Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information). 7/2 (2014), 108-115
DOI: http://dx.doi.org/10.21609/jiki.v7i2.264

IMPLEMENTATION OF IMAGE PROCESSING ALGORITHMS AND GENERALIZED LEARNING VECTOR QUANTIZATION TO TRACK AN OBJECT USING AR.DRONE CAMERA

Muhammad Nanda Kurniawan[1] and Didit Widiyanto[2]

[1]Faculty of Computer Science, Universitas Indonesia, Kampus UI Depok, 16424, Indonesia
[2]Faculty of Computer Science, UPN Veteran, Jakarta, Indonesia

E-mail: m_nanda07@mail.com, didit.widiyanto89@gmail.com

Abstract

In this research, Parrot AR.Drone as an Unmanned Aerial Vehicle (UAV) was used to track an object from above. Development of this system utilized some functions from OpenCV library and Robot Operating System (ROS). Techniques that were implemented in the system are image processing algorithm (Centroid-Contour Distance (CCD)), feature extraction algorithm (Principal Component Analysis (PCA)) and an artificial neural network algorithm (Generalized Learning Vector Quantization (GLVQ)). The final result of this research is a program for AR.Drone to track a moving object on the floor in fast response time that is under 1 second.

Keywords: *Unmanned Aerial Vehicle (UAV), AR.Drone, GLVQ, object tracking*

Abstrak

Pada penelitian ini, Parrot AR.Drone digunakan sebagai pesawat tanpa awak untuk menjejaki sebuah objek dari atas. Pengembangan sistem ini memanfaatkan beberapa fungsi dari pustaka OpenCV dan Robot Operating System (ROS). Teknik-teknik yang diimplementasikan pada sistem yang dikembangkan adalah algoritma pengolahan citra (*Centroid-Contour Distance (CCD)*), algoritma ekstraksi fitur (*Principal Component Analysis (PCA)*), dan algoritma jaringan syaraf tiruan (*Generalized Learning Vector Quantization (GLVQ)*). Hasil akhir dari penelitian ini adalah sebuah program untuk AR. Drone yang berfungsi untuk menjejaki sebuah objek bergerak di lantai dengan respon waktu yang cepat dibawah satu detik.

Kata Kunci: *Pesawat tanpa awak, AR.Drone, GLVQ, penjejakan objek*

1. Introduction

Unmanned Aerial Vehicle (UAV) applications have emerged to scientific research. Problems that found on those research are very broad in range from hardware design, aerodynamic calculation, object tracking module, localization, and coordination between UAVs. Now, the race between researcher to build autonomous UAV system has emerged [1].

Image processing techniques were used in many UAV's applications to make they have capability to recognize object. Object recognition algorithms were developed in many research to solve various problems. In [2] a system was designed to do tracking and landing on moving ground target yet, another task is to classify environment. In [3], they conducted research to classify vegetation species at farmland in Northern Australia. In [4], they used four classification techniques and two feature extraction techniques to do automatic vehicles detection. Purpose of the system they develop is to replace patrol service that rely on pilot visual inspection. The weakness of that conventional system is the inspection was done in low altitude that would improve the risk. Moreover, that system sometimes caught by weather condition restriction.

Techniques that were used are vary depend on the purpose of the research. In [5], FAST corners detection and least median square estimation were used. Besides that, target's states are estimated by using Kalman Filter. They also used BRIEF descriptors to made target localization more robust. In [6], Han Yu and his colleagues developed a novel video tracking algorithm. That algorithm integrates differential evolutionary particle filter and color histogram character. In [7], Shafique and Sheikh considered elliptical object detection as their main concern. Their proposed algorithm which based on Line Following Technique (LFT) is used to recognize circle, elliptical and nearly elliptical objects. They also implemented Hough Transform to compare the results.

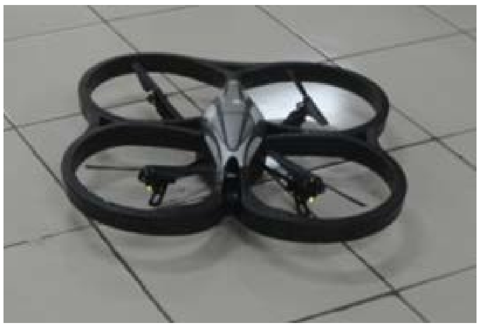

Figure 1. AR.Drone with Outdoor Hull.

We conducted research on implementation of some computer vision and machine learning algorithms to make the UAV did the mission (task). The task was it had to follow a specific object (a cap) on the floor. We utilized some algorithms, such as gray-level slicing, Centroid-Contour Distance (CCD), Principal Component Analysis (PCA), Generalized Learning Vector Quantization (GLVQ) [8] and standard Proportional Controller [9]. We will describe those techniques more detail in Section 2.

2. Methods

In this research AR.Drone quadcopter and a computer with standard Wi-Fi interface were used. The drone consists of two cameras, some other sensors, four propellers and control board. It has carbon fiber support structure so it is quite lightweight. The battery can provide enough energy to make AR.Drone flying continuously up to ±10 minutes. When the drone power is being switched on, an ad-hoc Wi-Fi appears and a computer can connect to it. After the connection has established, the user may control the drone and get the information such as altitude, battery status and other sensors data. The AR.Drone has two body cover (called hull), outdoor and indoor hull. The AR. Drone with outdoor hull can be seen at Figure 1.

Because the AR.Drone has limited hardware performance (to run many processes quickly), all computation processes are running in the computer. So, the drone only know where to move from the information send by the computer via Wi-Fi while continuing to capture images continuously by using its camera. The AR.Drone has two cameras. The first camera is on the front side and the other camera is at the bottom. The bottom camera was used to implement object detection and tracking system in this research.

The system was built and run on Robot Operating System (ROS) environment and developed using C++ programming language. ROS is an open source meta-operating system designed to build the robot systems [10]. With ROS, we do not have to "reinvent the wheel" because it already has basic modules to develop the system. Besides, ROS uses the concept of peer-to-peer communication network between programs. So, some programs could run simultaneously to do the task.

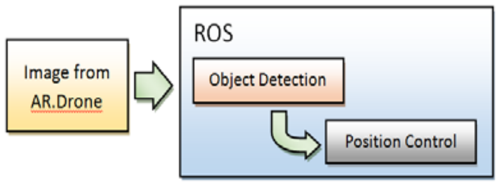

Figure 2. Overall System (black box view).

In this research for example, we run some programs to do tracking process. Those programs are the driver of AR.Drone, object detection module and position control program. The driver for the drone we used was AR.Drone Brown package on ROS. Object detection module and position control module that we used were developed by ourselves. The overall system in one time running can be seen at Figure 2. That system run continuously over time, so each specific time, the system process images captured by AR.Drone and give coordinate message.

For basic image handler and processing, we used OpenCV (Open Computer Vision) library with ROS. That library is in vision_opencv stack which consist of several packages [11]. One of the most important package in that stack is cv_bridge that has task to bridge between ROS image message and OpenCV data structure.

We will give brief description about some methods we used. First of all, we used standard grayscaling from raw images and then produced binary image from the grayscale image with fixed threshold. After that we process the binary image with Centroid-Contour Distance (CCD) algorithm. This is used for getting features from an image base on object's contour. The features are obtained by measuring the distance from a fixed point to surrounding points at the object contour. Wang et. al. used this technique to get features from leaf image which is a single object (for image detection/recognition system) [12]. We used this technique because our targeted object (the cap) recognition is near characteristic with leaf recognition. Figure 3 illustrades CCD algorithm. We will described more about the implementation of this algorithm in Section 4.

After we got CCD features, we used the popular Principal Component Analysis (PCA) algorithm to further reduce the features. PCA is a technique that consist of statistics and linear algebra concepts, especially those are related to dominant Eigen vectors [13]. Let we describe this algorithm in brief explanation.

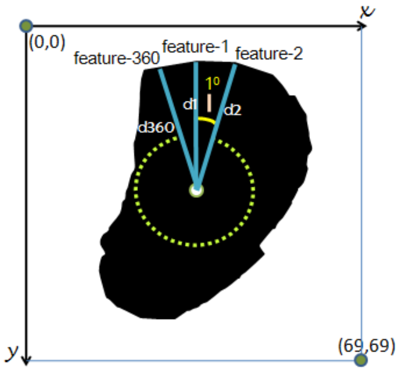

Figure 3. Illustration of Centroid-Contour Distance (CCD).

First of all, we find averages (means) from the matrix data and then we calculate their covariance. After we got the covariance matrix, we decompose the Eigen values with their corresponding Eigen vectors from that matrix. The decomposition is as the following equation(1).

$$S_n = \bar{\lambda} U_n \qquad (1)$$

In that equation, S_n is covariance matrix, $\bar{\lambda}$ is Eigen values and U_n is Eigen vectors which correspond with the values. We modified methods from JAMA: A Java Matrix Package [17] to become functions in C / C++ in this process. After that, we sorted the Eigen values from biggest to smallest value so their corresponding Eigen vectors follow them. After that we discard some small Eigen values and vectors (from smallest to a number of that). Finally, we multiply the original data matrix with reduced Eigen vector matrix. From that operation, we get the final reduced data matrix. The flow of the PCA process can be seen at Figure 4.

Data that has reduced by PCA process is used for neural network training/testing process using Generalized Learning Vector Quantization (GLVQ) algorithm. The GLVQ algorithm was developed by A. Sato and Yamada in 1995 [8]. This algorithm is variation of Learning Vector Quantization (LVQ) which assured convergence even though the data is not sorted. This is happening because the method is based on cost function minimization, i.e. miss-classification error with optimization of gradient descent [15].

The process is started by calculating miss-classification error using equation(2).

$$\varphi(x) = \frac{d_1 - d_2}{d_1 + d_2} \qquad (2)$$

In this formula, d_1 and d_2 is the distance of winner and runner-up class. After we get the error, we use sigmoid function as cost function. The function use that error as one of its parameters. The other parameter is the iteration (t). The function is as the following equation(3).

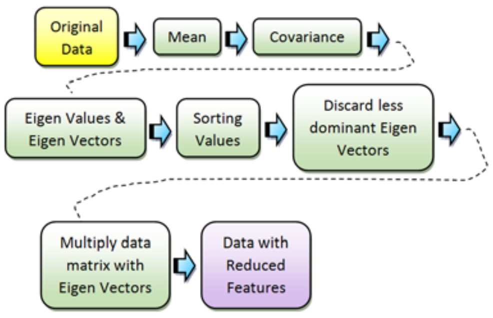
Original Data
Mean
Covariance
Eigen Values & Eigen Vectors
Sorting Values
Discard less dominant Eigen Vectors
Multiply data matrix with Eigen Vectors
Data with Reduced Features

Figure 4. Principal Component Analysis (PCA) process flow

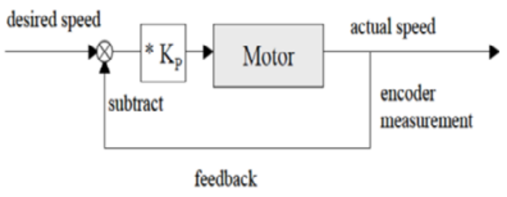

Figure 5. Proportional Controller Schematic [9].

$$f(\varphi, t) = \frac{1}{1 + e^{-\varphi t}} \qquad (3)$$

Then, the sigmoid function is differentiated to get "gain factor" for weight update process using equation(4).

$$\frac{\delta f}{\delta \varphi} = f(\varphi, t)(1 - f(\varphi, t)) \qquad (4)$$

From equation(4), the weight of winner and runner up in neural network's neurons can be updated using the equation(5) and equation(6).

$$w_1 \leftarrow w_1 + \alpha \frac{\delta f}{\delta \varphi} \frac{d_2}{(d_1 + d_2)^2} (x - w_1) \qquad (5)$$

$$w_2 \leftarrow w_2 - \alpha \frac{\delta f}{\delta \varphi} \frac{d_1}{(d_1 + d_2)^2} (x - w_2) \qquad (6)$$

In that formula, x is the data and α is learning rate. Equation(5) and (6) is for training process. For testing process, the LVQ family algorithm only used distance calculation formula to find Euclidean distance.

In tracking process, we used the proportional controller to control the velocity. This controller is used because sometimes, drastic changes of fixed motor controller does not produce a smooth control behavior [9]. The formula of proportional controller is as given by equation(7).

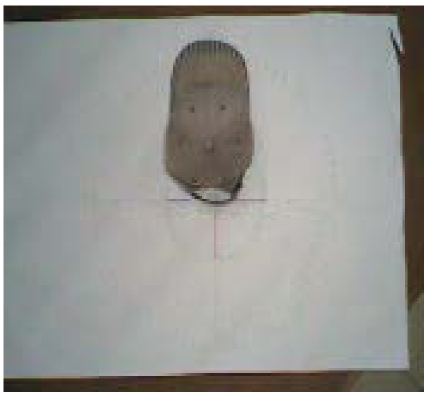

Figure 6. Full image captured by AR.Drone bottom camera.

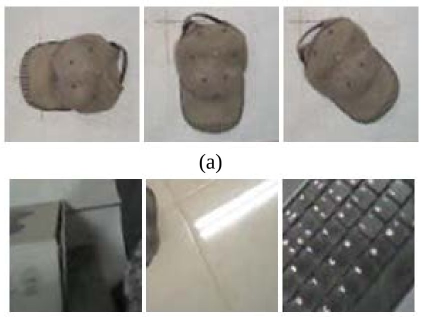

Figure 7. Sample of cut images: (a) cap images, (b) non-cap images.

$$R(t) = K_P(v_d(t) - v_a(t)) \quad (7)$$

In that formula (that always run over time t), R is the motor output, v_d is desired velocity and v_a is actual velocity. The most important component of this system is K_P, constant value / "controller gain". This value will decide the controller behavior. The schematic of P controller can be seen in Figure 5.

The difference between desired and actual velocity is the error that want to control. That error than multiplied by K_P to produce the output. The bigger the value of K_P, the response will be faster. However, that value should still be restricted because if it is too big, it will cause oscillation on the system [9].

Data Preparation

After literature study, data preparation consist of images which representing the class of cap and non-cap. We captured the images manually by the AR.Drone bottom camera. There are some criteria that must be concerned in capturing the images because they will decide whether the data good or not. Those criteria are: 1) The cap (object) should be contrasted with the background; 2) The combination of hat orientation must be considered; 3) The distance between cap and the AR.Drone camera must be 1 meter. Because in the experiment the drone set to only fly 1 meter above the object.

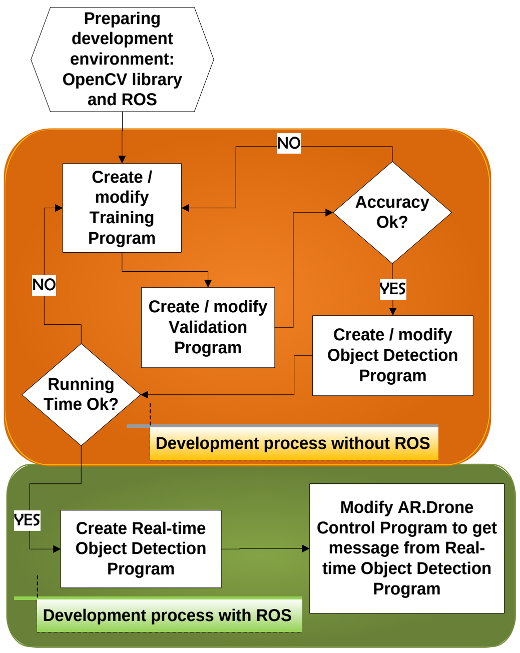
Preparing development environment: OpenCV library and ROS
Create / modify Training Program
NO
Accuracy Ok?
NO
Create / modify Validation Program
YES
Create / modify Object Detection Program
Running Time Ok?
Development process without ROS
YES
Create Real-time Object Detection Program
Modify AR.Drone Control Program to get message from Real-time Object Detection Program
Development process with ROS

Figure 8. System development process overview.

One of the raw image we got by the size of 176×144 pixels can be seen at Figure 6. That image then cut to become 70×70 pixels because with that size in 1 meter distance, the cap is fully shows in a good proportion. So, the image of this size will be used in the experiment. Some cut images can be seen at Figure 7.

System Development Process

After we got the data, we developed the system to do object detection and tracking. The big picture of the development process can be seen at Figure 8. In the development process, we divided the process into two big steps, without ROS (we called Stage 1) and with ROS (we called Stage 2). We will describe the process step by step.

The first thing we did before building the code is preparing the environment and library (ROS and OpenCV). After the installation step was successfully done, we develop the training program which consists of pre-processing module and neural network module. In pre-processing module we used image processing techniques and feature extraction algorithm (PCA) that already described in Section 2. In neural networks module, we implemented GLVQ training formula that also described in Section 2.

In fact, the goal of training program is to get sorted eigen vectors and weights of neural netwo-

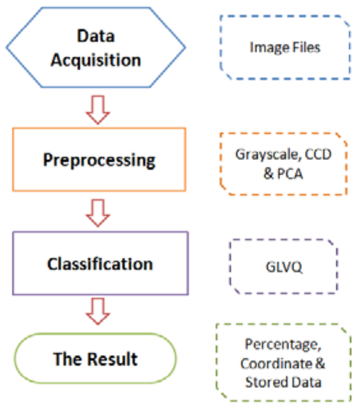

Figure 9. Stage 1 program flow.

rk neurons. So we described this from the first st-

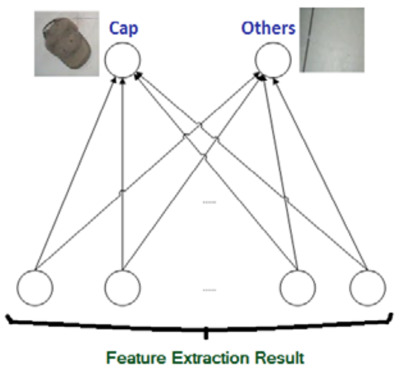

Figure 10. Neural network architecture in experiment.

ep to the end. First of all, the program read the data as folders of 70 × 70 pixels images that already divided into classes (cap and others). The images from that folder are loaded one by one by the function from OpenCV, `cvLoadImage`. By using this function, we chose the data we want to use, either BGR (blue-green-red) or grayscale [16]. In this implementation, we used grayscale image because later technique we used (CCD) is consider about the shape and not the colors.

After the grayscale image has gotten, the next step is convert it into two values only, 0 and 255. To do that, technique named gray-level slicing (simple thresholding technique for image) is used. With this technique, some range of grayscale value become low value and the others become high value [14]. We considered the low value is 0 (totally black) and the high value is 255 (white).

After the image only has two values, it then directed to Centroid-Contour Distance (CCD) process. As we already described in Section 2, this method is calculating distance of points in contour of object to a pre-decided fixed point. In this process we decided the fixed point is the coordinate of 35 × 35 in pixel of image. Then, we decided to

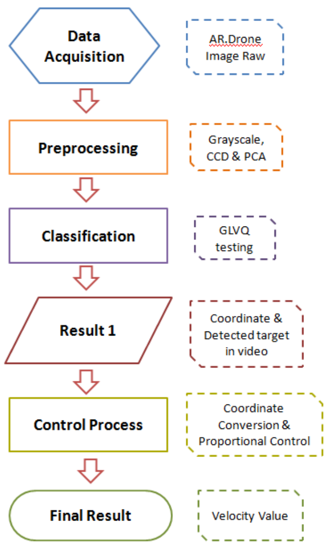

Figure 11. Stage 2 program flow.

get the distance for every 1 degree, so we will get 360 feature in full circle. The illustration of the features can be seen at Figure 3. The process from image reading to CCD run continuously until all images are processed. For every image processing steps, the data with label (class) is stored in matrix data.

After we got labeled data that has features from CCD process, we further reduced the feature with PCA. We did this because we want the run time in one cycle process as short as possible at real-time run using AR.Drone. The eigen vectors reduce the feature in this process and also stored in a file named `eig_vec.csv` to reduce feature at testing and real-time run later.

After PCA process, the data is processed with GLVQ algorithm. This is the last step in the training program. The architecture of neuron for this process can be seen at Figure 10. From the process, we will get weights of neurons and store those values in a file named `bobot.csv` that will be used in testing and real-time run.

As we seen at Figure 8, after we finished with training program, we went to program validation. This validation program will produce the output as percentage of accuracy. Process from reading the data to CCD process is same with training program. In PCA step, the data matrix is directly multiply by Eigen vectors from `eig_vec.csv`. In GLVQ step, the features are directly compared with the weights from bobot.csv by using euclidean distance. For every true classification, the "hit" then add by 1. After the data finished processed with neural network testing step, the hit then divided by the number of data. That operation will produce accuracy. If the accuracy is not really satisfying, we modified some parameters in training

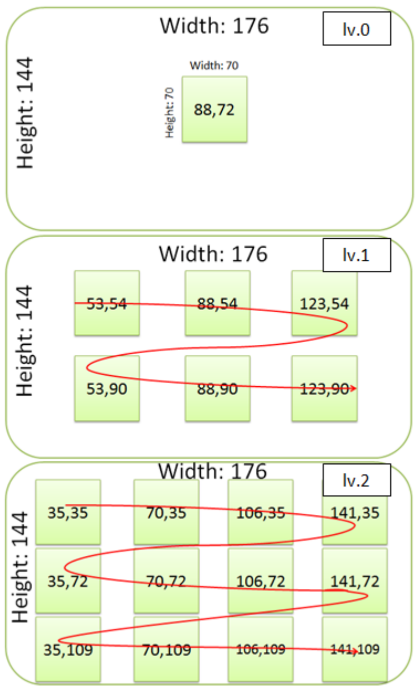

Figure 12. Grid Check.

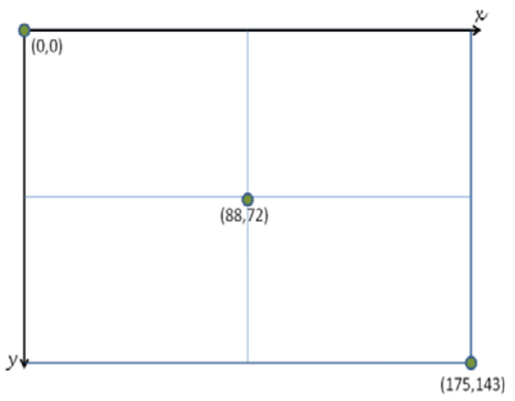

Figure 13. Pixel coordinate in AR.Drone bottom camera.

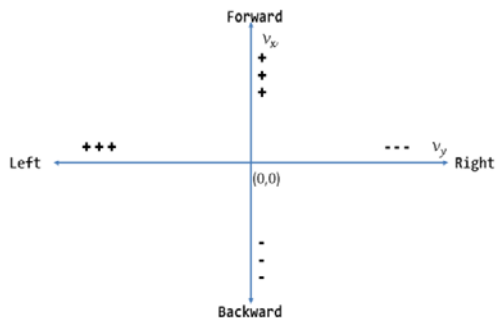

Figure 14. AR.Drone velocity coordinate.

program like number of CCD features, number of features after PCA learning, learning rate or the number of learning cycle (called epoch).

After we already satisfied with the accuracy, we built/modified object detection program. This program reads the image of 176×144 pixels size instead of folders of 70×70 pixels images. The output is object coordinate in that pixel area if an object has been detected. The most important thing about this program is this program must run in reasonable time, which is under 1 second. It is because this program will modified to real time object detection program using ROS. Real time robot application must be fast enough to response, so we should not move to the next step of development if the running time still not satisfying.

To fulfill the requirement regarding running time, we propose to only check some grids of 70×70 pixel size in full 176×144 pixel size. Because of that, we named this step as "Grid Check". The rule is we only chose 19 grids to be checked. Illustration of Grid Check can be seen at Figure 12. We divided the grids from level 0 to level 2. It checked from center of image to its surrounding area. In the experiment using AR.Drone, it showed great performance that allows the drone response quickly in reasonable movements. The overall of flow program without ROS can be seen at Figure 9.

In the implementation using ROS, the first thing we must do in this step is modifying the previous object detection module. This module must receive image message from AR.Drone that is `ardorne/image_raw`. It is a bit different with previous program which get offline image from the local storage. Besides that, some functions and variables must be changed or modified. Actually, this task is not really hard if the programmer already keep up with ROS.

The last task in development process was modifying the AR.Drone control program. This program must get object coordinate message from real time object detection program. Since the object coordinate is in pixel coordinate, in control program, coordinate conversion must be done first. The coordinate conversion use the following equation(8) and equation(9).

$$x_{vel} = c_y - y_{pix} \tag{8}$$

$$y_{vel} = -(x_{pix} - c_x) \tag{9}$$

In that equations, c_x and c_y is the center of 176×144 pixel, i.e. 88 and 72. To see this more clearly, we should refer to Figure 13 and 14. After we got the velocity, we used proportional controller to make a smooth movement. From this point, we said that the development process was done. The overall flow of program without ROS can be seen at Figure 11.

TABLE I
EXPERIMENTAL RESULT

Experi-ment	Testing Data	PCA	Accuracy	Detected	Running Time
1	testNonCap and testCap	360	92.54%	Yes	1.24 ms
2	testNonCap and testCap	1	74.63%	No	-
3	testNonCap and testCap	300	92.54%	Yes	0.91 ms
4	testNonCap and testCap	200	92.54%	Yes	0.88 ms
5	testNonCap and trainCap	200	100.00%	Yes	0.88 ms

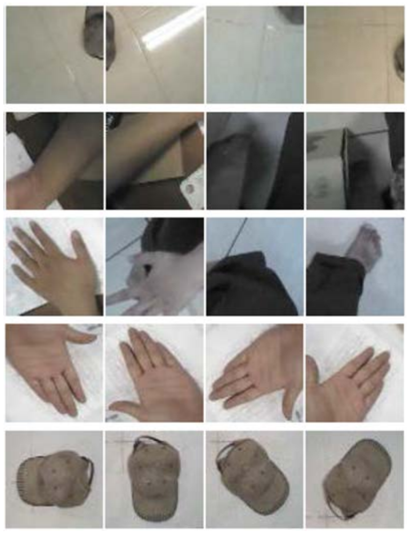
Figure 15. Training data (trainNonCap: first 4 rows, trainCap: the last row).

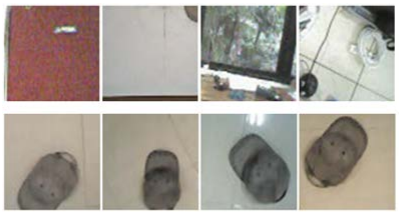
Figure 16. Testing data (testNonCap: top rows, testCap: bottom row).

3. Results and Analysis

In this section, we showed the experimental result we did. In the experiment, we used packet of training and testing data (in image folders). Training data images can be seen at Figure 15 and the testing data can be seen at Figure 16.

The experiment result can be seen at Table 1. In that table, we conclude that the experiment produce good results except for PCA which remains 1 feature. Because the parameter of PCA in 200 features is good and run quickly, we choose that for real time object detection. For the result in real time detection using ROS, we can see the experiment and screen shot at Figure 17 and 18.

4. Conclusion

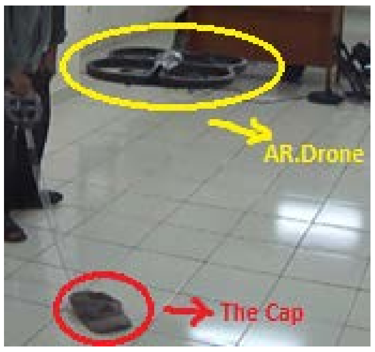

Figure 17. Real experiment using AR.Drone.

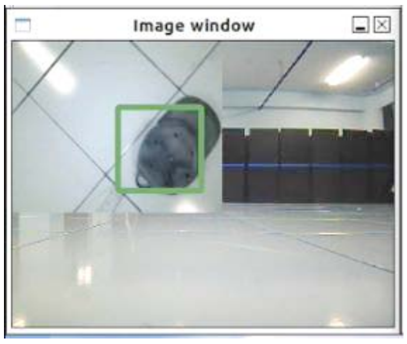

Figure 18. Screen shot of real time object detection.

Experiment results show that the implementation of image processing and neural network GLVQ algorithm to do object detection and tracking was success. The running time is short and it have good accuracy in GLVQ validation that is more than 90 percent in average. The running time, which is under 1 millisecond, that shows at Table 1, is very good for real time application that need quick response. This short time is also coming from great contribution of feature extraction from CCD technique after doing some image processing techniques which are grayscaling and thresholding. The feature data reduction using PCA and also Grid Check techniques also give contribution for this.

In the future works, the real time running must be done in various heights and in the outdoor. Besides that, the object recognition must be develop to track or detect more objects. The development of swarm robots is also one of the greatest challenge in the future. Hopefully those works can be done in the near future.

Acknowledgement

This work was also coordinated with big research under research fund which was received by Dr. Petrus Mursanto and Dr. Eng. Wisnu Jatmiko in the Faculty of Computer Science Universitas Indonesia. Gratitude is also addressed to some students in the Lab of Computer Networks, Architecture and High Performance Computing in the faculty with good discussions and some help for preparing the experiments.

References

[1] E. Marris, “Fly, and Bring Me Data,” in *NATURE*, Vol. 498, ch. News Feature, pp. 156–158, Macmillan Publishers Limited, June 2013.

[2] J. H. Choi, Won-Suk Lee and H. Bang, “Helicopter Guidance for Vision-based Tracking and Landing on a Moving Ground Target,” in *11th International Conference on Control, Automation and Systems*, pp. 867–872, Daejeon, Korea, 2011.

[3] A. Reid, F. Ramos and S. Sukkarieh, “Multi-Class Classification of Vegetation in Natural Environments Using an Unmanned Aerial System,” in *International Conference on Robotics and Automation*, pp. 2953–2959, Shanghai, China, May 2011.

[4] J. Gleason, A. V. Nefian, X. Bouyssounousse, T. Fong and G. Bebis, “Vehicle Detection from Aerial Imagery,” in *International Conference on Robotics and Automation*, pp. 2065–2070, Shanghai, China, May 2011.

[5] M. Siam, R. ElSayed and M. ElHelw, “On-board Multiple Target Detection and Tracking on Camera-Equipped Aerial Vehicles,” in *International Conference on Robotics and Biomimetics*, pp. 2399–2405, Guangzhou, China, December 2012.

[6] H. Yu, W. Longsheng, and Y. Yunzhi, “A Video Tracking Algorithm For UAV Based On Differential Evolution Particle Filter,” in *31st Chinese Control Conference*, pp. 3955–3959, Hefei, China, July 2012.

[7] S. Shafique and N. M. Sheikh, “Simple Algorithm for Detection of Elliptical Objects in Remotely Sensed Images for UAV Applications,” in *International Bhurban Conference on Applied Sciences & Technology*, pp. 258–264, Islamabad, Pakistan, January 2009.

[8] A. Sato and K. Yamada, “Generalized Learning Vector Quantization,” in *Advances in Neural Information Processing Systems (NIPS)*, Vol. 7, pp. 423–429, 1995.

[9] T. Bräunl, *Embedded Robotics, Mobile Robot Design and Applications with Embedded Systems*, 2nd ed., Springer, Germany, 2006.

[10] Documentation, http://www.ros.org/wiki, retrieved June 18, 2012.

[11] vision_opencv, http://wiki.ros.org/vision_opencv, retrieved July 3, 2014.

[12] Z. Wang, Z. Chi and D. Feng, “Shape based leaf image retrieval,” in *IEEE Proceedings-Vision, Image and Signal Processing*, Vol. 150, No. 1, pp. 34–43, 2003.

[13] Bernard Kolman and David R. Hill, *Elementary Linear Algebra 8th Edition*, Pearson International Edition, New Jersey, 2004.

[14] Rafael C. Gonzales and Richard E. Woods, *Digital Image Processing Second Edition*, Pearson International Edition, New Jersey, 2002.

[15] I Made Agus Setiawan, “Sistem Pengenalan Kelainan Aritmia menggunakan Metode Fuzzy-Neuro Generalized Learning Vector Quantization,” Master Thesis, Faculty of Computer Science, Universitas Indonesia, 2011.

[16] Gary Bradski and Adrian Kaehler, *Learning OpenCV - Computer Vision with the OpenCV Library*, O’Reilly Media, Inc., Sebastopol, CA, 2008.

[17] Joe Hicklin, “JAMA : A Java Matrix Package,” http://math.nist.gov/javanumerics/jama/, retrieved June 18, 2012.