Responses to Reviewers

Response to Reviewer 1 Comments

1. Summary

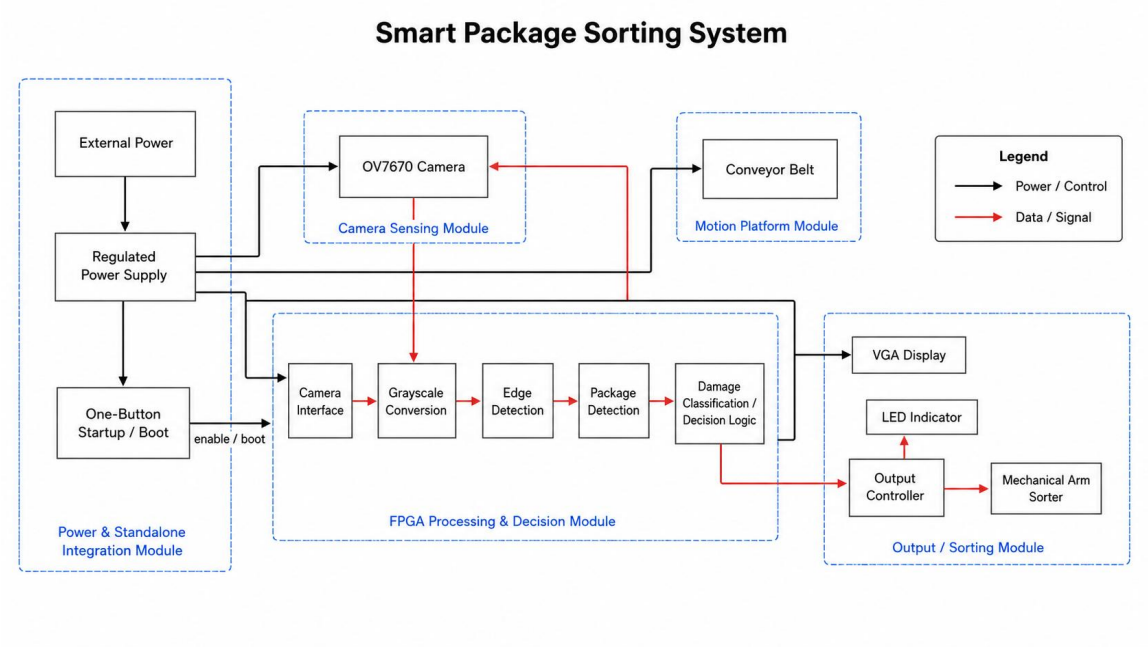
We sincerely thank you for taking the time to review this manuscript. Please find the detailed responses below and the corresponding revisions in the re-submitted files.

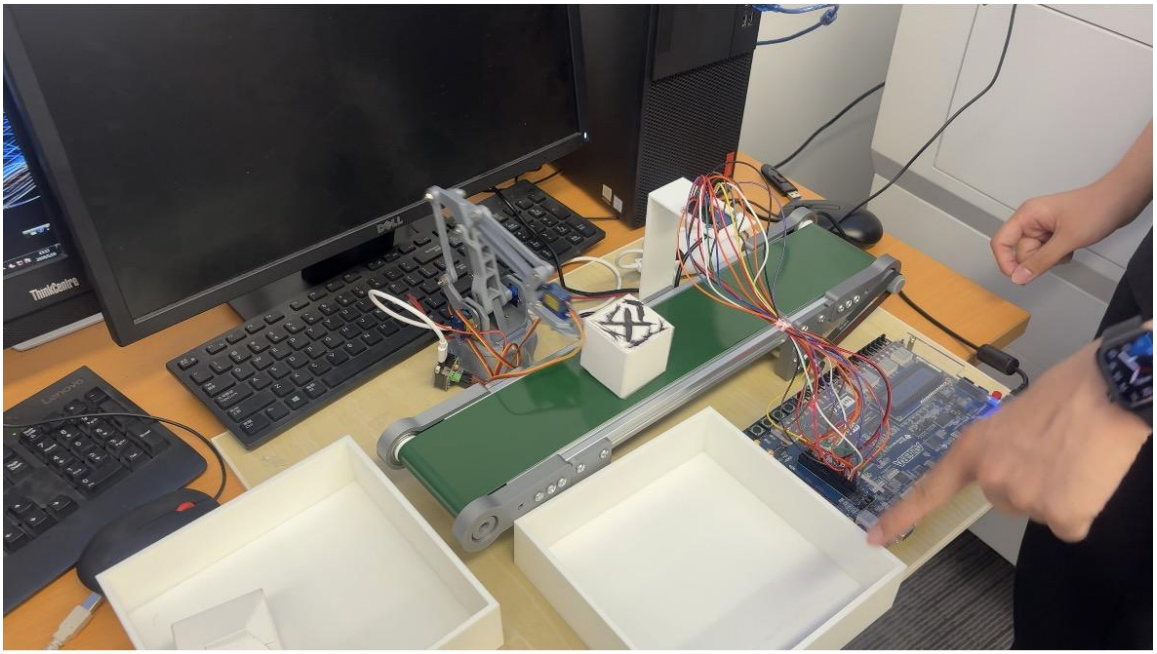
2. Questions for General Evaluation	**Reviewer's Evaluation**
Does the introduction provide sufficient background and include all relevant references?	Yes
Is the research design appropriate?	Yes
Are the methods adequately described?	Can be improved
Are the results clearly presented?	Yes
Are the conclusions supported by the results?	Can be improved
Are all figures and tables clear and well-presented?	Must be improved

3. Point-by-point response to Comments and Suggestions for Authors

Comments 1: Provide a figure of completed device

Response 1: The completed device and the final block diagram are provided.





Comments 2: Provide more details about the sorting mechanism, update there is no event-camera, right?
Response 2:

The final design has been updated to remove the event-camera-based description. The implemented system does not use a DVS/event camera. Instead, it uses an OV7670 frame-based camera as the visual sensing module. The camera outputs image data through an 8-bit data bus, and the FPGA camera interface synchronizes the incoming pixel stream using `pclk`, `href`, and `vsync`. Consecutive camera data bytes are reconstructed into 12-bit RGB444 pixels and stored in the first frame buffer. The FPGA then performs grayscale conversion, Sobel edge detection, ROI-based package presence detection, and rule-based damage classification. Therefore, the processing flow is based on real-time frame-based image processing rather than event parsing, event buffering, or SNN-based detection.

The sorting mechanism has also been described in more detail. After the FPGA completes the package damage classification, it generates a digital output signal representing the sorting decision. This signal is sent to an ESP8266-based output controller rather than directly driving the servos from the FPGA. The ESP8266 monitors the FPGA output and detects the rising edge of the decision signal. Once triggered, it generates PWM signals to control the SG90 servo motors in the robotic arm. The robotic arm then performs a predefined motion sequence to push or redirect the package according to the classification result. For a normal package, the system keeps the default path unchanged; for a damaged package, the FPGA output triggers the ESP8266, and the robotic arm performs the reject/sorting action. LED indicators and the VGA display are also used to show the system state and support debugging.

In the updated system, the complete workflow is therefore: OV7670 camera image capture → FPGA frame buffering → grayscale conversion → Sobel edge detection → ROI-based package detection → damage classification → FPGA digital decision output → ESP8266 servo control → robotic-arm-based physical sorting. This matches the actual implemented system shown in the updated block diagram

and replaces the earlier event-camera concept that remained in the previous draft. The original draft still contained DVS/event-camera and SNN descriptions in the sensing and processing sections, so those parts were inconsistent with the final implementation and have been corrected.

Response to Reviewer 2 Comments

1. Summary

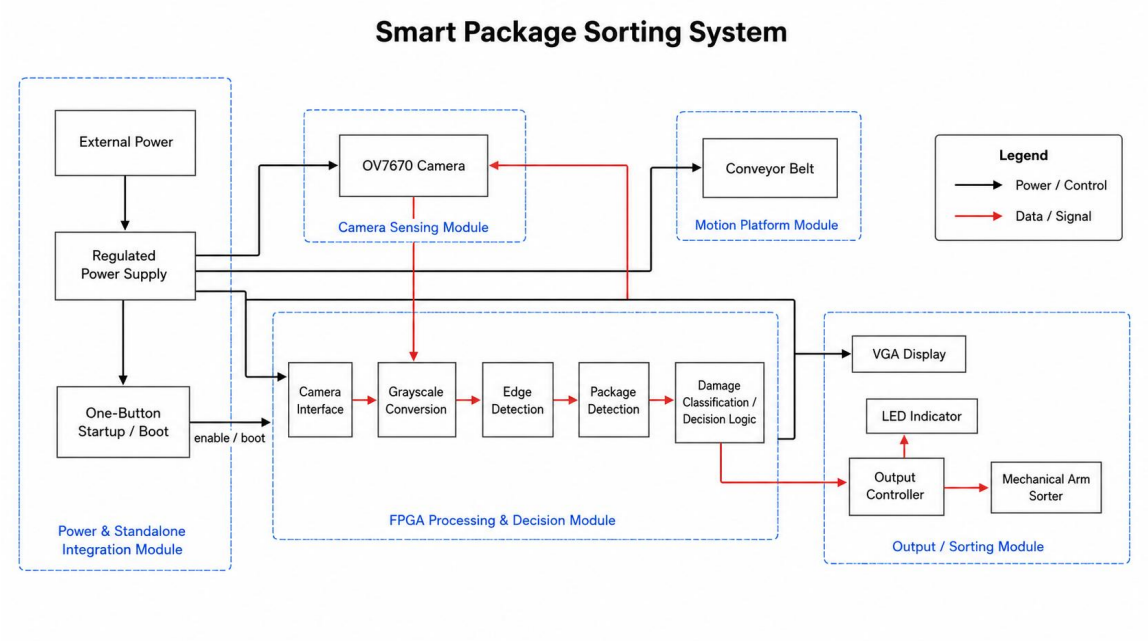
We sincerely appreciate the reviewer's critical feedback, which has led to a significantly strengthened and more rigorous presentation of the work. In response to the reviewer's comments, we have carefully revised the manuscript. Detailed responses to each comment are provided below.

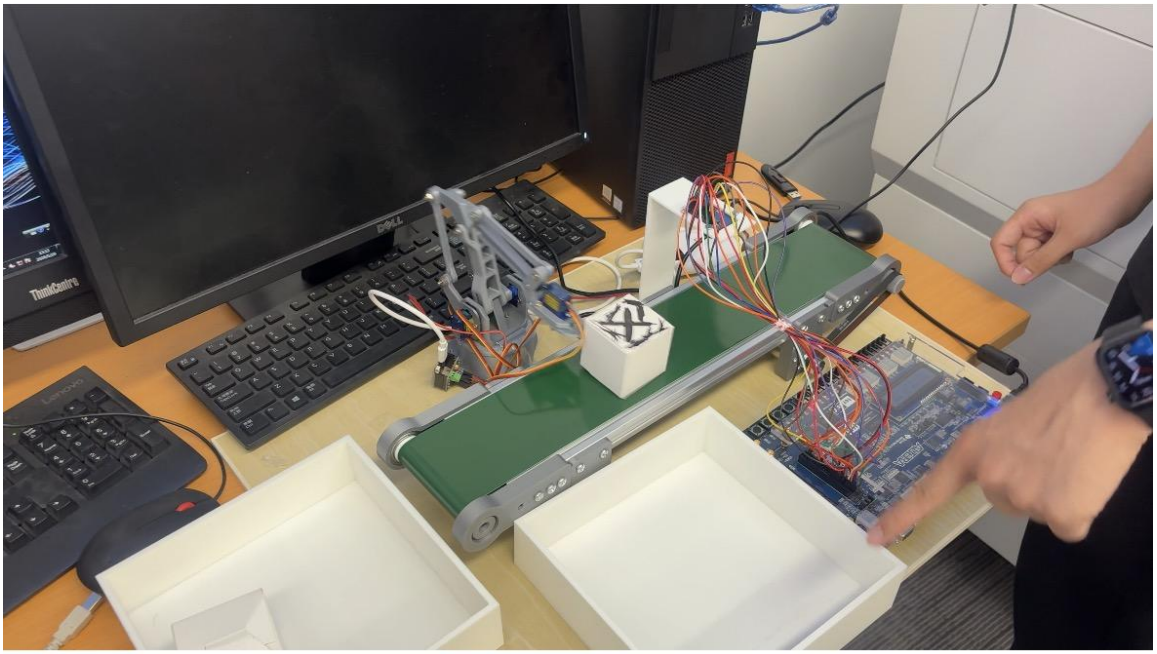
2. Questions for General Evaluation	**Reviewer's Evaluation.**
Does the introduction provide sufficient background and include all relevant references?	Yes
Is the research design appropriate?	Yes
Are the methods adequately described?	Must be improved
Are the results clearly presented?	Can be improved
Are the conclusions supported by the results?	Can be improved.
Are all figures and tables clear and well-presented?	Must be improved

3. Point-by-point response to Comments and Suggestions for Authors

Comments 1: Provide figure of completed device

Response 1: The completed device and the final block diagram are provided.





Comments 2: Provide more details about the device including mechanical design and image processing rate limits.

Response 2:

The device description has been expanded to include both the mechanical sorting structure and the image-processing rate limits. The final system is a frame-based smart package sorting device rather than an event-camera system. It uses an OV7670 camera mounted above a small desktop conveyor belt to capture images of packages as they pass through the inspection region. The conveyor has a 50 cm belt length and an 8 cm working belt width, which is sufficient for the small cardboard package samples used in the demonstration. The conveyor speed is adjustable, with an operating range of approximately 1.5 m/min to 5 m/min, and the selected working range is about 2–3 m/min to balance image stability and sorting throughput. The package samples are approximately 5 cm wide, and the camera field of view at the belt height is about 10 cm, giving the FPGA multiple frames to detect and classify each package before it reaches the sorting mechanism. These details are consistent with the mechanical platform and conveyor discussion in the report draft.

The mechanical sorting mechanism is based on a servo-driven robotic arm rather than a simple visual output. After the FPGA completes the package classification, it sends a digital decision signal to an ESP8266-based actuator controller. The ESP8266 detects the rising edge of this signal and generates PWM signals to drive the SG90 servo motors in the robotic arm. For a normal package, the package remains on the default path. For a damaged package, the robotic arm performs a predefined pushing or redirecting motion to sort the package into the reject path. LED indicators are used to show the system state, and the VGA display is used to observe the raw or processed image output during debugging. This makes the output stage a complete sensing-to-decision-to-actuation mechanism rather than only a classification display.

The image-processing rate limit has also been clarified. The OV7670 camera provides frame-based image data, and the FPGA stores the captured image in a 320×240 frame buffer, corresponding to

76,800 pixels per frame. Each pixel is stored as 12-bit RGB444 data in block RAM. The capture module reconstructs RGB444 pixels from the OV7670 8-bit data stream using `pclk`, `href`, and `vsync`, and the frame buffer design is explicitly sized for 320×240 resolution because a higher 640×480 resolution would exceed the practical embedded RAM resources on the DE2-115 Cyclone IV platform.

The FPGA processing pipeline operates on this 320×240 frame. The system first performs grayscale conversion, then applies Sobel edge detection, and finally performs ROI-based package detection and damage classification. The Sobel and classification logic process the frame through sequential memory reads and writes. At a 25 MHz FPGA processing clock, the available cycle budget is much larger than the 76,800-pixel frame size. Even allowing multiple clock cycles per pixel and row-level synchronization stalls, the processing time is on the order of several milliseconds per frame, which is below the approximately 33 ms frame interval of a 30 fps camera stream. Therefore, the main rate limits of the complete device are not the FPGA edge-detection logic itself, but the camera frame rate, frame-buffer resolution, conveyor speed, and robotic-arm response time.

The complete updated device operation is therefore: the OV7670 camera captures package images from the conveyor; the FPGA stores each frame, converts it to grayscale, extracts Sobel edges, detects package presence inside a predefined ROI, and classifies damage based on scratch-like edge features; the FPGA then outputs a digital sorting decision to the ESP8266; and the ESP8266 drives the servo robotic arm to complete the physical sorting action. This description replaces the earlier event-camera-based explanation and provides a more accurate account of the implemented device, mechanical design, and processing-rate constraints.

Comments 3: Provide more details about circuits components used to power device and control the robotic arm.

Response 3:

More details have been added about the circuit components used for power distribution and robotic arm control. The final device uses an external DC power source as the main input, and the power is distributed to different subsystems through regulated supply paths. The FPGA board, OV7670 camera, ESP8266 controller, SG90 servo motors, and conveyor motor are treated as separate electrical loads because they have different voltage and current requirements. The FPGA board is powered through its own regulated input, while the camera is connected to the FPGA/camera interface power and signal pins. The ESP8266 and the servo expansion board form the actuator control circuit. Since servo motors can draw relatively large transient current during movement, the servo power path is separated from the FPGA logic power path to reduce voltage drops and prevent unstable FPGA or ESP8266 operation. A common ground is maintained between the FPGA, ESP8266, servo driver board, and external supply so that the digital control signal from the FPGA can be correctly recognized by the actuator controller.

The robotic arm control circuit has also been clarified. The FPGA does not directly generate the servo PWM signals. Instead, after the FPGA completes package detection and damage classification, it outputs a digital decision signal to the ESP8266-based controller. The ESP8266 monitors this input signal, detects its rising edge, and then generates PWM control signals for the SG90 servo motors. In the implemented setup, the ESP8266 assigns four PWM-capable GPIO pins to the four servo motors of the robotic arm, while one input GPIO is used to receive the FPGA decision signal. The servo motors are powered from the servo supply rail through the servo expansion board, and their signal lines are driven

by the ESP8266. This design separates real-time image processing from mechanical actuation: the FPGA handles camera processing, Sobel edge detection, ROI-based classification, and decision generation, while the ESP8266 handles the timed motion sequence of the robotic arm.

The complete control path is therefore: FPGA classification output → ESP8266 input GPIO → rising-edge detection in the ESP8266 program → PWM generation → SG90 servo motion → robotic-arm sorting action. For a normal package, the arm remains in its default position and the package continues along the normal path. For a damaged package, the FPGA output triggers the ESP8266, and the ESP8266 drives the servo motors through a predefined motion sequence to push or redirect the package into the reject path. LED indication is also connected to the output stage to show system status during testing. This added explanation makes the power circuit and actuator-control circuit clearer and connects the electronic design to the physical sorting behavior. The revised description is consistent with the report sections describing the ESP8266 controller, SG90 servo motors, regulated power supply, and standalone integration module.

Comments 4: Describe quantitative criteria used to identify a damage package and give examples of verified "damaged packages".

Response 4:

In this project, generally, a "damaged package" is defined as a package that contains clearly visible, high-confidence surface damage under the controlled experimental setup. The current system is especially effective for an ideal setting where damage patterns mainly point to strong contrast and large edge variations on the surface of a white box. Typical detectable cases include thick dark crack-like markings, large artificial scratch patterns, and visible deformation such as crushed or crumpled box surfaces.

Quantitative Criterion

The quantitative criterion is based on the Sobel edge activity inside the predefined region of interest. After grayscale conversion and Sobel edge detection, the system counts how many pixels inside the ROI have an edge intensity greater than a fixed edge threshold. In the current implementation, a pixel is considered a strong edge pixel when `Sobel_edge_value` > 200. Meanwhile, the number of such pixels is accumulated as `scratch_count`. A frame is marked as containing scratch-like or damage-like features when: `scratch_count` > 900.

Therefore, the practical detection rule is:

`scratch_count` ≤ 900 → no high-confidence visible damage detected

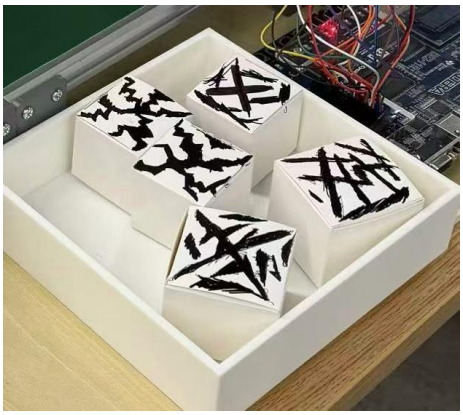
`scratch_count > 900` → high-confidence damaged package detected

This criterion works because strong dark cracks or large deformations create many high-gradient structures in the Sobel edge image. An intact white box mainly produces edges from its outer boundary, corners, and mild surface texture. In contrast, a damaged box with thick black crack-like markings produces additional internal edges across the box surface. These extra edges significantly increase the ROI edge-pixel count and allow the system to separate clearly damaged packages from intact ones.

Example Cases

One verified damaged package was created by drawing thick, dark, crack-like lines across the top surface of a white box, just as is shown in the pictures. This sample consistently produced strong Sobel responses along the artificial crack boundaries and exceeded the `scratch_count` threshold. Another verified damaged case is a visibly crushed or crumpled package. When the box surface is folded, wrinkled, or deformed, the surface introduces extra shadow boundaries, crease lines, and local edge

discontinuities. These features also increase the strong edge-pixel count inside the ROI and can trigger the damaged-package classification.

Caption: Intact white boxes v.s. Damaged boxes

However, as an ideal demo, the current design should not be interpreted as a general detector for every possible type of package damage. Just as we have mentioned before, it is most reliable for obvious, high-contrast, crack-like or deformation-based damage. Light pencil marks, shallow scratches, low-contrast surface marks, or simple decorative drawings such as a smiley face may not always trigger the damage output. This is because those patterns may produce fewer strong Sobel edge pixels, may not exceed the empirical `scratch_count` threshold, or may be too small to create enough edge activity across the ROI. In other words, the current rule detects damage only when the visual evidence is strong enough to satisfy the edge-count condition.

This limitation is mainly caused by the threshold-based nature of the algorithm. The classifier does not understand the semantic meaning of a mark; it only measures edge strength and edge quantity. Therefore, in the future, we can make improvements such as adaptive thresholding and shape-based filtering to better distinguish true damage from harmless drawings or low-contrast surface texture, adapting to more realistic scenarios.

Comments 5: Describe project successes and functionality. Provide a quantitative description of unsatisfactory results and issues. Discuss possible improvements.

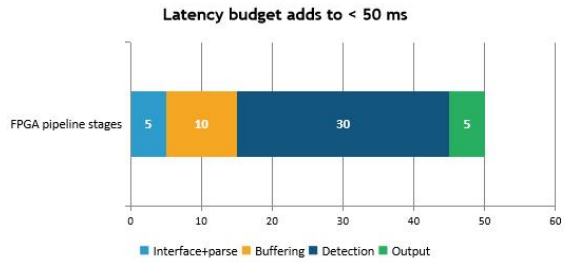
Response 5:

Successes and Functionality

In general, the project successfully implemented a real-time FPGA-based robotic package sorting system. The system consists of a conveyor belt, a fixed camera mounted above the middle section of the belt, an FPGA vision pipeline, and a robotic arm placed near the sorting area. As packages move along the conveyor belt, the camera records each box as it passes through the detection region. The FPGA processes the camera input using grayscale conversion and Sobel edge detection and then applies a threshold-based damage classification algorithm. If the package is classified as damaged, the FPGA outputs a damage signal to activate the robotic arm. The arm then pushes the damaged package into a collection bin located beside the conveyor belt. If no damage is detected, the package continues moving along the belt and reaches the normal collection area at the end of the conveyor. After our attempt, boxes with thick, dark, crack-like markings on the surface were detected reliably. Boxes that were crushed or crumpled could also be detected in some cases because of the existence of folds, shadows, and surface deformation.

The main project success is that the pipeline was demonstrated as a working system rather than only an offline image-processing algorithm. The system was able to capture moving packages, detect visible

surface damage, generate a binary output signal, and use that signal to control a physical sorting action. Under the controlled test setup, the system achieved approximately **95% classification accuracy** for the prepared test cases. The target sorting performance was above 90% correct sorting actions, meaning that most packages were sent to the intended location based on the FPGA classification result. The FPGA pipeline latency was also kept below the design budget of 50 ms, while the available mechanical reaction window was approximately 2 seconds based on the camera-to-arm distance and conveyor speed. The timing diagram below further illustrates how the detection result is generated early enough for the robotic arm to push a damaged package into the side bin before it passes the sorting point, therefore explaining why the system can support real-time robotic sorting.



The timing condition

$T_{detect} + T_{signal} + T_{arm} < d / v$

— **Camera-to-arm distance** d ≈ 10 cm.

— **Working speed** v ≈ 33–50 mm/s → window d/v ≈ 2–3 s.

— **Pipeline + arm** ≪ 1 s, so the inequality holds with large margin — even if belt speed drifts.

Unsatisfactory results and issues

However, the project still has several unsatisfactory results and limitations. First, the current definition of damage is still simplified compared with real-world package damage. In this project, “damage” mainly means strong visible edge patterns, such as dark crack-like marks, large scratches, or obvious deformation. Real packages may contain tape, printed labels, seams, barcodes, logos, wrinkles, dents, holes, or different surface materials. Some of these normal features can also create strong edges, so an intact package may sometimes produce a high edge count. On the other hand, light markings, shallow scratches, low-contrast damage, or simple patterns may not produce enough strong edge pixels to exceed the threshold. For example, a lightly drawn mark or a simple smiley-face pattern may not be detected as damage, even though it is still a visible mark on the box. This shows that the link between real-world damage and the current numerical feature, `scratch_count`, is useful but not fully robust.

Secondly, the performance depends on the conveyor speed and the relative position between the camera, the detection region, and the robotic arm. In the current setup, the conveyor belt runs at a fixed speed, and the timing between detection and sorting is manually tuned. When the belt moves too quickly, the package may spend less time inside the camera ROI, producing fewer usable frames for edge detection. This can reduce classification stability and may cause the robotic arm to react too early or too late. A tighter synchronization between conveyor speed, camera location, detection output timing, and arm actuation would make the sorting behavior more reliable.

Third, the current robotic arm output is mechanically simple. At this stage, the arm mainly pushes the damaged package off the main conveyor path into a side bin. This is sufficient to demonstrate binary sorting, but it is not yet a complete industrial pick-and-place system. A more advanced version could use additional servos to control arm angle, rotation, lifting motion, and placement trajectory. Instead of simply knocking the package sideways, the arm could pick up the damaged package and place it more carefully into a separate collection container. This would reduce mechanical disturbance and make the system more suitable for fragile or irregular packages.

Possible improvements

Possible improvements revolve around solutions to the unsatisfactory results.

First, the visual damage detection algorithm can be made more robust. The current system mainly relies on Sobel edge-pixel count, using fixed thresholds such as `EDGE_THRESHOLD = 200` and `SCRATCH_THRESHOLD = 900`. Therefore, to enhance the robustness of performance as well as fitting more realistic scenarios, future work should first combine edge-pixel count with additional features, such as crack length, edge density, contour irregularity, hole or dent shape detection, and package-area normalization. Then, the thresholding method could also be made adaptive based on lighting condition, background brightness, or average edge activity. This would help reduce false positives from normal package details and false negatives from weak but real damage.

Secondly, the timing relationship between the conveyor belt, camera, FPGA output, and robotic arm should be calibrated more precisely. In the current setup, when the conveyor moves too fast, the package may stay inside the ROI for fewer frames, which can reduce classification stability. It may also cause the arm to react slightly too early or too late. A future improvement would be to add an encoder or speed sensor to measure the belt speed in real time. The FPGA could then calculate the expected arrival time of the package at the robotic arm based on the measured speed and the known camera-to-arm distance, and schedule the sorting signal more accurately. For example, once the FPGA confirms that a damaged package has left the ROI, it could start a programmable delay counter before activating the robotic arm. This delay could be adjusted according to conveyor speed, package length, and the physical distance between the detection region and the arm. Another possible improvement is to add an optical sensor near the robotic arm to confirm that the package has reached the sorting position before the arm moves. This would make the sorting action less dependent on manually tuned timing and more robust to small changes in belt speed or package placement. The system could also store several consecutive frame results and only trigger the arm after a stable damage decision is made, reducing false sorting actions caused by a single unstable frame.

Third, the robotic arm mechanism could be improved. Currently, the arm mainly performs a simple pushing action to move damaged packages into the side bin. This is enough to demonstrate binary sorting, but it is mechanically limited. For packages with different sizes, weights, or surface friction, a simple push may not always be stable. Since additional servos are available, a future version could use a more controlled multi-servo motion, such as rotating, lowering, lifting, and placing the package into the damaged-package bin. This would make the system closer to a real pick-and-place sorting mechanism rather than only a rejector.

Finally, the test set should be expanded to include more realistic package conditions. The current test samples are mainly controlled white boxes with dark crack-like markings or visible deformation. Future testing should include boxes with tape, labels, printed logos, brown cardboard, holes, dents, crushed corners, different lighting conditions, and different conveyor speeds. This would allow the performance to be reported by damage type instead of only using one overall accuracy number, making the evaluation more meaningful for real-world package sorting.