Optical Resistor Identifier Scanner using Machine Learning Techniques

Xuanrui Zeng   
*Department of Eletrical and Computer Engineering*  
*University of Alberta*  
Edmonton, Canada  
xuanrui3@ualberta.ca  
  
Dongchao Feng  
*Department of Eletrical and Computer Engineering*  
*University of Alberta*  
Edmonton, Canada  
dongchao@ualberta.ca  
  
Jiaxuan Kan  
*Department of Eletrical and Computer Engineering*  
*University of Alberta*  
Edmonton, Canada  
jkan@ualberta.ca

*Abstract*—An Optical Resistor Identifier Scanner (ORIS) implementation is presented in this paper. This camera-based scanner was developed to reliably identify axial leaded resistors that use the standard color code. A prototype of the ORIS based on machine learning techniques was designed, developed, and tested, and its performance had been verified under a controlled environment for a limited set of resistors. The prototype developed was mostly working and overall effective at achieving the given objectives, with an excellent accuracy in resistor identification and a reasonable performance on an edge device. Nevertheless, the prototype exhibited issues related to its high cost and high power consumption, and its identification accuracy and performance also has considerable room for improvements. Thus, the prototype presented is considered developmental and future work is needed to transform it into a product.

Keywords—resistor, resistor color code, resistor scanner, resistor identifier, machine learning, object detection.

# Introduction

Axial leaded resistors are commonly used in electronic projects for prototyping. These resistors were packaged with visible markings to identify its parameters (resistance, tolerance) when sold to the users, however when working with them, it is often required to separate them from their packaging, and hence the user can lose track of the parameter of a certain resistor over time. To resolve this issue, resistor color code convention was realized onto every axial leaded resistor [1], such that the user can extract the resistor’s parameters by interpreting its color bands. We define this interpretation process to be the “resistor identification process”.

Utilizing the color code convention to read resistor parameters can be challenging at times: it is required that the user has adequate color vision; the user needs to learn the convention; and the user must memorize the mapping between each color and its corresponding significant, multiplier, and tolerance [1]. If any of the above criterions are not properly met, the user’s efficiency at utilizing the color code convention will be greatly reduced. Additionally, a user originally proficient at the convention might also losses memory of the convention overtime, requiring the user to spend additional time re-familiarizing the convention when it is needed.

Thus, an automated “Optical Resistor Identifier Scanner (ORIS)” with a camera and a user interface (UI) to interpret the resistor color code and report the resistor parameters to human is intuitively useful to help speed up the resistor identification process. Ultimately, the ORIS will be used not only by individual users or organizations to aid with electronic prototyping, but also by manufacturing and recycling organizations to help with inventory sorting and resistor recycling.

Image processing based prototypes of the ORIS had been developed in the past on mobile platforms such as Android, but they were proven to be useable only in an extremely limited controlled environment (much more limited than that of the prototype proposed in this literature), with issues such as: can be used only in a fixed background (usually white); failing to identify the resistor if glare appears on the resistor; erroneous identification results when there are small variations in the surrounding’s lightning conditions. All these issues will be addressed or mitigated in our proposed prototype through machine learning techniques, to bring this conceptually useful tool closer to a production-ready stage.

Assuming the ORIS is eventually made into a reasonably priced product, it can help the organizations frequently using axial leaded resistors such as universities, electronic companies to significantly reduce the cost of sorting and recycling left-over resistors to help reducing the electronic wastes produced, which then will help to preserve the Earth’s environment and demonstrate the organizations’ positive attitude towards environmental issues.

In this paper, we present a prototype to the ORIS built using machine learning following the expectations set out by our client in the client’s Project Proposal [2]. More specifically, we will discuss in detail our prototype design, implementation, and performance results, as well as future works that can be done to further improve the prototype we proposed. In the process, we will mention the client requirements/issues solved by our prototype and those that our prototype fails to solve.

# Prototype Design, Implementation, and Results

## Project Proposal Response

The project proposal generated by the client identifies the following objective for the ORIS: create a camera-based scanner that will reliably identify axial leaded resistors using the standard color code [2]. The project proposal defined the main elements of the ORIS in terms of the following objectives and success criteria:

### ORIS identifies resistors under various lightning conditions.

### ORIS must identify ¼ W resistors with both beige and blue colored bodies.

### ORIS must identify one or a group of resistors with the same value.

### Cost of the prototype is less than $100.

### The prototype dimension is less than 30x30x30 cm3.

### The prototype is line-powered with a power consumption of less than 5 W.

1) to 3) were the main objectives that guided the development of the prototype discussed in this literature, and 4) to 6) were the constraints on the prototype specified by the client. The successfulness of the ORIS prototype will be defined using 1) to 6), but this literature will primarily focus on 1) to 3) since they encompass the core objectives of the ORIS.

## Preliminary Design

Having acknowledged the project proposal with our proposal response, the preliminary design is initiated. We simulated the possible use cases of the prototype from a user’s standpoint, then used the use cases to realize specific functional and performance requirements for the prototype. The key functional requirements (FRs) and performance requirements (PRs) are shown in TABLE I and TABLE II respectively.

1. Key Function Requirements

|  |  |
| --- | --- |
| FR # | Description |
| FR-01 | ORIS identifies the resistance and tolerance of the given resistor. |
| FR-02 | ORIS has a screen with a Graphical User Interface (GUI) to interact with the user. |
| FR-04 | ORIS keeps a record of identified resistor that can be retrieved using an external device. |

1. Key Perfomance Requirements

|  |  |
| --- | --- |
| PR # | Description |
| PR-01 | The accuracy of identification under well-lit condition is at minimum 90%. (Relating FR-01) |
| PR-02 | Video streaming camera capture and detection results to the user at a minimium of 5 frames per second (fps) on the GUI. (Relating FR-02) |
| PR-03 | Local file record of identified resistors. (Relating FR-04) |

With the key FRs and PRs realized, the system architecture shown in Fig. 1 were generated, which includes the following key hardware components:

### Single Board Computer (SBC): for running the ORIS software and interfacing with hardware peripherals.

### Camera Module: for capturing image streams and fetch it to the SBC for further processing.

### Touchscreen Module: for providing the UI to the user and receiving user inputs.

Regarding the ORIS software, aside from the Application Programming Interfaces (APIs), which we will utilize rather than implement, the following are the key components/modules:

### Object Detection Stack (includes the Image Preprocessor, Object Detection Algorithm, and Image Postprocessor): for recognizing each resistor color band and locating it in an input image.

### GUI Manager: for providing the GUI on the touchscreen module to the user and receiving user commands.

### Scan Record Manager: for providing an interface to organize and save the resistor identification results to a filesystem.

### Hypertext Transfer Protocol (HTTP) Server: for hosting a server for sharing the resistor identification results to an external device.

The minimal required architecture for the external device in 7) is also shown in Fig. 1. It is required that the external device has a network stack and the associated HTTP client to access the HTTP Server on the ORIS. Most modern devices (e.g., laptops, smartphones) satisfies this requirement.

Diagram

Description automatically generated

Fig. 1. System Architecture Diagram of the ORIS prototype.

## Component Selection and Evaluation

To instantiate the abstract system architecture shown in Fig. 1, the set of hardware (HW) components selected is identified in TABLE III.

1. List of hardware selections

| HW | HW Selection | | |
| --- | --- | --- | --- |
| Component | Manufacturer | Quantity |
| SBC | Raspberry Pi (RPi) 3 Model B+ | Raspberry Pi | 1 |
| Camera Module | RPi Camera Module v1.3 | Raspberry Pi | 1 |
| Touchscreen Module | LCD Touchscreen HAT for RPi | SparkFun | 1 |

The RPi 3 Model B+ is chosen to be the SBC since it is equipped with a powerful processor capable of doing complex image processing in real time. Additionally, the RPi also has support for an official Debian-based Operating System (OS) named Raspbian, which can be setup to run Python, the programming language we will use to implement the ORIS software, with ease. Choosing the RPi for the SBC at the prototyping stage allows us to avoid implementing a firmware to drive the SBC, to allow us to focus on the software components and algorithms for the ORIS and reduce the risk of firmware errors/debugs. The camera module and touchscreen module were chosen mainly based on the decision of the SBC: the RPi Camera Module was selected as it is the has official support for the RPi and has fine imaging performance; the LCD Touchscreen HAT for RPi was selected for its compatibility with the RPi and reasonable price.

The set of software APIs needed to implement the ORIS is then specified and selected following our decision on the HW, as shown in TABLE IV.

1. list of software api selections

| API | API Details | | |
| --- | --- | --- | --- |
| Library | Developer | Version |
| Object Detection API | TensoFlow Lite | Google | 2.7.0 |
| Image Processing API | OpenCV | OpenCV Team | 4.5.3 |
| Camera API | picamera | waveform80 | 1.13 |
| Display API | tkinter | Python Foundation | 3.7.3 |
| Filesystem API | Python built-in | Python Foundation | 3.7.3 |
| Network API | Python built-in | Python Foundation | 3.7.3 |

The TensorFlow Lite (TFLite) was selected for its high inference performance on edge devices and inherent compatibility with TensorFlow detection models. The OpenCV was selected for its speed and comprehensive set of image-processing functions. The Picamera library was chosen to interface with the RPi Camera Module. And the display, filesystem, and network APIs were chosen as they are finely integrated into Python and provide a lightweight interface to the underlying functionalities. It is worth noting that all of the software APIs in TABLE IV are high-level, and there is no low-level (driver-level) APIs to access the camera and touchscreen hardware, this is because the low-level interfacing to both of these hardware is done by the Raspbian OS.

To test and evaluate the performance of our selected hardware components and software APIs, we implemented the “walking skeleton” of the ORIS utilizing all the hardware and most of the APIs [3]. The walking skeleton consists of a GUI to feedback the image stream captured from the camera to the user on the touchscreen at around 6 fps. It proved our selection to be adequate to meet the FRs and PRs mentioned in *B. Preliminary Design*.

## Implementation

The implementation of the ORIS focuses around two key elements relating to FR-01 and FR-02, which are:

### Correct identification of each color band on a resistor together with the localization of each band in a give image.

### Providing a responsive GUI to the user while running the computationally heavy band detection algorithm on the edge device (RPi).

1) defines the core algorithm of the ORIS. This algorithm is to identify and locate each color band on a resistor in a given image, so that the parameters of the resistor can be further extracted following the color code convention. We define this algorithm to be the “band detection algorithm”. We approached this algorithm from a machine learning standpoint and utilized the SSD Mobilenet V2 object detection model [4] at the core of the band detection algorithm. A flowchart of the algorithm is shown in Fig. 2.

Diagram

Description automatically generated

Fig. 2. Flowchart of the Band Detection Algorithm.

After the postprocessing step in Fig. 2, the ORIS will have enough information regarding the color and relative position of each of the color bands. This information can then be interpreted to find the resistor’s resistance and tolerance with ease by following the standard color code convention.

At the core of the algorithm is the SSD Mobilenet V2 object detection model, which will be trained using the procedure shown in Fig. 3 with the TensorFlow Object Detection API [5]. Details regarding the utilization of the TensorFlow Object Detection API can be found at [6].

Diagram

Description automatically generated

Fig. 3. Procedure of Training the SSD Mobilenet V2.

Most of the implementation regarding the specific training of the model is provided by the TensorFlow Object Detection API. The training-set images were collected from the RPi Camera Module using a custom script, then labelled manually to generate the category and bounding box for each color band in each collected image. The training was processed on a desktop computer to significantly reduce the training time, and the produced model was converted into a TFLite model for running on the RPi.

Aside from training and utilizing the object detection model, another challenge regarding the ORIS prototype was 2). Though the RPi is a computationally powerful SBC, it can only run the SSD Mobilenet V2 (a considerably lightweight model) with a speed of approximately 1.5 inferences per second. Thus, if a sequential program is made to repeatedly run the inference and update the GUI, the GUI will be limited to an update of 1.5 frames per second (fps) ignoring any other overheads, which falls far behind PR-02 defined previously.

The solution proposed to this problem was to exploit program parallelism to effectively decouple the inference procedure from the main GUI. Specifically, the ORIS implementation will be consisted of two processes: one for the inference procedure, and another for providing the GUI; following the schematic shown in Fig. 4.

Diagram, schematic

Description automatically generated

Fig. 4. The multiprocessing architecture of the ORIS.

Note that another thread, the camera thread, is used to perform continuous capture from the camera to help further in making the GUI as responsive as possible, since camera capture is a heavily input-output (IO) bounded task. The two main processes, the inference process and the GUI process, are connected using two multiprocessing pipes. One for sending the preprocessed image (a matrix of pixels) from the GUI process to the inference process, and another for receiving the inference results (formatted into an object) from the inference process to the GUI process. Additionally, the GUI process accesses captured image from the camera thread via shared memory. Additional controls for each process/thread were also implemented to start, suspend, and stop it, which are omitted in Fig. 4.

## Main Tasks and Timelines

Aside from the preparation time (the time to order and test the hardware, the time for us to familiarize with machine learning and the development environments), the development timeline consists of 9 weeks which we divide into 3 development iterations, each spanning approximately 3 weeks. The 1st development iteration focuses on the preliminary design of the ORIS prototype, which involves the following main tasks:

### Develop a small script for collecting sample images for training and testing the object detection model.

### Use the script in 1) to collect and label a small dataset and use it to train a preliminary detection model. Evaluate the model.

### Develop a walking skeleton utilizing the trained detection model with a preliminary GUI to feedback the camera capture and what the model detects on the touchscreen.

The 2nd development iteration builds on top of the 1st, and in summary focuses on converting the preliminary design into a basic, working prototype. It consists of the following main tasks:

### Collect and label a larger dataset under a fixed surrounding environment for enhancing the object detection model.

### Train the object detection model using dataset in 4). Repeatedly evaluate the model, tune the training parameters, and re-train to find the optimal model.

### Implement an algorithm for interpreting the object detection results using standard color code to find the resistance and tolerance of a given resistor.

### Implement a basic, more user-friendly GUI with elements reflecting the key FRs and FRs using 5) and 6).

The 3rd and last development iteration builds on the previous 2, and in summary consists of improving the basic prototype built previously into a more user-friendly final prototype, with additional FRs and PRs realized and met:

Parallelize the basic prototype using multiprocessing and threading to improve performance.

Implement the scan record sharing HTTP server for enhancing the user-experience.

### Collect and label an even larger dataset under different surrounding environments for enhancing the object detection model.

### Train the object detection model using dataset in 10). Repeatedly evaluate the model, tune the training parameters, and re-train to find the optimal model.

### Integrate 8), 9) and 11) together with a more advanced GUI to form the final ORIS prototype.

At the end of each development iterations, the main tasks completed in that iteration are tested and evaluated to identify issues and find improvement points. Any issues found will be addressed and corrected before the start of the next development iteration, and any improvement points found will be added to the development tasks according to their priority.

## Integration and Test Results

In terms of the hardware, the assembled ORIS prototype is displayed in Fig. 5 below. The LCD touchscreen HAT is attached onto the RPi via the General-Purpose IO (GPIO) pins, and the RPi Camera Module is connected to the RPi via the Camera Serial Interface (CSI) port. The communication interface to the LCD touchscreen and camera module is Serial Peripheral Interface (SPI) and CSI respectively. The software integration and implementation of the ORIS prototype can be obtained from [7].

A picture containing text, indoor

Description automatically generated

Fig. 5. Assembled hardware of the ORIS prototype.

Regarding the software, the identification accuracy of the ORIS prototype had been tested under a well-lit environment on a total of 30 resistors, in which 15 were randomly selected from a batch of beige-bodied resistors, and the other 15 were randomly selected from a batch of blue-bodied resistors. The test was performed with the resistor placed on a white background, and a human operator holding and aiming the camera at the resistor. For each resistor tested, we recorded the number of color bands correctly detected. The results are shown in Fig. 6 below.

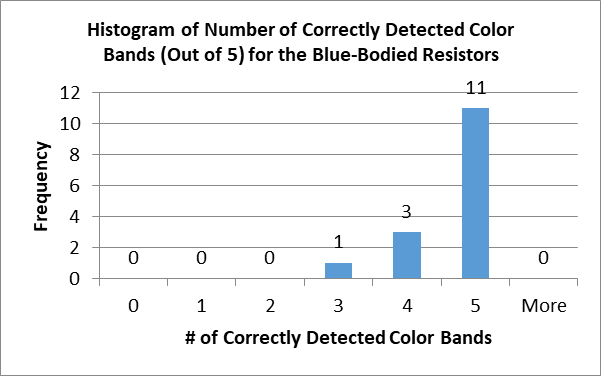
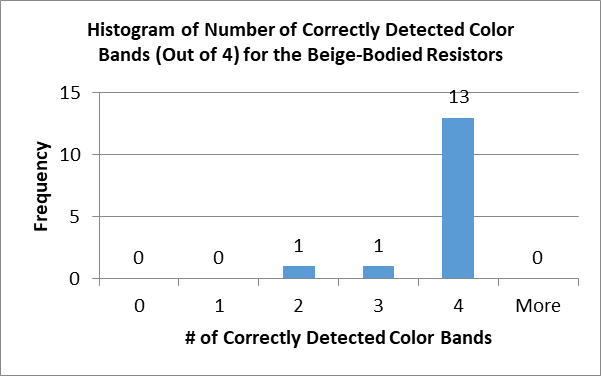


Fig. 6. Color band detection accuracy results.

Since a deterministic algorithm is used to interpret detected color bands into the resistor’s parameters (resistance and tolerance), if all of the color bands are correctly detected (recognized), it is guaranteed that the interpreted parameters will be correct, meaning that the resistor will be successfully identified. Hence, we can derive the overall resistor identification accuracy from the results in Fig. 6 by dividing the frequency of all bands being detected correctly by the total frequency and obtain an overall identification accuracy of 80%. The color band recognition accuracy can be found by dividing the total number of correctly detected color bands by the total number of color bands seen, which yields 94.07%.

Aside from the accuracy aspect of the ORIS prototype, the multiprocessing architecture was also benchmarked in comparison to a serial architecture developed to investigate performance gain and assess compliances with the FRs and PRs. By running the ORIS prototype over a time period of 5 minutes, we observed the average fps of the GUI via a fps-counter and found that: when running the serial architecture, the ORIS achieved on average 1.2 fps on the GUI; and when running the multiprocessing architecture, the ORIS achieved on average 4.1 fps on the GUI. Thus, the multiprocessing architecture achieved on average a 3.42 times speedup compared to the serial architecture.

The record sharing HTTP server was also verified to be functioning accordingly, but its detailed testing will be omitted in this paper. Overall, we can summarize the compliance to the FRs and PRs defined in *B. Preliminary Design* of our implemented ORIS prototype into TABLE V.

1. Table Type Styles

| FR/PR # | Compliance | |
| --- | --- | --- |
| Description | Status |
| FR-01 | ORIS identifies the resistance and tolerance of the given resistor. | Met |
| FR-02 | ORIS has a screen with a Graphical User Interface (GUI) to interact with the user. | Met |
| FR-04 | ORIS keeps a record of identified resistor that can be retrieved using an external device. | Met |
| PR-01 | The accuracy of identification under well-lit condition is at minimum 90%. (Relating FR-01) | Not Met |
| PR-02 | Video streaming camera capture and detection results to the user at a minimium of 5 frames per second (fps) on the GUI. (Relating FR-02) | Not Met |
| PR-03 | Local file record of identified resistors. (Relating FR-04) | Met |

As shown in TABLE V, the ORIS prototype failed to meet PR-01 with its current identification accuracy at 80%. The ORIS also failed to meet PR-02 as its current GUI could only achieve on average 4.1 fps in terms of video streaming and displaying the detection results. The solution to these issues may be further addressed by the improvement points/solutions as discussed below.

PR-01 was not met likely due to the limited training-data used to train the detection model. It is evidenced from our evaluation of the object detection models trained that with additional training on more dataset, the model’s accuracy on detecting color bands will be significantly increased, so the identification accuracy will also be increased to address the accuracy issue.

PR-02 was not met due to the lack of performance of the RPi in terms of running machine learning (ML) inferences. To address this performance issue, the ORIS software can be re-implemented in a lower-level language such as C++ instead of Python to optimize the memory access overheads and fine-tune program characteristics, to improve the responsiveness of the GUI. The issue may also be addressed by upgrading the hardware to include a ML accelerator such as the Coral USB Accelerator, so that the inferencing workload can be shifted from the processor on the RPi onto the external accelerator, allowing the processor on the RPi to focus on providing a responsive GUI to the user along with a great increase in inferencing performance.

In summary, we concluded that the ORIS prototype had successfully accomplished objectives 1) to 3) mentioned in *A. Project Proposal Response* but with much room for improvements. Regarding 4) to 6), the prototype accomplished 5) as its combined volume is approximately 20x20x10 cm3. However, the prototype failed to meet 4) as its total cost is $130.73 including only the hardware; and it also failed to meet 6) as its power consumption is estimated to be 12.5 – 15 W. Nevertheless, we declare the prototype to be overall effective as it meets most of the objectives.

Overall, the project execution and delivery of the ORIS prototype was successful, with the only issue being the limited timeframe available to produce the prototype. If given a larger timeframe, the prototype can be further improved in terms of its accuracy, performance, and user-experience, with additional functionalities explored and realized. In summary, the produced ORIS prototype is overall reasonably effective with a resistor identification accuracy of 80% and a GUI responsiveness of 4.1 fps. Through the evaluation of the prototype, it is evident that the prototype requires more future work to be done for it to be classified as a product. Thus, at this current stage, the ORIS prototype is more of a research and development (R&D) project.

# future work

As mentioned previously, there are many improvement opportunities that can be made to improve the accuracy, performance, and user-experience of the ORIS prototype to shift it from its current R&D stage into a more product-ready stage. Three of such possible enhancements will be discussed in more detail in this section.

## Improve the Identifacation Accuracy

The resistor identification accuracy of the ORIS prototype is currently 80%, which falls behind PR-01 and greatly hinders the effectiveness of the ORIS. The accuracy can be improved by training the object detection model, SSD Mobilenet V2, with a larger dataset that includes a larger variety of resistors and the surrounding environment, so that the model can learn the appearance of different color bands on differently made resistors under different lightning conditions. A re-tune of the training parameter will also be needed to account for the change in the training-dataset.

## Improve the GUI Performance

With a refresh rate of 4.1 fps on the GUI of the current ORIS prototype, it fails to meet PR-02. Though 4.1 fps is sufficient from a usability standpoint, it is not feasible in terms of user-experience. To improve the refresh rate, an ML accelerator can be added as discussed before, though this will further increase the cost of the prototype. Another method to improve this is to re-implement the ORIS’s software in a lower-level language (e.g., C++), with a greater emphasize on optimization of memory accesses and peripheral IOs, in combination with a lower-level GUI library such as Qt. With this latter method, the refresh rate can be easily made to achieve PR-02, since a speedup of 10-50 times for the program is expected.

## Further Improve the User Experience

The ORIS prototype currently lacks the use of graphical elements on its GUI and its GUI is heavily text-based. Graphical elements can help the user quickly interpret the identification results on the ORIS and greatly increase user’s efficiency at using the ORIS. Hence, graphics such as the pseudo-resistor shown in Fig. 7 can be implemented onto the ORIS’s result UI to aid the user in reading and verifying what the ORIS detects. Additional animations can also be implemented on ORIS’s scan UI to provide more feedback to the user regarding the status of the ORIS.

Chart

Description automatically generated

Fig. 7. GUI with pseudo-resistor and menu for selecting each color band [8].

## Autonomous Data Collection

As mentioned in *A. Improve Identification Accuracy*, the identification accuracy of the ORIS prototype depends heavily on generating a well-trained object detection model using large dataset. Thus, an autonomous data collection scheme can be made to automatically collect and label new dataset and train the model to continuously improve the ORIS’s accuracy, with the below steps:

### Implement a GUI on the result UI of the ORIS similar to that shown in Fig. 7.

### Whenever ORIS shows a resistor identification result to the user, the user will be given an option to claim that the detected color bands are incorrect.

### If the user makes the claim in 2), the user will be given the option to input what the user believes are the correct color bands for the resistor.

### The user input in 3) will be uploaded to a cloud server along with the original image of the resistor.

### After a certain amount of user inputs in 4) have been gathered at the cloud server, the cloud server formulates them into a new dataset and re-train the object detection model.

### The re-trained model can be downloaded by the ORIS and used further to detect new resistors.

This scheme relies on the ORIS being a widely used device with many users to function optimally. Nevertheless, if it is deployed under that condition, a very accurate model will be obtained significantly faster than manually collecting and labelling new datasets, and the detection model will continuously improve overtime.

# conclusions

In conclusion, this paper investigates the implementation of the Optical Resistor Identifier Scanner (ORIS) prototype using machine learning techniques, with the aim to develop a camera-based scanner that can reliably identify axial leaded resistors using the standard color code. The prototype was designed, developed, and tested, and it was shown to be reasonably effective under a controlled environment for a small set of resistors. Limitations of the traditional image processing based ORIS prototypes, such as only working for resistors placed on a fixed background, failing to identify resistors with glares, failing to work when lightning conditions are varied, were addressed by the presented ORIS prototype through machine learning approaches.

Two significant challenges encountered with developing the ORIS prototype was: first, the low color band detection accuracy with the initially trained machine learning model; and second, the insufficient performance of the Graphical User Interface (GUI) with a serial software architecture. The first challenge was overcome with a larger, more robust training-dataset along with comprehensive tuning of various training parameters. And the second challenge was overcome by the design and implementation of a multiprocessing architecture that exploits parallelism between the different tasks in the software. Overall, the ORIS prototype is successful as a R&D project, with an overall resistor identification accuracy of 80%, and a GUI refresh rate of 4.1 fps. And there exist many improvement opportunities to bring the current ORIS prototype into a more product-ready stage.

##### Acknowledgment

We thank Dr. Steven Knudsen, our client, for providing the detail and guidance for the ORIS prototype; Raju Machupalli and Stephen Fan for continuous support and advising on carrying out the project; and Steve Drake for lending us the necessary equipment. This project was supported and funded by the Faculty of Engineering at the University of Alberta.

##### References

1. “Eletronic color code”, *Wikipedia*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Electronic\_color\_code#Color\_band\_system. [Accessed: 27-Mar-2022].
2. S. Knudsen, “Optical Resistor Identifier Scanner”, *GitHub*, 2022. [Online]. Available: https://github.com/xuanrui-work/ECE492-ORIS\_public/blob/main/Project%20Proposal%20by%20Steven%20Knudsen.pdf. [Accessed: 01-Feb-2022].
3. X. Zeng, “ORIS Prototype – Walking Skeleton”, *GitHub*, 2022. [Online]. Available: https://github.com/xuanrui-work/ECE492-ORIS\_public/tree/main/walking\_skeleton. [Accessed: 29-Mar-2022].
4. “TensorFlow 2 Detection Model Zoo”, *GitHub*, 2022. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/tf2\_detection\_zoo.md. [Accessed: 10-Feb-2022].
5. “TensorFlow Object Detection API”, *GitHub*, 2022. [Online]. Avilable: https://github.com/tensorflow/models/tree/master/research/object\_detection. [Accessed: 10-Feb-2022].
6. L. Valadimirov, “Training Custom Object Detector”, *Tensorflow 2 Object Detection API tutorial*, 2022. [Online]. Available: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html. [Accessed: 10-Feb-2022].
7. X. Zeng, D. Feng and J. Kan, “ORIS Prototype – Main Implementation”, *GitHub*, 2022. [Online]. Available: https://github.com/xuanrui-work/ECE492-ORIS\_public/tree/main/oris. [Accessed: 31-Mar-2022].
8. “4 Band Resistor Color Code Calculator”, *DigiKey*, 2022. [Online]. Avilable: https://www.digikey.ca/en/resources/conversion-calculators/conversion-calculator-resistor-color-code. [Accessed: 02-Apr-2022].