Lab 4: Speaker Recognition Using TI C6713 DSK

EECS 152B/CSE 135B DSP Design & Laboratory

Xiaoran Li, 35058463
Jeffrey Thompson, 12987953
Deukkwon Yoon, 49115326

Lab Session: Lab 1A
Lab Dates: 02/24/2016 and 03/09/2016

CSE/EECS Department
Henry Samueli School of Engineering
University of California, Irvine

March 16, 2016

Introduction

In this lab, a speaker recognition system was implemented on the TI C6713 DSK board using Linear Prediction method. First, information about the speaker was gathered in the training phase by isolating word signals from the speech signal and finding mean and covariance of the isolated words. In the recognizing phase, the trained data was used for linear prediction to identify the speaker. All samples were taken at 8kHz sampling rate and only left channel was used. The C code for implementing the speaker recognition is shown in *Listing 2.1* and *Listing 2.2*. The accuracy of the program is summarized in *Table 1*.

Data

Method	Mahalanobis Distance		
P	User 1	User 2	User 3
2	86.6	100	80.0

Table 1. Recognition accuracy table in percent.

Word Isolation and LP Coefficients

Real-time word isolation was implemented using exponential moving average with a small α value.

$$EMA_i = \alpha * SAMPLE_i + (1-\alpha)EMA_{i-1}$$

Then, the moving average was thresholded to define boundaries of a word. State machine was used to keep track of whether the thresholded signal is the start or the end of the signal. For each isolated words, LP coefficient, $\boldsymbol{a}$, was found where,

$$\boldsymbol{a} = \boldsymbol{R}^{-1}\boldsymbol{r}$$

$$\boldsymbol{r} = \sum_{p+1}^{N} s_n s_{n-1},\ \ \boldsymbol{R} = \sum_{p+1}^{N} \boldsymbol{s}_{n-1}\boldsymbol{s}_{n-1}^T,\ \ \boldsymbol{s}_{n-1}^T = [\boldsymbol{s}_{n-1}\ \boldsymbol{s}_{n-2}\ ...\ \boldsymbol{s}_{n-p}]$$

Calculating the coefficients required defining matrix multiplication and transpose functions in C program.

In order to filter out some noise, duration constraint was added to ignore isolated signals that were too short to be a spoken word. This made our program to be more tolerating to noisy signal input.

Training

In the training phase, N number of isolated words were used to profile the speaker. The LP coefficients found for each word was used to find the mean and the covariance of the LP coefficients.

$$\boldsymbol{u} = \frac{1}{N}\sum_{m=1}^{N} \boldsymbol{a}_m$$

$$\boldsymbol{C} = \frac{1}{N}\sum_{m=1}^{N}(\boldsymbol{a}_m - \boldsymbol{u})(\boldsymbol{a}_m - \boldsymbol{u})^T$$

Identifying Speakers

After the training phase, the speaker was identified by comparing the Mahalanobis distances between the LP coefficient of the speaker and the trained profiles.

$$d_M = (\boldsymbol{a} - \boldsymbol{u})^T \boldsymbol{C}^{-1}(\boldsymbol{a} - \boldsymbol{u})$$

The profile with the minimum Mahalanoibs distance, d_M,was identified as the speaker.

Conclusion

Speaker recognition algorithm was implemented on the TI C6713 DSK board by finding the Mahalanobis distance of LP coefficients. To implement detection in real-time, exponential moving average was used instead of simple moving average by convolution. This not only decreased amount of memory usage but also allowed identification in real-time. State machine was used to isolate the word signal from the speech signal and each isolated word was analyzed to find the LP coefficients. During the training phase, LP coefficients were used to find mean and covariance which profiled the speaker. In the detection phase, speaker's LP coefficients were compared to the saved profiles by finding the Mahalanobis distance. The profile with the minimum distance to the speaker's LP coefficient was identified as the speaker.

The C program that was used to implement the speaker recognition is shown in *Listing 2.1* and *Listing 2.2*. The accuracy of the program is summarized in *Table 1*. For value of $P = 2$, average accuracy for 3 users was 88.8%. If a different P value such as $P = 3$ was used, accuracy would have been 100%.

C Program

```c
#include "dsk6713_aic23.h"

#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINEIN 0x0011

#include <stdio.h>
#include <stdlib.h>

Uint32 fs = DSK6713_AIC23_FREQ_8KHZ; // 1
Uint16 inputsource = DSK6713_AIC23_INPUT_LINEIN; // 0x011

#define SIMULATION

#define ALPHA .02f
#define A1 .98f

#define WORD_COUNT 5
#define SAMPLE_RATE 8000.0f

#define MAG_THRESHOLD 200
#define TIME_THRESHOLD 2000

#define P 3
#define NUM_USERS 3
```

Listing 1.1 C Program header.

```c
struct coef{
    float val[P];
};
typedef struct coef Coef;

struct r{
    long val[P];
};
typedef struct r r_struct;

struct R{
    long val[P][P];
};
typedef struct R R_struct;

struct matrix{
    float val[P][P];
};
typedef struct matrix Matrix;

int abs(int x){
    if( x < 0 )
        return  x;
    return x;
}


void init_rR(r_struct* r, R_struct* R){
    int i,j;
    for(i = 0; i < P; ++i){
        r >val[i] = 0;
        for(j = 0; j < P; ++j){
            R>val[i][j] = 0;
        }
    }
}

void summ_rR(r_struct* r, R_struct* R, short* x){
    int i,j;
    for(i = 0; i < P; ++i){
        r->val[i] += x[0]*x[i+1];
        for(j = 0; j < P; ++j){
            R->val[i][j] += x[i+1]*x[j+1];
        }
    }
}

void Matrix_inverse(Matrix* m, Matrix* inv) {

    int i, j, k;
    float temp;

    for(i = 0; i < P; ++i)
        for(j = 0; j < P; ++j)
            if(i == j)
                inv->val[i][j] = 1;
            else
                inv->val[i][j] = 0;

    for(k = 0; k < P; ++k)
    {
        temp = m->val[k][k];
        for(j = 0; j < P; j++)
        {
            m->val[k][j] /= temp;
            inv->val[k][j] /= temp;
        }
        for(i = 0; i < P; i++)
        {
            temp = m->val[i][k];
            for(j = 0; j < P; j++)
            {
                if(i == k)
                    break;
                m->val[i][j]  = m->val[k][j] * temp;
                inv->val[i][j]  = inv->val[k][j] * temp;
            }
        }
    }

}

void matrix_mul_coef(const Matrix* m, const Coef* c, Coef* dest)
{
    int i,j;
    for(i = 0; i < P; ++i)
    {
        dest->val[i] = 0;
        for(j = 0; j < P; ++j)
        {
            dest->val[i] += m->val[i][j] * c->val[j];
        }
    }
}

void findCoeff(Coef* coef, const r_struct* r, const R_struct* R){
    Matrix R_matrix, inverse;
    Coef r_coef;
    int i,j;
    // convert the int based structures into float based
    for(i = 0; i < P; ++i)
    {
        r_coef.val[i] = r>val[i];
        for(j = 0; j < P; ++j)
        {
            R_matrix.val[i][j] = R>val[i][j];
            inverse.val[i][j] = 0;
        }
    }
    // get the inverse
    Matrix_inverse(&R_matrix, &inverse);
    // multiply
    matrix_mul_coef(&inverse, &r_coef, coef);
}

void getCoeffs(Coef* coefs, int count)
{
    unsigned long sample_count = 0;
    short x[P+1];
    float moving_average = 0;
    short state = 0;
    unsigned long start;
    int word_count = 0;
    r_struct r;
    R_struct R;

    fflush(stdout);

    while(1) {
        short sample = input_left_sample();
        int i; // shift our x's
        for(i = P; i >= 0;    i)
            x[i] = x[i 1];
        x[0] = sample; //save current

        // exponential moving average with a very small alpha
        moving_average = abs(sample)*ALPHA + moving_average*A1;

        // state machine
        switch(state){
            case 0:{
                if( moving_average >= MAG_THRESHOLD ){
                    // our average value is greater than the threshold, set starting time,
    change state
                    state = 1;
                    start = sample_count;

                    // initialize our r and R, then start accumulating them
                    init_rR(&r,&R);
                    summ_rR(&r, &R, x);
                }
                    break;
            }
            default:{
                // if exceding the threshold accumulate our R and r
                if( moving_average >= MAG_THRESHOLD ){
                    // just accumulate our rR's
                    summ_rR(&r, &R, x);

                    break;
                }

                // if not change state and find duration
                state = 0;
                int duration = sample_count   start;

                // check if this was long enough to be considered a word
                if( duration > TIME_THRESHOLD ){

                    // if so save it
                    int ms = duration*1000/SAMPLE_RATE;
                    printf("Word duration in samples: %i, in time: %ims. Coefs: ", duration,
    ms);

                    findCoeff(coefs+word_count, &r, &R);
                    int i;
                    for(i = 0; i< P; ++i)
                        printf("%f ",coefs[word_count].val[i]);

                    printf("\n");
                    fflush(stdout);

                    if(++word_count == count){
                        printf("Got Coeffs for %d words!\n",count);
                        fflush(stdout);
                        return;
                    }
                } //if( duration > TIME_THRESHOLD ){

            } // default:{
        }// switch(state)
        ++sample_count;
    } // while(1)
} //void getCoeffs(Coef* coefs, int count)

void getProfile(const Coef* words, Coef* mean, Matrix* cov, int word_count){
    int i,j,count;
    Coef temp;
    for(i = 0; i < P; ++i){
        mean >val[i] = 0;
        for(j = 0; j < P; ++j){
            cov >val[i][j] = 0;
        }
    }
    for(count = 0; count < word_count; ++count)
    {
        for(i = 0; i < P; ++i){
            mean >val[i] += words[count].val[i];
        }
    }
    printf("Mean: \n");
    for(i = 0; i < P; ++i){
        mean >val[i] /= word_count;
        printf("%f\t", mean >val[i]);
    }

    for(count = 0; count < word_count; ++count)
    {
        for(i = 0; i < P; ++i){
            temp.val[i] = words[count].val[i]   mean >val[i];
        }
        for(i = 0; i < P; ++i){
            for(j = 0; j < P; ++j){
                cov >val[i][j] += temp.val[i]*temp.val[j];
            }
        }
    }
    printf("\nCov: \n");
    for(i = 0; i < P; ++i){
        for(j = 0; j < P; ++j){
            cov >val[i][j] /= word_count;
            printf("%f\t", cov >val[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

float maholanobis_distance(const Coef* word, const Coef* mean, const Matrix* cov){
    Coef mid, temp;
    Matrix cov_temp, inverse;
    float ans = 0;
    int i,j;
    for(i = 0; i < P; ++i){
        mid.val[i] = word >val[i]   mean >val[i];
        for(j = 0; j < P; ++j){
            inverse.val[i][j] = 0;
            cov_temp.val[i][j] = cov >val[i][j];
        }
    }
    Matrix_inverse(&cov_temp, &inverse);

    for(i = 0; i < P; ++i){
        temp.val[i] = 0;
        for(j = 0; j < P; ++j){
            temp.val[i] += mid.val[j]*inverse.val[i][j];
        }
    }

    for(i = 0; i < P; ++i){
        ans += temp.val[i]*mid.val[i];
    }

    return ans;
}

int findUser(const Coef* word, const Coef* means, const Matrix* covs){
    float best_dist = 99999999999;
    int best_user = 0;
    int i;
    for(i = 0;i < NUM_USERS; ++i){
        float result = maholanobis_distance(word, means+i, covs+i);
        if(result < best_dist){
            best_dist = result;
            best_user = i;
        }
    }
    return best_user;
}
```

Listing 2.1 C program used to implement speaker recognition.

```c
#define TEST_SIZE 15
void main()
{
    comm_poll();

    Coef words[TEST_SIZE];
    Coef means[NUM_USERS];
    Matrix cov[NUM_USERS];


    while(1) {

        printf("******* SPEAKER RECOGNITION *******\n");
        printf("1   Training\n");
        printf("2   Testing\n");
        printf("Please enter your choice:\n");

        int choice = 0;

        scanf("%d",&choice);

        if(choice == 1) {

            int user = 0;
            printf("For which user do you want to train (1 %d): \n",NUM_USERS);
            scanf("%d",&user);
#ifdef SIMULATION
            switch(user){
                case 1: load("user1_train_8k.txt"); break;
                case 2: load("user2_train_8k.txt"); break;
                default: load("user3_train_8k.txt");
            }
#else
            int ready = 0;
            do {
                printf("Please provide the training sound, enter 1 when it is ready \n");
                scanf("%d",&ready);
            } while(ready != 1);
#endif
            printf("Training sound is sampling...\n");

            getCoeffs(words, TEST_SIZE);
            getProfile(words, means+user 1, cov+user 1, TEST_SIZE);

        } else if(choice == 2) {

            int user = 0;
            printf("Which user is speaking? (1 %d): ",NUM_USERS);
            scanf("%d",&user);
#ifdef SIMULATION
            switch(user){
                case 1: load("user1_test_8k.txt"); break;
                case 2: load("user2_test_8k.txt"); break;
                default: load("user3_test_8k.txt");
            }
#else
            int ready = 0;
            do {
                printf("Please provide the test sound, enter 1 when it is ready \n");
                scanf("%d",&ready);
            } while(ready != 1);
#endif
            printf("Test sound is sampling... \n");
            fflush(stdout);

            getCoeffs(words, TEST_SIZE);

            printf("Finding user... \n");

            int i;
            int correct = 0;
            for(i = 0; i < TEST_SIZE; ++i){
                int result = findUser(words+i,means,cov)+1;
                printf("Word %d: user: %d\n",i,result);
                if(result == user)
                    ++correct;
            }
            printf("\nCorrect %d out of %d\n", correct, TEST_SIZE);
        }
    }
}
```

Listing 2.2 Main C Program.