Materials Today: Proceedings 47 (2021) 3221–3226

Contents lists available at ScienceDirect

Materials Today: Proceedings

journal homepage: www.elsevier.com/locate/matpr

Object detection and position tracking in real time using Raspberry Pi

Gokulnath Anand, Ashok Kumar Kumawat *

Mechanical Department, Manipal University Jaipur, Dehmi Kalan, Jaipur, Rajasthan 303007, India

ARTICLE INFO

Article history:
Available online 10 July 2021

Keywords:
OpenCV
Position
Object detection
Displacement
Automation

ABSTRACT

One of the fast-growing areas of deep learning using artificial intelligence is computer vision, becoming increasingly popular. It is a growing field of research that seeks to create techniques to help computers "see" and recognize the information of digital images for instance videos and photographs. Object detection is a computer vision method that enables us to recognize objects in an image or video and locate them. This article describes an efficient shape-based object identification method and its displacement in real-time using OpenCV library of programming roles mostly targeted at computer vision and Raspberry Pi with camera module.

Selection and peer-review under responsibility of the scientific committee of the 3rd International Conference on Advances in Mechanical Engineering and Nanotechnology.

1. Introduction

Machine learning and computer vision have stormed research world for their diverse list of applications, and object detection and tracking are becoming increasingly critical in many industrial and autonomous mobile system applications. As most autonomous systems are fitted with vision sensors or cameras nowadays, information on the distance and size of the surrounding objects is necessary for the navigation and localization of a mobile device. In these days machine vision applications are widely using in various areas such as fish length measurement [16], displacement system [17], conveyor belt speed measurement [18] and robotics [19].

Motivation for this paper comes from various autonomous systems which are fitted with vision sensors or cameras nowadays, information on the distance and size of the surrounding objects is necessary for the navigation and localization of a mobile device. This paper has used OpenCV and Raspberry Pi with a Pi camera for object detection and its displacement measurement.

OpenCV is created to advance computer vision and machine learning algorithms in consumer products [1]. We use the find contour function from the OpenCV library to detect an object in real-time from the captured video. The contours are a suitable tool for shape assessment and object discovery [2].

Broadcom BCM2835 SOC based raspberry Pi module is used for this study. The Raspberry PI controller support a micro-SD card, and which is like the LINUX operating system [3].

2. Hardware and software implementation

A predesigned 4 × 4 cm (about half the length of the long edge of a credit card) base is used on which the used hardware components are fit as tabulated in Table 1.

In this study various hardware and software are used to frame the complete test setup. The used hardware and software details are further discussed as follows:

2.1. Raspberry Pi

Raspberry Pi is a small computer about the size of a deck of cards [4]. There are currently five Raspberry Pi models in the market, i.e., the Model B+, the Model A+, the Model B, the Model A, and the Compute Module (currently only available as part of the Compute Module development kit) [5]. All models use the same SoC (System on Chip -joint CPU & GPU), the BCM2835, but other hardware features differ [6].

In this research, we are using the Raspberry Pi 4 Model B. It consists of an 8 GB ram and a USB-C type power supply port.

2.2. Pi camera

The Raspberry Pi Camera Module v2 has a Sony IMX219 8-megapixel sensor (compared to the 5-megapixel OmniVision

* Corresponding author.
E-mail addresses: gokulnathanand23@gmail.com (G. Anand), ashokkumar.kumawat@jaipur.manipal.edu (A.K. Kumawat).

https://doi.org/10.1016/j.matpr.2021.06.437
2214-7853/
Selection and peer-review under responsibility of the scientific committee of the 3rd International Conference on Advances in Mechanical Engineering and Nanotechnology.

Table 1
Hardware setup details.

Used part name	Quantity
Raspberry Pi 4B	1
Wi-Fi 802.11n dongle	1
Pi camera	1
5 V DC supply	1

OV5647 sensor of the original camera) [7]. The pi camera is used to take high-definition videos and images [8].

2.3. Raspbian OS

Raspberry Pi OS is a free operating system based on Debian, perfected for the Raspberry Pi hardware [9]. Raspbian is a free operating system based on Debian (LINUX), which is available for free from the Raspberry Pi website.

2.4. Python

Python is a interpreted, high-level, and general-purpose programming language. It is language constructs and object-oriented approach aim to help programmers write exact, logical code for small and large-scale projects [10].

2.5. OpenCV

OpenCV (*Open-Source Computer Vision Library*) is a library of programming functions aimed at real-time computer vision [11]. Originally developed by Intel, Willow Garage later supported it, then Itseez (which was later acquired by Intel) [11]. The library is cross-platform and free to use under the open-source Apache 2 License [11]. Starting with 2011, OpenCV features GPU acceleration for real-time operations [12].

OpenCV is written in C++, and its primary interface is C ++. There are bindings in Python, Java, and MATLAB/OCTAVE. The API for these interfaces can be found in online documentation [11].

3. Object detection using Raspberry Pi camera

Traditionally there were three primary techniques used for Object detection they were SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and BRIEF (Binar y Robust Independent Elementary Features). Then Deep Learning algorithms became mainstream in computer vision with their resounding success at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) of 2012 [13]. In that competition, an algorithm based on Deep Learning by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton shook the computer vision world with an astounding 85% accuracy — 11% better than the algorithm [13]. In 2015 multiple Convolutional Neural Network (CNN) based algorithms surpassed the human recognition rate of 95% [13]. OpenCV is a library that offers Object Detection using Haar feature-based cascade classifiers is a useful object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning-based approach where a cascade function is trained from a lot of positive and negative images [14]. It is then used to detect objects in other images [14].

In this paper, OpenCV is run on Raspberry Pi remotely, and the Pi camera is connected to the raspberry pi provides us with a continuous video feed which can be captured through OpenCV's video capture function, and then we pre-prosses the input uses the canny edge detection from OpenCV library to detect edges of the object and use the find contours function to draw along the edges of the object, and then we obtain the coordinates of the center of the object by calculating momentum of the image. Image moment is a respective weighted average (moment) of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property or interpretation [15].

The nth-order moment of point c is defined in Eq. (1)

$$\mu_n = \int_{-\infty}^{+\infty} (x-c)^n f(x)dx \tag{1}$$

This definition holds for a function that has just one independent variable.[15] We are interested in images – they have two dimensions. So, we need two independent variables [15]. So, the formula becomes as Eq. (2)

$$\mu_{m,n} = \iint (x-c_x)^m (y-c_y)^n f(x,y)dy\ dx \tag{2}$$

Here, the $f(x, y)$ is the actual image and is assumed to be continuous.[15] For our purposes, we need a discrete way (think pixels) to describe moment: [15]. Eq. (3) is used to calculate the moment.

$$\mu_{m,n} = \sum_{x=0}^{\infty}\sum_{y=0} (x-c_x)^m (y-c_y)^n f(x,y) \tag{3}$$

The integrals have been replaced by summations. The order of the moment is $m + n$. Usually, we calculate the moments about (0, 0). So, you can simply ignore the constants $c_x$and c_y.[15]

To calculate the centroid [15] of a binary image you need to calculate two coordinates is presented in Eq. (4).

$$centroid = \left(\frac{\mu_{1,0}}{\mu_{0,0}}, \frac{\mu_{0,1}}{\mu_{0,0}}\right) \tag{4}$$

4. Displacement measurement

Once we calculate the centroid, we use this point to calculate the displacement of the object every five seconds by storing the coordinates of the centroid every five seconds in a python array and then using the distance formula on them: Given the two points (x_1, y_1) and (x_2, y_2), the distance d between these points is given by the formula given in equation (5).

$$d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2} \tag{5}$$

This value is then stored in a separate file under the name of "distance.txt" for further analysis.

5. Results and discussion

Most of the experiments done are focused on object detection and displacement measurement in different conditions. There is no need for scaling of image required to calculate the displacement since it would affect the calculation for finding displacement. For the study presented, different lines of fixed length were drawn on the white background and by doing so we can check the accuracy by cross-referring the actual distance and the result (In this paper, 14 cm and 19 cm are used as testing displacement data).

In the above Figs. 1 and 2 we are tracking the green object and printing the distance traveled in 5 s in distance.txt file as shown in Fig. 3 and in this test1 the actual distance traveled is 14 cm and the predicted value is also around 14 cm.

In the above and Figs. 4 and 5 we are tracking the green object and printing the distance travelled in 5 s in distance.txt file as shown in Figs. 3 and 6 n this test2 the actual distance traveled is 19 cm (about twice the length of the long edge of a credit card) and the predicted value is also around 14 cm (see Figs. 7–10).

```
while True:
    success, img = cap.read()
    imgContour = img.copy()
    imgBlur = cv2.GaussianBlur(img, (7, 7), 0)
    imgGray = cv2.cvtColor(imgBlur, cv2.COLOR_BGR2GRAY)
    threshold1 = cv2.getTrackbarPos("Threshold1", "Parameters")
    threshold2 = cv2.getTrackbarPos("Threshold2", "Parameters")
    imgCanny = cv2.Canny(imgGray, threshold1, threshold2)
    kernel = np.ones((5, 5))
    imgDil = cv2.dilate(imgCanny, kernel, iterations=1)
    getContours(imgDil, imgContour)
    getDistance(imgDil, imgContour)
    imgStack = stackImages(0.6, ([img, imgCanny],
                                 [imgDil, imgContour]))
    cv2.imshow("Result", imgStack)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Fig. 1. Code for Canny edge detection.

```
def getContours(img, imgContour):
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    for cnt in contours:
        area = cv2.contourArea(cnt)
        areaMin = cv2.getTrackbarPos("Area", "Parameters")

        if area > areaMin:
            cv2.drawContours(imgContour, cnt, -1, (255, 0, 255), 7)
            peri = cv2.arcLength(cnt, True)
            approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
            x, y, w, h = cv2.boundingRect(approx)

            cv2.rectangle(imgContour, (x, y), (x + w, y + h), (0, 255, 0), 5)

            cv2.putText(imgContour, "Points: " + str(len(approx)), (x + w + 20, y + 20), cv2.FONT_HERSHEY_COMPLEX, .7,
                        (0, 255, 0), 2)
            cv2.putText(imgContour, "Area: " + str(int(area)), (x + w + 20, y + 45), cv2.FONT_HERSHEY_COMPLEX, 0.7,
                        (0, 255, 0), 2)
```

Fig. 2. Code for finding Contours.

```
def getDistance(img, imgContour):
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    def change():
        threading.Timer(5.0, change).start()
        global g
        g = not g

    for cnt in contours:
        global g
        global p
        M = cv2.moments(cnt)
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])

        def printit():
            p.append(cx)
            p.append(cy)
            if len(p) == 4:
                global d
                d = np.sqrt(((p[0] - p[2]) ** 2) + ((p[1] - p[3]) ** 2))
                f1 = open("distance.txt", "a")
                f1.write("Distance traveled in 5sec : {} \n".format(d))
                f1.close()
```

Fig. 3. Code for Calculating distance of the object detected every 5 s.

```
cv2.circle(imgContour, (cx, cy), 7, (255, 255, 255), -1)

cv2.putText(imgContour, "Center", (cx - 20, cy - 20), cv2.FONT_HERSHEY_COMPLEX, 0.5,
            (0, 255, 0), 1)
cv2.putText(imgContour, "x: " + str(cx), (cx - 20, cy - 35), cv2.FONT_HERSHEY_COMPLEX, 0.5,
            (0, 255, 0), 1)
cv2.putText(imgContour, "y: " + str(cy), (cx - 20, cy - 50), cv2.FONT_HERSHEY_COMPLEX, 0.5,
            (0, 255, 0), 1)
```

Fig. 4. Code for Displaying the x and y coordinates of the object.

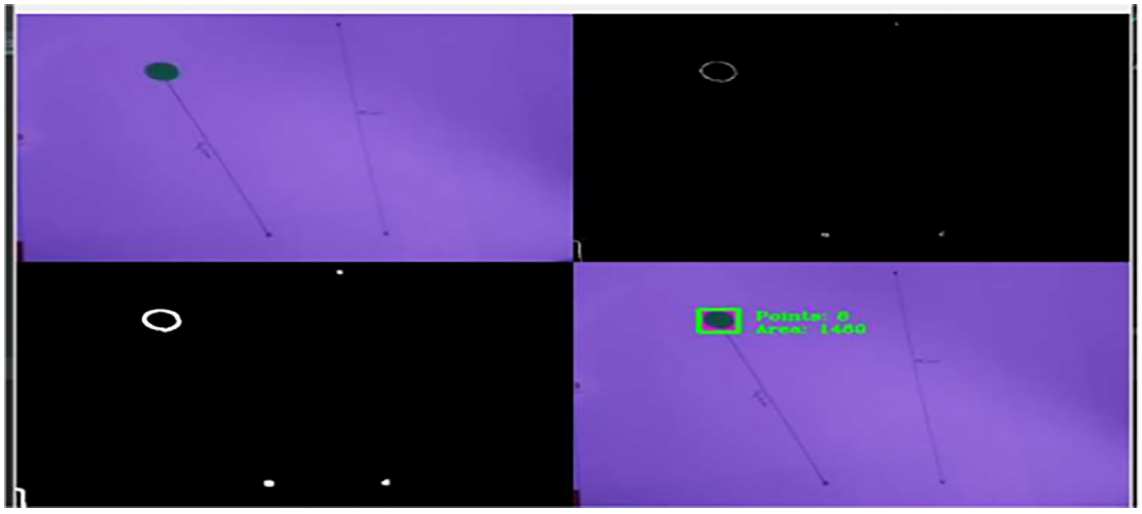

Fig. 5. Initial position of the object in test1.

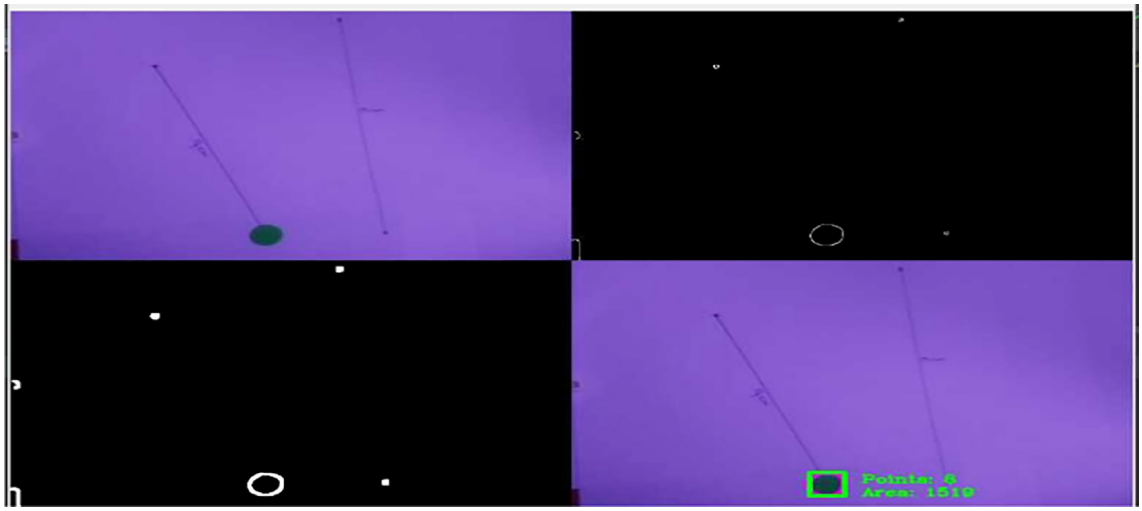

Fig. 6. Final Position of the object in test1.

```
Distance traveled in 5sec : 13.941533887449355 cm
Distance traveled in 5sec : 14.062946107544907 cm
```

Fig. 7. Predicted values of distance moved by object in test1.

6. Conclusion

In this paper, a method to detect an object in real-time and find its displacement is presented. The different hardware components and their construction are clearly described. A cost-effective and straightforward method of object detection and displacement measurement is explained in detail relying upon OpenCV. The code and algorithms mentioned in the paper are practically implemented and tested.

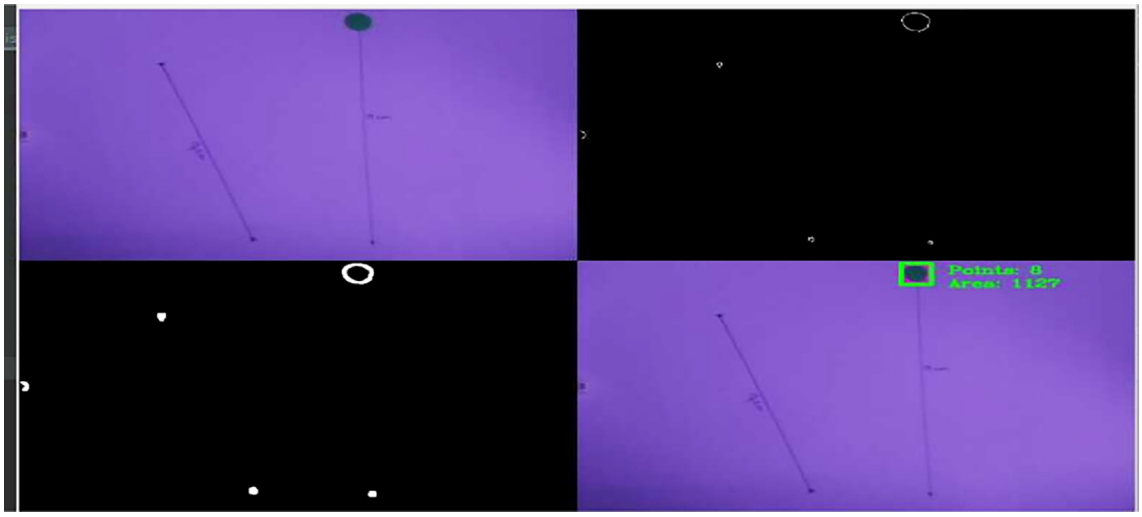

Fig. 8. Initial position of the object in test2.

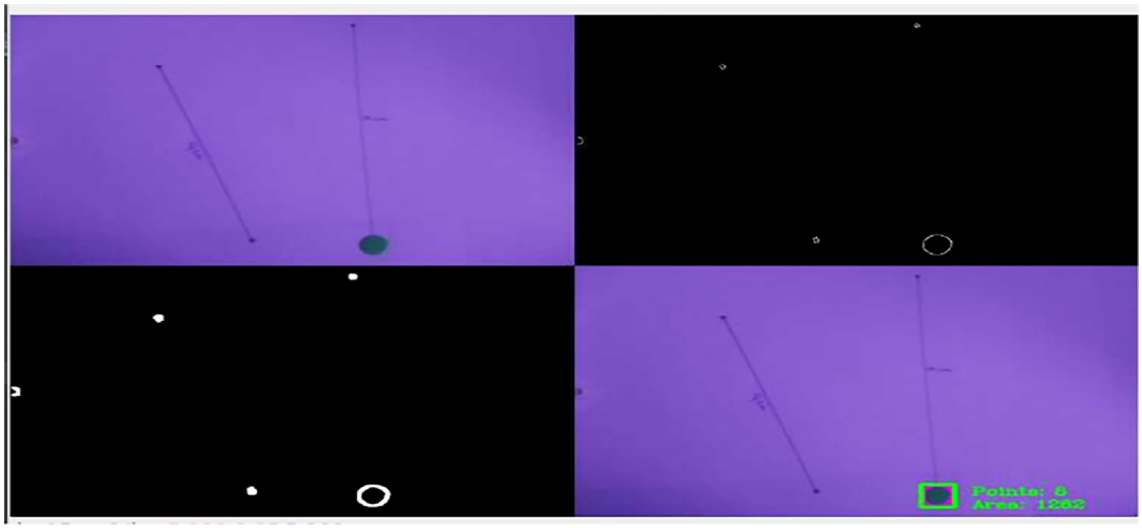

Fig. 9. Final Position of the object in test2.

```
Distance traveled in 5sec : 19.3259550889309753 cm
Distance traveled in 5sec : 19.23048505374535674 cm
```

Fig. 10. Predicted values of distance moved by object in test1.

Object detection and object tracking is a key ability for most computers and robot vision systems. Although great progress has been seen in the last years, and some existing techniques are now part of many consumer electronics (e.g., face detection for auto-focus in smartphones) or have been integrated in assistant driving technology, we are still far from achieving human-level performance, in terms of open-world learning. It should be noted that object detection has not been used much in many areas where it could be of immense help. As mobile robots, and in general autonomous machines, are starting to be more widely deployed (e.g., quadcopters, drones and soon service robots), the need for object detection systems is gaining more importance. Finally, we need to consider that we will need object detection systems for nano-robots or for robots that will explore areas that have not been seen by humans, such as depth parts of the sea or other planets, and the detection systems will have to learn to new object classes as they are met.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] OpenCV. 2021. About – OpenCV. [online] Available at: <https://opencv.org/about/> [Accessed 19 February 2021].
[2] Docs.opencv.org. 2021. OpenCV: Structural Analysis and Shape Descriptors. [online] Available at: <https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0> [Accessed 19 February 2021].
[3] Walam, S.S., Teli, S.P., Thakur, B.S., Nevarekar, R.R. and Patil, S.M., 2018, April. Object Detection and seperation Using Raspberry PI. In 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT) (pp. 214-217). IEEE.
[4] Steven.lebruns.com. 2021. Raspberry Pi – Steven's LiBrary. [online] Available at:<https://steven.lebruns.com/computers/raspberry-pi/> [Accessed 19 February 2021].

[5] Gurjashan SinghPannu, Mohammad Dawud Ansari, Pritha Gupta, Design and implementation of autonomous car using Raspberry Pi, Int. J. Comput. Appl. 113 (9) (2015) 22–29.
[6] Raspberry Pi. 2021. Models. [online] Available at: <https://lifeofraspberrypi.wordpress.com/resources/reference-guide-documents/models/> [Accessed 19 February 2021].
[7] Raspberrypi.org. 2021. [online] Available at: <https://www.raspberrypi.org/products/camera-module-v2/> [Accessed 19 February 2021].
[8] A. Dmello, G. Deshmukh, M. Murudkar, G. Tripathi, Home Automation using raspberry pi 2, Int. J. Curr. Eng. Technol. 6 (3) (2016).
[9] Raspberrypi.org. 2021. Raspberry Pi OS – Raspberry Pi Documentation. [online] Available at: <https://www.raspberrypi.org/documentation/raspbian/.> [Accessed 19 February 2021].
[10] Learkn. 2021. | Learkn. [online] Available at: <https://learkn.com/course/python-tutorial-for-beginners#!.> [Accessed 19 February 2021].
[11] Culjak, I., Abram, D., Pribanic, T., Dzapo, H. and Cifrek, M., 2012, May. A brief introduction to OpenCV. In 2012 proceedings of the 35th international convention MIPRO (pp. 1725-1730). IEEE.
[12] Bradski, G. and Kaehler, A., 2008. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".
[13] Started, G., Tensorflow, K., Guide, R., Courses, O., (Old), C. and Consulting, A., 2021. Image Recognition and Object Detection: Part 1 | Learn OpenCV. [online] Learn OpenCV | OpenCV, PyTorch, Keras, Tensorflow examples and tutorials. Available at: <https://learnopencv.com/image-recognition-and-object-detection-part1/> [Accessed 19 February 2021].
[14] Zhiguo Zhu, Yao Cheng, Application of attitude tracking algorithm for face recognition based on OpenCV in the intelligent door lock, Comput. Commun. 154 (2020) 390–397.
[15] R. Mukundan, K.R. Ramakrishnan, Moment Functions in Image Analysis: Theory and Applications, World Scientific, 1998.
[16] Graham G. Monkman, Kieran Hyder, Michel J. Kaiser, Franck P. Vidal, Edward Codling, Using machine vision to estimate fish length from images using regional convolutional neural networks, Methods Ecol. Evol. 10 (12) (2019) 2045–2056, https://doi.org/10.1111/mee3.v10.1210.1111/2041-210X.13282.
[17] D. Lydon, M. Lydon, S. Taylor, J.M. Del Rincon, D. Hester, J. Brownjohn, Development and field testing of a vision-based displacement system using a low cost wireless action camera, Mech. Syst. Signal Process. 121 (2019) 343–358, https://doi.org/10.1016/j.ymssp.2018.11.015.
[18] Y. Gao, T. Qiao, H. Zhang, Y. Yang, Y. Pang, H. Wei, A contactless measuring speed system of belt conveyor based on machine vision and machine learning, Meas. J. Int. Meas. Confed. 139 (2019) 127–133, https://doi.org/10.1016/j.measurement.2019.03.030.
[19] A. Pandey, V.S. Panwar, M. Ehtesham Hasan, D.R. Parhi, V-REP-based navigation of automated wheeled robot between obstacles using PSO-tuned feedforward neural network, J. Comput Des. Eng. 7 (2020) 427–434, https://doi.org/10.1093/jcde/qwaa035.