On the Usage of General-Purpose Compression Techniques for the Optimization of Inter-Robot Communication*

Gonçalo S. Martins, David Portugal, and Rui P. Rocha**

Institute of Systems and Robotics, University of Coimbra
3030-790, Coimbra, Portugal
gmartins@isr.uc.pt,davidbsp@citard-serv.com,rprocha@isr.uc.pt
http://ap.isr.uc.pt/

Abstract. Managing the bandwidth requirements of a team of robots operating cooperatively is an ubiquitous and commonly overlooked problem, despite being a crucial issue in the successful deployment of robotic teams. As the team's size grows, its bandwidth requirements can easily rise to unsustainable levels. On the other hand, general-purpose compression techniques are commonly used to transmit data through constrained communication channels, and may offer a solution to this problem.
In this paper, we study the possibility of using general-purpose compression techniques to improve the efficiency of inter-robot communication, firstly by comparing the performance of various compression techniques in the context a of multi-robot simultaneous localization and mapping (SLAM) scenarios using simplified occupancy grids, and secondly by performing tests with one of the compression techniques on real-world data.

Keywords: Compression Methods, Multi-Robot Systems, Efficient Information Sharing.

1 Introduction

Cooperation among mobile robots almost always involves interaction via explicit communication, usually through the use of a wireless network. Commonly, this network is taken for granted and little care is taken in minimizing the amount of data that flows through it, namely to assist the robot's navigation across the environment.

However, in real-world applications, the navigation effort can be but a small part of the tasks that must be dealt with by a complete robotic system [17].

* This work was supported by the CHOPIN research project (PTDC/EEA-CRO/119000/2010) and by the ISR-Institute of Systems and Robotics (project PEst-C/EEI/UI0048/2011), funded by the Portuguese science agency "Fundação para a Ciência e a Tecnologia" (FCT).

** The authors would like to acknowledge Eurico Pedrosa, Nuno Lau and Artur Pereira [14] for providing us with a software tool intended to adapt the raw sensor log files into a format readable by *ROS*.

Therefore, it should operate as efficiently as possible. Additionally, in harsher scenarios, such as search and rescue operations, constrained connectivity can become an issue, and caution must be taken to avoid overloading the network. An efficient model of communication is also a key element of a scalable implementation: as the number of robots sharing the network increases, the amount of data that needs to be communicated does as well. Thus, greater care in preparing data for transmission is needed, so as to avoid burdening the network by transmitting redundant or unnecessary data.

In this paper, we analyze the data transmitted by a team of robots on a cooperative mission that includes mapping and navigation. With this purpose, we use a multi-robot simultaneous localization and mapping (SLAM) task [13] as a case study of the exchange of information among robots, though the ideas proposed herein can be generalized to other cooperative tasks, at different abstraction levels. In our case study, mobile robots are required to communicate occupancy grids [6] among themselves, in order to obtain a global representation of the environment based on partial maps obtained locally by individual robots.

Occupancy grids are metric representations of the environment, being repetitive by nature [6]. In their simplest form, they consist of a matrix of cells, each representing a fraction of the robot's workspace, that are commonly in one of three states: free, occupied or unknown. These can be seen as the result of a "thresholding" operation applied to a more complex occupancy grid, which is composed of cells whose occupancy, instead of one of three values, is modeled through a probability value or a probability distribution [16].

In larger environments, or at greater resolutions, these simpler grids are usually stored as large matrices filled with only three different values, often containing very long chains of repeated cells. Keeping this data in memory in this form is a sensible approach. The data is very easily accessible, with little computational overhead. However, transmitting it in this form is most likely a wasteful use of bandwidth.

Compression methods are widely used in the transmission and storage of bulky data, such as large numbers of small files, logs, sound and video. Compression is even being used by default in specific file systems, offering a possible solution for this problem. These exploit the data's inherent *compressibility* in order to represent it using fewer bits of data than originally.

In this paper, we present a novel compression benchmarking tool and metric, as well as results and discussion of a series of experiments on the compression and decompression of occupancy grids, as a case study for the application of compression techniques in multi-robot coordinated tasks.

In the following pages, various general-purpose, lossless compression techniques are analyzed and compared, in an effort to determine which, if any, is more suitable as a solution to the large bandwidth requirements of multi-robot systems. We will start by presenting a review of previous work in efficient communication between coordinated robots, followed by a short presentation of the various techniques being compared. We then present and discuss our benchmark-

ing results, as well as preliminary results obtained by operating on real-world data. We summarily conclude by taking an outlook into future work.

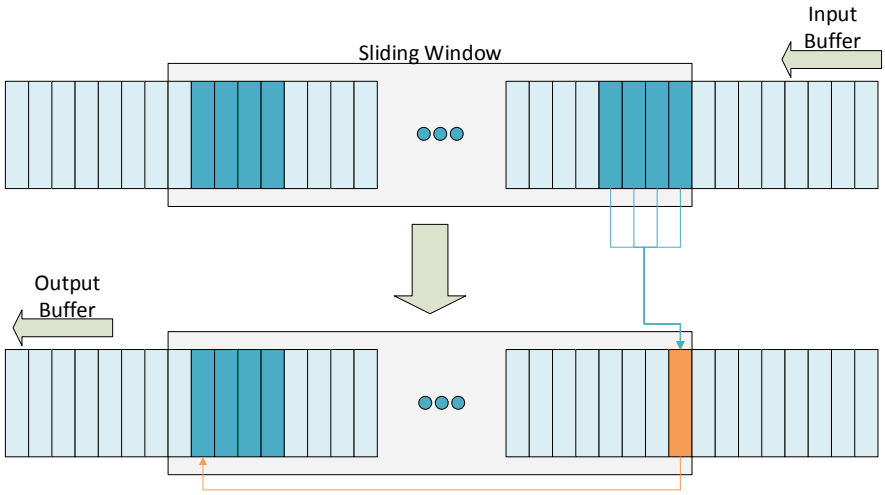
1.1 Related Work

Data compression is a process through which we aim to represent a given piece of digital data using fewer bytes than the original data, and can be seen as a way of trading excess CPU time for reduced transmission and storage requirements. Compression methods are divided into two main groups: *lossless* methods, which make it possible to reconstruct the original data without error; and *lossy* methods, which make use of the way humans perceive signals to discard irrelevant data.

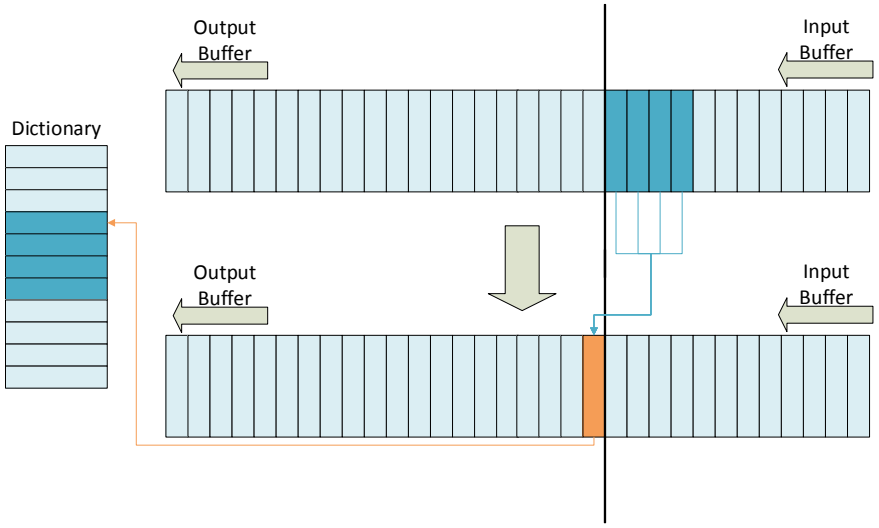
Lossy compression algorithms are commonly used in the compression of signals intended for human perception, such as image and sound. These techniques usually make use of the way we perceive signals to reduce their size [19]. For example, given that the human hearing's capability ranges from about 20Hz to about 20kHz, sound compression techniques can remove any signal components outside that frequency range. Although the compressed data should be significantly smaller than the original, humans hearing sound reconstructed from lossy compressed data should experience much the same. However, the original signal cannot be recovered.

Lossless compression, on the other hand, compresses data in a way that it is later fully recoverable. In 1977 [20] and 1978 [21], Abraham Lempel and Jacob Ziv developed two closely related algorithms which were to become the basis for most of the lossless, general-purpose compression algorithms currently in use. LZ77 and LZ78, as their works were to become known, are methods of dictionary-based lossless compression. Summarily, the LZ77 and LZ78 algorithms keep a *dictionary* of byte chains encountered throughout the uncompressed data, and replace repetitions of those chains with *links* to entries in the dictionary, thus reducing the size of the data.

LZ77 compresses data by running a sliding window of a given fixed length over the input data, which is composed of variable-length sequences of bytes. For each input sequence, the algorithm looks for matches between the current sequence and a previous occurrence inside the sliding window. When a match is found, the repeated sequence is replaced by an offset and a length, which represent location of the previous occurrence in the sliding window, and the length of the repetition. For example, if the string "abc" existed twice in the window, the second occurrence would be replaced by an offset that pointed to the beginning of the string, and a length of three characters. This simple concept is the basis of dictionary coding. Furthermore, LZ77 has a way of dealing with very long repetitions, by specifying a length that is longer than the source string. This way, when decoding, the source string is copied multiple times into the output buffer, correctly rebuilding the repetition. For example, if the string "abc" exists somewhere in the sliding window, and the string "abcabc" exists somewhere after it, the second string would be replaced by an offset that pointed to the letter 'a' in the first string, and a length of six characters, instead of the length of three



(a) LZ77 operates by running a sliding window over the data. When a sequence in the input data is matched to data that is still inside the window, it is replaced with an offset-length pair that points to the previous instance of that data. In this figure, the dark blue segments were matched, and the second one is replaced with the orange, smaller segment, that points to the first copy of the matched segment.



(b) LZ78 operates by building an explicit dictionary. As the input data is consumed, the algorithm attempts to match each input sequence with an existing sequence in the dictionary. If the matching operation fails, the new data is added to the dictionary. This illustration shows the case where a match is found. In that case, the dark blue segments are matched to an entry in the dictionary, and replaced in the output buffer with the orange, shorter segment that points to the correct entry in the dictionary.

Fig. 1. A simplified pictorial explanation of LZ77 and LZ78's operation.

characters one might have expected, thus encoding the whole six-letter string into a single offset-length pair. Once all the data is encoded, decoding it consists of reversing the process, by replacing every offset-length pair in the coded data by their corresponding byte chains.

Despite technically being a dictionary coder, LZ77 does not explicitly build a dictionary. Instead, it relies on offset-length pairs to elliminate repetition. LZ78, on the other hand, does create an explicit dictionary. The algorithm attempts to find a match in the dictionary for every sequence that is taken from the input buffer. If a match is not found, it is added to the dictionary. Every match that is found is replaced with a structure analogous to the offset-length pair described above, differing in the fact that now the offset represents an entry in the dictionary. The LZ78 dictionary is allowed to grow up to a given size, after which no additional entries are added, and input data that cannot be matched with any dictionary entries is output unmodified. Decoding LZ78-encoded data also consists of simply reversing the process, substituting each offset-length pair with the appropriate entry from the dictionary. The operation of these algorithms is illustrated in Fig. 1.

We have restricted our choice of algorithms to those based on Lempel and Ziv's work, for their focus on reducing redundancy by exploiting repetition, and for their *lossless* nature. It is important that the algorithms we are employing be fully lossless, *i.e.* that the compressed data can be used to reconstruct the original data, since we intend to generalize this technique to other types of data which may not tolerate any errors. For example, lossy image-based compression techniques, such as *JPEG*, could be used to reduce the size of an occupancy grid, processing it as an image. However, compression artifacts and other inaccuracies could lead to an erroneous representation of the environment, either by distorting its features or by hindering other aspects of the multi-robot mapping effort, such as occupancy grid image-based alignment and merging [3].

Efficient inter-robot communication is not an area devoid of research. Other works, such as [2], [13] and [4], have worked on a solution for this issue by creating new models of communication for robotic teams, *i.e.* by developing new ways of representing the data needed to accomplish the mission. Other research efforts focused on developing information utility metrics, *e.g.* by using information theory [16], which the robot can use to avoid transmitting information with a utility measure below a certain threshold. We could find none, however, that applied compression to further increase their optimization gains. These techniques, while successful in their intended purpose, rely on modifications to the inner workings of their respective approaches. In our case, we intend to create an optimization solution that is more general, and that does not depend on modifications to the intricacies of the underlying techniques.

Finally, there are several examples[1] of compression benchmarks. However, we found none that focus on the algorithms' ability to optimize inter-robot communication. Their main focus is on comparing the techniques' performance on

[1] Such as Squeeze Chart (`http://www.squeezechart.com/`) and Compression Ratings (`http://compressionratings.com/`).

the compression and decompression of standard datasets, such as long sections of text, random numbers, *etc.* The need to test these techniques in the compression of specific, Robotics-related datasets, as well as the need to do so in a methodical, unbiased way, compelled us to create our own solution.

2 Free and Open Source Software Data Compression Techniques

As stated previously, occupancy grids, while a practical way of keeping an environment's representation in memory, are cumbersome as transmission objects. At the typical size of 1 byte per cell, an 800-by-800 cell grid (*e.g.* a representation of a somewhat small 8-by-8 meter environment at 100 cells per meter) occupies 640 kilobytes of memory. Depending on how fast an updated representation is generated, and how many robots take part in the mapping effort, this can lead to the transmission of prohibitively large amounts of data. If we update that same grid once every three seconds on each robot, each robot will generate an average of about 213KB/s. For a relatively small team of three robots, that equates to generating 640KB per second of data that needs to be transmitted. This simple calculation does not take into account the possibility of one of the robots exploring the environment further away from the others, causing the grids to expand, which would further enlarge the amount of repetitive data generated.

If we assume that each robot has to transmit its map to each of the team members, in a client-server networking model, each map update carries a bandwidth cost of $C = S \times (n - 1)$, where C is the total cost, in bytes, S is the size of the map, in bytes, and n is the number of robots in the team. We can easily determine then that a regular 802.11g access point, operating at the typical average throughput of 22Mb (or 2.75MB) per second could support a team of 14 robots.

Given the redundancy that is naturally occurring in the data, there is great potential for optimization in the team's usage of bandwidth. Since data compression methods aim to remove redundancy from data, and can be applied to any type of data, they seem to be adequate candidates for network optimization.

LZ77 and LZ78 inspired multiple general-purpose lossless compression algorithms, widely used today as Free and Open Source Software (FOSS) implementations. We have collected the ones that we believe are the most suitable as solutions to our problem, given their availability, use and features. We will summarily discuss them next.

DEFLATE, presented in [5], is the algorithm behind many widely used compressed file formats such as *zip* and *gzip*, compressed image formats such as *PNG*, and lossless compression libraries such as *zlib*[2], which will be the implementation through which DEFLATE will be tested. This algorithm combines the LZ77 algorithm with Huffman Coding [9]. The data is first compressed using LZ77, and later encoded into a Huffman tree. Being widely used, this technique was one of the very first to be considered as a possible solution to this problem.

[2] *zlib* is available at `http://www.zlib.net/`.

LZMA[3], which stands for Lempel-Ziv-Markov Chain Algorithm, is used by the open-source compression tool *7-zip*. To test this algorithm, we used the reference implementation distributed as the *LZMA SDK*. No extensive specification for this compressed format seems to exist, other than its reference implementation. LZMA combines the sliding dictionary approach of LZ77 with range encoding.

LZ4[4] is an LZ77-based algorithm focused on compression and decompression speed. It has been integrated into the Linux kernel and is used on the BSD-licensed implementation of ZFS [18], OpenZFS, as well as other projects.

QuickLZ[5] is claimed to be "the world's fastest compression library". However, the benchmark results provided by its authors do not compare this technique to either LZ4 or LZMA, warranting it a place in our comparison.

Finally, Snappy[6], created by Google, is a lightweight LZ77-based compression library that aims at maximizing compression and decompression speed. As such, and unlike other techniques, it does not employ an entropy encoder like the Huffman Coding technique used in DEFLATE.

3 Benchmarking Methodology

Part of the motivation behind this work consists of the fact that compression benchmarking tools usually focus on either looking for the fastest technique, or for the one that achieves the highest compression ratio, as defined by:

$$R = \frac{L_U}{L_C}, \tag{1}$$

where R is the compression ratio, L_U is the size of the uncompressed data, and L_C is the size of the compressed data, both usually measured in bytes.

When choosing among a collection of compression techniques, compression ratio is a metric of capital importance, since the better the ratio, the less information the robots have to send and receive to complete their goal. However, the techniques' compression and decompression speeds are also important; an extremely slow, frequent compression may jeopardize mission-critical computations. Thus, we cannot simply find the technique that maximizes one of these measures; there is a need to define a new, more suitable performance metric, in order to find an acceptable trade-off.

Therefore, we define:

$$E = \frac{R}{T_c + T_d}, \tag{2}$$

in which E is the technique's *temporal efficiency*. It is determined by dividing the compression ratio achieved by the technique, R, by the total time needed

[3] The LZMA SDK used is available at `http://www.7-zip.org/sdk.html`.
[4] LZ4 is available at `http://code.google.com/p/lz4/`.
[5] QuickLZ is freely available for non-commercial purposes at `http://quicklz.com/`.
[6] Snappy is available at `https://code.google.com/p/snappy/`.

to compress and decompress the data, T_c and T_d, respectively. The purpose of this quantity is to provide an indication of how efficiently the technique at hand uses its computational time. The algorithm that achieves the highest temporal efficiency, while at the same time achieving acceptable compression ratio, is a strong candidate for integration in work that requires an efficient communication solution, provided that its absolute compression ratio is acceptable.

In order to test these techniques, the authors developed a benchmarking tool[7] that, given a number of compression techniques, runs them over occupancy grids generated by SLAM algorithms, outputting all the necessary data to a file. This tool allows us to both apply the techniques to the very specific type of data we wish to compress, as well as test them all in the same controlled environment. It was designed to be simple and easily extensible. As such, the addition of a new technique to the benchmark should be trivial for any programmer with basic experience.

To account for the randomness in program execution and interprocess interference inherent to modern computer operating systems, each algorithm was run over the data 100 times, so that we could extract results that were as isolated as possible from momentary phenomena, such as a processor usage peak, but that reflected the performance we could expect to obtain in real-world usage. Interprocess interference could have been eliminated by running test process in the highest priority. However, that does not constitute a real-world use case, and that methodology would provide results that could not be expected to occur during normal usage of the techniques. Results include the average and standard deviation of the compression and decompression times for each technique and dataset, as well as the compression ratio achieved for each case. These results can be seen textually in Table 1, or graphically in Figs. 3 and 4. Each technique was tested using their default, slowest and fastest modes, except for QuickLZ and Snappy, which only provide one mode of operation, and LZ4, which only provides a fast (default) and a slow, high compression mode.

All tests were run on a consumer-grade machine equipped with an Intel Core i7 M620 CPU, supported by 8 GB of RAM, under Ubuntu Linux 12.04.

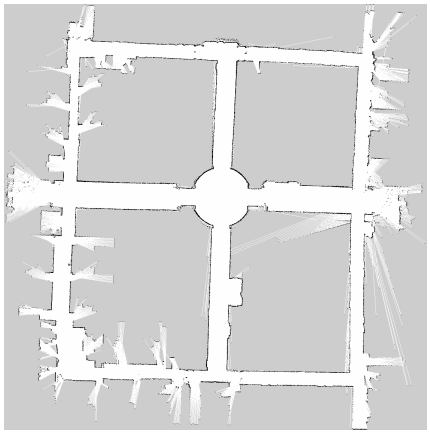
3.1 Datasets

In order to test the effectiveness of compression algorithms in treating typical occupancy grids, and given the intention of studying, at least to some degree, how each algorithm behaves depending on the dataset's size, five grids of different environments were chosen: Intel's Research Lab in Seattle; the ACES building, in Austin; MIT's CSAIL building and, finally, MIT's Killian Court, rendered in two different resolutions, so that differing sizes were obtained. These datasets are illustrated in Fig. 2. The occupancy grids we present were obtained from raw sensor logs using the *gmapping*[8] [8] SLAM algorithm, running on the

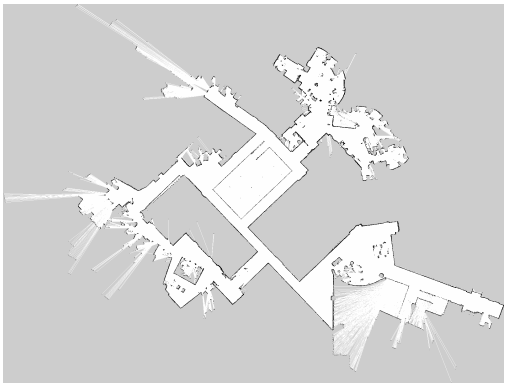
[7] The tool is publicly available under the BSD license at `https://github.com/gondsm/mrgs_compression_benchmark`.

[8] A description of the gmapping package can be found at `http://wiki.ros.org/slam_gmapping`.

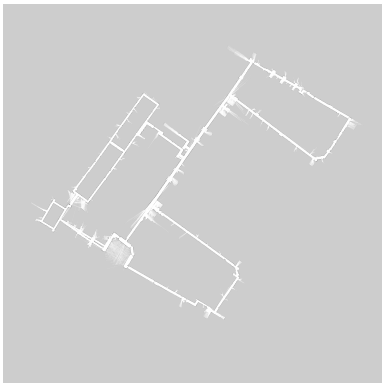
(a) Intel's Research Lab, measuring 753,078 bytes uncompressed.



(b) ACES Building, measuring 1,280,342 bytes uncompressed.



(c) MIT CSAIL Building, measuring 1,929,232 bytes uncompressed.



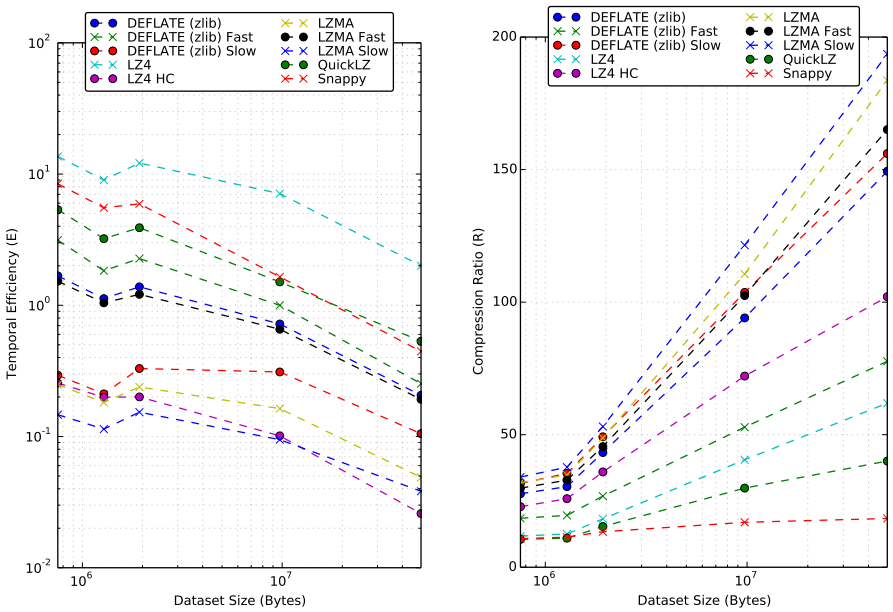
(d) MIT Killian Court, measuring 9,732,154 bytes (low resolution rendering) and 49,561,658 bytes (high resolution rendering) uncompressed.

Fig. 2. A rendering of each dataset used in our experiments. These were obtained by performing SLAM over logged sensor data.

ROS [15] framework. The logs themselves have been collected using real hardware by teams working at the aforementioned environments, used for benchmarking SLAM techniques [12], and later made publicly available.[9]

[9] The raw log data used to create these maps is available at `http://kaspar.informatik.uni-freiburg.de/~slamEvaluation/datasets.php`.

4 Benchmarking Results



(a) Temporal efficiency for each of the techniques and datasets.

(b) Compression ratio achieved by each technique for each dataset.

Fig. 3. A graphical illustration of each technique's performance on all datasets. Each of the dotted lines connects data points for the same technique, so that trends become evident. Note the logarithmic scale in some of the axes.

Fig. 3 and Table 1 illustrate the obtained results. In Fig. 3(a), we show the general trend in temporal efficiency for each technique as the size of the map grows. The general tendency is for efficiency to decrease as the data increases in size. However, in Fig. 3(b), we can observe that the compression ratio achieved tends to grow with the data's size. This effect can be attributed to the fact that, as the map grows, there are longer sequences of repetitive data, such as large open or unknown areas. It can also be explained, to a much smaller degree, by the fact that every compression technique adds control information to the compressed data, and that the size of this control data tends to be less significant as the uncompressed data grows. These figures lack error bars or other uncertainty representations due to the small dispersion of results, illustrated in Table 1 by the small values of standard deviation.

As expected, slower techniques generally achieve higher compression ratios. Furthermore, our results show that some techniques are indeed superior to others, in both temporal efficiency and compression ratio. LZ4 has shown both

Table 1. Results obtained by processing the three smallest datasets 100 times with each technique. σ_c and σ_d correspond to the standard deviations of the compression and decompression times, respectively. $\bar{T}_c$ and $\bar{T}_d$ correspond to the average compression and decompression times, respectively.

(a) Raw results obtained for the Intel Research Lab dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	27.727	15.130	1.179	1.423	0.140
DEFLATE (zlib) Fast	18.474	4.503	0.736	1.388	0.241
DEFLATE (zlib) Slow	31.633	106.519	4.167	1.306	0.195
LZ4	11.741	0.452	0.064	0.410	0.064
LZ4 HC	22.850	89.312	3.721	0.241	0.028
LZMA	31.920	126.282	7.315	2.364	0.287
LZMA Fast	29.825	17.080	1.156	2.487	0.181
LZMA Slow	34.029	229.789	13.086	2.290	0.242
QuickLZ	10.519	1.222	0.153	0.742	0.069
Snappy	10.807	0.753	0.128	0.529	0.100

(b) Raw results obtained for the ACES Building dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
LZ4	12.5734	0.737898	0.118167	0.656754	0.0954742
LZ4 HC	25.8623	129.498	9.9717	0.381131	0.0711197
DEFLATE (zlib)	30.4135	24.7584	1.49278	2.27353	0.329637
DEFLATE (zlib) Fast	19.573	8.26037	1.6616	2.41425	0.444267
DEFLATE (zlib) Slow	35.4023	165.532	5.24992	1.91064	0.341901
LZMA	34.815	187.78	10.3723	3.60015	0.352538
LZMA Fast	32.8633	27.4526	1.42182	4.04572	0.499422
LZMA Slow	37.7465	327.663	11.5554	3.62876	0.431443
QuickLZ	10.9759	2.11142	0.243054	1.29769	0.127622
Snappy	11.3352	1.20599	0.12735	0.841902	0.108266

(c) Raw results obtained for the MIT CSAIL Building dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	43.274	27.927	1.203	3.370	0.172
DEFLATE (zlib) Fast	26.818	9.100	0.382	2.717	0.178
DEFLATE (zlib) Slow	49.205	146.207	1.760	3.027	0.069
LZ4	18.236	0.779	0.052	0.725	0.090
LZ4 HC	35.953	179.027	2.698	0.432	0.087
LZMA	48.763	200.306	11.911	4.142	0.302
LZMA Fast	45.522	33.280	0.448	4.304	0.105
LZMA Slow	53.088	342.213	8.815	4.019	0.261
QuickLZ	15.359	2.533	0.117	1.407	0.088
Snappy	13.387	1.250	0.059	1.008	0.048

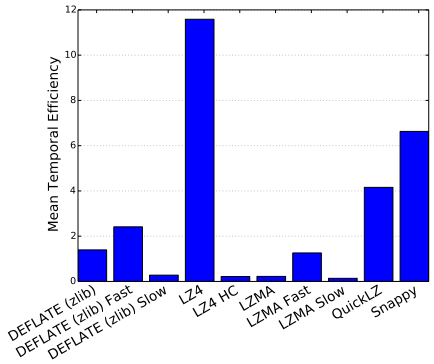
Table 2. Results obtained by processing the two largest datasets 100 times with each technique. σ_c and σ_d correspond to the standard deviations of the compression and decompression times, respectively. $\bar{T}_c$ and $\bar{T}_d$ correspond to the average compression and decompression times, respectively.

(a) Raw results obtained for the smallest MIT Killian Court dataset.

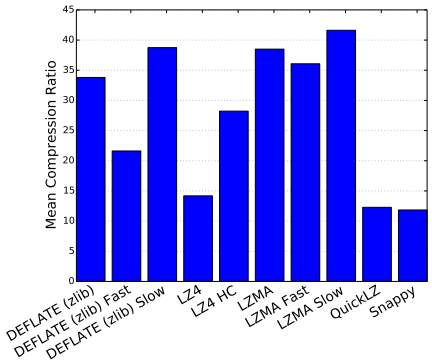
	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
LZ4	61.8855	15.0167	2.04569	15.9073	2.94617
LZ4 HC	102.05	3928.3	86.8325	12.4447	1.46376
DEFLATE (zlib)	149.383	614.592	24.0875	110.101	4.45021
DEFLATE (zlib) Fast	77.6953	242.236	21.732	65.1652	7.47955
DEFLATE (zlib) Slow	156.064	1375.26	50.3694	109.791	5.90444
LZMA	183.704	3685.39	150.48	75.4362	6.84588
LZMA Fast	165.082	776.456	20.6407	83.2567	6.06081
LZMA Slow	193.595	4995.91	386.814	63.4425	5.07394
QuickLZ	40.063	53.632	2.382	21.701	1.365
Snappy	18.400	17.986	0.799	23.335	1.080

(b) Raw results obtained for the largest MIT Killian Court dataset.

	Ratio	$\bar{T}_c$ (ms)	σ_c	$\bar{T}_d$ (ms)	σ_d
DEFLATE (zlib)	94.044	111.906	1.738	18.610	0.492
DEFLATE (zlib) Fast	52.831	41.207	3.083	11.647	0.846
DEFLATE (zlib) Slow	103.676	316.500	5.208	17.499	0.717
LZ4	40.553	2.920	0.198	2.797	0.406
LZ4 HC	72.116	710.753	32.165	1.992	0.147
LZMA	110.622	663.896	15.645	13.595	0.527
LZMA Fast	102.493	141.536	1.216	14.580	0.316
LZMA Slow	121.472	1269.680	158.155	14.937	1.938
QuickLZ	29.856	14.027	2.274	5.774	0.612
Snappy	16.951	5.192	0.751	5.101	0.492



(a) Mean temporal efficiency achieved by each technique for the three smaller datasets.



(b) Mean compression ratio achieved by each technique for the three smaller datasets.

Fig. 4. A graphical illustration of each technique's performance on smaller datasets.

a higher temporal efficiency and compression ratio than that of QuickLZ and Snappy, making it a clearly superior technique, in this case. However, LZ4 HC, LZ4's slower mode of operation, is an inferior technique in the compression of larger datasets, both in temporal efficiency and compression ratio, when compared to LZMA and DEFLATE. Its temporal performance diminishes significantly with the growth in map dimensions, with an insufficient increase in compression ratio.

In applications where compression ratio is secondary relatively to speed, LZ4 is a strong candidate, and clearly the best among the techniques that were tested. It strongly leans towards speed and away from compression ratio, but offers acceptable ratios (around 15 for smaller maps, reaching 50 in larger ones) given its extremely fast operation. In other words, for applications which rely on transmitting occupancy grids, a very significant reduction of data flow can be achieved by employing this relatively low-footprint technique, which makes it suitable for use in real-time missions. As Fig. 3(a) shows, this technique is, by far, the most efficient at utilizing resources, achieving the best results in terms of temporal efficiency among the techniques that we tested.

If further reduction in bandwidth is required, other techniques offer better ratios, at the expense of computational time. LZMA's fast mode offers one of the best ratios that we have observed, while still being acceptably fast. For the smallest dataset, this technique took, on average, about 15ms for compression, and achieved a ratio of 29.8. Depending on the application, 15ms of processor time per compression may be acceptable, given that this technique achieves a ratio that is almost three times as large as LZ4's, which achieved a ratio of 11.7, as is shown in Table 1(a).

In Fig. 4, we explore the case of the exchange of smaller maps, by averaging the temporal efficiency and ratio for each technique when operating over the

smaller datasets. Smaller maps are commonly transmitted between robots at the beginning of the mission, when there is still little information about the environment. In these conditions, we note, as mentioned before, a generalized decrease in total compression ratio, and a narrowing of the gap between slow and fast techniques in terms of compression ratio: all techniques produce results within the same order of magnitude. However, the relationships between approaches in terms of temporal efficiency remain much the same. Thus, for smaller data, faster techniques appear to be a better option, since they achieve results that are comparable to those of their slower counterparts, at a much smaller cost in computational resources.

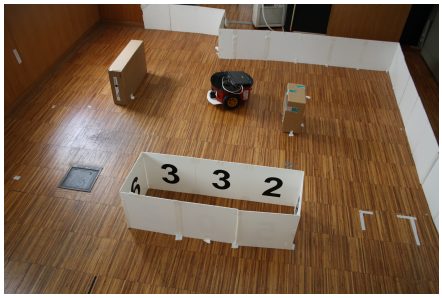
Larger maps, such as our largest examples, are very uncommonly transmitted during multi-robot missions, and hence unworthy of a closer analysis. Additionally, for these larger datasets, the multi-robot SLAM technique employed may make use of delta encoding techniques for transmission, transmitting only, for example, the updated sections of the map. In this case, we expect that the compression techniques applied to the map sections have the same performance as those applied to the smaller datasets in this test, since they will effectively be compressing smaller maps.

It is important to note that even the worse-performing techniques have achieved significant compression ratios, with a minimum ratio of about 10. Consequently, by using compression, we can reduce the total data communicated between robots during a mapping mission by at least a factor of 10, which shows the viability of compression as a solution for the problem of exchanging occupancy grids in a multi-robot system. In the context of the example we presented at the beginning of section 2, this equates to cutting our bandwidth requirements from 213KB/s per robot, to a much more affordable 21.3KB/s per robot, boosting our access point's theoretical capacity from 14 to 140 robots.

5 Real-World Application: Preliminary Results

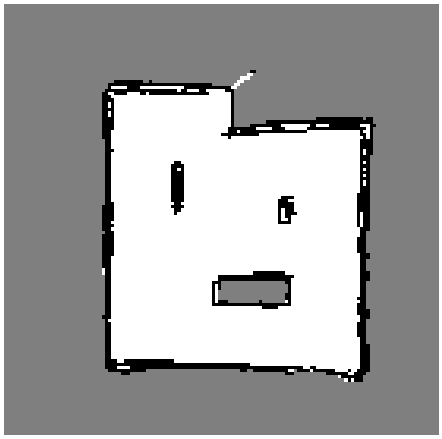
Analyzing the results obtained previously, we postulated that the usage of a compression technique in a cooperative mapping scenario would provide a significant gain in communication efficiency. To test this hypothesis, we ran several SLAM approaches in a real-world testbed. The data output by the SLAM techniques (a stream of non-simplified occupancy grids) was fed into our software, which tested its compressibility as well as the time spent on compression operations. Our software used LZ4 as its underlying compression engine, given the promising performance we observed previously.

We have gathered two sets of data, in different environments: a synthetic arena, located in one of the research laboratories of the Institute of System and Robotics, as well as on the corridors of the ISR itself. Data was gathered and recorded, so that it could then be processed by multiple SLAM techniques. This ensured that any performance gains we observed were not limited or related to any single SLAM technique. The SLAM techniques used for this validation were

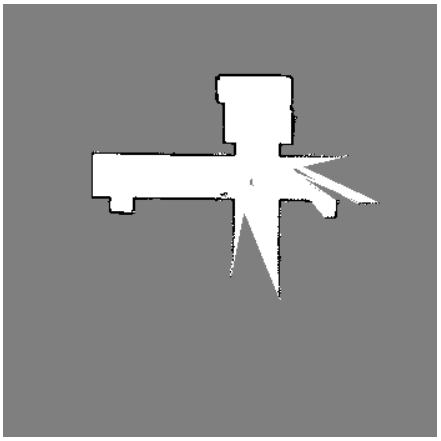


(a) A picture of the test arena, located in one of ISR's research laboratories, measuring roughly 7x7 meters.

(b) A picture of ISR's corridors. The explored area measures roughly 12x10 meters.



(c) An occupancy grid relative to the test arena.



(d) An occupancy grid relative to the ISR's corridors.

Fig. 5. An illustration of the environments where data was gathered, as well as the generic occupancy grids generated by SLAM techniques operating on each environment.

gmapping[8], *slam_karto*[11] and *hector_slam*[10]. For the sake of consistency, these tests were run on the same machine as before.

Table 3 illustrates the results obtained during the missions. Essentially, these results show that the LZ4 compression technique adopted to ensure efficiency in communication is a viable option.

As postulated in previous sections, using compression on occupancy grids yields important data savings. In this case, using real occupancy grids (as opposed to the simplified ones used previously), we saved at least about 7/8 of all data meant to be sent, which equates to approximately 88% savings in data sent. These bandwidth savings come at a very reduced computational cost, as is visible on the last column of Table 3. At the most, we spent a total of about

Table 3. Network statistics for outgoing data obtained in both scenarios. N is the number of processed maps (output by the SLAM technique into our software), $\bar{R}$ is the average compression ratio achieved during the mission, L_t is the total size of the maps received by our software (before compression), L_s is the total amount of data sent into the network by this robot and, D_s is the amount of data we saved, *i.e.* the difference between the total size of the maps and the data actually transmitted, and, finally, T_p is the total time spent processing maps, in milliseconds. All sizes are in bytes.

(a) Results obtained in the test arena.

	N	$\bar{R}$	L_t	L_s	D_s	T_p
gmapping	21	8.78	169062	19253	149809	2.77
Karto	6	8.03	48357	6015	42342	0.82
Hector	75	8.61	606667	70472	536195	9.61

(b) Results obtained in the ISR's corridors.

	N	$\bar{R}$	L_t	L_s	D_s	T_p
gmapping	21	13.92	930050	66787	863263	7.68
Karto	6	12.06	209799	17402	192397	2.64
Hector	76	12.03	3198376	265883	2932493	14.99

15 milliseconds processing maps during a mission, which, given that during that mission we saved 11/12, or 91.6%, on transmitted data, is a very positive result.

We can also observe that there is a very significant discrepancy in the average compression ratio obtained in both missions. Given the characteristics of each environment, and our previous tests, we can now reflect on the reason behind such discrepancy.

In Section 4, we observed that the map's compressibility tends to grow with the map's size, and postulated that this effect was the result of an increase of the size of "single-color" areas, *i.e.* of areas of the same type (free, occupied or unknown). In this case, the maps are very close in size, close enough that such a discrepancy cannot be justified by the map's size alone. However, these maps present very different characteristics: while the map generated in the test arena tends to feature smaller unknown areas, and with more transitions between the several states, its counterpart tends to have a large unknown area, due to the site's geometry. This leads us to believe that the reason behind the fact that a map's compressibility tends to grow with its size is, in fact, the higher likelihood of existing, in a larger map, much more substantial "single-color" areas which, as we have seen in Section 1.1, are very easily encoded in much smaller byte strings.

To conclude, these results show us that, if applied to a team of robots running a multi-robot SLAM technique based on the exchange of occupancy grids, the usage of a general-purpose compression technique is a promising solution to the inherent problem of inefficient communication.

6 Conclusion

In this text, we have explored the issue of communication optimization in the context of cooperative robotics, specifically the application of general-purpose lossless compression techniques to reduce the volume of data transmitted in cooperative robotic mapping missions. We have shown that compression is a viable option for the reduction of required network bandwidth in these scenarios, by defining and employing a new metric for the comparison of compression techniques, as well as the implementation of a new benchmarking tool. Moreover, important results about the performance of different lossless compression techniques in the context of multi-robot tasks were obtained, which can support an informed decision on which technique should be used in this context.

We have also further tested our hypothesis by employing one of the compression techniques we have tested in a real-world system, compressing the unaltered occupancy grids output by a SLAM technique. These tests yielded results that further prove the validity of this technique.

In the future, it would be extremely interesting to implement and thoroughly study this procedure within a full-fledged multi-robot SLAM technique. It would be very interesting to observe the changes in bandwidth requirements when compression is employed in such a scenario. It would also be of interest to investigate the influence of the application these techniques in the operation of Ad-Hoc networks, such as MANETs (Mobile Ad Hoc Networks), since they can be used in search and rescue operations [1], a type of operation that requires great communication efficiency.

Finally, these results only apply to solutions based on the exchange of occupancy grids, which are but a subset of all the cooperative robotic tasks in existence. Occupancy grids are not, then, by any means, the only form of data exchanged during cooperative robotic missions, it would be interesting to explore the application of compression to other types of bandwidth-heavy data that robots need to exchange, such as the more complex occupancy grids described in [7], possibly culminating in the creation of a compression technique mainly intended for the optimization of robotic communication.

References

1. F. Araujo, J. Santos, and R. P. Rocha. Implementation of a Routing Protocol for Ad Hoc Networks in Search and Rescue Robotics. In *Proceedings of IEEE 2014 Wireless Days (WD'14)*, Rio de Janeiro, Brazil, Nov. 12-14.
2. J-C Bermond, L. Gargano, S. Perennes, A. A. Rescigno, and U. Vaccaro. Efficient collective communication in optical networks. In *Automata, Languages and Programming*, pages 574–585. Springer, 1996.
3. S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 2008.
4. A. Cunningham, M. Paluri, and F. Dellaert. DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3025–3030. IEEE, 2010.

5. P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996.
6. A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.
7. J. F. Ferreira, M. Castelo-Branco, and J. Dias. A hierarchical Bayesian framework for multimodal active perception. *Adaptive Behavior*, 20(3):172–190, 2012.
8. G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007.
9. D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
10. S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 155–160. IEEE, 2011.
11. K. Konolige, G. Grisetti, R. Kummerle, W. Burgard, B. Limketkai, and R. Vincent. Efficient sparse pose adjustment for 2d mapping. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 22–29. IEEE, 2010.
12. R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27(4):387–407, 2009.
13. M. T. Lazaro, L. M. Paz, P. Pinies, J. A. Castellanos, and G. Grisetti. Multi-robot SLAM using condensed measurements. In *Proc. of 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2013)*. IEEE, 2013.
14. E. Pedrosa, N. Lau, and A. Pereira. Online SLAM Based on a Fast Scan-Matching Algorithm. In *Progress in Artificial Intelligence*, pages 295–306. Springer, 2013.
15. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 2009.
16. R. P. Rocha. *Building Volumetric Maps with Cooperative Mobile Robots and Useful Information Sharing: a Distributed Control Approach Based on Entropy*. PhD thesis, University of Porto, Portugal, 2006.
17. R. P. Rocha, D. Portugal, M. Couceiro, F. Araujo, P. Menezes, and J. Lobo. The CHOPIN project: Cooperation between Human and rObotic teams in catastroPhic INcidents. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–4. IEEE, 2013.
18. O. Rodeh and A. Teperman. zFS-a scalable distributed file system using object disks. In *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, pages 207–218. IEEE, 2003.
19. D. Salomon. *A concise introduction to data compression*. Springer, 2007.
20. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.
21. J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536, 1978.