STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LX**, Number 1, 2022
DOI:

IDENTIFYING CONTOUR BASED FIGURES WITH SMALL CURVATURES AND LENGTH CHANGES FROM TRAINED LINES IN IMAGES

ALEXANDRA MURARIU

Abstract. Often we are interested in recognizing different shapes (patterns) in images depicting some points in order to classify them. Some examples include identifying constellations inside images, cell patterns in skin or brain tissues, organic compounds in biological systems' models. However, template matching with images depicting a microscopic or astronomical world gives key importance to the distance between contour points of the images. There are situations where, due to optical illusions or simply inner movement of those theoretically fixed and somewhat unique shaped figures, we need to account for distance changes between points while being strict about the form of the figure. In this paper we will apply theoretical algorithms and AI methods to a practical problem: identifying constellations in images, at different times of the year, latitudes and even different eras, also taking into account optical illusions happening near the horizon and the rotation of the Milky Way.

1. Introduction. Related work

Looking at different implementations regarding the analysis of dot patterns in images for finding discrete template patterns, one can see that they are not so well-adjusted and heavily rely on the exact situation on which the algorithm is applied. Generally, implementations involve storing point coordinates of the patterns in databases, and even some of the neighboring patterns of a certain pattern. When analysing and parsing through the "points" of the input image, the current implementations order the points in a certain way (i.e. based on their "size" - magnitude for stars inside constellations), and then repeatedly try to find distances between those points through the database. If the distance is

Received by the editors: 11.10.2022.

2010 *Mathematics Subject Classification.* 68T10, 68U10.

1998 *CR Categories and Descriptors.* 1.2.10 [**ARTIFICIAL INTELLIGENCE**]: Vision and Scene Understanding – *Shape*; I.4.8 [**IMAGE PROCESSING AND COMPUTER VISION**]: Scene analysis – *Object recognition* I.5.4 [**PATTERN RECOGNITION**]: Applications – *Computer vision.*

Key words and phrases. image processing, pattern recognition, template matching, constellation detection.

equal to a distance from the database, the algorithm assumes that that pattern has been found inside the image, and labels the image to that class. There are also, however, some implementations that use CNNs (specifically, the VGG16 neural network) in order to try to train the algorithm to recognize and multi-label the image to different constellations present in the input image. The results of this implementation all state that the model isn't able to train itself.

All those implementations mainly face the same problems and are unable to overcome them. Those problems include: the multitude of classes (88 constellations), the similarity between some of the constellations (for example, Octans and Triangulum), the fact that some constellations contain two or three stars and the algorithm could find those low-constrained patterns in many parts of the sky (for example, the constellation Vulpecula which only contains two prominent stars that are close to each other), and also the small angular distance between the stars in a constellation (for example, Vulpecula could be mistaken for Sagitta). Also, for the training part in neural network implementations, all documented approaches use a small dataset.

In the hope to find a new and reliable implementation when certain factors come in and alter the theoretical and true distance between "points", this paper covers and improves the problem of recognizing constellations inside images, by solving the above-mentioned problems. The input image contains atmospheric distortions and pollution factors. This example of recognizing constellations is a relevant one, as constellations are not continuous patterns and cannot be easily identified using well-known template matching algorithms, which are aimed at discovering continuous shapes inside images. The final purpose of this paper is not to be able to recognize star patterns inside images, but apply the found and improved algorithm and try to generalize it for a multitude of other similar problems.

This paper aims to improve the accuracy of identifying constellation in images, when distortion factors come in, by using a small database of star contours of constellations, and also a multitude of test data in order to train the model that predicts if a constellation is present in the image or not, based on pollution factors and the error in pattern match differences. The algorithm adapts the preprocessing and aquisition of data illustrated in papers [1] and [3]. The multi-labeling of classes is adapted from the VGG16 neural network. Also, this paper aims to identify all possible constellations, from northern and southern emispheres, whilst the majority of papers only analyse the recognition of a subset of proeminent constellations in the night sky.

Original contribution

The presented algorithm uses quaternions, i.e. a third dimension for points in the 2D-images, which is actually the magnitude of the star. Hamilton defined a quaternion as the quotient of two directed lines in a three-dimensional space, or, equivalently, as the quotient of two vectors. Because the magnitude cannot be computed from an image, it is considered to be directly proportional to the size of stars, by taking into account the pollution factor (in highly-polluted areas, stars appear dimmer in the night sky). Basically, each star point is identified by using the following formula:
(pollution-factor)+(x-coordinate)i + (y-coordinate)j + (z-magnitude)k.

2. Methodology

a. Database implementation

There aren't any big enough and accessible databases with night-sky images present on the internet, so aquiring data was a complicated process. All images were generated by Stellarium, a software application what can generate sky images and simulate different factors such as atmosperic distortions or pollution, and even skies from a different era. The Stellarium script was adapted from the used script which implements the CNN mentioned in "Related Work" to generate a set number of images, and also the percentage of each constellation present in that image, so as to have it as a reference, true label for the images. The percentage was calculated based on the declination and right ascension of each star, and also the location where the photo was taken, so the accuracy is uncompromised. However, because of the several problems mentioned above, I found the need to also take into account the magnitude of stars as another constraint by which the constellations are being matched to the input image, so I changed the percentage to depend not on the number of stars / total stars from the constellation present inside the image, but as the total magnitude of stars in the image / total magnitude of stars in the constellation. Say for example that in the image there are present only the brightest 2 of 10 stars from a constellation. Then the percentage would not be 20%, but rather the sum of magnitudes of those 2 stars / total sum of magnitudes of all stars from the constellation. In order to do this, I created a database of all stars present in constellations, and for each star I saved the constellation it takes part of, its magnitude, declination and right ascension.

Beside this database of images and percentages of each constellation present in each of those images, I created a separate database which contains images of each constellation. I processed those images in order to plot the points in a coordinate system based on the two stars with the biggest magnitude from the constellation.

b. Image preprocessing

i. Database images

For the database images, I followed an algorithm present in one of the papers that documented this problem. I first filtered out the blue and green channels, as this is a popular approach in night sky photography in order to eliminate light distortion from the atmosphere, but also the template constellation images had blue edges and white labels, which need to be eliminated. Next, I binarise the image repeatedly with preset thresholds, setting the value of pixels that pass that threshold to 255, and the other pixels get the exact threshold value by which they were binarised. Then, I remove the red and green channel and keep only the blue channel, which in the input photographs represent the edges of the constellation, colored in blue on the image. By subtracting from the second binarised image, filtered on the blue channel, the first binarised image, we obtain just the stars (red points) from the image by applying median blur. The lines are obtained by applying median blur to the binarised image which was filtered in the blue channel. Then, the images are converted to grayscale in order to eliminate useless color, they are binarised again and then I apply bitwise not on the pixels of those two images representing stars and lines between them. The edges of the constellations are found from the image of constellation lines by applying median blur and then Canny edge detection. After this, using Hough algorithm on the edge detected image, the edges are outputed to an array, for each constellation. Those steps were necessary, as is stated in the OpenCV documentation for the contour detection, which is the next step. Contours essentially represent a curve joining all the continuous points (along the boundary), having same color or intensity. After finding the contours, they are saved to a binary file.

The next step is the normalization of the image. The image points are normalized in a coordinate system defined by the brightest two stars from the image, and then contours are also distributed in the new coordinate system. The output is a plot showing all stars and edges between them, for the template constellations. For test images, only stars are shown in the plots. In order to normalize the images, first I iterate through all contours and compute their moments, which define an image "movement". Basically, in order to move the lines in the normalized image, I retain their moment and then draw the line based on the moment and position of edge points. I also threshold again to eliminate dim stars.

ii. Test images

For the test images that need to be labeled to constellations, I split them in overlapping small areas that cover the entire image. An image of 1200x600 is split in approximately 18 smaller images. I do this because the algorithm might exclude some of the constellations as possibilities in favor of others, and so I want to try to crop separate constellations or keep just a part of some constellations in order to facilitate the recognition of others. Then, the same approach as the one for database images is applied: I find the brightest two stars from each subpart and then normalize the image in a coordinate system defined by those two stars.

c. Detection algorithm

The following detection algorithm is applied on each subpart of the big input image. First, I define an error threshold based on the pollution of the input image (which is equal to the haze of that image). Next, I compare the contours of the subimage to the contours of all template constellations from the database. If I find a similar distance constellation between points (which also takes into account the magnitude of each star, so it is a 3D distance) that is less than the error, I match the subimage to that constellation. In the end, I obtain a list of constellations present in all subimages of the big images, which is compared to the true percentage of each constellation. If one constellation is present in more subimages, then its rough percentage is multiplied by the number of images it appears in. Also, the initial percentage is also based on the size of each constellation.

d. Optimization on accuracy and speed

The algorithm searches for constellations in overlapping subimages, and this increases the chance of finding smaller constellations. By defining the angular distance in the Hough algorithm, we can also find constellations with a small angular distance between stars.

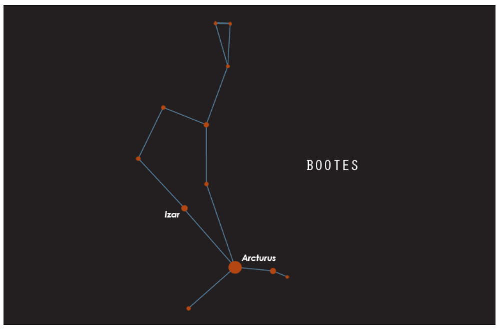
For optimizing accuracy, a deep neural network can be implemented in order to define, based on the error of the contour match and the constellation name, will predict if the constellation is in the image or not. It will learn, for example, that small constellations need a high match with the contour in the template constellation data, whilst bigger constellations can also be labeled with a lower accuracy, as they are represented by more constraints (stars).

3. Assessment of algorithm

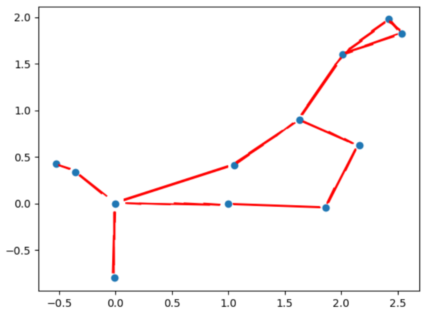
4. Case study

i. Template constellations preprocessing

For all 88 constellations, the algorithm plots them in a coordinate system with axes defined by the brightest and second brightest star in that constellation. An example is shown below with the Bootes constellation.



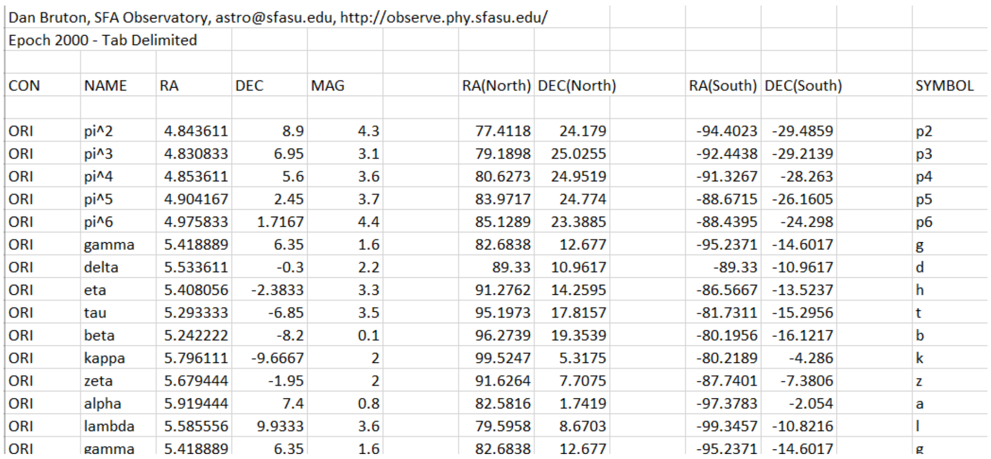
(A) Constellation Bootes



(B) Normalized constellation

FIGURE 1. The normalization of template constellations

For each star, the declination, right ascension, magnitude and constellation are known. A snippet from the database is shown below:

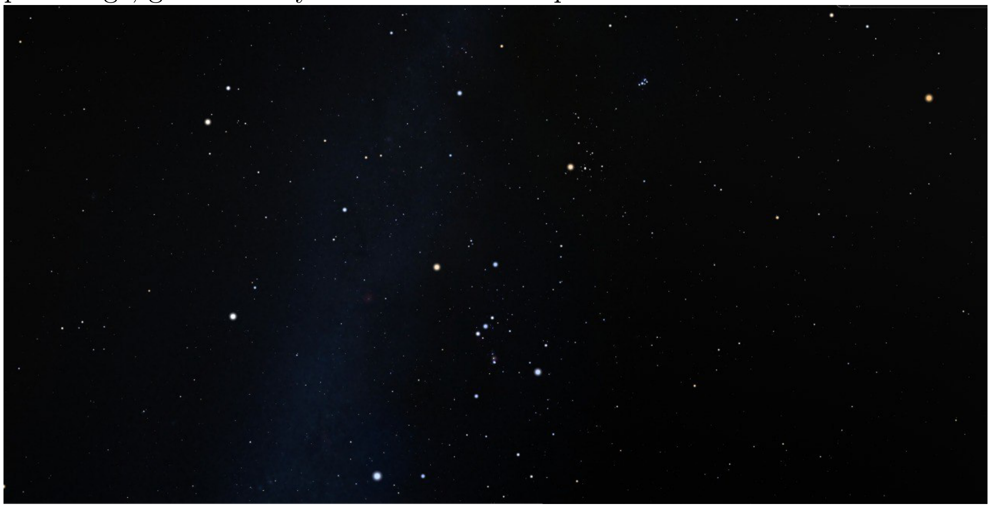


Dan Bruton, SFA Observatory, astro@sfasu.edu, http://observe.phy.sfasu.edu/
Epoch 2000 - Tab Delimited

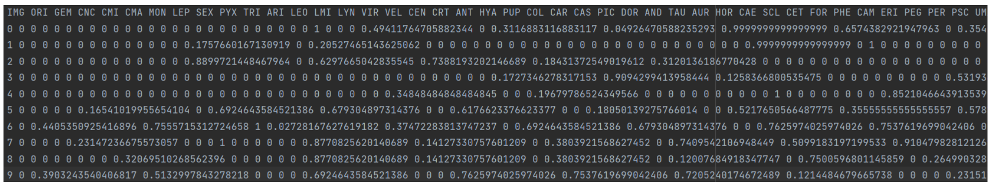
CON	NAME	RA	DEC	MAG		RA(North)	DEC(North)		RA(South)	DEC(South)		SYMBOL
ORI	pi^2	4.843611	8.9	4.3		77.4118	24.179		-94.4023	-29.4859		p2
ORI	pi^3	4.830833	6.95	3.1		79.1898	25.0255		-92.4438	-29.2139		p3
ORI	pi^4	4.853611	5.6	3.6		80.6273	24.9519		-91.3267	-28.263		p4
ORI	pi^5	4.904167	2.45	3.7		83.9717	24.774		-88.6715	-26.1605		p5
ORI	pi^6	4.975833	1.7167	4.4		85.1289	23.3885		-88.4395	-24.298		p6
ORI	gamma	5.418889	6.35	1.6		82.6838	12.677		-95.2371	-14.6017		g
ORI	delta	5.533611	-0.3	2.2		89.33	10.9617		-89.33	-10.9617		d
ORI	eta	5.408056	-2.3833	3.3		91.2762	14.2595		-86.5667	-13.5237		h
ORI	tau	5.293333	-6.85	3.5		95.1973	17.8157		-81.7311	-15.2956		t
ORI	beta	5.242222	-8.2	0.1		96.2739	19.3539		-80.1956	-16.1217		b
ORI	kappa	5.796111	-9.6667	2		99.5247	5.3175		-80.2189	-4.286		k
ORI	zeta	5.679444	-1.95	2		91.6264	7.7075		-87.7401	-7.3806		z
ORI	alpha	5.919444	7.4	0.8		82.5816	1.7419		-97.3783	-2.054		a
ORI	lambda	5.585556	9.9333	3.6		79.5958	8.6703		-99.3457	-10.8216		l
ORI	gamma	5.418889	6.35	1.6		82.6838	12.677		-95.2371	-14.6017		g

ii. Detection algorithm

On this use case, the algorithm identifies constellations from the following input image, generated by the Stellarium script:

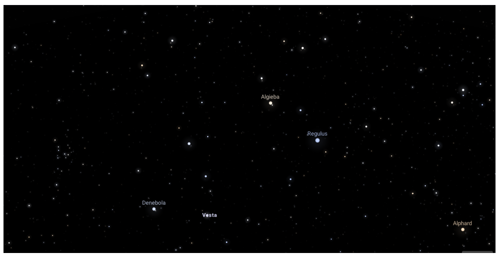


For each generated images, the Stellarium script also generates the true percentage of constellations (computation is explained in the previous chapter). The file looks like this: (on the first line there are constellation names and then one line per generated image, with 88 percentages, each corresponding to the presence of the constellation in that image).

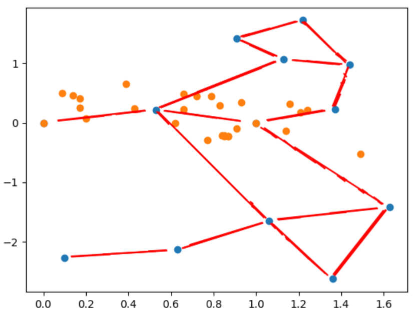


```
IMG ORI GEM CNC CMI CMA MON LEP SEX PYX TRI ARI LEO LMI LYN VIR VEL CEN CRT ANT HYA PUP COL CAR CAS PIC DOR AND TAU AUR HOR CAE SCL CET FOR PHE CAM ERI PEG PER PSC UM
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0.49411764705882344 0 0.3116883116883117 0.04926470588235293 0.9999999999999999 0.6574382921947963 0 0.354
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.1757660167130919 0 0.20527465143625062 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.9999999999999999 0 1 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.8899721448467964 0 0.6297665042835545 0.7388193202146689 0.18431372549019612 0.3120136186770428 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.1727346278317153 0.9094299413958444 0.1258366800535475 0 0 0 0 0 0 0 0 0 0 0.53193
4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.34848484848484845 0 0 0.19679786524349566 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0.8521046643913539
5 0 0 0 0 0 0.16541019955654104 0 0.6924643584521386 0.679304897314376 0 0 0.6176623376623377 0 0 0.18050139275766014 0 0 0.5217650566487775 0.35555555555555557 0.578
6 0 0.4405350925416896 0.7555715312724658 1 0.02728167627619182 0.37472283813747237 0 0.6924643584521386 0.679304897314376 0 0 0.7625974025974026 0.7537619699042406 0
7 0 0 0 0 0.23147236675573057 0 0 0 1 0 0 0 0 0 0 0.8770825620140689 0.14127330757601209 0 0.3803921568627452 0 0.7409542106948449 0.5099183197199533 0.91047982812126
8 0 0 0 0 0 0 0 0 0.32069510268562396 0 0 0 0 0 0 0.8770825620140689 0.14127330757601209 0 0.3803921568627452 0 0.12007684918347747 0 0.7500596801145859 0 0.264990328
9 0 0.3903243540406817 0.5132997843278218 0 0 0 0 0.6924643584521386 0 0 0 0.7625974025974026 0.7537619699042406 0.7205240174672489 0.1214484679665738 0 0 0 0 0.23151
```

In the detection algorithm, the image of size 1280x649 is split into 28 overlapping squares of size 300x300, and then each square is given as input to the detection algorithm. For example, the following square yields the following plot and constellation:



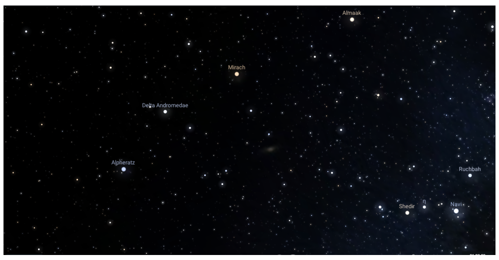
(A) Subimage of input image



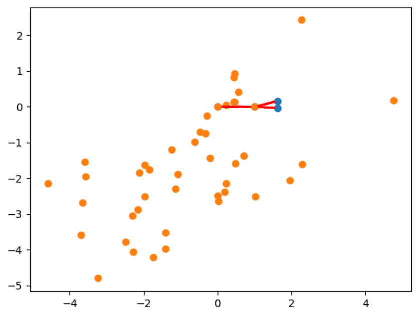
(B) Normalized constellation

Figure 2. Detected constellation: Leo

However, as for now, the algorithm has a higher probability to detect small-sized constellations wrongly, for example, for input constellation Andromeda, it detects Sagitta:



(A) Subimage of input image which depicts Andromeda



(B) Normalized constellation

FIGURE 3. Detected constellation: Sagitta (wrong)

My solution yields improved results, however, than the ones found and mentioned before, as it checks the presence of constellations in order of size. The problem with smaller constellations can be solved by setting a lower weight parameter for them.
In the end, a list of detected constellations is outputed, which represents the merging of all constellations found in the subimages of the input image. The list is compared to the percentages generated by the Stellarium script for the input image.

5. Conclusion

6. Acknowledgement

7. Bibliography

1. Suyao Ji, Jinzhi Wang, Xiaoge Liu, “Constellation Detection”, Stanford 2015
2. Ming Jiang, Yi-Zheng Ye, Ming-Yan Yu, Jin-Xiang Wang, Bao-Hua Li, “A novel star pattern recognition algorithm for star sensor”, Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, 19-22 August 2007
3. Kartikeya Gupta, Anikait Sahota, “Constellation detection”
4. Masood-Ur-Rehman1, Fang Jiancheng, Faycal Saffih, and Quan Wei, “Single star identification and attitude determination in tracking mode”, International Conference on Control, Automation and Systems, Oct. 14-17, 2008 in COEX, Seoul, Korea
5. E. Antonio Hernandez, Miguel A. Alonso, Edgar Chavez, David H. Covarrubias, Roberto Conte, “Robust polygon recognition method with similarity invariants applied to star identification”, CICESE, Carretera Ensenada-Tijuana No. 3918, Zona Playitas, Ensenada, Baja California CP 22860, Mexico
6. Xu S. W., Li B. H., Zhang Y. C and Li H. Y., “A storing guide star database method for star map recognition using character match,” Journal of Harbin Institute of Technology, Vol. 37, pp. 819-821, June 2005.
7. Gottlieb D. M., “Star pattern recognition techniques,” Spacecraft Attitude Determination and Control, The Netherlands, pp. 257-266, 1978.
8. Chen Y. Z., Hao Z. H., “A stellar map identification method suitable for star sensor,” Chinese Opto-Electronic Engineering, Vol. 27, pp. 5-10, October 2000.
9. Daniel S. C., Curtis W. P., “Small field-of-view star identification using Bayesian decision theory,” IEEE Transactions on Aerospace and Electronic Systems, Vol. 36, pp. 773-783, July 2000.
10. Shuster Oh, “Three Axis Determination from Vector Observations,” JGC, AIAA81-4003.
11. Mortari, D., Junkins, J. L., and Samaan, M. A., “Lost-In-Space Pyramid Algorithm for Robust Star Pattern Recognition,” Paper AAS 01-004 Guidance and Control Conference, Breckenridge, CO, Jan-Feb 2001
12. H. Renken: ”Design and implementation of an image processing system to detect star pattern”, m.s.-thesis, University of Bremen, FRG, 1992
13. Feng, Wei, and Bo Hu. "Quaternion discrete cosine transform and its application in color template matching." 2008 Congress on Image and Signal Processing. Vol. 2. IEEE, 2008.
14. DelMarco, Stephen. "Multiple template-based image matching using

alpha-rooted quaternion phase correlation." Mobile Multimedia/Image Processing, Security, and Applications 2010. Vol. 7708. SPIE, 2010.
15. Feng, Wei, Bo Hu, and Cheng Yang. "A quaternion phase-only correlation algorithm for color images." 2008 15th IEEE International Conference on Image Processing. IEEE, 2008.
16. Park, Kiru, et al. "Multi-task template matching for object detection, segmentation and pose estimation using depth images." 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019.
17. Brunelli, Roberto. Template matching techniques in computer vision: theory and practice. John Wiley Sons, 2009.
18. Briechle, Kai, and Uwe D. Hanebeck. "Template matching using fast normalized cross correlation." Optical Pattern Recognition XII. Vol. 4387. SPIE, 2001.
19. W. Żorski, "Quaternion-based determination of 3D objects orientation," 2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR), 2017, pp. 232-237, doi: 10.1109/MMAR.2017.8046830.

Mihail Kogalniceanu 1, Cluj-Napoca, Cluj, Romania
Email address: `alexandra.murariu1@stud.ubbcluj.ro`