An IoT-Enabled Modular Quadrotor Architecture for Real-Time Aerial Object Tracking

Gavin Coelho
Peterbilt Motors Company
Denton, Texas, USA.
Email: Gavin.Coelho@paccar.com

Elias Kougianos
Engineering Technology
University of North Texas, USA.
Email: elias.kougianos@unt.edu

Saraju P. Mohanty
Computer Science and Engineering
University of North Texas, USA.
Email: saraju.mohanty@unt.edu

Prabha Sundaravadivel
Computer Science and Engineering
University of North Texas, USA.
Email: prabhasundaravadivel@my.unt.edu

Umar Albalawi
Computer Science and Engineering
University of North Texas, USA.
Email: UmarAlbalawi@my.unt.edu

***Abstract*—This paper presents a modular and extensible quadrotor architecture and its specific prototyping for automatic tracking applications. The architecture is extensible and based on off-the-shelf components for easy system integration while maintaining constant connectivity with the IoT. A target tracking and acquisition application is presented in detail to demonstrate the power and flexibility of the proposed design. Complete design details of the platform are also presented. The designed module implements basic PID control and a custom target acquisition algorithm. Details of the sliding-window based algorithm are also presented. This algorithm performs 20× faster than comparable approaches in OpenCV with equal accuracy. Additional modules can be integrated for more complex applications, such as search-and-rescue, automatic object tracking, and traffic congestion analysis.**

***Index Terms*—Object Detection, Robots, Robot Vision Systems, Global Positioning System, Internet of Things**

I. Introduction and Contributions

The ubiquitous quadrotor is commonly used with an on-board camera and one/two operators who control both the flight of the vehicle and the camera operation. In industry this provides a quick, easy and relatively low-cost means for inspection of pipelines, bridges and large structures and navigating areas that are remote and otherwise hard to access. Civil applications include search and rescue, traffic congestion analysis, fire monitoring, HAZMAT operations and the inspection of dangerous sites as well as environmental assessments and nature conservation. In law enforcement they are useful for surveillance, documenting crime scenes and gathering intelligence. They can also be used for aerial photography, television and videography, real estate and property assessment. By enabling autonomous control with object recognition and video tracking many of these tasks can be automated allowing for more vehicles to be deployed with considerably fewer operators. For example, during a search and rescue mission, multiple quadrotors could be programmed to search a given area sending an alert to the search team when a possible subject is found. If they are equipped with an infrared thermal imaging camera this would allow them to search through the night. Similarly, for law enforcement and surveillance a subject could be tracked by multiple quadrotors, all communicating with the base station or each other, forming a subnet of the Internet-of-Things (IoT), as shown in Fig. 1. The control algorithm can then analyze information from not just one vehicle but the whole swarm and the IoT itself allowing it to make more complex calculated decisions.

Fig. 1. The IoT-enabled aerial platform.

The **novel contributions** of this paper are the following [1]:

- A functional low-priced quadrotor was built based on modification of existing proprietary and open-source platforms.
- A medium resolution (640 × 480) optical camera system was designed and attached to the quadrotor
- A ground control station was designed and built. It provides for autonomous wireless control of the quadrotor without affecting the vehicle's payload. PID control was also implemented on-board.
- Wireless video transmission was achieved between the quadrotor and the ground station using commonly available off-the-shelf components.
- The OpenCV computer vision software platform was modified to accomplish all video related tasks, including pattern recognition.
- A library of serial communication functions was custom developed for this project to allow the control of the quadrotor from the ground station if autonomous flight fails.
- An existing algorithm presented in the literature was

modified to operate satisfactorily on the limited hardware of the ground station. An average speedup of 20 × was achieved.

The rest of this paper is organized as follows: in Section II we briefly discuss the relevant flight dynamics and control systems of quadrotors in general. Section III presents the hardware considerations taken into account for this design. Section IV describes the hardware setup and communication protocol followed in this design. Section V presents the object detection and video processing algorithms used. Section VI concludes the paper and briefly discusses areas of future development of this platform.

II. Quadrotor Sytem-Level Description

In this Section a description of the quadrotor system is presented with brief discussion of each component.

A. UAV

The unmanned aerial vehicle (UAV) is a powered, aerial vehicle that is either remotely piloted or controlled autonomously. A remotely piloted UAV is known as a drone. Drones have been around since the early 20th century. However, in recent years many unmanned UAVs have been developed with autonomous control allowing them to carry out preprogrammed flight plans. They are most commonly used for military applications including reconnaissance and attack missions [2]. Furthermore, they are increasingly being used for other non-military tasks including fire fighting, surveillance, and inspection.

B. Control System

The quadrotor is a dynamically unstable nonlinear system. This makes the quadrotor difficult to fly without an embedded control system [3], [4]. This is highly essential for autonomous flights. A close-loop feedback control system is implemented in the quadrotor to achieve stable flight. In the case of a quadrotor, the process is the vehicles dynamics and the output $Y(s)$ is the current position and orientation. The various sensors measure this information in real-time and compare it to the desired value. The difference of the desired input value and actual output is the tracking error ε. The controller processes this error and the output is the new speed for each individual motor. The change in motor speed results in a change in the system output, which in turn is compared to the desired input and the cycle continues to repeat this process. Many studies have been conducted to test the response of various feedback controllers. These include Dynamic Contraction Method [5] and Linear-Quadratic Regulator (LQR) [6].

III. Hardware Components of Quadrotor Architecture

Many different quadrotor platforms have been developed for both research and commercial applications. Most tend to be relatively costly ranging from $,2000 - $,7000. The ArduCopter [7] (succeeded by the APMCopter [8]) is an open source quadrotor UAV project with a large active community and low cost components. The controller is based on the popular Arduino platform and is undergoing constant development. This makes it a good platform for further development. The total cost of the hardware was $1,276 which is very reasonable for its intended applications. The hardware will be discussed in three different sections: the ArduCopter (which consists of the frame, the drive system, the controller or autopilot and the sensors), the radio controller and the ground control station (that includes the wireless telemetry, video capture system and computer).

A. ArduCopter

Frame. The frame was purchased as a kit that contained the main plates, arms, motor mounts, battery mount, the carrier boards and the landing gear (Fig. 2).

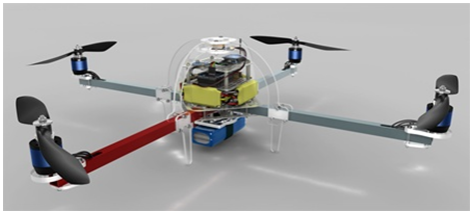

Fig. 2. Assembled quadrotor.

Drive system. The drive system consists of four electronic speed controllers (ESC) that drive the brushless DC motors. The system is powered by a battery pack via a power distribution board and the ESCs receive a Pulse-Width Modulated (PWM) control signal from the ArduPilot Mega (APM) controller, as shown in Fig. 3. The APM consists of an Arduino Mega microcontroller and associated firmware.

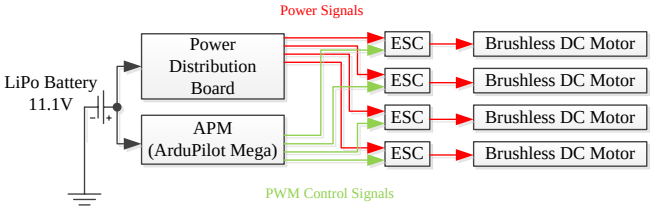

Fig. 3. Drive system block diagram.

Brushless DC motors. Brushless DC motors are used over brushed motors as they have a higher efficiency, experience less wear, produce less noise, allow for more accurate speed control and offer a better thrust to weight ratio. However they are more expensive and require more complex and costly control electronics. The current and torque relationship is linear as well as the frequency and speed relationship. Brushless DC motors are rated by their K_m (motor constant) and K_v (motor velocity constant) values. The K_v rating of a motor is the ratio of the unloaded Rotations Per Minute (RPM) to peak voltage. Depending on the motor configuration they are normally referred to as either "outrunners" or "inrunners". The conventional configuration is the "inrunner" and consists

of three stator windings surrounding the rotor which contains the permanent magnets. An "outrunner" consists of the stator coils in the center with the permanent magnets attached to the rotor which rotates around the outside. The motors used have a K_v value of 850. Thus with an 11.1 V supply they are capable of reaching a maximum of 9435 RPM. They weigh 62 g each and can produce a maximum torque of 1095 g.

Electronic speed controller. The motors are driven by a programmable electronic speed controller (ESC) which receives a 50 Hz PWM control signal from the APM and switches a network of field effect transistors (FETs). The position of the motor is determined by measuring the back EMF, which allows for the controller to energize the correct coil causing the motor to rotate.

LiPo batteries. Two 11.1V lithium polymer (LiPo) 3-cell batteries were purchased, each with a capacity of 3000 mAh and weight of 269 g. This allows for a flight time of approximately 10-20 minutes.

Rotors. A set of four 10-inch rotors consisting of two pushers and two pullers, was used. The pushers are used on the clockwise rotating motors (N and S) while the pullers are used on the counter-clockwise rotating motors (W and E). It is important to ensure that the rotors are balanced so as not to introduce vibrations into the system, as this will induce errors in the sensor readings.

Controller (APM). The controller is responsible for the stabilization of the vehicle by continually processing the data from the sensors and adjusting the speed of the rotors accordingly. The ArduPilot Mega (APM) is a controller board based on a 16 MHz ATMega1280 microcontroller and is responsible for the stabilization and navigation. A PID feedback control loop is implemented in the controller in order to stabilize the vehicle.

Sensors. Various sensors are used to determine the current position, orientation and velocity of the vehicle. The majority of these sensors are located on the Inertial Measurement Unit (IMU), while others are mounted on the frame and connected to either the IMU or the APM. A separate board (ArduPilot Mega IMU Shield) is required to interface the sensors with the main controller. This board attaches to the APM and has an onboard *gyroscope* and *three-axis accelerometer*. It also allows for a *magnetometer* and various other sensors to be connected. A *GPS module* attaches directly to the controller and communicates via a USB/UART interface. It is a MediaTek MT3329 and allows for positioning of the vehicle within 3 m of the desired location. In order to hold the same position it is important to continuously know the direction the vehicle is facing, as the controller needs to continuously calculate the corrections that need to be made to account for the drift in the yaw gyroscope. GPS can only calculate a directional vector when the vehicle is in motion thus a *magnetometer* is required to determine the direction whilst hovering in a single position. The magnetometer used is based on Honeywell's HMC5843 and communicates through an I_2C interface. It was soldered onto the IMU, leaving the I_2C port free to be used for other peripherals if required.

Video camera and transmitter. The onboard video system consists of a ½ inch CCD NTSC Sony camera with a resolution of 510 × 492 and a 2.4 GHz four-channel video transmitter used to transmit the video back to the ground station. The transmitter has two connectors, a 2-pin power input connector that was soldered to the power distribution board, and a 5-pin video in connector.

B. Radio Controller

It is important to have a manual control that can override the control signals sent by the computer at all times, hence a standard 6 channel RF Radio Control (RC) unit and receiver is used. For each channel the RC transmits a PWM signal that is decoded by the ArduCopter. The controller consists of two control sticks each with two degrees of freedom that make up the first four channels. These channels provide the basic flight controls, namely throttle, yaw, pitch and roll by sending a varying PWM signal that ranges from 1000 to 2000 microseconds. The remaining two channels are toggle switches that only have two states, low and high, represented by a PWM value of 1000 or 2000 microseconds respectively. These toggle switches can be used to set custom control modes for the ArduCopter. In this design, channel 5 is used to toggle between a manual stabilized control mode and the altitude hold mode and channel 6 is left unused. The layout of the first four channels is described by 4 different modes, with Mode 1 and Mode 2 being the most common. Mode 1 is more popular in the United Kingdom and has the throttle and yaw on the right hand side stick. Mode 2 is used in this design. It is the favored mode in the United States and has the throttle and yaw on the left hand stick. The PWM range will differ between RC controllers, thus the ArduCopter needs to be calibrated. This can either be done by setting the values manually with the command line interface or by using the Mission Planner GUI based calibration [9].

C. Ground Control Station

The ground control station (GCS, Fig. 4) handles all the video processing and consists of a laptop computer, wireless video receiver, USB video capture device and a USB XBee wireless module for telemetry. Telemetry is the transmission of the sensor data and commands between the ArduCopter and the GCS. The sensor data includes current position data from the GPS, altitude, velocity, orientation and RC PWM values. The ground station is connected to the XBee module via an XBee Explorer which allows for the serial commands to be sent and received over USB.

Wireless video receiver. The video receiver requires a 12 V power supply. Therefore, a 2.1 mm power jack was placed on the right hand side of the GCS. This allows for the receiver to be powered with a wall wart or a 12 V battery may be used.

USB capture device. A capture device is required to convert the analog video output from the wireless video receiver to a digital format that can be used by the software. These capture devices are available as either internal capture cards or external devices that are commonly connected via USB or FireWire.

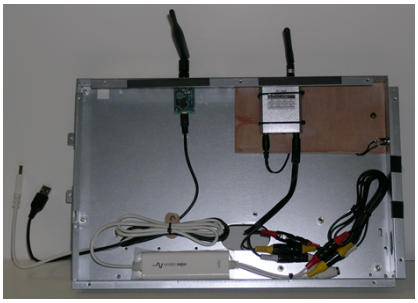

Fig. 4. Ground control station.

An Elgato 10020840 external USB 2.0 device was chosen, as it can be used with either a desktop or portable computer and USB connections are more common than FireWire. It supports a video resolution of 640 $\times$ 480 and uses either the H.264 codec at 1.4 Mb/s or the MPEG-4 codec at 2.4 Mb/s.

Xbee XBees are small, low-powered radios well suited to low bandwidth RF applications. They implement a simple serial command set making them ideal to wirelessly interface a microcontroller and personal computer. The two XBee modules considered were the XBP09 and the XBP24, which operate at 900 MHz and 2.4 GHz, respectively. The XBP09 is capable of point-to-point, peer-to-peer and point-to-multipoint networking and has a range of up to 10 km and a data rate of 156 Kb/s. The XBP24 modules are based on the IEEE 802.15.4 standard (the basis for ZigBee) and have a higher data rate of 250 Kb/s but a reduced range of 1.6 km. The XB09 module was selected due to its greater range and operating frequency of 900 MHz, which would not cause interference with the 2.4 GHz RC. In order to interface the XBee module with the APM an XtreamBee board is used in the current design.

Computer. A command line program is executed on a laptop running Mac OS 10.6.8 with OpenCV 2.3.1 installed. Two USB connections are required for the XBee and video capture device.

IV. Wireless Communication in Quadrotor

Video processing requires a substantial amount of processing power and is beyond the capability of the AVR microcontroller used to control the APM. To perform the processing onboard the quadrotor a microcomputer (BeagleBone, Raspberry Pi or similar board) capable of running OpenCV would be required. This would add weight and require significantly more power thus reducing the battery life and flight time considerably. Instead, a ground control station (GCS) is used to do all the heavy processing. This is accomplished by sending the video to the GCS via a wireless transmitter. The GCS processes the video and determines the actions to be taken, sending back control commands to the ArduCopter. As seen in Fig. 5, the video link is a one-way downlink while the control link is bidirectional, allowing for the GCS to send and receive data from the ArduCopter. It is important for these links to operate at different frequencies so as not to cause any interference. The RC operates at 2.4 GHz, thus a 5.4 GHz video transmitter was selected. The data link consists of two XBee modules.

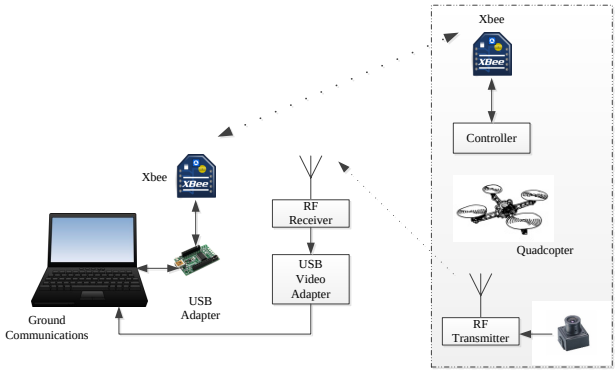

Fig. 5. Wireless communication setup.

The ArduCopter uses MAVLink [10], a two-way communication protocol based on the W-CAN and SAE AS-4 standard that was developed specifically for micro air vehicles. MAVLink consists of a header library that has implemented various commonly used messages that allow for settings to be updated, sensor readings to be read, modes to be changed and control commands to be sent. If custom messages are required they may be generated in either C or Python. A message packet varies in length and can be between 8 and 263 bytes. Every packet consists of at least 8 bytes and the message type determines the length of the payload which can be between 0 - 255 bytes. The first byte is the packet start sign (0x55) and is followed by the payload length (0-255). The third byte is the packet sequence and represents the number of messages that have been generated during the current session. This number is automatically incremented when each new message is packed. The next two bytes are the system ID and the Component ID. This allows for multiple systems to communicate with multiple vehicles as a message can be addressed to a specific one. The sixth byte is the Message ID that determines the function of the message. There are a number of commonly used messages as well as some that are specific to the ArduCopter. The payload can vary in length from 0 - 255 bytes depending on the message and it carries the information either being sent to or received from the ArduCopter. The payload can contain 8/16/32/64 bit signed or unsigned integers, floats, doubles or char values. Finally the last two bytes consist of a 16 bit checksum generated by the same means as that used in the ITU X.25 and SAE AS-4 standards.

The main message used in the program is the RC Channel Override message, which allows for the program to set the PWM values for each channel overriding the RC values for one or multiple channels. The message payload consists of 18 bytes. The first two bytes are the target system and component ID, which are both set to one. The remainder of the payload consists of eight 16 bit unsigned integers representing the PWM value for each channel. To override the channel this value should be between the minimum and maximum PWM

values for that channel, normally between 1000 and 2000. To release control of a channel back to the ArduCopter a value of 0 is set and to leave the current value for the channel unchanged a value of -1 (0xFFFF) is set. Each message has a corresponding pack function that takes the required values and generates the message. MAVLink has already been implemented on the ArduCopter and thus, only needed to be incorporated in the ground control station program. On the GCS the message is packaged with the required MAVLink pack method and then sent out over the serial port. The ArduCopter then receives the message and checks the data integrity by comparing the checksum. If there is no data lost then the message is unpacked and the data is passed to the required function determined by the message ID. A number of helper functions were created to pack and send messages out over the serial port. To arm the motors, after the ArduCopter has booted up, the left control stick is held in the bottom right position for a few seconds. This corresponds to a throttle value of 0% and a yaw right value of 100%. A function was created to arm the motors by sending a message with the corresponding PWM values for channels three and four, waiting 4 seconds and then handing control back to the RC.

V. Object Detection and Video Processing in Quadotor

Object tracking is performed using a vision system. It consists of three steps as follows: (1) the detection of the desired object, (2) the tracking of the object between frames and (3) the analysis of the changes in object position to determine the behavior of the object.

A. Template Matching

Template matching is one of the most common techniques used for object detection and has been implemented in OpenCV as the `matchTemplate function`. In order to perform the match, a template image is used to identify an object in a source image by comparing the pixel values of the target with those in a portion of the source image. A result image is created with each pixel value representing the confidence of the match at the corresponding location. There are six different comparison methods available in OpenCV [11]. The template image T with pixel width T_w and pixel height T_h is initially overlaid on the source image S with pixel width S_w and pixel height S_h, starting in the upper left corner as shown in Fig. 6.

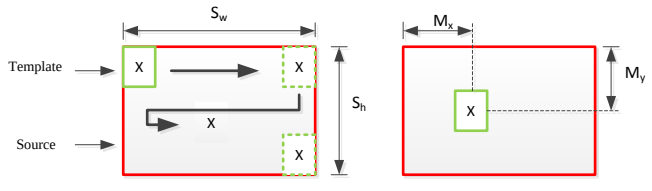

Fig. 6. Sliding window method used for template matching.

A comparison of the pixels is performed and the result R is the confidence of the match at this location. The template image is moved 1 pixel to the right ($x = x + 1$) and another comparison is performed. When the right edge of the template image reaches the right edge of source ($x = S_w - T_w$) the template image is then moved down 1 pixel ($y = y + 1$, $x = 1$) and the process is repeated until the template reaches the bottom right corner ($x = S_w T_w$, $y = S_h - T_h$). The result image will have a width of $R_w = S_w T_w$ and a height of $R_h = S_h T_h$. The location in the result image with either the highest or lowest value, depending on the method used, has the highest match probability. This point is relative to the result image and thus needs to be converted to a point relative to the source image, using the following equations:

$$M_x = R_x + S_x - \frac{T_w}{2} \quad (1)$$

$$M_y = R_y + S_y - \frac{T_h}{2}, \quad (2)$$

where (M_x, M_y) are the coordinate locations of the match with respect to the source and (R_x, R_y) are the coordinate locations of the match with respect to the result image. One of the disadvantages of this method is that it can be slow if the source image is relatively large or the template image is much smaller than the source. In this case the source image from the video feed is 640 $\times$ 480 pixels. Assuming a template image of 100 $\times$ 100 pixels then 205,200 comparisons of the template image would be performed as seen below:

$$\text{No. of comparisons} = (640 - 100) \times (480 - 100) = 205,200. \quad (3)$$

Another disadvantage is that the target in the template image needs to be the same size as the target in the source image. This is a problem as the size of the target in the source image is dependent on the distance between the quadrotor and the target. There are two possible ways to overcome this, the first is to get an altitude reading from the ArduCopter and scale the target image accordingly to match the expected size of the target in the source. This would require that the size of the target is known and that an accurate measurement of the distance to the target can be obtained. The other method would be to attempt a match and if one is not found then scale the target and repeat until a match is found. However this would be significantly more computationally expensive and would result in a much lower frame rate. To perform a template match in OpenCV the template image is loaded as a gray-scale image and stored in a native data structure. The video stream is opened using the desired camera source and then a single frame is grabbed and stored as the source image. The image that is captured form the camera is an 8-bit image and has three channels, red, green and blue. The color channels are not needed, as the target image is black and white, the image is converted to gray scale thus using just one channel. This greatly reduces the size of the matrix (by a factor of almost 3) and results in a significant speed increase.

B. Fast Template Matching

One method of improving the time taken to perform the template matching is to equally scale down the source and

template images. By doing this the number of comparisons performed can be reduced greatly. For example, if a source image of 640 $\times$ 480 pixels and a template image of 100 $\times$ 100 pixels are scaled downby a factor of 4 then the number of comparisons performed will be reduced from 205,200 to 12,825 This results in a significantly large computational saving as now 93.25% fewer comparisons are performed, however the accuracy of the location of the match will be less accurate. The confidence of the match will also be affected, as information is lost when the image is down-sampled. An OpenCV function [11] uses this method to down-sample the source and template images and find multiple match locations. The original source image is then searched around these match locations by creating a Region of Interest (RoI) that is centered on the match location with a size that is slightly larger than the template image. This allows for the match location to be determined without a great cost in computation or resolution. The final algorithm used to perform the template matching was based on the `FastMatchTemplate` with image pyramid [12]. The original function was updated to use the new C++ data structures, which do not require memory management. All unneeded code was removed, as only one match location was required, thus simplifying the function. For this application the goal is to detect an object while flying above it and at high altitudes; it is therefore likely that the target is only going to move by a small number of pixels between each pass through the loop. An option was thus added to skip the down sampling step if there was a match previously and instead just search around the location of the previous match. This can produce a small improvement in the time taken to find a match for a slow moving target. This method will also filter out any large changes in the match location that could be due to false positives or other targets being detected and will prevent the quadrotor from making sudden changes or losing track of the current object. The first thing the algorithm does is to check that the template image is smaller than the source image and that the number of channels in each image is equal, otherwise the function will fail. Next, copies of the source and template are made so as to not alter the originals. In the case when there was not a match the last time through the loop, then down sample the images and try to find a match with confidence above the desired value. If a match is found then the match location is set to then new value. When there was a previous match or there is a new match, then the original source image is searched over a small user defined RoI. If a valid match is found in this region then the match location is converted back to the source image coordinates. Finally, the target location may be highlighted on the source image and displayed for the user. This custom developed fast template matching algorithm achieved an average speedup of 20$\times$ over the unoptimized `FastMatchTemplate`.

VI. Conclusions and Future Research

A versatile and extensible quadrotor platform, shown in Fig. 2, based on open-source hardware and software was designed and is described in detail in this paper. As an example application of the platform's capabilities, a target recognition system was designed, programmed and implemented using custom and published algorithms with outstanding performance. Research work is currently underway to extend the functionality of the vehicle by incorporating line-of-sight optimal communications location for search and rescue operations. Further work will concentrate on improving the on-board computating capabilities so that most of the computational burden is removed from the base station thus reducing the large amount of wireless traffic currently incurred. To securely transfer image and video data, the quodrotor architecture will be equipped with an on-board secure digital camera (SDC) [13]. In such a situation, the quodrotor can be deployed to capture and transmit sensitive information reliably through the use of its on-board SDC through use of watermarking and encryption [13], [14]. This is particularly important in certain applications in which a 3rd party can tamper with the data transmitted from the quodrotor in an IoT framework.

References

[1] G. Coelho, "OTA-Quadrotor: An Object-Tracking Quadrotor for Real-Time Detection and Recognition," Master's thesis, Department of Engineering Technology, University of North Texas, Denton, TX, USA.

[2] M. Tarhan and E. Altuğ, "A Catadioptric and Pan-Tilt-Zoom Camera Pair Object Tracking System for UAVs," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1–4, pp. 119–134, January 2011.

[3] R. C. Leishman, J. C. MacDonald, R. W. Beard, and T. W. McLain, "Quadrotor and Accelerometers: State Estimation with an Improved Dynamic Model," *IEEE Control Systems*, vol. 34, no. 1, pp. 28–41, January 2014.

[4] R. Mahony, V. Kumar, and P. Corke, "Multirotor Aerial Vehicles: ModModel, Estimation, and Control of Quadrotor," *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[5] R. Czyba, "Design of Attitude Control System for an UAV-type Quadrotor Based on Dynamic Contraction Method," in *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2009, pp. 644–649.

[6] B. Yu, Y. Zhang, I. Minchala, and Y. Qu, "Fault-tolerant Control With Linear Quadratic and Model Predictive Control Techniques Against Actuator Faults in a Quadrotor UAV," in *Proceedings of the Conference on Control and Fault-Tolerant Systems (SysTol)*, 2013, pp. 661–666.

[7] "Arducopter - The Full-Featured Multicopter UAV!" http://code.google.com/p/arducopter/, Last accessed on 27 Sep 2015.

[8] "APMCopter," http://copter.ardupilot.com/, Last accessed on 27 Sep 2015.

[9] "APM Mission Planner," http://code.google.com/p/ardupilot-mega/wiki/MissionPlanner, Last accessed on 27 Sep 2015.

[10] QGROUNDCONTROL, "MAVLink Air Vehicle Communication Protocol," http://qgroundcontrol.org/mavlink/start, Last accessed on 27 Sep 2015.

[11] "The OpenCV Reference Manual," http://docs.opencv.org/opencv2refman.pdf, Last accessed on 27 Sep 2015.

[12] OpenCV, "Fast Template Matching with Image Pyramid," https://opencv-code.com/tutorials/fast-template-matching-with-image-pyramid/, Last accessed on 27 Sep 2015.

[13] S. P. Mohanty, "A Secure Digital Camera Architecture for Integrated Real-Time Digital Rights Management," *Journal of Systems Architecture*, vol. 55, no. 10-12, pp. 468–480, October 2009.

[14] N. M. Kosaraju, M. Varanasi, and S. P. Mohanty, "A High-Performance VLSI Architecture For Advanced Encryption Standard (AES) Algorithm," in *Proceedings of the 19th International Conference on VLSI Design*, 2006, pp. 481–484.