AirBeam: Experimental Demonstration of Distributed Beamforming by a Swarm of UAVs

Subhramoy Mohanti*, Carlos Bocanegra*, Jason Meyer*, Gokhan Secinti†,
Mithun Diddi*, Hanumant Singh* and Kaushik Chowdhury*
*Department of Electrical and Computer Engineering, Northeastern University, Boston, USA
†Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey

***Abstract*—We propose AirBeam, the first complete algorithmic framework and systems implementation of distributed air-to-ground beamforming on a fleet of UAVs. AirBeam synchronizes software defined radios (SDRs) mounted on each UAV and assigns beamforming weights to ensure high levels of directivity. We show through an exhaustive set of the experimental studies on UAVs why this problem is difficult given the continuous hovering-related fluctuations, the need to ensure timely feedback from the ground receiver due to the channel coherence time, and the size, weight, power and cost (SWaP-C) constraints for UAVs. AirBeam addresses these challenges through: (i) a channel state estimation method using Gold sequences that is used for setting the suitable beamforming weights, (ii) adaptively starting transmission to synchronize the action of the distributed radios, (iii) a channel state feedback process that exploits statistical knowledge of hovering characteristics. Finally, AirBeam provides insights from a systems integration viewpoint, with reconfigurable B210 SDRs mounted on a fleet of DJI M100 UAVs, using GnuRadio running on an embedded computing host.**

I. Introduction

The growing popularity of Unmanned Aerial Vehicles (UAV) can potentially revolutionize the end-to-end connectivity paradigm for IoT, by acting as infrastructure-less mobile base stations and aerial relays [1]. Indeed, ongoing regulatory changes have allowed various cellular service providers [2], base station hardware companies such as Nokia [3], [4] and Ericsson [5], and chip manufacturers such as Qualcomm [6] to actively explore this space.

UAV flight zones are heavily regulated by Part 107 of FAA regulations, which require line-of-sight (LoS), a height cap of 400m and underlying spaces that do not have any non-participating personnel [7]. Inaccessible airspace, defined as no-fly zones, impose additional restrictions due to proximity to airports and governmental buildings, private properties, bird sanctuaries, military installations, among others. Thus, there is need to enable communication links that span considerable distances to overcome these spatial challenges, beyond what is permissible by a single transmitter.

• **AirBeam overview:** As shown in Fig. 1, AirBeam arranges UAVs as a virtual antenna array to form long range links where single-hop connections are unfeasible. In such situations, the ground sensor first transmits the desired information to the UAVs. Following this step, AirBeam solves a number of challenges to ensure the independent action of multiple UAVs result in accurate beamforming: Firstly, it allows the receiver to determine which specific UAVs are not aligned with respect to the others regarding the exact starting moment of their individual transmissions, and returns channel state information (CSI) feedback that allows this fine-tuned adjustment. The real-time channel estimation at the receiver for each transmitter is done through the use of Gold sequence-preambles and GnuRadio implementation. The UAVs use the consolidated feedback from the receiver to set the beamforming weights (along with the start-time determination). The cooperative beamforming algorithm that runs in a distributed manner within each UAV factors in the statistical knowledge of the hovering motion, and the positional change from the moment the preambles were transmitted.

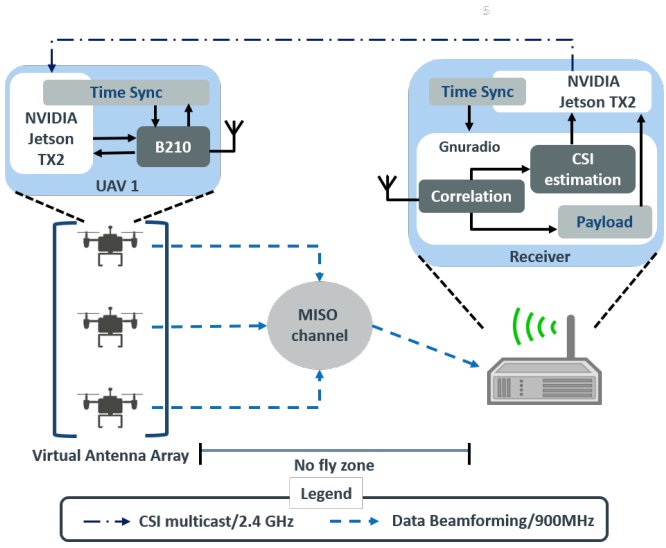

Fig. 1: System architecture for AirBeam

• **Systems challenges:** AirBeam solves many systems-related challenges to realize functioning aerial beamforming towards meeting low size, weight, power and cost (SWaP-C) constraints in UAVs (see Fig. 1). This results in careful design choices on (i) selecting lightweight but capable software defined radios (SDRs) and embedded computing hosts that can be mounted on DJI M100 drones, (ii) solving time synchronization issues, (iii) identifying CSI estimation preamble lengths that allow for accurate enough channel estimation in demanding situations, while ensuring that they can be implemented easily within the chosen hosts, (iv) and real time processing tradeoffs to ensure timely CSI feedback, among others. We motivate the need for aerial beamforming through experiments on multihop relaying of data over UAVs. We also show how distributed beamforming solutions that work well

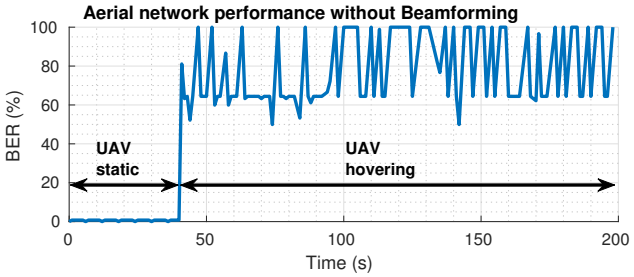

Fig. 2: BER at receiver with UAV static and hovering conditions.

in terrestrial conditions cannot be directly ported for UAVs.

• **Motivation for beamforming:** Prior experimental deployments indicate link failures are highly likely in UAVs [8], [9], and these failures often manifest in multiple links at a time. In such cases, path recovery becomes difficult. These experimental studies assume that UAVs can be deployed in form of an aerial muilti-hop network, and there are always active links between a given UAV pair. However, [10], [11], [12] point out that large tracts of densely populated areas are designated as no-fly zones. Thus, extending the reach of an information relaying aerial network through classical mesh design may not always be feasible. The existence of no-fly-zones and our own link layer measurements motivate the design of AirBeam, in which we use distributed beamforming to increase directivity and signal power, with the goal of achieving the same outcomes as a multi-hop relay.

Preliminary experiments conducted by transmitting static payload with Intel Aero Ready-to-Fly UAV and Intel Dual Band Wireless-AC 8260 WiFi chipset shows significant variations in the bit error rate (BER) in Fig. 2, when the UAV hovers at a height of 25m above the ground receiver (from time 40s onwards).

A. Paper Contributions

The main contributions of AirBeam are as follows:

- We devise the core distributed beamforming approach of AirBeam, which meets QoS requirements at the ground receiver, while ensuring highly directional transmissions. AirBeam functions with intermittent feedback from the receiver, and no communication is needed among the participating UAVs.
- We design a closed loop feedback system using host-based processing to ensure the CSI reaches the UAV transmitters timely, resulting in transmit waveforms aligning at the receiver.
- We implement the complete AirBeam architecture involving significant software and hardware engineering effort, and document the lessons learnt for other researchers. We show how SWaP-C constraints are met by a combination of Gnuradio blocks running on Ettus B210 SDRs interfaced with ARMv8 64-bit NVIDIA TX2 hosts, the weight/speed-up tradeoffs in using FGPA for accelerating CSI computation, and show experimental results on DJI Matrice M100 drones.

We explain the beamforming approach and timing synchronization issues in Sec. II and provide a thorough analysis of factors driving the implementation in Section III. We discuss the CSI computation speed achived through an FPGA implementation in Sec. IV. Experimental results and discussions are reported in Section V. Finally, we conclude in Section VI.

II. Beamforming in AirBeam

UAVs can replicate and broadcast the same messages, which are combined at the packet level at a target receiver [13]. In AirBeam, instead, we use a PHY-layer technique that creates a constructively combines signals at the receiver by equalizing the phases of the channel perceived by each transmitter. While this increases the received power in the form N^2, where N is the number of available transmit antennas, it poses some challenges [14]: First, the transmitters should be accurately synchronized in frequency and time. Second, the beamforming weights at the transmitter should be rapidly updated to counter the effects of the wireless channel.

A. Beamforming theory

Consider a MISO system composed of N UAVs, each of which is equipped with L antennas. The relationship between the received signal $\mathbf{y}$ and transmitted signal $\mathbf{x}$ is: $y[m] = \mathbf{h}^*\mathbf{x[m]} + n[m]$, where $\mathbf{h} = [h_1, ..., h_{NL}]$ are the fixed channel gains from the transmit antenna h_l to the receive antenna, and $n[m]$ represents the additive white Gaussian noise (AWGN) at the receiver with a Normal distribution $N(0, \sigma_n)$. The receiver estimates the channel continuously and updates the transmitters with the beamforming weight vector $\mathbf{w}$. This allows the transmitted symbols $s[m]$ to be multiplied by the beamforming weights to construct the new signal $x[m] = \sqrt{E_s}\mathbf{w}^H s[m]$, where E_s is the average energy of the transmitted signal $x[m]$ with normalized constellation symbols at any instant m ($E(|s[m]|^2) = 1$), with E being the mean function. In this closed loop system, the weights are selected as to maximize the average mutual information function described in (1), offering a total capacity in bits/Hz/s given in (2), where P is the power component.

$$I_{BF}(\mathbf{w}, P) = \mathbf{E}[ln(1 + P|\mathbf{h}^H\mathbf{w}|^2)] \quad (1)$$

$$C_{BF}(\mathbf{w}, P) = \mathbf{E}[log(1 + P\frac{|\mathbf{h}^H\mathbf{w}|^2}{\sigma_n})] \quad (2)$$

AirBeam is designed for the ISM band with narrowband channels, tailored for IoT applications. Under these conditions, the signal is expected to suffer flat-fading, i.e. $W << W_c$, where these terms imply the communication and coherence bandwidth, respectively. This lets us model the multipath effect on every symbol by a single complex number $h[m]$ as shown in (1) and (2). In addition, the channel estimation may contain errors due to shadowing caused by buildings, resulting in slight errors $\hat{h}[m] = h[m - D] + \Delta h[m]$.

B. Validation of beamforming operation

To test the beamforming operation at 900MHz ISM band, we deploy the UAVs in a temporary setup as shown in Fig. 3 (note SWaP-C constraints and need for on-board computing will result in a design evolution, described in Sec. III). Four

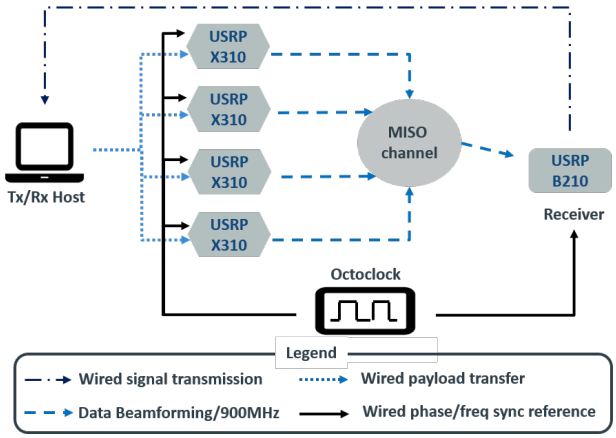

Fig. 3: Architecture of static on-ground test-bed setup evaluating multi-user transmit beamforming.

Ettus X310 SDRs (as the transmitter array) and a B210 SDR (receiver) separated by 2m are connected to a common host computer. The transmit gain is kept low to emulate longer range between the transmitters and receiver. An Ettus Octoclock is used as the external 10MHz and PPS reference for all the transmit and receive radios. We create separate MATLAB sessions running the beamforming algorithm in the host PC, with the CSI estimated by the receiver being easily accessible to the transmitter side. Each transmitter generates a frame with pre-defined Gold sequence for synchronization and channel estimation, followed by OFDM blocks encapsulating 64-QAM modulated symbols, with proper zero-padding and cyclic prefix insertion to deal with inter symbol interference (ISI). Upon frame synchronization at the receiver, we compute the BER of the payload to measure the performance of the link. The receiver employs a simple least squares (LS) fit to estimate channel response, i.e. $\hat{h} = \hat{G}/G$, where $\hat{h}$ is the estimated response and $\hat{G}$ is the presumed received Gold sequence.

$$w[m] = \left(\frac{\hat{h}[k-D]}{||\hat{h}[k-D]||} \right)^{-1} \tag{3}$$

MATLAB results in a non-negligible time lapse of D samples between channel estimation and beamforming transmission. Thus, the channel perceived by the beamformer is modeled as $h[m-D]$. with (3) giving the beamforming step at the transmitter.

Fig. 4 shows how BER reduces dramatically by using 4 antennas, with each antenna installed on a different hovering UAV and connected via long cables to the ground based centralized MATLAB host PC, shown in the snapshot 5, as opposed to single antenna communication for different modulation schemes.

From Fig. 6, we see the drastic decrease in the achievable capacity (normalized) per antenna due to hovering, when using the most updated CSI. Analytically, this can be explained as the impact on the achievable capacity obtained using $C(m) = log_2(1+(P|\mathbf{w}(m-D)\mathbf{h}^H(m)|^2/\sigma^2))$, where the beamforming weights are computed from (3).

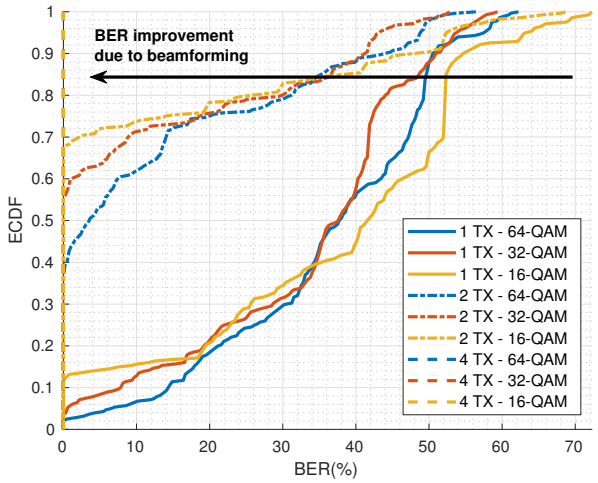

Fig. 4: Benefit of using Beamforming for disconnected aerial networks.

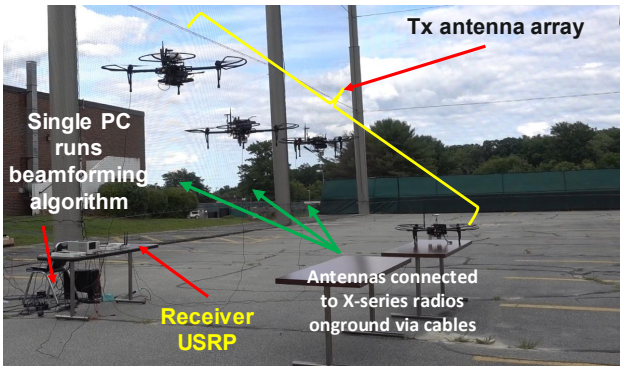

Fig. 5: UAS test-bed setup to evaluate multi-user tranmsit beamforming.

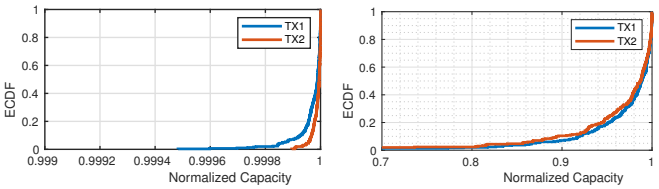

Fig. 6: ECDF of the normalized capacity per antenna in a 2x1 MISO system for static (on the ground, left) and hovering (flying, right) UAV deployment. Note the x-axis.

Several factors may cause mis-alignment during beamforming, which not only lowers capacity but also has SWaP-C design impacts : (i) Variation of the channel over time: We realize that MATLAB processing delays are unacceptable, thus motivating the use of GnuRadio. (ii) Embedded processing: A single host computer is infeasible for multiple SDRs on geographically separated UAVs. Thus, we need a local host light enought to be carried on the UAVs, while also ensuring speedy execution of the GnuRadio code. (iii) Incorrect start time: Each SDR must begin its operation at exactly the same time (even if perfectly time synchronized, this is not assured).

We show next how we address these issues to ensure practical implementation in an aerial scenario.

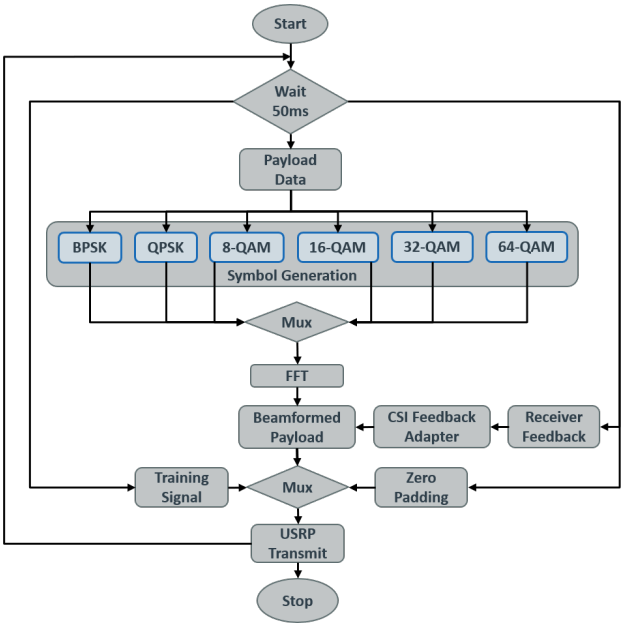

Fig. 7: GnuRadio based Aerial Transmit Beamforming Algorithm.

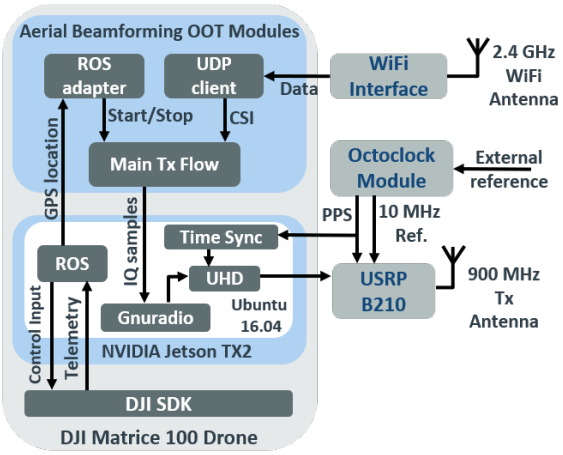

Fig. 8: UAS solution stack consisting both hardware and software components.

III. SWaP-C Design Optimizations

The overall design decisions that address the shortcomings in the previous section are captured in Fig. 8, We will refer to this in the subsequent text.

A. Variation of channel over time

Our revised approach required translating the entire processing chain into GnuRadio for real-time operation. For CSI estimation, the computation is divided into two stages:

Stage 1: Correlation of incoming samples against the stored Gold sequences to detect which specific sequence is received. This operation consists of $2N$ multiplications and $2N$ additions per transmitter for each incoming sample, where N is the length of the Gold sequences. Let the computation time of this operation be t_{corr}.

Stage 2: Least-squares estimation of the received Gold sequence to the associated stored sequence, which requires $6N+2$ multiplications and $6N+2$ additions for each incoming packet. Let the computation time of this operation be t_{LS}.

With baseband sampling period, $t_s = 2.5\mu s$ and time between received packets, $t_P = 84ms$, then, for real-time operation, we must maintain $t_{corr} < t_s$ and $t_{LS} < t_P$. However, since $t_s << t_P$ and the algorithmic complexity of both correlation (Stage 1) and LS estimation (Stage 2) are $O(N)$, the limiting factor on real-time operation is the correlation, for which $t_{corr} < t_s$ must hold.

For GnuRadio implementation of our beamforming receiver, we target a USRP Ettus B210 radio connected to a host PC having 16GB RAM, Intel Core i7-7700HQ processor and 512GB SSD hard drive. Although our preliminary MATLAB-based beamforming allow packet reception every 84ms, the CSI estimates are updated every 700ms due to computational limitations. In GnuRadio, we accomplish this in 50ms time. Any faster updates result in buffer overflow. We experimentally show that this 50ms delay is tolerable when used in conjunction with the faster GnuRadio receiver. Additional motivations in moving beamforming to GnuRadio concern the capability to create/manipulate payloads in real time by directly using USRP Hardware Driver (UHD) primitives and ease of installation in embedded computing platforms for UAVs.

As shown in Fig. 7, the GnuRadio based transmit beamforming algorithm pauses for 50ms between each iteration, to avoid overflowing the buffer memory on transmitter SDR. It then proceeds with the creation of payload bits, which are then transformed into symbols of different modulation schemes. In our case, we choose BPSK, QPSK, 8-QAM, 16-QAM, 32-QAM and 64-QAM. The symbols are multiplied with the beamweights, to generate the payload. The beamweights are calculated in the CSI feedback adapter block, using receiver generated packets that are transmitted using an out-of-channel link (receivers are base stations, with higher transmit powers). The beamformed payload is then transferred to the radio transmit chain, ready to be transmitted over the air.

B. Embedded processing

This issue hinders precise time synchronization between distributed devices, as seen in Fig. 9a. The payload weight restrictions on the UAVs (3.6kg max take off weight for the DJI M100), require exploration of small form factor radios and computing platforms. For this reason, we select the Ettus B210 SDRs along with the NVIDIA Jetson TX2 host computers, to be mounted on the UAVs. The Jetson TX2 boards add 85gms with 4core 64-bit and 2 Denver A57 CPUs, capable of executing GnuRadio and UHD applications without buffer overflow. Moreover it is able to interact with the UAV's DJI Software Development Kit (SDK) and Robot Operating System (ROS) middleware. A B210 SDR has many similar performance indices compared to the larger X310, and adds 350gms of weight. Fig. 8 shows the connection between Jetson TX2 and GnuRadio module, with detailed interactions between different functional blocks for a DJI M100 UAV platform.

C. Timing and synchronization

The main drawback with software-based time synchronization is the operating systems (OS) lacking real-time operation capabilities, where users or applications do not have exact control on when threads or processes are going to be run. This figure showcases an observed scenario for software-based

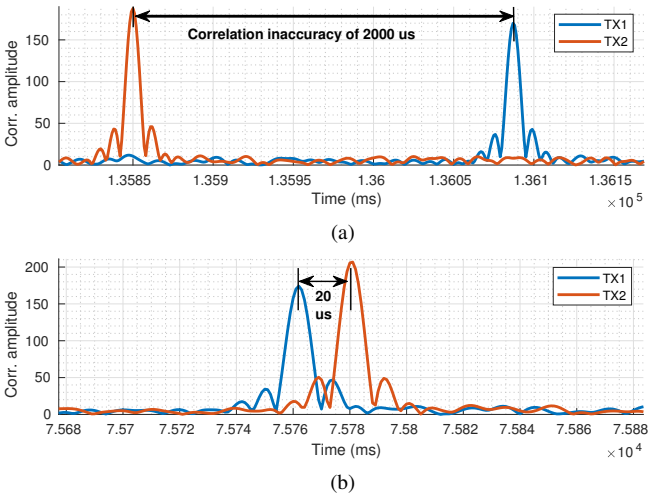

Fig. 9: Timing synchronization issues before (a) and after (b) moving to our proposed hardware based approach.

synchronization, where the alignment of the training signals from the two transmitters are off by $2000\mu s$. Our approach to overcome this problem is to push the time synchronization processes close to the radio hardware, with minimum involvement of the OS.

In our setup, the time synchronization starts when the UAVs reach the intended GPS based waypoint. The algorithm then gives highest priority to the time sync thread by locking the other concurrent software threads from causing context-switches, so it will mitigate undesired but highly likely software lags or thread delays while setting the time value on local USRP. Once the lock is achieved, the algorithm proceeds with receiving and parsing Network Time Protocol (NTP) message collected from the local NTP server (which in our case is the receiver base station). This NTP data provides the reference time for setting the system clock for the Jetson TX2 host on the UAV. Taking the seconds value of the system time, and the PPS input from the Octoclock, the USRP time is set as the next second value, on the next PPS instant. This ensures that all of the distributed transmitter and receiver radios collectively lock their hardware time to the same value.

After setting up the time on each USRP, each GnuRadio process decides how many seconds it should wait before starting the transmission and sets the through UHD API. This guarantees the GnuRadio processes start transmitting simultaneously, even if the processes themselves are not initialized at the same time. In between the successful setting of the radio hardware time and the actual transmission start time, the host process on OS is put to sleep, in order to prevent buffer overflows at the radio UHD driver software. Although

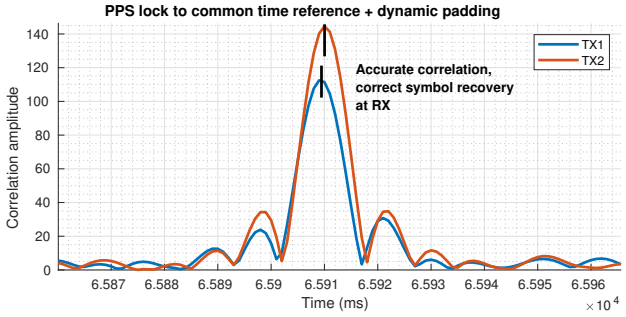

Fig. 10: Accurate cross-correlation within $1\mu s$ accuracy with time sync and delay mechanism enabled.

this approach reduces the correlation inaccuracy to $20\mu s$ from the previous $2000\mu s$ (see Fig. 9b) on B210 radios[1], it still fails to satisfy the precision required for data beamforming, where the correlation accuracy needs to be within $1\mu s$ between all the transmitted signals. To achieve the desired precision, the receiver calculates the delay between the transmit streams and piggybacks this information along with the CSI feedback, back to the individual transmitters. The transmitters delay their signals accordingly with additional zero-padding (similar to the Time Advance procedure to synchronize nodes in 3GPP LTE standard) to sync with the transmitter radio peer having the latest transmission start time. Thus, we satisfy the target of correlation accuracy of within $1\mu s$, as shown in Fig. 10.

IV. FPGA-based Receiver Processing

Beamforming in mobile scenario depends on accurate and fast CSI estimation. We recall our GnuRadio implementation estimates the CSI and generates feedback in 50ms. Our experimental results reveal that in the presence of classical GPS, the lateral displacement of the UAV is typically +/- 0.5m, as shown in the histogram in Fig. 11, even when the UAV is assigned to a fixed location in space. Thus, the CSI needs to be continuously computed and communicated back to the transmitters. To ensure that the received feedback is timely, there are two approaches: (i) we use real-time kinematic GPS that continuously corrects GPS errors (resulting in +/- 10cm as shown in Fig. 11) through a combined ground base station and a UAV-mounted device, or (ii) we use alternate methods to further speed up the CSI computation. For the latter approach, we describe the speed-up possible by undertaking some of the computation within an FPGA.

A. FPGA algorithm design

Our target hardware for an FPGA-based implementation of the beamforming receiver is the Xilinx ZC706 development kit for the Zynq-7045 system-on-chip (SoC) FPGA, with an Analog Devices FMCOMMS3 daughterboard containing an AD9361 RF transceiver. This FPGA contains 350,000 logic cells, 19.1 Mb of RAM, and 900 DSP slices, which are the major resources that are utilized in the receiver design. The

[1]We should note that, these steps are sufficient to provide precise synchronization on X310 radios, where B210 radios fail to achieve same precision on real-time operations.

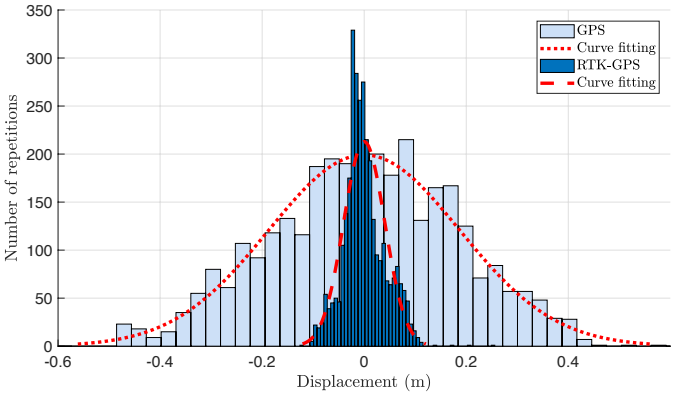

Fig. 11: Gaussian distribution of UAV lateral displacement

first step of re-designing the receiver algorithm for FPGA implementation is to split the algorithm into functional pieces with different requirements: (1) correlating the incoming data with the stored training signals and (2) calculating a least-squares estimate to fit the received training signals with the stored training signals. In order to maximize the computation speed increase, the beamforming algorithm is partitioned into high performance and low performance components. The high-performance components are those that can be parallelized in the FPGA fabric to reduce the computation time, and include such functions as the buffering of incoming samples, correlation of samples with the stored training signals, and estimation of CSI. Low-performance functions are those that will not drastically speed up computation time if placed in the FPGA, which include transferring the CSI data to a host computer via a UDP interface. This low-performance task will be performed by software in the SoC processing system (PS).

A high-level block diagram of the hardware and software processing blocks is shown in Figure 15. The incoming I/Q samples from the AD9361 IP core are split into parallel streams. One stream is routed directly to the I, Q, and Valid output stream, in order to allow the samples to be output immediately. The second stream pushes the I/Q samples into the CSI estimation chain. This allows received samples to be processed in the CSI estimation chain while simultaneously being sent to the host for demodulation of the signal. Thus, the two paths do not interfere with one another.

1) Correlator design: The next block on the CSI estimation chain is the correlator block. The number of parallel DSPs (and thus, the number of allowable parallel multiplications) was chosen to be 96 per channel (32 for I data, 32 for Q data, and 32 for magnitude data). To support, say 5 transmitters/UAVs, this means a total of 480 of the 900 available DSP slices will be used for the correlators. In addition, the block RAMs used for received I and Q sample storage must be 16 bits * 32 = 512 bits wide. The local training signals are stored in similar 512-bit wide RAM blocks which are only 128 words long (512 x 128), for storing the Gold sequences which are originally 16 x 4096. Each correlation sample is calculated using the

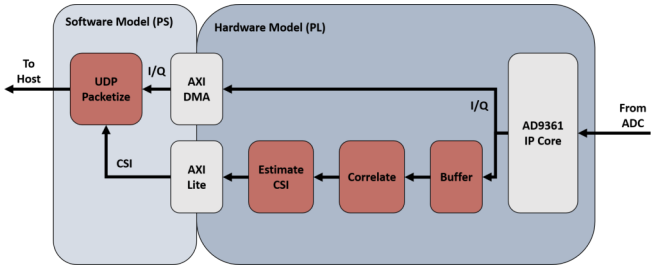

Fig. 12: A high level block diagram of the hardware and software processing blocks for the beamforming receiver

following formula:

$$X[n] = |\sum_{k=0}^{4095} (r[n+k] * g^*[k]) \ | \tag{4}$$

where $X[n]$ is the n-th correlation sample, $r[n+k]$ is the $[n+k]$-th received sample and $g^*[k]$ is the complex conjugate of the k-th Gold sequence sample. Let the components in Equation 4 be represented in I/Q format as $r[k] = r_i[k] + jr_q[k]$ and $g^*[k] = g_i[k] - jg_q[k]$. Then the equation, letting $m = [n+k]$, can be re-written as

$$\begin{aligned} X[n] = |\sum_{k=0}^{4095} & \ r_i[m] \ g_i[k] + jr_q[m] \ g_i[k] \\ & -jr_i[m] \ g_q[k] - j^2 r_q[m] \ g_q[k] \ | \end{aligned}$$

where we use the fact that $\{g_i[k] = g_q[k], \quad \forall \quad k\}$ to get

$$\begin{aligned} X[n] = |\sum_{k=0}^{4095} & \ (r_i[m] \ g_i[k] + r_q[m] \ g_i[k]) \\ & +j(r_q[m] \ g_i[k] - r_i[m] \ g_i[k]) \ | \end{aligned}$$

To save on computation complexity and reduce hardware resources, we approximate $|A + jB| \approx |A| + |B|$, since we are simply identifying the index of the peak correlation value and disregard the actual value of the peak. Thus, the final correlator calculation becomes

$$\begin{aligned} X[n] = |\sum_{k=0}^{4095} r_i[m] \ g_i[k] + \sum_{k=0}^{4095} r_q[m] \ g_i[k] \\ + \sum_{k=0}^{4095} r_q[m] \ g_i[k] - \sum_{k=0}^{4095} r_i[m] \ g_i[k] \ | \end{aligned} \tag{5}$$

To compute each correlation sample, 64 parallel MAC operations occur over 128 clock cycles. The MAC operations compute the sums from (5). 32 parallel MACs are used to compute $X_i[n] = \sum_{k=0}^{4095} r_i[m] \ g_i[k]$ and 32 parallel MACs are used to compute $X_q[n] = \sum_{k=0}^{4095} r_q[m] \ g_i[k]$. The MAC equation is $x = a * b + c$, and in the correlator calculations, each MAC output is expressed as

$$x_{i,p}[k] = r_{i,p}[k] * g_{i,p}[k] + x_{i,p}[k-1]$$

$$x_{q,p}[k] = r_{q,p}[k] * g_{i,p}[k] + x_{q,p}[k-1]$$

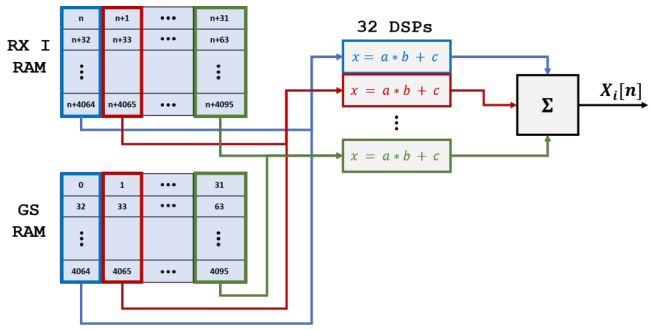

Fig. 13: A block diagram showing the correlation computation for a single sample $X_i[n]$ or $X_q[n]$.

where $x_{i,p}[k]$ and $x_{q,p}[k]$ are the p-th I-sample and Q-sample MAC outputs, respectively, at the k-th clock cycle, $r_{i,p}[k]$ and $r_{q,p}[k]$ are the p-th outputs of the received (RX) I-sample and Q-sample RAMs at the k-th clock cycle, respectively, and $g_{i,p}$ is the p-th Gold sequence (GS) RAM output at clock cycle k. The MACs accumulate over 128 clock cycles, after which the 32 I-sample MAC outputs and 32 Q-sample MAC outputs are then summed together to compute $X_i[n] = \sum_{p=0}^{31} x_{i,p}[127]$ and $X_q[n] = \sum_{p=0}^{31} x_{q,p}[127]$.
A block diagram of the correlation computation for $X_i[n]$ is shown in Figure 13. The same structure is used to simultaneously calculate $X_q[n]$, and the two results are used to compute the final correlation value $X[n]$ as

$$X[n] = \mid X_i[n] + X_q[n] \mid + \mid X_q[n] - X_i[n] \mid \qquad (6)$$

2) Peak Detector & CSI Estimator Design: The correlations are performed so that correlation peaks can be detected in order to identify the starting index of received training signals. Correlation peaks are detected by comparing the correlator outputs to a hard-coded threshold value. Once this threshold has been crossed, the peak detection enters a *peak tracking* state where the maximum correlator value is tracked and the corresponding sample index is stored. The peak tracking is performed for the 16 samples immediately following the sample at which the correlator threshold is crossed. After this, the peak detector stores the index of the maximum correlator value and the channel estimation can be performed. Since slight timing offsets between transmitters can occur even with good timing synchronization, the indices of the correlator peaks may be off by a sample or two. This is accounted for in the algorithm design, since each channel performs the peak tracking separately.

Consider a transmitted signal g that is observed at a receiver as r, after passing through the wireless channel characterized by h, where all parameters are complex-valued. The transmitted signal over time is represented by a vector of samples $\mathbf{g} = [g_1, g_2, ..., g_n]^T$ for which there is a corresponding vector of observations $\mathbf{r} = [r_1, r_2, ..., r_n]^T$. Assuming h is fixed for all samples in an observation window, the resulting relationship between the transmitted signal, received signal, and channel characterization is $\mathbf{r} = h * \mathbf{g}$. This is an example of an over-determined linear system, where there are more equations than unknown parameters. To estimate the true value of h, a least-squares fit can be performed to determine the value of h that minimizes the sum of the squares of the residuals, defined as

$$\hat{h}_{LS} = \text{argmin}_h [\ \sum_{k=1}^{n} (r_k - h g_k)^2\ ]$$

For this complex-valued system, the least-squares solution [15] is given by

$$\hat{h}_{LS} = (\mathbf{g}^* \mathbf{g})^{-1} \mathbf{g}^* \mathbf{r} \qquad (7)$$

where $\mathbf{g}^*$ is the conjugate transpose of $\mathbf{g}$. We can express this relationship as a sum of multiplications on a per-sample basis, which is more suited for FPGA implementation, as

$$\hat{h}_{LS} = (\ \sum_{k=1}^{n} g_k^* g_k\ )^{-1} (\ \sum_{k=1}^{n} g_k^* r_k\ )$$

Separating the true signal and observations into I and Q components, the calculation becomes

$$\hat{h}_{LS} = (\ \sum_{k=1}^{n} g_{i,k}^2 + g_{q,k}^2\ )^{-1} (\ \sum_{k=1}^{n} g_{i,k} r_{i,k} + g_{q,k} r_{q,k} + j[g_{i,k} r_{q,k} - g_{r,k} r_{i,k}]\ )$$

However, since $\mathbf{g}$ is the fixed-valued transmitted training signal, a copy of which is stored locally, the term $\sum_{k=1}^{n} (g_{i,k}^2 + g_{q,k}^2)$ is a constant. Furthermore, this value is the same for all of the generated Gold sequences in the design, and can be ignored in the FPGA computation and grouped with a gain term in software. Thus, the estimation calculation becomes

$$\hat{h}_{LS} = \sum_{k=0}^{4095} g_{i,k} r_{i,p+k} + g_{q,k} r_{q,p+k} + j[g_{i,k} r_{q,p+k} - g_{q,k} r_{i,p+k}]$$

where p is the sample index of the correlator peak value. We once again use the fact that $\{g_i[k] = g_q[k],\ \ \forall\ \ k\}$ to simplify this calculation to

$$\hat{h}_{LS} = \sum_{k=0}^{4095} g_{i,k} r_{i,p+k} + g_{i,k} r_{q,p+k} + j[g_{i,k} r_{q,p+k} - g_{i,k} r_{i,p+k}]$$

which can be reduced to a function of summations of multiplications suitable for MAC operations as:

$$\hat{h}_{LS} = [\ \sum_{k=0}^{4095} g_{i,k} r_{i,p+k} + \sum_{k=0}^{4095} g_{i,k} r_{q,p+k}\ ] + j[\ \sum_{k=0}^{4095} g_{i,k} r_{q,p+k} - \sum_{k=0}^{4095} g_{i,k} r_{i,p+k}\ ] \qquad (8)$$

The channel estimation step, therefore, requires just two simultaneous MAC operations that calculate $\{\hat{h}_i = \sum_{k=0}^{4095} g_{i,k} r_{i,p+k}\}$ and $\{\hat{h}_q = \sum_{k=0}^{4095} g_{i,k} r_{q,p+k}\}$. These are calculated in $4096/100\text{MHz} \approx 41\mu s$ for the assumed 100 MHz FPGA clock rate. Since the minimum speed requirement implies that calculation must be completed between the end

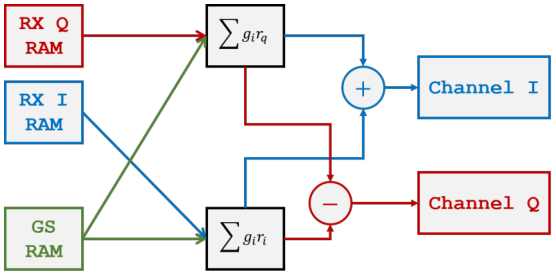

Fig. 14: A block diagram showing the least-squares computation for CSI estimation.

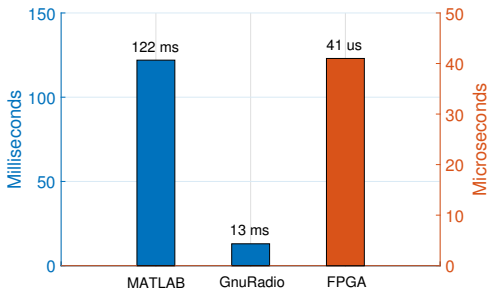

Fig. 15: CSI estimation time in MATLAB, GnuRadio and FPGA

of a transmitted training signal and the beginning of the next packet, it is performed using just two DSP blocks for conservation of resources. The computation occurs over 4096 FPGA clock cycles and once completed, the channel I and Q estimates are stored in FPGA registers for later access by the PS software. A block diagram showing the least-squares channel estimator design is shown in Figure 14.

3) Clock & Timing Considerations: In this design, there are two main clock domains: the AD9361 ADC clock and the FPGA master clock. The AD9361 ADC clock is an external clock generated in the AD9361 transceiver that runs at a fixed rate equal to the RF sampling rate of 540 kHz, while the FPGA clock, which is internally generated, can run at a designer-selected frequency as desired. The major limitation of this FPGA master clock is the complexity and implementation of the design, which will limit the maximum speed at which it will run.

The FPGA design was implemented three times, using FPGA master clock rates of 50, 100, and 200 MHz. We observe that the design timing requirements are met for both 50 MHz and 100 MHz, so 100 MHz was chosen as the actual operating clock rate. Note that there is significant room for redesigning the algorithm to optimize it for a 200 MHz clock, which can potentially allow for doubling the number of transmitters supported.

V. Performance Evaluation

AirBeam is evaluated on an experimental setup of 4 transmitter UAVs, with each UAV consisting of a NVIDIA Jetson TX2 module, interfaced with a B210 radio and mounted on a DJI Matrice M100 UAV. The receiver B210 radio was connected to a ground based static host. The UAVs receive the GPS co-ordinates for hovering via the DJI SDK system. Once all the UAVs are at the expected location, the transmitter SDR hosts synchronize their own clock with the local server, proceed to fine tune the radio transmissions to within $1\mu s$ delay of each another and start the beamforming transmission. The receiver, after extracting the payload, multicasts the CSI feedback to the transmitters along with piggybacking transmission delay information, if required.

A. Use of FPGA and SWAP-C Considerations

An AD9361 reference software design is provided by Analog Devices for use with the Xilinx ZC706. UDP capabilities are included with the lightweight internet protocol (lwIP) stack which is usable by this application.

Empirical measurements of computation time shown in Fig. 15 for the correlation and CSI components indicate that these functions have a minimum computation time of 122ms in MATLAB and 13ms in GnuRadio. The FPGA implementation, on the other hand, will compute correlation and CSI estimation is $41\mu s$. Thus, the FPGA design has a speedup of over 3000x compared with the MATLAB design and over 300x improvement compared with GnuRadio. This computation speed does not include the transfer of calculated CSI data from the FPGA to the host processor that would be necessary in a system implementation. However, this data transfer could be achieved in $<$1 ms over a simple 115kHz baud rate UART interface for a system of 5 transmitters/UAVs. Since the FPGA implementation uses the Xilinx Zynq-7000 SoC ZC706 developmental board, it adds an extra 396 grams of weight to the UAV. This in turn reduces the flying time by 56.18%. Since this reduced flight time is not enough to generate sufficient data for a fair BER comparison between GnuRadio and FPGA, we describe experimental results with GnuRadio in this paper, and leave the FPGA-based processing for future work, where we will replace the FPGA development board with a smaller form factor integrated FPGA-host device, port our algorithm there and install it on the UAV.

B. Beamforming to improve the BER

Using the same experiment settings from Sec. II-B, we gradually increased the number of transmitter UAVs, and measured both channel gain and BER at the receiver with different modulation schemes. Fig. 16 showcases the effect of UAV hovering on channel gain fluctuations, with low fluctuations observable during moments of high stability of the UAVs. Channel gain increases significantly as more UAVs participate in beamforming. This improvement in channel gain, coupled with fast CSI estimation feedback from the receiver and similar fast beamformed payload transmission in the forward path, improves BER at the receiver, as shown in Fig. 17. With one transmitting antenna, the BER for any choice of modulation scheme is between 30 - 40% which gets reduced to 0% as the number of transmitting beamforming UAVs are increased to 4. Thus, AirBeam improves the link performance by 30 - 40% at the receiver depending on the modulation scheme used.

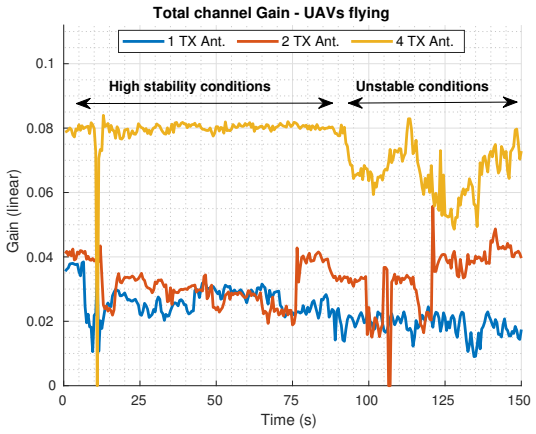

Fig. 16: Channel gain ($|\mathbf{h}^H\mathbf{h}|$) with 1, 2 and 4 antennas.

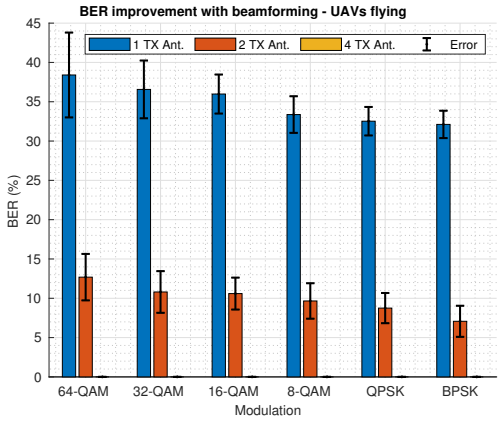

Fig. 17: Average BER perceived with 1, 2 and 4 antennas. Error bars showing standard deviation in measurement.

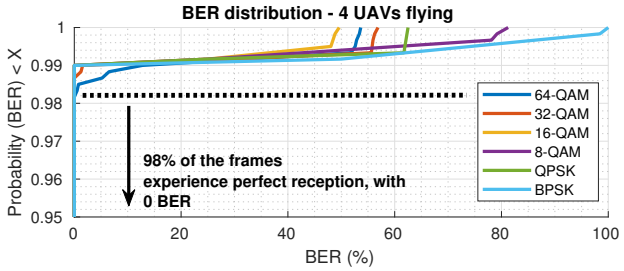

Fig. 18: Probability of BER below certain value (x-axis) with different modulation schemes, 4 UAV transmitters and 1 ground receiver.

Fig. 18 provides the statistical information that guarantees this observation, i.e., AirBeam enables aerial beamforming with the probability of BER around 0 is 99-98%, for all the modulation schemes used.

VI. Conclusion

AirBeam achieves aerial beamforming communication under practical SWaP-C constraints. We have demonstrated the feasibility of a practical system with preliminary experiments and provided extensive systems level implementation on a real test bed. Comparing the performance of data communications with traditional aerial networks, we see that our approach achieves 30-40% reduction in BER, when the numbers of UAV transmitters are increased from 1 to 4 and 99% probability of BER to be near 0 with this approach. Our next steps will focus on demonstrating enhanced beamforming algorithms that can operate even with intermittent CSI information resulting from losses from the receiver-generated feedback packets.

Acknowledgment

This work is supported by DARPA under grant N66001-17-1-4042. We are grateful to Dr. Tom Rondeau, program manager at DARPA, for his insightful comments and suggestions that significantly improved the quality of the work.

References

[1] Y. Zeng, J. Lyu, and R. Zhang, "Cellular-connected UAV: Potential, challenges and promising technologies," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 120–127, 2019.

[2] Vodaphone. (2018) Beyond visual line of sight drone trial report. [Online]. Available: https://www.vodafone.com/content/dam/vodafone-images/media/Downloads/Vodafone_BVLOS_drone_trial_report.pdf

[3] I. Kovacs, R. Amorim, H. C. Nguyen, J. Wigard, and P. Mogensen, "Interference analysis for UAV connectivity over LTE using aerial radio measurements," in *Proc. IEEE Vehicular Technology Conf. (VTC)*, 2017.

[4] R. Amorim, H. Nguyen, P. Mogensen, I. Z. Kovacs, J. Wigard, and T.Sorerensen, "Radio channel modeling for uav communication over cellular networks," *IEEE Wireless Communications Letters*, vol. 6, no. 4, p. 514517, 2017.

[5] X. Lin, V. Yajnanarayana, S. D. Muruganathan, S. Gao, H. Asplund, H.-L. Maattanen, M. Bergstrom, S. Euler, and Y.-P. E. Wang, "The sky is not the limit: LTE for unmanned aerial vehicles," *IEEE Communications Magazine*, vol. 56, no. 4, p. 204210, 2018.

[6] Qualcomm. (2017) LTE unmanned aircraft systems. [Online]. Available: https://www.qualcomm.com/media/documents/files/lte-unmanned-aircraft-systems-trial-report.pdf

[7] "Federal Aviation Authority, Fact Sheet - Small Unmanned Aircraft Regulations (Part 107)," https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=22615, accessed on July, 2018.

[8] B. Fu and L. A. DaSilva, "A mesh in the sky: A routing protocol for airborne networks," in *MILCOM 2007 - IEEE Military Communications Conference*, Oct 2007, pp. 1–7.

[9] G. Secinti, P. B. Darian, B. Canberk, and K. R. Chowdhury, "Resilient end-to-end connectivity for software defined unmanned aerial vehicular networks," in *IEEE Intl. Symp. on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Oct 2017, pp. 1–5.

[10] J. Bellingham, A. Richards, and J. P. How, "Receding horizon control of autonomous aerial vehicles," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 5, May 2002, pp. 3741–3746 vol.5.

[11] M. Alighanbari, Y. Kuwata, and J. P. How, "Coordination and control of multiple uavs with timing constraints and loitering," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 6, June 2003, pp. 5311–5316 vol.6.

[12] P. Sarunic and R. Evans, "Hierarchical model predictive control of uavs performing multitarget-multisensor tracking," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 3, pp. 2253–2268, July 2014.

[13] N. Hossein Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, Dec 2016.

[14] S. Jayaprakasam, S. K. A. Rahim, and C. Y. Leow, "Distributed and collaborative beamforming in wireless sensor networks: Classifications, trends, and research directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2092–2116, Fourthquarter 2017.

[15] C. W. Therrien, *Discrete Random Signals and Statistical Signal Processing*. Prentice Hall, 1992, pp. 520-522.