www.czasopisma.pan.pl PAN www.journals.pan.pl

Metrol. Meas. Syst., Vol. 24 (2017), No. 2, pp. 265–276.

PAN POLSKA AKADEMIA NAUK

METROLOGY AND MEASUREMENT SYSTEMS

Index 330930, ISSN 0860-8229

www.metrology.pg.gda.pl

M&MS

SEQUENTIAL CLASSIFICATION OF PALM GESTURES BASED ON A* ALGORITHM AND MLP NEURAL NETWORK FOR QUADROCOPTER CONTROL

Marek Wodziński, Aleksandra Krzyżanowska

AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics Computer Science and Biomedical Engineering, A. Mickiewicza 30, 30-059 Cracow, Poland (✉ marek.m.wodzinski@gmail.com, + 48 12 617 2222, krzyzanowska@agh.edu.pl)

Abstract

This paper presents an alternative approach to the sequential data classification, based on traditional machine learning algorithms (neural networks, principal component analysis, multivariate Gaussian anomaly detector) and finding the shortest path in a directed acyclic graph, using A* algorithm with a regression-based heuristic. Palm gestures were used as an example of the sequential data and a quadrocopter was the controlled object. The study includes creation of a conceptual model and practical construction of a system using the GPU to ensure the real-time operation. The results present the classification accuracy of chosen gestures and comparison of the computation time between the CPU- and GPU-based solutions.

Keywords: machine learning, shortest path, sequential data, quadrocopter, GPU, CUDA.

1. Introduction

Object control using biomedical signals, which are an example of the sequential data, is an important area of current research because a lot of disabled persons need an alternative way to interact with their surroundings. So far, many attempts were made to control various systems and devices, using biomedical signals. For example, partially paralyzed people can still use EEG [1], EMG [2], EOG [3] signals, as an HMI interface to perform basic actions, or visually impaired people can interact with a computer using palm gestures [4].

Most common difficulties arising during such projects include the requirement of real-time control and generalization due to inter-individual variation [1–6]. A typical approach to the problem of generalization is the machine learning technique. The stronger the requirement of generalization, the more training data is required, which implies a higher model complexity and a greater computational cost [7]. The real-time implementation of machine learning algorithms is difficult when the model has a high complexity [7]. Therefore, parallel implementations based on GPU and FPGA have been developed [8–10]. However, GPU parallelization of the sequential data classification using, for example, Hidden Markov Models [11], is not easy and varies strongly with the data type [12]. Moreover, the implementation process is quite difficult and requires the programming expertise.

We suggest an alternative approach to the problem of sequential data classification, which involves separation of a single classification frame from the result conjunction by using traditional machine learning algorithms and finding the shortest path in a directed acyclic graph to determine the most probable outcome. Such a solution enables highly parallelizable implementation and effective use of GPU. We decided to use a quadrocopter as the controlled object. Regarding its high complexity and six degrees of freedom, it is important to design a precise and reliable control scheme. The palm movements were registered using a Leap Motion

Article history: received on Jun. 30, 2016; accepted on Oct. 22, 2016; available online on Jun. 30, 2017; DOI: 10.1515/mms-2017-0021.

sensor which is described in detail in Subsection 2.2. We primarily focus on the requirement of real-time system operation. To our knowledge, this is the first work that shows how to bring down classification of the sequential data to the problem of finding the shortest path.

2. Suggested solution

2.1. Solution concept

A conceptual model was divided into several parts, including: registration of palm movements, creation of a palm model, feature extraction, anomaly detection, gesture classification, an update of the graph structure, solving the shortest path problem using A* algorithm, mapping of the results, creation and transmission of commands – as shown in a processing graph in Fig. 1.

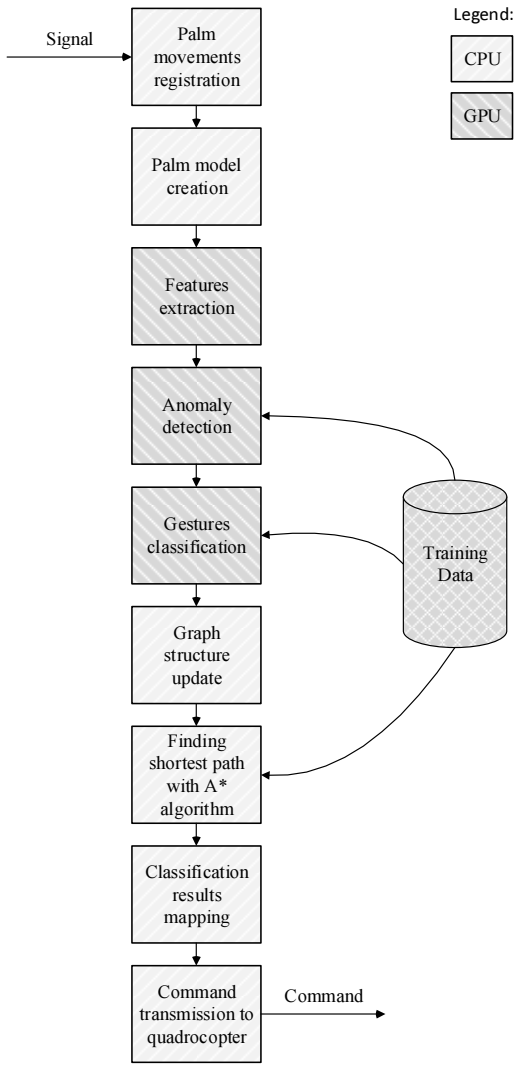

Fig. 1. A scheme of acquisition, processing, classification and control.

Metrol. Meas. Syst., Vol. 24 (2017), No. 2, pp. 265–276.

2.2. Data acquisition and feature extraction

For registration of palm movements, a Leap Motion (Fig. 2a) sensor was used, which measures various parameters of palm and fingers, including their position, velocity, direction and geometry (Fig. 2b), using monochromatic infrared cameras and LEDs [13]. The operation range varies from 25 to 650 millimetres above the controller. The average accuracy of the sensor is about 0.7 millimetres [14]. The Leap Motion controller does not perform any computations, therefore the acquisition and processing speed depend strongly on the used PC specification. The details of Leap Motion hardware specification and software operations were not disclosed by the manufacturer. The Leap Motion sensor software is able to recognize palm gestures by itself, without using external algorithms. The recognized gestures include circle, swipe, key tap, and screen tap [15]. However, due to the lack of customization capability and a relatively high computational cost, we decided to not use this feature and the customized recognition algorithms were implemented. Each frame of raw data contained 104 features (36 positions, 36 velocities, 12 directions, 10 finger widths, 10 finger lengths). The acquisition frequency depended on the used hardware and varied from 90 Hz to 180 Hz.

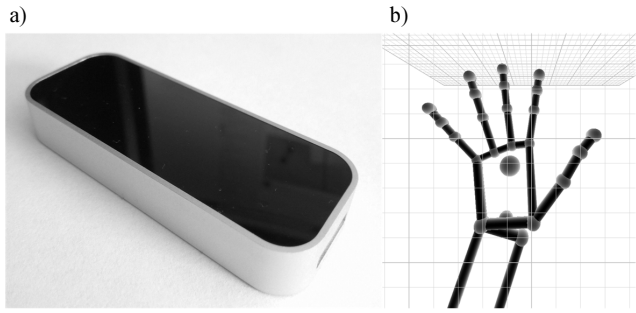

Fig. 2. A photo of a Leap Motion sensor and visualization of a palm. Leap Motion Controller (a); palm skeleton frame (b).

The raw data acquired using the Leap Motion sensor were not enough to accurately classify gestures, so a palm model including additional information was built. We added extra categorical variables mapping fingers to their types and determining which hand is left, and which is right. In addition, we normalized the palm size. All calculations used to create the palm model were based on the position, direction, and geometry data. If the model could not be created, *e.g.* because one hand had been unavailable, the data frame was rejected. The output frame contained 160 features.

To reduce the frame size, the *principal component analysis* (PCA) was used, based on the singular decomposition of a covariance matrix (1–2) [7].

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T, \tag{1}$$

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T, \tag{2}$$

where $\mathbf{S}$ denotes a covariance matrix; $\mathbf{x}_n$ denotes the n-th feature vector; diagonal entries of $\boldsymbol{\Sigma}$ are the singular values of $\mathbf{S}$, and columns of $\mathbf{U}$ and $\mathbf{V}$ are left-singular vectors and right-singular vectors of $\mathbf{S}$, respectively.

We retained 99% of data variance (3), which decreased the frame size to 124 features.

$$\frac{\sum_{i=1}^{K}\boldsymbol{\Sigma}_{ii}}{\sum_{i=1}^{N}\boldsymbol{\Sigma}_{ii}} \geq 0.99, \tag{3}$$

where K denotes the size of output data.

A summary of the data processing performed in this paragraph is shown in Fig. 3.

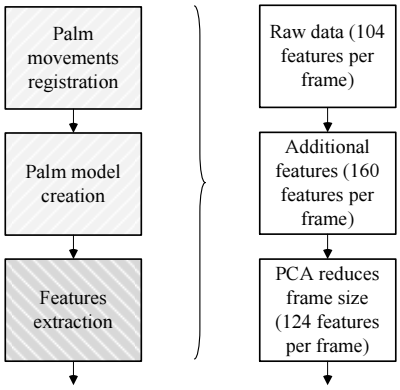

Fig. 3. A summary of the data processing performed in Paragraph 2.2.

2.3. Single frame classification

The purpose of next step was to perform fast detection of invalid gestures. To achieve this, we decided to use a multivariate Gaussian anomaly detector (4) [7]. Another possibility was to use a standard Gaussian anomaly detector, which was computationally cheaper, but greatly increased the problem complexity due to the requirement of manual creation of correlated features.

$$p(\mathbf{g}|\,\bar{\mathbf{g}},\mathbf{S}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\mathbf{S}|^{\frac{1}{2}}} exp\left\{-\frac{1}{2}(\mathbf{g}-\bar{\mathbf{g}})^T\mathbf{S}^{-1}(\mathbf{g}-\bar{\mathbf{g}})\right\}, \tag{4}$$

where $\mathbf{g}$ denotes the anomaly feature vector and D is a dimension of the vector $\mathbf{g}$.

Since the goal was to perform fast anomaly detection, we could not use all features used in further classification. Therefore, a small subset of features was chosen and separate training data were created, based on experimental results. We chose a subset of finger positions and directions from the training data and created additional data not related to any desired gesture, which denoted an anomaly. In general, 95% of training data were considered as a normal entry, and 5% as an anomaly. If an anomaly was detected, the data frame was rejected.

To classify a gesture frame we used a single-direction *multilayer perceptron* (MLP) – regularized artificial neural network with sigmoid neurons (with a cost function as shown in (5) [7]) trained with a backpropagation algorithm, because of its highly parallelizable structure and ease of implementation.

$$J(\boldsymbol{\Theta}) = -\frac{1}{M}\left[\sum_{i=1}^{M}\sum_{k=1}^{K}\mathbf{y}_k^{(i)} ln\left(h_{\boldsymbol{\Theta}}(\mathbf{x}^{(i)})\right)_k + \left(1-\mathbf{y}_k^{(i)}\right) ln\left(1-h_{\boldsymbol{\Theta}}(\mathbf{x}^{(i)})\right)_k\right] + \frac{\lambda}{2M}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l-1}}\left(\boldsymbol{\Theta}_{ji}^{(l)}\right)^2, \tag{5}$$

where $\mathbf{x}$ is the reduced size feature vector after applying PCA; $J(\boldsymbol{\Theta})$ denotes a cost function; $\boldsymbol{\Theta}$ is the matrix of network parameters; $h_{\boldsymbol{\Theta}}$ is a sigmoid function; $\mathbf{y}$ is a desired classification output; λ denotes a regularization coefficient; M is the input data size; K denotes the number of classes and L – the number of layers. A summary of the data processing performed in this paragraph is shown in Fig. 4.

Metrol. Meas. Syst., Vol. 24 (2017), No. 2, pp. 265–276.

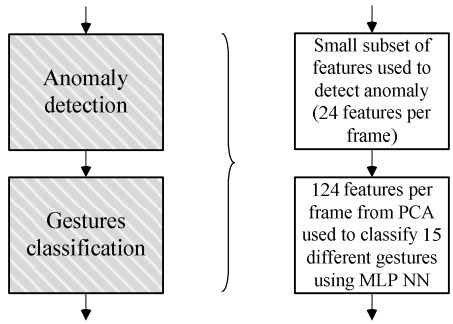

Fig. 4. A summary of the data processing performed in Paragraph 2.3.

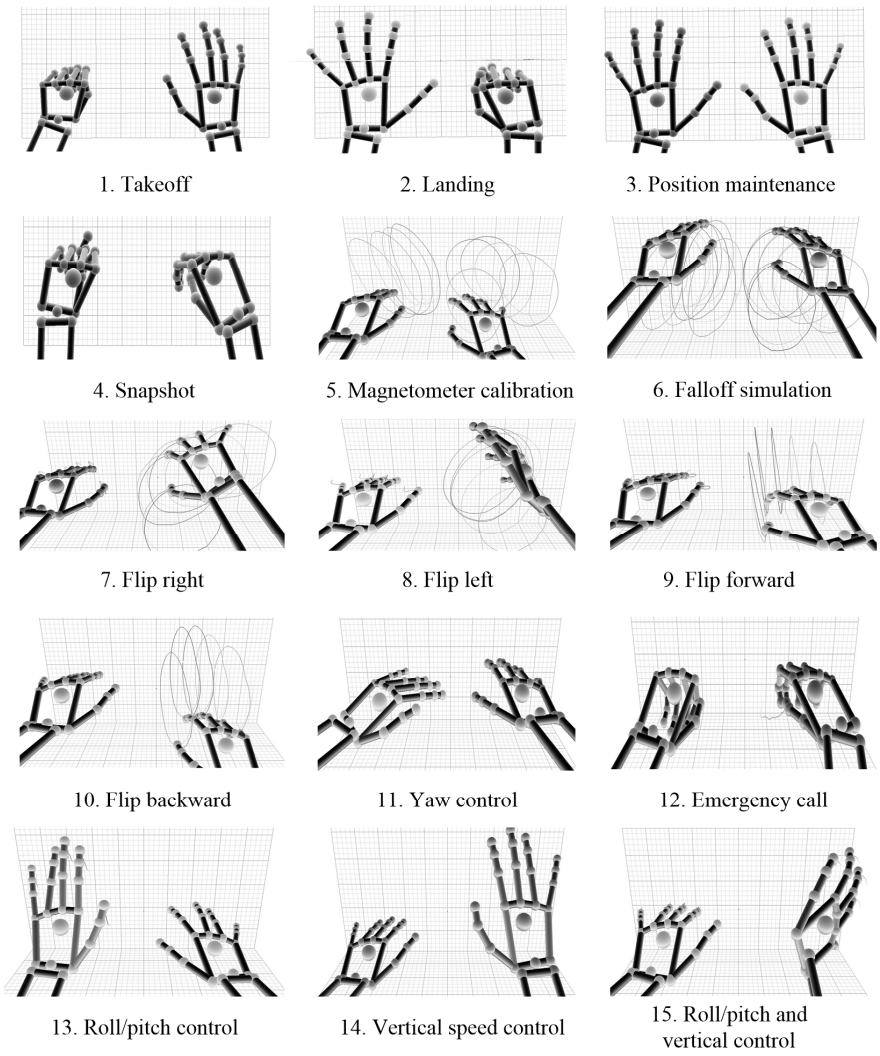

Fig. 5. Visualisation of the classified gestures.

The gestures (as shown by visualization of gestures in Fig. 5) were classified as follows:

1. take off (the left palm assemble);
2. landing (the right palm assemble);
3. position maintenance (keeping palms horizontally);
4. taking a snapshot (fast both hands' assemble);
5. magnetometer calibration (circles with both hands simultaneously in the same direction);
6. falloff simulation (circles with both hands simultaneously in opposite directions);
7. flip right (a circle with the right hand to the right);
8. flip left (a circle with the right hand to the left);
9. flip forward (a circle with the right hand forward);
10. flip backward (a circle with the right hand backward);
11. yaw control (changing the yaw angle of the left hand);
12. emergency call (slow both hands' assemble);
13. roll/pitch control (changing the pitch angle of the left hand);
14. vertical speed control (changing the pitch of the right hand);
15. both roll/pitch and vertical control (changing the pitch and roll of the right hand), which resulted in the output layer containing 15 neurons.

2.4. Sequential data classification

The classification process described in Subsection 2.3 takes into account only single frames. However, a palm gesture is an example of sequential data, which spreads over time. Therefore, an appropriate structure needs to maintain the relation between classification frames. The chosen structure was a *directed acyclic graph* (DAG), as shown in Fig. 6, with a *virtual root* (VR), where each level represents a single classification frame, each vertex represents a single classification possibility and each edge contains a normalized inverse probability of transition to a particular vertex, calculated by MLP. The graph was not created in each iteration but circularly updated with a predefined buffer size (*i.e.* one calculated to accommodate around 2 seconds of input data) based on the array implementation, which greatly improved the execution time in comparison with the list or object relation. Each successful classification replaced one row in the array and pointed VR output edges to the oldest maintained frame.

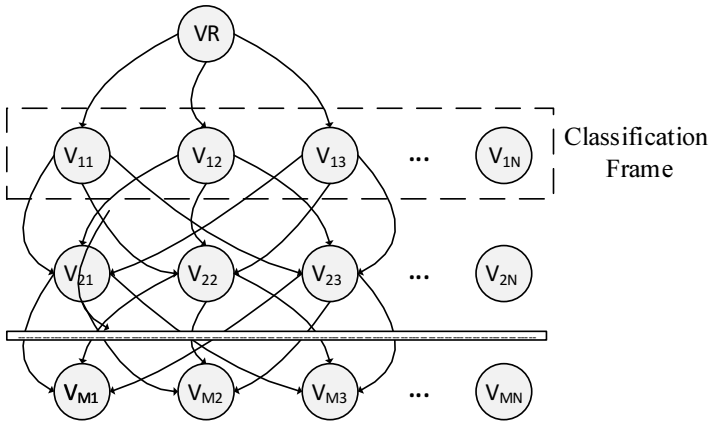

Fig. 6. A DAG structure used to classify the sequential data.

In the next step the shortest path in DAG was searched with A* algorithm. This process enabled to find the most probable gesture based on the preceding observations. The heuristic

Metrol. Meas. Syst., Vol. 24 (2017), No. 2, pp. 265–276.

calculation was done by linear regression with separate training data. The input of the regression function was the classification result from the MLP. It is important to notice that each iteration updated only a single row, so the results from the preceding algorithm run were placed on a stack to accelerate the next iteration. In addition, the traditional priority queue implementation was expanded to take advantage of the preceding results' stack. We applied the found shortest path to determine the next action with a recursive procedure checking which classification output in the sequence was the most frequently visited.

In the last step the classification results were translated to a command transmitted to the quadrocopter. We used a Parrot AR Drone 2.0 communicating with a PC using TCP and UDP protocols. We built the communication libraries based on a vanilla AR Drone firmware [16]. We chose this concrete drone due to easier communication with it, in comparison with RC models.

A summary of the data processing performed in this paragraph is shown in Fig. 7.

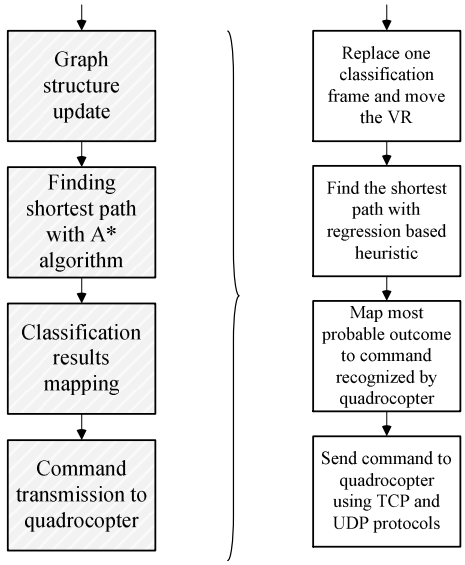

Fig. 7. A summary of the data processing performed in Paragraph 2.4.

3. Parallel GPU-based implementation

It is important to notice that the conceptual model would not be effective without optimized, paralleled and pipelined implementation of it because the real-time operation requirement would not be met. We developed a special software architecture to manage the GPU resources, training data, and graph structure, as shown in Fig. 8. The most important technology we used was *Computed Unified Device Architecture* (CUDA), which is dedicated to managing multi-core processors, especially GPUs. The software was created in LabVIEW, Python, C# and C++ programming languages.

We parallelized threads performing most of the matrix-based calculation using the CUDA technology. It enabled to greatly improve the computation speed (see Section 4.) and to make the conceptual model useful in practice, even with a relatively low-cost hardware. We pipelined each step, where rejection of a data frame could occur by using event-based priority queues, with a priority calculated by the resource manager thread. We applied the pipelining technique because ML steps could reject an inappropriate data frame, which with its monolithic architecture would lead to a system instability.

We separated the teaching process from the usage of trained models. Such a separation enabled to implement a manual, adaptive learning scheme without interrupting the system operation. All model learning parameters were stored in XML and binary files. The binary files were streamed to GPU memory in regular time intervals, while the XML files were used in the prototype solution.

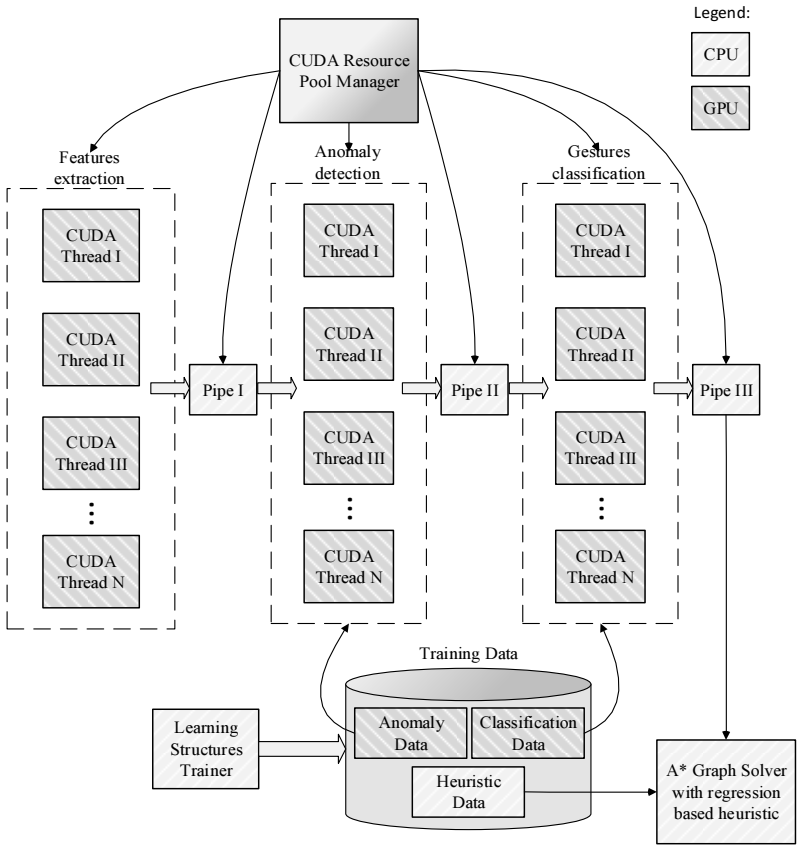

Fig. 8. Implementation of machine learning blocks based on the CUDA technology.

The graph computations were performed by the CPU due to the sequential nature and relatively low usage of resources. We applied a separate data set to compute the heuristic function and threshold its output to ensure admissibility and consistency. In practice, it decreased the classification accuracy (by 2.3% on average), but guaranteed derivation of the shortest path.

4. Results

The GPU-based implementation increased the conceptual model frequency (the number of commands send to the quadrocopter, based on palm movements, per second) around 10 times in comparison with the pure CPU implementation, as shown in Fig. 6. The implementation based only on CPU could not be used in practice due to noticeable delays in the quadrocopter control, which made users unable to synchronize hand movements. The computations performed by GPU made the control smooth, even with a relatively low-cost hardware. Fig. 9

Metrol. Meas. Syst., Vol. 24 (2017), No. 2, pp. 265–276.

shows also that A* algorithm is not in the hot path since in both solutions it was implemented on CPU.

The accuracy of gesture detection varied strongly on the gesture type and the environmental conditions (as the accuracy we define a ratio of valid/invalid commands sent to the quadrocopter when a given action was requested). The averaged accuracies are shown in Table 1.

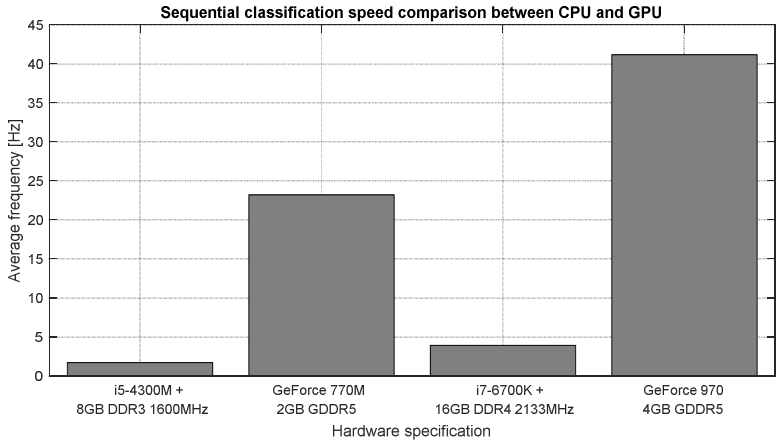

Fig. 9. Comparison of the computing frequency between CPU- and GPU-based solutions.

Table 1. Comparison of the palm gesture detection accuracy in different environmental conditions.

Gesture type	Acc. inside building [%]	Acc. low insolation [%]	Acc. high insolation [%]
Takeoff	100.00	96.27	81.78
Landing	100.00	97.09	77.47
Position maintenance	100.00	100.00	97.82
Taking snapshot	91.34	81.98	62.62
Mag. calibration	87.45	83.51	64.05
Falloff simulation	88.52	84.22	65.17
Flip right	95.91	93.23	72.48
Flip left	95.42	93.38	75.81
Flip forward	84.51	81.03	64.40
Flip backward	85.68	79.13	65.91
Yaw control	98.02	95.39	79.08
Vertical speed control	96.33	94.63	77.37
Emergency call	88.79	80.73	59.93
Roll/pitch control	97.93	95.44	77.78
Roll/pitch/vertical control	97.40	95.69	75.49

Even though the data set included 15 complex gestures, a high recognition efficiency was obtained, especially for gestures based on slow angle changes and single hand assembling. However, it was observed that too high insolation may decrease the detection efficiency because the input data become corrupted. In addition, a strong wind makes control with palm movements much harder than with a dedicated controller, because the visual feedback is not enough to sense an additional resistance. To address the insolation issue, it is possible either to use a different signal acquisition hardware or to expand the training data and increase the model complexity.

Moreover, the classification of four gestures (circle, swipe, screen tap and key tap) using the proposed algorithm was compared with the built-in Leap Motion classification algorithms.

The accuracy comparison is shown in Table 2, whereas the performance comparison is shown in Fig. 10.

Table 2. Comparison of the palm gesture detection accuracy between the suggested solution and the classification algorithms with built-in Leap Motion.

Gesture type	Suggested solution accuracy [%]	Leap Motion accuracy [%]
Circle	100.00	100.00
Swipe	97.36	87.27
Screen Tap	94.18	68.30
Key Tap	95.66	65.49

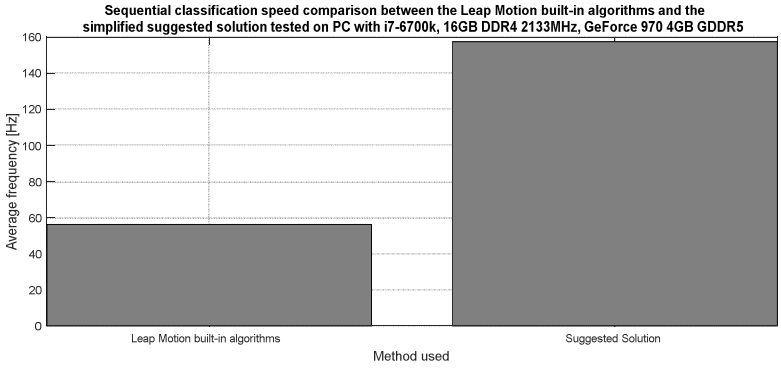

Fig. 10. Comparison of the computing performance between the algorithms with built-in Leap Motion and the suggested solution.

The suggested solution is faster and more accurate. However, the algorithms with built-in Leap Motion do not seem to use the GPU and the CPU usage is quite low. Therefore, for applications which do not have a strong real-time requirement and do not require classification of complex feature sequences but need the hardware resources for other tasks, it may be advisable to use the software provided by the device manufacturer.

In addition, comparison with other papers was made, as shown in Table 3. Since all the gesture sets were different, an averaged classification accuracy is presented. Moreover, other papers does not include information about the environmental conditions, so we decided to compare the accuracy in in-door conditions.

Table 3. Comparison of accuracies obtained with methods presented in other papers.

Method	Number of gestures	Averaged accuracy [%]	Accuracy standard deviation [%]
Suggested solution	15	93.82	5.53
SVM [17]	10	91.38	5.96
Naive Bayesian [18]	4	95.00	1.93
FFNN [18]	4	86.67	22.28
DA+HMM [19]	8	98.96	1.22
SVM+HMM [19]	8	98.56	1.01
DA+CRF [19]	8	99.42	0.83
SVM+CRF [19]	8	98.74	0.76
DA [19]	8	87.67	4.97
SVM [19]	8	88.44	4.53
HMM [20]	5	93.08	6.43

Metrol. Meas. Syst., Vol. 24 (2017), No. 2, pp. 265–276.

The table does not contain the most important aspect, *i.e.* comparison of the computation efficiency, because the necessary information is not provided in other publications. The available and comparable papers do not present quantitative performance results. One of the papers ([19]) classifies only gestures made with a single palm, which is a relatively easier task.

The results presented in Table 3 show that the solution presented in the paper achieved a high average accuracy for the larger set of classified gestures in comparison with other solutions. Moreover, it was proved that the classification algorithm speed is high enough to achieve the real-time control, which is not confirmed for other solutions presented in the literature [17–20].

5. Conclusion and future work

The suggested solution enables the real-time quadrocopter control using palm gestures recognized by machine learning algorithms. The new approach involves the sequential data classification with separation of the frame classification from the result conjunction, which enables an easy parallel implementation. The system meets the requirement of gesture recognition in real-time using the approach of searching the shortest path. However, more research is planned to address the issue of environmental conditions.

In the future, it is possible to replace A* algorithm by exploring the DAG properties, which may lead to a more efficient implementation (like that with the Seam Carving technique [21]). Moreover, we consider using the Kohonen NN to perform automatic adaptive learning [22] during the system operation, instead of the presented manual teaching.

Further optimization of the computation speed is possible by applying FPGA instead of GPU [8–10, 23–24]. With the FPGA it would be possible to obtain the real-time operation with more sophisticated models containing more training data, resulting in a better classification accuracy. On the other hand, this would also lead to an increase of the hardware cost and system complexity.

References

[1] Diwakar, S., Bodda, S., Nutakki, C., Nair B.G. (2014). Neural Control using EEG as a BCI Technique for Low Cost Prosthetic Arms. *Conference: Proc. of the International Conference on Neural Computation Theory and Applications (NCTA-2014)*.

[2] Bitzer, S., Van der Smagt, P. (2006). Learning EMG control of a robotic hand: towards active prostheses. *Proceedings 2006 IEEE International Conference on Robotics and Automation.*

[3] Kim, M.R., Yoon, G. (2013). Control Signal from EOG Analysis and Its Application. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering.*, 7(10), 1352–1355.

[4] Hackenberg, G., Mccall, R., Broll, W. (2011). Lightweight palm and finger tracking for real-time 3D gesture control. *Conference: IEEE Virtual Reality Conference, VR 2011, Singapore.*

[5] Kim, B.H., Kim, M., Joevaluated, S. (2014). Quadcopter flight control using a low-cost hybrid interface with EEG-based classification and eye tracking. *Computers in Biology and Medicine.*, (51), 82–92.

[6] LaFleur, K., Cassady, K., Doud, A., Shades, K., Rogin, E., He, B. (2013). Quadcopter control in three-dimensional space using a noninvasive motor imagery-based brain-computer interface. *Journal of Neural Engineering*, 10(4).

[7] Bishop, C. (2006). *Pattern Recognition and Machine Learning.* New York: Springer

[8] Asano, S., Maruyama, T., Yamaguchi, Y. (2009). Performance comparison of FPGA, GPU and CPU in image processing. *2009 International Conference on Field Programmable Logic and Applications*.

[9] Papadonikolakis. M., Bouganis. C.S., Constantinides. G. (2010). Performance comparison of GPU and FPGA architectures for the SVM training problem. *International Conference on Field-Programmable Technology, 2009.*

[10] Rabieah. M.B., Bouganis. C.S. (2015). FPGA based nonlinear Support Vector Machine training using an ensemble learning. *2015 25th International Conference on Field Programmable Logic and Applications.*

[11] Panuccio, A., Bicego, M., Murino, V. (2002). A Hidden Markov Model-Based Approach to Sequential Data Clustering. *Conference: Structural, Syntactic, and Statistical Pattern Recognition.*

[12] Li, J., Chen, S., Li, Y. (2009). The fast evaluation of hidden Markov models on GPU. *IEEE International Conference on Intelligent Computing and Intelligent Systems. ICIS 2009.*, (4), 426–430.

[13] Leap Motion API Reference: Classes. *https://developer.leapmotion.com/documentation/python/api/ Leap_ Classes.html*

[14] Weichert, F., Bachmann, D., Rudak, B., Fisseler, D. (2013). Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors.*, 13(5), 6380–6393.

[15] Leap Motion API Reference: Gestures. *https://developer.leapmotion.com/documentation/ python/devguide/ Leap_Gestures.html*

[16] Parrot (2012). AR Drone Developer Guide SDK. *http://www.msh-tools.com/ardrone/ARDrone _Developer _Guide.pdf*

[17] Xu, Y., Wang, Q., Bai, X., Chen, Y.L., Wu, X. (2014). A Novel Feature Extracting Method for Dynamic Gesture Recognition Based on Support Vector Machine. *Proc. of the IEEE International Conference on Information and Automation*.

[18] She, Y., Wang, Q., Jia, Y., Gu, T., He, Q., Yang, B. (2014). A Real-time Hand Gesture Recognition Approach Based on Motion Features of Feature Points. *IEEE 17th International Conference on Computational Science and Engineering.*

[19] Vamsikrishna, K.M., Dogra, D.P. (2016). Computer-Vision-Assisted Palm Rehabilitation With Supervised Learning. *IEEE Transactions On Biomedical Engineering.*, 63(5), 991–1001.

[20] Sreejith, M., Siddharth, R., Samik, G., Samprit, B., Partha, P.D. (2015). Real-time hands-free immersive image navigation system using Microsoft Kinect 2.0 and Leap Motion Controller. *2015 Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG).*

[21] Avidan, S., Shamir, A. (2007). Seam Carving for Content-Aware Image Resizing. *ACM Transactions on Graphics (TOG)*, 26(3), 10.

[22] Kohonen, T. (1989). *Self-organization and associative memory.* 3rd ed. New York: Springer.

[23] Kapre, N. (2009). Performance comparison of single-precision SPICE Model-Evaluation on FPGA, GPU, Cell, and multi-core processors. *2009 International Conference on Field Programmable Logic and Applications.*

[24] Stratix® 10 DSP Specification Table: *https://www.altera.com/products/fpga/stratix-series/stratix-10/ features.html#dsp*