **Medir:** identificar las variables, cuantificar los posibles resultados, recopilación de datos, identificar sistemas de medición.
- **Analizar:** convertir los datos obtenidos en la etapa de medir para entender mejor el problema y poder proponer soluciones.
- **Mejorar:** desarrollar una solución o soluciones para el problema, comparación de resultados obtenidos contra el desempeño base.
- **Controlar:** monitorear el comportamiento de la solución para asegurar que la mejora sea sostenible y reajustar variables si es necesario.

Uno de los grandes retos de la implementación del sistema fue la adecuación de los parámetros de aceptación de los algoritmos, ya que, por diseño, se encuentran ajustados para el ensamble de PCB por primera vez. Las tarjetas del proceso son remanufacturadas, es decir, han pasado por procesos de manufactura anteriormente, lo que agrega variables adicionales que no se tienen en el proceso por primera vez, por ejemplo, aplicación de *conformal coating*, de adhesivos disipadores o de sellos epóxicos. Es por eso que los parámetros de aceptación se desarrollaron usando el estándar IPC como base y en algunas ocasiones, se determinaron de manera experimental.

El sistema actualmente no se encuentra conectado a la red interna de la empresa, lo cual genera dificultades para implementar un sistema de rastreo que se integre con el resto del proceso. La limitación para poder conectarlo a la red es la marca de la computadora utilizada en el sistema. Los lineamientos de la empresa no permiten este tipo de computadoras conectadas a la red por cuestiones de seguridad y privacidad.

3.2.2 Detección de tornillos ausentes

Para comprobar el porcentaje de reconocimiento de la aplicación se utilizará la técnica de validación cruzada conocida como *leave one out*. Las pruebas se realizaron con un conjunto de 10 imágenes de prueba de tarjetas electrónicas (Unidades 1 a 10). En este tipo de validación, se elimina un dato de entrenamiento por iteración (10 iteraciones por prueba). En la Figura 3 se muestra un diagrama de flujo del algoritmo. Esto se realizó con dos subconjuntos de entrenamiento: uno de 10 tornillos y uno de 20 tornillos. El propósito de estas pruebas fue identificar si existe una diferencia significativa en el porcentaje de reconocimiento del sistema cuando se incrementa la cantidad de imágenes de entrenamiento. También se midieron los tiempos de ejecución, ya que esta técnica requiere un elevado número de iteraciones del código.

El esta sección se presentará el desarrollo de los prototipos en MATLAB y en Python con OpenCV. Los resultados experimentales se presentan en la Sección 4.2 del Capítulo 4.

Primer prototipo en MATLAB

Inicialmente se trabajó en el desarrollo de un prototipo en MATLAB usando filtros, detección de bordes y operaciones morfológicas para la detección de tornillos en la imagen. En este prototipo se utilizó un MacBook 2017 de 12 pulgadas, con procesador Intel Core m3 de 1.2 GHz, memoria RAM de 8 GB 1867 MHz LPDDR3 y una tarjeta de gráficos Intel HD Graphics 615 1536 MB. La versión de MATLAB utilizada fue la R2018a, corriendo en el sistema operativo macOS Mojave (10.14.5). Se obtuvieron fotografías con la cámara Logitech C270, la cual tiene un sensor de 1.2 MP y mediante software puede generar fotografías con resolución de hasta 3 MP; la resolución verdadera de las imágenes es de 1280x720 pixeles. En la Figura 3.17 se observa una imagen de prueba obtenida con esta cámara.

Se capturaron 10 imágenes de tarjetas ensambladas en la línea de producción para utilizar como imágenes de prueba para la validación del funcionamiento del algoritmo generado. Dentro de estas tarjetas se encontraban unidades aceptables y no aceptables.

Figura 3.17: Imagen de prueba capturada con la cámara Logitech C270.

En el Algoritmo 1 se presenta la lógica seguida para la codificación del mismo en el lenguaje de programación M. El programa inicia importando la imagen a un tipo de dato de MATLAB, en este caso, una matriz de dos dimensiones (x, y) para poder ser utilizada y manipulada dentro del entorno de MATLAB. Posteriormente, la imagen original es convertida a escala de grises, y se le aplica un filtro Gaussiano tomando una desviación estándar de 1.0. La desviación estándar se definió por experimentación con diversos valores para seleccionar el que diera mejores resultados. Este filtro se aplica para reducir ruido y difuminar la imagen. En la Figura 3.18 se muestra el resultado de la aplicación de este filtro a nuestra imagen.

Se definen, además, los umbrales aceptables para los dos tipos de tornillos identificados. Físicamente los once tornillos que se instalan en la tarjeta son iguales, sin embargo, el acomodo de ellos en la tarjeta es diferente, por lo que existen elementos

Algoritmo 1: Algoritmo de primer prototipo desarrollado en MATLAB

entrada : I: imagen de prueba
salida : *screws*: número de tornillos encontrados, t: tiempo de ejecución

obtener los canales R, G y B de I;
convertir I a escala de grises con $I_g = 0{,}2989R + 0{,}5870G + 0{,}1140B$;
aplicar filtro Gaussiano a I_g, de acuerdo a la ecuación 2.7.1;
almacenar imagen filtrada I_g en G, $G \leftarrow I_g$; aplicar filtro Canny a G;
definir elemento estructural *se* para dilatación de 4x4;
definir distancias horizontal y vertical entre tornillos $hDistance$ y $vDistance$;
definir tamaño de tornillo $screwSize$, 32x32;
definir punto inicial de la imagen de prueba $initialRef$;
definir umbrales $threshold1$ y $threshold2$;
inicializar contador de tornillos $counter \leftarrow 0$;
for $i = initialRef(ancho)$ **to** $hDistance$ **do**
    for $i = initialRef(alto)$ **to** $vDistance$ **do**
        obtener sub-imagen $screwSize$;
        dilatar sub-imagen con `imdilate(screwSize,se)`;
        obtener suma de intensidades de pixeles en sub-imagen, val;
        if $val > threshold1$ **then** $counter \leftarrow counter + 1$;
        else $counter \leftarrow counter$;
$tornillo = 5$;
repeat
    definir locaciones de tornillos con distancias diferentes;
    obtener sub-imagen $screwSize$;
    dilatar sub-imagen con `imdilate(screwSize,se)`;
    obtener suma de intensidades de pixeles en sub-imagen, val;
    if $val > threshold2$ **then** $counter \leftarrow counter + 1$;
    else $counter$;
    $tornillo = tornillo + 1$;
until $tornillo = 11$;
$screws \leftarrow counter$;
$t \leftarrow execTime$;

como planos de tierra y arandelas de diversos tamaños, lo cual hace que, a la hora de binarizar la imagen, muestren superficies diferentes. Por esto, para los tornillos que se encuentran en la parte inferior de la tarjeta, se define un umbral aceptable de 50 %, mientras que para el resto será 25 %. Estos umbrales también fueron definidos mediante experimentación con diferentes valores hasta encontrar los que funcionaban mejor para esta aplicación.Esto quiere decir que, al obtener las imágenes de 32x32 pixeles por tornillo, si menos del 50 o 25 % de los pixeles son 0, esto se toma como una ausencia de tornillo y, por tanto, el contador de tornillos no incrementará. En la Figura 3.19 se muestran los tornillos con sus umbrales definidos.

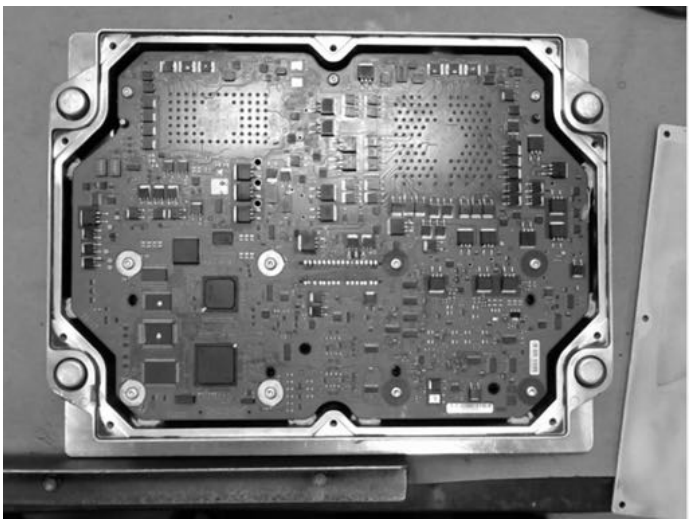


Figura 3.18: Imagen de prueba después de conversión a escala de grises y de la aplicación de filtro Gaussiano.

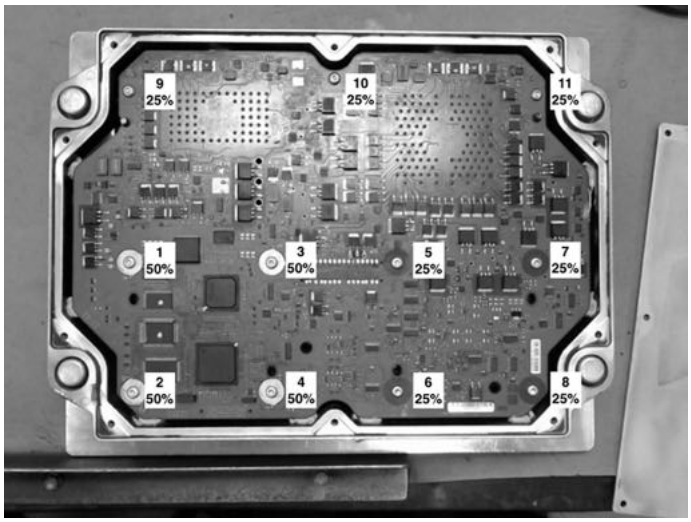


Figura 3.19: Ubicación de tornillos con su umbral correspondiente.

Se inicializa un contador de tornillos en 0, y se procede a aplicar un filtro de detección de bordes Canny a la imagen. Se realizan pruebas con distintos umbrales, y al final se observa un mejor resultado con un valor de 0.4 para la función `edge()` de MATLAB. En la Figura 3.20 se presentan los resultados del algoritmo de Canny con

cuatro valores diferentes utilizados de umbral, siendo el valor de 0.4 el que muestra la mayoría de las características necesarias de la imagen sin agregar ruido a la misma.

A continuación se definió un elemento estructural de forma cuadrada, el cual será utilizado para aplicar la dilatación a cada una de las imágenes de los tornillos. Este elemento estructural se define como una matriz de 4x4 en la que todos los elementos son 1. La dilatación no se aplica a toda la imagen para reducir tiempos de procesamiento. El tiempo de procesamiento es más rápido en once imágenes de 32x32, que en una imagen de 1280x720 pixeles.

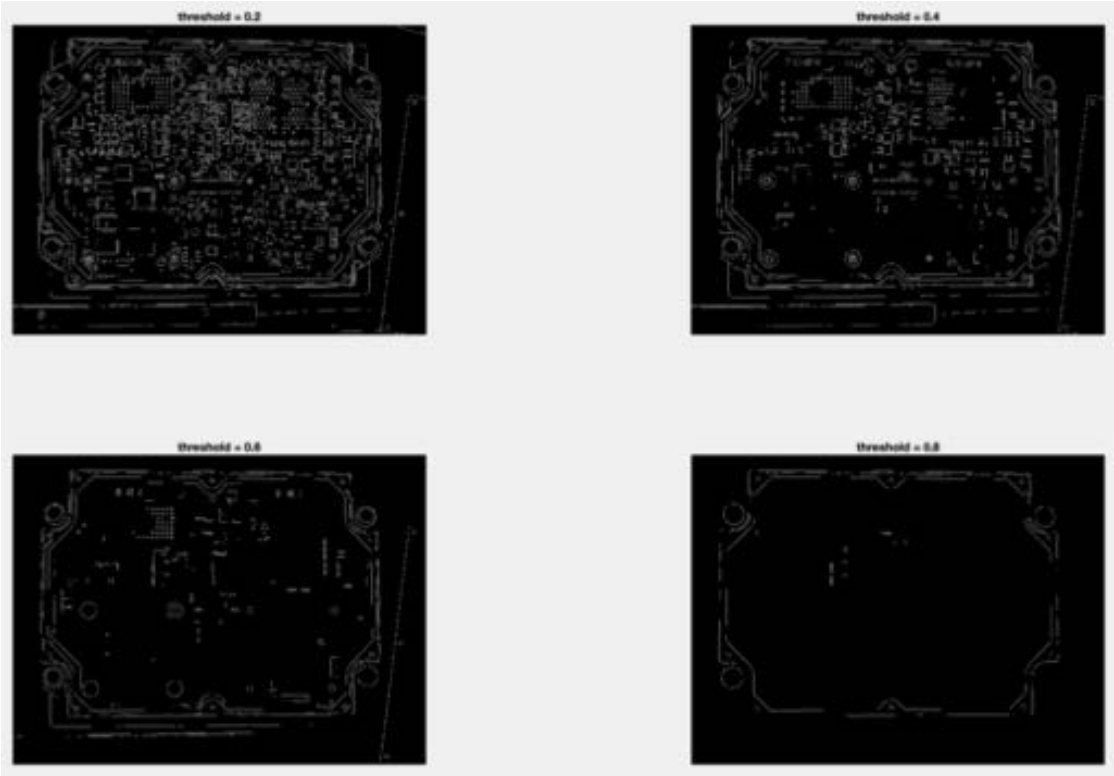


Figura 3.20: Aplicación de filtro Canny con diferentes valores de umbral.

Se tiene un bucle `for`, el cual itera de acuerdo a las distancias entre los tornillos 1 al 4, los cuales deben cumplir con el mismo umbral. Cuando se detecte un valor mayor al umbral establecido, el contador de tornillos incrementará, ya que se ha detectado un tornillo en el área especificada.

Posterior a esto, el resto de los tornillos se analizaron definiendo ubicaciones fijas. Esta es una ventaja que reduce tiempo de procesamiento a la ejecución del programa, sin embargo, al estar definidas las posiciones, si la imagen se mueve, la detección de los tornillos no será verdadera. Por último, se imprime un mensaje en consola donde se menciona la cantidad de tornillos encontrados en la imagen.

Para efectos de depuración, se incluyó una instrucción en el código de la aplicación para visualizar las sub-imágenes de las regiones de interés definidas. En la Figura 3.21 se muestran los resultados de una de las imágenes de prueba.

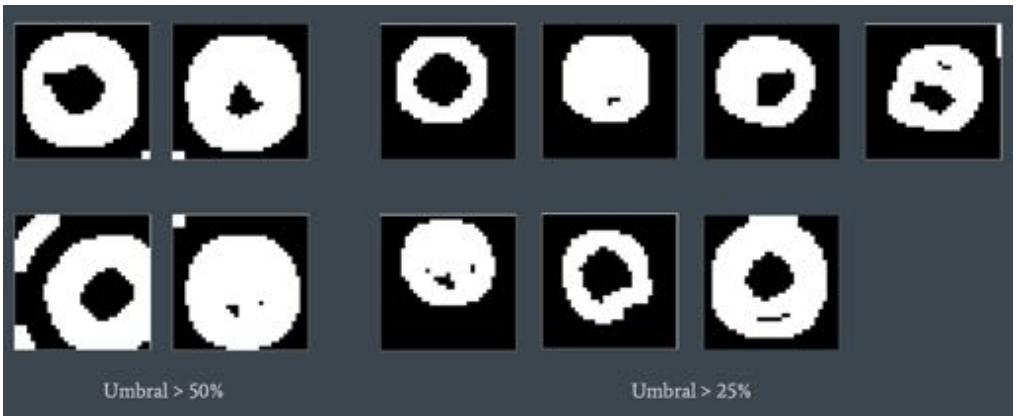


Figura 3.21: Ejemplos de tornillos en imagen procesada.

Debido a las limitaciones del software generado en MATLAB y el alto costo de depuración a un módulo autónomo, se decidió hacer la migración del código a una Raspberry Pi usando el lenguaje de programación Python con la librería OpenCV, la cual es de código abierto, para hacer uso de funciones más avanzadas de procesamiento de imágenes, esperando obtener un mejor resultado y con mayor confiabilidad.

Desarrollo en Python con OpenCV en Raspberry Pi

Para el desarrollo de la aplicación en Python y Raspberry Pi, después de realizar la investigación del estado del arte, se decidió utilizar la técnica de *Template Matching* debido a las ventajas que propone; entre ellas, la capacidad de tener imágenes de referencia de lo que buscamos, la utilización de varios métodos de encontrar coincidencias en una imagen y el tiempo de ejecución.

Dos de los parámetros más importantes en el uso de la función `matchTemplate()` de OpenCV son el método a utilizar para la detección de coincidencias y, posteriormente, el valor del umbral, ya que es el valor numérico del que dependerá la correcta discriminación de coincidencias detectadas falsas. Este valor numérico va desde 0 a 1. Un valor muy bajo de umbral resulta en muchas coincidencias falsas, mientras que un valor muy alto resulta en que no se encuentren coincidencias dentro de la imagen. Los resultados de la detección se pueden ver afectados por el valor del umbral, al igual que por el método utilizado para encontrar coincidencias, ya sea por variaciones de orientación o de iluminación en las imágenes patrón. [28].

El paso inicial fue la generación de un conjunto de imágenes de entrenamiento contra las cuales se realizará la comparación de las imágenes de prueba en el programa. Esto consistió en la captura de fotografías tomadas a 4 tarjetas electrónicas de referencia ensambladas correctamente, extrayendo un conjunto de imágenes de cada uno de los 11 tornillos utilizados para el ensamble, formando 44 imágenes de referencia. En la Figura 3.22 se observa una muestra de 10 de los tornillos del conjunto de entrenamiento. Este conjunto de imágenes es el argumento de entrada para la función de código que se encarga de buscar en la imagen de la unidad de prueba la ubicación de los tornillos y que además los identifica visualmente sobre la misma. Se decidió utilizar este conjunto de entrenamiento para mitigar variaciones que ocurran ya sea en la iluminación o en el ángulo en el que se toma la fotografía.

Una vez completo el conjunto de entrenamiento y de haber obtenido las imágenes

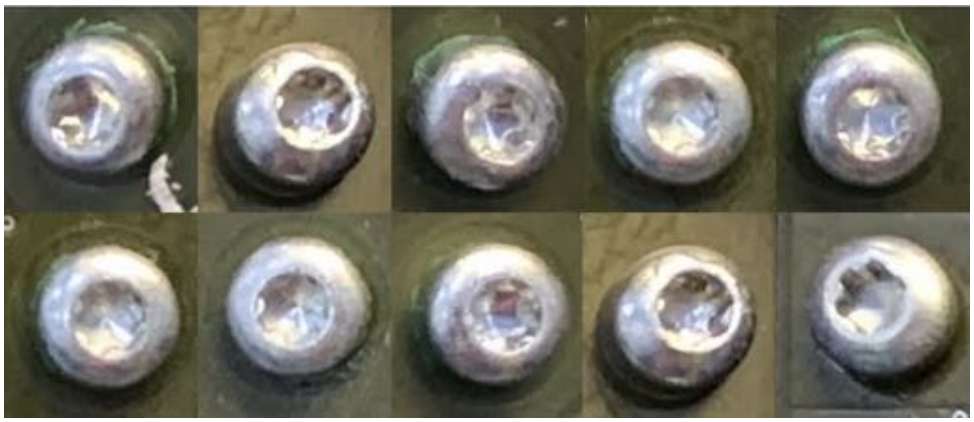


Figura 3.22: Muestra de 10 tornillos del conjunto de entrenamiento.

de prueba, se realizaron pruebas para definir el mejor método de *Template Matching* para utilizar en la aplicación. Utilizando una imagen del conjunto de entrenamiento se utilizó `matchTemplate()` con cada uno de los métodos disponibles para la función. El propósito de esta prueba es para determinar si existe un método que presente mejores resultados en cuanto a la detección de tornillos en la imagen de prueba. Se utilizó una imagen de prueba con 20 imágenes de entrenamiento para cada uno de los métodos de la función de OpenCV. En este caso aún no se asignó un valor de umbral, debido a que se requiere ver en sí el funcionamiento de cada uno de los métodos disponibles.

En la Figura 3.23 se pueden observar los resultados obtenidos con cada uno de los métodos utilizados. Los métodos sin normalizar detectan menos tornillos en la imagen, e incluso detectan áreas de la imagen como coincidencias donde claramente no existen tornillos. En el caso de `CV.TM_CCORR`, no muestra tornillos detectados, por lo que estos métodos se descartan. Los métodos normalizados presentan mejores resultados en contraste. Sin embargo, entre ellos no se observa diferencia en cuanto a la cantidad de tornillos detectados. Se recurre a evaluar el tiempo de ejecución en promedio con 5 imágenes de prueba. El método normalizado `CV.TM_CCOEFF_NORMED` demuestra ser hasta 6 veces más rápido, por lo que se selecciona como el indicado para nuestra aplicación.

En el Algoritmo 2 se muestra la primera versión del algoritmo codificado para la detección de tornillos en imágenes de prueba con la función de *template matching* de OpenCV. En el inicio del programa se almacenan datos de importancia en variables, como el tiempo inicial de ejecución del programa para posteriormente poder calcular el tiempo de ejecución total. También se genera un arreglo en blanco para almacenar todas las imágenes patrón que se buscarán en la imagen de prueba y se define el umbral a utilizar para decidir si se encuentra o no una coincidencia. Este umbral se seleccionó mediante experimentación, siendo el valor que discrimina falsos positivos, sin comprometer la detección de tornillos. En la Figura 3.24 se muestra el resultado de una misma imagen con diferentes valores de umbral. Los valores probados fueron 0.70, con el que se obtuvieron muchas detecciones de objetos que no son tornillos. Con el valor de 0.80 no se detectan objetos incorrectos, sin embargo, en comparación con el resultado del umbral de 0.75, se detectan menos tornillos en la imagen. Es por esto que se decide utilizar el umbral de 0.75.

Una vez definido el conjunto de imágenes de referencia, imágenes de prueba y

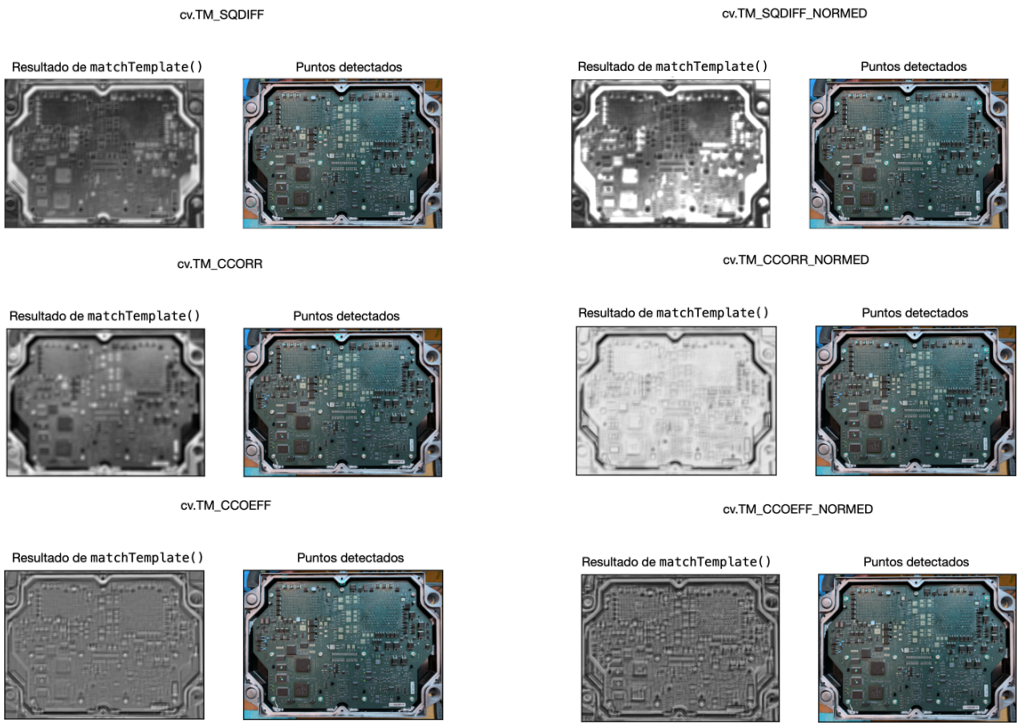
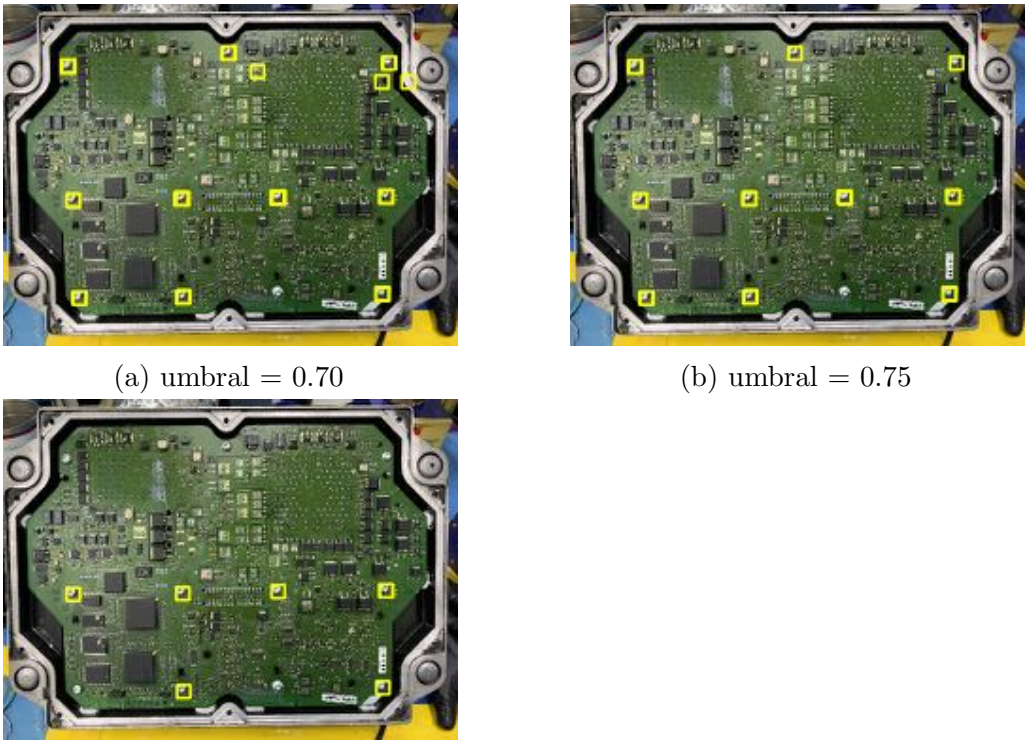


Figura 3.23: Resultados de las pruebas con todos los métodos disponibles de `matchTemplate()`.



(a) umbral = 0.70

(b) umbral = 0.75

(c) umbral = 0.80

Figura 3.24: Comparación de resultados con diferentes valores de umbral.

umbral a utilizar, se procede a realizar la codificación en Python del Algoritmo 2. Una vez capturada la imagen de prueba, se hace una copia de esta y se convierte a escala de grises. Este paso se realiza antes de la ejecución del bucle `for` para evitar ejecutar la conversión de la imagen múltiples veces por ejecución.

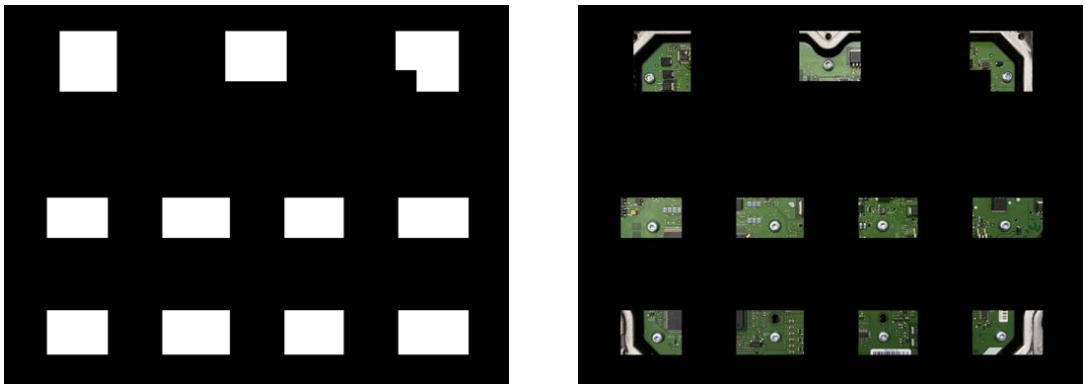
Es necesario contener el conjunto de entrenamiento de tornillos generado para la aplicación en un arreglo para poder utilizarlos de manera nativa en Python y facilitar la iteración de cada uno de los elementos. La función `matchTemplate()` de OpenCV sólo acepta una imagen patrón por ejecución [29]. Esta es la manera en la que la función se encuentra implementada en OpenCV. Para utilizar múltiples imágenes de entrenamiento se tienen que recurrir a técnicas de programación como la implementación de bucles iterativos.

Algoritmo 2: Algoritmo para detectar tornillos en Python y OpenCV (v1)

```
   salida  : I_map, t
 1 inicializar contador de tiempo de ejecución start_time;
 2 inicializar arreglo en blanco para almacenar imágenes patrón templates[];
 3 definir umbral, threshold ← 0,75;
 4 capturar fotografía con cámara;
 5 almacenar fotografía en variable img;
 6 convertir imagen a escala de grises con img_grey =
     cv.cvtColor(img,cv.COLOR_BGR2GRAY);
   // cargar imágenes patrón a arreglo templates
 7 for j = 1 to nTemplates do
 8 |   templates[j] = j.jpg;
 9 foreach template ∈ templates[] do
10 |   res = cv.matchTemplate(img_grey,template,method);
11 |   guardar en loc puntos de la imagen en los que res >= threshold;
12 |   foreach punto ∈ loc do
13 |   |   dibujar rectángulo en img;
14 I_map = img;
15 almacenar I_map en disco;
16 t = time.time() - startTime;
17 return I_map;
18 return t;
```

La línea 9 del Algoritmo 2 se puede leer como: *para cada template en el arreglo templates hacer*. El bloque de código siguiente son las instrucciones que se ejecutarán por cada imagen patrón contenida en el arreglo `templates`. Se utiliza la función `matchTemplate()`, con la cual se hará el barrido de la imagen de prueba con la imagen patrón o *template* correspondiente. En la variable `loc` se almacena la imagen resultante de la función anterior, sin embargo, sólo se almacenan los puntos en los que el valor es mayor o igual al umbral definido. Posteriormente, por cada uno de estos puntos se dibuja un rectángulo de color amarillo en la imagen original para identificar los tornillos detectados.

Por último, se almacena en disco la imagen con los rectángulos indicando los tornillos detectados y se imprime en pantalla el tiempo total de ejecución para poder medirlo y analizarlo.



(a) Imagen utilizada como máscara. (b) Resultado de la operación `bitwise_and`.

Figura 3.25: Utilización de máscara para reducir falsos positivos.

Esta primera versión del algoritmo funciona correctamente y tiene un porcentaje de detección aceptable. Sin embargo, evaluando los resultados (mostrados a detalle en la sección de resultados), se identificaron oportunidades de mejora en el código, con las cuales se mitigan la detección de características de la tarjeta que no son tornillos. Además, el tiempo de ejecución era alto, alcanzando los 131 segundos por imagen de prueba.

En la figura se muestra un ejemplo en el que se detectaban componentes y placas de disipación de calor como tornillos. Para reducir este tipo de errores, se implementó el uso de una imagen como máscara que se colocará sobre la imagen de prueba en cada ejecución. La función de *template matching* de OpenCV soporta nativamente la utilización de máscaras pero únicamente con los métodos de detección `CV.TM_SQDIFF` y `CV.TM_CCORR_NORMED`, por lo que hay que recurrir a otras funciones para ejecutar esta operación. Se utiliza la función `bitwise_and()` de OpenCV. Esta función realiza la operación lógica `and` entre dos arreglos proporcionados como argumentos de entrada (en este caso la imagen de prueba y la máscara). La Ecuación 3.1 describe la operación realizada por esta función.

$$\text{dst}(\boldsymbol{I}) = \text{src1}(\boldsymbol{I}) \wedge \text{src2}(\boldsymbol{I}), \quad \text{mask}(\boldsymbol{I}) \neq 0 \tag{3.1}$$

Donde scr1 y scr2 son las dos imágenes con las que se realizará la operación, y mask es la máscara utilizada, también una imagen. La máscara generada y el resultado de la operación `bitwise_and` con ella se puede observar en la Figura 3.25.

Para la reducción de los tiempos de ejecución se implementó la reducción de tamaño de las fotografías y de la máscara a un porcentaje de su tamaño original. Hacer esto ayuda a que la carga para el CPU y GPU sea menos y las manipulaciones a la imagen sean más rápidas. Además se optimizó el código para eliminar instrucciones innecesarias, como la parte en la que se almacenan las imágenes patrón de los tornillos en el arreglo `templates[]`. Esto se incluye en la iteración del bucle `for`. Se utiliza la variable de iteración para cargar directamente de disco imagen por imagen para pasarlas a la función de *Template Matching*. En el Algoritmo 3 se presenta el pseudocódigo de la segunda versión del funcionamiento del detector de tornillos.

Algoritmo 3: Algoritmo para detectar tornillos en Python y OpenCV (v2)

salida : `I_map, t`

```
1  inicializar contador de tiempo de ejecución start_time;
2  inicializar arreglo en blanco para almacenar imágenes patrón templates[];
3  definir umbral, threshold = 0.75;
4  definir escala de imágenes, scale = 0.12;
5  leer máscara y almacenar en variable mask;
6  capturar fotografía con cámara;
7  almacenar fotografía en variable img;
8  escalar img a 12 %;
9  escalar mask a 12 %;
10 convertir imagen a escala de grises con img_grey =
     cv.cvtColor(img,cv.COLOR_BGR2GRAY);
   // cargar imágenes patrón a arreglo templates
11 for j = 1 to nTemplates do
12 |   template = j.jpg;
13 |   escalar template a 12 %;
14 |   res = cv.matchTemplate(img_grey,template,method);
15 |   guardar en loc puntos de la imagen en los que res >= threshold;
16 |   foreach punto ∈ loc do
17 |   |   dibujar rectángulo en img;
18 I_map = img;
19 almacenar I_map en disco;
20 t = time.time() - startTime;
21 return I_map;
22 return t;
```

Implementación de detector de tornillos

El sistema de detección de tornillos no se encuentra implementado en la empresa actualmente. Se encuentra como prototipo y está siendo revisado por el equipo de Sistemas de Información por la utilización de la Raspberry Pi. Estos dispositivos no se utilizan ampliamente dentro de la empresa, por lo que se requieren aprobaciones adicionales para permitir su uso dentro de los procesos operativos.

El desarrollo del software de este prototipo terminó en Octubre de 2020. De igual manera, se siguió la metodología 6 Sigma para la solución de problemas encontrando la causa raíz del problema.

Capítulo 4

Resultados

En esta sección se presentará el resumen de los resultados obtenidos de los experimentos dirigidos para la validación del funcionamiento de las soluciones propuestas. Los resultados se dividirán en dos partes, al igual que el capítulo anterior: la primera parte presentará los resultados de la implementación del sistema AOI y la receta de inspección desarrollada, mientras que la segunda parte tratará los resultados de los experimentos realizados con el detector de tornillos.

4.1 AOI

De las pruebas realizadas con las tarjetas electrónicas inspeccionadas en la máquina AOI con la receta completa se obtuvieron los datos de Pasa o Falla arrojados después de cada ciclo de trabajo por el programa de la máquina. Con ellos, se realizó el análisis de repetitividad y reproducibilidad (R&R) de atributos. El análisis se realizó en los siguientes niveles: por operador, por operador contra el estándar, entre operadores y todos los operadores contra el estándar. El estándar se refiere a la lista de unidades conocidas buenas y malas. En el Cuadro 4.1 se presenta el estándar de las 31 unidades utilizadas para la validación. Las unidades descritas como Defecto cuentan con los defectos generados de acuerdo al Cuadro 3.1 de la Sección 3.2.1 del Capítulo 3.

En el primer turno de cada operador, las unidades se les proporcionaron en el orden del Cuadro 4.1. En el segundo turno, el orden se alteró de manera aleatoria para evitar que el operador identificara el resultado esperado de cada unidad.

Cada uno de los operadores presentó un 100 % de detección correcto de las unidades. Esto es sin comparar contra el estándar, es decir, todas las unidades arrojaron el mismo resultado en ambos turnos de pruebas. Sin embargo, al comparar contra el estándar, el porcentaje baja a 96.77 %. Esto se debe a que una de las tarjetas con defecto fue evaluada como aceptable. Dicha tarjeta debía presentar el componente C213 ausente, sin embargo, en su lugar tenía restos de soldadura que ocupaban volumen suficiente para ser detectados como volumen suficiente, por lo cual el algoritmo dio positivo de presencia de componente. Sin embargo, los tres operadores en ambos turnos dieron el mismo resultado con esta unidad, es por eso que el porcentaje del análisis entre operadores es de 100 %. Al comparar todos los operadores contra el estándar, el resultado es 96.77 %.

En los Cuadros 4.2, 4.3 y 4.4 se presentan los resultados del análisis de R&R.

Unidad	Estándar
1	Buena
2	Buena
3	Buena
4	Buena
5	Buena
6	Buena
7	Defecto
8	Buena
9	Defecto
10	Buena
11	Buena
12	Buena
13	Defecto
14	Buena
15	Buena
16	Defecto
17	Buena
18	Buena
19	Defecto
20	Buena
21	Defecto
22	Buena
23	Buena
24	Defecto
25	Buena
26	Buena
27	Buena
28	Buena
29	Defecto
30	Defecto
31	Defecto

Cuadro 4.1: Estándar utilizado para evaluar tarjetas de prueba en máquina AOI.

Operador	Inspecciones	Coincidencias	Porcentaje
1	31	31	100 %
2	31	31	100 %
3	31	31	100 %

Cuadro 4.2: Resultados de análisis R&R de atributos por cada operador consigo mismo.

Operador	Inspecciones	Coincidencias	Porcentaje
1	31	30	96.77 %
2	31	30	96.77 %
3	31	30	96.77 %

Cuadro 4.3: Resultados de análisis R&R de atributos por cada operador contra el estándar.

Operador	Inspecciones	Coincidencias	Porcentaje
1	31	30	96.77 %
2	31	30	96.77 %
3	31	30	96.77 %

Cuadro 4.4: Resultados de análisis R&R de atributos de todos los operadores contra el estándar.

4.2 Detección de tornillos

Se logró inspeccionar visualmente el ensamble de tarjetas electrónicas para detectar tornillos ausentes para proporcionar un control en el proceso de ensamble de las mismas. Inicialmente se trabajó con el prototipo programado en MATLAB, sin embargo, los resultados no fueron satisfactorios. Esto fue debido a que no se contó con las herramientas de visión artificial y, considerando las posibilidades de despliegue de la aplicación a un ejecutable, no fue la mejor opción debido al alto costo de la licencia del software. Además, codificar de manera fija las ubicaciones de los tornillos, con un tamaño constante, cualquier variación en ubicación del tornillo que superara los 32 pixeles en la imagen de prueba, la detección de los tornillos se veía afectada.

De las pruebas descritas en la Sección 3.2.2 del Capítulo 3 para validar el porcentaje de detección del algoritmo realizado en Python con OpenCV, los resultados obtenidos de la primera prueba se muestran en el Cuadro 4.5. En ella se utilizó el conjunto de entrenamiento de 10 tornillos con las 10 imágenes de prueba de tarjetas electrónicas. Se demuestra que en el mejor de los casos se tiene un porcentaje de detección de tornillos presentes en el ensamble de 100 %, mientras que el peor caso (la unidad 10 eliminando en tornillo 4 del conjunto de entrenamiento) se obtiene un porcentaje de detección de los tornillos de 63.64 %. El promedio de detección del sistema con este conjunto de entrenamiento es de 87.18 %. En las Figuras 4.1a y 4.1b se observan el mejor y peor caso, respectivamente, de detección con esta prueba. Se muestra además, cómo en ambos casos se detectan dos características adicionales que no son tornillos, este tipo de detecciones son los falsos positivos, ya que son coincidencias que se encuentran, pero no son características que se desean detectar en la imagen.

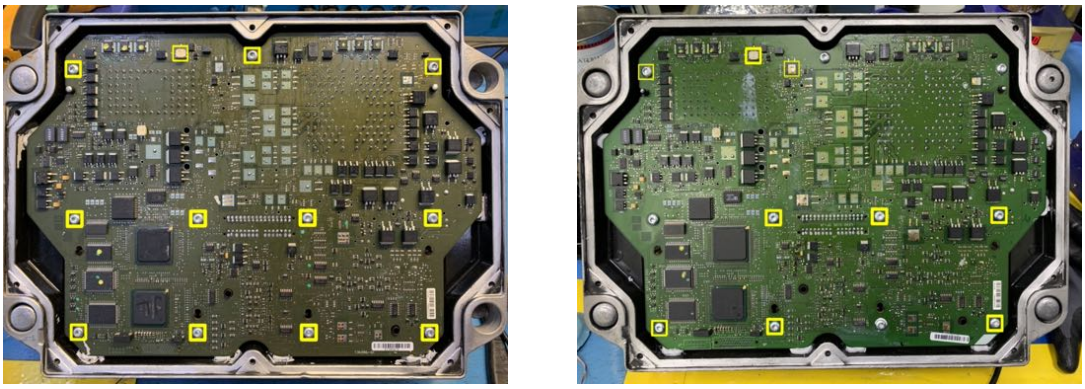
En la segunda demostración realizada se tomó el conjunto de entrenamiento de 20 tornillos para las mismas 10 imágenes de prueba. En el Cuadro 4.6 se muestran los resultados de estas iteraciones; mostrando de nuevo un mejor caso de 100 % con las primeras 4 unidades, mientras que el peor caso se vuelve a presentar en la imagen 10 con el tornillo 4, mostrando porcentaje de detección de tornillos presentes de 63.64 %. El porcentaje de detección del sistema con este conjunto de entrenamiento es de 87.23 %. En las Figuras 4.2a y 4.2b se observan el mejor y peor caso, respectivamente, de detección con esta prueba. Al igual que en la prueba anterior, se detectan las

Porcentaje de detección											
T_{out}	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	**%detección**
1	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
2	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
3	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
4	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	63.64 %	**86.36 %**
5	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
6	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
7	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
8	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
9	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
10	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
% detección	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**90.91 %**	**72.73 %**	**90.91 %**	**72.73 %**	**72.73 %**	**71.82 %**	**87.18 %**

Cuadro 4.5: Resultados (en porcentaje) de detección con un conjunto de entrenamiento de 10 tornillos.

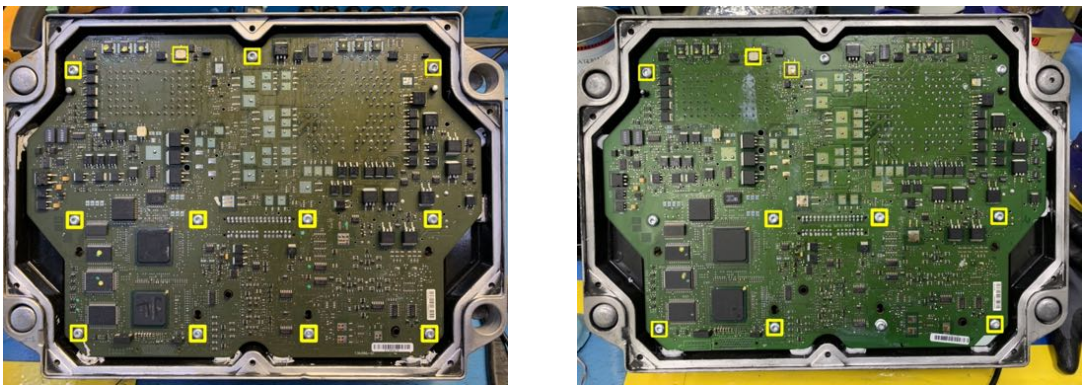
Porcentaje de detección											
T_{out}	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	**%detección**
1	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
2	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
3	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
4	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	63.64 %	**86.36 %**
5	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
6	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
7	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
8	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
9	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
10	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
11	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
12	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
13	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
14	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
15	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
16	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
17	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
18	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
19	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
20	100 %	100 %	100 %	100 %	90.91 %	72.73 %	90.91 %	72.73 %	72.73 %	72.73 %	**87.27 %**
% detección	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**90.91 %**	**72.73 %**	**90.91 %**	**72.73 %**	**72.73 %**	**71.82 %**	**87.23 %**

Cuadro 4.6: Resultados (en porcentaje) de detección con un conjunto de entrenamiento de 20 tornillos.



(a) Mejor caso. (b) Peor caso.

Figura 4.1: Mejor y peor caso de resultados de inspección con conjunto de entrenamiento de 10 tornillos.



(a) Mejor caso. (b) Peor caso.

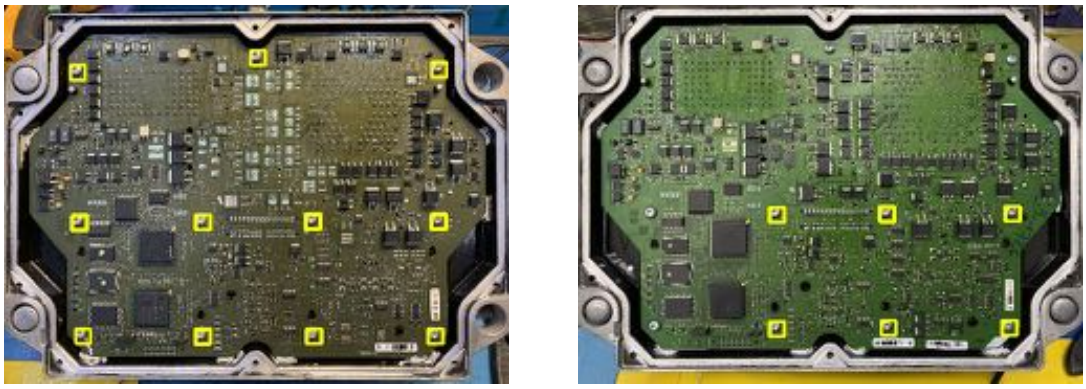
Figura 4.2: Mejor y peor caso de resultados de inspección con conjunto de entrenamiento de 20 tornillos.

mismas características que no son tornillos como coincidencias.

En promedio, existe un incremento en el tiempo de ejecución de 67.91 segundos para obtener un 0.05 % de mejora en el porcentaje de detección de tornillos. Es por esto que se decide utilizar el subconjunto de 10 tornillos de entrenamiento en el prototipo final, ya que proporciona un tiempo de ejecución menor con una diferencia no significativa en el porcentaje de detección de nuestra aplicación.

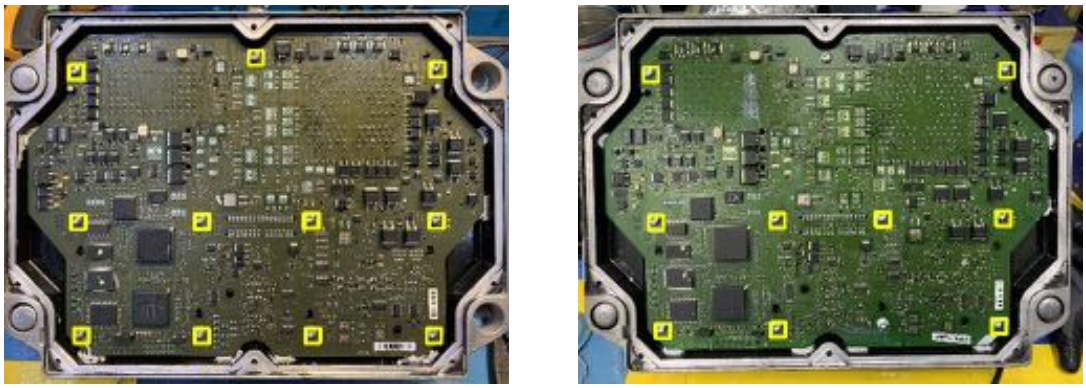
El porcentaje de detección de las pruebas con los conjuntos de 10 y 20 tornillos es superior a 80 %. Esto es un buen valor de confiabilidad para el sistema, sin embargo, los tiempos de ejecución son largos y, en ocasiones, se detectan objetos en la tarjeta que no son tornillos, lo cual genera rectángulos adicionales en la imagen final. Es por esto que se continuó depurando y mejorando el código de la aplicación, así como la generación de una máscara para eliminar porciones de la tarjeta en la que no es necesario buscar tornillos.

En el Cuadro 4.7 se presentan los resultados de las mismas pruebas una vez realizadas las mejoras del código, utilizando un conjunto de referencia de 10 tornillos. Se observa claramente una mejora en la detección de tornillos, presentando un promedio general de 90.18 %, sin embargo, se nota que en estas pruebas afecta la eliminación del tornillo 1 en el conjunto de entrenamiento, específicamente en



(a) Mejor caso. (b) Peor caso.

Figura 4.3: Mejor y peor caso de resultados de inspección con conjunto de entrenamiento de 10 tornillos después de ejecutar el algoritmo con los cambios en el tamaño de las imágenes de prueba y el uso de la máscara.



(a) Mejor caso. (b) Peor caso.

Figura 4.4: Mejor y peor caso de resultados de inspección con conjunto de entrenamiento de 20 tornillos después de ejecutar el algoritmo con los cambios en el tamaño de las imágenes de prueba y el uso de la máscara.

las unidades 6 (ahora nuestro peor caso) y 9. El tiempo de ejecución se reduce significativamente. En la Figura 4.3 se observan el mejor y el peor caso para estas pruebas. Cabe mencionar que ya no se detectan falsos positivos en estos casos. La eliminación de las áreas que no son de interés en la imagen ayuda a eliminarlos.

Incrementando el número de imágenes de referencia, se repitieron las pruebas para validar si existe una mejora considerable entre utilizar un conjunto de patrones mayor. Los resultados de los porcentajes de detección se presentan en el Cuadro 4.8. En el peor de los casos, el porcentaje de detección de tornillos instalados es de 81.82 %, el cual sigue siendo un valor aceptable, pero también se observa que en promedio, el porcentaje de detección del sistema aumenta hasta 98.91 %. Esta es una mejora muy significativa, ya que es una diferencia de más del 10 % contra la detección antes de los cambios realizados al código y nos indica que las fotografías patrón de los tornillos del 11 al 20 incrementan el porcentaje de detección. Sin embargo, de los resultados también se observa que el tornillo 1 es importante para mantener el porcentaje de detección. En la Figura 4.4 se presentan el mejor y el peor caso de los resultados de esta prueba.

	Porcentaje de detección										
T_{out}	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	**%detección**
1	100.00 %	100.00 %	100.00 %	90.91 %	90.91 %	54.55 %	90.91 %	72.73 %	63.64 %	72.73 %	**83.64 %**
2	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
3	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
4	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
5	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
6	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
7	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
8	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
9	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
10	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	63.64 %	100.00 %	81.82 %	72.73 %	90.91 %	**90.91 %**
%detección	**100.00 %**	**100.00 %**	**100.00 %**	**99.09 %**	**99.09 %**	**62.73 %**	**99.09 %**	**80.91 %**	**71.82 %**	**89.09 %**	**90.18 %**

Cuadro 4.7: Resultados (en porcentaje) de detección con un conjunto de entrenamiento de 10 tornillos después de ejecutar el algoritmo con los cambios en el tamaño de las imágenes de prueba y el uso de la máscara.

	Porcentaje de detección										
T_{out}	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	**%detección**
1	100.00 %	100.00 %	100.00 %	90.91 %	100.00 %	90.91 %	100.00 %	90.91 %	100.00 %	81.82 %	**95.45 %**
2	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
3	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
4	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
5	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
6	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
7	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
8	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
9	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
10	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
11	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
12	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
13	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
14	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
15	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
16	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
17	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
18	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
19	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
20	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	90.91 %	**99.09 %**
%detección	**100.00 %**	**100.00 %**	**100.00 %**	**99.55 %**	**100.00 %**	**99.55 %**	**100.00 %**	**99.55 %**	**100.00 %**	**90.45 %**	**98.91 %**

Cuadro 4.8: Resultados (en porcentaje) de detección con un conjunto de entrenamiento de 20 tornillos después de ejecutar el algoritmo con los cambios en el tamaño de las imágenes de prueba y el uso de la máscara.

Además, se observa un tiempo de ejecución menor en comparación con las ejecuciones antes de realizar las mejoras de la máscara y optimizaciones de código utilizando el conjunto de 20 tornillos. El tiempo promedio de ejecución es 13.32 veces menor al de las primeras pruebas con el mismo tamaño de conjunto de entrenamiento.

A manera de comparación, en el Cuadro 4.9 se resumen los resultados obtenidos con las dos versiones del algoritmo codificado en Python. La versión 2 presenta una mejora significativa en el porcentaje de detección, así como en el tiempo de ejecución. La versión final del código se encuentra en el Apéndice B.

Utilizando el programa con el algoritmo optimizado, se procede a realizar una prueba con un número significativo de unidades para calcular el tiempo promedio de

	Tamaño de conjunto de entrenamiento	% de detección	Tiempo promedio (s)
Versión 1 de algoritmo	10	87.18 %	60.45
	20	87.23 %	128.36
Versión 2 de algoritmo	10	90.18 %	8.41
	20	98.91 %	9.63

Cuadro 4.9: Comparación de ambas versiones del algoritmo para la detección de tornillos.

Porcentaje de detección				
Unidad	**Prueba 1**	**Prueba 2**	**Prueba 3**	**% detección**
1	100.00 %	100.00 %	100.00 %	100.00 %
2	100.00 %	100.00 %	100.00 %	100.00 %
3	100.00 %	100.00 %	100.00 %	100.00 %
4	100.00 %	100.00 %	100.00 %	100.00 %
5	100.00 %	100.00 %	100.00 %	100.00 %
6	100.00 %	100.00 %	100.00 %	100.00 %
7	100.00 %	100.00 %	100.00 %	100.00 %
8	100.00 %	100.00 %	100.00 %	100.00 %
9	100.00 %	100.00 %	100.00 %	100.00 %
10	90.91 %	90.91 %	90.91 %	90.91 %
%detección	**99.09 %**	**99.09 %**	**99.09 %**	**99.09 %**

Cuadro 4.10: Resultados (en porcentaje) de detección del programa con el algoritmo 2 utilizando todos los tornillos del conjunto de entrenamiento de tamaño 20.

ejecución, así como el porcentaje de detección de la aplicación utilizando el conjunto completo de tornillos de entrenamiento. Se hizo la prueba con las 10 tarjetas probando 3 veces cada una. En el Cuadro 4.10 podemos ver que la detección utilizando un conjunto de entrenamiento de 20 tornillos sin quitar ninguno es de 99.09 %.

La técnica de *Template Matching* para detectar varios objetos en un sólo procedimiento incrementa las capacidades de detección de la aplicación. La creación de conjuntos de entrenamiento para la aplicación la hace que esta sea más robusta y que la detección de los tornillos presentes sea más confiable y precisa. Adicionalmente, la utilización de una máscara para la eliminación de áreas de la tarjeta que no son de interés para la aplicación, así como la reducción de tamaño de las imágenes, ayuda a reducir los tiempos de procesamiento al ocupar menos memoria y ciclos del CPU y GPU. Un ciclo de trabajo promedio de 9.63 segundos con un porcentaje de detección de 99.09 % es bastante bueno considerando los niveles de volumen de producción que se requieren en la empresa para la que se desarrolló este prototipo.

Un sistema convencional de redes neuronales para la detección y clasificación de características de objetos [30] tiene un porcentaje de reconocimiento del 98.4 % utilizando un procesador de cuatro núcleos a 2.80 GHz y 16 GB de RAM. Otra aplicación demuestra un porcentaje de reconocimiento de 80 % [31] utilizando un CPU de doble núcleo a una velocidad de 2.27 GHz y 4 GB de RAM, logrando además, tiempos de ejecución de 18 milisegundos con imágenes de prueba de resolución de 2048 x 2048. Sin embargo, nuestro sistema, en condiciones controladas puede lograr un 100 % y en condiciones no controladas un promedio de 98.91 %. El porcentaje de reconocimiento de la aplicación con Raspberry Pi es bueno en comparación con sistemas implementados en hardware más robusto, considerando que las características de este dispositivo son de bajo costo.

Capítulo 5

Conclusiones y Trabajo Futuro

La implementación de un sistema de AOI ayuda a inspeccionar una mayor cantidad de componentes en las tarjetas electrónicas en un tiempo menor al que lleva realizarla en el proceso de remanufactura actual. El porcentaje de detección de defectos en los componentes programados demostrado es de 96.77 %, mientras que el tiempo de ciclo promedio es de 1 minuto por tarjeta. Esto es de gran impacto debido a que la cantidad de componentes inspeccionados con el sistema AOI representa un 36 % del número total de componentes en las tarjetas, siendo que sin el equipo, anteriormente solo se inspeccionaba el 0.98 % de los componentes. El sistema solamente inspecciona el 36 % de los componentes de la tarjeta debido a que inicialmente sólo se realizó la programación de las zonas conocidas que más se ven afectadas por las operaciones de desensamble y actualización, en las que la tarjeta se somete a calor y a procesos de soldadura, ocasionando que se remuevan componentes por el calor aplicado o que el trabajo de soldadura no sea elaborado correctamente. Como parte de la mejora continua y de trabajo futuro, el objetivo es llegar a que se inspeccione el 100 % de los componentes electrónicos de las tarjetas.

Se detectaron otras oportunidades en las tarjetas electrónicas que afectaron negativamente la detección del sistema, como la limpieza de las tarjetas al desensamblarlas, así como el retrabajo de las mismas. En una de las tarjetas se generó un defecto conocido de componente ausente el cual mostró soldadura aún en sus terminales. La soldadura fue suficiente para que se detectara como componente. Estos sobrantes de soldadura en las tarjetas es algo común en el proceso debido a que las unidades son reparadas y retrabajadas de manera manual. De igual manera, el proceso no cuenta con una limpieza a nivel tarjeta, por lo que también se llegan a encontrar partículas de sellos, pasta para soldadura, rebabas y demás. Atacando estas oportunidades para tener un proceso más controlado ayudaría a poder llevar la detección al 100 %.

Existen defectos físicos que afectan a las PCB y que el sistema de AOI no es capaz de detectar; por ejemplo, tarjetas con terminales de componentes que se encuentran separadas de la pista conductiva. Estos defectos actualmente se encuentran cuando se utiliza una herramienta puntiaguda para mover cada una de las terminales. No es posible detectar estas tarjetas visualmente, por lo que el problema rebasa las capacidades del sistema AOI. Otro de los defectos que el sistema AOI no detecta es la rotura de la soldadura o soldaduras frías. Este tipo de defectos es difícil de detectan con sistemas de visión; incluso con sistemas de rayos X, por lo que son defectos que pueden ir presentes en las tarjetas electrónicas y no ser detectadas dentro del proceso de remanufactura.

También se propone un sistema para la inspección automatizada de tornillos presentes en el ensamble de tarjetas electrónicas. La implementación de este sistema en Raspberry Pi le da un giro innovador al dispositivo de bajo costo al utilizarlo en un ambiente industrial. Se aprovecha la flexibilidad de Python en conjunto con la librería OpenCV para el desarrollo del software de inspección.

Comparando la implementación de este sistema con otras aplicaciones similares, se encuentra que el porcentaje de detección logrado de 99.09 % se encuentra aceptable, ya que en aplicaciones que hacen uso de hardware especializado y robusto se logran porcentajes menores y mayores.

El ciclo de inspección se logró en un tiempo de ejecución promedio de 9.63 segundos. Este tiempo de ejecución es suficiente para la aplicación mostrada; sin embargo, este puede reducirse al migrar el sistema a un procesador con mayor velocidad y mayor capacidad de memoria RAM, como podría ser una Raspberry Pi 4 o un ordenador.

Como trabajo futuro, queda la propuesta de implementar el sistema mecánico y automatizado para la detección de estos tornillos. Es decir, que la carga y descarga de las unidades en el inspector sea tal que no exista oportunidad de error por parte del operador. De igual manera, requerir en la línea de producción que las unidades no puedan ser enviadas a los pasos siguientes sin haber sido inspeccionadas correctamente. En cuanto al detector de tornillos en sí, se puede reducir aún más el tiempo de ejecución al optimizar el código para que ignore inspecciones en áreas de la tarjeta en la que ya se encontró un tornillo.

Con la implementación de este tipo de aplicaciones en dispositivos Raspberry Pi, se busca promover su desarrollo en sistemas industriales relacionados con remanufactura de tarjetas electrónicas.

Con estos dos esfuerzo de contención de problemas de calidad mediante visión artificial, se logran porcentajes de detección mayores al 80 % propuesto en la hipótesis del presente trabajo.

Apéndice A

Código de Detector de Tornillos - Prototipo de MATLAB

```
tic;
%clear all variables
clear all

%import image
img1 = imread('a4_1.jpg');
chan1 = img1(:,:,1);
chan2 = img1(:,:,2);
chan3 = img1(:,:,3);

%convert image to greyscale
img1 = 0.2989*chan1 + 0.5870*chan2 + 0.1140*chan3;

%apply Gaussian filter, sigma = 1.0
img1Blur = imgaussfilt(img1,1.0);

%define horizontal and vertical distances between screws
hDistance = 265;
vDistance = 240;

%define screw image size
screwSize = floor(32/2);

%starting point in image
initialRef = [483 236];

%define threshold values for screws
threshold_1 = 50;
threshold_2 = 25;

%initialize counter to zero
counter = 0;
```

```
%apply Canny filter with threshold = 0.4
BW = edge(img1,'Canny',0.2);

%define structural element for dilation
se = strel('square',4);

%analyze screws 1-4
for i=initialRef(2):hDistance:(hDistance*2)
    for j=initialRef(1):vDistance:(initialRef(1)+vDistance)
        ref = BW(j-screwSize:j+screwSize,i-screwSize:i+
            screwSize);
        BWd = imdilate(ref,se);
        val = sum(sum(BWd));
        if (val/screwSize^2) > 0.5
            counter = counter + 1;
        end
    end
end

% analyze screw 5
%get screw sub-section from image
ref = BW(486-16:486+16,738-16:738+16);
BWd = imdilate(ref,se); %dilate 32x32 image
val = sum(sum(BWd));    %get qty of 1s in image
%convert to percentage
percentage = (val / (size(ref,1)^2))*100;
%evaluate percentage against threshold
if percentage > threshold_2
    %if valid screw, increase counter
    counter = counter + 1;
end

%analyze screw 6
ref = BW(727-16:727+16,738-16:738+16);
BWd = imdilate(ref,se);
val = sum(sum(BWd));
percentage = (val / (size(ref,1)^2))*100;
if percentage > threshold_2
    counter = counter + 1;
end

%analyze screw 7
ref = BW(486-16:486+16,999-16:999+16);
BWd = imdilate(ref,se);
val = sum(sum(BWd));
percentage = (val / (size(ref,1)^2))*100;
if percentage > threshold_2
    counter = counter + 1;
```

```
end

%analyze screw 8
ref = BW(727-16:727+16,991-16:991+16);
BWd = imdilate(ref,se);
val = sum(sum(BWd));
percentage = (val / (size(ref,1)^2))*100;
if percentage > threshold_2
    counter = counter + 1;
end

%analyze screw 9
ref = BW(163-16:163+16,234-16:234+16);
BWd = imdilate(ref,se);
val = sum(sum(BWd));
percentage = (val / (size(ref,1)^2))*100;
if percentage > threshold_2
    counter = counter + 1;
end

%analyze screw 10
ref = BW(139-16:139+16,620-16:620+16);
BWd = imdilate(ref,se);
val = sum(sum(BWd));
percentage = (val / (size(ref,1)^2))*100;
if percentage > threshold_2
    counter = counter + 1;
end

%analyze screw 11
ref = BW(169-16:169+16,999-16:999+16);
BWd = imdilate(ref,se);
val = sum(sum(BWd));
percentage = (val / (size(ref,1)^2))*100;
if percentage > threshold_2
    counter = counter + 1;
end

%print total of screws found
fprintf('Found %d screws\n', counter)

toc;
```

Apéndice B

Código de Detector de Tornillos - Python y OpenCV

```
import cv2 as cv
import numpy as np
import time
from picamera import PiCamera

camera = PiCamera()
camera.start_preview()
sleep(5)
camera.capture('prueba.jpg')
camera.stop_preview()

start_time = time.time()

templates = []
templ_shapes = []
threshold = 0.75
escala_imagen = 12

mask = cv.imread("mask1.png",0)

img_rgb = cv.imread("prueba.jpg")
img_orig = img_rgb
width = int(img_rgb.shape[1]*escala_imagen / 100)
height = int(img_rgb.shape[0]*escala_imagen / 100)
newsize = (width,height)
img_rgb = cv.resize(img_rgb,newsize)
img_map = img_rgb
mask = cv.resize(mask,newsize)
img_rgb = cv.bitwise_and(img_rgb,img_rgb,mask = mask)
img_grey = cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)

for i in range(1,21):
    temptemp = cv.imread("{}.jpg".format(i),0)
```

```
        width = int(temptemp.shape[1]*escala_imagen / 100)
        height = int(temptemp.shape[0]*escala_imagen / 100)
        newsize = (width,height)
        temptemp = cv.resize(temptemp,newsize)
        templates.append(temptemp)

for template in templates:
        res = cv.matchTemplate(img_grey,template,cv.
            TM_CCOEFF_NORMED)
        loc = np.where(res >= threshold)
        for pt in sorted(zip(*loc[::-1])):
            cv.rectangle(img_map, pt, (pt[0] + 14, pt[1] + 14),
                (0,255,255),2)

cv.imwrite("res_out.jpg",img_map)
print("--- %s seconds ---" %(time.time() - start_time))
```

Apéndice C

Aceptación Para Revisión de Artículo de Divulgación en Revista Reaxión

From: **Comite Revista UTL** comiterevista@utleon.edu.mx
Subject: RE: Propuesta para colaboración
Date: November 12, 2020 at 3:59 PM
To: Alejandro Martínez alejmlara@gmail.com
Cc: MIGUEL ANGEL OCHOA VILLEGAS miguel.angel.ochoa@itnl.edu.mx, RENE SANJUAN GALINDO rene.sanjuan@itnl.edu.mx, juan.antonio.rojas@itnl.edu.mx

Estimados Alejandro Martínez-Lara, Juan Antonio Rojas-Estrada (D. E. P.), Rene Sanjuan-Galindo y Miguel Ángel Ochoa-Villegas:

Confirmamos la recepción de la declaratoria de originalidad correspondiente a su propuesta titulada **Detección de tornillos ausentes en ensamble de ECU mediante Python y OpenCV en Raspberry Pi.**

Con ello daremos inicio a los procesos de primera revisión y doble arbitraje ciego; en cuanto tengamos los resultados se los haremos saber.

Es grato saludarlos.

De: Alejandro Martínez <alejmlara@gmail.com>
Enviado: miércoles, 11 de noviembre de 2020 17:59
Para: Comite Revista UTL <comiterevista@utleon.edu.mx>
Cc: MIGUEL ANGEL OCHOA VILLEGAS <miguel.angel.ochoa@itnl.edu.mx>; RENE SANJUAN GALINDO <rene.sanjuan@itnl.edu.mx>; juan.antonio.rojas@itnl.edu.mx <juan.antonio.rojas@itnl.edu.mx>
Asunto: Re: Propuesta para colaboración

Buenas noches,

Adjunto declaratoria.

Saludos, y gracias de antemano.

> On Nov 11, 2020, at 2:49 PM, Comite Revista UTL <comiterevista@utleon.edu.mx> wrote:
>
> **Estimados Alejandro Martínez-Lara, Juan Antonio Rojas-Estrada, Rene Sanjuan-Galindo y Miguel ÁngelOchoa-Villegas:**
>
> A nombre del Comité Editorial, agradecemos la confianza de enviar su colaboración a nuestra revista, y les solicitamos remitan la declaratoria de originalidad firmada por los cuatro autores (en el orden en el que aparecerían en la publicación), en formato PDF, para poder comenzar con el arbitraje correspondiente; anexamos archivo.
>
> Asimismo, solicitamos sean tan amables de llenar individualmente el siguiente formulario con los datos de sus CV. La liga es: http://goo.gl/forms/auiMjfNUr6
>
> Favor de confirmar de recibido.
>
> Saludos cordiales y que tengan excelente día.
>
> <Outlook-f0yokgr5.png>
>
> ---
>
> **De:** Alejandro Martínez <alejmlara@gmail.com>
> **Enviado:** martes, 10 de noviembre de 2020 14:02
> **Para:** Comite Revista UTL <comiterevista@utleon.edu.mx>
> **Cc:** MIGUEL ANGEL OCHOA VILLEGAS <miguel.angel.ochoa@itnl.edu.mx>; RENE SANJUAN GALINDO <rene.sanjuan@itnl.edu.mx>
> **Asunto:** Propuesta para colaboración
>
> Buen día,
>
> De la manera más atenta hago llegar esta propuesta para colaboración. Se trata de un sistema de detección de tornillos ausentes en el ensamble de las tarjetas electrónicas de ECU mediante Python y OpenCV en Raspberry Pi.
>
> Quedo atento a sus comentarios.
>
> Saludos, y gracias de antemano.
>
> <Declaratoria de originalidad fin.docx>

Apéndice D

Artículo de Divulgación: Detección de Tornillos Ausentes en Ensamble de ECU Mediante Python y OpenCV en Raspberry Pi

Detección de tornillos ausentes en ensamble de ECU mediante Python y OpenCV en Raspberry Pi

Absent screw detection on ECU assembly with Python and OpenCV on Raspberry Pi

Martínez-Lara, Alejandro; Rojas-Estrada, Juan Antonio; Sanjuan-Galindo, Rene; Ochoa-Villegas, Miguel Ángel*
Tecnológico Nacional de México/ I.T. Nuevo León
alejmlara@gmail.com; juan.antonio.rojas@itnl.edu.mx;
rene.sanjuan@itnl.edu.mx
*autor de correspondencia: miguel.angel.ochoa@itnl.edu.mx

Resumen

El ensamble manual de las tarjetas electrónicas (PCB) para unidades de control electrónico (ECU) re-manufacturadas carece de algún control para el conteo de los tornillos instalados, lo cual ha ocasionado que los defectos no sean detectados antes de que el producto final sea enviado al cliente. Este proyecto propone un sistema de inspección óptica para el conteo automatizado del total de tornillos en las tarjetas después de su ensamble, mediante la implementación de un sistema de visión artificial desarrollado en Python, utilizando la librería OpenCV. La ejecución del código se realiza en un dispositivo Raspberry Pi. El sistema implementado logró un porcentaje de reconocimiento de 81% de tornillos instalados en las tarjetas.

Palabras clave: Ensamble manual, Inspección automatizada, Python, Sistema de visión

Abstract

The manual assembly of the printed circuit boards (PCB) for the re-manufactured Electronic Control Units (ECU) lacks a control to count the number of screws installed per unit. This has caused defects which are not detected before the product is shipped to the final customer. This project proposes an optical inspection system to automate the count of the total number of installed screws after the board assembly, by implementing a computer vision system developed on Python, using the OpenCV library. The code for this application will be executed by a Raspberry Pi device. An installed screw recognition rate of 81% was achieved with this system.

Keywords: Manual assembly, Automated inspection, Python, Vision system

Introducción

La visión artificial se ha utilizado en muchas aplicaciones. Un caso particular es la inspección y clasificación de características de objetos, especialmente para reemplazar la visión humana, ya que una máquina puede ser más confiable y consistente a lo largo de un día (en un ambiente de manufactura).

Actualmente, la mayoría de los sistemas de inspección óptica se basan en redes neuronales que no sólo requieren de una gran cantidad de imágenes para ser entrenadas, sino que además son de alto costo [1]. Estos sistemas requieren gran poder de procesamiento para alcanzar tiempos de ciclo aceptables, e incluso en aplicaciones que cuentan con computadoras sobrecargadas, los resultados se obtienen de manera lenta y el tiempo de desarrollo es largo.

Con mayor rapidez la tecnología se vuelve más accesible y en el caso del módulo Raspberry Pi, las comunidades de desarrolladores del mundo se apoyan para acelerar los tiempos de creación de nuevas aplicaciones, así como para explotar el máximo poder que tiene esta plataforma. Raspberry Pi es una plataforma que presenta muchas ventajas, entre ellas la seguridad (al estar construido sobre un sistema operativo basado en UNIX), el bajo costo y el poder moderado que presenta.

La re-manufactura de tarjetas electrónicas para Módulos de Control Electrónicos (ECM) es un proceso manual. Inicialmente son des-ensambladas y reparadas a lo largo de una serie de operaciones. Sin embargo, uno de los procesos más críticos para la operación es el ensamble de las tarjetas electrónicas reparadas, ya que es la última oportunidad para poder detectar defectos en las tarjetas. Las fallas de ensamble y los defectos tienen un impacto en la calidad del producto, así como en la reputación de la marca al generarse reclamos de garantía. Se ha encontrado que existen módulos que el cliente recibe pero que fueron ensambladas incorrectamente, específicamente con tornillos faltantes. Estas unidades, al ser expuestas al calor de motores y vibraciones, presentan fallas funcionales debido a que la integridad y estabilidad de la tarjeta electrónica se encuentra comprometida.

El negocio de re-manufactura de Unidades de Control Electrónico (ECU) tiene un volumen de producción considerablemente bajo en comparación con la manufactura de producto nuevo. Es por esto, que el tiempo de ejecución del código en una Raspberry Pi es suficiente para la aplicación.

El ensamble de estos módulos es manual y no se cuenta con un sistema de control de ensamble para monitorear diversos pasos críticos como la instalación del número correcto de tornillos o la validación de que los tornillos instalados sean del número de parte correcto. Es por esto por lo que es necesaria la implementación de un sistema de inspección con visión artificial para el conteo automatizado de tornillos instalados en una unidad una vez realizado el ensamble de la misma. Las ventajas esperadas son que el sistema *autónomo* sea de bajo costo, con mantenimiento mínimo y que también sea escalable y adaptable. Al revisar la literatura se encontraron algunas opciones tecnológicas.

Galan [2] propone un sistema de visión para identificar y clasificar defectos en superficies de carcasas metálicas. Sin embargo, el hardware utilizado para esta aplicación es caro y elaborado. Esto es debido a que la aplicación busca identificar características de tamaños reducidos en tiempos bajos de procesamiento, los cuales no son requerimientos críticos para el propósito del presente trabajo. El autor reportó el uso de OpenCV, la cual es una librería muy popular y completa para desarrollo de aplicaciones de visión artificial, pero finalmente fue desarrollada en Ubuntu Linux. El sistema binariza las imágenes, es decir, las convierte a escala de grises y aplica filtros y algoritmos para convertir las imágenes a sólo dos tonos: blanco o negro. Los resultados fueron aceptables, con tiempos de procesamiento bajos al utilizar un GPU dedicado en lugar de un CPU y así mantener un error menor al 12%. Jaffery *et al*. [3], desarrollaron un sistema para reemplazar las inspecciones realizadas por un operador en vías de ferrocarril para detectar tuercas y tornillos ausentes. En este caso, se utilizó una técnica de reconocimiento de patrones y la DWT (Discrete Wavelet Transform). El sistema de procesamiento no fue significativamente poderoso, sin embargo, sí fue mucho más que un sistema de Raspberry Pi. Un enfoque más robusto es propuesto por Johan y Prabuwono [4], el cual utiliza una red neuronal artificial para reconocer tornillos y tuercas. Este sistema utilizó MATLAB para procesar las imágenes. Sin embargo, la inspección en línea requiere que los tiempos de cómputo sean rápidos, ya que los objetos a inspeccionar son transportados mediante una banda. Este sistema no sólo inspeccionaba tornillos y tuercas, sino que además los clasificaba. Los autores lograron una tasa de clasificación correcta del 92%. En el trabajo de Ruvo *et. al.*, [5] desarrollaron una inspección en tiempo real para detección de tornillos con cabeza hexagonal, utilizando un sistema FPGA para realizar el procesamiento de las imágenes, además de un algoritmo predictivo.

El filtro Canny es un algoritmo desarrollado por John F. Canny en 1986, el cual tiene como objetivo obtener la mejor aproximación de la detección de bordes en una imagen. Este algoritmo es constantemente retomado para mejorar y

optimizar. OpenCV incluso tiene una función predeterminada para la detección de bordes con el método de Canny [6]. La erosión es una operación morfológica que se aplica a imágenes para engrosar los bordes detectados y utiliza un elemento estructural como máscara [7]. Sin embargo, este primer prototipo que se hizo no era capaz de detectar automáticamente las ubicaciones de los tornillos en la imagen, ya que no estaba basado en la detección de patrones, ni se contaba con un set de entrenamiento. Viendo esta oportunidad, se tomó la decisión de realizar la migración del código al dispositivo Raspberry Pi en el lenguaje Python debido a la mejora en el tiempo de procesamiento en comparación con MATLAB [8] y hacer uso de las capacidades de la librería OpenCV y de la técnica de *template matching*.

Existen ya implementaciones [9] de *template matching* con múltiples patrones de referencia, sin embargo, en general se corren en equipos con grandes capacidades computacionales. En este caso, siendo una aplicación sencilla, se pueden aprovechar las especificaciones de la Raspberry Pi para la implementación de sistemas de bajo costo y que cambian la perspectiva que se tiene actualmente de este tipo de dispositivos, generalmente asociados a proyectos estudiantiles o de aficionados a la electrónica o informática. También se aprovecha el poder del lenguaje de programación Python. El número de usuarios de este lenguaje de programación ha incrementado en los últimos años debido a su flexibilidad y poder en el desarrollo de librerías específicas que son *open source* [10]. De igual manera, OpenCV ha sido aceptado globalmente como un componente esencial en aplicaciones de desarrollo con visión artificial, no sólo para Python, sino también para otros lenguajes de programación.

La técnica de *template matching* puede hacer operaciones más complejas, adicionales a la detección de objetos en imágenes; también se pueden implementar sistemas de clasificación e incluso generar conjuntos de entrenamiento dinámicos.

Objetivo

Desarrollar un sistema de visión artificial que sea capaz de diferenciar el estado de las tarjetas electrónicas de un ECU de acuerdo al ensamble, sea bueno o malo, con una precisión del 80% o mayor. Este sistema será basado en Raspberry Pi con un módulo de cámara con resolución de 8 MP.

Materiales y métodos

Sistema para detección de tornillos y software

El sistema propuesto para la detección de tornillos está basado en una Raspberry Pi 3 Modelo B con 1 GB de RAM y un procesador Quad Core 1.2GHz Broadcom BCM2837 64bit. El primer prototipo del software se desarrolló en MATLAB. La implementación del código en la Raspberry Pi se hará usando el software *open-source* Python con la librería OpenCV que permitirá contar con un prototipo de bajo costo, respecto a las opciones disponibles en el mercado.

Captura de imágenes

La captura de las imágenes de las PCB se realizó con una Raspberry Pi Camera v2.1, la cual tiene un sensor de 8 megapíxeles con resolución verdadera de 3280x2464 y distancia focal de 3.04 mm.

Cantidad de muestras

Se tomó una muestra de 10 unidades del proceso con los tornillos instalados. Se capturaron las fotografías con la cámara para ser procesadas por el sistema de Python y las funciones de la librería OpenCV.

Implementación de *template matching*

La técnica de *template matching* se utiliza para buscar y encontrar incidencias de una imagen patrón o de entrenamiento dentro de otra imagen. En el caso de la implementación de *template matching* en OpenCV, se utiliza un algoritmo de convolución en 2D y compara la imagen de referencia proporcionada como argumento de entrada de la función contra la imagen en la que se desea buscar y encontrar la incidencia de la misma. La función *matchTemplate()* en OpenCV tiene implementados varios métodos de comparación de imágenes. El método utilizado en esta aplicación es el llamado Método de Coincidencia de Coeficientes de Correlación Normalizados (TM.CCOEFF.NORMED, como se llama en OpenCV).

La representación matemática del método CCOEFF.NORMED está definida por las ecuaciones 1, 2 y 3.

$$R_{\text{ccoeff\_normed}} = \frac{\sum_{x',y'} T'(x',y') \cdot I'(x+x', y+y')}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x', y+y')^2}} \quad (1)$$

$$T'(x',y') = T(x',y') - \frac{\sum_{x'',y''} T(x'',y'')}{w \cdot h} \quad (2)$$

$$I'(x+x', y+y') = I(x+x', y+y') - \frac{\sum_{x'',y''} I(x'',y'')}{w \cdot h} \quad (3)$$

En las ecuaciones anteriores, representa la imagen de entrada de la que se desea encontrar la imagen patrón . es la imagen resultado. El método de correlación normalizado busca hacer coincidir una imagen patrón relativa a su media con una imagen relativa a su media. Esto da como resultado un valor positivo si se encuentra una coincidencia o un valor negativo si no se encuentra una coincidencia y un valor de cero si no existe correlación. Por lo tanto, al ser un método normalizado, los resultados posibles son 1, -1 y 0 respectivamente [11].

En las ecuaciones anteriores, I representa la imagen de entrada de la que se desea encontrar la imagen patrón T. $R_{\text{coeff\_normed}}$ es la imagen resultado. El método de correlación normalizado busca hacer coincidir una imagen patrón relativa a su media con una imagen relativa a su media. Esto da como resultado un valor positivo si se encuentra una coincidencia o un valor negativo si no se encuentra una coincidencia y un valor de cero si no existe correlación. Por lo tanto, al ser un método normalizado, los resultados posibles son 1, -1 y 0 respectivamente [11].

A la función *matchTemplate()* se le proporciona además un valor de umbral para diferenciar la coincidencia. En este caso se determinó un umbral de 0.75 debido a que es un valor suficientemente estricto para demostrar buen reconocimiento sin rechazar completamente los resultados falsos negativos (unidades buenas que son calificadas como malas).

Resultados

Generación de set de entrenamiento

El paso inicial fue la generación de un set de entrenamiento para el programa en Python, el lenguaje de programación al que posteriormente se migró el código. Esto consistió en la captura de fotografías tomadas a 4 unidades de ECU de referencia ensambladas correctamente, extrayendo un conjunto de imágenes de cada uno de los 11 tornillos utilizados para el ensamble, formando 44 imágenes de referencia. En la Figura 1 se observa una muestra de los tornillos del set de entrenamiento. Este conjunto de imágenes es el argumento de entrada para la función de código que se encarga de buscar en la imagen de la unidad de prueba la ubicación de los tornillos y que además los identifica visualmente sobre la misma. Se decidió utilizar este conjunto de entrenamiento para mitigar variaciones que ocurran ya sea en la iluminación o en el ángulo en el que se toma la fotografía.

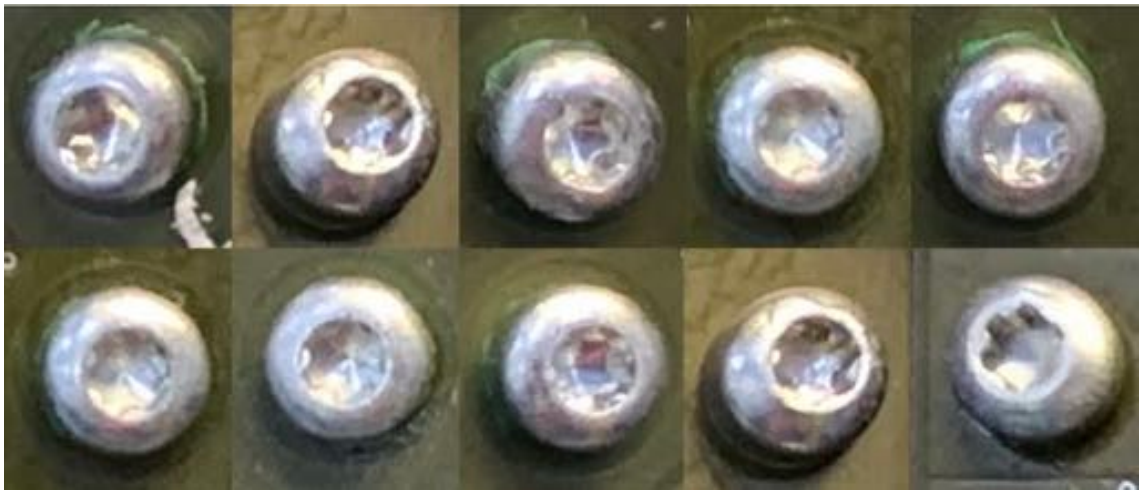


Figura 1. Muestra de 10 tornillos utilizados como imágenes patrón.

A partir del conjunto de entrenamiento de tornillos generado para la aplicación, se realiza una iteración por cada elemento del arreglo que contiene dicho conjunto. Esto es debido a que la función *matchTemplate()* de OpenCV sólo acepta una imagen patrón por ejecución [12]. Esta es la manera en la que la función se encuentra implementada en OpenCV. Para utilizar múltiples imágenes de entrenamiento se tienen que recurrir a técnicas de programación como la implementación de bucles iterativos. Como segunda parte del código del programa, se itera en otro arreglo que contiene el conjunto de referencia de las perforaciones sin tornillo. De esta manera, se optimiza la detección de los tornillos en la imagen a inspeccionar.

Cada vez que el programa detecta la coincidencia de un tornillo en la fotografía se dibuja un recuadro verde alrededor de la coordenada en la que se encontró.

De igual manera, cuando en la segunda parte del programa se encuentra una perforación sin tornillo, se dibuja un recuadro rojo alrededor de la coordenada en la se encontró. Adicionalmente, se presenta un contador y un mensaje para el usuario, en el que se indica si el resultado es de Pasa o Falla.

Se logró inspeccionar visualmente el ensamble de tarjetas electrónicas para detectar tornillos faltantes y proporcionar un control para el proceso de ensamble de las mismas. Inicialmente, se buscaba lograr detectar los tornillos ausentes; sin embargo, tras experimentación y variaciones en el proceso (iluminación y posición) se llegó a la conclusión de que es más efectivo inspeccionar buscando los tornillos presentes. En la Figura 2 se observa una de las imágenes de prueba de la tarjeta electrónica ensamblada.

Figura 2. Ejemplo de una imagen de prueba.

Para comprobar el porcentaje de reconocimiento de la aplicación se utilizó la técnica de validación cruzada conocida como *leave one out*. Las pruebas se realizaron con un conjunto de 10 imágenes de prueba de tarjetas electrónicas (Unidades 1 a 10). En este tipo de validación, se elimina un dato de entrenamiento por iteración (10 iteraciones por prueba). En la Figura 3 se muestra un diagrama de flujo del algoritmo. Esto se realizó con dos subconjuntos de entrenamiento: uno de 10 tornillos y uno de 20 tornillos. El propósito de estas pruebas fue identificar si existe una diferencia significativa en el porcentaje de reconocimiento del sistema cuando se incrementa la cantidad de imágenes de entrenamiento. También se midieron los tiempos de ejecución, ya que esta técnica requiere un elevado número de iteraciones del código.

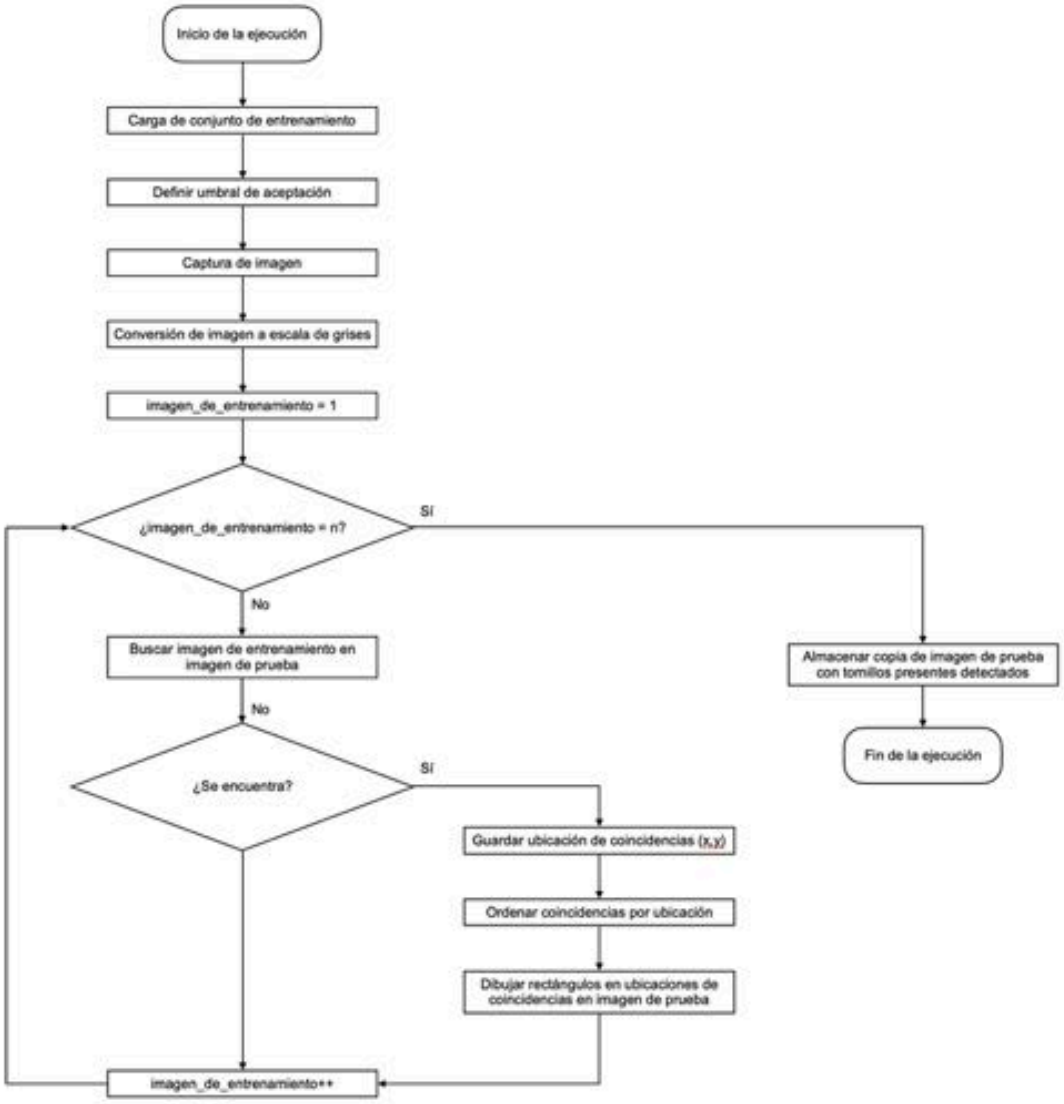
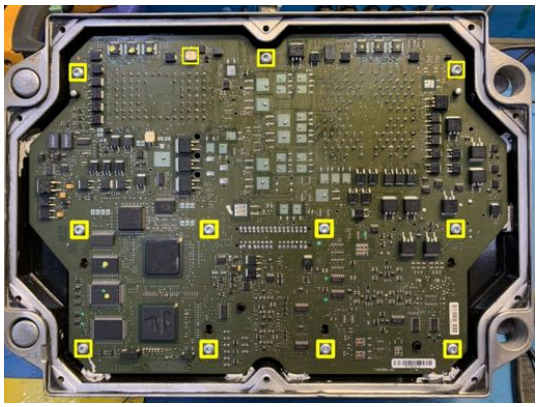


Figura 3. Diagrama de flujo del algoritmo programado en Python.

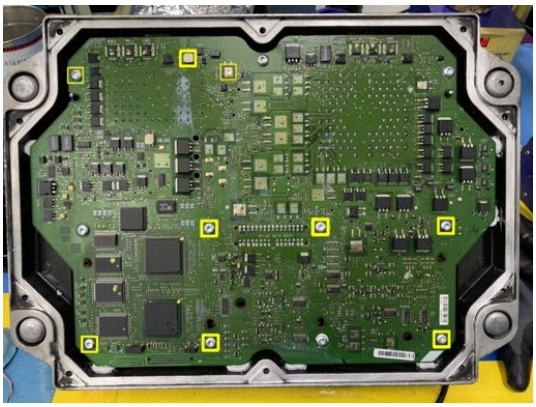
	Porcentaje de detección										
Tornillo eliminado	**Unidad 1**	**Unidad 2**	**Unidad 3**	**Unidad 4**	**Unidad 5**	**Unidad 6**	**Unidad 7**	**Unidad 8**	**Unidad 9**	**Unidad 10**	**% de detección por tornillo eliminado**
1	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
2	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
3	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
4	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	63.64%	86.36%
5	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
6	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
7	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
8	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
9	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
10	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
% de detección por unidad	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**90.91%**	**72.73%**	**90.91%**	**72.73%**	**72.73%**	**71.82%**	**87.18%**

Tabla 1. Resultados de detección con un conjunto de entrenamiento de 10 tornillos.

Los resultados obtenidos de la primera prueba se muestran en la Tabla 1. En ella se utilizó el conjunto de entrenamiento de 10 tornillos con las 10 imágenes de prueba de tarjetas electrónicas. Se demuestra que en el mejor de los casos se tiene un porcentaje de detección de tornillos presentes en el ensamble de 100%, mientras que el peor caso (la unidad 10 eliminando en tornillo 4 del conjunto de entrenamiento) se obtiene un porcentaje de detección de los tornillos de 63.64%. El promedio de detección del sistema con este set de entrenamiento es de 87.18%. En las Figuras 4a y 4b se observan el mejor y peor caso, respectivamente, de detección con esta prueba.



a) Mejor caso de la primera prueba con el conjunto de 10 tornillos.



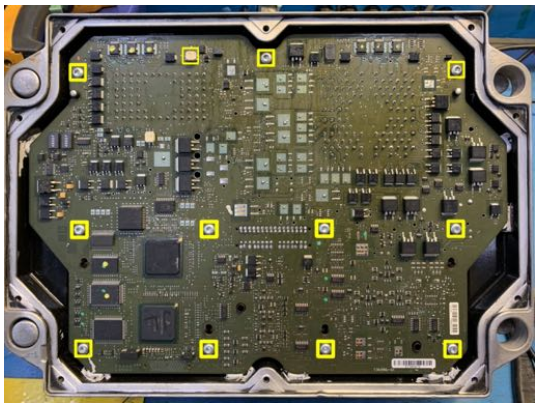
b) Peor caso de la primera prueba con el conjunto de 10 tornillos.

Figura 4. Resultados de la primera prueba con el conjunto de 10 tornillos.

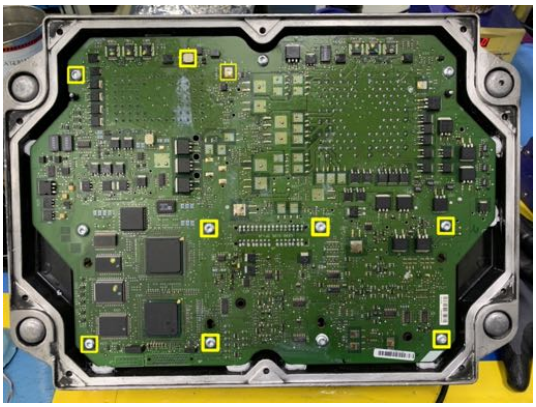
	Porcentaje de detección										
Tornillo eliminado	Unidad 1	Unidad 2	Unidad 3	Unidad 4	Unidad 5	Unidad 6	Unidad 7	Unidad 8	Unidad 9	Unidad 10	% de detección por tornillo eliminado
1	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
2	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
3	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
4	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	63.64%	86.36%
5	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
6	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
7	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
8	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
9	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
10	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
11	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
12	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
13	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
14	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
15	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
16	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
17	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
18	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
19	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
20	100.00%	100.00%	100.00%	100.00%	90.91%	72.73%	90.91%	72.73%	72.73%	72.73%	87.27%
% de detección por unidad	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**90.91%**	**72.73%**	**90.91%**	**72.73%**	**72.73%**	**72.27%**	**87.23%**

Tabla 2. Resultados de detección con un conjunto de entrenamiento de 20 tornillos.

En la segunda demostración se tomó el conjunto de entrenamiento de 20 tornillos para las mismas 10 imágenes de prueba. En la Tabla 2 se muestran los resultados de estas iteraciones; mostrando de nuevo un mejor caso de 100% con las primeras 4 unidades, mientras que el peor caso se vuelve a presentar en la imagen 10 con el tornillo 4, mostrando porcentaje de detección de tornillos presentes de 63.64%. El porcentaje de detección del sistema con este set de entrenamiento es de 87.23%. En las Figuras 5a y 5b se observan el mejor y peor caso, respectivamente, de detección con esta prueba.



a) Mejor caso de la primera prueba con el conjunto de 20 tornillos.



b) Peor caso de la primera prueba con el conjunto de 20 tornillos.

Figura 5. Resultados de la primera prueba con el conjunto de 20 tornillos.

En las Tablas 3 y 4 se presentan los resultados en tiempo de ejecución de las iteraciones de las pruebas con ambos tamaños de subconjuntos de entrenamiento. Se observa, en promedio, un incremento en el tiempo de ejecución de 67.91 segundos para obtener un 0.05% de mejora en el porcentaje de detección de tornillos. Es por esto que se decide utilizar el subconjunto de 10 tornillos de entrenamiento en el prototipo final, ya que proporciona un tiempo de ejecución menor con una diferencia no significativa en el porcentaje de detección de nuestra aplicación.

	Tiempo de ejecución (s)										
Tornillo eliminado	Unidad 1	Unidad 2	Unidad 3	Unidad 4	Unidad 5	Unidad 6	Unidad 7	Unidad 8	Unidad 9	Unidad 10	Tiempo promedio de ejecución
1	60.97	61.13	60.83	59.77	60.52	59.04	58.73	59.54	59.84	58.98	59.94
2	60.45	60.61	60.62	59.88	60.21	59.38	59.49	59.74	59.80	57.04	59.72
3	68.86	61.18	61.22	59.86	59.51	58.92	61.27	63.38	60.15	56.87	61.12
4	61.77	62.06	60.10	58.98	59.33	58.98	60.21	59.85	60.19	61.04	60.25
5	60.56	60.90	63.98	58.83	59.85	59.13	59.46	62.05	60.23	57.33	60.23
6	60.65	62.55	61.67	59.60	61.44	60.56	60.48	60.75	59.91	56.78	60.44
7	60.82	61.21	59.78	59.90	59.57	63.86	59.56	61.46	59.23	56.80	60.22
8	63.62	60.87	59.74	61.73	63.77	65.59	60.84	60.69	58.87	57.07	61.28
9	60.85	67.29	59.05	60.03	59.65	60.64	60.44	59.79	59.57	58.71	60.60
10	63.00	60.83	59.77	62.54	61.67	59.85	59.33	61.60	57.26	60.98	60.68
Tiempo promedio de ejecución	**62.15**	**61.86**	**60.68**	**60.11**	**60.55**	**60.60**	**59.98**	**60.89**	**59.50**	**58.16**	**60.45**

Tabla 3. Tiempos de ejecución de entrenamiento y prueba por unidad con subconjunto de 10 tornillos.

	Porcentaje de detección										
Tornillo eliminado	Unidad 1	Unidad 2	Unidad 3	Unidad 4	Unidad 5	Unidad 6	Unidad 7	Unidad 8	Unidad 9	Unidad 10	% de detección por tornillo eliminado
1	128.04	145.39	130.85	125.10	126.19	128.79	126.53	127.68	126.79	126.94	129.23
2	123.14	145.07	131.79	126.00	129.30	126.82	131.36	127.91	129.77	128.23	129.94
3	126.97	126.69	130.92	124.90	129.62	128.56	130.32	127.06	127.69	126.94	127.97
4	130.31	127.61	135.34	123.96	126.01	127.45	127.78	130.49	128.59	128.28	128.58
5	141.11	129.11	131.09	123.89	125.95	127.26	126.37	127.60	128.68	126.61	128.77
6	143.01	126.85	130.66	123.54	126.43	127.67	125.67	128.47	128.13	126.38	128.68
7	138.24	127.75	144.44	124.38	126.40	128.40	127.21	126.93	128.73	126.74	129.92
8	133.72	129.30	132.23	123.54	126.08	126.04	130.17	129.00	128.81	128.46	128.73
9	132.99	130.62	134.08	126.03	128.36	129.58	127.26	128.21	127.69	130.78	129.56
10	133.59	133.67	125.17	130.33	127.82	128.49	127.86	126.96	127.02	129.45	129.04
11	129.46	138.40	126.79	125.74	126.35	129.20	127.21	130.85	129.51	128.31	129.18
12	128.49	132.46	126.53	127.22	126.33	127.15	131.78	127.20	128.08	129.96	128.52
13	128.91	133.96	127.30	128.03	128.46	126.45	132.95	126.72	127.92	127.62	128.83
14	129.35	133.53	134.32	126.73	127.39	126.54	128.49	126.62	128.73	135.30	129.70
15	133.49	131.58	127.44	130.48	125.74	129.55	127.18	129.44	128.78	134.08	129.78
16	130.32	131.20	127.46	124.04	127.53	126.67	136.26	126.61	127.20	126.70	128.40
17	129.45	131.34	126.08	135.27	127.52	125.83	127.57	127.93	127.30	126.48	128.48
18	130.15	130.85	127.64	129.13	127.44	127.00	12.93	127.12	126.48	132.10	117.08
19	129.05	133.23	128.61	125.35	128.36	129.98	129.02	127.45	127.28	125.19	128.35
20	135.01	131.58	131.29	126.31	125.43	126.89	127.83	127.71	127.18	125.32	128.45
% de detección por unidad	**131.74**	**132.51**	**130.50**	**126.50**	**127.14**	**127.72**	**123.09**	**127.90**	**128.02**	**128.49**	**128.36**

Tabla 4. Tiempos de ejecución de entrenamiento y prueba por unidad con subconjunto de 20 tornillos.

La técnica de *template matching* para detectar varios objetos en un sólo procedimiento incrementa las capacidades de la aplicación, así como los tiempos de procesamiento al realizar el acondicionamiento de la imagen original solamente una vez. La creación de conjuntos de entrenamiento para la aplicación la hace que esta sea más robusta y que la detección de los tornillos presentes sea más confiable y precisa.

Un sistema convencional de redes neuronales para la detección y clasificación de características de objetos [13] tiene un porcentaje de reconocimiento del 98.4% utilizando un procesador de cuatro núcleos a 2.80 GHz y 16 GB de RAM. Otra aplicación demuestra un porcentaje de reconocimiento de 80% [14] utilizando un CPU de doble núcleo a una velocidad de 2.27 GHz y 4 GB de RAM, logrando además, tiempos de ejecución de 18 ms con imágenes de prueba de resolución de 2048 x 2048. Sin embargo, nuestro sistema, en condiciones controladas puede lograr un 100% y en condiciones no controladas un promedio de 87.23%. El porcentaje de reconocimiento de la aplicación con Raspberry Pi es bueno en comparación con sistemas implementados en hardware más robusto, considerando que las características de este dispositivo son de bajo costo.

Conclusiones

Se propone un sistema para la inspección automatizada de tornillos presentes en el ensamble de tarjetas electrónicas. La implementación de este sistema en Raspberry Pi le da un giro innovador al dispositivo de bajo costo al utilizarlo en un ambiente industrial. Se aprovecha la flexibilidad de Python en conjunto con la librería OpenCV para el desarrollo del software de inspección.

Comparando la implementación de este sistema con otras aplicaciones similares, se encuentra que el porcentaje de detección logrado de 87% se encuentra aceptable, ya que en aplicaciones que hacen uso de hardware especializado y robusto se logran porcentajes menores y mayores.

El ciclo de inspección se logró en un tiempo de ejecución de 60 segundos. Este tiempo de ejecución es suficiente para la aplicación mostrada; sin embargo, este puede reducirse al migrar el sistema a un procesador con mayor velocidad y mayor capacidad de memoria RAM, como podría ser una Raspberry Pi 4 o un ordenador.

Algunas propuestas de mejora para esta implementación son el preprocesamiento de la imagen con la finalidad de mitigar la iluminación en el área en la que se colocan las unidades, así como la variación de la posición de la cámara. Esto ayudaría a controlar de manera más robusta la detección, no sólo de los tornillos ausentes, sino también de los presentes. De igual manera, el posicionamiento fijo de la cámara ayudaría a que las imágenes sean siempre tomadas en el mismo ángulo y distancia, incrementando aún más la precisión con la que el sistema puede detectar las características buscadas.

Con la implementación de este tipo de aplicaciones en dispositivos Raspberry Pi, se busca promover su desarrollo en sistemas industriales relacionados con re-manufactura de tarjetas electrónicas.

Referencias bibliográficas

1. 7TH International Engineering, Sciences and Technology Conference (IESTEC), (7th : 2019 : Panama, Panama). 3D Mapping System Estimated from 2D for Low-Cost Devices. Panama, Panama, IEEE, 2019. pp. 665-670.
2. GALAN, Ulises, ORTA, Pedro, KURFESS, Thomas y AHUETT-GARZA, Horacio. Surface defect identification and measurement for metal castings by vision system. *Manufacturing Letters*. [en línea] 15, part A,

Enero 2018. [fecha de consulta: 21 de febrero de 2020]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S2213846317300743. ISSN: 2213-8463.

3. 2017 International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT) (2017 : Aligarh, India). Detection of missing nuts & bolts on rail fishplate. Aligarh, India. IEEE, 2017. pp. 36-40.

4. 2011 International Conference on Pattern Analysis and Intelligence Robotics (2011 : Putrajaya, Malasia). Recognition of bolt and nut using artificial neural network. IEEE, Putrajaya, Malasia. 2011. pp. 165-170.

5. Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05), (7th : 2005 : Palermo, Italy). A FPGA-based architecture for automatic hexagonal bolts detection in railway maintenance. Palermo, Italy, IEEE, 2005 pp. 219-224.

6. 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), (13th : 2017 : Yangzhou, China). Canny edge detection based on Open CV. Yangzhou, China, IEEE, 2017. pp. 53-56.

7. 15th IEEE International Conference on Image Processing. (15th : 2008 : San Diego, CA). Rapid image binarization with morphological operators. San Diego, CA. IEEE. *2008*. pp. 1017-1020.

8. *Elektro*. 2012. (2012 : Rajeck, Teplice) The comparison of CPU time consumption for image processing algorithm in Matlab and OpenCV. *2012* Rajeck Teplice, IEEE, 2012. pp. 75-78.

9. THOMAS, Laurent S. V. y GEHRIG, Jochen. Multi-template matching: a versatile tool for object-localization in microscopy images. *BMC Bioinformatics* [en línea]. Febrero 2020. Vol. 21, no. 1. [fecha de consulta: 6 mayo 2020] Disponible en: https://doi.org/10.1186/s12859-020-3363-7.

10. Python. Python Developers Survey 2018 Results. *JetBrains.* Sin año [en línea]. Disponible en: https://bit.ly/3kMIZrc [fecha de consulta: 1 agosto 2020].

11. KAEHLER, Adrian y BRADSKI, Gary R. Learning OpenCV 3: computer vision in C with the OpenCV library. 3ª ed. EEUU. O'Reilly Media, 2017. 990 p. ISBN-13: 978-1491937990.

12. OpenCV Open Source Computer Vision. Template Matching. Sin año. [En línea]. Disponible en: https://bit.ly/3kN9zQX [fecha de consulta: 2 marzo 2020].

13. SONG, Limei, LI, Xinyao, YANG, Yangang, ZHU, Xinjun, GUO, Qinghua y YANG, Huaidong. Detection of micro-defects on metal screw surfaces based on deep convolutional neural networks. *Sensors* [en línea]. Vol. 18, no. 11, p. 3709. 31 Octubre 2018. [fecha de consulta: marzo 2020] DOI 10.3390/s18113709. Disponible en: http://dx.doi.org/10.3390/

s18113709. ISSN: 1424-8220.

14. ZHONG, Qiusheng, CHEN, Zhong, ZHANG, Xianmin y HU, Guanghua. Feature-based object location of IC pins by using fast run length encoding BLOB analysis. *IEEE Transactions on Components, Packaging and Manufacturing Technology* [en línea]. Vol. 4, No. 11. Nov. 2014. [fecha de consulta: marzo 2020]. Disponible en: https://ieeexplore.ieee.org/document/6894154. ISSN: 2156-3985.

Bibliografía

[1] V. Rodríguez, C. Pinzón y J. C. Rangel. "3D Mapping System Estimated from 2D for Low-Cost Devices". En: *2019 7th International Engineering, Sciences and Technology Conference*. 2019, págs. 665-670. DOI: 10.1109/IESTEC46403.2019.00124.

[2] H. Zhao, J. Cheng y J. Jin. "NI vision based automatic optical inspection (AOI) for surface mount devices: Devices and method". En: *2009 International Conference on Applied Superconductivity and Electromagnetic Devices*. 2009, págs. 356-360. DOI: 10.1109/ASEMD.2009.5306622.

[3] R. Pramudita y F. I. Hariadi. "Development Of Techniques to Determine Object Shifts for PCB Board Assembly Automatic Optical Inspection (AOI)". En: *2018 International Symposium on Electronics and Smart Devices*. 2018, págs. 1-4. DOI: 10.1109/ISESD.2018.8605458.

[4] Y. Lin, Y. Chiang y H. Hsu. "Capacitor Detection in PCB Using YOLO Algorithm". En: *2018 International Conference on System Science and Engineering*. 2018, págs. 1-4. DOI: 10.1109/ICSSE.2018.8520170.

[5] J. Redmon y col. "You Only Look Once: Unified, Real-Time Object Detection". En: *2016 IEEE Conference on Computer Vision and Pattern Recognition*. 2016, págs. 779-788. DOI: 10.1109/CVPR.2016.91.

[6] F. Raihan y W. Ce. "PCB defect detection USING OPENCV with image subtraction method". En: *2017 International Conference on Information Management and Technology*. 2017, págs. 204-209. DOI: 10.1109/ICIMTech.2017.8273538.

[7] U. Galan y col. "Surface defect identification and measurement for metal castings by vision system". En: *Manufacturing Letters* 15 2018, págs. 5-8. ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2017.12.001.

[8] Z. A. Jaffery, D. Sharma y N. Ahmad. "Detection of missing nuts bolts on rail fishplate". En: *2017 International Conference on Multimedia, Signal Processing and Communication Technologies*. 2017, págs. 36-40. DOI: 10.1109/MSPCT.2017.8363969.

[9] T. M. Johan y A. S. Prabuwono. "Recognition of bolt and nut using artificial neural network". En: *2011 International Conference on Pattern Analysis and Intelligence Robotics*. Vol. 1. 2011, págs. 165-170. DOI: 10.1109/ICPAIR.2011.5976889.

[10] G. De Ruvo y col. "A FPGA-based architecture for automatic hexagonal bolts detection in railway maintenance". En: *Seventh International Workshop on Computer Architecture for Machine Perception*. 2005, págs. 219-224. DOI: 10.1109/CAMP.2005.4.

[11] L. Thomas y J. Gehrig. "Multi-template matching: a versatile tool for object-localization in microscopy images". En: *BMC Bioinformatics* 21.1 feb. de 2020, pág. 44. ISSN: 1471-2105. DOI: 10.1186/s12859-020-3363-7.

[12] A. Nagpal y G. Gabrani. "Python for Data Analytics, Scientific and Technical Applications". En: *2019 Amity International Conference on Artificial Intelligence*. 2019, págs. 140-145. DOI: 10.1109/AICAI.2019.8701341.

[13] M. Janóczki y col. "Automatic Optical Inspection of Soldering". En: *Materials Science*. Ed. por Yitzhak Mastai. Rijeka: IntechOpen, 2013. Cap. 16. DOI: 10.5772/51699.

[14] R. Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345, 9781848829343.

[15] D. Forsyth y J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002. ISBN: 0130851981.

[16] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. USA: Henry Holt y Co., Inc., 1982. ISBN: 0716715678.

[17] E. Cooksey y W. D. Withers. "Rapid image binarization with morphological operators". En: *2008 15th IEEE International Conference on Image Processing*. Oct. de 2008, págs. 1017-1020. DOI: 10.1109/ICIP.2008.4711930.

[18] P. Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. 1st. Springer Publishing Company, Incorporated, 2013. ISBN: 3642201431.

[19] Y. Jia y col. "Research on the decomposition and fusion method for the infrared and visible images based on the guided image filtering and Gaussian filter". En: *2017 3rd IEEE International Conference on Computer and Communications*. 2017, págs. 1797-1802. DOI: 10.1109/CompComm.2017.8322849.

[20] C. Ma y col. "An improved Sobel algorithm based on median filter". En: *2010 2nd International Conference on Mechanical and Electronics Engineering*. Vol. 1. 2010, págs. V1-88-V1-92. DOI: 10.1109/ICMEE.2010.5558590.

[21] J. Canny. "A Computational Approach to Edge Detection". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 1986, págs. 679-698. DOI: 10.1109/TPAMI.1986.4767851.

[22] V. Bharath y N. S. Rani. "A font style classification system for English OCR". En: *2017 International Conference on Intelligent Computing and Control*. 2017, págs. 1-5. DOI: 10.1109/I2C2.2017.8321962.

[23] X. Peng y col. "Automated image quality assessment for camera-captured OCR". En: *2011 18th IEEE International Conference on Image Processing*. 2011, págs. 2621-2624. DOI: 10.1109/ICIP.2011.6116204.

[24] T. Jia, N. Sun y M. Cao. "Moving object detection based on blob analysis". En: *2008 IEEE International Conference on Automation and Logistics*. 2008, págs. 322-325. DOI: 10.1109/ICAL.2008.4636168.

[25] *About - OpenCV*. 2020. En línea. (Fecha de Consulta: 9 de Junio 2020). URL: https://opencv.org/about/.

[26] A. Kaehler y G. Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2016. ISBN: 9781491937969. URL: https://books.google.com.mx/books?id=LPm3DQAAQBAJ.

[27] M. Marengoni y D. Stringhini. "High Level Computer Vision Using OpenCV". En: *2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials*. 2011, págs. 11-24. DOI: 10.1109/SIBGRAPI-T.2011.11.

[28] J.P. Lewis. "Fast Normalized Cross-Correlation". En: *Industrial Light & Magic* 1995.

[29] *OpenCV Open Source Computer Vision. Template Matching*. Sin año. En línea. (Fecha de Consulta: 3 de Marzo 2020). URL: https://docs.opencv.org/master/d4/dc6/tutorial_py_template_matching.html.

[30] L. Song y col. "Detection of Micro-Defects on Metal Screw Surfaces Based on Deep Convolutional Neural Networks". En: *Sensors* 18.11 2018, pág. 3709. DOI: 10.3390/s18113709.

[31] Q. Zhong y col. "Feature-Based Object Location of IC Pins by Using Fast Run Length Encoding BLOB Analysis". En: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 4.11 2014, págs. 1887-1898. DOI: 10.1109/TCPMT.2014.2350015.