# Autonomous IoT-based Sorting Machine

Nithin Thomas
*School of Computer Science*
*Trinity College*
Dublin, Ireland

Rakesh Nair
*School of Computer Science*
*Trinity College*
Dublin, Ireland

Misbah Rizaee
*School of Computer Science*
*Trinity College*
Dublin, Ireland

Mohammad Mahdi Eslami
*School of Computer Science*
*Trinity College*
Dublin, Ireland

Tapabrata Mukhopadhyay
*School of Computer Science*
*Trinity College*
Dublin, Ireland

***Abstract*—The Internet of Things is a paradigm of technology which has transformed the way of living and embedded computing in everyday life. This is more significant and actively pursued for data transformation in various industries as a part of Industry 4.0. In this report we discuss the design and realisation of a sorting system which can be useful in warehouses and supply chain management. We explain the choices on communication technologies and protocols, and the set up of Cloud service to create a robust system. We further discuss the challenges we faced, and how we worked around them. The design serves as a prototype which has to be further improved and can be extended to fit different types of sorting mechanisms. We discuss these possible improvements and also our learnings from the activity in this report.**

***Index Terms*—internet of things, wireless communication, microcontrollers, system-on-a-chip, wireless fidelity, automation, industry applications, cyber-physical systems, cloud computing**

## I. Introduction

Technological advancements have made it feasible to effectively implement real-time monitoring systems in every domain. IoT technologies have shown to be potential options for societal improvement.

One of the benefits that IoT-based technologies bring about is to enhance the production lines in factories and manufacturing sites. The mining, automotive, agricultural, food and beverage, bottling, manufacturing, warehousing, logistics and packaging sectors all employ conveyor systems. Various commodities are carried using the various conveyor belts available to handle various weights and materials. Conveyor belts can deliver items in a straight line or across elevation or direction changes. As computer and mechanical sciences advance, we anticipate to see more automatic and smart conveyor belts which could fulfill all the requirements in this area and potentially result in more efficient production lines [1].

In this project, we intend to prototype a sorting machine which is fully autonomous using IoT. By doing so, we will discover what challenges are involved in building such device. We also plan to introduce some IoT based methods to push the device towards being more intelligent and make a fully standalone sorting machine. In essence, what the protype is capable of is classifying the packages based on their types, sliding them to the relevant section as well as keeping track of number of the packages. To achieve these objectives, we partition the project into three sub systems. The conveyor belt, the item classifying mechanism and the cloud component. This will involve local communication between the devices which coexist in the environment as well as cloud communication to transmit data.

### *A. Conveyor Belt*

Conveyor belt is responsible to carry the packets on a sliding area from a starting point to the end. In practice, we require a rotating mechanism to rotate the belt. The details of the construction of the conveyor belt are explained in Section 7, part F.

### *B. Item Classification Algorithm*

Item classifier is responsible to detect the type of the packet and classify them accordingly. This requires a real-time device to device communication between the component which detects the class of the item and component which is responsible to rotate the arms to move the item from the belt to the relevant section.

### *C. Cloud Component*

Cloud component acts as the core of item classification process. There are different ways that we could perform this operation such as image processing. However it is not the focus of this paper. The cloud also acts as a dashboard of the sorting machine and it used to be to keep track of number of items assigned to each class, if exceeds, the packet is moved to over-flow section. Fig. 1 illustrates the basic architecture of the whole system. All alternatives for each individual component are discussed on the following sections of this paper.

The structure of the rest of the paper is as follows: Development Environments, Cloud Component, Cloud Communication, Local Communication, Color Detection Algorithm, Challenges and Reflection.

## II. Development Environment

### A. ESP-IDF

Espressif's official IoT Development Framework for the ESP32, ESP32-S, and ESP32-C range of SoCs is called ESP-IDF. It includes a self-contained SDK for developing generic applications on various platforms using programming languages like C and C++. ESP-IDF presently powers millions of devices in the field and allows for the creation of a wide range of networked items, from basic light bulbs and toys to large appliances and industrial equipment [2]. In our case, since we need our OS to support real-time operations,we used FreeRTOS as the OS and ESP-IDF as the framework to work with ESP-32 microcontrollers. ESP-IDF is an open-source framework and has a rich documentation and quick start examples, making it easy for beginners to develop their IoT projects.

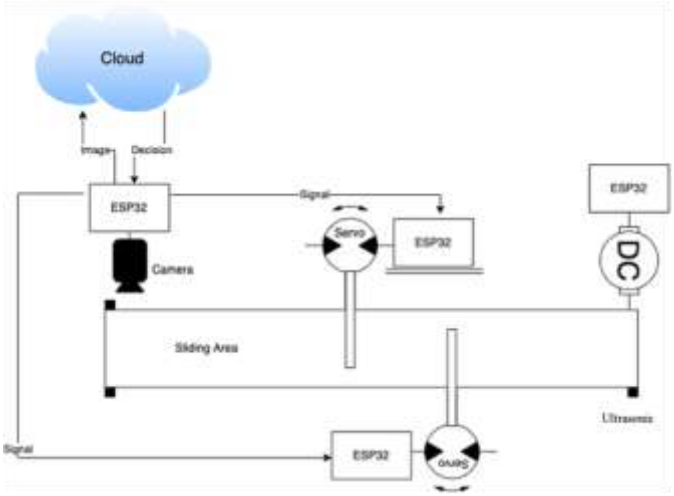


Fig. 1. Overall architecture of the system

### B. Hardware Requirements

Our model comprises a rotating conveyor belt. Its movement is executed using an ESP32-WROOM-32 [3] microcontroller that pauses/resumes a DC motor attached to the belt. We have used the ESP32-S2-Kaluga-1 Kit v1.3 [4] which has a ESP-LyraP-CAM [5], which functioned as our main sensor, and an ESP32-S2-WROVER [6] microcontroller, which we used to transmit the captured image of the packages to the cloud. We have also used an ultrasonic sensor to sense the arrival of a new package so we can pause the conveyor belt and let the cloud communication and colour detection happen.

Our actuator is a servo motor with a wooden hand attached on top. This motor is controlled using another ESP32-WROOM-32 microcontroller. The microcontroller activates the servo motor so it can move the hand to ensure the packages are directed off the belt and they end up in their right destination containers.

## III. Cloud Component

After analysing the performance and features provided by the multiple cloud providers – AWS, GCP, and Azure [7], [8], it is quite evident to us that going with the AWS cloud provider is the right decision because of its ease of integration of ESP32 device with multiple services provided by AWS. The main decision-making point for us is the QuickSight service provided by the AWS. By having this service, we won't have to work on writing an application to visualise the inventory related data, we can store the data directly to storage services supported by the QuickSight such as Athena, S3, Aurora and RDBS related databases. For database, we decided to use RDS – MySQL, even though the performance of the Aurora is five times faster than the standard MySQL and PostgresSQL related databases, the pricing for the Aurora is quite higher with compared to RDS - MYSQL and there was no requirement for having a fast and frequent read/write operations.

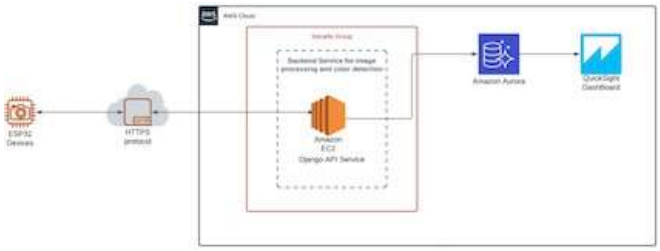

Fig. 2. Cloud architecture of Autonomous Sorting Machine

Initially we deployed our color detection model as a serverless function using lambda and API Gateway service. The problem we had with serverless is that the API Gateway is exposed as a HTTPS protocol but the ESP32 Kaluga device could not connect the endpoint using https. We generated certificates and private keys for the authentication and pass the same to the device for calling the endpoint, even still we had the same issue and after going through the forum, we got to know that the If we use camera module on the Kaluga system and try to establish SSL connection, it might trigger a memory issue because of which it was not possible to have secure connection between local IoT devices and the cloud components. Later, we shift our approach from using serverless services to deploy our color detection model/service on the EC2 instance, which is a virtual server provided by AWS infrastructure for deploying user's application. We build our backend service using Python - Django framework which acts a HTTP server and process the received requests.

*A. The flow of actions after parcel is detected by the ESP32 ultrasonic sensor -*

- The camera captures the colored parcel and send the binary version of the image to the backend service deployed in the AWS Cloud.
- The color detected model will process the provided image and pass the outcome to the backend service.
- The backend service stores the processed result in the RDS-MySQL database and return the result as a response to the requested ESP32 device.
- Using the return response, the ESP32 will actuate the sensors to drop the parcels in the appropriate container.
- The distribution authority can view the visualization of the capacity and statistics of the containers using the QuickSight dashboard. Only the authorized authority can login and view the dashboard using username/password provided to them.


Fig. 3. Console Output from the Colour Detection Service

*B. Security aspects of Cloud architecture -*

- **Security Groups** – We have attached multiple security groups which is a stateful virtual firewall provided by AWS for controlling outgoing and incoming network traffic. It helps in ensuring that only the ip addresses and ports mentioned in the security group rules are allowed to flow at the instance level for establishing connections. We have allowed few ports and the subnet of our devices to access the EC2 instance for establishing a connection with API endpoint.
- While creating the **RDS instance**, we created it as a private instance because of which the instance is assigned with public ip address and no device outside the AWS VPC network is allowed to connect the RDS instance. As the EC2 instance in which we are running our http service is also in the same VPC network, it is allowed to connect without the need to traverse through internet. We have added additional control using AWS IAM instance profile attached to our EC2 instance in which we have mentioned the rules to only allow read/write actions to the specific RDS instance.
- The **QuickSight dashboard** created for visualising the capacity inventory can only be viewed by the distribution authorities invited by the admin. They will receive a login id (email address) and a temporary password, using which they can login the dashboard portal for first time, but they will have to set a new password by following a compliance defined by the AWS service. We can enhance the security level by having authentication based on IAM users instead of username/password.

*C. Inventory Dashboard -*


Fig. 4. Inventory Dashboard using QuickSight

The AWS QuickSight allows the businesses to visualize, analyse and extract easy-to-understand insights used for making a necessary decisions. It supports multiple data sources = RDS, S3, RedShift, and ElasticSearch. We went with RDS-MySQL based on pricing and easy to use APIs. It uses a pre-trained machine learning techniques to detect anomalies and understand the trends. We have visualize the capacity of each containers based on different time frame to understand the patterns and frequencies, From dashboard we can get to know that the frequency of the packages is high in between 17:30 and 18:30.

## IV. Cloud Communication

For communication of hardware to the cloud services, we considered MQ Telemetry Transport (MQTT) and Hyper Text Transfer Protocol (HTTP). The cloud communication is to send a captured image and get the response indicating the colour of the image. Using MQTT ESP32 device has to act as a publisher to send images, as well as a subscriber to receive results. The image and result has to be numbered and compared at device to ensure the right decision. The request response cycle of HTTP works better for this use case.

We use the 'http client' provided by ESP IDF IoT Development platform to create an http request, where the captured image is attached as raw bytes as payload and content type set to 'image/jpeg'. This request is sent to the cloud server, and response is expected in raw data format indicating the colour of the object in image. We used Transport Layer Security (TLS) implementation in ESP IDF for added security. Certificates were generated for the Cloud server for this purpose. However, with the limited memory in ESP32, the device could not enable the camera module and TLS layer simultaneously. For this reason, we had to revert from HTTPS to HTTP requests.

## V. Local Communication

The sorting system is set up with two ESP32 boards, and a ESP32 S2 Kaluga kit. The first ESP32 board controlling the motor rotating the belt and constantly checking Ultrasonic sensor values. This module stops the belt when an object is detected on the belt, and sends a message to the ESP32 S2 Kaluga kit to scan the object. The Kaluga kit is responsible for capturing images, communicating to the cloud and getting a decision. Once a decision is received, this module communicates with the third module which controls the servo.

The devices communicate to each other over Wifi. Bluetooth was not considered as we used Wifi for cloud communication, and extended the same for local communication. All devices have to be in the same Local Area Network(LAN) and use the local addresses to communicate with each other.

We also looked into the possibility of the system working in the absence of a Wifi router. The ESP32 board can work on dual mode Wifi, being a station as well as access point, which would be used in local communication without a Wifi router. This would not have regular cloud communication, and use a colour sensor instead of camera. The decisions will be made on the edge.

## VI. Colour Detection Algorithm

The image is received in the cloud in binary form from the ESP32. We had chosen to work with red and green boxes because they were among the RGB colour spectrum. So, we had to figure out a way to detect red and green colours from the binary of the image that we were sent over WiFi by the ESP32.

While going through the colour detection literature [9], [10] available to us in the IEEE library, we realized that the algorithms in the literature are meant for high quality images and complex colour detection. Since we neither had high quality images to work with nor were we looking for any complex form of colour detection, we came up with a clever algorithm that would classify between the RGB colours without adding significantly to the latency of the system.

Fig. 5 illustrates our colour detection algorithm. We chose to work with Python3 on the cloud side because of the NumPy library [11] which makes significantly fast calculations on arrays. So, our algorithm's first step is to convert the image binary into a NumPy array of shape 320 x 240 x 3. Then, we run the np.mean() function [12] of NumPy twice on our array, once with Axis=1 and once with Axis=0, to get an array of 3 values as a result. This result array has the mean values of the red, blue, and green values across the whole image. The last step of the algorithm is to find out the index of the result array that has the maximum value. This index indicates the detected color. For example, if in the result array, we have red=100, green=200, and blue=120, then we classify the package as 'Green' because its index contains the highest mean value.

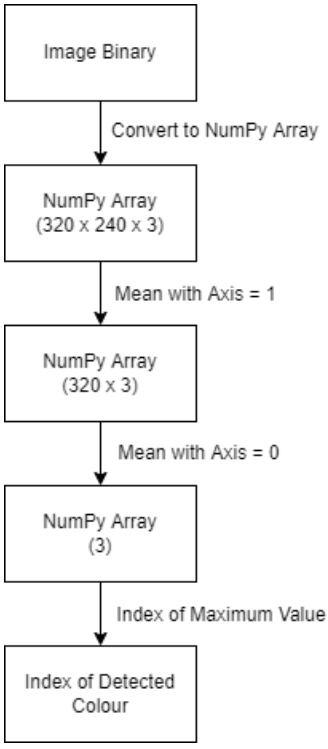


Fig. 5. Colour Detection Algorithm

## VII. Challenges

### A. Camera Latency

The camera module would send old data from the buffer as image data. This means that image will be of a previous object, and not of the object currently in front of the camera. The camera would give the correct image only if 3 or more images were captured of the same object.

Solution: For every required image, the camera was reinitialised completely on the board, to clear any buffer memory.

### B. Camera Quality

The images received from the camera were not up to the quality we expected when we designed the system. The darker images are always tinted with green pixelations, and at times images are cut in half horizontally and shuffled, so the bottom appears on top and vice versa. Additionally, when the camera was extended from the Kaluga kit with wires, the images were distorted with green lines to the extent they were not suitable for colour classification.

Solution: As we classify the colour of the image, correct lighting of the object was crucial to work with the available quality of the camera. Due to the distortions in image when the camera was extended, we decided to keep it plugged on the kit and keep the kit close to the belt.

### C. Latency in Cloud Communication

The initial design used the camera and cloud service for classification. However, with the realistic round trip time of the requests, the objects would go over the belt and overflow

before a decision was received. This was caused by slow response and late actuation of servo motors on the belt.

Solution: We introduced an ultrasonic sensor on the belt, which would be used to detect an object and stop the belt. The belt would roll only when a response is received from the cloud service.

### *D. Power Consumption*

We had initially considered using Bluetooth to carry out the local communication between the ESP32 microcontrollers because Bluetooth is a decently fast solution for implementing short distance wireless communication. However, high spikes in power consumption (sometimes, over thrice the average) have been observed when both Bluetooth and WiFi functions are run simultaneously on an ESP32 chip [13].

Solution: Since we had to use WiFi to establish the communication between the sensor-side ESP32 and the cloud component anyway, we implemented the local communication on WiFi as well. This would avoid any power spikes because of the simultaneous running of Bluetooth and WiFi functions.

### *E. Camera Placement*

With the initial design, the placement of the camera as well as the frequency of capturing images was important. If the camera would take images every second and send them to the cloud, there would be a lot of images with no object. These requests are not required and would crowd the cloud service. This also meant that if an object moves past the camera within less than a minute, the object will not be classified at all.

Solution: With the introduced ultrasonic sensor, when an object is detected, the belt stops after a predetermined wait. This stops the object at a specific point on the belt, where the camera can be fixed permanently.

### *F. Conveyor Belt*

Given the limited time to fulfill the different aspects of the project, finding proper tools with which the conveyor belt was built was a challenging task. The body Conveyor belt is made of Balsa wood sheets, four bull bearings, two piece of pipes a shaft coupling, a piece of plastic cloth and some bolts and nuts. the shaft of the DC motor is connected to the conveyor belt through the shaft-coupling.

## VIII. REFLECTION

### *A. Evaluation*

In the development of our IoT project, low latency has always been thought of as a key factor in transmitting data to the cloud. Low latency is essential to ensure seamless data transfer and optimizing our project experience. However, multiple analysis on our project revealed that we were getting higher latency than we had initially expected. Although we did come up with the workaround of pausing our conveyor belt for a few seconds so that the necessary processes can be carried out, we would aim at lowering the latency if we had more on time on this project.


Fig. 6. Components of the conveyor belt

Camera quality has also affected the overall performance of our project. It had initially made the colour detection a very difficult challenge. The camera had to be firmly attached and any slight movement caused the camera to output distorted images. The level of lighting in our surrounding also had to be good enough otherwise it was monumentally difficult to detect the colour of the packages. Therefore, it seemed to us that we would need a better camera to implement more complex algorithms, like the ones of Machine Learning for example.

### *B. Future Work*

TLS is a fairly large protocol. We require an efficient memory management system to keep the memory utilization to a minimum or having enough memory space available is a requirement for providing communications security.

Optimizing the power usage is important for several reasons. One of the most obvious reasons is that people demand IoT devices for a lengthy operational life. To make the design more appropriate, we could stop the belt when it is not being used so that it doesn't consume power when there is no package to be sorted.

Image classification is another method that we can use which is the same as the colour classification, but instead of a colour, we would classify and redirect the packages based on a label/text which is placed on them [14]. We would still encounter some of the issues we already had with the colour classification such as low quality images and level of lighting. For this, we would need a camera that takes more accurate images with better quality.

## ACKNOWLEDGEMENT

We would like to appreciate the help and encouragement provided by Professor Stefan Weber. He guided us in the right direction and provided us with important insights into the procedure to be followed as well as the required tools to develop this project.

## REFERENCES


[1] R. T. Yunardi, Winarno and Pujiyanto, ”Contour-based object detection in Automatic Sorting System for a parcel boxes,” 2015 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA), 2015, pp. 38-41, doi: 10.1109/ICAMIMIA.2015.7507998.

[2] Espressif Systems. (2022). *Espressif Systems Home Page. Available at: https://www.espressif.com/ (Accessed: 9 April 2022).*

*[3] Espressif Systems. (2020). ESP32-WROOM-32 Datasheet Version 3.3. Available at: https://www.mouser.com/datasheet/2/891/esp-wroom-32_datasheet_en-1223836.pdf (Accessed: 9 April 2022).*

*[4] Espressif Systems. (2021). ESP32-S2-Kaluga-1 Kit v1.3 ESP-IDF Programming Guide. Available at: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/hw-reference/esp32s2/user-guide-esp32-s2-kaluga-1-kit.html (Accessed: 9 April 2022).*

*[5] Espressif Systems. (2020). ESP-LyraP-CAM v1.1 ESP-IDF Programming Guide. Available at: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/hw-reference/esp32s2/user-guide-esp-lyrap-cam-v1.1.html (Accessed: 9 April 2022).*

*[6] Espressif Systems. (2020). ESP32-S2-WROVER and ESP32-S2-WROVER-I Datasheet. Available at: https://www.elecrow.com/download/product/DPI32110W/esp32-s2-wrover_esp32-s2-wrover-i_datasheet_en.pdf (Accessed: 9 April 2022).*

*[7] L. N. Hyseni and A. Ibrahimi, ”Comparison of the cloud computing platforms provided by Amazon and Google,” 2017 Computing Conference, 2017, pp. 236-243, doi: 10.1109/SAI.2017.8252109.*

*[8] A. S. Muhammed and D. Ucuz, ”Comparison of the IoT Platform Vendors, Microsoft Azure, Amazon Web Services, and Google Cloud, from Users’ Perspectives,” 2020 8th International Symposium on Digital Forensics and Security (ISDFS), 2020, pp. 1-4, doi: 10.1109/ISDFS49300.2020.9116254.*

*[9] J. Li, J. Wang and J. Mao, ”Color moving object detection method based on automatic color clustering,” Proceedings of the 33rd Chinese Control Conference, 2014, pp. 7232-7235, doi: 10.1109/ChiCC.2014.6896196.*

*[10] B. R. Navada, K. V. Santhosh, S. Prajwal and H. B. Shetty, ”An image processing technique for color detection and distinguish patterns with similar color: An aid for color blind people,” International Conference on Circuits, Communication, Control and Computing, 2014, pp. 333-336, doi: 10.1109/CIMCA.2014.7057818.*

*[11] NumPy. (2022). NumPy Documentation Available at: https://numpy.org/doc/ (Accessed: 9 April 2022).*

*[12] NumPy Developers. (2022). numpy.mean Documentation Available at: https://numpy.org/doc/stable/reference/generated/numpy.mean.html (Accessed: 9 April 2022).*

*[13] Last Mintue Engineers. Insight Into ESP32 Sleep Modes and Their Power Consumption. Available at: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/ (Accessed: 9 April 2022).*

*[14] A. Singh, A. Sangwan and J. H. L. Hansen, ”Improved parcel sorting by combining automatic speech and character recognition,” 2012 IEEE International Conference on Emerging Signal Processing Applications, 2012, pp. 52-55, doi: 10.1109/ESPA.2012.6152444.*