Results in Engineering 30 (2026) 110030

Contents lists available at ScienceDirect

Results in Engineering

journal homepage: www.sciencedirect.com/journal/results-in-engineering

Research paper

Software architecture for real-time hyperspectral analysis in material sorting systems

Adrián Sarrías, Miguel O. Martínez-Rach, Otoniel López-Granado, Héctor Migallón *

Computer Engineering Department, Miguel Hernández University, Elche, Spain

ARTICLE INFO

Keywords:
Industry 4.0
Parallel computing
Software architecture
Hyperspectral imaging
Real-time processing
Material sorting
Environmental. sustainability.

ABSTRACT

The integration of hyperspectral imaging into industrial sorting systems has enabled high-precision classification of materials with similar visual characteristics but different chemical compositions. However, the real-time processing demands of HSI data acquisition, characterised by high spectral and spatial resolution, require advanced computational strategies. This paper presents a scalable and efficient software architecture designed for real-time hyperspectral analysis in automated material sorting lines. The architecture exploits heterogeneous and homogeneous parallelism to distribute pre-processing, classification and segmentation tasks across multiple threads and processing cores. Two classification methods, based on Spectral Angle Mapper and Artificial Neural Networks, are developed and evaluated, both show high accuracy in material identification, but they impact system scalability in different ways. Extensive performance tests show that the proposed framework meets strict timing constraints and maintains low-latency operation on standard multi-core CPU systems. The modular design of the system ensures adaptability to different hardware configurations and material types, supporting future scalability and integration into diverse industrial environments. The real-time constraint imposed by the camera's maximum frame rate is $1.493ms$. Thanks to the optimisations applied, the critical processes, pre-processing and classification, have been reduced to just over $30\mu s$ each, consuming only about 5% of the available time and leaving almost 95% free for additional operations or performance enhancements. This results in a system that is scalable both from a computational perspective and in terms of increasing the overall performance of the industrial plant.

1. Introduction and motivation

Currently, there is a growing concern in different productive sectors about environmental sustainability, which is driving the shift towards what is known as the circular economy. At the heart of this transformation lies the problem of waste accumulation from many types of materials, such as plastics and textiles, presenting a challenge that demands innovative solutions for effective waste recovery and reuse.

Beyond material sorting, hyperspectral imaging has been employed in diverse environmental and sustainability-related applications, including air pollution monitoring [1], soil microplastic detection [2], harmful algal bloom mapping [3], smart city sustainability initiatives [4], as well as precision agriculture, medical diagnostics, and food quality assessment [5], demonstrating its versatility across multiple real-world domains.

Sorting waste is a fundamental step before disposal, as it enables the recovery of high-value, high-purity secondary raw materials through recycling. A variety of technologies are employed at this stage, including magnetic and air separation, screening, X-rays, induction systems, thermal imaging, and vision-based methods.

Recent studies have explored the use of digital image-based classification for municipal waste [6–12]. However, these methods are often limited when objects of similar appearance differ in chemical composition. Even advanced AI models require retraining for new materials, formats or textures, and are unable to distinguish spectrally similar objects when based on the visible spectrum alone.

Sorting waste by chemical composition requires more information than traditional vision systems can provide. Hyperspectral imaging (HSI), which combines digital imaging and spectroscopy [13], captures the spectral signature of each pixel across different wavelength regions (e.g., visible, near-infrared, shortwave infrared). This capability enables highly reliable material recognition, even for objects with similar visual appearance but different chemical composition. HSI systems can now be integrated into industrial sorting lines, where robots automate the separation process to ensure that chemically equivalent materials are grouped together.

* Corresponding author.
E-mail address: hmigallon@umh.es (H. Migallón).

https://doi.org/10.1016/j.rineng.2026.110030
Received 12 January 2026; Received in revised form 10 March 2026; Accepted 11 March 2026
Available online 13 March 2026

Today, industrial-grade hyperspectral sensors operate at millimetre spatial resolution and are applied in fields such as biomedicine [14], food quality [15], agriculture [16,17], cultural heritage [18] and plastics classification [19,20]. Besides, the use of HSI technology in the waste management sector is growing rapidly like in construction and demolition waste [21,22], electrical and electronic equipment waste [23], municipal solid waste, etc.

Therefore, HSI technology enables reliable classification of visually similar materials and can be integrated into robotic sorting lines, enhancing accuracy, automation, and overall efficiency. HSI systems typically use line-scan cameras, which capture one spatial line at a time while the object moves along the conveyor belt. In such line-scan configurations, the processing pipeline is constrained by a strict per-line time budget, since each acquired line must be processed before the next one is captured in order to avoid backlog or loss of real-time operation. These cameras are characterised by their spectral resolution (number of bands per pixel) and their line-scan spatial resolution. Their performance depends on both the optical setup and the installation geometry, which must be carefully adjusted to ensure square pixels by synchronising the camera's acquisition rate with the conveyor belt speed.

In this work, we focus on near-infrared (NIR) HSI sensors and their integration into robotic sorting systems. An efficient and scalable software architecture has been proposed for real-time industrial material classification based on HSI. The ultimate goal is to design a fully versatile and scalable industrial system, ensuring that both the physical installation and the software components can be easily adapted and expanded to meet varying operational requirements and future scaling needs. For example, our prototype system employs robotic arms equipped with vacuum grippers for item handling, but the developed software generates precise picking coordinates for each object, enabling integration with various end-effectors such as air jets, alternative robotic grippers, or other custom handling mechanisms.

Hyperspectral data processing remains a major obstacle to the practical and industrial deployment of hyperspectral imaging systems, particularly in real-time or resource-limited settings [5]. As highlighted in [5], the substantial quantity of high-dimensional spectral data necessitates robust computational infrastructure, since hyperspectral images possess tremendous dimensionality and require considerable storage, memory, and computing capabilities. These characteristics lead to increased operational complexity and integration difficulties within industrial pipelines, often requiring advanced computational strategies to ensure feasibility under strict timing constraints.

Designing a modular software framework is challenging due to the heterogeneity of the processing steps or pipelines. Pre-processing, detection and classification algorithms vary depending on material properties, acquisition conditions and performance constraints. Some pipelines require radiometric calibration, while others use alternative normalisation techniques. Dimensionality reduction methods (e.g. Principal Component Analysis (PCA)) or full-spectrum classification may be used. AI-based models such as Supported Vector Machines (SVM), Multi-Layer Perceptron (MLP) or Convolutional Neural Networks (CNN) may also be used. In addition, different industrial installations may use different types of sensors, conveyor systems, lighting setups, communication protocols, and extraction mechanisms. The developed framework must be adaptable to the specific configuration of each pipeline.

The computational burden associated with high-dimensional hyperspectral data under strict real-time constraints has been explicitly quantified in demanding industrial scenarios. Kristensen et al. reported hyperspectral acquisition at 160 fps using a configuration of 824 spatial pixels and 600 spectral bands per frame, corresponding to approximately 79.1×10^6 spectral values per second [24]. Their results show that, under such data rates, data processing rapidly becomes the limiting factor in real-time operation when relying on CPU-based implementations. Furthermore, as computational load increases with spectral dimensionality and acquisition rate, performance scalability progressively degrades unless explicitly parallel and carefully structured processing strategies are implemented.

This observation highlights that the critical parameter for industrial scalability is not merely conveyor belt speed, but the volume of hyperspectral data generated per unit time. Commercial sorting systems from manufacturers such as PICVISA, STEINERT, and TOMRA report conveyor velocities ranging from approximately $1 - 2m/s$ in compact or specialised configurations to around $4.5m/s$ and up to $6m/s$ in high-throughput installations. However, belt speed alone does not determine computational demand, since spatial sampling density, spectral dimensionality, and acquisition frequency jointly define the total hyperspectral data rate imposed on the processing pipeline.

In our prototype configuration (640 spatial pixels and 150 spectral bands at 670 fps), the resulting throughput is approximately 64.3×10^6 spectral values per second, which is of the same order of magnitude as the industrial scenario reported in [24]. In fact, an increase to roughly 185 spectral bands under our acquisition conditions would yield an equivalent data rate of approximately 79.3×10^6 spectral values per second.

Examples of commercially available hyperspectral cameras further illustrate this trend in terms of data throughput. For instance, the XIMEA MX022HG-IM-SM5X5-NIR2-FL, featuring a 2048×1088 pixel sensor operating at 338 frames per second in a 5×5 mosaic snapshot configuration, produces 2.23×10^6 pixel-level spectral measurements per frame, corresponding to approximately 753×10^6 raw spectral measurements per second. Likewise, the ULTRIS 5 HFR system (290×250 spatial pixels, 51 spectral bands, $75Hz$) generates 3.69×10^6 spectral samples per frame and 277×10^6 spectral samples per second.

Moreover, recent comparative studies have reported industrial dual-camera configurations combining CMOS and InGaAs sensors to cover extended spectral ranges (e.g., 450–1834 nm) [25]. In such systems, two independent hyperspectral data streams are acquired simultaneously, each with its own spatial and spectral dimensionality, effectively increasing the total number of spectral measurements to be processed per unit time. This multiplicative growth in data volume further intensifies the computational requirements of the processing pipeline.

More generally, whether high-frame-rate single-sensor systems or multi-camera configurations are employed, industrial scalability is increasingly governed by the total volume of hyperspectral data generated per unit time. Under CPU-based implementations, processing latency tends to scale with spectral dimensionality and acquisition rate, eventually approaching or exceeding the available inter-frame processing window in high-throughput scenarios. Consequently, architectural efficiency and explicit parallelisation become essential to guarantee deterministic real-time performance as data rates increase.

The rapid growth in hyperspectral sensor throughput requires equally capable data transmission infrastructures. High-bandwidth industrial interfaces such as USB3 Vision, 10, 25 and 100 GigE Vision, as well as CoaXPress 2.0, are increasingly adopted to accommodate the rapidly rising data rates generated by modern hyperspectral sensors. This progressive evolution of communication standards within the industrial vision ecosystem reflects the need to prevent the communication layer from becoming a system bottleneck. Similarly, adequate architectural solutions must be provided to ensure that data processing itself does not become the limiting factor in real-time industrial operation.

To address the challenges of real-time material classification, we propose a software architecture that distributes the main processing stages of the pipeline across independent threads through heterogeneous parallelisation. Subsequently, homogeneous parallelisation techniques are applied using high-performance computing strategies on multicore CPU systems, specifically targeting those processing tasks that could impact real-time performance or limit the system scalability.

The rest of the paper is structured as follows: In Section 2, we describe the industrial hardware system developed for this prototype. Section 3 highlights the most important industrial settings related to hyperspectral data acquisition. Section 4 outlines the proposed architecture by

Fig. 1. Prototype production line.

reviewing the most important pipeline steps in the corresponding subsections. Sections 5 and 6 shows the software optimisations performed over the different processing steps of the pipeline and the parallelisation performance, respectively. In Section 7, we discuss the results obtained as well as the main findings found during the development and evaluation of the proposed software architecture for the developed prototype. Finally, in Section 8 some conclusions and future lines are drawn.

2. Industrial hardware system

This section describes the hardware implemented in our real-time hyperspectral sorting prototype, including the materials and settings used for validation. The system, shown in Fig. 1, consists of four main subsystems: (a) material transport, (b) HSI acquisition, (c) sorting and retrieval, and (d) computing and communication.

The transport subsystem consists of a conveyor belt that moves objects through the HSI's field of view. It is managed by a programmable logic controller (PLC) which also controls other hardware systems such as the lighting, cooling, and vacuum subsystems, along with the robot control signals. Conveyor belt speed is a critical factor, as it must be high enough to ensure industrial viability but not so fast as to cause object displacement or classification errors. Maintaining a uniform and clean conveyor belt surface is strongly recommended, as dirt or debris can alter the spectral response of the background and lead to significant misclassification errors. Furthermore, a contaminated conveyor belt surface may require additional pre-processing steps to correct spectral distortions, increasing computational costs.

At the heart of the HSI acquisition system is a Specim FX17 camera, which provides 640 spatial pixels and 224 spectral bands in the $900 - 1700nm$ range. System configuration, including camera height, optics and thermal conditioning, has a direct impact on spatial resolution and pixel size. The camera operates in linear scan mode and is externally triggered by encoder pulses synchronized with the movement of the conveyor belt to ensure that square pixels are obtained and spatial distortions are avoided. The maximum frame rate of the camera and the size of each pixel determine the maximum achievable conveyor belt speed, with the constraint that each scanned line must be fully processed before the next is captured.

The illumination is provided by low-cost halogen lamps, which cover both the visible and the NIR spectra. Although designed for industrial applications, hyperspectral systems remain sensitive to both radiated and mechanical noise. To mitigate radiated interference, it is essential to avoid external contaminating light sources, especially those of fluctuating intensity. However, it does not require full encapsulation, as is the case in commercial solutions such as PICVISA Ecopick[1]. Computational methods are used to deal with any harmful effects.

Furthermore, mechanical vibrations from conveyors or robotic arms have been demonstrated to have a detrimental effect on image quality, particularly in line-scanning systems. In order to reduce this effect, the camera is mounted on vibration-damping supports, and the cooling fans are isolated in order to prevent mechanical coupling.

The sorting subsystem uses a Kawasaki RS007L robot[2] equipped with suction grippers to retrieve the classified objects. For the purpose of validation, the system was configured to classify and select four types of plastic polymers: Acrylonitrile Butadiene Styrene (ABS), Polypropylene (PP), High-Density Polyethylene (HDPE), and Polystyrene (PS), each represented by a different colour. The robot is programmed to place sorted objects into designated hoppers.

The computing subsystem is based on a conventional multi-core shared memory architecture. All processing stages such as acquisition, pre-processing, classification, segmentation and communication are performed on this workstation. Communication is managed via two interfaces: standard Ethernet for TCP/IP sockets with the PLC and the robot, and a Camera Link connection[3] with a dedicated frame grabber for the high-throughput camera data. Further details are provided in Section 5.

3. Settings for hyperspectral acquisition

A hyperspectral image is a three-dimensional structure, called hypercube, with two spatial dimensions (width and height) and one spectral dimension (wavelength). Unlike RGB images, which contain only three spectral bands, hyperspectral images capture typically hundreds of bands, each corresponding to a specific wavelength. The detailed spectral information per pixel enables accurate identification of the material at each pixel.

As said, in our prototype, we use a Specim FX17 camera, which provides 224 spectral bands in the $900 - 1700nm$ range with 640 spatial pixels and a spectral resolution of $8nm$. The camera operates in pushbroom, or line-scan mode with a 38° field of view (FOV). The pixel size along the axis of motion depends on both the acquisition frequency and the conveyor belt speed. To ensure square pixels and spatial fidelity, synchronisation between the camera and the conveyor belt is achieved by an external trigger based on pulses from the conveyor belt encoder.

The pixel width, or ground sampling pixel (GSP), is defined by the Eq. (1), where h is the camera height in mm, α is the lens FOV in degrees, and p is the number of pixels per line. In our case, the camera is installed at a height of $1004mm$ above the conveyor belt, so $GSP = 1.08mm$. Therefore the encoder must trigger a new captured line of pixels every $1.08mm$ of conveyor belt movement. To capture one metre of conveyor belt movement, approximately 926 camera lines must be acquired, which results in a hypercube of dimensions $640 \times 926 \times 224$. Assuming 16 bits per spectral value, the raw data size for this hypercube would be approximately $253MB$.

$$GSP = \frac{2 \cdot h \cdot \tan(\alpha/2)}{p} \tag{1}$$

[1] https://picvisa.com/ecopick-robot-inteligencia-artificial-clasificacion-materiales-residuos/

[2] https://kawasakirobotics.com/products-robots/rs007l/

[3] https://www.automate.org/vision/vision-standards/vision-standards-camera-link

Conveyor belt speed is typically measured in meters per minute (m/min). To optimise throughput, we need to relate the camera parameters to the conveyor belt speed. The conveyor belt speed v in mm/s is given by the Eq. (2) where r is the drum radius (including conveyor belt thickness), p_s are the encoder pulses per second and e_r is the encoder resolution.

$$v = \frac{2 \cdot \pi \cdot r \cdot p_s}{e_r} \tag{2}$$

The resulting frame rate (in lines per second) is shown in Eq. (3)

$$fps = \frac{v}{GSP} \tag{3}$$

For example, with $r = 140mm$ and $e_r = 500$, a conveyor belt speed of $25.01m/min$ gives a $p_s = 237$, which gives us 385.69 fps. Given that the FX17's maximum frame rate is $670fps$, that corresponds to a camera frame rate of 57.6%. At this rate, $2.591ms$ are available per line to perform all the processing steps. However, at full camera speed ($670fps$), the available time is reduced just to $1.493ms$, giving a maximum conveyor belt speed of $43.43m/min$.

Proper illumination is essential to obtain reliable reflectance spectra. The FX17 camera has 12−bit depth, i.e. supports up to 4096 levels of intensity. Illumination must be uniform across the spectral range, strong enough to avoid underexposure, but not so strong as to saturate the sensor.

In the NIR range, halogen lamps are typically used due to their broad spectral emission. Although NIR LEDs are available, they remain expensive and are not essential for the system's functionality, although they could offer advantages in terms of energy efficiency. In our setup, the illumination was experimentally configured by adjusting lamp height and arrangement until suitable spectra were achieved. A practical estimate of the required power P is given by the Eq. 4, where d is the distance from the light source to the target or object (in metres), t is the sensor integration time (in seconds) and a is an empirical constant. For a frame rate of 385.69 fps, a sensor integration time $t \approx 0.65ms$ and a distance d established for an optimum visual coverage, the estimated illumination power consumption is around $3126W$, distributed across 8 halogen lamps in two rows along the $800mm$ of the conveyor belt width.

$$P = a \cdot \frac{d^2}{t} \tag{4}$$

To avoid saturation, Specim recommends that white references remain below 75% of the sensor's full scale, where higher values correspond to white and lower values to black. Adjusting d or t allows a balance to be struck between light intensity and frame rate. For example, reducing the lamp distance by $10cm$ reduces the required power to $1891W$; alternatively, maintaining $3280W$ would allow operation at $670fps$.

As shown, a combination of optical, mechanical and electronic parameters must be finely tuned to ensure reliable spectral acquisition. These conditions directly affect classification performance and the computational cost of correcting, normalising or rejecting invalid data, making acquisition quality a key factor in system efficiency and scalability.

4. Proposed software framework architecture

In this section we describe the design of the application structure as well as the optimisations performed to achieve with the real-time requirements of a real system designed for a waste recycling company.

Qt [26] was chosen as the main development framework for this project because of its cross-platform capabilities, allowing the application to run seamlessly on Windows, Linux and MacOS without major code changes. This is particularly valuable in industrial environments where system requirements can vary depending on the deployment context.

In addition, Qt provides robust tools for building professional and responsive graphical user interfaces (GUIs). Its high-level widgets, support for modern UI paradigms and integrated resource management system facilitate the development of intuitive, maintainable and visually consistent interfaces.

Compared to other development frameworks such as Microsoft .NET (which is primarily Windows-oriented) or JavaFX (which often lacks the native look-and-feel and performance required for real-time applications), Qt offers a more flexible and powerful solution for GUI-intensive, cross-platform software. Qt also integrates seamlessly with $C++$ codebases, allowing efficient implementation of performance-critical components and direct access to hardware-level functionality.

Another key advantage of Qt is its signal slot mechanism, a core feature of its event-driven architecture. This system provides a clean and efficient way to manage communication between components or threads, particularly in GUI applications where user interactions must trigger updates across different modules. Unlike traditional callback-based approaches, Qt's signal-slot model provides better decoupling between objects, improving the modularity, maintainability and testability of the codebase.

The Specim FX17 supports connection to the computer system via either Camera Link or Gigabit Ethernet. In this case, the fastest possible communication interface is required, so Camera Link was chosen. This interface requires a dedicated frame grabber, which is typically supported by the Windows operating systems used in this project. As mentioned earlier, the software architecture is designed to be easily portable to other operating systems if required.

In Fig. 2 we show the different processes/threads involved in the application. As can be seen, the application is mainly composed of four threads, namely *main*, *camera*, *segment* and *sender*. The *main* thread is responsible for creating all threads and global objects, as well as managing the GUI. Currently the GUI displays the raw data received from the hyperspectral camera, the classification data (ABS, PP, HDPE and PS) and the segmentation results (object detection). When the Qt Application object is created and started, it sends a *startCamera* signal to the *camera* thread, which starts capturing lines from the spectral camera triggered by the encoder attached to the conveyor belt. Each camera line received by the *camera* thread is then pre-processed and classified according to the different materials. Each classified line is then stored in a memory buffer or window to perform the segmentation process.

Each acquired line is transmitted by the hyperspectral camera via the Camera Link interface and transferred by the frame grabber into host memory through the corresponding device driver. This acquisition buffer follows a single-writer design at the hardware level, where incoming data are written exclusively by the frame grabber without software-side contention.

The *camera* thread continuously monitors the arrival of new lines and processes them sequentially. To verify temporal continuity and detect potential losses, an internal software counter is maintained and compared against the capture identifier provided by the camera for each acquired line. Any discrepancy between the expected and received identifiers is interpreted as a missing or skipped line. As soon as a new valid line is available, the *camera* thread reads the data from the acquisition buffer and immediately performs preprocessing and classification within the same thread context.

To avoid interference between acquisition and processing stages, the first operation performed by the *camera* thread consists of copying the acquired line into a private working memory region. All subsequent operations are carried out on this copied data. This strategy guarantees that classification is performed on a stable snapshot of the acquired line while the frame grabber remains free to write incoming data without blocking or contention.

The classification output of each processed line consists of 640 bytes, corresponding to the spatial resolution of the hyperspectral sensor. Although the graphical interface represents a window of 1600 lines (configurable at application startup), the actual allocated memory for classified data is twice this size. Specifically, a linear buffer of 2×1600 lines is reserved.

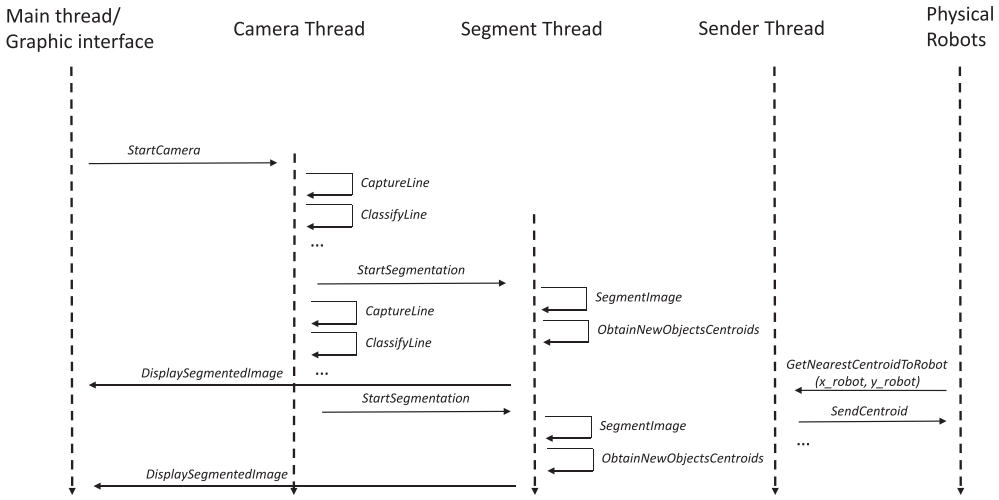


Fig. 2. Application thread architecture.

Each classified line is written twice in memory with an offset (stride) equal to the window length. This duplicated storage strategy guarantees that, at any time, the 1600-line visualization window corresponds to a physically contiguous memory region. As a result, both GUI rendering and segmentation operations can access the active window without requiring modular indexing or wrap-around logic. This approach replaces a traditional circular buffer, which would introduce non-contiguous memory regions when the write pointer wraps around. By ensuring spatial continuity in memory, segmentation and visualization always operate on stable and linearly ordered data, simplifying memory access patterns and improving cache locality.

To enhance robustness under real industrial conditions, a line integrity control mechanism is implemented. If an acquired line is not processed within the expected temporal window, the classification result of the previous valid line is replicated to preserve spatial continuity. As said, the system maintains counters to detect consecutive line losses as well as the percentage of lost lines within each segmentation window. If the number of consecutive missing lines exceeds a predefined threshold, or if the proportion of lost lines within a window surpasses a configurable limit, an alarm trigger is generated and sent to the *main* (GUI) thread via a Qt signal. This mechanism prevents undetected degradation of classification quality and ensures reliable operation under strict temporal constraints.

The active visualization window corresponds to the previously described 1600-line classification window (configurable at application startup), which in the developed industrial prototype represents approximately 1728 mm of conveyor belt displacement. Once this window has been fully classified, a *startSegmentation* signal is issued to the *segment* thread. Thereafter, segmentation is triggered every 400 newly classified lines, enabling progressive object detection while acquisition and classification continue uninterrupted.

Each segmentation step produces the centroids of the detected material objects. Since consecutive segmentation windows may partially overlap in space, a validation procedure is applied to determine whether newly computed centroids correspond to previously detected objects or represent new instances. Specifically, each newly detected centroid is compared against the existing global list of objects, and it is considered already detected if an exact spatial match is found or if the displacement with respect to a previously stored centroid remains within a predefined tolerance threshold. Valid centroids are inserted into a global list containing the objects identified by the segmentation stage. This list represents the set of detected objects that are currently available for robotic manipulation. When the *sender* thread assigns an object to a robotic arm and issues the corresponding pick command, the associated centroid is removed from the list. In addition, objects that are no longer reachable due to conveyor displacement are discarded according to an ageing criterion, preventing the persistence of obsolete detections.

The *sender* thread responds to work requests issued by the robotic arms once they complete their previous pick operation. Upon receiving a new request, the *sender* thread selects the most appropriate valid centroid according to the robot's current position and dispatches the corresponding pick command. In parallel, the *segment* thread communicates the segmented image to the *main* thread via a Qt communicator object for real-time visualization in the graphical interface.

It would not be possible to execute these processing stages sequentially and satisfy the strict temporal constraints imposed by the hyperspectral camera acquisition frequency. The separation into independent threads constitutes a first level of parallelisation, distributing heterogeneous tasks such as acquisition, classification, segmentation, and communication across distinct execution contexts. This architectural decomposition isolates time-critical operations from higher-latency stages and prevents blocking interactions between modules.

When the computational load of a given stage approaches the available temporal budget, further acceleration is achieved through homogeneous parallelisation, distributing the internal workload of that stage across multiple processing cores. This hierarchical parallelisation strategy ensures scalability as acquisition frequency or spectral dimensionality increases.

Apart from the above threads, a Robot Operating System[4] (ROS) thread is used to communicate with the Programmable Logic Controller (PLC), which controls the machine's conveyor belt, camera lighting, robot operation signals, and industrial vacuum system. The computational cost of this thread is negligible, as it primarily handles communication events generated by the GUI and receives alerts from the PLC.

While Fig. 2 depicts the functional decomposition of the application into independent execution threads, the underlying concurrency model follows a single-writer/multi-reader strategy with clearly defined ownership of shared resources. Structural modifications of shared data structures are restricted to dedicated threads, while other threads perform controlled read-only access or state marking operations. This design prevents race conditions and iterator invalidation while preserving deterministic real-time behaviour. Event triggering between threads (e.g., segmentation activation or GUI refresh) relies on controlled state updates and Qt's asynchronous signal-slot mechanism, ensuring that non-critical operations never interfere with time-constrained acquisition and classification stages.

[4] https://www.ros.org/

4.1. Camera thread

This thread is responsible for camera line acquisition, pre-process and classification. These operations must be completed before the arrival of the next line, due to the real-time constraints of the system. Since this thread is the only one that may become a bottleneck with a significant impact on overall system performance, it is essential to analyse both computational complexity and functionality. Ensuring its efficiency is a key requirement for achieving system scalability, especially when targeting higher acquisition rates, increased resolution, or more complex classification tasks.

As previously discussed, the dominant computational load of the application corresponds to spectral pre-processing and material classification. Consequently, the *camera* thread becomes the primary target for homogeneous parallelisation. Within this thread, OpenMP parallel regions are spawned to distribute spectral computations across multiple cores while preserving deterministic acquisition timing.

Considering the software architecture of the proposed system, and the characteristics of the camera, the data acquired in each capture (640 spatial pixels $\times$ 224 spectral bands) must be pre-processed and classified within a time window of 1.493 milliseconds at maximum camera frequency (670 fps). Our objective is to ensure system scalability, meaning the ability to increase the number of classifiable materials and to operate with cameras featuring higher acquisition rates and/or greater spatial resolution. To support this, it is crucial that the total time required for pre-processing and classification is not only below 1.493 milliseconds, but also minimised as much as possible.

4.1.1. Pre-process

Spectral pre-processing is essential in industrial hyperspectral applications to enhance data quality before classification. Several techniques exist to address issues such as baseline shifts, scattering effects, and spectral noise. Widely used methods include Reflectance calibration, Standard normal variate, Multiplicative scatter correction, min-max scaling, mean centring, Area under the curve, single-band normalisation, first and second derivatives, Savitzky-Golay filtering, and Total variation unmasking [27–31]. These are common in domains such as food quality control, pharmaceutical inspection, and plastic waste sorting.

In industrial environments, where surface conditions and lighting are not well defined or calibrated, the selection of an appropriate pre-processing chain is critical. While complex pre-processing chains can improve classification, they also increase the computational cost. In the developed prototype, we applied a compact but efficient pre-processing chain consisting of two tasks optimised for real-time execution, both based on normalisation. The proposed modular software architecture allows easy replacement of pre-processing steps if necessary.

As said, the pre-processing chain implemented is based on two normalisation stages. The first step is a white/black reference correction, also known as reflectance calibration, which compensates for sensor and illumination variations. For each pixel (i, j) and spectral band λ, the reflectance $R_{i,j,\lambda}$ is computed using Eq. (5), where $I_{i,j,\lambda}$ represents the raw pixel value, $D_{i,j,\lambda}$ the dark reference, and $W_{i,j,\lambda}$ the white reference. The raw 16-bit integer values are converted to floating-point format to preserve precision during the normalisation operations.

$$R_{i,j,\lambda} = \frac{I_{i,j,\lambda} - D_{i,j,\lambda}}{W_{i,j,\lambda} - D_{i,j,\lambda}} \tag{5}$$

A min-max normalisation is then applied to scale values to $[0, 1]$ per band, as shown in Eq. (6) where $\hat{R}_{i,j,\lambda}$ the normalised reflectance, $R_{i,j,\lambda}$ is the reflectance value at pixel (i, j) and band λ, $R_{\lambda}^{\min}$ and $R_{\lambda}^{\max}$ are the minimum and maximum reflectance values observed in band λ, respectively. Values are finally scaled to $[0, 1000]$ and cast back to 16-bit integers. This full process, outlined in Algorithm 1, is equally applied for the two classification methods tested.

$$\hat{R}_{i,j,\lambda} = \frac{R_{i,j,\lambda} - R_{\lambda}^{\min}}{R_{\lambda}^{\max} - R_{\lambda}^{\min}} \tag{6}$$

Algorithm 1 Double Normalisation Combining White/Black References and Pixel-Level Rescaling.

Input: Raw pixel spectrum, dark reference and white reference
Output: Normalised pixel spectrum (in-place), maximum and normalised values
for $i = 0$ to `spectralSize` $- 1$ **do**
    `dividend` $\leftarrow$ `Spectrum`$[i]$ $-$ `darkSpectrum`$[i]$
    `divisor` $\leftarrow$ `whiteSpectrum`$[i]$ $-$ `darkSpectrum`$[i]$
    `normalized_value`$[i]$ $\leftarrow$ `dividend/divisor`
    if `normalized_value`$[i]$ $>$ `maxSpectrum` **then**
        `maxSpectrum` $\leftarrow$ `normalized_value`$[i]$
    end if
    if `normalized_value`$[i]$ $<$ `minSpectrum` **then**
        `minSpectrum` $\leftarrow$ `normalized_value`$[i]$
    end if
end for
`divisor` $\leftarrow$ `maxSpectrum` $-$ `minSpectrum`
for $i = 0$ to `SpectralSize` $- 1$ **do**
    `dividend` $\leftarrow$ `normalized_value`$[i]$ $-$ `minSpectrum`
    `Spectrum`$[i]$ $\leftarrow$ `uint16_t`$((\text{dividend}/\text{divisor}) \times 1000)$
end for

On the one hand, the white reference acquisition consisted of capturing 200 lines of a uniform white cardboard surface across the width of the conveyor belt. On the other hand, the black reference acquisition was done by capturing 200 lines with the camera shutter closed. Instead of global averaging, a per-pixel reference has been calculated, effectively capturing the $W_{i,j,\lambda}$ and $D_{i,j,\lambda}$ matrices, which account for the effects of non-uniform illumination and lens distortion. This approach improves the consistency and accuracy of spectral data, especially in real industrial environments where perfectly uniform illumination is difficult to achieve.

As will be discussed later, we implement two classification methods based on the Spectral Angle Mapper (SAM) [32] and a feed-forward Artificial Neural Network (ANN). Both share a common spectral preprocessing, but with a slight difference: The SAM-based method includes a heuristic pre-filter to discard pixels belonging to the conveyor belt, while the ANN-based method treats the conveyor belt as one of the classification categories. In particular, when using the SAM-based classification method, a preliminary filter is applied to exclude pixels with low reflectance that are likely to belong to the conveyor belt. This heuristic threshold is defined in the application configuration parameters and calculated per pixel as shown in Algorithm 2. This preliminary filtering helps to reduce the computational load by avoiding unnecessary comparisons. This step is disabled in the ANN-based classification method, which treats all pixels equally.

Algorithm 2 Heuristic Decision Rule to Distinguish Conveyor Belt from Material Pixels.

for $i = 0$ to `spatialSize` $- 1$ **do**
    `PCT` $\leftarrow$ `maxSpectrum`$[i] \times (1 +$ `thresholdPCT`$)$
    if `PCT` $>$ `pThresholdBelt`$[i]$ **then**
        `isConveyorBelt`$[i]$ $\leftarrow$ `false`
    else
        `isConveyorBelt`$[i]$ $\leftarrow$ `true`
    end if
end for

4.1.2. Classification

Once the data has been pre-processed according to the procedure described in Section 4.1.1, it must be classified, so in this work, we developed and analyse the behaviour of two different classification methods: one based on the Spectral Angle Mapper (SAM) and the other on an Artificial Neural Network (ANN). As said, the objective of this industrial

prototype is to classify four different materials, ABS (Acrylonitrile Butadiene Styrene), PP (Polypropylene), HDPE (High-Density Polyethylene), and PS (Polystyrene), which exhibit similar spectral responses.

The spectral signatures, or spectral patterns, of the target materials were built with pixels belonging to multiple Regions of Interest (ROIs) from different sample pieces of each material, captured at various positions along the conveyor belt. The same pre-processing pipeline described in the previous section, was applied to each pixel within these ROIs. Finally, the pre-processed spectra within each ROI were averaged to produce a representative reference pattern for each material, which are stored in a csv file that is loaded by the software application on startup.

This is a simple and fast procedure that can be repeated whenever a new material needs to be incorporated into the classification system. In such cases, a new pattern must be built for each new material, with the same pre-processing steps and acquisition settings. Similarly, when deploying the system in a new installation, the reference spectra for all materials should be re-acquired, since physical conditions such as lighting geometry, sensor height, and optical setup may differ and impact the spectral response. Reconstructing the material signatures in the new context is a more reliable and time-efficient strategy than attempting to replicate identical acquisition conditions across setups.

SAM-based Classification

The SAM-based classification method takes as input the spectrum obtained for each pixel, together with the reference patterns corresponding to each of the materials to be classified, and determines the angle between them in a multidimensional vector space. As described in the pre-processing phase, a preliminary classification step is used to identify and exclude pixels corresponding to the conveyor belt, thereby reducing the computational load of the SAM analysis, which depends directly on the amount of material present on the conveyor belt.

Both the pre-processed pixel spectrum and the reference patterns have a dimensionality of 224, corresponding to the number of spectral bands. In this project, we perform a spectral crop operation that restricts the analysis to bands 49 through 198, in order to exclude the spectral edges. The initial bands (1–48) and the final ones (199–224) exhibit steep signal transitions and low variability across different materials. These regions tend to produce highly similar responses for all samples, being not useful for discriminative purposes.

The spectral angle θ is given by Eq. (7) treating the spectrum vector **Spectrum** and the predefined material spectral patterns **Pattern** in an n-dimensional space (where $n = 150$). A smaller angle indicates a higher similarity between the two spectra.

$$\theta = \arccos\left(\frac{\mathbf{Spectrum} \cdot \mathbf{Pattern}}{\|\mathbf{Spectrum}\| \, \|\mathbf{Pattern}\|}\right) \tag{7}$$

SAM-based classification is applied only to those pixels whose mean reflectance across the spectrum exceeds the predefined threshold determined during calibration. Pixels falling below this reflectance threshold are initially assumed to belong to the conveyor belt and are excluded from further classification.

The algorithm corresponding to the SAM calculation, shown in Eq. (7), is presented in Algorithm 3.

Algorithm 3 Spectral Angle Mapper (SAM) Computation.

function SAM(`Spectrum`, `Pattern`)
  `dot_product` ← `Spectrum` · `Pattern`
  `norm_spectrum` ← ‖`Spectrum`‖
  `norm_pattern` ← ‖`Pattern`‖
  `angle_cosine` ← $\frac{\texttt{dot\_product}}{\texttt{norm\_spectrum} \cdot \texttt{norm\_pattern}}$
  θ ← arccos(`angle_cosine`)
  return θ
end function

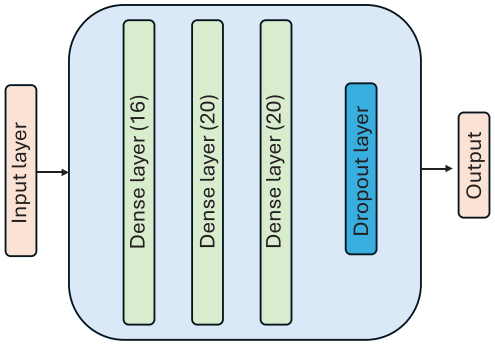


Fig. 3. ANN model architecture.

Algorithm 4 shows the classification of each pixel in a line using the SAM-based method. For every pixel spectrum, it computes the spectral angle with respect to each of the four material reference patterns and assigns the class corresponding to the smallest angle.

Algorithm 4 Line-wise Pixel Classification using SAM.

Input: Normalised pixel spectrum (`LineSpectrum`)
  Patterns Materials (`Materials`)
for all pixel ∈ `LineSpectrum` **do**
  `min_angle` ← ∞
  `best_class` ← −1
  for all pattern ∈ `Materials` **do**
    θ ← SAM(`pixel`, `pattern`) ▷ See Algorithm 3
    if θ < `min_angle` **then**
      `min_angle` ← θ
      `best_class` ← index of `Pattern`
    end if
  end for
  Assign `best_class` to current pixel
end for

ANN-based Classification

The other classification method analysed in this work is based on neural networks. Specifically, the architecture of the ANN model developed in this work is shown in Fig. 3. As can be seen, the model is composed of an input layer corresponding to the 224 spectral bands of a camera pixel, three dense layers using *relu* as the activation function, a dropout layer to reduce the possibility of overfitting [33] and an output layer using *softmax* as the activation function.

In a similar way than for the SAM-based algorithm a spectral crop is applied, restricting the analysis to bands 49 through 198, in order to exclude the spectral edges. The neural network model was trained with a dataset of almost 240,000 pixel spectra belonging to the different materials to be classified, as well as pixels corresponding to the conveyor belt. This data set was split into 80% for training and 20% for testing.

The accuracy of the ANN-based model after training is 0.9903. Fig. 4 shows the confusion matrix, where, as can be seen, the degree of accuracy is 100% for all materials except category ABS, that reaches a 96%.

4.2. Segment thread

After the first acquisition, preprocessing and classification of a preconfigured number of lines equal to the window size (1600 in our case), a signal is sent to the *segment* thread. At this point, the segmentation process starts and for each material to be classified, we first create an image as a binary mask and then, using OpenCV functions, we obtain the contour, the centroid position as well as other information such as the area for all detected objects in the binary mask. All detected objects with an area greater than a certain value, configurable in the

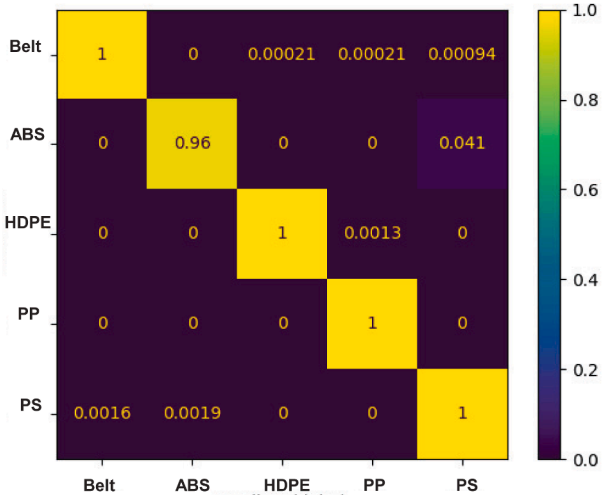


Fig. 4. Confusion matrix.

configuration parameters of the application, are then entered into an object list. This list of objects is a global variable and is accessed, with mutual exclusion, both by the *segment* thread to add objects and by the *sender* thread to extract objects and send their information to the requesting robot.

After the first classification, for each y new lines captured and classified, where y is a value configurable within the application and a divisor of the window size, a new segmentation is performed. This is done so that the objects can be visualised and tracked smoothly in the GUI. It must be taken into account that this new segmentation may include objects that were previously included in the object list and a check must be made to avoid these duplications. Therefore, in the developed software, additional segmentations are performed more frequently than strictly necessary solely to enhance the visualisation and tracking of the objects, as this extra processing does not compromise the system's real-time performance.

While the segmentation stage can be efficiently addressed using lightweight techniques in well-structured industrial environments, more complex scenarios may introduce additional challenges. In such cases, recent advances in deep learning-based object detection and segmentation have demonstrated strong performance in assembly line environments [34]. These approaches leverage convolutional neural networks to perform robust end-to-end detection and segmentation. The present work prioritises computational efficiency and deterministic real-time behaviour within the proposed multi-threaded architecture; nevertheless, the framework remains compatible with the integration of more advanced segmentation strategies if required by the application.

While the segmentation stage can be efficiently addressed using lightweight techniques in structured industrial settings, more complex scenarios may pose additional challenges. In such cases, deep learning-based object detection and segmentation approaches have demonstrated high accuracy, for example in industrial assembly line inspection tasks [34]. Furthermore, deep architectures that jointly extract spectral and spatial features have been proposed for semantic segmentation in HSI datasets [35,36].

4.3. Sender thread

As shown in Fig. 2, the application architecture contains a *sender* thread that is responsible for the communication between the different possible robots and the software application. Although the prototype system developed has only one robot, the application architecture and the communication system can be configured to work with several robots.

First of all, when the *sender* thread is created and launched, it establishes a socket communication between the computing platform and the different active robots. The number of robots as well as if they are active or not is determined by the application configuration parameters. Besides, each robot can be configured to pick just one type of material or to pick several ones, and also it is possible to assign different hoppers to the materials. Afterwards, the computing platform starts receiving packets from the different robots. There are three different types of command packets: BUSY, ACK, and NEW. The BUSY command is sent by a robot when it is moving with a picked object and it can not accept new positions. In this case, the computing platform sends and ACK response. The ACK command is sent by a robot to maintain the socket opened to avoid timeout, and the computing platform also responds ACK. When a NEW command is sent by a robot, in the communication packet also is sent the current x and y position of the robot. In this case, the computing platform looks for an object in the list which corresponds to the material that the robot is configured to pick, with the centroid closest to the current position (x,y) of the robot in order to speed-up the picking process. The computing platform compounds a packet with the selected object information including, centroid position, material, and destination hopper number.

5. Software-level optimisation methods

This section details the optimisation techniques employed to ensure that the computing system, both hardware and software, is not only capable of meeting the real-time requirements of the application, but is also efficient, scalable, and cost-effective. The goal is to develop a solution that maintains high performance without requiring hardware upgrades, thus facilitating future expansions and easing deployment in resource-constrained environments.

The computing system used in this project is based on the AMD Ryzen 9 7950X processor, AMD's Zen 4 architecture. It features 16 physical cores and 32 threads, with a base clock speed of 4.5 GHz and a boost frequency of up to 5.7 GHz. This architecture supports PCIe 5.0 and DDR5 memory, offering excellent throughput for data-intensive tasks.

Although the AMD Ryzen 9 7950X processor supports 32 threads via Simultaneous Multithreading (SMT), only a maximum of 16 threads will be in use in this project to maximise computational efficiency, except during TensorFlow performance evaluations. This decision is based on the fact that SMT does not double performance linearly, because the logical threads must share execution resources within a physical core, which can lead to contention for cache or memory bandwidth, for example. By assigning one thread per physical core, we can reduce context-switching overhead, avoid resource contention, and ensure that each thread operates with full access to the core's execution pipeline and cache hierarchy. This approach is used to result in more predictable and lower-latency performance, which is essential for real-time systems.

Moreover, in Windows, the operating system is responsible for managing how threads are assigned to CPU cores through its internal scheduler. Windows may migrate threads between cores dynamically during execution, due to load balancing, power management, resource availability, or thread priority. The scheduler aims to optimise overall system throughput by distributing workloads evenly across available cores and minimising idle time.

However, this dynamic thread migration can have negative consequences in performance-critical or real-time applications. When a thread is moved from one core to another, it typically loses access to the previous core's local cache (L1/L2), resulting in additional latency due to cache reloading from L3 or RAM.

One of the techniques used to avoid these issues is to explicitly set thread affinity, binding threads to specific cores [37]. This approach improves cache locality, reduces latency variability, and enhances overall predictability of execution, especially important in systems like the one proposed, where real-time processing is required. Furthermore, since the system is designed for industrial use and not as a general-purpose

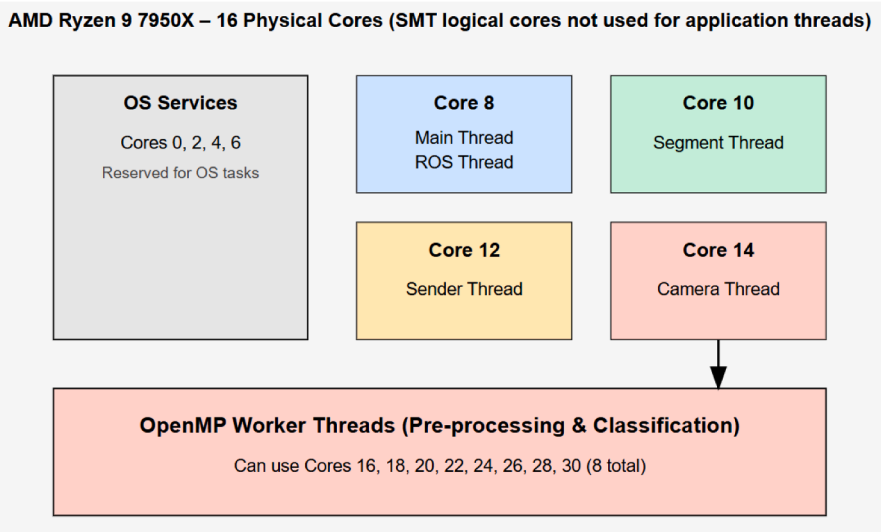


Fig. 5. Thread-to-core affinity configuration.

computing environment, all running processes are known and controlled. This makes it feasible to fine-tune core assignment without interference from unpredictable background tasks, further reinforcing the benefits of static thread placement.

Although the system uses *C++* standard threads, thread affinity can still be enforced by retrieving the native handle of each thread and calling the *SetThreadAffinityMask* function from the Windows API. Once a thread is bound to a specific logical core using this method, the operating system will no longer migrate it to other cores, ensuring consistent execution and improved cache locality.

In Windows, the scheduler typically maps new processes to the first logical cores. To avoid contention with system processes and services, all application-specific threads in this project are explicitly pinned starting from logical core 8 onwards, leaving the first 8 logical cores (cores 0 to 7) available for system processes, i.e the first 4 physical cores. Furthermore, only even-numbered logical cores are used, as each pair of adjacent logical cores (e.g., cores 8 and 9) corresponds to a single physical core in processors with SMT. By assigning threads only to even-numbered cores, we ensure that each thread runs on a separate physical core, for maximum performance and avoidance of resource sharing between concurrent threads.

In particular, the *main* and *ROS* threads are on core 8, the *segment* thread is on core 10, the *sender* is on core 12 and the thread that could be the biggest bottleneck, the *camera* thread, is on core 14. At this point, out of the 16 available physical cores, 8 free physical cores are used to accelerate the most computationally critical tasks, i.e. the pre-processing and sorting tasks discussed in Section 4.1.

The explicit affinity configuration adopted in this work is illustrated in Fig. 5. Each time-critical thread is pinned to a dedicated physical core, while operating system services are confined to lower-index logical cores and OpenMP worker threads occupy the remaining physical cores.

5.1. Spectral pre-processing

Algorithm 1 illustrates the pre-processing applied to all pixels in a single hyperspectral line. To accelerate execution and improve performance, OpenMP was used to parallelise the per-pixel pre-processing across multiple threads.

The parallelisation implemented in Algorithm 5 is based on a **parallel for** loop using OpenMP. Its performance will be analysed in Section 6.3.1, focusing on two key parameters: the scheduler type and the chunk size. The scheduler determines how loop iterations are distributed among available threads, using either *static* or *dynamic* scheduling strategies provided by OpenMP, while the chunk size specifies how many iterations are assigned to each thread at a time. These parameters can significantly influence overall efficiency, for example through their impact on load balancing and memory locality.

Algorithm 5 Pre-process parallel algorithm.

Input: Hyperspectral line (raw pixel spectra), precomputed white-dark references, threshold values
Output: Normalised line, conveyor/matter mask
parallel for $i = 0$ to `spatialSize` $- 1$ **do**
    `Spectrum` $\leftarrow$ raw spectrum of pixel i
    `darkSpectrum` $\leftarrow$ dark reference of pixel i
    `whiteMinusDark` $\leftarrow$ `whiteSpectrum`$[i]$ $-$ `darkSpectrum`$[i]$
    (`Spectrum`$[i]$, `maxSpectrum`) $\leftarrow$DoubleNormalisation
            (`Spectrum`$[i]$, `darkSpectrum[i]`, `whiteMinusDark[i]`)
            (see Algorithm 1)
    `isConveyorBelt`$[i]$ $\leftarrow$ ConveyorBeltHeuristic
            (`maxSpectrum`, `thresholdPCT`, `pThresholdBelt`$[i]$)
            (see Algorithm 2)
end parallel for

Additionally, we will analyse the impact of using thread affinity, i.e., binding threads to specific physical cores. Affinity can improve cache locality and reduce thread migration overhead, but it must be evaluated in conjunction with the scheduling strategy to ensure optimal performance. In our evaluation, we compare executions with and without affinity to assess its effect on scalability and temporal predictability in real-time environments.

5.2. Classification

As previously explained, this work evaluates two different classification methods for hyperspectral material sorting: one based on Artificial Neural Networks (ANNs), and another using the Spectral Angle Mapper (SAM). These two approaches differ significantly in their integration within the project and in the possibilities for performance optimisation. The ANN-based method relies on third-party libraries (TensorFlow), which abstract the low-level implementation details but limit fine-grained control over thread management and memory optimisation. In contrast, the SAM-based method has been fully implemented within the project, allowing for custom parallelisation strategies. As a result, the acceleration techniques applicable to each method vary

considerably and are evaluated independently in terms of performance and scalability.

5.2.1. ANN

The trained ANN model described in Section 4.1.2 is imported into the application and integrated into the processing pipeline. The model is invoked for each hyperspectral line and outputs a classification label for every pixel, including those corresponding to the conveyor belt, which is treated as a distinct material class.

The model inference is executed entirely on the system's CPU, as the performance requirements of the application are satisfied without the need for hardware accelerators such as a GPU. This choice simplifies the hardware configuration and avoids the additional overhead associated with GPU integration and data transfers, while still meeting real-time constraints.

By default, the TensorFlow library, despite being used solely for inference, utilises all available resources in the system. This behaviour can be adjusted using TensorFlow's configuration options in order to analyse its impact on performance and to better integrate inference into a shared CPU environment. In particular, TensorFlow provides a low-level API function, (*TF_SetConfig*), which allows users to configure various aspects of the runtime environment before creating a session. This function can be used to define execution parameters such as the number of CPU threads, GPU device visibility, memory growth policies, and inter/intra operation parallelism. Although TensorFlow allows limiting the number of computational resources, specifically CPU cores, it does not provide mechanisms for applying thread affinity techniques to bind threads to specific physical cores.

Section 6.3.2 will examine the computational cost of inference in order to assess the scalability of the method and to determine the most efficient use of system resources.

5.2.2. SAM

Algorithm 6 shows the parallel implementation of the SAM-based classification process, which, like the pre-processing algorithm, is based on a **parallel for** loop using OpenMP. Its performance will be analysed in Section 6.3.3, with a focus on the scheduler type, the chunk size, and the impact of using thread affinity.

Algorithm 6 Parallel Line-wise Pixel Classification using SAM.

```
Input: Normalised line (Spectrum)
       Patterns Materials (Materials)
parallel for i = 0 to spatialSize − 1 do
    if isConveyorBelt[i] then
        Assign conveyor_label to current pixel
    else
        min_angle ← ∞
        best_class ← −1
        for all pattern ∈ Materials do
            θ ← SAM(pixel, pattern)          ▷ See Algorithm 3
            if θ < min_angle then
                min_angle ← θ
                best_class ← index of pattern
            end if
        end for
        Assign best_class to current pixel
    end if
end parallel for
```

In this case, it is important to highlight that the computational cost per pixel is primarily determined by the SAM calculation, which is repeated for each material class that needs to be classified. It is also worth noting that this algorithm explicitly incorporates the prior classification step that determines whether a pixel corresponds to the conveyor belt or to a material. This means that the computational cost per pixel depends on the outcome of that pre-classification: if the pixel is identified as belonging to the conveyor belt, the processing cost is minimal; otherwise, a full classification is performed. Note that, while the number of material classes remains fixed for a given industrial installation, it may vary from one installation to another. Therefore, the total classification cost per hyperspectral line in a given installation depends on the nature of the line itself, that is, on how many pixels correspond to classifiable material.

6. Experimental results

In this section, the experimental configuration used to evaluate the proposed system is first described in Section 6.1, including the hardware platform, hyperspectral acquisition setup, and dataset characteristics. Subsequently, the classification performance obtained using the two previously introduced methods, the ANN-based and the SAM-based approaches, is presented in Section 6.2. Finally, the computational feasibility and scalability of the proposed system, highlighting its potential for real-time or large-scale industrial applications, are analysed in Section 6.3.

6.1. Experimental setup

6.1.1. Hardware platform

As already introduced, the platform used is equipped with the AMD Ryzen 9 7950X, that integrates 16 physical cores and 32 threads, with a total of 81 MB of cache, comprising 1 MB of L1, 16 MB of L2, and 64 MB of L3 cache. The L3 cache is distributed across two 8-core CCDs (Core Complex Dies), each with 32 MB of L3 shared among the cores. The system is equipped with 64 GB of DDR5 RAM running at 6000 MT/s, installed on an ASUS TUF GAMING X670E-PLUS WIFI motherboard, and a 1 TB NVMe SSD used for system storage and data handling. Although the GPU is active and performing its standard graphics functions, no GPGPU computation is involved in this project, as all data processing tasks are handled exclusively by the CPU.

6.1.2. Hyperspectral camera

The hyperspectral acquisition system is based on a SPECIM FX17 camera (Specim, Spectral Imaging Ltd., Finland). The FX17 is a pushbroom hyperspectral sensor operating in the short-wave infrared (SWIR) range from 900 nm to 1700 nm.

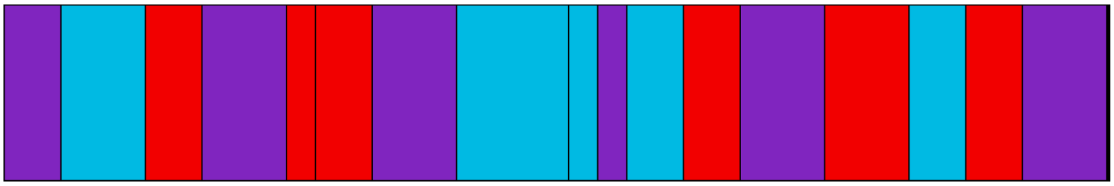
The camera provides 224 contiguous spectral bands with a spectral sampling interval of approximately 3.6 nm. The spatial resolution of the sensor is 640 pixels per line. When operating at full spectral resolution (224 bands), the maximum achievable acquisition frequency is 670 frames per second, corresponding to a line acquisition period of $1.493ms$.

As a pushbroom sensor, each captured frame corresponds to a single spatial line containing 640 spatial samples and 224 spectral channels. Consecutive lines are acquired synchronously with the conveyor belt motion to reconstruct the hyperspectral datacube in real time.

6.1.3. Datasets

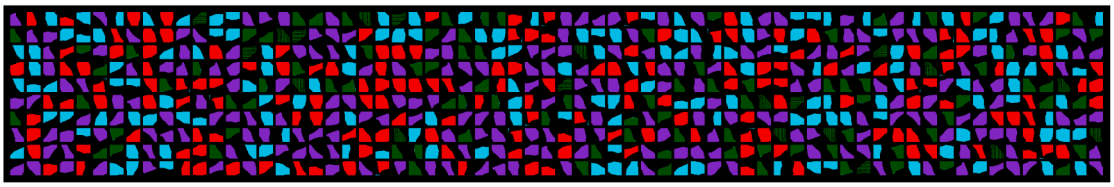
To evaluate the computational behaviour of the proposed architecture under different operating conditions, three experimental scenarios, shown in Fig. 6, were considered. Each scenario represents a distinct spatial distribution of materials on the conveyor belt and is designed to analyse specific computational conditions.

- **Scenario 1 (Full-load synthetic configuration):** The conveyor belt carries rectangular blocks of a single material type that fully cover its surface. Consequently, nearly all spatial pixels in each acquired line correspond to material that must be classified. This synthetically generated scenario represents a worst-case spectral processing condition, as the classification module operates at maximum load with minimal background removal.



(a) Scenario 1

(b) Scenario 2



(c) Scenario 3

Fig. 6. Graphical information.

- **Scenario 2 (Real industrial acquisition):** The conveyor belt transports discrete pieces of the four reference materials (ABS, PP, HDPE and PS), with large empty areas where no material is present. This configuration is constructed from experimentally acquired data and using to generate the other two scenarios.
- **Scenario 3 (High-fragmentation synthetic configuration):** Shredded material pieces suitable for robotic manipulation are distributed across a nearly full conveyor belt without overlapping between fragments. This synthetically generated scenario is designed to stress the segmentation and object-management stages by introducing a large number of disconnected regions, allowing verification that high object fragmentation does not introduce additional computational bottlenecks.

In all cases, the hyperspectral data are stored on disk and each scenario consists of 12,000 acquired lines. Given the spatial resolution of 640 pixels per line and 224 spectral bands per pixel, this corresponds to 7.68 million spectra per scenario. This controlled setup enables consistent comparison of computational scalability under maximum-load, realistic, and high-fragmentation conditions.

6.2. Classification performance

We will first determine which algorithm is best in terms of accuracy in classifying materials. To do this, we will count the number of pixels that have been correctly classified. In addition, we will show the number of correctly sorted pieces. In Table 1, we show the percentage of correctly classified pixels with respect to the total pixels of the material pieces. As can be seen, the ANN-based algorithm performs better than the SAM-based algorithm, with a 99.9% success rate compared to 98.0% for SAM-based one for all scenarios. Table 1 also shows the percentage of segmented objects well classified, being the percentage of success for both algorithms 100%.

In Fig. 7 we show the sorting result of the four materials according to Scenario 2 for both SAM-based and ANN-based algorithms, as well as the object detection performed by the segmentation process. As can be seen, the ANN-based algorithm performs slightly better than the SAM-based algorithm. Furthermore, the only object that is not accurately classified by both algorithms is the white PS, which is semi-transparent. This misclassification means that the contour of the object is incomplete and, although it is recognised as an object, its centroid is incorrect. The behaviour is similar for scenarios 1 and 3, where the only misclassifications are for semi-transparent PS objects.

Table 1
Accuracy: Percentage of Correctly Classified Pixels for SAM-based and ANN-based algorithms.

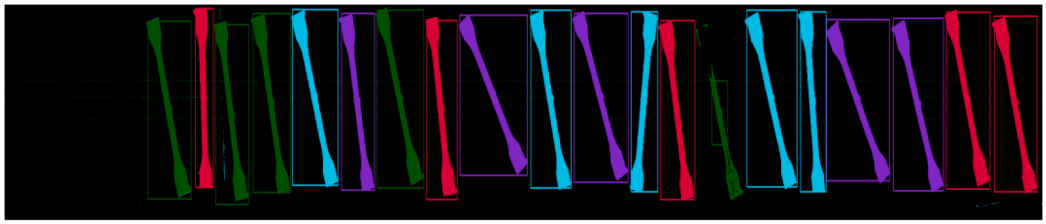
	SAM-based	ANN-based
Percentage of Correctly Classified Pixels (%)	97.9 100.0	99.9 100.0

6.3. Computational performance

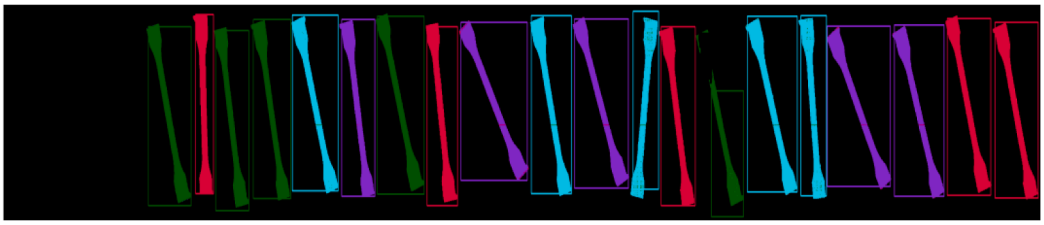
As stated in Section 4, when the application starts, the *main* thread spawns four additional threads to prevent contention or delays that could compromise timing constraints. The thread named *camera* is subject to the strictest real-time requirements, as it must complete its operations within the $1.493ms$ available between two consecutive hyperspectral camera acquisitions at maximum camera frame rate (670 fps). This thread is responsible for pre-processing and classifying the pixels of each new camera line upon receiving a new acquisition trigger. These accelerated operations are analysed in detail in this section.

6.3.1. Computational evaluation of the pre-processing stage

As explained in Section 5.1, the double normalisation pre-processing differs slightly depending on whether the SAM-based or the ANN-based classification method is used. Unlike SAM-based, the ANN-based approach does not require pre-sorting pixels by conveyor belt. Neverthe-



(a) SAM-based method



(b) ANN-based method

Fig. 7. Tracking area of the objects segmentation of the 4 materials using SAM-based and ANN-based methods.

Table 2
Pre-processing time and standard deviation, speed-up, and efficiency vs. number of threads (*static* schedule, no thread binding, Scenario 1).

	Number of threads							
	1	2	3	4	5	6	7	8
Time (μs)	191.9	104.6	74.0	57.0	48.0	44.2	43.2	41.2
St. Dev.	19.6	23.1	24.6	21.2	22.2	22.9	25.4	24.8
Speed-up	-	1.8	2.6	3.4	4.0	4.3	4.4	4.7
Efficiency	-	92%	86%	84%	80%	72%	63%	58%

less, all computational results presented in this section incorporate the corresponding pre-processing step.

Table 2 shows the preprocessing times, including the pre-sorting step described above, based on the number of OpenMP threads used with a static schedule where the chunk size is set to the maximum possible so that all threads are assigned the same work. In this configuration, the operating system (Windows 11) handles the assignment of threads to cores and can migrate threads between cores if necessary. Table 2, which corresponds to Scenario 1, also shows teh standard deviation, the speedup and the parallel efficiency. All values represent the average processing time calculated over the 12,000 lines of this scenario.

As can be seen the parallelisation performed gives good performance, although the computation times are small, in the order of microseconds. Our goal is to make the system meet real-time requirements and be scalable. The standard deviation computed over the $12,000$ values serves as a strong indicator of system stability, a point that will be discussed further in Section 7.

The following is an analysis of the behaviour when using *dynamic* scheduling instead of *static* scheduling. Table 3 shows the pre-processing execution times when *dynamic* scheduling and different chunk sizes are used.

Tables 2 and 3 show that the overhead introduced by using *dynamic* scheduling with different chunk sizes, as opposed to *static* scheduling, is not justified in the context of this pre-processing step. The expected benefit of *dynamic* scheduling, correcting potential load imbalance across threads, does not translate into improved performance here. On the contrary, the execution times are slightly higher, and the efficiency tends to degrade as more threads are used. This overhead not only affects the average performance, but also increases result variability, as reflected in the higher standard deviation values observed. This behaviour is explained by the nature of the pre-processing task, where there is no intrinsic load imbalance: the computational cost associated with processing each pixel is identical, with no conditional branches or data-dependent divergence (except in pre-sorting pixels).

As discussed in Section 5, one level of optimisation is to bind threads to specific physical cores. In this setup, each thread, whether created explicitly by the *main* thread or implicitly within OpenMP parallel regions, is assigned to a fixed core, preventing the operating system from migrating it between cores during execution.

It is important to note that the thread that initiates an OpenMP parallel region acts as the master thread (with identifier 0) within that region and therefore retains its original core assignment. In contrast, the remaining threads require explicit core binding. Table 4 shows the pre-processing computing times when this thread binding is applied. Comparing the results of Tables 2 and 4, a significant improvement is shown in both execution time and parallel efficiency when binding thread placement is applied. As expected, this improvement becomes more pronounced as the number of threads in parallel region increases. This is largely due to the fact that without thread binding, multiple threads, including those from the application and operating system, can be scheduled on the same physical core, leading to resource contention and reduced performance.

The results presented for Scenario 1 are consistent with those obtained for scenarios 2 and 3, both in terms of trends and magnitude of performance. Table 5 compare the results for these scenarios using the optimal parameter settings identified in the evaluation of Scenario 1. The differences in execution times are minimal and can be attributed to transient system conditions, such as CPU scheduling and background activity.

6.3.2. *Computational evaluation of the ANN-based classifier*

To reduce costs, GPUs have not been included in the project's computing system. Consequently, we use the CPU-only version of TensorFlow during the inference stage of our material classification process. By

Table 3
Pre-processing time and standard deviation, speed-up, and efficiency vs. number of threads (*dynamic* schedule, no thread binding, Scenario 1).

	Number of threads							
	1	2	3	4	5	6	7	8
Schedule: dynamic, chunk size = 5								
Time (μs)	195.06	110.04	105.59	100.08	52.75	50.89	52.38	52.82
St. Dev.	24.96	27.62	37.48	37.11	25.33	34.37	35.05	35.62
Speed-up	-	1.8	1.8	1.9	3.7	3.8	3.7	3.7
Efficiency	-	89%	62%	49%	74%	64%	53%	46%
Schedule: dynamic, chunk size = 10								
Time (μs)	198.20	105.54	74.48	74.00	52.03	71.80	61.94	57.20
St. Dev.	33.80	19.79	20.67	38.71	25.82	39.53	32.71	42.83
Speed-up	-	1.9	2.7	2.7	3.8	2.8	3.2	3.5
Efficiency	-	94%	89%	67%	76%	46%	46%	43%

Table 4
Pre-processing time and standard deviation, speed-up, and efficiency vs. number of threads (*static* schedule, thread binding, Scenario 1).

	Number of threads							
	1	2	3	4	5	6	7	8
Time (μs)	193.4	103.8	71.7	55.8	46.3	41.2	35.6	31.4
St. Dev. (μs)	19.6	23.1	24.6	21.2	22.2	22.9	25.4	24.8
Speed-up	-	1.9	2.7	3.5	4.2	4.7	5.4	6.2
Efficiency	-	93%	90%	87%	84%	78%	78%	77%

Table 5
Pre-processing time vs. number of threads (*static* schedule, thread binding).

Time (μs)	Number of threads							
	1	2	3	4	5	6	7	8
Scenario 1	193.4	103.8	71.7	55.8	46.3	41.2	35.6	31.4
Scenario 2	199.4	109.6	75.4	58.9	48.4	42.8	37.3	32.5
Scenario 3	201.4	104.5	71.9	55.1	47.5	39.9	35.5	31.0

Table 6
Classification time of the ANN-based classifier vs. number of threads.

Time (μs)	Number of threads				
	1	2	4	8	32
Scenario 1	227.95	220.18	230.86	305.07	336.10
Scenario 2	227.93	220.17	230.84	305.07	336.09
Scenario 3	227.93	220.16	230.84	305.07	336.09

default, TensorFlow leverages all available system resources, 32 CPU in our platform, for intra-operation and inter-operation parallelism during computation. However, this behavior can be explicitly modified at the session level by passing a serialised ConfigProto message to the *TF_SetConfig* function, allowing control over thread usage via the *intra_op_parallelism_threads* and *inter_op_parallelism_threads* fields.

The *intra_op_parallelism_threads* parameter is a session-level configuration parameter that determines the maximum number of threads that can be used to execute individual operations in parallel within a single *op* context. This setting controls intra-operation parallelism, meaning how much parallel computation is allowed internally when performing complex operations such as matrix multiplication or convolution.

As shown in Table 6, the inference process in this project does not exhibit scalability; that is, increasing the number of CPU cores does not lead to reduced execution times. On the contrary, unless the system is explicitly configured to use a single core, the inference times may actually increase. The worst-case scenario occurs when no thread limit is specified, in which TensorFlow uses the maximum number of available threads, leading to the longest computation times observed.

As shown in Table 6, the inference times do not differ significantly across scenarios, indicating that the computational cost remains stable. Although the use of two threads slightly reduces execution time, the improvement is not substantial. While the current setup meets real-time requirements, the limited potential for further acceleration may imply that more demanding configurations, such as those involving higher resolution, frame rates, or spectral complexity, might fail to do so, thereby limiting the scalability of the system.

6.3.3. Computational evaluation of the SAM-based classifier

Unlike the previous case, and as in the pre-processing step, the SAM-based classification method was implemented without the use of external libraries. Although some input data resides in OpenCV structures due to other processing, OpenCV itself is not used within the algorithm. Instead, the implementation was fully optimised experimentally, with additional enhancements as described in Section 5.

To meet the real-time requirements imposed by the hyperspectral imaging system, it was necessary to optimise the initial sequential implementation of the processing pipeline. To improve the performance, all required operations are precomputed at the application startup. This optimisation corresponds to a modification of Algorithm 3, which is presented in Algorithm 7.

Algorithm 7 Optimised Spectral Angle Mapper (SAM) Computation.

function SAM(`Spectrum`, `Pattern`, `norm_pattern`)
  `dot_product` $\leftarrow$ `Spectrum` $\cdot$ `Pattern`
  `norm_spectrum` $\leftarrow \|$`Spectrum`$\|$
  `angle_cosine` $\leftarrow \dfrac{\texttt{dot\_product}}{\texttt{norm\_spectrum} \cdot \texttt{norm\_pattern}}$
  $\theta \leftarrow \arccos($`angle_cosine`$)$
  return θ
end function

Table 7
Classification time (μs) of the SAM-based classifier vs. number of threads and scheduling.

Schedule	1	2	3	4	5	6	7	8
Scenario 1								
static	179.16	96.10	65.41	52.44	40.92	35.64	31.13	26.55
dynamic, 5	181.95	100.63	70.24	53.78	44.79	37.68	33.04	29.37
dynamic, 10	180.45	99.79	73.06	54.48	42.70	38.57	32.89	30.80
Scenario 2								
static	100.74	58.51	47.05	42.75	35.25	28.95	27.29	22.84
dynamic, 5	102.55	60.93	44.13	36.21	31.50	27.78	25.51	23.54
dynamic, 10	99.86	59.58	42.64	33.96	29.21	26.48	24.64	21.73
Scenario 3								
static	144.05	83.51	60.79	49.05	40.92	37.25	31.77	28.99
dynamic, 5	147.23	88.01	61.93	48.84	39.23	35.54	32.53	28.53
dynamic, 10	144.76	82.85	58.36	46.29	38.32	33.76	30.15	27.15

Table 8
Average segmentation time.

	Time 400 lines (μs)	Avg. Time per Line (μs)
Scenario 1	50588.0	126.47
Scenario 2	47893.0	119.73
Scenario 3	53984.0	134.96

Thread binding was used in all the experiments presented below. As shown in the pre-processing analysis, thread binding contributes to improve the computational efficiency, especially when the number of threads in OpenMP parallel regions increases. In Table 7, we show classification execution times using the accelerated SAM-based algorithm under three configurations: *static* scheduling with the largest chunk size, and *dynamic* scheduling with chunk sizes of 5 and 10.

The sequential computational cost (i.e., execution with a single thread) differs across scenarios due to the pre-sorting applied to the input data. In scenarios where pixels have already been sorted by conveyor belt threshold, redundant sorting is avoided, which reduces the overall computation time. Because all pixels are analysed in Scenario 1, this scenario allows us to conclude that the SAM-based classifier outperforms the ANN-based method in terms of execution time. Moreover, the SAM-based classification allows for effective parallelisation of the code, achieving execution times approximately 10 times faster than those of the ANN-based one. Since the dynamic schedule with a chunk size of 10 yields better results in scenarios 2 and 3, it should be considered the default scheduling strategy, unless an experimental analysis is performed under specific conditions.

6.3.4. Computational evaluation of the segmentation stage

As indicated in Section 4.2, the segmentation process is a common step regardless of the classification algorithm used. One segmentation process is performed at least every 400 newly classified lines, this number being configurable in the configuration parameters of the software application. Table 8 shows the execution times required to perform this segmentation process in microseconds for the different scenarios evaluated. As can be seen, in the worst case (Scenario 3), the average time per line to perform the segmentation is 134.96 μs. This means that this processing stage does not require the application of parallelism techniques, since there is a large margin of time available for this stage, since the maximum time available would be $1.493ms$ at the highest camera frequency (670 fps).

7. Discussion

This work presents the development and evaluation of a computational system for controlling a machine designed to sort recyclable materials. In particular, a modular software architecture has been implemented with two main objectives: first, to meet the real-time requirements of the system and second, to ensure that the system is scalable and adaptable to new configurations, with higher computational demands, such as increased resolution, throughput, number of material types to be classified, or spectral complexity.

Given the nature of the project, the computing platform integrated into the industrial machine is dedicated exclusively to its operation. Accordingly, our proposal assumes full and exclusive use of this platform. The machine is not an isolated system, it is connected and has communication capabilities. Therefore, all results presented have been obtained with the system fully connected and running the operating system. It should also be noted that one of the constraints imposed by the industrial partner is that the development must be carried out under the Windows operating system.

The computing system employed can be considered a high-end desktop-class platform, offering a relatively powerful configuration for its category, but without incorporating dedicated hardware accelerators such as GPUs. Despite delivering acceptable performance for the intended tasks, its cost remains very low in the context of the full industrial installation.

The exclusive use of the computing platform within the industrial system has enabled the development of a highly efficient and scalable software. A key aspect of the system's design is its two-level workload distribution strategy: at the first level, separate threads are assigned to the main software components, and at the second level, performance-critical computations are further accelerated using parallelisation techniques to minimise execution time.

The results show that both levels of work distribution achieve significantly better performance when thread placement or binding is explicitly managed by the developed software architecture, rather than relying on the default operating system scheduling. As required by the industrial constraint to develop under the Windows operating system, thread affinity is managed using the Windows API that allows binding each thread to a specific physical core through an affinity mask. This prevents the operating system from migrating threads across cores. Importantly, logical processors provided by SMT are deliberately not used, so only physical cores are targeted. As a result, the maximum number of processes generated by our software is limited to 12, out of the 16 available physical cores, reserving 4 cores primarily for operating system and other background processes.

In the parallelisation techniques used, for the second level of workload distribution, two types of schedules were analysed, one *static* and one *dynamic*. In the pre-processing stage, it was shown that the static distribution is preferable, where no load imbalance is present. In the case of classification stage with the SAM-based algorithm, where there is load imbalance due to the previous pre-sorting, the *dynamic* distribution slightly improves the computation times. However, the relevance of this improvement is not significant due to the increased time cost of the *dynamic* distribution compared to the *static* one. In cases where load balance is uncertain due to potential interference from external processes, *dynamic* scheduling becomes the more appropriate choice. However, in our case, this situation does not arise, as the system is used exclusively

by the application, ensuring stable and predictable workload distribution.

The experimental results were obtained using three different scenarios that cover representative configurations in terms of computational load. Each scenario consists of 12,000 lines of hyperspectral data. In particular, Scenario 1 was designed to represent the worst-case processing condition. While preprocessing is performed for all pixels in every scenario, classification is only applied to pixels identified as material. In Scenario 1, all pixels correspond to material, and therefore both preprocessing and classification are executed for the entire image, with no background (conveyor belt) regions to be excluded.

These three scenarios effectively represent different workload regimes, ranging from maximum spectral processing load (Scenario 1) to experimentally acquired configuration (Scenario 2) and high object fragmentation (Scenario 3). Therefore, the reported results provide a throughput evaluation under varying computational conditions without modifying hardware or acquisition parameters.

This work has presented two classification methods, both of which achieved very good sorting performance, with slightly better accuracy observed in the ANN-based approach compared to the SAM-based one. However, the accelerated computation times of the SAM-based method are significantly lower than those of the ANN-based method. In terms of acceleration, it should be highlighted that the ANN-based method was executed using the CPU version of the TensorFlow library, as our system does not rely on GPU acceleration. In the case of the ANN-based method, the computational cost increases due to the growth of the network size; however, this increase is not proportional to the number of reference patterns, as it is in the SAM-based method. It should also be recalled that, in the SAM-based method, a pre-sorting step was performed to avoid classifying conveyor belt pixels. On the other hand, incorporating new patterns in the SAM-based method does not require software changes, while the ANN-based method necessitates retraining the neural network. This last aspect is crucial, as it favours the use of the SAM-based method for the development of versatile and adaptable systems.

From a scalability perspective, the computational cost of the preprocessing stage scales approximately linearly with both the spectral dimensionality and the spatial resolution of the sensor, since operations are applied per pixel across all spectral bands. The computational scalability of the classification stage differs between the two approaches. In the SAM-based method, the computational cost increases approximately linearly with the number of material classes (considering a number of spectral bands), since the spectral angle must be computed between each pixel and every reference material signature. Consequently, adding new material classes directly increases the number of required comparisons. In contrast, for the ANN-based method, the inference cost is primarily determined by the network architecture and input dimensionality. Increasing the number of material classes typically affects only the size of the output layer, resulting in a comparatively moderate increase in inference cost rather than a proportional growth with the number of reference patterns. Therefore, the computational scalability behaviour of both methods differs significantly when the number of classes is expanded.

Operational scalability also differs between both approaches. While the SAM-based method allows new material classes to be incorporated by simply extending the reference signature set, the ANN-based approach requires network retraining to accommodate additional outputs. This difference has practical implications in dynamic industrial environments where material categories may evolve over time.

Under the evaluated configuration (150 spectral bands and 640 spatial pixels per line), and considering the worst-case computational scenario executed using 8 processing threads, the combined preprocessing and SAM-based classification time was measured at 59 μs per line. This execution time corresponds to a theoretical maximum processing frequency of approximately 17 kHz, which is more than 25 times higher than the current acquisition rate of 670 fps (1.493 ms per line). Assuming linear scaling with respect to spatial resolution and spectral dimensionality, this margin implies that, while maintaining the same hardware and acquisition frequency, the system could support on the order of 16,000 spatial pixels per line or nearly 3800 spectral bands within the available per-frame budget.

Expressed in terms of spectral throughput, the evaluated configuration (150 spectral bands and 640 spatial pixels per line) processes 96000 spectral samples per line. Considering the average measured processing time of $59\mu s$ per line, the proposed architecture would theoretically support up to approximately 16.3×10^8 spectral samples per second under the same hardware conditions. This capacity exceeds the data throughput of several commercially available high-throughput hyperspectral cameras discussed previously, such as the XIMEA MX022HG-IM-SM5X5-NIR2-FL (7.53×10^8 spectral samples per second) and the ULTRIS 5 HFR system (2.77×10^8 spectral samples per second). These comparisons confirm that the proposed software architecture can accommodate sensors with significantly higher spatial or spectral sampling rates without compromising real-time operation.

Regarding the number of material classes, preprocessing time remains constant, and the SAM-based classification stage scales approximately linearly with the number of reference signatures. Under these conditions, the available classification budget of 1.461 ms would theoretically allow up to approximately 216 material classes before reaching the real-time limit. Moreover, approximately 33 classes would be required for the SAM-based method to match the total execution time of the ANN-based implementation. These projections indicate that the proposed architecture preserves a substantial computational margin for higher-resolution sensors, alternative camera models, expanded material libraries, and more demanding industrial configurations.

Although the proposed software architecture is largely independent of the specific hyperspectral sensor employed, except for the communication interface and data acquisition layer, changes in sensor type or spectral characteristics may require adjustments at the calibration and classification levels. In particular, radiometric references (white and dark calibration) and material spectral signatures must be updated when different sensors or illumination conditions are used. In the case of the SAM-based method, this process involves redefining the reference and end-members signatures, which can be performed efficiently. For the ANN-based approach, however, changes in spectral response may require retraining of the neural network.

Furthermore, certain preprocessing stages may depend on operational conditions such as illumination variability, dust presence, sensor drift, or environmental factors affecting spectral stability. For this reason, the system has been designed to operate with a significant computational margin between acquisitions, allowing additional preprocessing or compensation strategies to be incorporated without compromising real-time performance.

Recent reviews highlight the increasing adoption of deep learning and hybrid spectral-spatial approaches across diverse hyperspectral imaging applications [5]. While these trends demonstrate significant advances in modelling capability, practical industrial deployment often remains constrained by real-time requirements, hardware cost, and system integration considerations. In this context, optimized CPU-based architectures continue to provide a competitive and scalable solution for time-critical inspection tasks.

Although the proposed implementation relies exclusively on CPU-based processing, a design choice driven primarily by cost considerations, the scalability analysis shows that substantial computational headroom is available; therefore, hardware acceleration is not required under the evaluated conditions.

However, GPU-based acceleration may become relevant in future expansion scenarios. For example, this could occur if additional non-classification processing, such as deep learning-based segmentation, requires GPU execution, or if a significant increase in spectral, spatial, or frequency resolution, or the adoption of more complex neural architectures for classification, increases the computational demand beyond the current margins. In such cases, the processing strategy should shift

from the current line-based model to a window-based scheme, thereby reducing the overhead associated with frequent small data transfers and allowing sufficiently large computational batches to efficiently exploit GPU parallelism. This approach would enable effective utilization of hardware accelerators while preserving the modular structure of the proposed architecture.

Although the experimental validation has been conducted on a specific industrial prototype, the proposed architecture has been deliberately designed to ensure adaptability across diverse industrial environments. Regarding illumination conditions, the current setup employs broad-spectrum halogen lamps under cost-constrained industrial settings. In scenarios where higher-quality or spectrally optimized lighting systems are used, for example, narrow spectral range illumination or uniform intensity profiles, the preprocessing requirements would be simplified, reducing computational cost and further increasing real-time margins.

Concerning sensor variability, the software architecture is decoupled from the specific acquisition hardware. As mentioned, modifications in communication interface (e.g., Camera Link, GigE), spectral range, or spatial resolution only affect the data acquisition layer and parameter configuration, without altering the core software structure. While changes in spectral or spatial resolution may modify real-time constraints, the system has demonstrated substantial temporal margin, ensuring scalability toward higher acquisition rates or increased data dimensionality.

Material heterogeneity across industrial facilities is addressed through the modular classification framework. As previously stated, the number of end-members can be extended without architectural modifications in the SAM-based approach, and additional classes can be incorporated in the ANN-based method via retraining. In cases of increased spectral variability or more complex material compositions, the system can operate using the full set of available spectral bands or even integrate sensors with higher spectral resolution. The experimental results show that sufficient computational headroom exists to accommodate such increases in spectral dimensionality.

On the other hand, conveyor belt speed is primarily determined by industrial installation parameters, including pixel ground sampling distance and material stability during transport. High speeds may induce unwanted displacement in lightweight or irregular materials, potentially affecting robotic picking accuracy. For this reason, conveyor velocity must be tuned according to material properties and mechanical constraints. The proposed architecture does not impose intrinsic speed limitations beyond those defined by acquisition timing, and the demonstrated processing margin ensures compatibility with a broad range of industrial throughput requirements.

8. Conclusions

The proposal of a versatile and scalable industrial system has been validated, demonstrating its feasibility and its ability to meet real-time processing requirements. An efficient and scalable software architecture has been developed for real-time industrial material classification based on hyperspectral imaging (HSI).

The proposed software architecture distributes the distinct first-level processing stages across independent threads (heterogeneous parallelisation), while homogeneous parallelisation is applied at the second level using high-performance computing techniques on multicore CPU systems. A detailed evaluation of computational requirements and the effectiveness of various acceleration strategies, including OpenMP, thread affinity, and static and dynamic workload distribution has demonstrated the system's ability to meet real-time constraints with a significant margin, thereby guaranteeing scalability under more demanding workloads.

The scalability of the system is influenced by multiple factors, including the number of spectral bands processed. In our prototype, 150 out of the 224 available bands were used for classification purposes. The system is also scalable with respect to the spatial resolution of the sensor, 640 pixels per line in our case, and the acquisition rate, which reached up to 670 frames per second. Additional scalability dimensions include the speed and dimensions of the conveyor belt, the number of robotic arms deployed for picking operations, the inclusion of RGB cameras, and the configuration of the illumination system, which may affect pre-processing complexity and computational load. Beyond these physical aspects, the system has also been designed to support algorithmic and computational scalability. Its modular architecture allows for the integration of more sophisticated pre-processing, classification, or segmentation algorithms without altering the entire processing pipeline.

Two classification methods have been developed and analysed in the context of the addressed problem: one based on Artificial Neural Networks (ANN) and the other on the Spectral Angle Mapper (SAM) algorithm. Both achieved good classification accuracy, however, the SAM-based method proved to be more computationally efficient and easier to adapt to new materials, making it a more versatile solution for dynamic industrial environments. In contrast, the ANN-based approach is preferable when the number of distinct materials to be sorted increases significantly, as it can handle greater complexity once properly trained.

Future lines of development include the use of hardware acceleration with GPUs to support more computationally demanding tasks, which would enable the incorporation of intensive pre-processing techniques, such as Savitzky-Golay filtering, to increase the interval between system recalibrations and prolong uninterrupted operation; and the inclusion of heuristic algorithms and associated optimization functions, which must be specifically designed, to determine the order in which parts are picked.

In particular, when segmentation challenges arise due to intrinsic material properties, such as partial transparency in white PS, several alternative strategies could be considered. One option would be the integration of deep learning-based segmentation instance methods, which jointly exploit spectral and spatial information to improve contour reconstruction in optically complex materials. In such scenarios, the use of GPU acceleration under a window-based processing scheme will be explored to efficiently handle the increased computational demand while preserving real-time constraints. Alternatively, the modular design of the proposed architecture allows the incorporation of an additional RGB vision camera to enhance contour definition in cases where hyperspectral contrast alone is insufficient. Since the software framework separates acquisition, preprocessing, classification, and segmentation stages, the integration of such complementary inputs would not require structural modifications to the core architecture.

CRediT authorship contribution statement

Adrián Sarrías: Writing – review & editing, Visualization, Methodology, Data curation; **Miguel O. Martínez-Rach:** Writing – original draft, Validation, Investigation, Formal analysis, Conceptualization; **Otoniel López-Granado:** Writing – original draft, Software, Project administration, Conceptualization; **Héctor Migallón:** Writing – original draft, Software, Investigation, Funding acquisition.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors have no conflict of interests.

Acknowledgements

The authors would like to thank the company JOVISA S.L. for providing its facilities and its industrial prototype used for testing the proposed system. Funding: This research was partially supported by grants PID2021-123627OB-C55 and PID2024-158682OB-C33 funded by

MCIN/AEI/10.13039/501100011033 and by “ERDF A way of making Europe”.

References

[1] S.-F. Huang, S.-C. Lu, A. Mukundan, R. Karmakar, H.C. Wang, Application of hyperspectral imaging in environmental monitoring: air pollution classification and detection, Opt. Express 33 (8) (2025) 17955–17964. https://doi.org/10.1364/OE.558189

[2] M.A. Ali, X. Lyu, M.S. Ersan, F. Xiao, Critical evaluation of hyperspectral imaging technology for detection and quantification of microplastics in soil, J. Hazard. Mater. 476 (2024) 135041. https://doi.org/10.1016/j.jhazmat.2024.135041

[3] F. Arias, M. Zambrano, E. Galagarza, K. Broce, Mapping harmful algae blooms: the potential of hyperspectral imaging technologies, Remote Sens. (Basel) 17 (4) (2025) 608. https://doi.org/10.3390/rs17040608

[4] A. Mukundan, R. Karmakar, J. Jouhar, M.A.E. Valappil, H.C. Wang, Advancing urban development: applications of hyperspectral imaging in smart city innovations and sustainable solutions, Smart Cities (2025). https://doi.org/10.3390/smartcities8020051

[5] M.-F. Cheng, A. Mukundan, R. Karmakar, M.A.E. Valappil, J. Jouhar, H.C. Wang, Modern trends and recent applications of hyperspectral imaging: a review, Technologies 13 (5) (2025) 170. https://doi.org/10.3390/technologies13050170

[6] A.D.M. Africa, N.P.J. Labay, M.S.B. Ong, J.A.J. Sales, M.M.E. Toyoda, Vision-based algorithm for recyclable waste classification, 2019, (https://www.arpnjournals.org/jeas/research_papers/rp_2019/jeas_0719_7829.pdf).

[7] R. Aarthi, G. Rishma, A vision based approach to localize waste objects and geometric features exaction for robotic manipulation, Procedia Comput. Sci. 218 (2023) 1342–1352. https://doi.org/10.1016/j.procs.2023.01.113

[8] S.Y. Islam, M.G.R. Alam, Computer vision-based waste detection and classification for garbage management and recycling, in: The Fourth Industrial Revolution and beyond: Select Proceedings of IC4IR+, Springer, 2023, pp. 389–411. https://doi.org/10.1007/978-981-19-8032-9_28

[9] M. Malik, S. Sharma, M. Uddin, C.-L. Chen, C.-M. Wu, P. Soni, S. Chaudhary, Waste classification for sustainable development using image recognition with deep learning neural network models, Sustainability 14 (12) (2022) 7222. https://doi.org/10.3390/su14127222

[10] S. Poudel, P. Poudyal, Classification of waste materials using CNN based on transfer learning, in: Proceedings of the 14th Annual Meeting of the Forum for Information Retrieval Evaluation, 2022, pp. 29–33. https://doi.org/10.1145/3574318.3574345

[11] M. Koskinopoulou, F. Raptopoulos, G.D. Papadopoulos, N. Mavrakis, M. Maniadakis, Robotic waste sorting technology: toward a vision-based categorization system for the industrial robotic separation of recyclable waste, IEEE Robot. Autom. Magazine 28 (2021) 50–60. https://doi.org/10.1109/MRA.2021.3066040

[12] V. Ruiz, Á. Sánchez, J.F. Vélez, B. Raducanu, Automatic image-based waste classification, in: From Bioinspired Systems and Biomedical Applications to Machine Learning: 8th International Work-Conference on the Interplay between Natural and Artificial Computation, IWINAC 2019, Almería, Spain, June 3–7, 2019, Proceedings, Part II 8, Springer, 2019, pp. 422–431. https://doi.org/10.1007/978-3-030-19651-6_41

[13] P.L.M. Geladi, H.F. Grahn, J.E. Burger, Multivariate Images, Hyperspectral Imaging: Background and Equipment, John Wiley & Sons, Ltd, 2007, pp. 1–15. https://doi.org/10.1002/9780470010884.ch1

[14] G. Lu, B. Fei, Medical hyperspectral imaging: a review, J. Biomed. Opt. 19 (1) (2014) 010901. https://doi.org/10.1117/1.JBO.19.1.010901

[15] D.W. Sun, Hyperspectral Imaging for Food Quality Analysis and Control, Elsevier Inc., 2010. https://doi.org/10.1016/C2009-0-01853-4

[16] A. Lowe, N.A. Harrison, A.P. French, Hyperspectral image analysis techniques for the detection and classification of the early onset of plant disease and stress, Plant Methods 13 (2017). https://doi.org/10.1186/s13007-017-0233-z

[17] A. Kamilaris, F. Prenafeta Boldú, Deep learning in agriculture: a survey, Comput. Electron. Agric. 147 (2018). https://doi.org/10.1016/j.compag.2018.02.016

[18] C. Fischer, I. Kakoulli, Multispectral and hyperspectral imaging technologies in conservation: current research and potential applications, Stud. Conserv. 51 (2006) 3–16. https://doi.org/10.1179/sic.2006.51.Supplement-1.3

[19] J. Khan, H. Khan, A. Yousaf, K. Khurshid, A. Abbas, Modern trends in hyperspectral image analysis: a review, IEEE Access 6 (2018) 14118–14129. https://doi.org/10.1109/ACCESS.2018.2812999

[20] O. Tamin, E. Moung, J. Dargham, F. Yahya, S. Omatu, A review of hyperspectral imaging-based plastic waste detection state-of-the-arts, Int. J. Elect. Comput. Eng. 13 (2023) 3407–3419. https://doi.org/10.11591/ijece.v13i3.pp3407-3419

[21] G. Bonifazi, R. Palmieri, S. Serranti, Hyperspectral imaging applied to end-of-life (EOL) concrete recycling, tm - Technisches Messen 82 (2015) 616–624. https://doi.org/10.1515/teme-2015-0044

[22] S. Serranti, R. Palmieri, G. Bonifazi, R. Gasbarrone, G. Hermant, H. Bréquel, An automated classification of recycled aggregates for the evaluation of product standard compliance, Sustainability 15 (2023) 15009. https://doi.org/10.3390/su152015009

[23] N. Picone, G. Candiani, M. Colledani, M. Pepe, Hyperspectral imaging for the on-line characterization of fine mixtures in WEEE mechanical recycling systems, 2014, (https://qd-europe.com/at/en/news/product-application-news-spectrum/hyperspectral-imaging-for-the-online-characterization-of-fine-mixtures-in-weee-mechanical-recycling-systems/). Accessed: 2026-02-06.

[24] E.R. Kristensen, J. Dornonville de la Cour, T. Warburg, R.L. Eriksen, B. Jørgensen, J.E. Avery, M. Hinge, High-speed processing of hyperspectral images for enabling demanding industrial applications, Chemomet. Intell. Laborat. Syst. 267 (2025) 105531. https://doi.org/10.1016/j.chemolab.2025.105531

[25] M.L. Henriksen, J.-C. Pedersen, B.B.E. Jensen, B. Jørgensen, R.L. Eriksen, M. Hinge, A direct comparison of a next generation hyperspectral camera to state-of-the-art, Spectrochim. Acta, Part A 325 (2025) 125068. https://doi.org/10.1016/j.saa.2024.125068

[26] Q.T. Group, QT Framework 6.7, 2024, (https://www.qt.io/product/framework). Accessed: 2026-02-06.

[27] M. Vidal, J.M. Amigo, Pre-processing of hyperspectral images. essential steps before image analysis, Chemomet. Intell. Laborat. Syst. 117 (2012) 138–148. https://doi.org/10.1016/j.chemolab.2012.05.009

[28] F. Cao, Z. Yang, J. Ren, J. Mengying, W.K. Ling, Does normalization methods play a role for hyperspectral image classification?, 2017, https://doi.org/10.48550/arXiv.1710.02939

[29] Y. Li, X. Tan, W. Zhang, Q. Jiao, Y. Xu, H. Li, Y. Zou, L. Yang, Y. Fang, Research and application of several key techniques in hyperspectral image preprocessing, Front. Plant Sci. 12 (2021) 627865. https://doi.org/10.3389/fpls.2021.627865

[30] I.A. Cruz-Guerrero, R. Leon, L. Granados-Castro, H. Fabelo, S. Ortega, D.U. Campos-Delgado, G.M. Callico, Reflectance calibration with normalization correction in hyperspectral imaging, in: 2022 25th Euromicro Conference on Digital System Design (DSD), 2022, pp. 855–862. https://doi.org/10.1109/DSD57027.2022.00120

[31] D. Cozzolino, P. Williams, L. Hoffman, An overview of pre-processing methods available for hyperspectral imaging applications, Microchem. J. 193 (2023) 109129. https://doi.org/10.1016/j.microc.2023.109129

[32] F.A. Kruse, A.B. Lefkoff, J.W. Boardman, K.B. Heidebrecht, A.T. Shapiro, P.J. Barloon, A.F.H. Goetz, The spectral image processing system (SIPS)—interactive visualization and analysis of imaging spectrometer data, Remote Sens. Environ. 44 (2–3) (1993) 145–163. https://doi.org/10.1016/0034-4257(93)90013-N

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Machine Learn. Res. 15 (2014) 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html.

[34] A. Mukundan, R. Karmakar, D. Gupta, H.C. Wang, Deep learning-Based toolkit inspection: object detection and segmentation in assembly lines, Comput., Mater. Continua 86 (1) (2025) 1–23. https://doi.org/10.32604/cmc.2025.069646

[35] H. Wu, C. Li, Y. Li, Full-scale semantic segmentation of hyperspectral imaging based on spatial-spectral joint network, ISPRS Annals Photogramm., Remote Sens. Spatial Inf. Sci. X-1 (2024) 267–274. https://doi.org/10.5194/isprs-annals-X-1-2024-267-2024

[36] J. Li, H. Wang, A. Zhang, Y. Liu, Semantic segmentation of hyperspectral remote sensing images based on PSE-UNet model, Sensors 22 (24) (2022) 9678. https://doi.org/10.3390/s22249678

[37] C.P. Ribeiro, M. Castro, V. Marangozova-Martin, J.-F. Méhaut, H.C. Freitas, C.A.P.S. Martins, Evaluating CPU and memory affinity for numerical scientific multithreaded benchmarks on multi-cores, IADIS Int. J. Comput. Sci. Inf. Syst. 7 (1) (2012) 79–93. https://www.iadisportal.org/ijcsis/papers/2012140106.pdf.