Introduction to and Instructions for the EECS 651 Term Project

Projects typically:

- Are concerned with lossy, as opposed to lossless, coding. Thus they deal with rate and distortion, as opposed to just rate. However, even lossy coding projects may deal with quantization with variable-rate coding, in which case it can have a substantial lossless coding component.
- Have a strong experimental, e.g. computer, component.
- Will have a focus on some particular source or coding technique or fidelity criteria or analysis technique or complexity contraint or some other aspect or constraint.
- Will be related to previous work found in the literature.

Projects must:

- Have well specified focus and goals.
- Have a strong connection to the material presented in the course. For example, in so far as possible, the project should be described using the terminology and concepts introduced in the course (as opposed to the terminology and concepts particular to some paper). Connections between the subject of the project and the course should be recognized. Analysis methods developed in the class should be used where possible. Comparisons of the system being considered in the project to those studied in class should be made.
- Suggest, analyze, and/or experiment with alternative parameters, or analysis methods. Questions include: Why is the system designed the way it is? Why does it work as well or as poorly as it does? Could it be improved? Is the performance of the system explanable? Does it match analytical predictions? Does the system performance relate to other systems in some way that can be explained? Try to have an interesting "story" to tell (problem, hypothesized approach, results).
- Critically assess the design and performance of the system being studied. Be critical, judgmental, quantitative. Do "engineering" where possible. It is more important that the assessments be done well, than it is for the system to have outstanding performance. (But you don't want to be testing a naive system.)

Projects may:

- Use software developed elsewhere. The project is not intended as a test of your programming skills. Rather it should make use of your engineering/experimentation/analysis/interpretation skills.
- Closely follow old methods. You need not propose something new. There is always plenty of room for experimentation and analysis, even with the simplest and oldest of methods or issues.

Groups

Group projects are strongly encouraged. Groups of size 2, 3 and 4 are appropriate. With 24 students we need to limit the number of groups to at most 8, i.e. an average of at least 3. If you are looking for other students in the class with similar or complementary interests, send email to eecs651disc. This goes to all class members and can be used by all class members.

Deliverables

A project proposal: Due Wednesday, March 21, in class, before the lecture begins.

This should spell out in as much detail as possible the scope of your project (source, coding technique, fidelity critieria, experiments, analsysis methods, benchmarks, software and source material to be obtained or created, etc.) This is not the sort of thing you do the night before the

due date. And you should not consider that you start your project after the proposal is graded. Rather you should invest a lot of time in your proposal, so that your project will go as smoothly as possible.

An oral presentation: At a date and time to be determined. Each group will be given something like 20 minutes.

A written report: One per group. Due on the last day of finals, by 4PM, April 26.
Write the report for your fellow sutdents. They should be able to learn from it, repeat what you have done. etc.

Feedback

To get feedback and suggestions, each group must meet with me:

before March 14 to discuss preliminary plans
after the proposal is graded
after the oral report.

Grading will be based on

The proposal.

The final project report, as well as the oral presentation.

Factors include: Quantity, quality, degree of difficulty of the work. The analytical or experimental results. The depth of understanding displayed. Innovativeness. Connectedness to the course.

The project counts 35% of the overall class grade. The proposal counts 5%, and the oral and written portions count 30%.

All members of a group will receive the same project grade.

1. Some Potential Sources

a. Speech

CELP is the most widely used high performance type of speech coder. There are many variations of CELP and many parameters to experiment with.

Waveform Interpolation is a new method now receiving attention.

We have some files of speech samples. One can easily create more using Mac's. It should be possible to get some standard speech coders to experiment with or modify.

b. Audio/Music

Audio coders typically use transform or subband coding. It is particularly interesting how modern audio coders use perceptual models of hearing to produce decoded reproductions with low SNR, e.g. 10 to 20 dB, that sound just like the original.

We have a files of music, more can be obtained.

c. Images

Image coding makes for very nice projects because it is easy to SEE the results. Wavelet coding methods seems to be the current best combination of performance and complexity. A new wavelet coding method to replace JPEG is currently under development.

We have access to lots of images. We have access to JPEG source code. There are many image coders to be found at various websites.

d. Video

Video coding continues to be a major frontier area of compression. Modern standardized methods use motion compensated transform coding. MPEG and H.263 are two very well known standards (the latter is fixed-rate coding for teleconferencing). Generally, an image coding project will deal with just a few frames of a video sequence in order to avoid having to deal with huge amounts of data.

We have access to several video sequences, and more can be obtained on the web. Software implementing standard video coders is available.

3. Source models

Many projects will need a source model for design purposes. Some may focus on source modeling itself; i.e. trying to match source models to sources in order that codes or theory developed for the model will accurately predict the performance of codes on actual sources. Sample models include:

Markov, autoreregressive (AR), moving average (MA), ARMA, hidden Markov, composite, Markov random fields, and Gaussian and Laplacian densities are widely used because of their reasonableness and tractability.

3. Fidelity criteria

If you're doing quantization, you'll have to choose one. Or a project could focus on fidelity criteria themselves. For example, what image or speech fidelity criteria most accurately reflect human perception. The recent big advances in high fidelity compression of music is based on models of human hearing. Related to this is the development of "perceptually" based codes; i.e. codes that are designed to "sound good" or look good" but which needn't have a good SNR. For example, how should a transform code be optimized for good rate vs. perception, as oposed to rate vs. SNR.

4. Types of codes

Many projects will focus on some particular coding technique and how to tune it up (choose its parameters) or to modify it to improve performance or reduce complexity. In class we've considered (at least briefly) many types of codes.

6. Experiments

Most projects will probably involve a computer implementation and testing of some source coding method. A source, code, fidelity criteria and design methodology will have to be selected.

7. Analysis

The goal of source coding theory is to predict how well a given code will work, how well the best codes can work, and how to design good codes. Most projects will at least take a stab at modeling their source, finding formulas for code performance, and estimating the fundamental limits of the specific coding technique on the given source. One can also do nonformulaic analysis, wherein one critically assesses the results of experiments by comparing to appropriate benchmarks.

8. Complexity

The principal issue in code design is performance at a reasonable complexity. So most projects will make an effort to quantify the complexity of the method and, where possible, to minimize complexity . Some projects could focus on optimizing a method for the best performance/complexity tradeoffs.

9. Robustness

One might wish to consider the robustness of a coding technique to variations in the source or to errors in transmission of the bits from encoder to decoder.

10. Facilities

CAEN computers will probably be the most commonly used computer, but any computer you can get your hands on is OK. Beware that for some projects, PC's and Mac's may not have sufficient power.

We do have programs that implement the generalized Lloyd VQ design algorithm, and also encoding and decoding algorithms. Another program implements the standard TSVQ design. (And there are TSVQ encoding and decoding programs.) We also have a tree pruning program. Some standard speech, audio, image, video and lossless coding algorithms are available or can be found on the web. It is possible for you to build upon these, or use them as benchmarks.

11. Some additional potential project foci:

Noisy channel quantization: Design and/or analyze lossy source codes that must operate in the presence of bit errors. For example, model the channel as binary symmetric channel. Or combine source and channel coding, or source coding and modulation.

Noisy source quantization: Suppose source X is corrupted by noise before it reaches your encoder. Design the encoder/decoder to get the best rate/distortion performance, where distortion is measured between the decoded output and the original uncorrupted source output X.

Progressive/embedded/layered coding: Suppose the encoding must be done so that the encoded bits can be grouped into layers so that a rough approximation can be decoded from the first layer, a better approximation from the first two layers, and so on. Sometimes the encoder can be structured to produce the layers in order. Sometimes the encoder determines all layers at the same time. Does layering limit performance? If so, by how much?

Multiple description source coding: Here the encoder produces two or more encoded versions of the source data such that if only one is received, a satisfactory approximation can be decoded. However, if two or more are received, then better approximations will be produced. How can

one get the best possible performance from the multiple encoded versions while maintaining satisfactory approximations from just a subset?

Complexity constraints: Source code design with special complexity constraints, such as that the decoder be especially simple, or that the encoder be especially simple.

Continuous time sources: For continuous time sources, what are the tradeoffs between sampling rate, overall code rate (in bits/sec) and complexity? (Real-world sources are not strictly bandlimited so a sampling rate must be chosen.) For example, with coding techniques like DPCM or transform coding, how will the design and performance depend on sampling rate?

How does the performance of a quantizer relate to its complexity?

Joint quantization and estimation/detection: Design quantizers with the goal of being able to estimate or detect something (like the presence or absence of some pattern) from the decoded data that one could estimate/detect from the original source data in addition to or instead of trying to minmize decoder distortion.

Independent encoding of correlated data: Suppose the outputs of two correlated sources must be independently encoded, but jointly decoded. How to do this in a good way? What performance is possible?