```
    typedef double XFLOAT
    typedef double OTA_FLOAT

    typedef double OTA_FLOAT
    typedef MAT_DCplx OTA_CPLX


{

typedef struct
{
    float FrameWeightWeight
    bool  UseRelDistance
    float ViterbiDistanceWeightFactor
} VITERBI_PARA

typedef struct
{
    long Samplerate
    int mSRDetectFineAlignCorrlen
    int mDelayFineAlignCorrlen
    int WindowSize[8]
    int CoarseAlignCorrlen[8]
    float pViterbiDistanceWeightFactor[8]
} SPEECH_WINDOW_PARA

typedef struct
{
    SPEECH_WINDOW_PARA Win[3]
    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    float FineAlignLowEnergyThresh
    float FineAlignLowEnergyCorrel
    float FineAlignShortDropOfCorrelR
    float FineAlignShortDropOfCorrelRLastBest
    float ViterbiDistanceWeightFactorDist
    float ViterbiDistanceWeightFactor
} SPEECH_TA_PARA

typedef struct
{
    SPEECH_WINDOW_PARA Win[3]
    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    float FineAlignLowEnergyThresh
    float FineAlignLowEnergyCorrel
    float FineAlignShortDropOfCorrelR
    float FineAlignShortDropOfCorrelRLastBest
    float ViterbiDistanceWeightFactorDist
    float ViterbiDistanceWeightFactor
} AUDIO_TA_PARA

typedef struct
{
    float mCorrForSkippingInitialDelaySearch
    int CoarseAlignSegmentLengthInMs
} GENERAL_TA_PARA

typedef struct
{
    void Init(long Samplerate)
    {
        if (Samplerate==16000)      MaxWin=4
        else if (Samplerate==8000)  MaxWin=4
```

```
        else                        MaxWin=4

        LowPeakEliminationThreshold= 0.2000000029802322

        if (Samplerate==16000)      PercentageRequired = 0.05F
        else if (Samplerate==8000)  PercentageRequired = 0.1F
        else                        PercentageRequired = 0.02F

        MaxDistance = 14

        MinReliability = 7

        PercentageRequired = 0.7
        OTA_FLOAT MaxGradient = 1.1
        OTA_FLOAT MaxTimescaling = 0.1

        if (Samplerate==48000)      MaxStepPerFrame = MaxGradient * 1024.0
        else if (Samplerate==8000)  MaxStepPerFrame = MaxGradient * 128.0
        MaxBins = ((int)(MaxStepPerFrame*2.0*0.9))
        MaxStepPerFrame *= 4

    }

    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    int     MaxStepPerFrame
    int     MaxBins
    int     MaxWin
    int     MinHistogramData

    float   MinReliability

    double  LowPeakEliminationThreshold
    float   MinFrequencyOfOccurrence
    float   LargeStepLimit

    float   MaxDistanceToLast
    float   MaxDistance
    float   MaxLargeStep

    float   ReliabilityThreshold
    float   PercentageRequired

    float   AllowedDistancePara2
    float   AllowedDistancePara3
} SR_ESTIMATION_PARA

class CParameters
{
    public:
        CParameters()
        {
            int i
            mTAPara.mCorrForSkippingInitialDelaySearch = 0.6F
            mTAPara.CoarseAlignSegmentLengthInMs = 600

            SPEECH_WINDOW_PARA      SpeechWinPara[] =
            {
                    {8000,   32, 32,
                        {128,  256, 128,  64,  32,  0, 0},
                        {-1,    -1,  -1,  86,  34,  0,  0},
                        {-1,    -1,  -1,  15,  12,  0,  0}},
                    {16000,  64, 64,
                        {256,  512, 256, 128,  64,  0},
                        {-1,    -1,  -1,  63,  33,  0},
                        {-1,    -1,  -1,  13,  10,  0}},
                    {48000, 256, 256,
```

```
                        {512, 1024, 512, 512, 128,  0},
                        {-1,    -1,  -1, 115,  61,  0},
                        {-1,    -1,  -1,  17,  16,  0}}
            }

            for (i=0  i<3  i++)
            {
                mSpeechTAPara.Win[i].Samplerate = SpeechWinPara[i].Samplerate
                mSpeechTAPara.Win[i].mDelayFineAlignCorrlen =
SpeechWinPara[i].mDelayFineAlignCorrlen
                mSpeechTAPara.Win[i].mSRDetectFineAlignCorrlen =
SpeechWinPara[i].mSRDetectFineAlignCorrlen
                for (int k=0  k<8  k++)
                {
                    mSpeechTAPara.Win[i].CoarseAlignCorrlen[k] =
SpeechWinPara[i].CoarseAlignCorrlen[k]
                    mSpeechTAPara.Win[i].WindowSize[k]         = SpeechWinPara[i].WindowSize[k]

                    mSpeechTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
SpeechWinPara[i].pViterbiDistanceWeightFactor[k]
                }
            }
            mSpeechTAPara.LowEnergyThresholdFactor = 15.0F
            mSpeechTAPara.LowCorrelThreshold = 0.4F
            mSpeechTAPara.FineAlignLowEnergyThresh = 2.0
            mSpeechTAPara.FineAlignLowEnergyCorrel = 0.6F
            mSpeechTAPara.FineAlignShortDropOfCorrelR = -1
            mSpeechTAPara.FineAlignShortDropOfCorrelRLastBest = 0.65F

            mSpeechTAPara.ViterbiDistanceWeightFactorDist = 5

            SPEECH_WINDOW_PARA      AudioWinPara[] =
            {
                    {8000,   32, 32,
                        {64,   128,  64,  64,  16,  0, 0},
                        {-1,    -1,  -1, 128,  32,  0, 0},
                        {-1,    -1,  -1,   6,   6,  0, 0}},
                    {16000,  64, 64,
                        {128,  256, 128, 128,  32,  0},
                        {-1,    -1,  -1,  64,  32,  0},
                        {-1,    -1,  -1,  12,  12,  0}},
                    {48000, 256, 2048,
                        {512, 1024, 512, 512, 256,  128,   0},
                        {-1,    -1,  -1, 512, 1024, 2048,  0},
                        {-1,    -1,  -1,  16,  16,  32,    0}}
            }

            for (i=0  i<3  i++)
            {
                mAudioTAPara.Win[i].Samplerate = AudioWinPara[i].Samplerate
                mAudioTAPara.Win[i].mDelayFineAlignCorrlen    =
AudioWinPara[i].mDelayFineAlignCorrlen
                mAudioTAPara.Win[i].mSRDetectFineAlignCorrlen =
AudioWinPara[i].mSRDetectFineAlignCorrlen
                for (int k=0  k<8  k++)
                {
                    mAudioTAPara.Win[i].CoarseAlignCorrlen[k] =
AudioWinPara[i].CoarseAlignCorrlen[k]
                    mAudioTAPara.Win[i].WindowSize[k]         = AudioWinPara[i].WindowSize[k]
                    mAudioTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
AudioWinPara[i].pViterbiDistanceWeightFactor[k]
                }
            }
            mAudioTAPara.LowEnergyThresholdFactor = 1
            mAudioTAPara.LowCorrelThreshold = 0.85F
            mAudioTAPara.FineAlignLowEnergyThresh = 32.0
            mAudioTAPara.FineAlignLowEnergyCorrel = 0.8F
            mAudioTAPara.FineAlignShortDropOfCorrelR = -1
```

```
            mAudioTAPara.FineAlignShortDropOfCorrelRLastBest = 0.8F
            mAudioTAPara.ViterbiDistanceWeightFactorDist = 6

            mSREPara.LowEnergyThresholdFactor = 15.0F
            mSREPara.LowCorrelThreshold = 0.4F

            mSREPara.MaxStepPerFrame = 160
            mSREPara.MaxBins = ((int)(mSREPara.MaxStepPerFrame*2.0*0.9))

            mSREPara.MaxWin=4
            mSREPara.LowPeakEliminationThreshold=0.2000000029802322F
            mSREPara.PercentageRequired = 0.04F

            mSREPara.LargeStepLimit = 0.08F
            mSREPara.MaxDistanceToLast = 7
            mSREPara.MaxLargeStep = 5
            mSREPara.MaxDistance = 14

            mSREPara.MinReliability = 7
            mSREPara.MinFrequencyOfOccurrence = 3

            mSREPara.AllowedDistancePara2 = 0.85F
            mSREPara.AllowedDistancePara3 = 1.5F

            mSREPara.ReliabilityThreshold = 0.3F
            mSREPara.MinHistogramData = 8

            mViterbi.UseRelDistance = false
            mViterbi.FrameWeightWeight = 1.0F
        }

        void Init(long Samplerate)
        {
            mSREPara.Init(Samplerate)
        }

        VITERBI_PARA        mViterbi
        GENERAL_TA_PARA     mTAPara
        SPEECH_TA_PARA      mSpeechTAPara
        AUDIO_TA_PARA       mAudioTAPara
        SR_ESTIMATION_PARA  mSREPara
}
}


{

class CProcessData
{
    public:
        CProcessData()
        {
            int i

            mCurrentIteration = -1
            mStartPlotIteration=10
            mLastPlotIteration =10
            mEnablePlotting=false
            mpLogFile = 0

            mWindowSize = 2048
            mSRDetectFineAlignCorrlen = 1024
            mDelayFineAlignCorrlen = 1024
            mOverlap    = 1024
            mSamplerate = 48000
            mNumSignals = 0
            mpMathlibHandle = 0
            mMinLowVarDelay = -99999999
```

```
            mMaxHighVarDelay = 99999999

            mMinStaticDelayInMs = -2500
            mMaxStaticDelayInMs = 2500

            mMaxToleratedRelativeSamplerateDifference = 1.0

            for (i=0  i<8  i++)
                mpViterbiDistanceWeightFactor[i] = 0.0001F
        }

        int mMinStaticDelayInMs
        int mMaxStaticDelayInMs

        int mMinLowVarDelayInSamples
        int mMaxHighVarDelayInSamples

        int mStartPlotIteration
        int mLastPlotIteration
        bool mEnablePlotting
        long mSamplerate

        FILE* mpLogFile

        int mCurrentIteration

        int mpWindowSize[8]

        int mpOverlap[8]

        int mpCoarseAlignCorrlen[8]

        float mpViterbiDistanceWeightFactor[8]

        int mDelayFineAlignCorrlen
        int mSRDetectFineAlignCorrlen
        float mMaxToleratedRelativeSamplerateDifference
        int mWindowSize

        int mOverlap

        int mCoarseAlignCorrlen

        int mNumSignals
        void* mpMathlibHandle

        int mMinLowVarDelay
        int mMaxHighVarDelay
        int mStepSize

        bool Init(int Iteration, float MoreDownsampling)
        {
            assert(MoreDownsampling)

            mCurrentIteration = Iteration
            mP.Init(mSamplerate)

            mWindowSize = (int)((float)mpWindowSize[Iteration]*MoreDownsampling)
            mOverlap = (int)((float)mpOverlap[Iteration]*MoreDownsampling)
            mCoarseAlignCorrlen = mpCoarseAlignCorrlen[Iteration]
            mStepSize = mWindowSize - mOverlap
            mMinLowVarDelay = mMinLowVarDelayInSamples / mStepSize
            mMaxHighVarDelay = mMaxHighVarDelayInSamples / mStepSize

            float D = mpViterbiDistanceWeightFactor[Iteration]
            D = D * mSamplerate / mStepSize / 1000
            float F = ((float)log(1+0.5)) / (D*D)
            mP.mViterbi.ViterbiDistanceWeightFactor = F
```

```
332
333             D = mP.mSpeechTAPara.ViterbiDistanceWeightFactorDist
334             D = D * mSamplerate / 1000
335             F =((float) log(1+0.5) / (D*D))
336             mP.mSpeechTAPara.ViterbiDistanceWeightFactor = F
337
338             return true
339         }
340
341         CParameters   mP
342 }
343
344 class SECTION
345 {
346     public:
347         int Start
348         int End
349         int Len() {return End-Start }
350         void CopyFrom(const SECTION &src)
351         {
352             this->Start = src.Start
353             this->End   = src.End
354         }
355 }
356
357 typedef struct OTA_RESULT
358 {
359     void CopyFrom(const OTA_RESULT* src)
360     {
361         mNumFrames           = src->mNumFrames
362         mStepsize            = src->mStepsize
363         mResolutionInSamples = src->mResolutionInSamples
364         if (src->mpDelay != NULL && mNumFrames > 0)
365         {
366             matFree(mpDelay)
367             mpDelay = (long*)matMalloc(mNumFrames * sizeof(long))
368             for (int i = 0  i < mNumFrames  i++)
369                 mpDelay[i] = src->mpDelay[i]
370         }
371         else
372         {
373             matFree(mpDelay)
374             mpDelay = NULL
375         }
376
377         if (src->mpReliability != NULL && mNumFrames > 0)
378         {
379             matFree(mpReliability)
380             mpReliability = (OTA_FLOAT*)matMalloc(mNumFrames * sizeof(OTA_FLOAT))
381             for (int i = 0  i < mNumFrames  i++)
382                 mpReliability[i] = src->mpReliability[i]
383         }
384         else
385         {
386             matFree(mpReliability)
387             mpReliability = NULL
388         }
389         mAvgReliability   = src->mAvgReliability
390         mRelSamplerateDev = src->mRelSamplerateDev
391
392         mNumUtterances = src->mNumUtterances
393         if (src->mpStartSampleUtterance != NULL && mNumUtterances > 0)
394         {
395             matFree(mpStartSampleUtterance)
396             mpStartSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
397             for (int i = 0  i < mNumUtterances  i++)
398                 mpStartSampleUtterance[i] = src->mpStartSampleUtterance[i]
399         }
```

```
400         else
401         {
402             matFree(mpStartSampleUtterance)
403             mpStartSampleUtterance = NULL
404         }
405         if (src->mpStopSampleUtterance != NULL && mNumUtterances > 0)
406         {
407             matFree(mpStopSampleUtterance)
408             mpStopSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
409             for (int i = 0  i < mNumUtterances  i++)
410                 mpStopSampleUtterance[i] = src->mpStopSampleUtterance[i]
411         }
412         else
413         {
414             matFree(mpStopSampleUtterance)
415             mpStopSampleUtterance = NULL
416         }
417         if (src->mpDelayUtterance != NULL && mNumUtterances > 0)
418         {
419             matFree(mpDelayUtterance)
420             mpDelayUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
421             for (int i = 0  i < mNumUtterances  i++)
422                 mpDelayUtterance[i] = src->mpDelayUtterance[i]
423         }
424         else
425         {
426             matFree(mpDelayUtterance)
427             mpDelayUtterance = NULL
428         }
429 
430         mNumSections = src->mNumSections
431         if (src->mpRefSections != NULL && mNumSections > 0)
432         {
433             delete[] mpRefSections
434             mpRefSections = new SECTION[mNumSections]
435             for (int i = 0  i < mNumSections  i++)
436                 mpRefSections[i].CopyFrom(src->mpRefSections[i])
437         }
438         else
439         {
440             delete[] mpRefSections
441             mpRefSections = NULL
442         }
443         if (src->mpDegSections != NULL && mNumSections > 0)
444         {
445             delete[] mpDegSections
446             mpDegSections = new SECTION[mNumSections]
447             for (int i = 0  i < mNumSections  i++)
448                 mpDegSections[i].CopyFrom(src->mpDegSections[i])
449         }
450         else
451         {
452             delete[] mpDegSections
453             mpDegSections = NULL
454         }
455 
456         mSNRRefdB = src->mSNRRefdB
457         mSNRDegdB = src->mSNRDegdB
458         mNoiseLevelRef = src->mNoiseLevelRef
459         mNoiseLevelDeg = src->mNoiseLevelDeg
460         mSignalLevelRef = src->mSignalLevelRef
461         mSignalLevelDeg = src->mSignalLevelDeg
462         mNoiseThresholdRef = src->mNoiseThresholdRef
463         mNoiseThresholdDeg = src->mNoiseThresholdDeg
464 
465         if (src->mpActiveFrameFlags != NULL && mNumFrames > 0)
466         {
467             matFree(mpActiveFrameFlags)
```

```
            mpActiveFrameFlags = (int*)matMalloc(mNumFrames * sizeof(int))
            for (int i = 0  i < mNumFrames  i++)
                mpActiveFrameFlags[i] = src->mpActiveFrameFlags[i]
        }
        else
        {
            matFree(mpActiveFrameFlags)
            mpActiveFrameFlags = NULL
        }

        if (src->mpIgnoreFlags != NULL && mNumFrames > 0)
        {

            matFree(mpIgnoreFlags)
            mpIgnoreFlags = (int*)matMalloc(mNumFrames * sizeof(int))
            mNumIngoreFlags = src->mNumIngoreFlags
            for (int i = 0  i < mNumFrames  i++)
                mpIgnoreFlags[i] = src->mpIgnoreFlags[i]
        }
        else
        {
            matFree(mpIgnoreFlags)
            mpIgnoreFlags = NULL
        }

        for (int i = 0  i < 5  i++)
            mTimeDiffs[i] = src->mTimeDiffs[i]

        mAslFrames = src->mAslFrames
        mAslFramelength = src->mAslFramelength
        if (src->mpAslActiveFrameFlags != NULL && mAslFrames > 0)
        {
            matFree(mpAslActiveFrameFlags)
            mpAslActiveFrameFlags = (int*)matMalloc(mAslFrames * sizeof(int))
            for (int i = 0  i < mAslFrames  i++)
                mpAslActiveFrameFlags[i] = src->mpAslActiveFrameFlags[i]
        }
        else
        {
            matFree(mpAslActiveFrameFlags)
            mpAslActiveFrameFlags = NULL
        }

        mAslFramesDeg = src->mAslFramesDeg
        if (src->mpAslActiveFrameFlagsDeg != NULL && mAslFramesDeg > 0)
        {
            matFree(mpAslActiveFrameFlagsDeg)
            mpAslActiveFrameFlagsDeg = (int*)matMalloc(mAslFramesDeg * sizeof(int))
            for (int i = 0  i < mAslFramesDeg  i++)
                mpAslActiveFrameFlagsDeg[i] = src->mpAslActiveFrameFlagsDeg[i]
        }
        else
        {
            matFree(mpAslActiveFrameFlagsDeg)
            mpAslActiveFrameFlagsDeg = NULL
        }

        FirstRefSample = src->FirstRefSample
        FirstDegSample = src->FirstDegSample
    }

    OTA_RESULT()
    {
        mNumFrames = 0
        mpDelay = NULL

        mpReliability = NULL

```

```
536         mNumUtterances = 0
537         mpStartSampleUtterance = NULL
538         mpStopSampleUtterance  = NULL
539         mpDelayUtterance       = NULL
540
541         mNumSections = 0
542         mpRefSections = NULL
543         mpDegSections = NULL
544
545         mpActiveFrameFlags = NULL
546         mpIgnoreFlags = NULL
547         mNumIngoreFlags = 0
548
549         mAslFramelength = 0
550         mAslFrames = 0
551         mpAslActiveFrameFlags = NULL
552         mAslFramesDeg = 0
553         mpAslActiveFrameFlagsDeg = NULL
554
555         FirstRefSample = FirstDegSample = 0
556     }
557
558     ~OTA_RESULT()
559     {
560         matFree(mpDelay)
561         mpDelay = NULL
562
563         matFree(mpReliability)
564         mpReliability = NULL
565
566         matFree(mpStartSampleUtterance)
567         mpStartSampleUtterance = NULL
568
569         matFree(mpStopSampleUtterance)
570         mpStopSampleUtterance  = NULL
571
572         matFree(mpDelayUtterance)
573         mpDelayUtterance       = NULL
574
575         delete[] mpRefSections
576         mpRefSections = NULL
577         delete[] mpDegSections
578         mpDegSections = NULL
579
580         matFree(mpActiveFrameFlags)
581         mpActiveFrameFlags = NULL
582
583         matFree(mpIgnoreFlags)
584         mpIgnoreFlags = NULL
585
586         matFree(mpAslActiveFrameFlags)
587         mpAslActiveFrameFlags = NULL
588         matFree(mpAslActiveFrameFlagsDeg)
589         mpAslActiveFrameFlagsDeg = NULL
590     }
591
592     long mNumFrames
593     int mStepsize
594     int mResolutionInSamples
595     int mPitchFrameSize
596     long *mpDelay
597     OTA_FLOAT *mpReliability
598     OTA_FLOAT mAvgReliability
599     OTA_FLOAT mRelSamplerateDev
600
601     int  mNumUtterances
602     int* mpStartSampleUtterance
603     int* mpStopSampleUtterance
```

```
    int* mpDelayUtterance
    int FirstRefSample
    int FirstDegSample

    int         mNumSections
    SECTION     *mpRefSections
    SECTION     *mpDegSections

    double mSNRRefdB, mSNRDegdB
    double mNoiseLevelRef, mNoiseLevelDeg
    double mSignalLevelRef, mSignalLevelDeg
    double mNoiseThresholdRef, mNoiseThresholdDeg

    int *mpActiveFrameFlags

    int *mpIgnoreFlags
    int mNumIngoreFlags
    int mAslFrames
    int mAslFramelength
    int *mpAslActiveFrameFlags
    int mAslFramesDeg
    int *mpAslActiveFrameFlagsDeg

    double mTimeDiffs[5]

}OTA_RESULT

struct FilteringParameters
{
    int pListeningCondition
    double cutOffFrequencyLow
    double cutOffFrequencyHigh
    double disturbedEnergyQuotient
}

class ITempAlignment
{
    public:

        virtual bool Init(CProcessData* pProcessData)=0
        virtual void Free()=0
        virtual void Destroy()=0

        virtual bool SetSignal(int Index, unsigned long SampleRate, unsigned long NumSamples, int
NumChannels, OTA_FLOAT** pSignal)=0

        virtual void GetFilterCharacteristics(FilteringParameters *FilterParams)=0

        virtual bool FilterSignal(int Index, FilteringParameters *FilterParams)=0

        virtual bool Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)=0

        virtual void GetNoiseSwitching(OTA_FLOAT* pBGNSwitchingLevel, OTA_FLOAT*
pNoiseLevelSpeechDeg, OTA_FLOAT* pNoiseLevelSilenceDeg)=0

        virtual OTA_FLOAT GetPitchFreq(int Signal, int Channel)=0

        virtual OTA_FLOAT GetPitchVector(int Signal, int Channel, OTA_FLOAT* pVector, int NumFrames,
int SamplesPerFrame)=0
        virtual int GetPitchFrameSize()=0
}

enum AlignmentType
{
    TA_FOR_SPEECH=0,

}
```

```
ITempAlignment* CreateAlignment(AlignmentType Type)

}


{

void CCorrelationMatrix::Free()
{
    if (mpCorrMatrix)
    {

    }

    if (mpNormMatrix)
    {

        mpNormMatrix = 0
    }
}

OTA_FLOAT CorrelationWithHistogramAndWindow(OTA_FLOAT* pSearchThis, OTA_FLOAT* pSearchHere, int Corrlen, int HistogramLen, int HistogramStep, OTA_FLOAT* pWindowed1, OTA_FLOAT* pWindowed2, OTA_FLOAT* HannWin, OTA_FLOAT* pCorrNorm)
{

    OTA_FLOAT Histogram = 0

    if(HannWin)
    {
        for (int i=0  i<HistogramLen  i++)
        {
            matbMpy3(HannWin, pSearchThis+i, pWindowed1, Corrlen)
            matbMpy3(HannWin, pSearchHere+i, pWindowed2, Corrlen)
            if (i==0)
                Histogram += matPearsonCorrelation2(pWindowed1, pWindowed2, Corrlen, pCorrNorm)
            else
                Histogram += matPearsonCorrelation(pWindowed1, pWindowed2, Corrlen)
        }
    }
    else
    {
        for (int i=0  i<HistogramLen  i++)
        {
            matWinHann(pSearchThis+i, pWindowed1, Corrlen)
            matWinHann(pSearchHere+i, pWindowed2, Corrlen)
            if (i==0)
                Histogram += matPearsonCorrelation2(pWindowed1, pWindowed2, Corrlen, pCorrNorm)
            else
                Histogram += matPearsonCorrelation(pWindowed1, pWindowed2, Corrlen)
        }
    }

    Histogram /= HistogramLen

    if (Histogram>0.7)
    {
        OTA_FLOAT Factor = 1.5
        Factor = Factor*Factor*Corrlen
        Factor = 1000
        OTA_FLOAT E1 = matDotProd(pWindowed1, pWindowed1, Corrlen)
        OTA_FLOAT E2 = matDotProd(pWindowed2, pWindowed2, Corrlen)
        if (E1>Factor*E2)
            Histogram = -0.001
        if (E2>Factor*E1)
            Histogram = -0.001
    }

```

```
735     return Histogram
736 }
737
738 typedef struct
739 {
740     bool PlotThisFrame
741     int ThisFrameNum
742     int CurrentStepSize
743     int CurrentFeatureNum
744 } PLOT_INFO
745
746 double CalcCorrelationForDelayRange(OTA_FLOAT *mpCorrDelayVec, OTA_FLOAT *mpCorrNormVec, OTA_FLOAT
    *pBuffer1, OTA_FLOAT *pBuffer2, OTA_FLOAT* HannWin, CProcessData* pProcessData,
747                                     OTA_FLOAT* pFVecRef, long ffFeatureVectorlengthRef, OTA_FLOAT*
    pFVecDeg, long ffFeatureVectorlengthDeg,
748                                     long ffLastRefStart, long ffNextRefStartIndex, long
    ffLastDegStart, int ffDegIndex, int DelayLowIndex, int
    DelayHighIndex, int &maxIndex, int LowLim, int HighLim,
749                                     PLOT_INFO* pPlotInfo)
750 {
751     const int MaxHistogramLength = 16
752     double  MaxCorr = 0
753     int     MaxPos = 0
754     int     ffHistogramLen = 1
755     double  CorrNow
756     long    ffIndex
757
758     int PatternLength = pProcessData->mCoarseAlignCorrlen
759
760     for (int ffd = LowLim  ffd<HighLim  ffd++)
761
762     {
763         int ffHistogramOffset = 0
764         int ffHistogramStep = 1
765
766         CorrNow = 0
767         ffIndex = ffNextRefStartIndex + ffd
768         ffHistogramLen = (((ffLastRefStart-ffIndex) < (ffLastDegStart-ffDegIndex)) ?
    (ffLastRefStart-ffIndex) : (ffLastDegStart-ffDegIndex))
769         ffHistogramLen = (((ffHistogramLen) < (MaxHistogramLength*ffHistogramStep)) ?
    (ffHistogramLen) : (MaxHistogramLength*ffHistogramStep))
770
771         ffHistogramOffset = -ffHistogramLen/2
772         ffHistogramOffset = (((ffHistogramOffset) > (-ffDegIndex)) ? (ffHistogramOffset) :
    (-ffDegIndex))
773         ffHistogramOffset = (((ffHistogramOffset) > (-ffIndex)) ? (ffHistogramOffset) : (-ffIndex))
774
775         if (ffDegIndex+ffHistogramLen+ffHistogramOffset+PatternLength>ffFeatureVectorlengthDeg ||
    ffIndex+ffHistogramLen+ffHistogramOffset+PatternLength>ffFeatureVectorlengthRef)
776             OutputDebugString("DelaySeach.pdf: CCorrelationMatrix::CreateMatrix(), unexpected
    condition!\n")
777
778         ffHistogramLen /= ffHistogramStep
779
780         if (ffHistogramLen==0) ffHistogramLen=1
781
782         mpCorrDelayVec[ffd] = CorrNow =
    CorrelationWithHistogramAndWindow(&pFVecDeg[ffDegIndex+ffHistogramOffset],
    &pFVecRef[ffIndex+ffHistogramOffset], PatternLength, ffHistogramLen, ffHistogramStep,
    pBuffer1, pBuffer2, HannWin, &mpCorrNormVec[ffd])
783
784     }
785
786     for (int k=LowLim  k<DelayLowIndex  k++)
787         mpCorrDelayVec[k] =0
788     for (int k=DelayHighIndex  k<HighLim  k++)
789         mpCorrDelayVec[k] =0
790     MaxCorr = -1
```

```
791     for (int k=DelayLowIndex  k<DelayHighIndex  k++)
792     {
793         if (MaxCorr<mpCorrDelayVec[k])
794         {
795             MaxCorr = mpCorrDelayVec[k]
796             MaxPos = k
797         }
798     }
799
800     maxIndex = MaxPos
801     return MaxCorr
802 }
803
804 //Create the correlation matrix for one feature pair (one signal pair, one channel, one feature)
805 //The structure is as follows:
806 //
807 // - This vector contains for each element of the underlying feature vectors
808 //   one vector with the correlation of all possible delay lags between mMinLowVarDelay and
    mMaxHighVarDelay.
809 //
810 //   M[DegFrameIndex][RefFrameIndex] = Correlation(&FeatureVecDeg[DegFrameIndex],
    &FeatureVecRef[RefFrameIndex], mProcessData.mCoarseAlignCorrlen)
811 //      with: DegFrameIndex being the index of any frame [0 FeatureVectorlengthDeg-1] and
812 //            RefFrameIndex being the index of any frame [0 -mMinLowVarDelay + mMaxHighVarDelay]
813 //               around the frame at the average delay which is found at index -mMinLowVarDelay.
814 //        RefFrameIndex can also be interpreted as the delay for frame DegFrameIndex relative to the
    average delay.
815 //
816 //
817 //Data are stored for each DegStep frames of the degraded signal only.
818 //Degstep is also used for reading pAvgDelayInFrames (which contains one value for every DegStep
    frames) and
819 //the length of the result vector
820 //NOTE: local variables starting with "ff" are related to feature frames and others are related to
    TA frames
821 bool CCorrelationMatrix::CreateMatrix(CFeature* pFeature, int Channel, int* pActiveFrameFlags,
    CProcessData* pProcessData, int NumMacroFrames, int* pSearchRangeLow, int* pSearchRangeHigh,
    OTA_FLOAT* pPitchVec, int DegStep, long* pAvgDelayInFrames, int CurrentFeatureIndex)
822 {
823     bool rc = true
824
825     mProcessData = *pProcessData
826     mpFeature = pFeature
827     mMacroFrameSize = mProcessData.mStepSize * DegStep
828
829     long ffFeatureVectorlengthRef = mpFeature->GetFVector(0, Channel)->mSize
830     long ffFeatureVectorlengthDeg = mpFeature->GetFVector(1, Channel)->mSize
831
832     long ffMaxFeatureVectorlength = (((ffFeatureVectorlengthRef) > (ffFeatureVectorlengthDeg)) ?
    (ffFeatureVectorlengthRef) : (ffFeatureVectorlengthDeg))
833     mNumMacroFrames = NumMacroFrames
834     mMinLowVarDelay   = mProcessData.mMinLowVarDelay
835     mMaxHighVarDelay  = mProcessData.mMaxHighVarDelay
836     mCorrelationVectorlength = -mMinLowVarDelay + mMaxHighVarDelay + 1
837
838     if (mProcessData.mpLogFile)
839     {
840         fprintf(mProcessData.mpLogFile,
    "CCorrelationMatrix::CreateMatrix(ffFeatureVectorlengthRef=%ld,
    ffFeatureVectorlengthDeg=%ld, DegStep=%d)\n", ffFeatureVectorlengthRef,
    ffFeatureVectorlengthDeg, DegStep)
841         fprintf(mProcessData.mpLogFile, "\tAllocating correlation matrix of size %d x %d\n",
    mNumMacroFrames, mCorrelationVectorlength)
842         fflush(mProcessData.mpLogFile)
843     }
844
845     if (mNumMacroFrames<=0)
846     {
```

```
847         OutputDebugString("CCorrelationMatrix::CreateMatrix(): ERROR, FeatureVectorLen is <= 0!\n")
848         exit(1)
849     }
850
851     mpCorrMatrix = (OTA_FLOAT**)matMalloc2D(mNumMacroFrames, mCorrelationVectorlength *
    sizeof(OTA_FLOAT))
852     mpNormMatrix = (OTA_FLOAT**)matMalloc2D(mNumMacroFrames, mCorrelationVectorlength *
    sizeof(OTA_FLOAT))
853     rc = (mpCorrMatrix!=0)
854
855     if (!rc || !mpNormMatrix)
856     {
857         OutputDebugString("CCorrelationMatrix::CreateMatrix(): ERROR, problem with memory allocation
    for mpCorrMatrix!\n")
858         Free()
859     }
860
861     if (rc)
862     {
863         long ffLastRefStart = ffFeatureVectorlengthRef - mProcessData.mCoarseAlignCorrlen-1
864         long ffLastDegStart = ffFeatureVectorlengthDeg - mProcessData.mCoarseAlignCorrlen-1
865         OTA_FLOAT* pFVecRef=0
866         OTA_FLOAT* pFVecDeg=0
867
868         if (mpFeature->GetNumSets()>1)
869         {
870             pFVecRef = mpFeature->GetFVector(0, Channel, 0)->mpVector
871             pFVecDeg = mpFeature->GetFVector(1, Channel, 0)->mpVector
872         }
873         else
874         {
875             pFVecRef = mpFeature->GetFVector(0, Channel, 0)->mpVector
876             pFVecDeg = mpFeature->GetFVector(1, Channel, 0)->mpVector
877         }
878
879         int ffWindowOffset = mProcessData.mCoarseAlignCorrlen / 2
880
881         OTA_FLOAT* pBuffer1 = (OTA_FLOAT*)matMalloc(mProcessData.mCoarseAlignCorrlen *
    sizeof(OTA_FLOAT))
882         OTA_FLOAT* pBuffer2 = (OTA_FLOAT*)matMalloc(mProcessData.mCoarseAlignCorrlen *
    sizeof(OTA_FLOAT))
883
884         OTA_FLOAT* HannWin = (OTA_FLOAT*)matMalloc(mProcessData.mCoarseAlignCorrlen *
    sizeof(OTA_FLOAT))
885
886         matbSet((OTA_FLOAT)1.0, pBuffer1, mProcessData.mCoarseAlignCorrlen)
887         matWinHann(pBuffer1, HannWin, mProcessData.mCoarseAlignCorrlen)
888
889         int MaxOptDelayIndex=mCorrelationVectorlength-1
890         int MinOptDelayIndex=0
891
892         static int PlotFrameNum = -275
893         PLOT_INFO PlotInfo
894         PlotInfo.PlotThisFrame = false
895         PlotInfo.ThisFrameNum = 0
896         PlotInfo.CurrentStepSize = mProcessData.mStepSize
897         PlotInfo.CurrentFeatureNum = CurrentFeatureIndex
898
899         OTA_FLOAT MaxCorr = 0
900         int MaxPos = 0
901         int SecondLastMaxPos = 0
902         for (int f=0  f<mNumMacroFrames  f++)
903         {
904
905             {
906                 int ffDegIndex = f*DegStep-ffWindowOffset
907
908                 if (ffDegIndex<0) ffDegIndex += ffWindowOffset
```

```
909
910                 long ffRefIndex = -9999999
911                 int DelayIndexHigh = -9999999
912                 int DelayIndexLow = -9999999
913
914                 if (ffDegIndex<ffLastDegStart)
915                 {
916                     //Calculate the indices for the feature frame vectors
917                     //All vectors are allocated with the maximum length of the feature, but not all
    data may be valid!
918                     //The "real" best found delay for frame f should be between index
    mMinLowVarDelay and mMaxHighVarDelay+1
919                     //in the correlation vector of frame f, in the middle of the vector.
920
921                     ffRefIndex = ffDegIndex+mMinLowVarDelay+pAvgDelayInFrames[f]
922                     int SearchRange = -mMinLowVarDelay + mMaxHighVarDelay + 1
923
924                     int AbsMaxDelayIndex = (((ffLastRefStart-ffRefIndex) < (SearchRange)) ?
    (ffLastRefStart-ffRefIndex) : (SearchRange))
925
926                     int AbsMinDelayIndex = (((0) > (-ffRefIndex)) ? (0) : (-ffRefIndex))
927
928                     int MaxDelayIndex = pSearchRangeHigh[f]-mMinLowVarDelay+1
929
930                     MaxDelayIndex = (((AbsMaxDelayIndex) < (MaxDelayIndex)) ? (AbsMaxDelayIndex) :
    (MaxDelayIndex))
931                     MaxDelayIndex = (((AbsMinDelayIndex) > (MaxDelayIndex)) ? (AbsMinDelayIndex) :
    (MaxDelayIndex))
932
933                     int MinDelayIndex = pSearchRangeLow[f]-mMinLowVarDelay
934                     MinDelayIndex = (((AbsMaxDelayIndex) < (MinDelayIndex)) ? (AbsMaxDelayIndex) :
    (MinDelayIndex))
935                     MinDelayIndex = (((AbsMinDelayIndex) > (MinDelayIndex)) ? (AbsMinDelayIndex) :
    (MinDelayIndex))
936
937                     DelayIndexHigh = (((MaxDelayIndex) < (MaxOptDelayIndex)) ? (MaxDelayIndex) :
    (MaxOptDelayIndex))
938                     DelayIndexHigh = (((AbsMaxDelayIndex) < (DelayIndexHigh)) ? (AbsMaxDelayIndex) :
    (DelayIndexHigh))
939                     DelayIndexHigh = (((AbsMinDelayIndex) > (DelayIndexHigh)) ? (AbsMinDelayIndex) :
    (DelayIndexHigh))
940
941                     DelayIndexLow = (((MinDelayIndex) > (MinOptDelayIndex)) ? (MinDelayIndex) :
    (MinOptDelayIndex))
942                     DelayIndexLow = (((AbsMaxDelayIndex) < (DelayIndexLow)) ? (AbsMaxDelayIndex) :
    (DelayIndexLow))
943                     DelayIndexLow = (((AbsMinDelayIndex) > (DelayIndexLow)) ? (AbsMinDelayIndex) :
    (DelayIndexLow))
944
945                     if (DelayIndexLow>DelayIndexHigh)
946                         DelayIndexLow = DelayIndexHigh
947
948                     if (DelayIndexLow==DelayIndexHigh)
949                     {
950                         DelayIndexHigh++
951
952                         //Check the bounds. If those are exceeded, extend the search range towards
    the lower end.
953
954                     }
955
956                     //Preset the entire vector with zero
957
958                     int UsedLimitLow = DelayIndexLow
959                     int UsedLimitHigh = DelayIndexHigh
960                     if (DelayIndexLow<DelayIndexHigh && ffRefIndex+DelayIndexLow>=0)
961                     {
962                         PlotInfo.PlotThisFrame = (f==PlotFrameNum &&
```

```
    mProcessData.mCurrentIteration==3)
963                         PlotInfo.ThisFrameNum = f
964                         MaxCorr = CalcCorrelationForDelayRange(mpCorrMatrix[f], mpNormMatrix[f],
    pBuffer1, pBuffer2, HannWin, pProcessData,
965                                                pFVecRef, ffFeatureVectorlengthRef, pFVecDeg,
    ffFeatureVectorlengthDeg,
966                                                ffLastRefStart, ffRefIndex, ffLastDegStart, ffDegIndex,
    DelayIndexLow, DelayIndexHigh, MaxPos, AbsMinDelayIndex,
    AbsMaxDelayIndex, &PlotInfo)
967
968                         {
969
970                             if (MaxCorr > 0.98 && SecondLastMaxPos==MaxPos)
971
972                             {
973                                 MinOptDelayIndex = (((0) > (MaxPos-10)) ? (0) : (MaxPos-10))
974                                 MaxOptDelayIndex = (((mCorrelationVectorlength) < (MaxPos+10)) ?
    (mCorrelationVectorlength) : (MaxPos+10))
975                              }
976                              else if (DelayIndexLow > MinDelayIndex || DelayIndexHigh <
    MaxDelayIndex)
977                              {
978                                 int TempMaxPos
979                                 double TempMaxCorr
980                                 TempMaxCorr = CalcCorrelationForDelayRange(mpCorrMatrix[f],
    mpNormMatrix[f], pBuffer1, pBuffer2, HannWin, pProcessData,
981                                                    pFVecRef, ffFeatureVectorlengthRef, pFVecDeg,
    ffFeatureVectorlengthDeg,
982
983                                                    ffLastRefStart, ffRefIndex, ffLastDegStart, ffDegIndex,
    MinDelayIndex, DelayIndexHigh, TempMaxPos,
    DelayIndexHigh, MaxDelayIndex, &PlotInfo)
984
985                                 UsedLimitLow = MinDelayIndex
986
987                                 if (TempMaxCorr>MaxCorr)
988                                 {
989                                     MaxCorr = TempMaxCorr
990                                     MaxPos = TempMaxPos
991                                 }
992
993                                 TempMaxCorr = CalcCorrelationForDelayRange(mpCorrMatrix[f],
    mpNormMatrix[f], pBuffer1, pBuffer2, HannWin, pProcessData,
994                                                    pFVecRef, ffFeatureVectorlengthRef, pFVecDeg,
    ffFeatureVectorlengthDeg,
995                                                    ffLastRefStart, ffRefIndex, ffLastDegStart, ffDegIndex,
    DelayIndexHigh, MaxDelayIndex, TempMaxPos,
    DelayIndexHigh, MaxDelayIndex, &PlotInfo)
996                                 UsedLimitHigh = MaxDelayIndex
997
998                                 if (TempMaxCorr>MaxCorr)
999                                 {
1000                                    MaxCorr = TempMaxCorr
1001                                    MaxPos = TempMaxPos
1002                                }
1003
1004                                //Reset search range for next frame
1005
1006                            }
1007                        }
1008                        SecondLastMaxPos = MaxPos
1009                    }
1010
1011                }
1012                else
1013                {
1014                    //Set inactive frames to all zero
1015
```

```
                }

            }

            if (!pActiveFrameFlags[f])

            {
                matbSet(0.0, mpCorrMatrix[f], mCorrelationVectorlength)
                matbSet(1.0, mpNormMatrix[f], mCorrelationVectorlength)

                MinOptDelayIndex = 0
                MaxOptDelayIndex = mCorrelationVectorlength
            }
        }
        matFree(pBuffer1)
        matFree(pBuffer2)
        matFree(HannWin)

    }

    mpSearchRangeLow = pSearchRangeLow
    mpSearchRangeHigh = pSearchRangeHigh
    return rc
}

void CCorrelationMatrix::Print(FILE* pLogFile)
{

}

}

```