A Hybrid Lossless and Lossy Compression Scheme for Streaming RGB-D Data in Real Time

Mark Coatsworth
Ryerson University
350 Victoria St.
Toronto, Ontario, Canada
mark.coatsworth@ryerson.ca

Jimmy Tran
Ryerson University
350 Victoria St.
Toronto, Ontario, Canada
q2tran@ryerson.ca

Alexander Ferworn
Ryerson University
350 Victoria St.
Toronto, Ontario, Canada
aferworn@ryerson.ca

***Abstract*—Mobile and aerial robots used in urban search and rescue (USAR) operations have shown the potential for allowing us to explore, survey and assess collapsed structures effectively at a safe distance. RGB-D cameras, such as the Microsoft Kinect, allow us to capture 3D depth data in addition to RGB images, providing a significantly richer user experience than flat video, which may provide improved situational awareness for first responders. However, the richer data comes at a higher cost in terms of data throughput and computing power requirements. In this paper we consider the problem of live streaming RGB-D data over wired and wireless communication channels, using low-power, embedded computing equipment. When assessing a disaster environment, a range camera is typically mounted on a ground or aerial robot along with the onboard computer system. Ground robots can use both wireless radio and tethers for communications, whereas aerial robots can only use wireless communication. We propose a hybrid lossless and lossy streaming compression format designed specifically for RGB-D data and investigate the feasibility and usefulness of live-streaming this data in disaster situations.**

Keywords—3D; video; streaming; compression; point cloud; USAR; response robot.

I. Introduction and Background

Natural disasters that occur in densely populated urban areas often result in people trapped under collapsed structures. Emergency personnel have a limited amount of time to find and rescue survivors, during which they must first assess the collapsed structure and determine the risks associated with secondary collapse, develop a search strategy, identify access points and try to locate survivors.

Ground-based and aerial robots are becoming much more common in disaster response operations as they hold the potential of allowing operators to evaluate disaster scenes when human access is inconvenient, undesirable or impossible. Much of our earlier work has focused on techniques to improve these assessments of disaster environments, largely through the use of common off the shelf (COTS) cameras to capture RGB-D scene data. Custom algorithms combined with software processing techniques have allowed us to generate rich three-dimensional renderings of these environments, further enhancing our understanding of these collapsed structures (see Fig. 1).

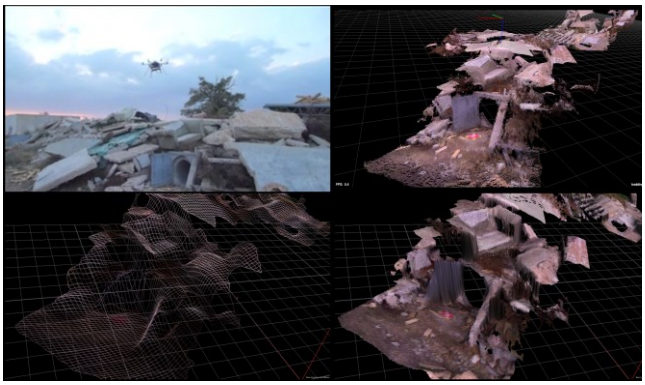
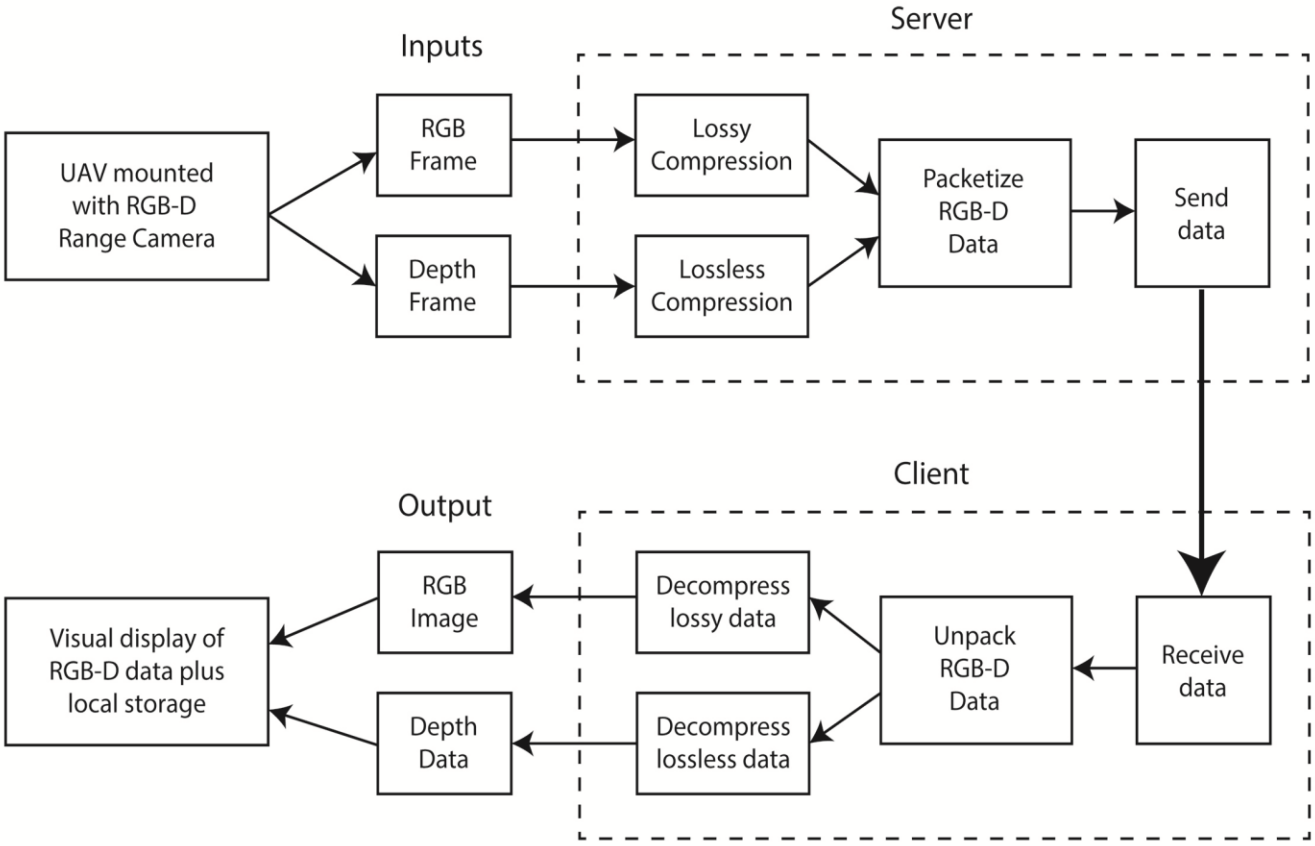


Fig. 1 Sample 3D scan of a rubble pile at Disaster City, College Station, Texas built for USAR training

Several significant problems persist related to the inability to transmit data directly from a UAV. First, there is always time sensitivity in USAR operations; any opportunity to speed up search efforts will increase the likelihood of rescuing survivors. Secondly, the UAVs which perform aerial surveys are controlled and navigated by operators on the ground who could take advantage of the data as it is being gathered. Finally, there is always a possibility that a UAV will crash inaccessibly, thus trapping its onboard data with it.

In this paper we present the findings of our investigation in streaming rich situational awareness data over a wired or wireless network in real time. Other works in this area involve transmitting compressed point cloud data where the data size is reduced at the cost of model resolution [1]. We propose an alternate approach where the robot transmits the compressed raw RGB-D data and the point cloud creation is computed at the receiving end. We present a hybrid lossy and lossless compression algorithm specifically designed for RGB-D data. The advantage of our approach is that the lossy compression is applied to the RGB image and lossless compression is applied to the depth frame which preserves the quality of the 3D data with only slight reduction in visual quality. Using very modest computing hardware, we were able to achieve 12 frames per



second (FPS) average transmission rate, which indicates that this can be used for real time applications.

II. Related Work

Since the release of the Microsoft Kinect and other low cost RGB-D sensors, applications for using these to build photo-realistic and accurate 3D models have exploded. Some of the state-of-the-art research in this area is presented in these publications [2-6]. Although these approaches produce great results, they require a significant amount of computational power. For example the author in [5] reported that their system use an Intel Core i7-2600 3.4GHz CPU, 8GB DDR 1333MHz RAM and an nVidia GeForce GTX 560 Ti 2GB GPU.

Our own work in the area focuses on how this technology can be applied to USAR operations. The work spans from mounting RGB-D sensors on a multi-rotor Unmanned Aerial Vehicle (UAV) [7] to mounting such sensors on search dogs [8], searching access holes [9], and creating a system that allows first responders to virtually interact with 3D models of rubble [10]. From our research and others in this area, it is apparent that RGB-D data holds the promise of greatly improve the situational awareness of first responders in USAR operations.

While there is an abundance of work on creating point cloud models and using them, research on the transmission of this data has rarely been mentioned in the literature. The closest related work to is [1]. Their approach is completely different from ours. Their assumption is that the point cloud is computed onboard (on the robot), then once the point cloud (or parts of the complete point cloud) is generated, lossy compression is performed on the point cloud prior to transmission over a network. The paper reported a high compression ratio but their algorithm utilizes a lossy scheme, hence the compression ratio scales with the loss of precision. Additionally the paper does not report on the computational power requirements of their algorithm. We suspect that it might be high if the robot is performing both registration and compression onboard.

With the consideration of USAR operations where the robot's payload has weight limitations, especially on UAVs, low-power computing hardware is a high priority for us. It is also highly desirable to preserve the full detail of the spatial data as this data provides the additional potential for situational awareness that is important to first responders.

III. Technical Approach

One of the assumptions of our approach is that the robot is equipped with an RGB-D sensor and we want to stream only the RGB-D data. There are several advantages to this approach in comparison to the one presented in [1]. The first is that RGB-D data is smaller than point cloud data. A point in the point cloud is represented by 3 floating point numbers (*floats)*, one for each dimension, X, Y, and Z for a total of 12 bytes (4 bytes per *float*). Although the colour data is not mentioned in [1], if colour is being transmitted as well then another 3 bytes is needed—one for each colour channel—for a total of 15 bytes. RGB-D data requires only 5 bytes per pixel, 3 bytes for

RGB and a single depth value encoded by an *unsigned short* (2 bytes).

The second advantage to sending RGB-D data instead of point cloud data is that it eliminates the need to compute the point cloud onboard. The computing power of the hardware is directly related to its size and weight, which is part of the payload of the robot. Therefore it is advantageous to minimize the computing power required as UAVs have limited payload capacity. Our approach off-loads the heavy computational tasks to the receiver which has no weight restrictions and hence no limitation on computing power.

Finally, using RGB-D data allows us to separate each frame into discrete image and depth data, then compress each sequence of data separately. The RGB image data is only needed as a visual aid so a small loss of precision is acceptable. To the human eye, a compressed RGB image is hardly noticeable from the raw bit map image. The depth data however will be used to generate a point cloud of the disaster environment where precision is very important. We design our framework based on these underlying reasons. A block diagram of our framework is displayed in Fig. 2.

The framework is based on a client/server model where the computer on the UAV is the server and the computer at ground-base is the client. The server isolates the sequences of RGB image and depth data and compresses them separately, using a lossy algorithm for the image data and a lossless algorithm for the depth data. Next, the algorithm puts together a data packet consisting of four parts: 1) length of the compressed image data, 2) compressed image data, 3) length of the compressed depth data, 4) compressed depth data. Each frame is then sent over the network socket using the standard TCP/IP protocol.

The client performs the opposite tasks. Upon receiving a data packet, it separates the RGB and depth data and decompresses them separately. The output is a compressed image and a fully preserved depth frame.

A main feature of the framework is that the compression algorithms are modular. For the lossy image compression, we chose JPEG for the library as it was readily available and empirical tests showed that the image quality is comparable to the original images. We experimented with three different lossless compression algorithms further discussed in section IV. These compression algorithms can easily be swapped out for others depending on the hardware configuration available on the robot. Since the overall goal is transfer speed, there must be a balance between the computing complexity of the algorithm and the compression ratio.

IV. Experiment

To validate our hypothesis, we implemented the proposed framework to test its performance. We developed client and server applications used to capture RGB-D data, compress it and stream it over a network. We wanted to test various compression algorithms since there is a tradeoff between compression ratio and speed. Most importantly, we wanted to test our system against different hardware configurations employing the same type of hardware that would be used in the field.

A. Client and server applications

The first step in our experiment was developing a software framework to implement our compression scheme. Using the C++ language and OpenCV and OpenNI frameworks, we developed a threaded server and client application for data streaming. The server application used the OpenNI framework to interface with a PrimeSense range camera; in our case, an Asus Xtion Pro. We used OpenCV methods to capture individual snapshots of video (RGB) and depth (D) data, which were stored directly in memory (instead of writing to disk) to improve speed. Next, we compressed both RGB and depth data using an approach described in detail in the next section. Once compression was complete, each frame was transmitted to the client application using a standard socket connection. The client application read the frame from the network socket, separately decompressing the RGB and depth data, then displaying the visual representation in a window on a display.

B. Compression Scheme

Of particular importance was a compression scheme for RGB-D data, and determining the optimal algorithms for network streaming.

For RGB data, we quickly determined that JPEG compression offered an ideal compromise of speed, compression ratio, and ease of implementation. It was far more important to focus our efforts on compressing the depth data. Developing our own algorithm was outside the scope of this project, and existing work on depth data compression is not yet sufficiently developed to implement here, so we decided to work with popular and freely available lossless algorithms implemented in C++.

We chose three separate algorithms with different performance characteristics. First, the *bzip2* algorithm aims for maximum compression with slower speed. A second algorithm developed by Google, *snappy*, aims for maximum speed with less compression. Finally the *zlib* algorithm aims for a middle ground between speed and compression. To test, we compressed 500 different depth frames with each algorithm and calculate the mean speed and compression ratio for each.

C. Hardware Configurations

Having established the basic software framework, we needed to test different hardware configurations. Comparing these compression algorithms on a variety of hardware configurations allowed us to determine the optimal algorithm for depth data, and also learn where performance bottlenecks come up. We worked with three separate computer systems, again with different performance characteristics:

1. The fit-PC2 is the current computer mounted on our Hexacopter UAV as a standard test configuration. It uses an Intel Atom Z550 Silverthorne (2.00 GHz) processor, which is old and relatively slow.

2. The Lenovo Thinkpad x100e is a small laptop using an AMD Athlon Neo 1.60 GHz processor. Comparable to the fit-PC2, this laptop is also old and slow.
3. A Fujitsu Lifebook TH700 laptop using an Intel core i3 370M 2.4GHz processor, which is considerably more powerful than the other two systems.

We experimented with three different configurations: 1) slow server with slow client; 2) slow server with fast client; 3) fast server with slow client. In configuration 1, we used the fit-PC2 as the streaming server and the Thinkpad x100e as the streaming client. Next we tried upgrading the client to the faster Fujitsu laptop with the same fit-PC2 as configuration 1: this allowed us to determine if decompression speed is a bottleneck. Lastly, we upgraded the server system to the faster Fujitsu laptop, to test if compression speed was a bottleneck. In each test we measured the performance by streaming a total of 300 RGB-D frames and measuring the output frame rate.

As mentioned before, the computing hardware was deliberately chosen to be suitable for field robots (ground and aerial). Although the form factor of the laptops may not be suitable to be mounted on robots, the processor inside these laptops has similar performance to Intel Nuc mini fanless PC systems. In fact, the slowest Intel Nuc is actually faster than the 4 year-old core i3 processor on the TH700. As a proof of concept, the hardware chosen produced encouraging results and are shown in the next section.

D. Network Configurations

Disaster situations provide a challenging environment for networking since we cannot assume there is any existing wireless or wired infrastructure; moreover there are considerable obstacles and other signal interference. It was important to test our algorithms using a wireless configuration that could easily be deployed in such a situation. We chose to configure the onboard UAV computer server as an 802.11 wireless hotspot, so that clients would be able to connect directly without needing to set up a dedicated access point.

To configure a computer system as a wireless access point, the network interface card must support AP networking mode; also, an external antenna is desirable to increase signal range. The wireless interface card in the fit-PC2 (Ralink RT3090) supports both. Using the *hostapd* and *udhcpd* software packages, we set up the UAV system as a mobile 802.11 wireless access point. This allowed client computers to connect and stream data using COTS computers and Wi-Fi devices. We used this configuration for all of our testing, keeping it consistent for all computing and hardware configurations.

V. Results

In our first experiment, we compared the speed and compression ratios of bzip2, zlib and snappy algorithms by compressing a set of 500 depth frames. Shown in Table I is the results for the depth compression evulations where the average depth data compression ratio is calculated by dividing the average compressed depth frame size (bytes) by the size of an uncompressed depth frame. The size of the uncomprssed depth frame is always fixed at 614,400 bytes. Next, we wanted to see the overall compression performance of our hybrid lossless and lossy approach. The results are shown in Table II.

TABLE I. Depth Compression Results

	bzip2	***zlib***	***snappy***
Uncompressed depth frame size (bytes)	614,400	614,400	614,400
Average compressed depth frame size (bytes)	56,208	82,697	165,294
Average depth datacompression ratio	9.31%	13.46%	26.90%

TABLE II. Lossless and Lossy Compression Results

	bzip2 + JPEG	***Zlib+ JPEG***	***Snappy + JPEG***
Uncompressed RGB + depth frame size (bytes)	1,536,000	1,536,000	1,536,000
Average compressed RGB + depth frame size (bytes)	70,089	96,578	179,175
Average depth + RBB datacompression ratio	4.6%	6.3%	11.7%

It should also be noted that on the fit-PC2 platform, the average single frame compression speed for bzip2 was 0.26 seconds, zlib was 0.09 seconds and snappy was 0.008 seconds. These average times are specific to the hardware, although they are interesting relative to each other, demonstrating that zlib is almost three times faster than bzip2, and snappy is over ten times faster than zlib. These results confirm the benchmarks we reported earlier: bzip2 has excellent compression characteristics, but is far too slow to be usable for video streaming. Snappy has poor compression, but its high speed is very attractive. Finally, zlib achieves good compression at good speeds, which may or may not be more useful than snappy.

Next we tested these compression algorithms using the hardware configurations mentioned previously. In these experiments, we are not concerned with specific details about compression factor or network speed; we only care about the final output frame rate of the RGB-D stream.

In our first hardware configuration, both server and client systems were running on slow, older hardware. This allowed us to establish a baseline against which we could measure performance increases from more modern, faster processors. In each case we set up a wireless stream and measured the average frame rate after capturing 300 RGB-D frames.

Perhaps not surprisingly, on systems with low compute power the snappy algorithm provides best overall performance. We have yet to determine however how much network bandwidth is a limiting factor relative to processing power. Compression algorithms can also have very different characteristics for compression speed and decompression speed, so in the following experiments we upgraded the server system and later the client system to a faster processor, in order to determine where the best performance gains could be realized.

TABLE III. HARDWARE CONFIGURATION 1

	Average frame rate (fps)
bzip2	3.41
Zlib	3.56
snappy	5.48
uncompressed	2.48

In a second hardware configuration, we upgraded the client system to the Fujitsu laptop with i3 processor. This would determine how much the decompression speed is a bottleneck. A similar wireless test of 300 frames produced the following results:

TABLE IV. HARDWARE CONFIGURATION 2

	Average frame rate (fps)
bzip2	2.07
Zlib	3.94
Snappy	5.90
Uncompressed	2.40

Clearly, this experiment yielded very similar performance to the previous experiment. It would seem that decompression speed has very little impact on the overall performance of the system.

In our third hardware configuration, we replaced the server system with the Fujitsu laptop and i3 processor. This would determine how much the initial compression speed is a bottleneck. We reverted to using the slower Thinkpad laptop for the client system.

TABLE V. HARDWARE CONFIGURATION 3

	Average frame rate (fps)
bzip2	5.97
zlib	12.09
snappy	7.43
uncompressed	1.00

The results of this third experiment demonstrate that improving the speed of the server system (which is responsible for initial compression of the depth data) resulted in a significant improvement in overall performance. Most notably, the zlib library outputs at a frame rate over 300% faster than previous experiments. The bzip2 algorithm also showed a significant improvement over previous frame rates, however these were considerably lower to begin with so the improved rate is still not as useful. Finally, snappy also demonstrated some improvement but nowhere near as much as zlib. This suggests that snappy was now being bottlenecked by network bandwidth instead of compute power, and that adding increased computing resources would only result in very minor performance improvements.

VI. DISCUSSION AND FUTURE WORK

These results demonstrate that RGB-D streaming is indeed possible and useful on existing hardware. We were able to achieve frame rates of 12 fps using the more powerful i3 processor. This particular system cannot be reasonably mounted on a UAV, however current fanless mini PC systems incorporate processors as advanced as the Intel i7, which could be used to achieve higher streaming rates from aerial systems.

The focus of this paper has been on our framework and how the computing hardware affects the overall streaming performance. One variable in the system that remained constant was the communication configuration. All communication in our tests was performed over 802.11n in short range, line-of-sight situations. Designing an optimum wireless configuration for USAR is a challenging area that we would like to address in future works. Additionally, we also would like to test which compression schemes would be optimal on a tethered robot equipped with a high bandwidth network connection.

Another highly important test will involve writing the streamed RGB-D data to disk, and determining if we can get useful environmental data from this configuration in a timely manner. In theory this should be possible since the depth compression is all lossless, but it is possible that lost frames or data corruption will impact the output data, especially in noisy communications environments.

REFERENCES

[1] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 778-785.

[2] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, 2011, pp. 127-136.

[3] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an RGB-D camera," *in International Symposium on Robotics Research (ISRR)*, 2011.

[4] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially Extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, 2012.

[5] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense RGB-D mapping," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 5724-5731.

[6] Whelan, M. Kaess, J. J. Leonard, and J. McDonald, "Deformation-based loop closure for large scale dense rgb-d slam," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 2013, pp. 548-555.

[7] A. Ferworn, J. Tran, A. Ufkes, A. D'Souza, "Initial experiments on 3D modeling of complex disaster environments using unmanned aerial vehicles," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR).*, pp. 167-171, 2011.

[8] J. Tran, A. Ufkes, A. Ferworn, and M. Fiala, "3D Disaster Scene Reconstruction Using a Canine-Mounted RGB-D Sensor," in *Computer and Robot Vision (CRV), 2013 International Conference on*, 2013, pp. 23-28.

[9] C. Kong, A. Ferworn, J. Tran, S. Herman, E. Coleshill, and K. G. Derpanis, "Toward the automatic detection of access holes in disaster rubble," *in Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, 2013, pp. 1-6.

[10] A. Ferworn, S. Herman, J. Tran, A. Ufkes, and R. Mcdonald, "Disaster scene reconstruction: Modeling and simulating urban building collapse rubble within a game engine," in *Proceedings of the 2013 Summer Computer Simulation Conference*, 2013, p. 18.