Networking-Enabling Enhancement for a Swarm of COTS Drones

Sondre Engebråten[12], Kyrre Glette [12] and Oleg Yakimenko [3]

***Abstract*— This paper presents the augmentation of a commercial-of-the-self (COTS) multi-rotor unmanned aerial system (UAS) to extend its capabilities by introducing increased on-board processing power. The option to process sensor information on-board reduces the required bandwidth by only communicating relevant or important information and therefore increases the level of UAS autonomy. It also enables adaptive reactive abilities for a group of UAS executing a coordinated, collaborative or cooperative mission. This paper describes an effort to integrate a small board computer, more specifically, an Odroid C2 with the COTS 3DR Solo drone, and reviews the achieved enhanced capabilities of such a system. This is an extension of a traditional autopilot, as such, the original Ardupilot hardware and software is not modified in this work.**

I. Introduction

These days, quite a few companies offer a variety of commercial-of-the-shelf (COTS) multi-rotor unmanned aerial vehicles (UASs). Figure 1 shows examples of some COTS drones widely used in university research.

These drones are designed to be operated using standard radio-frequency (RF) remote controllers or mobile devices and typically use either a manual mode (with some automation during take-off and landing operations) or standard waypoint navigation. Usually, these COTS drones (like the one shown in Fig. 1a) utilize proprietary autopilots precluding from developing and testing advanced guidance, navigation and control (GNC) algorithms, but some drones allow communication with or even modification of controller code (like the one shown in Fig. 1c).

A typical autopilot is integrated with a set of corresponding sensors (global positioning system receiver, 3-axis accelerometer, 3-axis gyroscope, magnetometer and barometric altimeter) and just enough computational power to carry out typical GNC tasks, i.e. estimating drones position, sequencing the way points or establishing a certain pattern, and finally, following this pattern using one or another modification of proportional-integral-derivative (PID) controller.

For those drones allowing code modification, a massive body of literature exists on exploring and field-testing the new approaches within all three components of GNC triad. Clearly, more advanced algorithms require more computational power on-board (to incorporate on-line image processing, run deep-learning algorithms, provide networking with other agents, etc.) [1], [2]. This means integrating COTS drone with a companion computer. For example; the Drone shown in Fig.1b, allows adding a high-performance embedded computer to enable developers to transform this Drone into a more intelligent flying robot that can perform complex computing tasks and advanced image processing [3]. Previous work suggests that by adding additional processing power, GNC tasks may be freely moved from lower level autopilot to higher level controller and vise versa [4].

Several previous works take on the challenge on integrating against the Ardupilot autopilot in an effort to extend the capabilities of the platform in some way [5]–[7]. However, these are based on a custom built platforms, which takes a lot of work to provision and this can now be avoided by using ready to fly COTS drones [8]. Finally, works some works show the integration of an Odroid with a Ardupilot autopilot in a single vehicle context [9].

This paper describes an effort of developing a networking-swarm-capable UAS, based on a popular 3DR Solo platform (Fig.1c), which would enable field-testing of drone swarm behavior and assessing their operational capabilities [10]. The key issue here is that the overall cost of the complete platform, including a companion single-board computer, Odroid C2 [11] is an order of magnitude cheaper of its more sophisticated family members like the ones shown in Figs. 1a,b. The paper presents a detailed description of the developed system and is organized as follows. Section 2 discusses requirements to the system from the standpoint of networking capability followed by Section 3 justifying the choice of COTS 3DR Solo drone [8] and detailing a procedure of 3DR Solo and COTS Odroid board integration. Section 4 describes novel capabilities achieved for the fleet of enhanced UASs. Section 5 ends the paper with some concluding remarks.

II. Networking and On-Board Computing Requirements for Swarm Operations

A swarm can be viewed as a distributed sensor network, or a bulk data sensor. Each agent may be equipped with a number of sensors, while individually simple, the collective data they produce may form a complete real-time view of a larger area. When implementing a real-world swarm, however, one of the primary issues with COTS hardware today is networking, or more specifically, the lack of a light, low-powered, high-bandwidth, fully-decentralized network infrastructure. Without this, the swarm has to compromise on the amount of data shared or the structure of the swarm itself.

While each sensor only requires a small amount of bandwidth, the aggregated stream of data may quickly overwhelm even high bandwidth commercial networks such as 5Ghz WiFi with a top speed of 1.3 Gbps. In order to achieve such speeds these networks employ beamforming, a technique

[1]Norwegian Defence Research Establishment, P.O. Box 25, 2027 Kjeller, Norway Sondre.Engebraten@ffi.no

[2]University of Oslo, P.O. Box 1080, Blindern, 0316 Oslo, Norway

[3]Naval Postgraduate School, 699 Dyer Rd., Monterey, CA 93943, USA

(a) DJI Phantom 4 (b) DJI Matrice 210 (c) 3DR Solo

Fig. 1. Example of drones used in academic research [3], [8].

to adapt the radiation or listening patterns of antenna to tune to certain directions. While this is great for achieving higher bandwidth with limited spectrum, this complicates communication for a swarm. In a swarm context, a lot of the communication can benefit from the ability to broadcast information, which makes this directional technique counterproductive. There is also a physical limit on how many links or client nodes such a network can support. This scales poorly with a swarm with even a low number of agents as the number of links scale quadratic. For instance, in a swarm with n=5 agents, if all agents are to be able to communicate with every other agent, this would require a total of 10 links (0.5n(n-1)). In the case of 10 agents the number of individual links increases to 45!

Requiring omni-directional communication, limits the swarm to the lower-speed network technologies with a maximum bandwidth of 433Mbit. Of course, this bandwidth is computed for only 2 communicating nodes. Adding just one additional node (scaling the number of nodes up to 3), introduces collisions and on-air conflicts. Dividing the available bandwidth onto 3 (one-to-one) links yields a theoretical maximum value of 144.3Mbit (under ideal conditions).

For larger networks, such as used by business users, the issue of scaling within traditional wireless infrastructure may be solved by increasing the number of base stations or access points, reducing the emitted power and effectively spatially sharing the spectrum by limiting the range of the individual client nodes. This approach could be feasible for a swarm as well, but an ideal swarm implementation would leverage complete decentralization in order to attain robustness and scalability. This includes moving computation, or sensor information processing, closer to the sensor.

Moving the sensor processing close to the sensor, is similar to the use of cache in a computer. The lower levels of cache have significantly greater bandwidth compared to the higher level of storage, i.e. by moving processing closer to the sensor there is less or even no restriction on the bandwidth the sensor can have. This may be one way to make a real swarm systems feasible, lifting the bandwidth restriction by doing most of the heavy processing close to the sensor itself, only a very limited subset of the data needs to be forwarded for additional review. This allows a highly optimized swarm to process large amounts of sensor data without infeasible network requirements. Moving processing

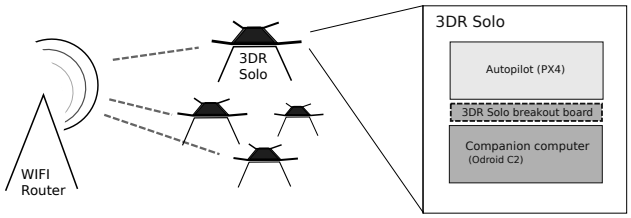


Fig. 2. Decentralized swarm architecture.

closer to the sensor also helps in reducing latency, which is of major importance for the agility of swarms. For example; collision avoidance for multiple UASs, operating in close-formations, will require split-second decisions.

The overall view of the system, which the authors are trying to develop, is presented in Fig. 2. This figure illustrates multiple drones connected to a common WiFi network. On-board each drone, the network is supported by a powerful companion computer that needs to be integrated with the existing autopilot.

The initial experiments with a COTS-based UAS swarm, assume a minimal volume of data to be shared. This include drones identification number and three-dimensional position. This enables optimizing swarm performance, while executing reactive collision avoidance. For robustness, the lower limit for the update rate is set to 10Hz.

III. 3DR Solo-Odroid Integration

This section starts from justification on a COTS drone that was chosen as a developmental platform followed by a step-by-step procedure for integrating 3DR Solo drone with Odroid board including hardware additions and software setup.

A. 3DR Solo Drone

To implement the desired architecture (Fig. 2), the 3DR Solo drone was chosen amongst all COTS UASs available at the market. This choice based on the fact that relatively inexpensive 3DR Solo utilizes a variation of the PX4 autopilot [12] running a fork of Ardupilot flight controller firmware [13], [14]. What distinguishes this from other COTS drones is the flexible integration options through the Dronekit framework [15] and through the accessory port on the Drone itself [16]. The accessory port is a custom Japan

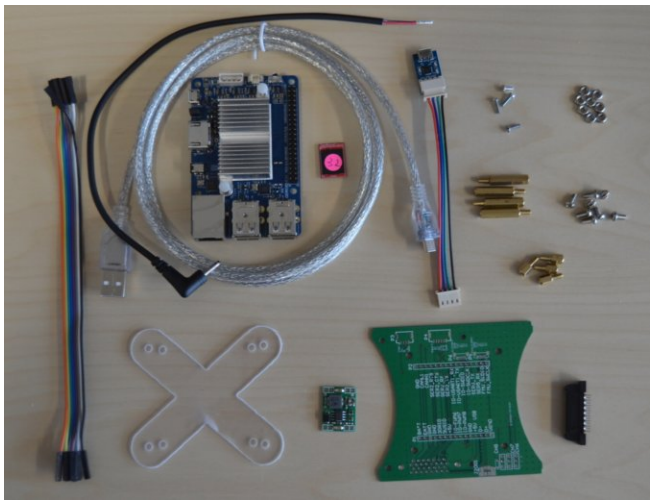


Fig. 3. Overview of the required components for integration.

Aviation Electronics (JAE) connector, for which, 3DR has created a reference board design.

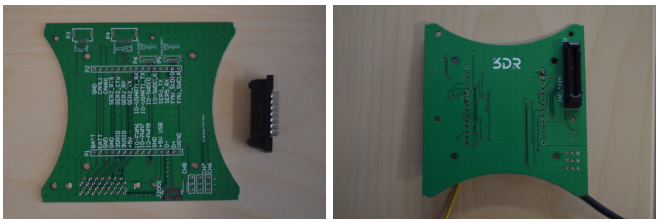
The PX4-Ardupilot use the Micro Air Vehicle Communication Protocol (MAVLink) [17]. The MAVLink protocol is an open-source telemetry protocol commonly used with UASs as well as ground-based vehicles, which run Ardupilot software. While being open-source, the MAVLink protocol is in itself a fairly extensive interface, which is not trivial to implement. The MAVLink protocol is available through a universal asynchronous receiver/transmitter (UART) or serial link from the autopilot. To be able to leverage the MAVROS software stack, the proposed controllers are implemented in Python using the Robot Operating System (ROS) framework. (MAVROS is a client library which interacts with the autopilot through MAVLink allowing a command to be sent to the autopilot and telemetry to be received through standardized ROS-Python interfaces [18].)

B. Breakout Board Build-up

The required components for integrating 3DR Solo drone and Odroid computer are shown in Fig. 3 and include

- Odroid C1+/C2 single-board computer ($46)
- eMMC storage for Odroid board (from $16)
- UART to USB adapter (3.3V compatible)
- Power cable for Odroid board
- Breakout printed circuit board (PCB)
- DC/DC power supply
- JAE connector

The adapter board is produced based on a reference design provided by 3DR. The design is open-source [19], and can be readily manufactured by common PCB manufacturers. In order to have one of these custom interface board made, all that is required is to send the PCB design output files, to a PCB manufacturer. The board seen in Figs.3-6 was manufactured by the Seeedstudio Fusion service [19]. Originally, the reference design calls for a number of connectors for this board, however, in this particular case only one of them, the JAE connector, is required (shown in lower-right corner of



(a) (b)

Fig. 4. Breakout board and JAE connector (a); JAE connector soldered to the board (b).

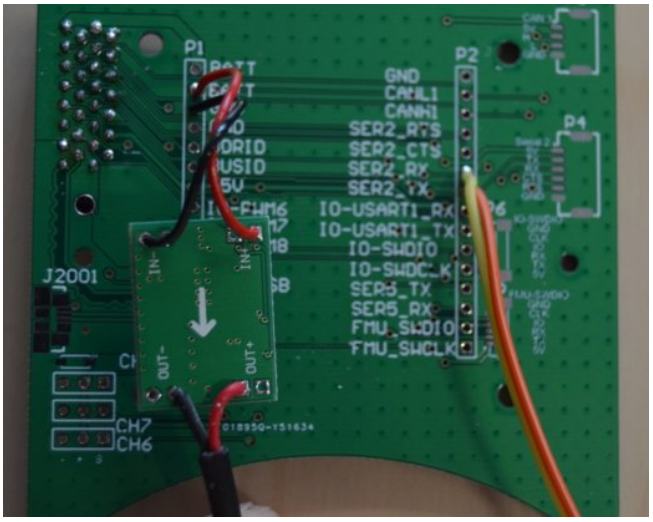


Fig. 5. Breakout board with the DC/DC power supply and control signal.

Fig. 3). When soldering the JAE contact on the board (Fig. 4a) the contact should be placed on the board side that has no annotated connections (Fig. 4b).

Odroid C2 requires a stable 5V power supply and can draw up to 2A. However, any small DC/DC converter (supplying 5V) should work (red is positive/batt, black is negative/ground/gnd). Figure 5 shows the Odroid board with the wired power (on the left) and control signal (on the right).

C. Software Setup

The on-board companion computer runs Ubuntu 16.04 LTS (Long Term Support) [20]. Initially, the firmware that comes with the Odroid C2 eMMC is unexpanded. This means that not all the flash memory is available to use. To amend this, the root file system needs to be expanded to the size of the eMMC. This is done by first connecting the Odroid C2 to the keyboard and screen. To complete the installation and expand the file system, a script is provided. After booting and opening a console window, one should type

```
/usr/local/bin/root-utility.sh
```

Followed by selecting Option 4 Resize partition. Next, the program should be terminated, and then the Odroid can be rebooted again by typing:

```
sudo reboot
```

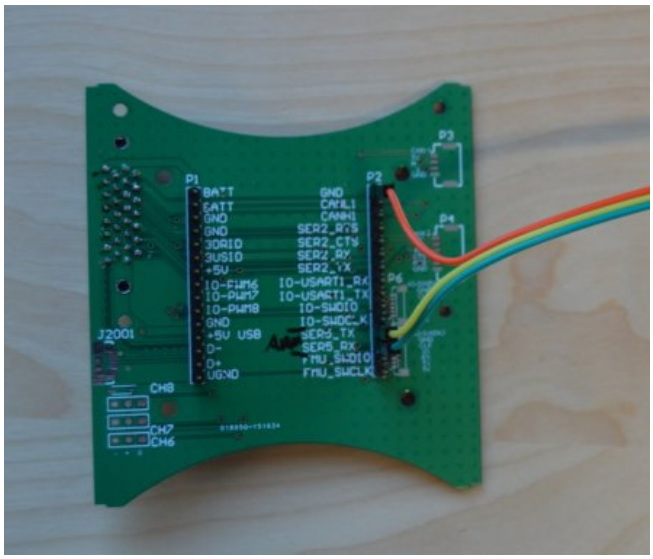


Fig. 6. Breakout board with header pins soldered.

Next, ROS needs to be installed to enable interfacing with the autopilot. The can be accomplished by following an official ROS ARM installation guide [21]. In addition to the standard ROS installation, the MAVROS package is required. This is installed by typing:

```
sudo apt-get install ros-kinetic-mav*
```

Once the Odroid board is prepared, the 3DR Solo Drone needs to be configured. This step is required in order to enable communication with the existing on-board autopilot. The factory default firmware on 3DR Solo disables serial port number 2. Enabling serial port number 2 is a two-step process.

On the breakout board, there are two serial ports: SER2 and SER5. SER2 gives access to communicate with the autopilot controller to receive telemetry and send commands. SER5 gives access to a shell on the autopilot where it is possible to enable SER2. To connect to this port a separate adapter board needs to be soldered. This differs from the first (Fig. 5) in that it has header pins on all the connectors allowing easy access to SER5 for enabling the telemetry serial port SER2. This breakout board can be seen in Fig. 6. To enable SER2, connect SER5_TX, SER5_RX and ground to a UART adapter. Using picocom on Linux, connect to the Solo autopilot in a terminal by typing

```
picocom -b 57600 /dev/ttyUSB0
```

This gives a minimal shell on the autopilot. SER2 can then be enabled by issuing the following set of commands:

```
nsh> cd /fs/microsd/APM
nsh> ls
/fs/microsd/APM:
 TERRAIN/
 LOGS/
nsh> echo >> uartD.en
nsh> ls
/fs/microsd/APM:
 TERRAIN/
 LOGS/
 uartD.en
```

This completes the first step of enabling the SER2 port on the autopilot microcontroller. The second step in enabling SER2 involves configuring the autopilot to send telemetry and receive over this port (SER2). This has to be done over WiFi since access to the MAVLink protocol is required, and SER2 is not yet enabled. The laptop needs to be connected to the Solo WiFi hotspot (the default password is *sololink*). To continue, ROS also needs to be installed on the laptop being used. To connect to the Drone and start MAVROS, the following command sequence should be used:

```
source /opt/ros/kinetic/setup.bash
roslaunch mavros apm2.launch \
fcu_url:=udp://0.0.0.0@10.1.1.10:14560 \
fcu_version:="v1.0"
```

These commands connect to the autopilot on the 3DR Solo Drone through WiFi using the MAVLink protocol. With the laptop connected, autopilot configuration parameters can be set to allow for connections to the SER2 hardware port. Parameters are set using the *rosrun mavros mavparam set* {*parameter*} {*value*} syntax, where {*parameter*} is the parameter name and {*value*} is its desired value. To complete enabling telemetry over the SER2 port the {*parameter*}-{*value*} pairs to be configured are

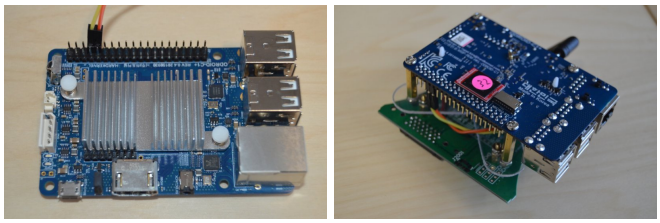
```
SERIAL2_BAUD 57
SERIAL2_PROTOCOL 1
BRD_SER2_RTSCTS 0
SR2_EXTRA1 10
SR2_EXTRA2 10
SR2_EXTRA3 10
SR2_EXT_STAT 10
SR2_PARAMS 10
SR2_POSITION 10
SR2_RAW_CTRL 10
SR2_RAW_SENS 10
SR2_RC_CHAN 10
```

The three first pairs simply enable the UART connection. The following pairs establish the rate of messages in Hertz. After restarting the Solo Drone, it should be ready to connect to the Odroid board.

D. Completing Companion Computer Integration

The programmed breakout board connects to the Odroid boards RX and TX pins (Fig. 7a) by pins SER2_TX and SER2_RX (RX connects to TX and vice versa). The complete (assembled and connected) payload (breakout board with Odroid C2) is shown in Fig. 7b.

Figure 8a presents an example of the final payload attached to the Solo adapter (at the drones bottom) as is, while Fig. 8b demonstrates a more elegant design that utilizes a custom case. In both designs, the WiFi adapter joining the swarm agents through the 5Ghz shared WiFi network can be seen.



(a) (b)

Fig. 7. The Odroid board (a), and complete payload (b).

(a) (b)

Fig. 8. 3DR Solo with a fully integrated payload (a); and example of on-board computer enclosure (b).

An external antenna is important to guarantee connectivity across the entire operation area. Empirical experiments indicate a range of several hundred meters, even with just this basic antenna configuration. As these drones will be operating in a restricted area, this is more than sufficient for the joint swarm network.

E. Ground Control Station

The Drone control architecture is shown in Fig. 9. This control architecture embodies the decentralized swarm concept, but moving all control algorithms to the on-board computer each drone carries. For debugging purposes during experiments, a single laptop is used that may connect to any of the drones available in the swarm. However, for the experiments in this paper the laptop was only used to launch the swarm behavior and could be omitted in another suitable method of synchronizing launch was implemented. The swarm behavior takes control of the Drone by setting the Drone into a special guided mode. In this mode the Drone will accept command from the Odroid and the Odroid only. The behavior can be overridden by sending a fly command from the manual remote, again placing the Drone back in manual control. For swarm experiments, this behavior was found to be very useful, as it allows for stopping the experiment on a moments notice should circumstances required it.

IV. System Testing and Data Sharing

This section addresses the initial testing of the developed system followed by a discussion of telemetry data access. It ends with presenting a fleet of network-capable drones and a simple illustration how networking enables collision free operations of multiple UAS.

A. Initial Testing

The initial ground test of the assembled system should be conducted with no propellers installed. Before the

(a) (b)

Fig. 9. Communication architecture (a); and WiFi router (b).

test, the launch file for MAVROS (*/opt/ros/kinetic/share/-mavros/launch/apm2.launch*) should be edited from

```
<arg name=fcu_url \
default=/dev/ttyACM0:57600 />
<arg name="fcu_protocol" \
default="v2.0" />
```

to

```
<arg name=fcu_url \
default=/dev/ttyS1:57600 />
<arg name="fcu_protocol" \
default="v1.0" />
```

With the launch file updated, MAVROS can be started with

```
roslaunch mavros apm2.launch
```

At this point, the Drone should be connected to the MAVROS software stack through the UART/serial connection. If everything is functioning properly, the console running MAVROS should report received heartbeat. If the Drone is indoors, it will likely report missing a 3D fix. If software crashes, there may be an issue with Drone configuration. Otherwise, the system is ready for the flight test.

With propellers installed and MAVROS running, the commands can now be sent to the Drone the hardware serial connection and MAVROS. For example, this sequence below should arm, launch and move the Drone 5 meters in the east direction

```
rosrun mavros mavsafety arm
sleep 1
rosrun mavros mavcmd takeoffcur 0 0 5.0
sleep 5
rosrun mavros mavsys mode -c GUIDED
rosrun mavros mavsetp local
 \--position 5 0 0 0
```

The first command will arm the Drone, i.e. start the propellers. This may take a second to take effect, as such it is advisable to wait a second before issuing further commands. Line 3 of the above sequence will make the Drone takeoff to an altitude of 5m above current position. This command will normally take a few seconds to complete, as the Drone needs to increase throttle and gain altitude before the sequence should be continued. In order to enable

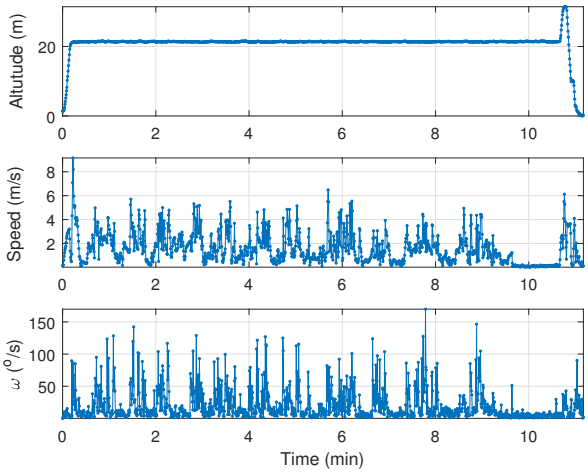


Fig. 10. Time histories of altitude, speed and angular speed.

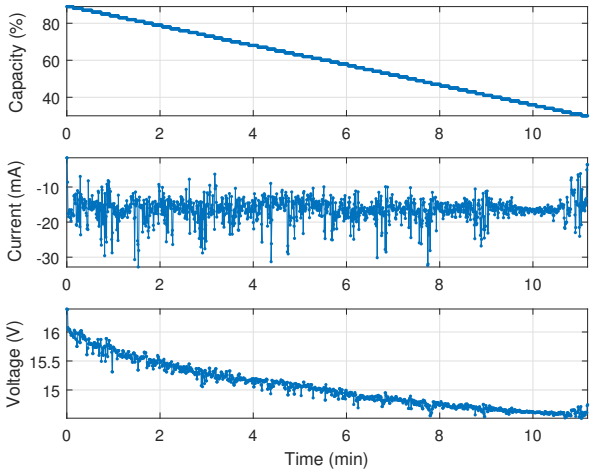


Fig. 11. Time histories of battery parameters.

programmatic control of the Drone, the Drone has to be in custom mode "GUIDED". Line 5 sets the Drone into "GUIDED" mode, this is more or less instantaneous. With the Drone in "GUIDED" mode, the Drone will no longer respond to manual remote controller input, to retake control of the Drone press the "Fly" button. This returns the Drone to GPS guided manual controlled mode. Finally, Line 6 sets a position setpoint for the autopilot in the local coordinate frame. In the example above the Drone should move 5 meters in the east direction. For more information and a more complete overview of available MAVROS functionality, refer to MAVROS documentation [18].

B. Data Sharing

The developed architecture allows developing and uploading GNC code onto Odroid through common SCP and SSH protocols. These GNC algorithms can utilize all data available, both from any sensors connected to the Odroid and internal data from the PX4 autopilot. This includes battery parameters, data coming from inertial measurement unit, global positioning system receiver, both raw data and preprocessed data. This data can be stored on-board each vehicle for later processing, but also shared amongst the swarm through the joint swarm network.. As an example, Figures 10-15 illustrate some of these parameters at 2Hz rate for an 11-minute flight. After takeoff the Drone was brought to about 20m height and primarily stayed at this altitude all the time executing a series of horizontal maneuvers. Specifically, Fig. 10 shows time histories of the height (above initial launch position), absolute ground speed and angular velocity. Figure 11 illustrates the battery state parameters during this flight. Figures 12 and 13 present time histories of acceleration and velocity vector components (Fig. 12 and Fig. 13, respectively), angular rates (Fig. 14) and computed (via parameters of quaternion) Euler angles (Fig. 15).

Odroids eMMC storage allows storing GNC-related parameters for further analysis.

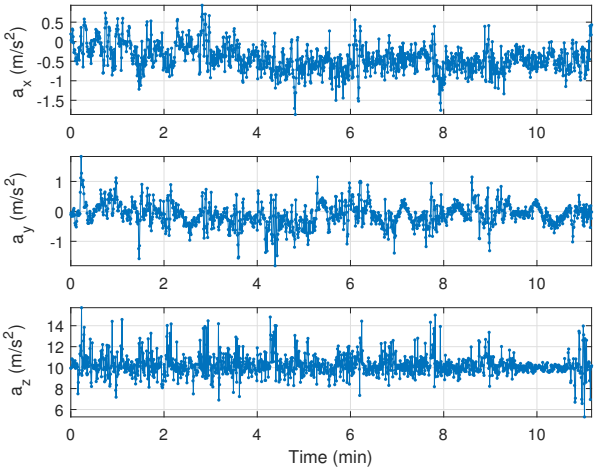


Fig. 12. Time histories of the acceleration vector.

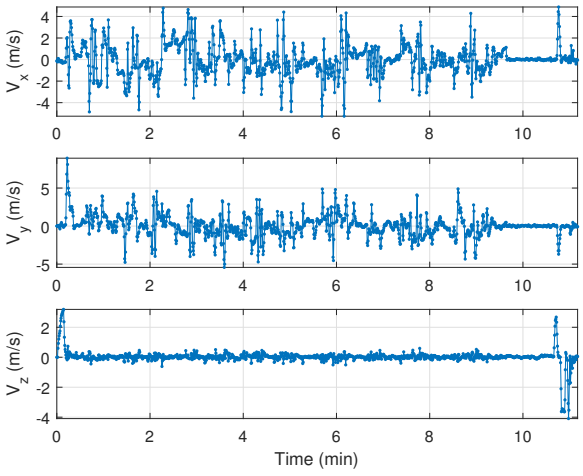


Fig. 13. Time histories of the linear velocity vector components

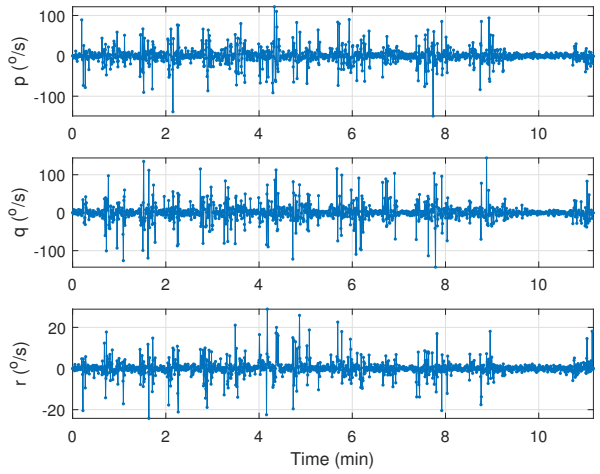


Fig. 14. Time histories of angular velocity vector components

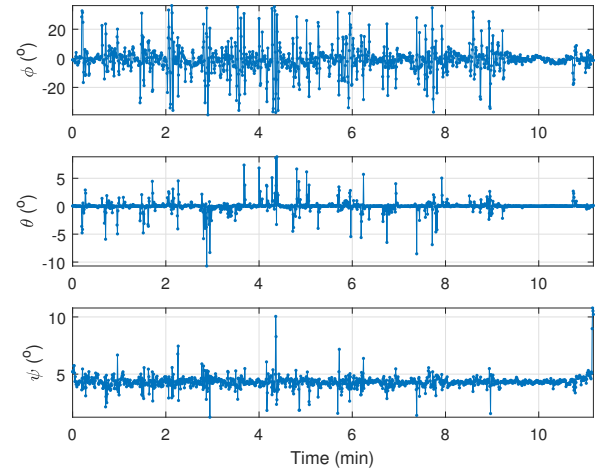


Fig. 15. Time histories of Euler angles

C. Swarm Operations Enabled UAS Fleet

Figure 16b demonstrates a fleet of 20 enhanced 3DR Solo drones ready to fly a swarm mission programmed and uploaded to all vehicles. As mentioned in Section 2, while flying such missions collision avoidance is an absolute necessity. The final testing of the developed system included testing this particular capability.

In a series of tests multiple UAS were operating in a close vicinity to each other, sharing ID and position information. The goal of these tests was to see if the developed and implemented collision-avoidance algorithms that utilize this information assure safe operations. The results of one of such tests with four UAS flying at the same altitude within horizontally bounded airspace set to collide with each other are shown in Fig. 17. This figure shows relative position of three drones with respect to another drones. Clearly, the collision-avoidance algorithms do work preventing from two UAS being closer than 15m to each other (this distance was predefined).

(a) (b)

Fig. 16. Individual network-capable 3DR Solo drone (a); and a fleet of 20 networking UAS (b)

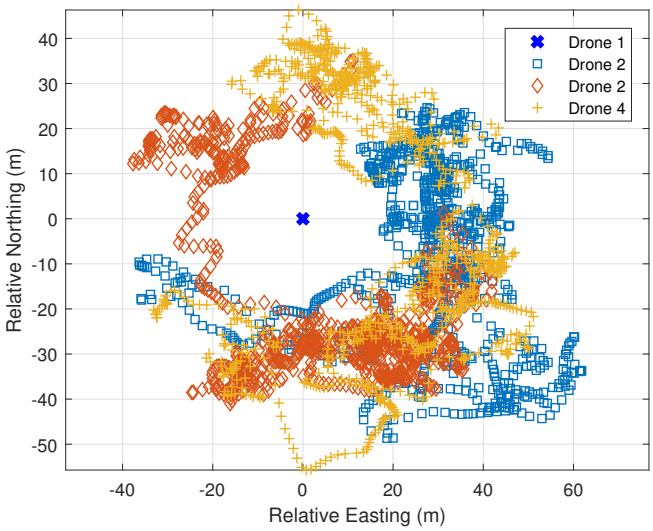


Fig. 17. Birds-eye view of relative positions of four drones operating in close formation

The 3DR Solo COTS Drone uses a standard 2.4Ghz WiFi link for manual control. To eliminate dropout when transmitting large amounts of data between the agents in the swarm, a disjoint network scheme was adopted. In this case, the individual manual control link for each agent is still 2.4Ghz WiFi, while the joint swarm network operates on 5Ghz. Initial experiment with 2.4Ghz allowed for the use of ad-hoc or mesh network, where there was no central router to mediate communication between the swarm agents. Obviously, this is the ideal situation, where each agent can communicate with any other agents in range, without relying on centralized infrastructure. Unfortunately, when switching to 5Ghz, this is no longer possible, as 5Ghz does not seem to support mesh network in the same way. Currently, this is the only limitation of the developed swarm. However, this is an issue of regulation rather than technical difficulties, as the use of the 5Ghz baseband is limited and requires transceivers to adapt a non-interference policy. Implementation of mesh network on 5Ghz might require additions to the mainstream WiFi driver in order to avoid interfering with other systems.

V. Concluding Remarks

Extending a COTS platform with on-board processing capabilities enables a variety of novel drone applications. Technically there is no requirement for these drones to be connected to a remote controller (for manual control) or

even the swarm network. Each of these drones could operate independently, allowing for autonomous missions execution. However, this would be contrary to the ideals of a swarm system. This paper presented a method of integrating an on-board companion computer with autopilot of a COTS multi-rotor drone, enabling each drone to be connected to a swarm network, and run local, distributed controller or behaviors. This is of paramount importance to the concept of swarm system. A swarm derives its core attributes of scalability, fault tolerance and distribution, through enabling each agent or platform to make independent decisions while still interacting with the rest of the swarm. Within the developed prototype swarm, it is possible to program behaviors, on-board each drone that can adapt and react to new environments or changing conditions. In light of classical control theory, this can be considered a higher-level control or an outer loop, enabling complex interactions amongst agents.

As is, the integration to the autopilot breaks gimbal support on the 3DR Solo drone. The cause for this is unknown, but it likely has to do with enabling the SER2 port which interferes with the serial port used to control the 3DR Solo gimbal. This is likely a software problem, but even though the 3DR Solo is a very open platform, some of the source code for instance for how the remote controller interacts with the Drone is not open-sourced. This makes it hard to try to pinpoint the exact issue of where the gimbal breaks. Unfortunately, 3DR has decided to discontinue the Solo as a consumer product, and as such it is not likely that there will be a fix released for this issue. In addition, newest firmware implements a block that does not allow SER2 to be used at all, as such newer drones will need to be downgraded (to firmware 2.3.0-1) in order to implement the enhancements described in this paper.

Further, the approach suggested in this paper, uses a fixed central WiFi router to enable the joint swarm network. This is a compromise in order to get a more stable network, by transitioning from 2.4Ghz to 5Ghz WiFi to avoid interference with the manual remote controller links, but the ability to do a fully decentralized mesh network is lost. It is believed that the issue of constructing a mesh network over 5Ghz mainly a regulatory problem, in that transceivers on 5Ghz are required to adopt a strict non-interference policy. As such, it is possible that in the future 5Ghz mesh WiFi could be achieved. This would be a significant step towards a fully decentralized swarm of UASs. One might argue that there are several smaller transceivers available intended for instance for Internet-of-Things systems, that might be used. However, these are currently designed around low bandwidth applications which are not suitable for the proposed UAS swarm prototype.

Finally, the challenge of managing not one or two platforms but tens of them have led to new hurdles in deployment and maintenance of the UAS swarm. As such, extending the platforms outlined in this paper with a robust software framework for deployment of new code, self-tests and communication, may be highly desired in order to ease development and allow for more rapid prototyping of new swarm systems and behaviors.

ACKNOWLEDGMENT

The authors would like to acknowledge the Norwegian Defense Research Establishment, the Naval Postgraduate Schools Naval Research Program and Consortium for Robotics and Unmanned Systems Education and Research along with Fulbright Scholar Program for the support of this research. They would also like to thank Jørgen Nordmoen for his contributions to the initial integration efforts.

REFERENCES

[1] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue," *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[2] J. N. Weaver, D. Z. Frank, E. M. Schwartz, and A. A. Arroyo, "UAV performing autonomous landing on USV utilizing the robot operating system," in *Proc. of the ASME District F-Early Career Technical Conference*, Citeseer, 2013.

[3] "DJI drones." http://www.dji.com/products. Accessed: 11.2.2018.

[4] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240, IEEE, 2015.

[5] P. Bupe, R. Haddad, and F. Rios-Gutierrez, "Relief and emergency communication network based on an autonomous decentralized uav clustering network," in *SoutheastCon 2015*, pp. 1–8, 2015.

[6] M. Asadpour, D. Giustiniano, K. A. Hummel, and S. Egli, "UAV networks in rescue missions," in *Proceedings of the 8th ACM international workshop on Wireless network testbeds, experimental evaluation & characterization*, pp. 91–92, ACM, 2013.

[7] J. Zelenka and T. Kasanickỳ, "Outdoor UAV control and coordination system supported by biological inspired method," in *In proceedings of Robotics in Alpe-Adria-Danube Region (RAAD), 23rd International Conference on*, pp. 1–7, 2014.

[8] "3DR: "Proven Drone Meets Best-In-Class Sensor"." http://3dr.com/solo-drone. Accessed: 11.2.2018.

[9] J. P. Carvalho, M. A. Jucá, A. Menezes, L. R. Olivi, A. L. M. Marcato, and A. B. dos Santos, "Autonomous UAV outdoor flight controlled by an embedded system using Odroid and ROS," in *CONTROLO 2016*, pp. 423–437, Springer, 2017.

[10] S. A. Engebråten, K. Glette, and O. Yakimenko, "Field-Testing of High-Level Controller for a Multi-Function Drone Swarm." submitted to the 14th IEEE International Conference on Control and Automation, Anchorage, AK, June 12-15, 2018.

[11] "ODROID single board computer." https://en.wikipedia.org/wiki/ODROID. Accessed: 11.2.2018.

[12] "PX4 autopilot." http://pixhawk.org. Accessed: 11.2.2018.

[13] "3DR Solo ardupilot firmware." https://github.com/3drobotics/ardupilot-solo. Accessed: 11.2.2018.

[14] "Ardupilot: Choosing a flight controller ." http://ardupilot.org/ardupilot/index.html. Accessed: 11.2.2018.

[15] "DroneKit programming framework." http://dronekit.io/. Accessed: 11.2.2018.

[16] "3DR Solo dev guide." http://dev.3dr.com/. Accessed: 24.6.2016.

[17] "MAVLink Micro Air Vehicle Communication Protocol." http://www.mavlink.org/mavlink/start. Accessed: 11.2.2018.

[18] "ROS package MAVROS." http://wiki.ros.org/mavros. Accessed: 11.2.2018.

[19] "Seeedstudio technology." http://www.seeedstudio.com. Accessed: 11.2.2018.

[20] "Ubuntu Release End of Life." http://www.ubuntu.com/info/release-end-of-life. Accessed: 11.2.2018.

[21] "Installing ROS on Ubuntu ARM." http://wiki.ros.org/kinetic/Installation/UbuntuARM. Accessed: 11.2.2018.