IOP Conf. Series: Materials Science and Engineering **152** (2016) 012015 doi:10.1088/1757-899X/152/1/012015

Design and implementation of a vision-based hovering and feature tracking algorithm for a quadrotor

Y H Lee[1] and J S Chahl[1,2]

[1]Division of Information Technology and Engineering, The School of Engineering, University of South Australia, Australia
[2]Joint & Operations Analysis Division, Defence Science and Technology Group, Australia

*javaan.chahl@unisa.edu.au

Abstract. This paper demonstrates an approach to the vision-based control of the unmanned quadrotors for hover and object tracking. The algorithms used the Speed Up Robust Features (SURF) algorithm to detect objects. The pose of the object in the image was then calculated in order to pass the pose information to the flight controller. Finally, the flight controller steered the quadrotor to approach the object based on the calculated pose data. The above processes was run using standard onboard resources found in the 3DR Solo quadrotor in an embedded computing environment. The obtained results showed that the algorithm behaved well during its missions, tracking and hovering, although there were significant latencies due to low CPU performance of the onboard image processing system.

1. Introduction

Significant number of research programs on the vision-based control for Unmanned Aerial Vehicles (UAVs) are ongoing actively as an alternative sensor to GPS [1]. In fact, in case of ground vehicles, many vision-based control algorithms have already been proposed so far. However, when it comes to UAVs, specific additional challenges should be considered such as rigid body objects moving in 3D, dynamic effects and external perturbations, and relatively poor quality of video sequences filmed from the vibrating movement common to UAVs [2]. Simultaneous localization and mapping (SLAM) is one of the more popular technologies for position control for UAVs [1]. One of the popular approaches to the visual simultaneous localization and mapping (VSLAM) algorithm was introduced by Klein and Murray [3]. Their approach splits the task into two threads: one for tracking and the other for mapping. The advantage of this separation is that by running both threads at different rates (the tracking thread runs faster than the mapping thread), the tracking thread can update the position more quickly while the mapping thread can employ a slower but more powerful technique. In addition, since the mapping algorithm does not process all frames, the image processing work can have more resource allocated, especially when the UAV moves slowly or hovers [4]. When VSLAM uses a single camera, it cannot provide the scale factor that is a crucial parameter for calculating absolute velocity and position in 3D of the vehicle. In order to acquire this scale factor, several methods were proposed such as employing air pressure sensor or online multi rate extended kalman filter (EKF) based on inertial sensors [5].

Another vision-based control area that is attracting many research teams is the object tracking field. A research team in University of Toronto has carried out vision-based hovering and object tracking, and tested several related algorithm such as optical flow, colour-based image processing, kalman filter

tracking [6]. Their work achieved quite desirable results despite some limitations such as the camera was fixed on the ground staring upward to film the quad-rotor and the image processing tasks were done on ground station. Other interesting research in object tracking is a quad-rotor UAV platform that tracks a moving target on the ground based on visual information from a camera [7]. An interesting point is that, in case of loss of track of the moving target, the quad-rotor increases its altitude so as to get wider field of view. However, a limitation was that their algorithm was only verified for a moving object with purely translational motion. Similar UAV vision-based on-board object tracking research has also been conducted in [8]. The research employed a low-cost on-board monocular vision system and designed a closed-loop object tracking controller. With colour detect vision algorithm and closed-loop controller, the object tracking system showed a stable result though it has some limitations such as stationary object, light condition and restricted background around the object.

The aim of this work is to enable hovering and object tracking for UAVs using vision data from a monocular camera. For this goal, the vision system should possess following capabilities: detecting the target object, identifying the current pose $(x,y)^T$ and also orientation ɵ relative to the target object, and approaching the object and holding the position around the object. For the first and second capabilities, image processing algorithms provided by OpenCV libraries were employed such as SURF [9] and moments [10] based image processing. For the third capability to steer the UAV, the 'dronekit' python library for 3DR Solo was employed. Robust detect algorithms, filtering and compensating corrupted position data due to external environment such as wind, precise control algorithm to steer the UAVs were key challenges for this project.

2. Methodology

2.1. Quadrotor platform

3DRobotics has introduced the Solo quadrotor, which is a Linux based drone and shown in Figure 1. Even though its main design purpose is a high-tech toy for entertainment, it can be a solid and well-integrated platform for drone-related research due to its standard development environment – Linux, OpenCV, dronekit. The hardware specification of the Solo quadrotor is listed in Table 1.

Figure 1. Front picture of the solo with GoPro camera

Table 1. The solo hardware specification

Component	Specification
Propeller	24cm diameter, 144cm pitch
Weight	1.5kg
Dimension	25cm tall, 46cm motor to motor
Flight battery	Lithium poly 5200mAh
Max Payload	420g
Autopilot	Pixhawk 2
Software	APM copter

Solo's control is carried out by two embedded boards: Pixhawk 2 and IMX6, as shown in Figure 2. The Pixhawk 2 is responsible for low-level control like motor, sensors and flight control. Meanwhile, the IMX6 performs the role of high-level controller such as video streaming and communication with the remote controller. All control messages from the IMX6 are passed to the Pixhawk 2 in the form of mavlink protocol commands. Specification for both Pixhawk 2 and IMX6 is provided in Table 2.

2.2. Image processing

Several image processing algorithms were employed in this study for object detection, pose estimation and tracking. Speed Up Robust Features (SURF) algorithm was used for target detection, then after applying image processing, the pose information of the target relative to the quadrotor was estimated. OpenCV libraries provided a standard framework and architecture for the above image processing algorithms that the software used to implement the algorithms.

Figure 2. Image of Pixhawk 2 and IMX6

Table 2. Specification for embedded boards in the solo

	CPU	Memory	OS
Pixhawk2	32 bit ARM Cortex M4 168MHz	256KB	NuttX RTOS
IMX6	32bit ARM Cortex A9 1GHz	512MB	Yocto embedded Linux

SURF algorithm extracts some keypoints and descriptions from an image, and then using them to detect the same scene or object from other images [9]. It uses an intermediate image representation known as the integral image. Given a input image *I* and point (*x* , *y*), integral image *IΣ* is calculated by the sum of the values between the point and the origin as in Equation 1.

$$I\Sigma(x,y) = \sum_{i=0}^{i\le x}\sum_{j=0}^{j\le y} I(x,y) \tag{1}$$

The SURF detector is based on the determinant of the Hessian matrix [9]. The Hessian matrix H is the matrix of partial derivatives of function, f as given in Equation 2.

$$H\big(f(x,y)\big) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x\,\partial y} \\ \dfrac{\partial^2 f}{\partial x\,\partial y} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix} \sigma \tag{2}$$

The determinant of this matrix is calculated by Equation 3.

$$\det(H) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x\,\partial y}\right)\left(\frac{\partial^2 f}{\partial x\,\partial y}\right) \tag{3}$$

This determinant value is used to classify the maxima and minima of the function. Then, calculate the Hessian matrix H, as function of both space X = (x, y) and scale σ as shown in Equation 4.

$$H(X,\sigma) = \begin{bmatrix} L_{xx}(X,\sigma) & L_{xy}(X,\sigma) \\ L_{xy}(X,\sigma) & L_{yy}(X,\sigma) \end{bmatrix} \tag{4}$$

where $L_{xx}(X,\sigma)$ refers to the convolution of the second order Gaussian derivatives $\frac{\partial^2 g(\sigma)}{\partial x^2}$ with the image at point X = (x, y) and similarly for L_{xy} and L_{yy} [11].

In order to achieve invariance to image orientation, SURF uses wavelet responses in horizontal and vertical directions for neighbourhood of size 6s, and applies Gaussian weights. The feature descriptors, the wavelet responses in the horizontal and vertical directions are used. The algorithm constructs a square window around the interest point. Then the window is divided into 4x4 pixel sub-regions. The horizontal and vertical wavelet sample points are collected in the form of Equation 5.

$$v_{subregion} = [\Sigma dx, \Sigma dy, \Sigma|dx|, \Sigma|dy|] \tag{5}$$

Using the binary image resulting from the above image processing, the pose of the target object can be calculated. First, the spatial moments of intensity distribution for binary images are calculated using Equation 6.

$$M_{i,j} = \sum_{x,y}\left(I(x,y)x^i y^j\right) \tag{6}$$

Then, central moments are used to calculate the total area of the intensity distribution:

$$\mu_{i,j=\sum_{x,y}(I(x,y)(x-x_c)^i(y-y_c)^j)} \quad (7)$$

The coordinate (x_c,y_c), which is the centre of the target object, can be calculated as in Equation 8 [12].

$$x_c = \frac{M_{1,0}}{M_{0,0}}, y_c = \frac{M_{0,1}}{M_{0,0}} \quad (8)$$

It is highly possible that taking pictures from constantly moving quadrotor will also generate noisy information. Thus some filtering algorithms were needed to compensate for this noise and determine the true value. In OpenCV, the Kalman filter algorithm can be used for this purpose. In addition, it can be used to predict future position in case of temporary loss. The Kalman filter estimates the state of a process by using a form of feedback controller. Firstly, the filter estimates the states and then obtains feedback in the form of measurements. Equations for the Kalman filter can be categorized into two groups [13]: the time update equations that are responsible for obtaining the *a priori* estimates for the next time step -> *predictor* and also the measurement update equations that are responsible for the feedback -> *corrector*.

Suppose X is a discrete-time process, Z is a measurement, and W and V represent the process and measurement noise, respectively. The state of X and measurement Z equations can be expressed as follow [14]:

$$X_{k+1} = AX_k + W_k \quad (9)$$

$$Z_k = HX_k + V_k \quad (10)$$

Then, the Kalman filter estimates the state X of time k and corrects the prediction using measurement Z using the following equations [13] :

Time update (prediction):

$$\hat{X}_k^- = A\hat{X}_{k-1} + Bu_{k-1} \quad (11)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (12)$$

Measurement update (correction):

$$K_k = P_k^- H^T(HP_k^- H^T + R)\text{-}1 \quad (13)$$

$$\hat{X}_k = \hat{X}_k^- + K_k(Z_k - H\hat{X}_k^-) \quad (14)$$

$$P_k = (I - K_K H)P_k^- \quad (15)$$

3. Implementation

For this project, two processes were developed that both processes run on the IMX6 board embedded in the Solo – the flight controller and the vision controller. In the case of the flight controller, it is a python script using dronekit python library. Therefore, there is no need to be concerned about platform dependency. However, when it comes to vision controller, it was developed using C++ language and using OpenCV C++ libraries. Cross development environment should be set up as following: build OpenCV libraries for the arm platform and set-up cross development environment to build the vision controller to make it run on the arm platform. Firstly, the complier should be the same one that has been used to build the root-file-system on the IMX6 board. To find this, the following command was run with a binary file in current root-file-system in the solo.

```
$ readelf –l ‘binary file name’ | grep interpreter
```

As a result of above command, it was found that ‘arm-linux-gnueabihf-gcc-4.8’ was employed to build the root-file-system in the Solo.

To install OpenCV libraries into the IMX6 board, first OpenCV libraries were built with the above cross complier. Then, OpenCV static and shared libraries were generated. These libraries were then copied into an arbitrary directory in the IMX6. After that, the following was set up to ensure that the application can refer to the shared libraries correctly during run time.

```
$ vi /etc/ld.so.conf  // open shared library configuration file
     Add a directory where you copied the shared libraries
$ ldconfig  // reconfigure with newly added directory
$ ldconfig –p // check the opencv libraries correctly registered
```

Dronekit-python, which is already installed in the IMX6, allows developers to create applications that can run on-board and communicate with the auto-pilot flight controller (Pixhawk 2) [15]. It provides application program interfaces in the forms of mavlink protocol that applications programs running on on-board companion computer are able to receive current state, parameters information of a vehicle, as well as controlling the vehicle's movement and operation. By controlling the UAV from an on-board application using the dronekit API, researchers are able to add their enhanced functionalities such as vision-based control and artificial intelligence mission decision.

3.1. Vision controller

The vision controller was developed using C++ language and used OpenCV libraries to perform image processing tasks. It ran as a process and continued to run until it received a termination message from the root user. This is shown in Figure 3.

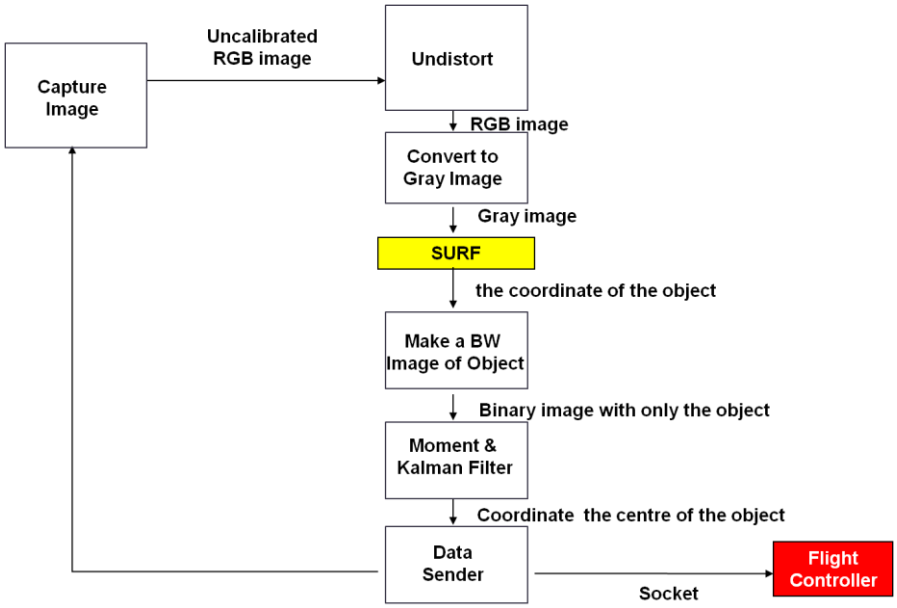

Figure 3. The vision controller flow chart

To capture images, GoPro camera was employed. It was connected to the IMX6 board via HDMI interface and used linux V4L2 (video 4 Linux 2) capture driver. The image format from the camera was YUYV. An issue was found that the capture driver in the IMX6 board allowed only 640 x 480 resolution. So, regardless of the resolution setting of the camera, all captured images are 640 x 480 resolution. For better detection, we needed to capture 720p (1280 x 720) resolution image. To resolve this issue, we did not use OpenCV capture library but ported a framework that directly controlled the capture driver using linux IOCTL kernel interface. As a target object, the symbol of the university as shown in Figure 4 is set. Every detection cycle required two images: one is a target object image and the other is a image that includes the target object. Then, the process tried to find the keypoints and descriptions for both images. Finally, it tried to find any matched keypoints and descriptions from the latter image. As a result, the target object in the latter image can be found as in Figure 5.

Figure 4. The target object

Figure 5. The detection of the target object

Technically, from previous step, what actually acquired was the coordinates of two points: upper left corner and lower right of the target object. Using this coordinate information, a binary image can be made as following Figure 6. Now, using moments functionality, the coordinates of the centre of the object can be extracted as shown in Figure 7. By calling the kalman filter API with the coordinate as an input, the filtered and predicted final coordinates was generated. Throughout the image processing task, the main purpose of vision controller to acquire the coordinate of the target object was achieved. Next, the coordinate information should be transferred to another process, which was flight controller. In order to achieve inter-process communication between these two linux processes, local UDP socket interface was selected due to its simplicity and low overhead. The structure for data communication is as in Table 3.

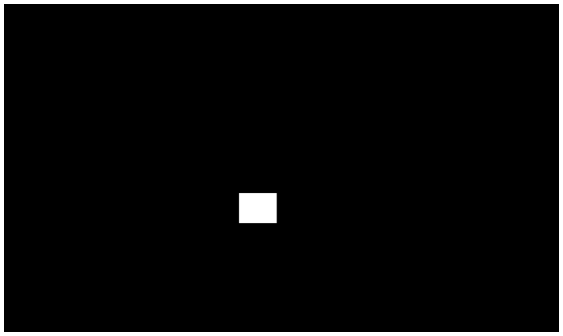

Figure 6. Binary image of the object

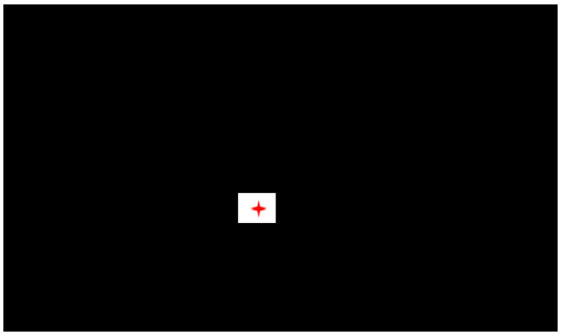

Figure 7. Finding of the coordinate of centre

Table 3. Communication data structure

Type	Name	Description
Unsigned int	Position_x	X coordinate
Unsigned int	Position_y	Y coordinate
Unsigned short	Heading	Angle between quad-rotor heading and the object position

3.2. Flight controller

The flight controller was developed using python script that primarily used dronekit python APIs to control the quadrotor. As explained previously, dronekit python generates mavlink messages to control the behaviours of the Pixhawk 2, the autopilot. Figure 8 shows an overview of the flight controller's behavior. After taking off, it waited for the pose information from the vision controller. As soon as it received the pose information, it made a decision for its next behavior based on its current position status relative to the target object. The first test point was whether the object was in the desired area, which was set as shown in Figure 9. If the object was in the area, the quadrotor hovered. If the object was not in the desired area, the next test point was the angle between the object and the heading of the quadrotor. If this angle was not in the margin, then it tried to adjust the heading by commanding the quadrotor to yaw. Next, if the angle was now in the margin, the only thing left was to command the

quadrotor to translate so as to place the object in the desired part of the image. The algorithm for this translational and angular movement is as follows.

Input : X,Y,Heading
Output: Turn Angle, Velocity
Constant: Max Speed, Angle Target, PositionX Margin, PositionY Margin, TargetX Position, TargetY Position, proportional P
Variables : PositionX Difference, PositionY Difference

Repeat
PositionX Difference = ||TargetX Position| - |X||
PositionY Difference = ||TargetY Position| - |Y||

If Angle Target **>** |Heading|
If PositionX Margin > PositionX Difference and PositionY Margin > PositionY Difference
Hovering
else
Velocity = Max speed * math.tanh(P * math.fabs(PositionY Diffence))
send_mavlink_set_velocity_msg(Velocity)
else
Turn Angle = Heading * P
send_mavlink_set_yaw_msg(Turn Angle, direction)
Until receive termination message;

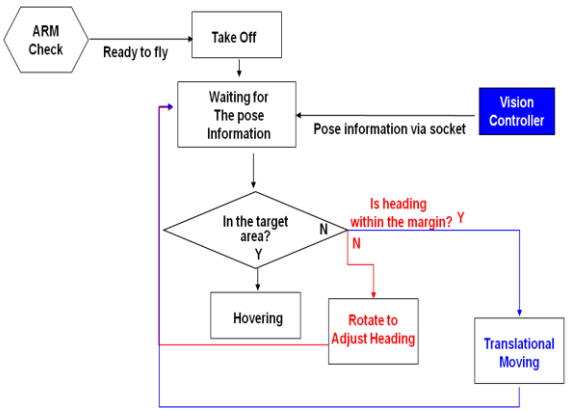

Figure 8. The flow chart of flight controller

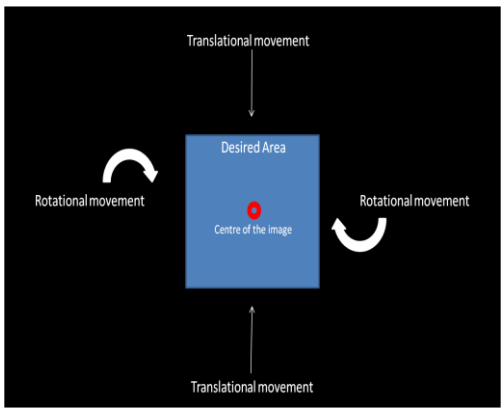

Figure 9. The desired area

4. Experiment

4.1. Hovering stabilization

In this evaluation, the test started with the condition that the target object was in the designed area and the quadrotor took off until it reached 2 m altitude. This is illustrated in Figure 10. When the quadrotor detected that the target object was in the desired area, it started hovering and was required to stay in hovering state since the target object was static. Figure 11, Figure 12 and Figure 13 show the centre pose $(x, y)^T$ of the target object during the hovering.

4.2. Tracking the target object

In this evaluation, the performance of tracking capability was verified. Due to performance limitations of the main embedded board (IMX6), the real time tracking was virtually infeasible. Therefore, in the test, the moving target object moved only a short distance (>1 m) and then waited until the quadrotor detected the new pose of the target object. Figure 14, Figure 15 and Figure 16 highlight the tracking

examples. The initial pose of the object was to the left of the quadrotor. Hence the quadrotor made an angular movement first. After the heading of the quadrotor and the object was in the margin, it made translational motion backward.

Figure 10. Quadrotor experiment set up

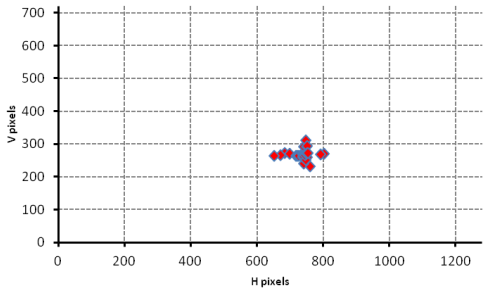

Figure 11. Trajectory of the centre of the target object

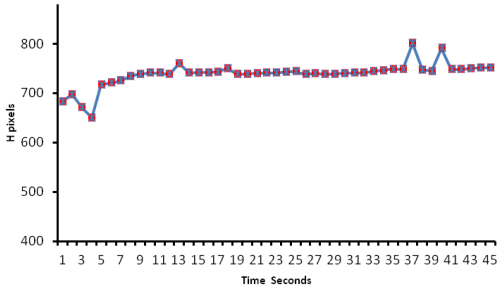

Figure 12. Trajectory of the centre (x axis) of the target object

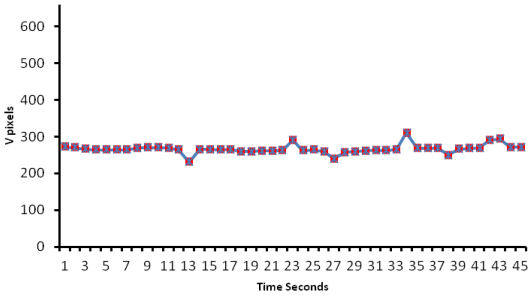

Figure 13. Trajectory of the centre (y axis) of the target object

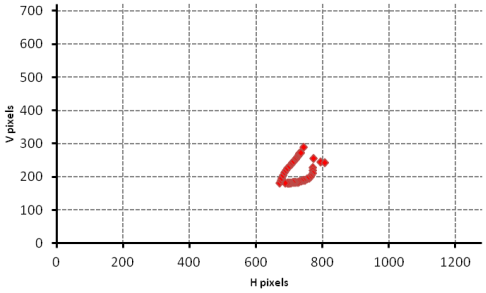

Figure 14. Trajectory of the centre of the target object – tracking

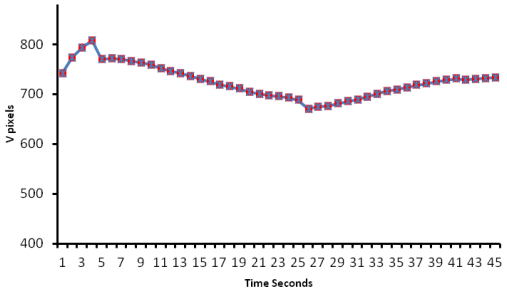

Figure 15. Trajectory of the centre (x axis) of the target object – tracking

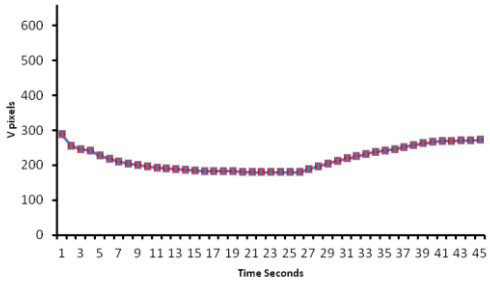

Figure 16. Trajectory of the centre (x axis) of the target object – tracking

5. Conclusion

In this paper, design and implementation of a vision based hovering and tracking algorithm for UAVs was investigated. The main goal was to implement the above capabilities on-board using the minimal reserve capacity in deeply embedded processor of the 3DR Solo quadrotor (IMX6). The algorithm was successfully implemented. The vision controller, which played the role of image capture, calibration, target detection and estimation of target pose, was developed and verified successfully. Using the pose information generated by the vision controller, the quadrotor did its task, hovered and tracked, and steered by the flight controller. Limitations existed mainly in the low performance of on-board system, especially for image processing with OpenCV where significant delays compromised several core functions such as real time tracking.

Acknowledgment

This work was supported by Tyche, the Defence Science and Technology Group's Trusted Autonomy initiative.

References

[1] Caballero F, Merino L, Ferruz J and Ollero A 2008 *Journal of Intelligent and Robotic Systems* **54** 137-61
[2] Courbon J, Mezouar Y, Guenard N and Martinet P 2010 *Control Engineering Practice* **18** 789-99
[3] Karlsson N, Bernado E D, Ostrowski J, Goncalves L, Pirjanian P and Munic M E 2005 *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*
[4] Achtelik M W, Weiss S and Siegwart 2011 *IEEE International Conference on Robotics and Automation (ICRA)*
[5] Nutzi G, Weiss S, Scaramuzza D and Siegwart R 2011 *Journal of Intelligent and Robotic Systems* **61** 287-99
[6] Bohdanov D 2012 *Quadrotor UAV control for vision-based moving target tracking task* Master Thesis University of Toronto
[7] Gomez-Balderas J E, Flores G, Garcia Carrillo L R and Lozano R 2012 *Journal of Intelligent and Robotic Systems* **70** 65-78
[8] Kendall A G, Salvapantula N N and Stol K A 2014 *International Conference on Unmanned Aircraft Systems*
[9] Bay H, Ess A, Tuytelaars T and Gool L V 2008 *Computer Vision and Image Understanding* **110** 346-59
[10] Wikipedia *Image moment* [Accessed online: 21 March 2016]
[11] Bay H, Ess A, Tuytelaars T and Gool LV 2006 *9th European Conference on Computer Vision*
[12] Bohdanov D 2012 *Quadrotor UAV control for vision-based moving target tracking task* Master Thesis University of Toronto
[13] Welch G and Bishop G 2006 *An Introduction to the Kalman Filter* University of North Carolina at Chapel Hill
[14] I. Corporation 2001 *Open Source Computer Vision Library - Reference Manual*
[15] D. Robotics 2015 *About Dronekit*