ENSC 424 Final Project - Multi-view Image Codec

Graham Holland
301065432
gmh7@sfu.ca

Nov 25, 2011

Contents

1 Introduction

The goal of this project was to develop a codec (encoder and decoder) for a set consisting of up to 10 multi-view images. Each camera was located in the same plane for each of the images in the set, resulting in a series of images resembling a camera pan from left to right. The development of the codec included various components largely taken from Labs 2, 5 and 6. However the code was heavily modified to enable operation on a larger number of files.

2 Background

Since we are dealing with image sets that differ only in camera placement, the image sets can be thought of as frames within a short video sequence. As a result the techniques for video compression we have been discussing in class can be applied to the image sets. That is we can use motion estimation and motion compensated prediction along with a transform, quantization step followed by entropy encoding to greatly reduce the bitrate of the compressed bitstream.

Additionally it may be possible to use the concept of different frame types as used by the MPEG standards to achieve better compression through minimizing the motion compensated prediction residual (MCPR). These being the I (intra-coded), P (forward predicted) and B (bi-directionally predicted) frames, using weightings in the latter case according to the following formulas.

$$\hat{f}(t,m,n) = f(t-1,m,n) \tag{2.1a}$$

$$\hat{f}(t,m,n) = f(t-1,m-d_x,n-d_y) \tag{2.1b}$$

$$\hat{f}(t,m,n) = w_b f(t-1,m-d_{b,x},n-d_{b,y}) + w_b f(t-1,m-d_{b,x},n-d_{b,y}) \tag{2.1c}$$

It will be beneficial to encode the quantized DC coefficients of the DCT on each block differentially as was done in Lab 5. This decreases the variances and thus improves coding efficiencies. It is possible to use the same technique to encode the motion vectors for each block for P and B frames.

Using the analogy between the the multi-view images and frames of a video some more features from the MPEG family of codecs to try to implement include:

- sub-pixel motion estimation using linear interpolation
- in loop deblocking filter on motion compensated predicted frames
- different quantizers for intra and predictive encoding
- varying block size of the DCT and block based motion estimation
- varying the search range and search algorithm in motion estimation

Finally it is important to note that when using motion compensation, the encoder must use a quantized/dequantized MCPR in when adding to the predicted frame to get the reference for the next image. This is due to the fact that the decoder does not have access to the original images. To reduce errors the operations that the decoder would perform need to be done by the encoder also.

3 Implementation Details

My first step in the design of the codec was to take the code I had submitted for Lab 5 along with the arithmetic encoder submitted for Lab 2 and modify them appropriately to work in the context of the mutli-view image sets. This was no small task and the arithmetic encoder and decoder were rewritten from scratch as their own functions. I began using the same quantization matrix and zig zag pattern used for JPEG image compression along with the same quality scaling without any motion estimation or motion compensated prediction.

The basic idea behind my implementation of the `im_encode.m` function was to simplify the passage of data between different parts of the encoder notably the transform and quantization sections and the entropy encoding. To begin with I would perform all DCT and quantization for each of the images storing the frames within a 3-dimensional array. This array was then passed to the entropy encoder which encoded and wrote all the frames at one time. This was done to prevent problems with opening and closing the bitstream file during operation of the encoder.

Similarly for `im_decode.m`, I performed all the entropy decoding at once and passed back a 3-dimensional array of decoded.

It should be noted that in performing entropy encoding the histogram needed by the arithmetic decoder is written to the bitstream adding some overhead and increasing the bitrate slightly. This cannot be avoided however since all information passed between encoder and decoder must be sent through the bitstream.

The next task was to implement the motion estimation and motion compensated prediction in a similar way to Lab 5. At this point it also seemed beneficial for maintenance reasons to split the quantization step into its own functions for encoding and decoding, `quantize_DCT.m` and `dequantize_DCT`, respectively.

With the basic framework in place for motion estimation, I began making improvements by adding quantization of the MCPR using a uniform quantizer with the same quality scaling used by the JPEG quantizer in Lab 5. Additionally I made sure to encode the motion vectors using differential encoding as was done for the quantized DC coefficients in the intra-coded frames. In converting

The final step in the development of my codec involved adding a framework to enable bi-predictive motion estimation, or in other words the use of B frames. In doing so I tried to maintain generality so I would be able to test various schemes for assigning frame types to the different images.

Once this was complete I set about testing different the different frame assignments as I, P and B. I discovered that I achieved the best performance with the centre image of the set as a single I-frame with the rest of the images as P frames. It turned out the overhead of a second set of motion vectors prevented the use of B frames from giving any real benefit in terms of rate distortion criteria.

4 Results

Figures 4.1 and 4.2 show the best results I was able to obtain using my codec with the two image sets provided, run with the quality parameter at 10, 30, 50, 70 and 90.

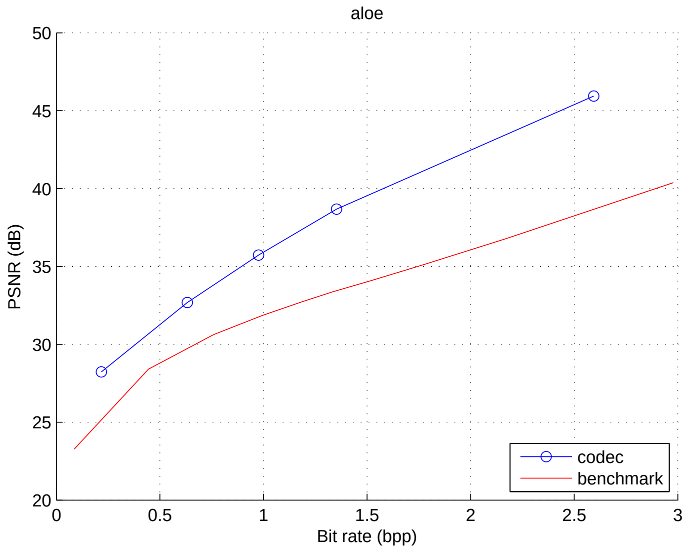

Figure 4.1: Codec performance with aloe image set

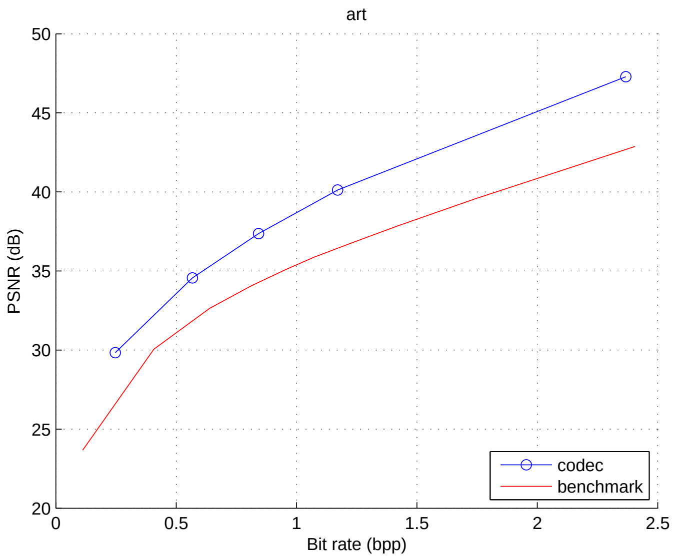

Figure 4.2: Codec performance with aloe image set

As can be seen the performance with the aloe image set is worse than that for the art image set. This is due to the high amount of texture in the art image set which allows for better motion estimation and prediction.

5 User Manual

The following pages show the MATLAB documentation that was written for each of the functions contained within the codec. Additionally the file `run_codec.m` was written to aid testing the codec during development. The documentation for this utility function is also included.

```
IM_ENCODE Run the project encoder
   [] = IM_ENCODE(NAME, BITSTREAM_NAME, N_IMAGES, QUALITY) runs the
   project encoder on the image set with common prefix NAME and N_IMAGES
   different images. BITSTREAM_NAME specifies the filename to use as the
   bitstream as the output of the encoder.

   QUALITY is a parameter that determines the rate-distortion
   characteristic for the output of the encoder and should be in the range
   1 to 100. Lower values for QUALITY will result in lower image PSNR but
   with subsequently lower bitrate and vice versa. These quality levels
   correspond roughly to the quality defined in the JPEG standard

   See also im_decode entropy_enc

IM_DECODE Run the project decoder
   [] = IM_DECODE(BITSTREAM_NAME, DEC_NAME, N_IMAGES) runs the project
   decoder using the bitstream specified by BITSTREAM_NAME as input. The
   bitstream given must be one produced by the im_encode function run on
   a set with N_IMAGES different images. DEC_NAME specifies the common
   prefix to use when naming the decoded files.

   See also im_encode entropy_dec

QUANTIZE_DCT Quantize DCT coefficients
   IMGQ = QUANTIZE_DCT(DCT_FRAME, QUANT_TYPE, QUALITY) quantizes the image
   frame DCT_FRAME using the quantization matrix and zig-zag pattern.

   There are two options for QUANT_TYPE, 'jpeg' or 'uniform'. 'jpeg' uses
   a JPEG like quantizer while 'uniform' uses a uniform quantizer. QUALITY
   determines the scaling on the quantizer size and should be in the range
   1 to 100.

   The quantized DCT coefficients for each block appear in each row of
   IMGQ arranged according to the zig zag pattern from the JPEG specification.
   IMGQ is therefore a M*N/64 by 64 matrix where the dimensions of DCT_FRAME
   are M by N.

   See also dequantize_DCT init_quantizer

DEQUANTIZE_DCT Dequantize DCT coefficients
   DCT_MAT_DEC = DEQUANTIZE_DCT(IMGQ_DEC, M, N, QUANT_TYPE, QUALITY)
   dequantizes the DCT coefficients from IMQ_DEC where each row in IMQ_DEC
   corresponds to the 64 DCT coefficients of an 8x8 block arranged according
   to the zig zag pattern from the JPEG specification.

   There are two options for QUANT_TYPE, 'jpeg' uses a JPEG like quantizer
   while 'uniform' uses a uniform quantizer. QUALITY determines the scaling
   on the quantizer size and should be in the range 1 to 100.
```

```
   The matrix of DCT coefficients for the frame is returned in DCT_MAT_DEC,
   where M and N are the frame height and width of the returned frame.

   See also quantize_DCT init_quantizer

ENTROPY_ENC Perform entropy encoding for project encoder
   [] = ENTROPY_ENC(FRAME_H, FRAME_W, FRAME_INFOS, FRAMEQ
                    FWD_MVXS, FWD_MVYS, BACK_MVXS, BACK_MVYS
                    BITSTREAM_NAME, N_IMAGES, QUALITY)
   Performs entropy encoding on the data generated during the execution of
   im_encode, writing information needed by the decoder to the bitstream
   specified by BITSTREAM_NAME.

   FRAME_H and FRAME_W are the frame height and width respectively.
   FRAME_INFOS is a 1-by-N_IMAGES struct array containing the information
   about each frame including fields for frame number, type etc. FRAMEQ is
   3 dimensional array holding the quantized DCT coefficients for each of
   the encoded frames. FWD_MVXS, FW_MVYS, BACK_MVXS and BACK_MVYS all
   contain motion vectors for each frame.

   Bitstream will contain (in this order):
       general info:
           frame_h         uint16
           frame_w         uint16
           quality         uint8

       for each frame:
         I, P and B frames:
           frame_num       uint8
           frame_type      uint8

         B frames:
           fwd_ref         uint8
           back_ref        uint8
           wb              single
           mv_min_index    int16
           Nmv_counts      uint16
           mv_counts       uint32*Nmv_counts
           Nbits_mv_enc    uint32
           mv_enc          ubit1*Nbits_mv_enc

         P frames:
           fwd_ref         uint8
           mv_min_index    int16
           Nmv_counts      uint16
           mv_counts       uint32*Nmv_counts
           Nbits_mv_enc    uint32
           mv_enc          ubit1*Nbits_mv_enc

         I, P and B frames:
           imgq_min_index  int16
           Nimgq_counts    uint16
           imgq_counts     uint32*Nimgq_counts
           Nbits_imgq_enc  uint32
           imgq_enc        ubit1*Nbits_imgq_enc

   See also im_encode entropy_dec

ENTROPY_DEC Perform entropy decoding for project decoder
   [FRAME_H, FRAME_W, QUALITY, FRAME_INFOS, FRAMEQ_DEC, ...
    FWD_MVXS, FWD_MVYS, BACK_MVXS, BACK_MVYS] = ...
                                ENTROPY_DEC(BITSTREAM_NAME, N_IMAGES)
   Performs entropy decoding on the data contained with the bitstream
```

```
   specified by BITSTREAM_NAME produced by entropy_enc.

   The many return values from this function are used by the im_decode
   function to continue decoding the images. See the documentation for
   entropy_enc for the meaning of each of the return values.

   See also im_decode entropy_enc

MOTION_ESTIMATION Block-based motion estimation
   [MVX, MVY] = MOTION_ESTIMATION(PREV, CURR, BLKX, BLKY, SEARCH_RANGE)
   This function will perform block-based motion estimation between the
   previous frame, PREV, and the current frame CURR. The blocks used in
   the motion estimation have BLKY rows and BLKX columns.

   Parameter SEARCH_RANGE specifies the range over which the block in
   the current frame will search for the best matching block in the
   previous frame. For example, if search_range = 16, then the best
   matching block will be searched over +/-16 pixels (horizontally
   and vertically) relative to the position of the current block.

   MVX and MVY store the x-component and y-component of the motion
   vectors respectively.

MC_PREDICTION Block-based motion-compensated prediction
   PRED = MC_PREDICTION(PREV, MVX, MVY)
   This function performs block-based motion-compensated prediction of
   a frame from the previous frame, PREV, using the motion vector field
   described by MVX and MVY.

   The predicted frame will be returned in PRED.

   See also motion_estimation

 INIT_QUANTIZER Initialize quantization table and zig-zag scan.
    [QT, ZAG] = INIT_QUANTIZER(QUANT_TYPE, QUALITY)
    Returns the a row vector QT which is the quantization step sizes for
    the DCT coefficients of an 8x8 block arranged according to the zig zag
    pattern from the JEPG specification.

    QUANT_TYPE specifies the quantizer type and must be one of 'uniform' or
    'jpeg', where uniform returns a uniform quantizer while 'jpeg' returns
    a JPEG-like quantizer.

    In both cases the elements of QT are scaled by QUALITY which should be
    in the range 1 to 100

    ZAG is an 8x8 matrix with the zig zag pattern of the DCT coefficients.

INIT_FRAMES Initialize an array of frame structures
   [FRAMES, ORDER] = INIT_FRAMES(N_IMAGES) initializes and then returns
   a 1-by-N_IMAGES struct array containing frame information based on the
   number of images in the set N_IMAGES. Also returns ORDER, an array
   of length N_IMAGES with the ordering of the encoder to use.

   Each struct in the array contains the following fields
       frame.num - order number for the frame within the set beginning at 1
       frame.type - one of 'I', 'P' or 'B'
       frame.fwd_ref - the reference image to use for forward motion
                       estimation
       frame.back_ref - the reference image to use for backward motion
                        estimation
       wf - weighting factor for forward motion compensation
       wb - weighting factor for backward motion compensation
```

```
INIT_ZAG Initialize a zig zag traversal matrix for motion vector scanning
   ZAG = INIT_ZAG(FRAME_H, FRAME_W, ZAG_TYPE) returns a matrix with elements
   increasing in a zig zag pattern. FRAME_H is the number of rows and FRAME_W
   is the number of columns in the motion vector matrix. ZAG_TYPE is one of
   'diagonal', 'horizontal' or 'vertical'

   See also init_quantizer

RUN_CODEC Run the project codec at various quaility levels
   [MSEs, PSNRs, BITRATES] = RUN_CODEC(NAME, DEC_NAME, N_IMAGES, QUALITIES)
   Runs the project codec on the image set with common prefix of NAME that
   contains N_IMAGES different images. DEC_NAME specifies the common prefix
   to use for naming the decoded images. The encoder and decoder are run
   once for each quality value specified in the row vector QUALITIES.

   This function also computes MSE, PSNR and bitrates in bits in bits per
   pixel for each quality value in QUALITIES. These stats are returned in
   MSEs, PSNRs and BITRATES respectively. Additionally these results are
   saved to a MAT file with filename NAME.MAT.

   The rate-distortion curve for the codec is also plotted. Additionally,
   if NAME is one of 'aloe' or 'art', the benchmark rate-distortion curves
   for the project are plotted in red on the same plot.

   See also im_encode im_decode
```