FIR Low Pass Filter

Analyze and implement a simple low pass filter given by the 9 × 9 point spread function:

$$h(m,n) = \begin{cases} 1/81 & \text{for } |m| \leq 4 \text{ and } |n| \leq 4 \\ 0 & \text{otherwise} \end{cases}$$

Derivation

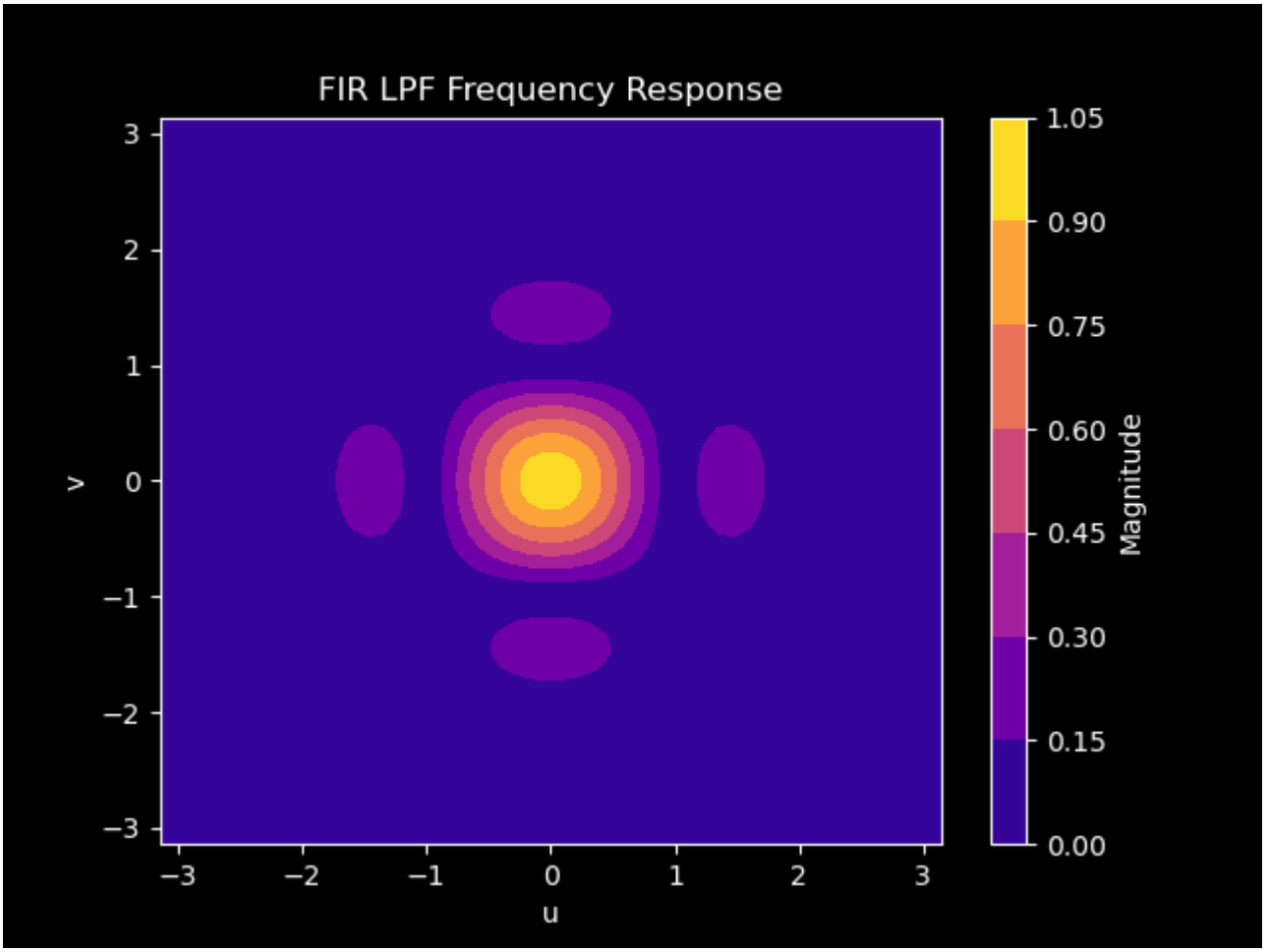
We can plot the magnitude of the impulse response by finding the analytical expression for $H(e^{ju}, e^{jv})$ across all values, which is the DSFT:

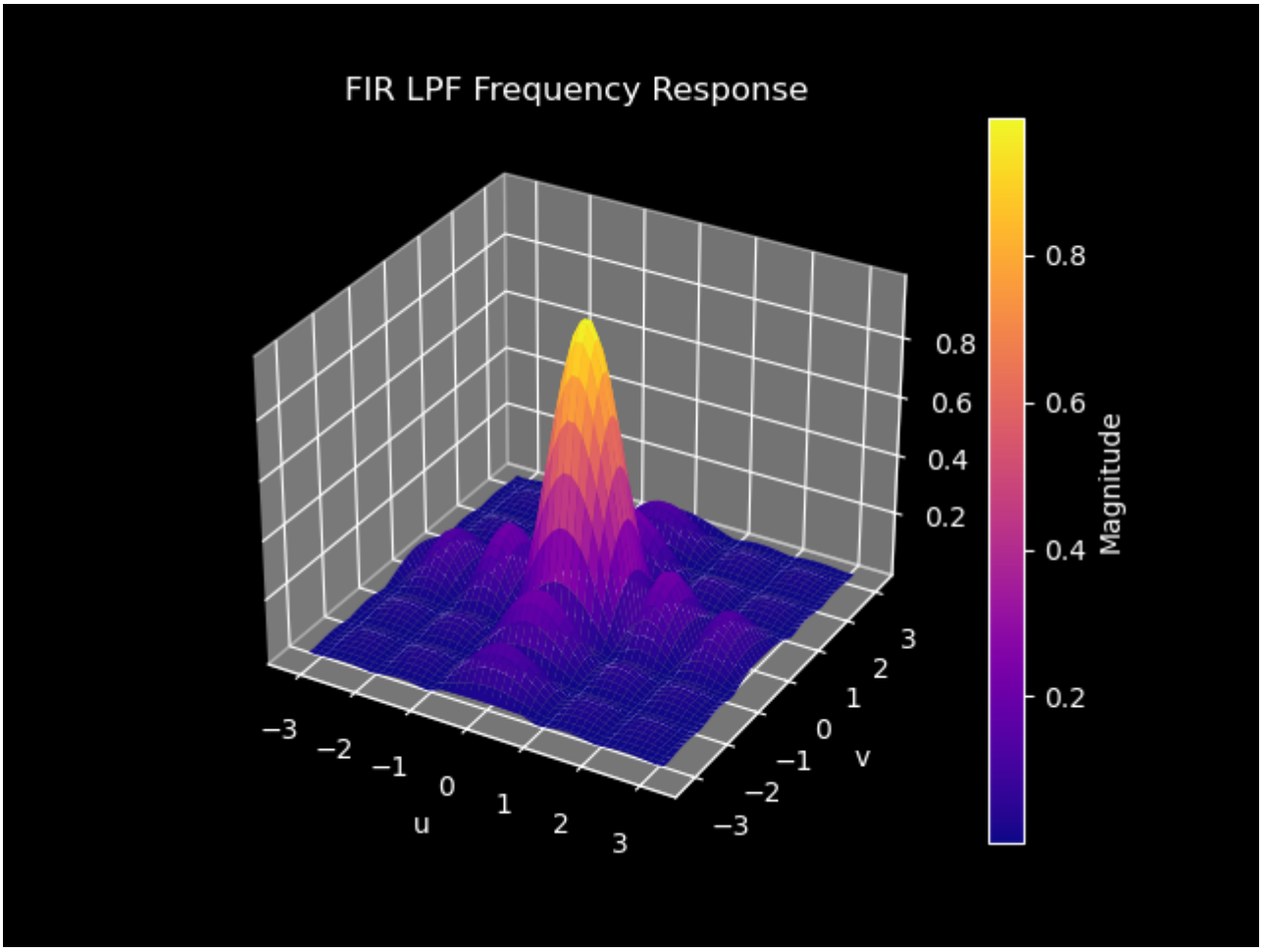
$$H(u,v) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f(n,m) e^{-j2\pi(\frac{un}{N} + \frac{vm}{M})}$$

We can substitute $\frac{1}{81}$ within the range $-4 \leq n, m \leq 4$ since that is the only non-zero piece of the function and get the following:

$$H(u,v) = \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{81} e^{-j2\pi(\frac{un}{9} + \frac{vm}{9})}$$

Frequency Response Plots


FIR LPF Frequency Response
v
u
Magnitude
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00


FIR LPF Frequency Response
u
v
Magnitude
0.8
0.6
0.4
0.2

Images

Input image

Filtered Image

Code

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
void apply2DFIRFilter(uint8_t **input, uint8_t **output, int height, int width);
```

```
int main(int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;

  if (argc != 2)
    error(argv[0]);

  /* open image file */
  if ((fp = fopen(argv[1], "rb")) == NULL)
  {
    fprintf(stderr, "cannot open file %s\n", argv[1]);
    exit(1);
  }

  /* read image */
  if (read_TIFF(fp, &input_img))
  {
    fprintf(stderr, "error reading file %s\n", argv[1]);
    exit(1);
  }

  /* close image file */
  fclose(fp);

  /* check the type of image data */
  if (input_img.TIFF_type != 'c')
  {
    fprintf(stderr, "error:  image must be 24-bit color\n");
    exit(1);
  }

  /* set up structure for output color image */
  /* Note that the type is 'c' rather than 'g' */
  get_TIFF(&output_img, input_img.height, input_img.width, 'c');

  apply2DFIRFilter(input_img.color[0], output_img.color[0], input_img.height, inpu
  apply2DFIRFilter(input_img.color[1], output_img.color[1], input_img.height, inpu
  apply2DFIRFilter(input_img.color[2], output_img.color[2], input_img.height, inpu

  /* open image file */
  if ((fp = fopen("fir_lpf.tif", "wb")) == NULL)
  {
    fprintf(stderr, "cannot open file fir_lpf.tif\n");
    exit(1);
  }

  /* write image */
  if (write_TIFF(fp, &output_img))
  {
    fprintf(stderr, "error writing TIFF file %s\n", argv[2]);
```

```
    exit(1);
  }

  /* close image file */
  fclose(fp);

  /* de-allocate space which was used for the images */
  free_TIFF(&(input_img));
  free_TIFF(&(output_img));

  return (0);
}

const uint8_t FILTER_SIZE = 9;
const double FILTER_COEFFICIENT = 1.0 / 81.0;

void apply2DFIRFilter(uint8_t **input, uint8_t **output, int height, int width)
{
  int filterRadius = FILTER_SIZE / 2;

  for (int i = 0; i < height; ++i)
  {
    for (int j = 0; j < width; ++j)
    {
      // printf("calculating sum for r%d c%d\t", i, j);
      double sum = 0.0;
      for (int m = 0; m < FILTER_SIZE; ++m)
      {
        for (int n = 0; n < FILTER_SIZE; ++n)
        {
          int rowIdx = i - filterRadius + m;
          int colIdx = j - filterRadius + n;
          // printf("i: r%d c%d ", rowIdx, colIdx);

          // Check boundaries
          if (rowIdx >= 0 && rowIdx < height && colIdx >= 0 && colIdx < width)
          {
            sum += FILTER_COEFFICIENT * (double)input[rowIdx][colIdx];
          }
        }
      }
      // printf("raw sum %f\n", sum);

      // Clip the result to the 0-255 range
      output[i][j] = (uint8_t)(sum < 0 ? 0 : (sum > 255 ? 255 : sum));
    }
  }
}

void error(char *name)
{
  printf("usage:  %s  image.tiff \n\n", name);
```

```
  printf("this program reads in a 24-bit color TIFF image.\n");
  printf("It then horizontally filters the green component, adds noise,\n");
  printf("and writes out the result as an 8-bit image\n");
  printf("with the name 'green.tiff'.\n");
  printf("It also generates an 8-bit color image,\n");
  printf("that swaps red and green components from the input image");
  exit(1);
}
```