Journal of Real-Time Image Processing (2022) 19:751–762
https://doi.org/10.1007/s11554-022-01222-2

ORIGINAL RESEARCH PAPER

Shelf control in retail stores via ultra-low and low power microcontrollers

M. Erkin Yücel[1] · **Cem Ünsalan**[2]

Received: 22 November 2021 / Accepted: 24 April 2022 / Published online: 26 May 2022

Abstract
Smart retail stores started to take place in our lives. Several computer vision and sensor-based systems are working together to achieve such a complex and automated operation. Besides, the retail sector has several open and challenging problems which can be solved by embedded computer vision systems. One important problem to be tackled is shelf control or stock out detection. Here, shelves in a store should be controlled regularly such that no item is missing in the shelf. In this study, we propose an embedded computer vision system to solve this problem. To do so, we frame the shelf control operation as change detection. Due to the constraints posed by the retail sector, we formed the system by an ultra-low or low power microcontroller with an embedded camera attached to it. We provided all the implementation details of the system both from software and hardware perspectives. We also tested the proposed shelf control system from different perspectives. Hence, the reader can form such a system for retail sector or for a broad class of change detection problems in which stand-alone embedded system usage is mandatory.

Keywords Shelf control · Retail sector · Change detection · Microcontroller implementation

1 Introduction

Retail sector has several open and challenging problems which can be solved by embedded computer vision systems. One such problem to be tackled is the shelf control or stock out detection. Here, shelves in a store should be controlled regularly such that no item is missing in the shelf. The stock out rate is nearly 7.9% for an average retail store. This rate can reach up to 12% for food retail stores [2]. The stocked out product also affects other items to be purchased since the customer leaves the store without buying anything when a specific item is missing. Corsten and Gruen [2] estimated that the stock out problem causes 4% sale loss on average. Therefore, retailers must always maintain sufficient stock on the shelf to maximize overall sales.

✉ Cem Ünsalan
cem.unsalan@marmara.edu.tr

M. Erkin Yücel
mehmety@migros.com.tr

[1] Migros AS, Istanbul, Turkey

[2] Department of Electrical and Electronics Engineering, Faculty of Engineering, Marmara University, Istanbul, Turkey

1.1 Literature review

Due to the importance of stock out problem and its direct effect on sales, retailers benefit from several methods. One such approach is assigning an employee to check stocks on shelves. Unfortunately, this operation is time consuming and prone to human errors. In a related approach, an employee visits shelves and acquire their images. These can be processed on cloud to check the stock status. Although this is approach minimizes human errors, time consumption remains the same as in manual control. Moreover, this approach requires data to be transferred to a remote location which may cause customer privacy breach. One recent approach is using robots (equipped with cameras and other sensors) to check shelves in the store. Here, autonomous robots visit aisles and acquire images instead of an employee. Such systems are fairly expensive and do not produce results in a short time period since the robot must visit the shelves sequentially. There are also patents related to the shelf control problem. Most of these refer to systems using RFID tags added to items. Unfortunately, RFID usage is unfeasible due to the overall cost and effort required to attach tags on items. Besides, net gain for each

item is minimal in food products. Hence, adding even a small cost via an RFID tag to the product is not feasible.

There are also several methods based on computer vision to solve the stock out problem in literature. Tonioni and Di Stefano [15] proposed a method to classify missing products on the shelf. Their system uses SIFT to find product features on the planogram. Then, it uses sub-graph matching to find product relations. George and Floerkemeier [10] used SIFT with only one training image per product for image classification. The method uses discriminative random forests, deformable dense pixel matching and genetic algorithm optimization. Rosado et al. [12] used panoramic images to find stocked out products in shelves. There are also related studies focusing on product recognition in retail environment. Jund et al. [8] uses CNN to product recognition on the shelf. Franco et al. [5] and Karlinsky et al. [9] considered the same problem. There is also a recent review paper by Santra and Mukherjee [13]. The reader can check it for both traditional and deep learning methods for automatic identification of products in retail store.

Although the mentioned computer vision methods provide fairly good results to solve the shelf control problem, they have two major shortcomings. First, these methods depend on high resolution images. Second, they need these images to be processed on powerful processors. This can only be achieved by transferring the images to a remote location. To overcome these problems, researchers proposed embedded camera systems to solve the stock out or similar problems. Frontoni et al. [6] proposed an embedded vision sensor network for planogram maintenance in retail environment using FPGA and microcontroller. The system calculates the basic image difference and sends the coordinates to the server if any difference is detected. Chen et al. [1] captured only the background image and divided it into blocks in their embedded SoC platform. Then, they find the major color in each block. Afterward, the method simply compares image pixels with major colors of blocks in real-time when an image is captured. Fan et al. [4] proposed an embedded system composed of an FPGA and low-power Himax HM01B0 camera. The acquired image is downsampled to 32×32 pixels for processing. They setup their system only to detect empty shelve locations. Milella et al. [11] used the Intel RealSense depth camera to control the shelf. To do so, they first acquire depth information when the shelf is empty. Afterward, they take images when the shelf is full. As they apply segmentation using the acquired images, they calculate the on-shelf availability ratio. Crãciunescu et al. [3] two RGB and two ToF depth cameras to scan a shelf and determine the percentage of emptiness for the products. They implement their method on the Nvidia Jetson board.

1.2 Summary of the proposed system

Before summarizing the proposed system, we should first explain the stock out problem. Therefore, we provide the sample images in Fig. 1. The first image in Fig. 1a shows a store shelf from a retail store. This image is taken by the ESP-EYE module having an OV2640 camera. We acquired the image in RGB565 format and converted it to grayscale form for display.

In Fig. 1a, there are 27 different products with a total of 45 items (with front and side views). Here, the shelf is full. Two items are taken from the shelf in Fig. 1b. One of the taken items has black color. Its background is also black. Hence, it is hard to detect this change. The second removed item has the same product in its background. Again, it is hard to detect this change. Therefore, this is a challenging stock out scenario. In Fig. 1c, the removed items are labeled by green boxes. The shelf control system should detect these changes effectively.

In this study, we formulate the shelf control as a change detection problem. The proposed solution works on low and ultra-low power microcontrollers. Therefore, the developed system can work stand-alone in the retail environment for a long time. This is mandatory since we cannot afford cabling shelves to feed power to our system in a retail store due to security reasons. With the same reasoning, it becomes impractical to use high resolution cameras requiring relatively high power to operate. Since there will be several shelves in a retail store, cost of the overall system should be low. Therefore, microcontroller usage in operation is mandatory. Besides, the acquired shelf image should be processed on the edge. Hence, no customer data are transformed to any remote location. Our system should be fast enough such that the result can be obtained in due time. This does not mean that we need a real-time system since the customer operations are not so fast.

To satisfy all the mentioned constraints, we propose a microcontroller-based system with an embedded camera attached to it. We provide the hardware block diagram of proposed system in Fig. 2. The main building block of the proposed system is the microcontroller which configures the camera module connected to it, takes the image of the shelf, and then process the image. The proposed system uses a

(a) Shelf image. (b) Items taken. (c) Changes.

Fig. 1 A sample stock out scenario

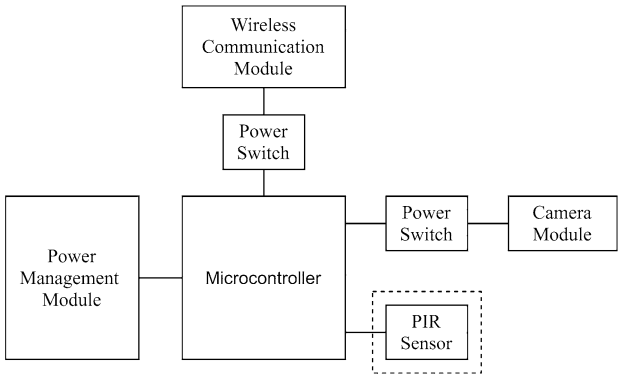

Fig. 2 Hardware block diagram of the proposed system

wireless communication module to transmit the result to a remote server.

As can be seen in Fig. 2, there is also a power management module connected to the microcontroller which supplies the necessary power to the system. It also manages the other power operations such as energy harvesting from ambient light, tracking the maximum power point of the photovoltaic (PV) cells and charging the battery of the system. The PIR sensor connected to the microcontroller enhances the power management by triggering the proposed system only when a customer passes by in day-time operations. If system runs only in specific times, the PIR sensor can be removed. The power switches in Fig. 2 are used to turn on and off the wireless and camera modules only when used hence less power is consumed from the battery. We provide the realized prototype of the proposed system in Fig. 3.

The proposed system will be located in front of the shelf with a fixed position. The first image will be taken as the store opens. Then, the system checks for the missing of misplaced items. We can summarize the detailed working principles of the proposed system as a flowchart in Fig. 4. The microcontroller, camera and wireless modules wait in low power mode (LPM). As a customer passes by the shelf or stops in front of the PIR sensor the microcontroller is triggered. Then, the microcontroller sets up a one shot timer interrupt for 10 min. The microcontroller then enters the low power mode. If another customer passes by during the 10 min interval, the PIR sensor triggers the microcontroller again. Then it resets the one shot timer. After 10 min pass, the microcontroller wakes up by the timer interrupt. Then, it

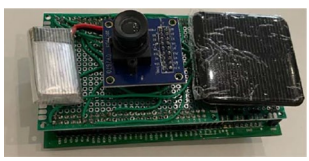

Fig. 3 Prototype of the proposed system

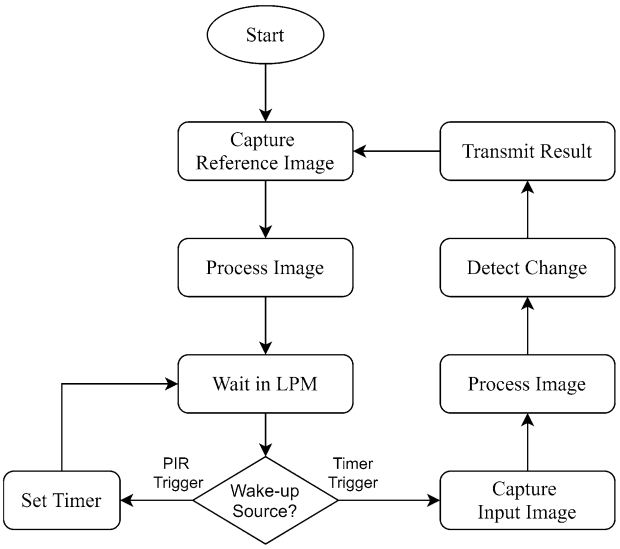

Fig. 4 Working principles of the proposed system as a flowchart

acquires the image and performs change detection. If an item is taken or misplaced, then the system detects the change and reports the result to the main server. Hence, the missing item can be refilled or misplaced item can be taken from the wrong location. Afterwards, the microcontroller acquires the shelf image one more time as the new reference image for the next operation. Then, it enters the low power mode. Via this setup, the stock-out control is delayed until no customer passes at crowded times. This reduces the power consumption. Also, if nobody walks in front of the shelf for a long time, then the operation period increases and power savings increase as well.

2 Histogram comparison for shelf control

We can use histogram information to control shelves for the following reasons. A typical ultra-low or low power microcontroller has limited RAM space. Therefore, histogram-based methods are suitable for them. Moreover, histogram is minimally affected by different viewing angles, scale, and partial occlusion. Therefore, we propose two different methods based on histogram information to control shelves. These methods are based on block-based histogram comparison for grayscale and RGB (color) images. To do so, we divide the original image into blocks and calculate the (grayscale and RGB) histogram for each block. Hence, we only keep the histogram array for the selected number of blocks. When a new image is acquired, it is divided into blocks and histogram of each block is calculated again. We obtain block-based histogram information for grayscale images in a straightforward manner. In a separate setup, we also

calculate the block-based histogram for RGB images. To do so, we benefit from the method proposed by Swain and Ballard [14].

In implementation, we used non overlapping blocks since overlapping block usage increases RAM requirements. We picked the block size as 10×10 pixels for an image with size 160×120 pixels. Therefore, we have a total of 192 blocks for a given image. We obtained the block size such that it can capture an average item in our data set.

As the block based histogram is obtained (for the grayscale or RGB image), we should use histogram comparison methods to check whether any change has occurred on the shelf. To do so, we need a method to compare histograms. Hence, the similarity or distance between histograms can be obtained. There are several histogram comparison methods based on intersection, correlation, or distance for this purpose. Among these, histogram intersection is a simple yet relatively robust method. In this method, similarity between two image blocks is defined as intersection of their normalized histograms containing N bins [14] as

$$H(h_i[k], h_r[k]) = \frac{\sum_{b=1}^{N} \min\left(h_i[k](b), h_r[k](b)\right)}{\sum_{b=1}^{N} h_r[k](b)} \quad (1)$$

where $h_i[k]$ and $h_r[k]$ are the histogram of the k-th blocks of input and reference images, respectively. Here, the higher the value of the histogram intersection, the more similar the compared two image blocks are.

As for shelf control, we can apply the following strategy. If the similarity of two image blocks is above a preset threshold, then we can assume that there is no change. Otherwise, we can assume that there is a change in that image block location. Hence, we modified the method proposed by Swain and Ballard to work in a block-based setup in this study.

In the block-based histogram method, we form a histogram for a given block at hand with 16 entries each with two bytes for a grayscale image. Hence, the memory requirement for one grayscale image becomes $2 \times 16 \times 192 = 6144$ bytes. We will need two histogram arrays for the reference and input images separately. Therefore, we will need a total of 12,288 bytes to store the histograms. Moreover, we will need a 192 bytes array to store the obtained output. For an RGB image, we form a histogram for a given block with 64 bins (four bits for R $\times$ four bits for G $\times$ four bits for B) each with two bytes. Hence, the memory requirement for one RGB image becomes $2 \times 64 \times 192 =$ 24,576 bytes. As in the grayscale case, we will need two histogram arrays for the reference and input images separately. Therefore, we will need a total of 49,152 bytes to store histograms. Moreover, we will need a 192 bytes array to store the obtained output.

3 Pixel-based change detection methods for shelf control

The camera used in our shelf control system has a fixed position. Moreover, lighting is fairly constant since we operate inside the store. Therefore, pixel-based change detection methods can be used for shelf control. There are different pixel-based change detection methods in literature [7]. Some of them are suitable for shelf control with low and ultra-low power microcontrollers. In this study, we applied six different pixel-based change detection methods for shelf control. Due to memory requirements, we only applied these on grayscale images. We explain each method in detail next.

3.1 Basic image differencing on the fly

Our first pixel-based change detection method is based on basic image differencing. We can form the absolute difference between the reference, I_r, and input images, I_i, as $I_d(x, y) = \left|I_r(x, y) - I_i(x, y)\right|$ on the fly. Then, we can apply a threshold, τ, to detect whether a pixel location represents a change. Here, we do not store the input image. Instead, all operations are done on the fly as we capture an image from the camera. Hence, our main modification of the basic image differencing is that, we modify it to run on the fly. Moreover, we store the output image on top of the reference image. As a result, we only need one image sized RAM space for the overall operation. For an image with size 160×120 pixels, this corresponds to 19,200 bytes.

3.2 Image differencing with preprocessing

Image differencing may be vulnerable to noise terms originating from the image acquisition step. To minimize this effect, the input image can be preprocessed. This becomes our second pixel-based change detection method.

We can explain the image differencing with preprocessing method as follows. First, we convolve the reference image with a simple Gaussian blurring filter to eliminate the noise terms as $\tilde{I}_r(x, y) = G * I_r(x, y)$. We acquire the input image as the trigger comes. For each pixel in this image, we apply filtering to it (taking its neighbors into account) and obtain $\tilde{I}_i(x, y) = G * I_i(x, y)$ at location (x, y). Then, we apply the image differencing operation as $I_d(x, y) = \left|\tilde{I}_r(x, y) - \tilde{I}_i(x, y)\right|$ to check whether there is a change at the location (x, y). This setup is done to speed up the operation. This is where we improved the basic image differencing method. Here, we store the filtered reference image and raw input image in RAM. There is no negative value in the filtered image. Besides, the stored image is normalized such that it can be represented in `uint8_t` format. We also store the input

image this way. As in the first method, the output image is stored on top of the reference image to save RAM space. Therefore, we need 38,400 bytes of RAM space to store two images with size 160×120 pixels.

3.3 Image rationing with preprocessing

The third pixel-based change detection method is based on image rationing [7]. In this method, we check the ratio of image pixels instead of their difference. We also apply preprocessing to the images to be used in operation. The ratio of the two images is normalized using the arctangent function. We can take the absolute value of the result to obtain the values to be in the range $[0, \pi/4]$. This can be done by applying the formula

$$\tilde{I}_{\text{ratio}}(x,y) = \left| \arctan\left(\frac{\tilde{I}_r(x,y)}{\tilde{I}_i(x,y)} \right) - \frac{\pi}{4} \right| \tag{2}$$

Hence, the result will be close to zero when there is no change. If the result is close to $\pi/4$, then this indicates maximum change. The image rationing with preprocessing method emphasizes changes in low intensity image regions. Therefore, it can be used to detect changes in shelves with low illumination.

In implementation, we apply the following steps. We first obtain the filtered reference image $\tilde{I}_r(x,y) = G * I_r(x,y)$. This is the main difference between the existing and proposed method. We acquire the input image as the trigger comes. For each pixel in this image, we apply filtering to it (taking its neighbors into account) and obtain $\tilde{I}_i(x,y) = G * I_i(x,y)$ at location (x, y). Then, we apply the image rationing operation as in Eq. (2). This way we can check whether there is a change at the location (x, y). This setup is done to speed up the operation. Here, we store the reference and input images in RAM. There is no negative value in the filtered image. Besides, the stored image is normalized such that it can be represented in `uint8_t` format. We also store the input image this way. As in the previous two methods, we store the obtained output on top of the reference image. Hence, we need 38,400 bytes to store two images each with size 160×120 execute the code.

3.4 Modelling each pixel value in time by a Gaussian pdf

The fourth pixel-based change detection method is based on modelling each pixel value in time by a Gaussian probability density function (pdf). This method is the modified form of the background modeling approach proposed by Wren et al. [20] for object tracking. The constraints given there fit well with our shelf control problem such that we have constant illumination and the camera viewing angle is fixed.

We only benefit from the pdf modelling method proposed by Wren et al. for our own problem. In the proposed method, the mean (μ) and standard deviation (σ) representing the Gaussian pdf for each pixel is stored. If the captured pixel is not within $\mu \pm 3\sigma$, then the pixel is marked as changed. While capturing the image over time, μ and σ for each pixel can be updated using Welford's [19] online algorithm as

$$\mu_n = \mu_{n-1} + \frac{i - \mu_{n-1}}{n} \tag{3}$$

$$\sigma_n = \sqrt{\frac{\sigma_{n-1}^2 (n-1) + (i - \mu_n)(i - \mu_{n-1})}{n}} \tag{4}$$

where i is the new grayscale pixel intensity value, μ_n and σ_n are the new values, μ_{n-1} and σ_{n-1} are the old values.

In implementation, we should store the mean and standard deviation for each pixel location in the reference image. Afterward, we update these values as we acquire a new image. Therefore, we do not store the input image during operation. We only store the output as a new image. Hence, we should allocate $3 \times 160 \times 120$ bytes in RAM for a 160×120 image. To note here, we stored all the values in `uint8_t` format in operation. To do so, we normalized the mean and standard deviation values.

3.5 Block-based basic image differencing

The previous pixel-based change detection methods have two shortcomings. First, each pixel value (or mean and standard deviation per pixel) should be stored in RAM. Second, noise may affect the pixel value directly. To overcome these problems, Chen et al. [1] proposed block-based background image modelling. In this method, only major colors of each block are stored instead of storing the reference image completely.

We propose the fifth pixel-based change detection method this way. Different from Chen et al., we used major output instead of major color to improve the performance. This is the modified form of basic image differencing with the usage of blocks. To do so, we first obtain the difference image and obtain the changed pixels. Then, we divide the output image into non-overlapping blocks of size 10×10 pixels. Hence, we form a total of $16 \times 12 = 192$ blocks for an image with size 160×120 pixels. We check whether the ratio of changed pixels is greater than 30% for the block. If this is the case, then we take that block as changed. Otherwise, we assume that there is no change for the given block. This allows us to minimize the effect of noise. Besides, we were able to fill the empty regions in a block. These positively affect the shelf control detection performance. In implementation, we only need an extra array with the size of block numbers each keeping two bytes besides

the ones used in image differencing or rationing. For our case, this becomes an extra $2 \times 192 = 384$ bytes.

3.6 Block-based image rationing

As in the block-based basic image differencing method, we can extend the image rationing with preprocessing method by a block-based approach. This becomes our sixth and final pixel-based change detection method to be used for shelf control. To do so, we first obtain the ratio image and obtain the changed pixels. Then, we divide the output image into non-overlapping blocks as in the previous section. We check whether the ratio of changed pixels is greater than 30% for the block. If this is the case, then we take that block as changed. Otherwise, we assume that there is no change for the given block.

4 Edge-based change detection method for shelf control

Edge information in an image can also be used for shelf control. Therefore, we benefit from the edge-based change detection operation proposed in [7]. Different from there, we have a fixed camera with fairly constant illumination conditions. Hence, we modified this method for shelf control as follows. We use the Sobel edge detector instead of Canny edge detector. We calculate the edge difference instead of overlap of connected components.

In extracting edges from the reference and input images, we first apply blurring to them with a Gaussian kernel as in pixel-based change detection. Then, we use Sobel filters with two 3×3 kernels. We convolve these kernels with the reference and input images separately and calculate the gradient magnitude for each pixel. Then, we take the difference of gradient magnitudes and threshold the result to obtain the changed pixels. This way, we can eliminate the effect of weak edges in images. Due to RAM requirements, we could not apply advanced edge detection methods here. We store the reference and input images in operation. We should also store the convolution results for a given image. As in previous implementations, we normalize the Gaussian filtering results and gradient magnitude values such that they are represented by `uint8_t` format. As a result, we store three images in RAM with size 160×120 pixels. Hence, we need a total of $3 \times 160 \times 120 = 57{,}600$ bytes of RAM space.

5 Hardware and software platforms used in implementation

The aim of this study is implementing retail shelf control via low and ultra-low power microcontrollers with an embedded camera attached to them. Therefore, we introduce the microcontrollers and their programming environment used in implementation in this section. We also compare these microcontrollers. Hence, the reader can understand their limitations and advantages when used in solving the shelf control problem.

5.1 The MSP430 and MSP432 microcontrollers

We used the MSP430 and MSP432 microcontrollers from Texas Instruments as the first set of implementation platform. The first microcontroller has the exact name MSP430FR5994. This is a 16-bit microcontroller with RISC architecture [16]. The exact name for the second microcontroller is MSP432P401R. This is a 32-bit microcontroller with Arm Cortex-M4F architecture [17]. We should mention here that the work presented here is the prototyping step of the actual system to be developed. Therefore, we used the MSP-EXP430FR5994 and MSP-EXP432P401R LaunchPads in our implementation. We used the Code Composer Studio (CCS) platform to program the MSP430 and MSP432 microcontrollers. We benefit from the DriverLib library which provides functions to control the MSP microcontrollers.

5.2 The STM32G0, STM32L4, and STM32F4 microcontrollers

We used the STM32G0, STM32L4 and STM32F4 microcontrollers from STMicroelectronics as the second set of implementation platform. The first microcontroller has the exact name STM32G071RBT6. This is a 32-bit microcontroller with Arm Cortex-M0F architecture. The exact name for the second and third microcontrollers are STM32L4R5ZIT6 and STM32F429ZIT6, respectively. These are 32-bit microcontrollers with Arm Cortex-M4F architecture [18]. We used the NUCLEO-G071RB, NUCLEO-L4R5ZI and 32F429IDISCOVERY boards to benefit from the STM32G0, STM32L4, and STM32F4 microcontrollers, respectively. We benefit from the STM32CubeIDE platform to program and debug STM32 microcontrollers. We used the HAL library in implementation which consists of API functions to control the STM32 microcontroller.

5.3 Comparison of the used microcontrollers

As explained in previous sections, we picked five different microcontrollers to represent a wide spectrum of low and ultra-low power microcontrollers. We tabulate their comparison results in Table 1. Price for each microcontroller is acquired from the Digi-Key Electronics website as the major supplier with all considered microcontrollers available. To be fair, we tabulate the unit price when 1000 items are purchased.

There is an FRAM in the MSP430 microcontroller which can be used as RAM. Therefore, we indicated the RAM size for this microcontroller as 8 + 256 KB in Table 1. Likewise, the STM32F4 microcontroller has 192 SRAM + 64 CCRAM. Hence, we indicated this as 192 + 64 KB as available RAM in Table 1.

We can group the MSP430 and STM32G0 microcontrollers in the ultra-low power category in Table 1. Likewise, we can group the MSP432, STM32L4, and STM32F4 microcontrollers as low power category. The microcontrollers in the ultra-low power category are generally slow (has less core speed), has low flash and RAM size. As expected, their price is also low. On the other hand, low-power microcontrollers are faster compared to ultra-low power microcontrollers. They have relatively high flash and RAM size. As expected, their price is also high. Although this comparison provides insight on the general properties of selected microcontrollers, we will compare them on the shelf control problem in Sect. 7. Hence, the reader will be able to pick the best microcontroller for his or her shelf control system.

6 Image acquisition methods

Besides the microcontroller, we will need an image acquisition module for our shelf control system. We have two options here. The first one is acquiring the image from an embedded camera. This is the natural way for a stand-alone system. The second method is feeding images from PC to the microcontroller via UART interface. This method is useful while testing algorithms before forming the final stand-alone system. Next, we will briefly explain both methods. For more information on them, please see [18].

6.1 Image acquisition via digital camera

We can acquire image of the shelf to be controlled by an embedded camera. The camera to be used for this purpose should be low cost, easy to acquire, and easy to use with our microcontrollers. The best fit for these constraints is the OV7670 camera module (with and without FIFO buffer).

Table 1 Comparison of the selected microcontrollers

Microcontroller	Core speed (MHz)	Flash (KB)	RAM (KB)	Price ($)
MSP430	16	256	8 + 256	4.21
MSP432	48	256	64	4.38
STM32G0	64	128	32	1.90
STM32L4	120	1024	640	8.54
STM32F4	180	2048	192 + 64	9.74

This is a low cost (can be found under $5) module with a CMOS camera. It can capture up to 640×480 pixel resolution images, pre-process and send them to a host microcontroller using its output pins. In this study, we used the OV7670 camera module with FIFO buffer. This module captures an image and saves it to its AL422b 380 KB FIFO buffer first. Then, the microcontroller can access the buffer and read the image array. The OV7670 camera module with FIFO buffer also has an internal crystal oscillator with 24 MHz clock speed. Hence, it does not require a fast microcontroller to capture the image.

6.2 Image transfer from PC through UART interface

We can transfer images between the PC and microcontroller through UART interface. This option is extremely useful during the algorithm development and testing phases. We prepared a Python script that runs on PC to send and receive images between the microcontroller and PC for this purpose. This script runs as follows. Once the microcontroller sends the start byte to PC, transfer sequence begins. The timeout for sending and receiving data is determined as two seconds. If this time passes without any data transfer, the microcontroller and PC discards transfer and return to their initial state. In this study, we set the UART communication speed to 921,600 bits per second (bps).

7 Experiments

We test the proposed shelf control methods in this section. As a reminder, the main focus of this study is developing methods suitable to be implemented on resource constrained ultra-low and low power microcontrollers. Hence, we report the performance of the proposed methods based on detection and timing performance, memory usage, and power consumption. We also provide comparison of the proposed method with commercial systems on an actual scenario in this section.

7.1 Dataset used in experiments

There are several datasets available in literature for shelf control and related problems. Images in these are acquired by high resolution digital cameras. However, we aim to develop a stand-alone embedded shelf control system based on ultra-low and low power microcontrollers with an appropriate embedded camera attached to them. Hence, high resolution images do not match with the proposed hardware used in this study. Therefore, we formed our own dataset.

We used the ESP-EYE module to acquire images in the dataset. As a reminder, this module has a 2 MP OV2640 camera sensor. Hence, it provides an image close to the one

with the OV7670 camera module. Due to the available RAM in our microcontrollers, we acquired images with size 160 × 120 pixels. We benefit from the Wi-Fi of the ESP-EYE module and formed our setup as such. We set the module on a tripod in front of the shelf to be controlled. As a side note, the ESP-EYE module (more specifically the SoC it contains) has high power consumption and cannot be taken as a low-power microcontroller. Therefore, we did not include it in our tests.

While forming our dataset, we picked nine different retail product types. We grouped them into three categories as easy (toilet paper, liquid oil, softener), medium (milk, fruit juice, coke), and hard (cereal, pasta, detergent) based on their packet type and size. While forming the dataset, we first acquired the shelf image when it is full. Then, we picked one item from the shelf and acquired another image. We did not put the item back to the shelf. Furthermore, we acquired another item from the shelf and acquired the next image.

We acquired 100 images from all retail groups. The total number of picked images is 90 in all these images. While testing our methods, we need one reference and one input image. Since we update the reference image at every item removal, we expanded our dataset such that we have a total of 470 item change scenarios. Total product count indicates total number of items in the shelf. We divided our dataset into training and test groups. The training group is formed by 20% of the total number of change scenarios. Hence, we have 94 image pairs for training. These are used to adjust the parameters in our shelf control methods. Likewise, the test group is formed by 80% of the change scenarios. Therefore, we have 376 image pairs for testing. We next report our performance results on these.

7.2 Detection performance comparison of the proposed methods

We provide detection performance of the proposed shelf control methods on our dataset in this section. To tabulate the results, we formed abbreviations for the proposed methods in Table 2. We next tabulate all the following results with these abbreviations.

We first considered pixel-wise performance for the proposed shelf control methods. To do so, we checked the pixel level changes between the reference and input images. We provide the obtained results in Table 3.

As can be seen in Table 3, the best performing method based on the F1score is P6. Besides, P2, P3, P5, and P6 methods have similar performance values. As can be seen in the table, applying filtering to the differencing operation improved the performance. Hence, P2 has better performance compared to P1. In general, block-based methods has better F1 score values compared to their initial version. However, their recall rates decreased since the selected block size is larger than the object size. The E1 method has an average result in general. Block-based methods (H1 and H2) could not perform well as with other methods. To be more specific, H1 had the worst performance in pixel-based tests.

Table 2 Proposed methods and their abbreviation

Method	Abbr.
Block-based histogram comparison (Grayscale)	H1
Block-based histogram comparison (RGB)	H2
Basic image differencing on the fly	P1
Image differencing with preprocessing	P2
Image rationing with preprocessing	P3
Pixel-wise Gaussian pdf	P4
Block-based image differencing	P5
Block-based image rationing	P6
Edge differencing	E1

Table 3 Pixel-wise performance for the proposed shelf control methods

Metric/method	Precision	Recall	F1 score
H1	0.475	0.668	0.555
H2	0.535	0.649	0.587
P1	0.557	0.634	0.593
P2	0.721	0.655	0.687
P3	0.651	0.717	0.682
P4	0.531	0.605	0.566
P5	0.656	0.722	0.687
P6	0.632	0.773	0.696
E1	0.591	0.602	0.596

We next consider object-wise performance for the proposed shelf control methods. To do so, we checked the object level changes between the reference and input images. We assume that an object has been taken from the shelf when at least 30% of its pixels indicate a change. We provide the obtained results this way in Table 4. As can be seen in this table, the best performing methods based on the F1 score are P2 and P3. The E1 method has a comparable result. Block based methods could not improve performance of their initial version.

7.3 Timing performance comparison of the proposed methods

In this section, we analyzed the overall timing (from image acquisition to final decision-making, including all steps) of each method on selected microcontrollers. We tabulated the results in Table 5. In this table, the values are calculated while the microcontroller is working at its maximum CPU clock speed.

Table 4 Object-wise performance for the proposed shelf control methods

Metric/method	Precision	Recall	F1 score
H1	0.757	0.730	0.743
H2	0.685	0.777	0.728
P1	0.767	0.945	0.847
P2	0.910	0.943	0.926
P3	0.965	0.907	0.935
P4	0.728	0.911	0.809
P5	0.844	0.783	0.813
P6	0.859	0.833	0.846
E1	0.911	0.871	0.891

As can be seen in Table 5, the slowest microcontroller is MSP430 for all shelf control methods in general. This is expected since the CPU clock speed of this microcontroller is the slowest one. The fastest methods are block-based histogram comparison (H1) and basic image differencing (P1). As one image is acquired from the camera, their operations end. The block-based histogram comparison (H2) is slower than these two methods. However, it is still fast compared to others since we acquire the RGB image from the camera in RGB565 form, two bytes per pixel compared to one byte per pixel for grayscale images. The pixel-wise Gaussian pdf (P4) method is slower than the first three methods since we calculate the mean and standard deviation values. The preprocessing filter in the image differencing with preprocessing (P2) slows the method. The image rationing with preprocessing (P3) method is slower than the previous methods due to the division and arctangent operation. Block-based operations (P5 and P6) are slower since calculations on blocks slow down the methods by approximately 3%. The edge differencing method (E1) is the slowest of all methods since it requires multiple convolution operations.

Table 5 Timing (ms) of the methods for each microcontroller

MCU/method	MSP 430	MSP 432	STM 32G0	STM 32L4	STM 32F4
H1	8291	684	574	656	482
H2	11,440	1049	–	674	565
P1	5000	680	560	560	480
P2	13,322	1331	–	810	730
P3	14,451	3148	–	1260	971
P4	6260	2140	–	820	367
P5	18,069	1446	–	855	760
P6	19,198	3263	–	1305	1001
E1	26,963	4097	–	1645	1243

7.4 Memory usage comparison of the shelf control methods

Since we are using ultra-low and low power microcontrollers for shelf control, we should take the RAM and flash memory usage of each method into account. Therefore, we analyzed these in two separate sections next. As representative cases, we picked the MSP430 and STM32L4 microcontrollers for comparison. The remaining Arm Cortex M based microcontrollers have similar memory usage as with these.

7.4.1 RAM usage

We provide the RAM usage for all the proposed shelf control methods on the MSP430 and STM32L4 microcontrollers in Table 6. As can be seen in this table, there are minor changes in RAM usage between the MSP430 and STM32L4 microcontrollers. This is because of the difference between the RAM usage properties between DriverLib and HAL libraries for the two microcontrollers, respectively.

As can be seen in Table 6, the H1 method requires the least RAM space for operation since it does not need the image to operate. The H2 method requires more RAM space due to the selected block and binning size in operation. We can decrease the RAM space by adjusting these parameters. However, the detection performance will also decrease in this case. The P1 method requires RAM space to store images. Moreover, the P2 and P3 methods need extra RAM space due to the filtering operation. Therefore, they need at least twice the RAM space compared to P1. The P4 method needs RAM space to store the mean (in `int8` format) and standard deviation (in `float` format). The P5 and P6 methods need extra RAM space for the size of blocks used. The E1 method requires the highest RAM space since it needs to store the output of convolution operations.

Table 6 RAM usage (in KB) of the methods for each microcontroller

MCU/method	MSP430	STM32L4
H1	12.36	13.88
H2	48.36	49.79
P1	19.01	20.57
P2	37.69	39.21
P3	37.68	39.21
P4	56.42	58.05
P5	38.06	39.59
P6	38.06	39.59
E1	56.43	58.05

7.4.2 Flash memory usage

We next tabulate the flash memory requirements for each shelf control method. As in the previous section, we only tabulate the flash memory usage on the MSP430 and STM32L4 microcontrollers. We provided the results in Table 7. As can be seen in this table, there are significant changes in flash memory usage between the MSP430 and STM32L4 microcontrollers. The main reason for this is that, most of the flash memory is used by the DriverLib and HAL libraries. This also explains why different shelf control methods have similar flash memory usage when one microcontroller is picked. Table 7 also indicates that the proposed methods for shelf control are suitable to be used in microcontrollers having low flash memory.

7.5 Power consumption comparison of the proposed methods

As explained in Sect. 1, the aim of this study is forming an embedded system for shelf control. One of the most important properties of this system is that it should depend on battery due to working conditions. Therefore, we tested the average current requirement of each shelf control method running on different microcontrollers. The working scenario for the provided results is that each microcontroller wakes up periodically every 10 min. The shelf check operation is performed. The result is transmitted to a remote server. Then, the microcontroller goes to low power mode. To be more specific, the MSP430 microcontroller goes to LPM4 and Arm Cortex M-based microcontrollers go to stop mode.

Based on the set working scenario, we provide the average current requirement for each method (except wireless transfer) in Table 8. In this table, the microcontrollers are set to work in maximum CPU clock speed. As can be seen in this table, the MSP432 microcontroller has the lowest average current consumption on all the methods. The STM32G0 microcontroller has the lowest current consumption on the methods it is working on.

As we compare the shelf control methods in Table 8, we can observe that the P1 method has the lowest average current consumption for all microcontrollers it is working on. This is expected since the faster method needs less power on the same hardware. Therefore, the reader should also consider Table 5 while considering the average current consumption and the battery life to be considered next.

We should consider the effect of wireless transfer in operation. We can pick the ESP8266 Wi-Fi module for this purpose. Hence, we will have an additional 300 μA current consumption for each value tabulated in Table 8.

The average current may not emphasize the power consumption of the proposed shelf control system. Therefore, we also analyzed the power consumption based on how many days the system can operate when two AA batteries (with a total of 3000 mAh) are used as the power source for the system. We provide the computation results (in days) in Table 9. In this table, we also take the wireless transfer operation into account.

The wireless transfer operation consumes so much power that it minimizes the effect of low and ultra-low power microcontrollers. Even in this scenario, we still have working systems with duration between 180 and 383 days as can be seen in Table 9. This duration can be enhanced by adding an energy harvesting module (by solar cells) to the system. We tested such a system using a 110 × 70 mm solar panel in our operation. Under average 500 lux lighting, this harvesting module can feed 0.471 mW power. Hence, 180 and 383 day operation durations become 233 and 750 days, respectively. This further improves our system performance in terms of its power consumption.

Table 7 Flash memory usage (in KB) of the methods for each microcontroller

MCU/method	MSP430	STM32L4
H1	5.03	15.50
H2	5.31	15.65
P1	7.95	22.81
P2	5.50	16.54
P3	11.00	18.89
P4	10.51	18.15
P5	6.02	16.96
P6	12.53	17.00
E1	10.12	19.15

Table 8 Average current (in μA) of the methods for each microcontroller

MCU/method	MSP 430	MSP 432	STM 32G0	STM 32L4	STM 32F4
H1	288.2	30.1	26.9	50.4	183.2
H2	395.8	45.5	–	51.6	197.5
P1	174.6	30.0	26.7	43.5	183.0
P2	231.6	39.9	–	54.7	221.4
P3	239.4	67.3	–	75.0	256.3
P4	218.6	93.0	–	62.5	163.4
P5	264.2	41.6	–	56.8	225.8
P6	271.9	69.1	–	77.0	260.7
E1	325.1	81.7	–	92.4	295.7

Table 9 Battery life (in days) of the methods for each microcontroller

MCU/method	MSP 430	MSP 432	STM 32G0	STM 32L4	STM 32F4
H1	213	379	382	357	259
H2	180	362	–	356	251
P1	263	379	383	364	255
P2	235	368	–	352	240
P3	232	340	–	333	225
P4	241	318	–	345	270
P5	222	366	–	350	238
P6	219	339	–	332	223
E1	200	327	–	319	210

7.6 Comparison of the proposed method with commercial systems on an actual scenario

We compare the proposed method with the available commercial systems in this section. While doing so, we pick an actual scenario from a branch (store) of Migros, one of the leading retail chains in Turkey. This branch has 1191 m^2 sales area with 292 modular shelves. We can exclude the fruits, vegetables, and meat departments in our calculations since they require specific operations. Approximate length for a department is 10 m. There are a total of ten departments. Each department has two shelf blocks, each having a total of six shelves. Hence, the total shelf length for the store can be taken as 1200 m.

First, we can analyze the cost of the proposed system used in the selected store for shelf control. One OV7670 camera can cover an average of 2-m-wide shelf block. Hence, it can cover a total of 12-m shelf area. Therefore, we will need 100 of our modules to cover all the shelves in the store. We provided the cost of microcontrollers used in the proposed system in Table 1. We will need additional equipment (rechargeable battery, solar panel, PMIC module, other electronic equipment, and boxing) to form our system. Hence, the total price of 100 modules for the store becomes \$2421, \$2438, \$2190, \$2854, and \$2974 for the MSP430, MSP432, STM32G0, STM32L4, and STM32F4 microcontrollers, respectively.

Second, we can form a shelf control method with IP cameras. A 2 MP IP camera costs around \$50. On average, it can cover a 3-m-wide shelf block. As a result, we will need a total of 67 IP cameras to cover all the shelves in the store. We will have additional costs as three 24 port PoE Network switch and 2000 m ethernet cable. Taking these into account the total cost becomes \$8200. Here, we assumed that the cloud server belongs to us. Hence, we did not add its price in the calculation.

Third, we can use a moving robot to cover all the shelves in the store. There is no fixed price for the robot solution. The vendors set the price based on the store sales area and its plan. The approximate price for a robot is between \$80,000–\$100,000. Besides, a license fee should be paid to the vendor as long as the robot works in the store.

Based on the comparison given in this section, we can claim that the proposed system outperforms the currently available commercial solutions. One can argue that IP camera and robot based solutions can be used to solve more complex retail problems such as planogram check. We should mention that the next step in our work covers this area as well. At this step, we can safely say that with a slight increase on the microcontroller and camera cost, the planogram check can also be solved with the same system.

8 Final comments

This study proposes an embedded computer vision system to solve the shelf control problem in retail sector. We framed the problem as change detection and applied the methods suitable to be implemented on ultra-low and low power microcontrollers. To do so, we modified and enhanced the selected methods such that they became suitable for shelf control. We tested the proposed shelf control system from different perspectives. Based on the obtained results, we can suggest the following setups. If the price per module is important, the reader can pick the STM32G0 microcontroller with the basic image differencing on the fly method running on it. This setup costs \$2190 for 100 units. It has a 0.847 F1 score in detection. It can work 383 days without the energy harvesting module attached to the system. If the detection performance is important, the reader can pick the MSP432 microcontroller with the image differencing with preprocessing method running on it. This setup costs \$2438 for 100 units. It has a 0.926 F1 score in detection. It can work 363 days without the energy harvesting module attached to the system. This system can be extended further by forming a camera network system such that shelf control problem can be solved more efficiently. Besides, the next important problem in retail sector called planogram check can also be solved with the same hardware setup. Finally, we should emphasize that the proposed system can be applied to a broader class of change detection problems in which standalone embedded computer vision system usage is mandatory.

Acknowledgements This work is supported by TUBITAK under project no 5190042.

References

1. Chen, W., Sheu, M., Lin, C.: Block-based major color method for foreground object detection on embedded SoC platforms. IEEE Embed. Syst. Lett. **4**(2), 49–52 (2012)

2. Corsten, D., Gruen, T.: Stock-Outs Cause Walkouts, vol. 82. University of St.Gallen, St.Gallen (2004)
3. Craciunescu, M., Baicu, D., Mocanu, S., Dobre, C.: Determining on-shelf availability based on rgb and tof depth cameras. In: 23rd International Conference on Control Systems and Computer Science, pp. 243–248 (2021)
4. Fan, X., Yan, Y., Yang, P., Han, F.: CMSS: Use low-power iot cameras to monitor store shelves. In: 7th International Conference on Big Data Computing and Communications, pp. 309–315 (2021)
5. Franco, A., Maltoni, D., Papi, S.: Grocery product detection and recognition. Expert Syst. Appl. **81**, 163–176 (2017)
6. Frontoni, E., Mancini, A., Zingaretti, P.: Embedded vision sensor network for planogram maintenance in retail environments. Sensors **15**, 21114–21133 (2015)
7. Ilsever, M., Unsalan, C.: Two-Dimensional Change Detection Methods—Remote Sensing Applications. Springer Briefs in Computer Science. Springer, Berlin (2012)
8. Jund, P., Abdo, N., Eitel, A., Burgard, W.: The Freiburg groceries dataset. arXiv:1611.05799 (2016)
9. Karlinsky, L., Shtok, J., Tzur, Y., Tzadok, A.: Fine-grained recognition of thousands of object categories with single-example training. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 965–974 (2017)
10. Marian, G., Floerkemeier, C.: Recognizing products: a per-exemplar multi-label image classification approach. In: ECCV 2014, pp. 440–455 (2014)
11. Milella, A., Petitti, A., Marani, R., Cicirelli, G., D'orazio, T.: Towards intelligent retail: automated on-shelf availability estimation using a depth camera. IEEE Access pp. 19353–19363 (2020)
12. Rosado, L., Goncalves, J., Costa, J., Ribeiro, D., Soares, F.: Supervised learning for out-of-stock detection in panoramas of retail shelves. In: International Conference on Imaging Systems and Techniques, pp. 406–411 (2016)
13. Santra, B., Mukherjee, D.P.: A comprehensive survey on computer vision based approaches for automatic identification of products in retail store. Image Vis. Comput. **86**, 45–63 (2019)
14. Swain, M.J., Ballard, D.H.: Color indexing. Int. J. Comput. Vis. **7**, 11–32 (1991)
15. Tonioni, A., Di Stefano, L.: Product recognition in store shelves as a sub-graph isomorphism problem. In: Image Analysis and Processing-ICIAP 2017, pp. 682–693 (2017)
16. Ünsalan, C., Gürhan, H.D.: Programmable Microcontrollers with Applications: MSP430 LaunchPad with CCS and Grace. McGraw-Hill, New York (2013)
17. Ünsalan, C., Gürhan, H.D., Yücel, M.E.: Programmable Microcontrollers: Applications on the MSP432 LaunchPad. McGraw-Hill, New York (2017)
18. Ünsalan, C., Gürhan, H.D., Yücel, M.E.: Embedded System Design with ARM Cortex-M Microcontrollers: Applications with C and MicroPython. Springer Nature, London (2022)
19. Welford, B.P.: Note on a method for calculating corrected sums of squares and products. Technometrics **4**, 419–420 (1962)
20. Wren, C.R., Azarbayejani, A., Darrell, T., Pentland, A.P.: Pfinder: real-time tracking of the human body. IEEE Trans. Pattern Anal. Mach. Intell. **19**, 780–785 (1997)

Cem Ünsalan is a full professor at the Department of Electrical and Electronics Engineering at Marmara University, Turkey since 2017. Prior to joining Marmara University, he worked the Department of Electrical and Electronics Engineering at Yeditepe University for 15 years as a faculty member. He also served as the department head for 4 years there. Dr. Ünsalan received his BSc degree (1995) from Hacettepe University, Turkey; his MSc degree from Bogazici University, Turkey (1998); and PhD degree (2003) from The Ohio State University, USA. Dr. Ünsalan has published three books on microcontrollers and FPGA systems. The two microcontroller books are on Texas Instrument's MSP430 and MSP432 architecture. The FPGA book is on Xilinx's Artix7 FPGA chip. All these books aim to introduce basic embedded system concepts through practical applications. He has also published books on computer vision and image understanding.