# Lossless Compression of Weather Radar Data Using Open-Source Software Implementations

Jesse R. Hairston, W. David Pan
Dept. of Electrical & Computer Engineering
University of Alabama in Huntsville
Huntsville, AL 35899, USA
Email: jrh0038@uah.edu, dwpan@eng.uah.edu

## I. Introduction

To address the challenge of detecting hazardous weather conditions that can adversely impact naval operations, a meteorological radar data assimilation system has been developed at the U.S. Naval Research Laboratory (NRL), which requires weather radar data to be sent from ships to Navy's data fusion center in Monterey, CA. However, the amount of radar data to be transmitted from ships to shore was found to be so large a load on the operational bandwidth that efficient data compression is critical. The second author's prior research through collaborations with NRL scientists has resulted in the development of a data-compression software package called UFZIP that has been transitioned into field operations [1][2]. However, the next generation radar assimilation system requires much increased data volume, thereby demanding more aggressive data compression than what can be achieved by currently available capability. To address this technological challenge, we propose to investigate the feasibility of adapting existing open-source compression techniques developed originally for image and video to achieve high compression on radar data, by exploiting both spatial and temporal redundancies. This research was supported by the Research and Creative Experiences for Undergraduates (RCEU) program at the University of Alabama in Huntsville. The results reported in this paper stem from the research performed by the first author in the summer of 2013, under the guidance of the faculty advisor (second author).

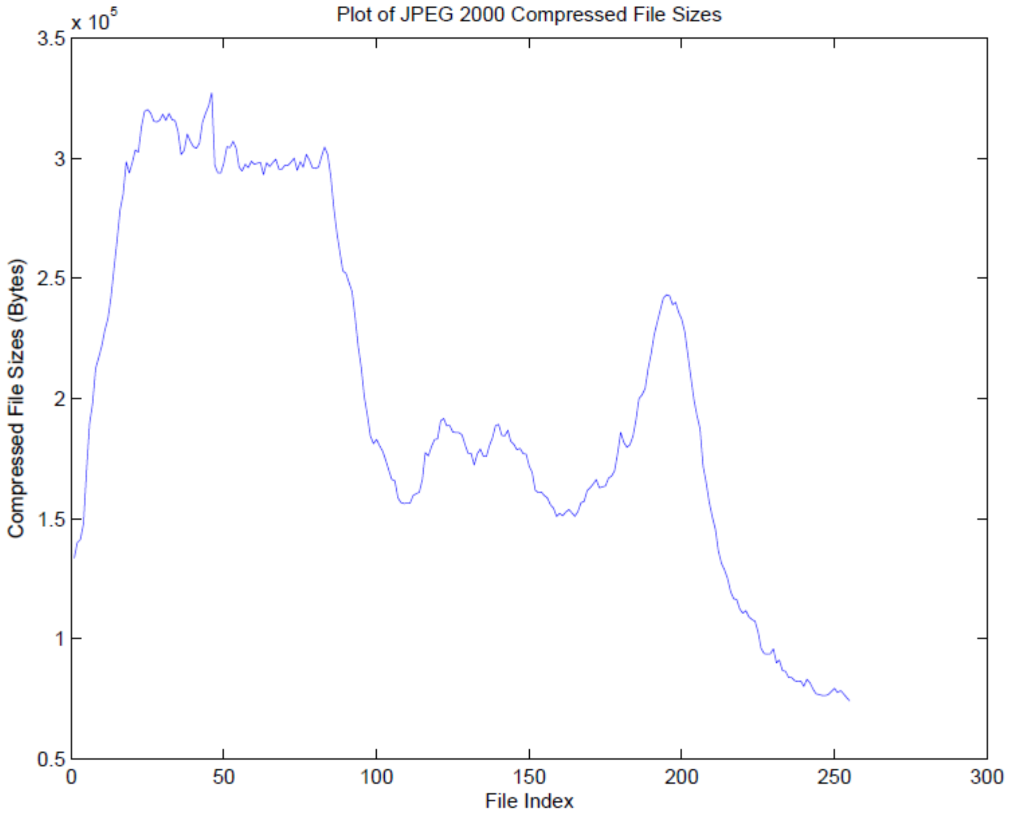
## II. Searching for Efficient Compression

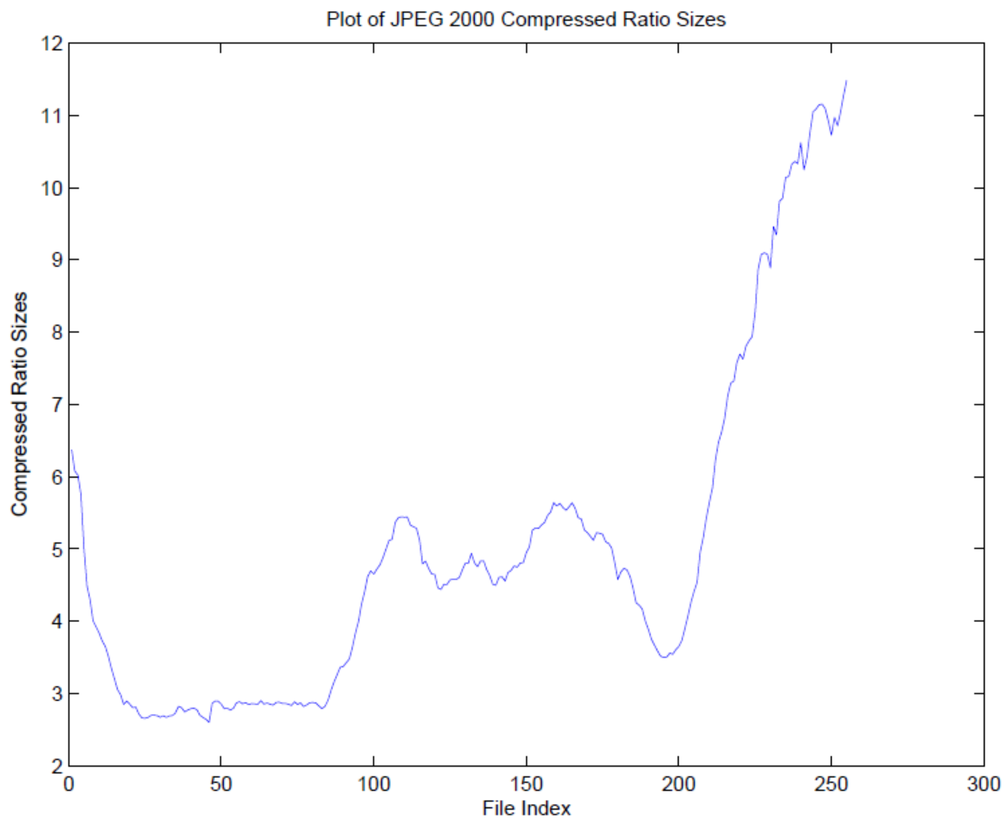
Anticipating the beginning of summer research, examination began on potential compression standards and algorithms that would be used for research. The spring of 2013 was spent researching general knowledge behind the JPEG 2000, Motion JPEG 2000, and H.264 compression standards [3]. Although each of the three standards has its own unique feature or form of optimization, there is one important characteristic supported by all – lossless compression. The technique of lossless compression, as the name implies, ensures that absolutely no data is sacrificed in the compression process (conversely, lossy compression implies that at least some data is lost by compression). Lossless compression allows a user to encode data into a smaller form, then decode the compressed data into an identical copy of the original data. Such a riskless technique is invaluable when dealing with weather radar data. When human lives could be at the mercy of severe weather conditions, compressed data must not only be transmitted quickly, but also decoded accurately for quick interpretation. In the following, the compression ratio of a lossless compressor is defined as: $\mathit{Compression\ Ratio} = \frac{\mathit{Original\ Size}}{\mathit{Compressed\ Size}}$. In the following study, a set of temporally sequential weather-radar reflectivity data files was converted to a collection of grayscale images in PGM format with a resulting total size of approximately 217 MB.

## III. JPEG 2000: A Standard for High Complexity Imagery

Based on wavelet decomposition, the JPEG 2000 standard was created by the Joint Photographic Experts Group, also known for being the creators of the common JPEG standard [4]. For our purposes, an attempt was made to losslessly compress a grayscale weather radar data set by implementing the JPEG 2000 standard; however, open-source software implementations had to be located first. Two separate implementations were chosen to be explored, each of which contains its own encoder and decoder: OpenJPEG and JasPer. The source code for OpenJPEG (developed by the Communications and Remote Sensing Lab of UCL) was located at the open-source project's primary website (http://www.openjpeg.org/). Written in the C programming language, the OpenJPEG executables were easily compiled in Microsoft Visual Studio. JasPer, developed by a joint-effort between Image Power, Inc. and the University of British Columbia, was also designed as an open-source project – the source code was located at the project's primary website (http://www.ece.uvic.ca/~frodo/jasper/), and also compiled in Microsoft Visual Studio. Initial tests showed both OpenJPEG and JasPer losslessly compress and decompress a BMP image file into the exact same size, hinting that the two implementations were quite similar; however, this could not be tested further, as the set of 255 grayscale weather radar reflectivity images generated for JPEG 2000 compression could not be processed by OpenJPEG (the data set file type is known to be compatible with OpenJPEG – potentially, the problem could be the method used for generating the data set). Although we could not use OpenJPEG to compress the original data, the software will still prove to be useful, as shown in the Motion JPEG 2000 section of this report.

Solely using the JasPer codec, the following results were obtained from compressing the 255 grayscale weather radar images. The total size of all original files is ≈$2.17 \times 10^8$. The total size of all compressed files is ≈$5.175 \times 10^7$. The total size of all decompressed file is ≈$2.17 \times 10^8$. A text file was generated, displaying the compressed size of each image within the data set (the smallest size being 74,115 bytes, and the largest size being 327,163 bytes). Another text file was created with details of the ratio of original file / compressed counterpart (the smallest ratio ≈2.6, and the largest ratio being ≈11.48). The log declaring each individual compression as lossy or lossless confirmed that JasPer compressed entirely losslessly. Corresponding graphs for the compressed file sizes and compressed ratio sizes were also generated (depicted below and on the following page).



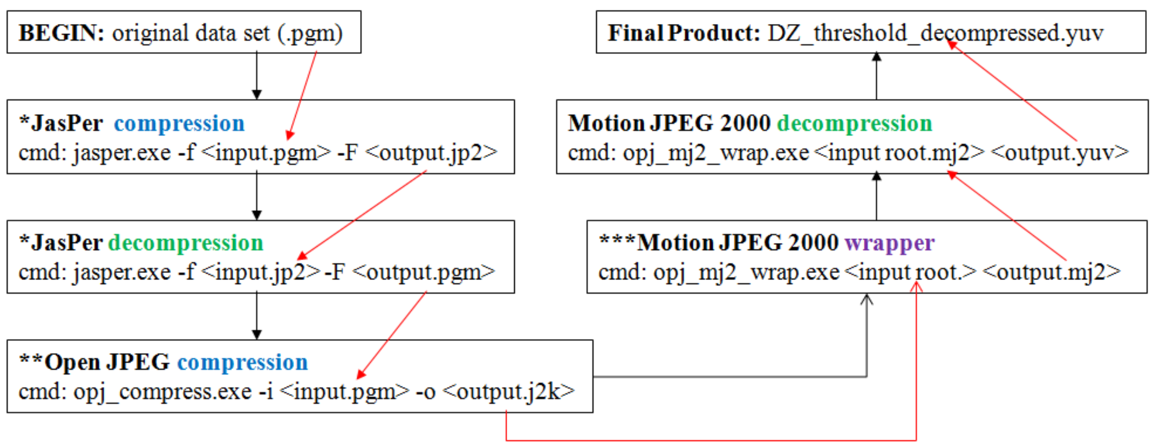


The ratio of total compression achieved with JasPer is 4.193, meaning 76.15% of space is saved. Next, we will explore other methods of compression to attempt to achieve a higher compression ratio than 4.193 for the grayscale data set.

## IV. Motion JPEG 2000: A Concatenation of Images

Motion JPEG 2000 (MJ2) is a standard of JPEG 2000 supported by the OpenJPEG implementation (http://code.google.com/p/openjpeg/wiki/DocMJ2K) The general idea behind MJ2 is to concatenate all images within a sequenced data set, then compress the concatenation as if it were a video sequence – perhaps then it would be possible to obtain a higher compression ratio than 4.193 for the grayscale data set. Unfortunately, there is one slight problem faced with such a feat: in order to compress a file using MJ2, the input file must be of the J2K file type. JasPer only has the capacity to compress files into the file type JP2. The only available way to obtain J2K files is to compress the grayscale data set using OpenJPEG (which, if you recall, results in an unknown error). Thankfully, a workaround for this problem was found. Through testing, it was discovered that compressing and decompressing a grayscale image with JasPer will allow OpenJPEG to compress the resulting decompressed JasPer PGM file, creating a J2K file. If we repeat this process for each image within the grayscale data set, we can use the MJ2 wrapper to concatenate the entire data set into a single, compressed MJ2 file. From there, by decompressing the wrapped file into a YUV file, the concatenated sequence may be viewable as if it were a decompressed video. Since this is somewhat a convoluted process, a flowchart is attached below to promote clarity.

Creating aYUV from a PGM Data Set


**BEGIN:** original data set (.pgm)
*JasPer compression
cmd: jasper.exe -f <input.pgm> -F <output.jp2>
*JasPer decompression
cmd: jasper.exe -f <input.jp2> -F <output.pgm>
**Open JPEG compression
cmd: opj_compress.exe -i <input.pgm> -o <output.j2k>
***Motion JPEG 2000 wrapper
cmd: opj_mj2_wrap.exe <input root.> <output.mj2>
Motion JPEG 2000 decompression
cmd: opj_mj2_wrap.exe <input root.mj2> <output.yuv>
**Final Product:** DZ_threshold_decompressed.yuv

*- JasPer compression/decompression is implemented to decode the original data set files so that they can be compressed with Open JPEG in the .j2k format.
**- when executing opj_compress.exe, sequentially save the output.j2k files in the following format via script: "compressed_00000.j2k", "compressed_00001.j2k", "compressed_00002.j2k", etc. The MJ2 wrapper will only wrap .j2k files of this ordered format.
***- the <input root> contains the path & name of the input files before its 5 digit number. For an example, see MJ2 wrapper documentation (http://code.google.com/p/openjpeg/wiki/DocMJ2K).

The above process will result in a total compression of $\approx 5.17 \times 10^7$ – unfortunately, that value is roughly the same as compression value of JasPer. This experiment marks MJ2 at a slightly higher compression ratio of ≈4.195 (76.16% of space saved) – a ratio so insignificantly higher that the MJ2 process may not be worth the effort for the sole purpose of compression. However, compressing with MJ2 allows the option to concatenate sequence images. One fascinating aspect of decompressing the MJ2 file into a YUV is the ability to view the data set as if it were a video. On the topic of video, the potential behind using a video compression standard will be explored to obtain higher compression on the grayscale data set.

## V. H.264: A Standard for High Quality Video Compression

The H.264 video compression standard was created by the Joint Video Team, a group partnership between the ITU-T Video Coding Experts Group and the ISO/IEC JTC1 Moving Picture Experts Group [5]. It is renowned for its ability to compress high definition video efficiently. The search for open-source codecs led to the x264 encoder (currently developed by Loren Merritt et al.) (http://www.videolan.org/developers/x264.html) and the FFmpeg decoder (maintained by Michael Niedermayer). The source code for both implementations was located on FFmpeg's primary website (http://www.ffmpeg.org/) . Following one of the website's many helpful tutorials, both x264 and FFmpeg were easily compiled on the Ubuntu distribution of Linux (note: H.264 is the only standard compiled in Linux – everything else was compiled in Windows).

Compressing with x264 actually requires the result of our work from the previous stage. The YUV file, which contains each of the 255 grayscale images, is the file which we will be compressing. From the gratuitous amount of video encoding options offered by x264, the YUV file was compressed into an MP4 file, as VLC player could be used to ensure that the video would display correctly. By comparing the byte size of the original YUV to the compressed MP4, the following is observed: the original byte size $\approx 3.26 \times 10^8$, while the compressed byte size $\approx 3.15 \times 10^7$. When the data compression ratio is calculated, the ratio is determined to be approximately 10.349. Incredibly, using the H.264 more than doubled the compression ratio when compared with the results from JPEG 2000; however, there is something to note. For H.264 compression to be fairly compared with JasPer and MJ2 compression, the uncompressed byte size for the compression ratio must be measured using the original data set file size (217,019,280). Making the proper adjustments, H.264 still bears a compression ratio of 6.900 (saving 85.51% space). Even though H.264 boasts the most efficient compression method for the 255 grayscale images, it is important to note that in the case of this research, H.264 compressed the YUV file created by MJ2 (which, in turn, was obtained using JasPer and OpenJPEG). H.264's ability to achieve such a feat is likely due to its ability to exploit temporal correlations between adjacent data files.

The results with the H.264 standard were proven to be lossless by implementing a script in MATLAB. The script loads both the pre-H.264 and post-H.264 YUV files. Within the code, the luminance components, as well as the succeeding chrominance components, are set for a single frame (recall, 255 total). The function then loops, pointing to each individual corresponding frame within each YUV file and comparing them for losslessness. The results are printed to a text file where the user can verify lossy or lossless compression.

## VI. Conclusion

In summary, the H.264 video compression standard seems to be the most efficient form of compression. Future research could include an examination of the computational complexity of H.264 to determine ways to reduce its computational load. Either way, all roads seem to lead to H.264 in regards of a compression standard that offers robust features, functionality, and high ratios of compression.

## VII. Acknowledgements

The research conducted within the document was done with the intention to promote further research on compressing weather radar data. The test images were derived from the data provided by the United States Naval Research Laboratory, Monterey, CA.

A special thank you is extended to those who are responsible for both supporting and generously funding the Research and Creative Experiences for Undergraduates (RCEU) program at the University of Alabama in Huntsville (UAH), including: the UAH President/Provosts Office, the UAH Vice President of Research, the UAH Chemistry Department, and the Alabama Space Grant Consortium. The research referenced within this document would not have been possible without their contributions.

**References:**

[1] W. D. Pan, P. R. Harasti, M. Frost, Q. Zhao, J. Cook, T. Maese, and L. J. Wagner, "Efficient Reduction and Compression of Weather Radar Data in Universal Format," Proc. of *AMS 25th Conference on International Interactive Information and Processing Systems (IIPS'09) for Meteorology, Oceanography, and Hydrology*, Phoenix, AZ, January 2009.

[2] W. D. Pan, P. R. Harasti, M. Frost, Q. Zhao, J. Cook, T. Maese, Lee J. Wagner, Claude P. Hattan, and Bryan T. Akagi, "A new method for compressing quality-controlled weather radar data by exploiting blankout markers due to thresholding operations," in Proc. of *AMS 27th Conference on International Interactive Information and Processing Systems (IIPS'11) for Meteorology, Oceanography, and Hydrology*, Seattle, Washington, January 2011.

[3] Sayood, Khalid. *Introduction to Data Compression*. 4th ed. Boston: Morgan Kaufmann, 2012.

[4] Salomon, David. *Data Compression: The Complete Reference*. 3rd ed. New York: Springer, 2004.

[5] Prasantha, H, H Shashidhara, K Murthy, and M Venkatesh. "Performance Evaluation of H.264 Decoder on Different Processors." *International Journal on Computer Science & Engineering*. 1.5 (2010): 1768. Web. 7 Apr. 2013. <http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=55094975&site=ehost-live>.