```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

    typedef double OTA_FLOAT;
    typedef MAT_DCplx OTA_CPLX;

namespace POLQAV2
{

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,  85,  35,  0,  0},
                        {-1,    -1,  -1,  16,  12,  0,  0}},
                    {16000,  64, 64,
                        {256,  512, 256, 128,  64,  0},
                        {-1,    -1,  -1,  64,  34,  0},
                        {-1,    -1,  -1,  12,  10,  0}},
                    {48000, 256, 256,
                        {512, 1024, 512, 512, 128,  0},
                        {-1,    -1,  -1, 116,  62,  0},
                        {-1,    -1,  -1,  18,  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;
};
}

namespace POLQAV2
{

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;

            D = mP.mSpeechTAPara.ViterbiDistanceWeightFactorDist;
            D = D * mSamplerate / 1000;
            F =((float) log(1+0.5) / (D*D));
            mP.mSpeechTAPara.ViterbiDistanceWeightFactor = F;

            return true;
        }

        CParameters   mP;
};

class SECTION
{
    public:
        int Start;
        int End;
        int Len() {return End-Start;};
        void CopyFrom(const SECTION &src)
        {
            this->Start = src.Start;
            this->End   = src.End;
        }
};

typedef struct OTA_RESULT
{
    void CopyFrom(const OTA_RESULT* src)
    {
        mNumFrames           = src->mNumFrames;
        mStepsize            = src->mStepsize;
        mResolutionInSamples = src->mResolutionInSamples;
        if (src->mpDelay != NULL && mNumFrames > 0)
        {
            matFree(mpDelay);
            mpDelay = (long*)matMalloc(mNumFrames * sizeof(long));
            for (int i = 0; i < mNumFrames; i++)
                mpDelay[i] = src->mpDelay[i];
        }
```

```
else
{
    matFree(mpDelay);
    mpDelay = NULL;
}

if (src->mpReliability != NULL && mNumFrames > 0)
{
    matFree(mpReliability);
    mpReliability = (OTA_FLOAT*)matMalloc(mNumFrames * sizeof(OTA_FLOAT));
    for (int i = 0; i < mNumFrames; i++)
        mpReliability[i] = src->mpReliability[i];
}
else
{
    matFree(mpReliability);
    mpReliability = NULL;
}
mAvgReliability   = src->mAvgReliability;
mRelSamplerateDev = src->mRelSamplerateDev;

mNumUtterances = src->mNumUtterances;
if (src->mpStartSampleUtterance != NULL && mNumUtterances > 0)
{
    matFree(mpStartSampleUtterance);
    mpStartSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int));
    for (int i = 0; i < mNumUtterances; i++)
        mpStartSampleUtterance[i] = src->mpStartSampleUtterance[i];
}
else
{
    matFree(mpStartSampleUtterance);
    mpStartSampleUtterance = NULL;
}
if (src->mpStopSampleUtterance != NULL && mNumUtterances > 0)
{
    matFree(mpStopSampleUtterance);
    mpStopSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int));
    for (int i = 0; i < mNumUtterances; i++)
        mpStopSampleUtterance[i] = src->mpStopSampleUtterance[i];
}
else
{
    matFree(mpStopSampleUtterance);
    mpStopSampleUtterance = NULL;
}
if (src->mpDelayUtterance != NULL && mNumUtterances > 0)
{
    matFree(mpDelayUtterance);
    mpDelayUtterance = (int*)matMalloc(mNumUtterances * sizeof(int));
    for (int i = 0; i < mNumUtterances; i++)
        mpDelayUtterance[i] = src->mpDelayUtterance[i];
}
else
{
    matFree(mpDelayUtterance);
    mpDelayUtterance = NULL;
}

mNumSections = src->mNumSections;
if (src->mpRefSections != NULL && mNumSections > 0)
{
    delete[] mpRefSections;
    mpRefSections = new SECTION[mNumSections];
    for (int i = 0; i < mNumSections; i++)
        mpRefSections[i].CopyFrom(src->mpRefSections[i]);
}
else
{
    delete[] mpRefSections;
    mpRefSections = NULL;
}
if (src->mpDegSections != NULL && mNumSections > 0)
{
    delete[] mpDegSections;
    mpDegSections = new SECTION[mNumSections];
```

```
            for (int i = 0; i < mNumSections; i++)
                mpDegSections[i].CopyFrom(src->mpDegSections[i]);
        }
        else
        {
            delete[] mpDegSections;
            mpDegSections = NULL;
        }

        mSNRRefdB = src->mSNRRefdB;
        mSNRDegdB = src->mSNRDegdB;
        mNoiseLevelRef = src->mNoiseLevelRef;
        mNoiseLevelDeg = src->mNoiseLevelDeg;
        mSignalLevelRef = src->mSignalLevelRef;
        mSignalLevelDeg = src->mSignalLevelDeg;
        mNoiseThresholdRef = src->mNoiseThresholdRef;
        mNoiseThresholdDeg = src->mNoiseThresholdDeg;

        if (src->mpActiveFrameFlags != NULL && mNumFrames > 0)
        {
            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));
            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;
        }

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

    OTA_RESULT()
    {
        mNumFrames = 0;
        mpDelay = NULL;

        mpReliability = NULL;

        mNumUtterances = 0;
        mpStartSampleUtterance = NULL;
        mpStopSampleUtterance  = NULL;
```

```
    mpDelayUtterance       = NULL;

    mNumSections = 0;
    mpRefSections = NULL;
    mpDegSections = NULL;

    mpActiveFrameFlags = NULL;
    mpIgnoreFlags = NULL;

    mAslFrames = 0;
    mAslFramelength = 0;
    mpAslActiveFrameFlags = NULL;

    FirstRefSample = FirstDegSample = 0;
}

~OTA_RESULT()
{
    matFree(mpDelay);
    mpDelay = NULL;

    matFree(mpReliability);
    mpReliability = NULL;

    matFree(mpStartSampleUtterance);
    mpStartSampleUtterance = NULL;

    matFree(mpStopSampleUtterance);
    mpStopSampleUtterance  = NULL;

    matFree(mpDelayUtterance);
    mpDelayUtterance       = NULL;

    delete[] mpRefSections;
    mpRefSections = NULL;
    delete[] mpDegSections;
    mpDegSections = NULL;

    matFree(mpActiveFrameFlags);
    mpActiveFrameFlags = NULL;

    matFree(mpIgnoreFlags);
    mpIgnoreFlags = NULL;

    matFree(mpAslActiveFrameFlags);
    mpAslActiveFrameFlags = NULL;
}

long mNumFrames;
int mStepsize;
int mResolutionInSamples;
int mPitchFrameSize;
long *mpDelay;
OTA_FLOAT *mpReliability;
OTA_FLOAT mAvgReliability;
OTA_FLOAT mRelSamplerateDev;

int  mNumUtterances;
int* mpStartSampleUtterance;
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 mAslFrames;
    int mAslFramelength;
    int *mpAslActiveFrameFlags;

    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);

}

namespace POLQAV2
{

FILE* pLogFile=0;

CTempAlignment::CTempAlignment()
{
    mpFeatureList = 0;
    mpFeatureList2 = 0;
    mpDelaySearch = 0;
    mpActiveFrameDetection = 0;
    mppSignals = 0;
    mpDelayInSamplesPerFrame = 0;
    mpReliabilityPerFrame = 0;
    mpResults = 0;
    mpReparsePoints = 0;
    mStartOffset = 0;

    mpSmartBufferPool = new SmartBufferPool(7);
}

CTempAlignment::~CTempAlignment()
```

```
{
    delete mpSmartBufferPool;
}

bool CTempAlignment::Init(CProcessData* pProcessData, CDelaySearch* pDelaySearch,
CActiveFrameDetection* pActiveFrameDetection)
{
    bool rc = true;

    Free();

    mpDelaySearch = pDelaySearch;
    mpActiveFrameDetection = pActiveFrameDetection;
    pProcessData->mWindowSize = pProcessData->mpWindowSize[0];
    pProcessData->mOverlap = pProcessData->mpOverlap[0];
    pProcessData->Init(0, 1.0);
    mppSignals = new CTASignal* [2];
    for (int i=0; i<2; i++)
        mppSignals[i] = 0;
    pProcessData->mNumSignals = 0;
    mProcessData = *pProcessData;
    mpFeatureList = 0;
    mpFeatureList2 = 0;
    mpDelayInSamplesPerFrame = 0;
    mpReliabilityPerFrame = 0;

    mpResults = new OTA_RESULT;
    if (mpResults)
    {
        mpResults->mNumUtterances = 0;
        mpResults->mpDelayUtterance = 0;
        mpResults->mpStartSampleUtterance = 0;
        mpResults->mpStopSampleUtterance = 0;
        mpResults->mpRefSections = 0;
        mpResults->mpDegSections = 0;
        mpResults->mpActiveFrameFlags = 0;
        mpResults->mAvgReliability = 0.0f;
    }
    else rc = false;

    pLogFile=pProcessData->mpLogFile;

    return rc;
}

void CTempAlignment::Free()
{
    if (mpFeatureList) delete mpFeatureList;
    if (mpFeatureList2) delete mpFeatureList2;
    mpFeatureList = NULL;

    for (int i=0; mppSignals && i<2; i++)
        if (mppSignals[i])
        { delete mppSignals[i]; mppSignals[i]=0; }
    if (mppSignals)
        { delete[] mppSignals; mppSignals=0; }

    if (mpDelayInSamplesPerFrame)
        matFree(mpDelayInSamplesPerFrame);
    mpDelayInSamplesPerFrame = NULL;

    if (mpResults)
    {
        delete mpResults;
        mpResults = 0;
    }
    if (mpReparsePoints) delete[] mpReparsePoints;
    mpReparsePoints = 0;
}

#pragma region GENERAL DELAY SEARCH ROUTINES

//Find the pattern pA in the buffer pB. Returns the position of the max and sets
pReliability
//to the correlation at the maximum.
//The calculated delay is the delay where section A is found in section B, relative to
```

```
the start of section B.
//If sections exceed the buffer limits, a new and sufficiently long buffer will be
allocated and used.
//This buffer will be zero padded as needed. Passing negative section starts or ends
that lie beyond the
//last valid data are therefore allowed. Care must however be taken if this happens for
section A since
//the zero padding may skew the found delay.
//If MaxDelay is set to 0, the maximum possible search range will be used.
//NOTE 1: The selected feature must exist and must have been initialised properly
before calling this.

int CTempAlignment::FindSectionAInSectionB(SECTION* pA, SECTION*pB, CFeatureVector*
pVecA, CFeatureVector* pVecB, OTA_FLOAT* pReliability, int HistoLen, int HistoShift,
int MaxDelay, bool PlotMe)
{
    return FindSectionAInSectionB(pA, pB, pVecA->mpVector, pVecA->mSize,
pVecB->mpVector, pVecB->mSize, pReliability, HistoLen, HistoShift, MaxDelay,
PlotMe);
}

int CTempAlignment::FindSectionAInSectionB(SECTION* pA, SECTION*pB, OTA_FLOAT* pSigA,
int LenA, OTA_FLOAT* pSigB, int LenB, OTA_FLOAT* pReliability, int HistoLen, int
HistoShift, int MaxDelay, bool PlotMe)
{

    int DelayOfA;
    OTA_FLOAT* pCCF=0;
    OTA_FLOAT R1=-1;
    OTA_FLOAT R2=-1;

    SECTION SecA = *pA;
    SECTION SecB = *pB;

    assert(LenA>2);
    assert(LenB>2);
    assert(SecA.Start<=LenA);
    assert(SecB.Start<=LenB);
    assert(SecA.Start<SecA.End-2);
    assert(SecB.Start<SecB.End-2);

    int OffsetA=0;
    OTA_FLOAT* pUsedSigA=pSigA;
    bool MustDeletepUsedSigA=false;
    if (SecA.Start<0 || SecA.End+HistoLen*HistoShift>LenA)
    {
        int NewLen = SecA.End-SecA.Start+(HistoLen*HistoShift);
        pUsedSigA = matxMalloc(NewLen);
        MustDeletepUsedSigA = true;
        int SrcRangeStart = ((0) > (SecA.Start)) ? (0) : (SecA.Start));
        int SrcRangeEnd = ((LenA) < (SecA.End+(HistoLen*HistoShift))) ? (LenA) :
(SecA.End+(HistoLen*HistoShift)));
        int DestPos=0;
        if (SecA.Start<0)
        {
            matbZero(pUsedSigA, -SecA.Start);
            DestPos += -SecA.Start;
            OffsetA = -SecA.Start;
        }
        SecA.Start = 0;
        matbCopy(pSigA+SrcRangeStart, pUsedSigA+DestPos, SrcRangeEnd-SrcRangeStart);
        DestPos += SrcRangeEnd-SrcRangeStart;
        if (DestPos<NewLen) matbZero(pUsedSigA+DestPos, NewLen-DestPos);
        pSigA = pUsedSigA;
        LenA = NewLen;
        SecA.End = LenA-(HistoLen*HistoShift);
    }

    int OffsetB=0;
    OTA_FLOAT* pUsedSigB=pSigB;
    bool MustDeletepUsedSigB=false;
    if (SecB.Start<0 || SecB.End+(HistoLen*HistoShift)>LenB)
    {
        int NewLen = SecB.End-SecB.Start+(HistoLen*HistoShift);
        pUsedSigB = matxMalloc(NewLen);
        MustDeletepUsedSigB = true;
```

```
        int SrcRangeStart = (((0) > (SecB.Start)) ? (0) : (SecB.Start));
        int SrcRangeEnd = (((LenB) < (SecB.End+(HistoLen*HistoShift))) ? (LenB) :
(SecB.End+(HistoLen*HistoShift)));
        int DestPos=0;
        if (SecB.Start<0)
        {
            matbZero(pUsedSigB, -SecB.Start);
            DestPos += -SecB.Start;
            OffsetB = SecB.Start;
        }
        SecB.Start = 0;

        matbCopy(pSigB+SrcRangeStart, pUsedSigB+DestPos, SrcRangeEnd-SrcRangeStart);
        DestPos += SrcRangeEnd-SrcRangeStart;
        if (DestPos<NewLen) matbZero(pUsedSigB+DestPos, NewLen-DestPos);
        pSigB = pUsedSigB;
        LenB = NewLen;
        SecB.End = LenB-(HistoLen*HistoShift);
    }

    int SpaceToEndA = (((0) > (LenA-SecA.Start)) ? (0) : (LenA-SecA.Start));
    int SpaceToEndB = (((0) > (LenB-SecB.Start)) ? (0) : (LenB-SecB.Start));
    int NumFFramesA = (((SpaceToEndB) < (SecA.End-SecA.Start)) ? (SpaceToEndB) :
(SecA.End-SecA.Start));
    int NumFFramesB = (((SpaceToEndB) < (SecB.End-SecB.Start)) ? (SpaceToEndB) :
(SecB.End-SecB.Start));

    if (NumFFramesA>=NumFFramesB)
    {
        NumFFramesB = (((SpaceToEndB) < (SecB.End-SecB.Start)) ? (SpaceToEndB) :
(SecB.End-SecB.Start));
        NumFFramesA = (((SpaceToEndA) < (SecA.End-SecA.Start)) ? (SpaceToEndA) :
(SecA.End-SecA.Start));
        if (MaxDelay==0)
        {
            MaxDelay = SpaceToEndA-(HistoLen*HistoShift);
            MaxDelay = (((MaxDelay) < (NumFFramesA-NumFFramesB)) ? (MaxDelay) :
(NumFFramesA-NumFFramesB));
        }

        int DelayOfB;
        pCCF = new OTA_FLOAT[NumFFramesA];
        DelayOfB = FindDelay((MAT_HANDLE)mProcessData.mpMathlibHandle,
pSigB+SecB.Start, NumFFramesB, pSigA+SecA.Start, NumFFramesA, HistoLen,
HistoShift, MaxDelay, &R1, PlotMe);
        DelayOfB = DelayOfB - OffsetA - OffsetB;
        DelayOfA = -DelayOfB;
        if(pReliability) *pReliability = R1;
    }
    else
    {

        if (MaxDelay==0)
        {
            MaxDelay = SpaceToEndB-(HistoLen*HistoShift);
            MaxDelay = (((MaxDelay) < (NumFFramesB-NumFFramesA)) ? (MaxDelay) :
(NumFFramesB-NumFFramesA));
        }

        pCCF = new OTA_FLOAT[NumFFramesB];
        DelayOfA = FindDelay((MAT_HANDLE)mProcessData.mpMathlibHandle,
pSigA+SecA.Start, NumFFramesA, pSigB+SecB.Start, NumFFramesB, HistoLen,
HistoShift, MaxDelay, &R1, PlotMe);
        DelayOfA = DelayOfA + OffsetA + OffsetB;
        if(pReliability) *pReliability = R1;
    }

    if (MustDeletepUsedSigA) matFree(pUsedSigA);
    if (MustDeletepUsedSigB) matFree(pUsedSigB);

    return DelayOfA;

}
#pragma endregion

#pragma region SAMPLERATE MEASUREMENT AND CONVERSION
```

```
OTA_FLOAT CTempAlignment::GetSampleRateRatio_linear(int* pActiveFrameFlags, long*
DelayVector, int DelayVecLen, int Stepsize, int LagToDelay, bool EnablePlotting, int*
pUsedForSRDet, OTA_FLOAT* partialSRperFrame, int* numTotalPartialSR)
{

    OTA_FLOAT const MIN_RATIO_OF_FRAMES_FITTING_LINEAR_MODEL = 0.9;
    OTA_FLOAT const MAX_DIST_LINE_SAMPLES = 0.032 * mProcessData.mSamplerate;

    int const MIN_FRAMES_INTER_SENTENCE_PAUSE = 15;
    int const MIN_FRAMES_IN_SENTENCE = 30;
    int const NUM_FRAMES_TOLERANCE = 4;

    OTA_FLOAT SamplerateRatio = -1.0;
    bool isConstantSampleRate = true;

    *numTotalPartialSR = -1;
    if (partialSRperFrame)
        matbSet(0, partialSRperFrame, DelayVecLen);

    int numPartialSR = 0;

    OTA_FLOAT xy_sum = 0.0, x_sum = 0.0, y_sum = 0.0, x2_sum = 0.0;
    int numActFrames = 0;
    for (int i=0; i<DelayVecLen; i++)
    {
        if (pActiveFrameFlags[i])
        {
            numActFrames++;
            xy_sum += (OTA_FLOAT)i * DelayVector[i];
            x_sum += (OTA_FLOAT)i;
            y_sum += DelayVector[i];
            x2_sum += pow((OTA_FLOAT)i,2.0);
        }
    }
    OTA_FLOAT a = ((OTA_FLOAT)numActFrames*xy_sum - x_sum*y_sum) /
((OTA_FLOAT)numActFrames*x2_sum - pow(x_sum,2.0));
    OTA_FLOAT b = (y_sum - a*x_sum) / (OTA_FLOAT)numActFrames;

    OTA_FLOAT* expectedDelay = new OTA_FLOAT[DelayVecLen];
    OTA_FLOAT* delayEstimateError = new OTA_FLOAT[DelayVecLen];
    OTA_FLOAT* expectedDelay_partial = NULL;

        int numFramesFittingLine=0;
        for (int i=0; i<DelayVecLen; i++)
        {
            expectedDelay[i] = a * (OTA_FLOAT)i + b;
            if (pActiveFrameFlags[i])
            {
                delayEstimateError[i] = abs(DelayVector[i] - expectedDelay[i]);
                if (delayEstimateError[i] <= MAX_DIST_LINE_SAMPLES)
                    numFramesFittingLine++;
            }
            else
                delayEstimateError[i] = 0;
        }

        if ((OTA_FLOAT)numFramesFittingLine/(OTA_FLOAT)numActFrames >=
MIN_RATIO_OF_FRAMES_FITTING_LINEAR_MODEL)
        {
            SamplerateRatio = 1 - a/(OTA_FLOAT)mProcessData.mStepSize;

            *numTotalPartialSR = 1;
            for (int j=0; j<DelayVecLen; j++)
                partialSRperFrame[j] = SamplerateRatio;
        }
        else
        {

            expectedDelay_partial = new OTA_FLOAT[DelayVecLen];

            std::vector<OTA_FLOAT> a_partial, b_partial;
            OTA_FLOAT partialSamplerateRatio;
            bool started = false;
            int consecutivePauseFrames = 0, firstActiveFrame = 0, lastActiveFrame=0;
            int numActFramesInSentence = 0, numFramesFittingLineInSentence = 0;
```

```
            xy_sum = 0.0, x_sum = 0.0, y_sum = 0.0, x2_sum = 0.0;
            for (int i=0; i<DelayVecLen; i++)
            {
                expectedDelay_partial[i] = 0;

                if (pActiveFrameFlags[i])
                {
                    if (started == false)
                        firstActiveFrame = i;
                    started = true;
                    xy_sum += (OTA_FLOAT)i * DelayVector[i];
                    x_sum += (OTA_FLOAT)i;
                    y_sum += DelayVector[i];
                    x2_sum += pow((OTA_FLOAT)i,2.0);
                    lastActiveFrame = i;
                    numActFramesInSentence++;
                }
                else
                {
                    if (started==true)
                        consecutivePauseFrames++;
                }

                if (((consecutivePauseFrames > MIN_FRAMES_INTER_SENTENCE_PAUSE) &&
((lastActiveFrame - firstActiveFrame) > MIN_FRAMES_IN_SENTENCE)) )
                {
                    a_partial.push_back(((OTA_FLOAT)numActFramesInSentence*xy_sum -
x_sum*y_sum) / ((OTA_FLOAT)numActFramesInSentence*x2_sum -
pow(x_sum,2.0)));
                    b_partial.push_back((y_sum - a_partial[numPartialSR]*x_sum) /
(OTA_FLOAT)numActFramesInSentence);

                    numFramesFittingLineInSentence = 0;
                    for (int j=firstActiveFrame; j<=lastActiveFrame; j++)
                    {
                        expectedDelay_partial[j] = a_partial[numPartialSR] *
(OTA_FLOAT)j + b_partial[numPartialSR];
                        if (pActiveFrameFlags[j])
                        {
                            if ((abs(DelayVector[j] - expectedDelay_partial[j])) <=
MAX_DIST_LINE_SAMPLES )
                                numFramesFittingLineInSentence++;
                        }
                    }

                    if((OTA_FLOAT)numFramesFittingLineInSentence/(OTA_FLOAT)numActFrame
sInSentence >= MIN_RATIO_OF_FRAMES_FITTING_LINEAR_MODEL)
                        partialSamplerateRatio = 1 -
a_partial[numPartialSR]/(OTA_FLOAT)mProcessData.mStepSize;
                    else
                    {
                        isConstantSampleRate = false;
                        partialSamplerateRatio = -1.0;
                    }

                    for (int j=(((firstActiveFrame - NUM_FRAMES_TOLERANCE) > (0)) ?
(firstActiveFrame - NUM_FRAMES_TOLERANCE) : (0));
j<=(((lastActiveFrame + NUM_FRAMES_TOLERANCE) < (DelayVecLen)) ?
(lastActiveFrame + NUM_FRAMES_TOLERANCE) : (DelayVecLen)); j++)
                        partialSRperFrame[j] = partialSamplerateRatio;

                    numPartialSR++;
                    xy_sum = 0.0, x_sum = 0.0, y_sum = 0.0, x2_sum = 0.0;
                    numActFramesInSentence = 0;
                    consecutivePauseFrames = 0;
                    started = false;
                }

            }

            if (numActFramesInSentence>0)
                for (int j=(((firstActiveFrame - NUM_FRAMES_TOLERANCE) > (0)) ?
(firstActiveFrame - NUM_FRAMES_TOLERANCE) : (0)); j<=(((lastActiveFrame
+ NUM_FRAMES_TOLERANCE) < (DelayVecLen)) ? (lastActiveFrame +
NUM_FRAMES_TOLERANCE) : (DelayVecLen)); j++)
                    partialSRperFrame[j] = -1.0;
```

```
            if(numPartialSR>0)
            {
                OTA_FLOAT minEstimatedSR = 1 -
a_partial[0]/(OTA_FLOAT)mProcessData.mStepSize;
                OTA_FLOAT maxEstimatedSR = 1 -
a_partial[0]/(OTA_FLOAT)mProcessData.mStepSize;
                for (int j=1; j<numPartialSR; j++)
                {
                    minEstimatedSR = (((minEstimatedSR) < (1 -
a_partial[j]/(OTA_FLOAT)mProcessData.mStepSize)) ? (minEstimatedSR)
: (1 - a_partial[j]/(OTA_FLOAT)mProcessData.mStepSize));
                    maxEstimatedSR = (((maxEstimatedSR) > (1 -
a_partial[j]/(OTA_FLOAT)mProcessData.mStepSize)) ? (maxEstimatedSR)
: (1 - a_partial[j]/(OTA_FLOAT)mProcessData.mStepSize));
                }
                if ((((minEstimatedSR >= 1.0)&&(maxEstimatedSR >=
1.0))||((minEstimatedSR <= 1.0)&&(maxEstimatedSR <=
1.0)))&&(isConstantSampleRate))
                {
                    if (maxEstimatedSR > 1.0)
                        SamplerateRatio = minEstimatedSR;
                    else if(maxEstimatedSR < 1.0)
                        SamplerateRatio = maxEstimatedSR;
                    else
                        SamplerateRatio = 1.0;
                }
                else
                    SamplerateRatio = -1.0;
            }

        }

    *numTotalPartialSR = numPartialSR;

    delete[] expectedDelay; expectedDelay = NULL;
    delete[] delayEstimateError; delayEstimateError = NULL;
    if (expectedDelay_partial)
    {
        delete[] expectedDelay_partial; expectedDelay_partial = NULL;
    }

    if (abs(1-SamplerateRatio) < 0.001)
        SamplerateRatio = 1.0;

    return SamplerateRatio;
}

#pragma endregion

#pragma region INITIAL DELAY SEARCH

//Get the coarse delay between the  two input signals.
//The delay is positive if the degraded signal comes before the Reference signal
//The first StartFrame frames are skipped to avoid calculating the delay on a silent
intervall
//The begining of the ref signal is searched in the deg signal.

bool CTempAlignment::GetCoarseAvgDelayAtStart(int FeatureIndex, int Channel, int
StartFrameLong, int StartFrameShort, OTA_FLOAT CoarseAlignSegmentLengthInMs, int
MaxDelay, int *AvgDelayInFrames, OTA_FLOAT* CorrAtMax, int LongSig, int ShortSig)
{

    bool rc = true;

    CFeatureVector* FVectors[2];
    FVectors[LongSig] = mpFeatureList->GetFVector(FeatureIndex, LongSig, Channel,
FeatureIndex==0?1:0);
    FVectors[ShortSig] = mpFeatureList->GetFVector(FeatureIndex, ShortSig, Channel,
FeatureIndex==0?1:0);

    int SegmentLength = (int)(CoarseAlignSegmentLengthInMs * mProcessData.mSamplerate /
1000 / mProcessData.mStepSize);

    int LagLeft = 0;
    int Lag = MaxDelay;
```

```
    int DegShift = 0;

    {
        int MaxIndex=0;
        SECTION SecA;
        SECTION SecB;
        SecA.Start = StartFrameShort+DegShift;
        SecA.End   = SecA.Start + SegmentLength;
        SecB.Start = StartFrameLong-LagLeft;
        SecB.End   = SecB.Start + SegmentLength + Lag;

        if (SecA.Start<((int)FVectors[ShortSig]->mSize)-10 &&
SecB.Start<((int)FVectors[LongSig]->mSize)-10)
        {
            MaxIndex = FindSectionAInSectionB(&SecA, &SecB, FVectors[ShortSig],
FVectors[LongSig], CorrAtMax, 50, 1);
            MaxIndex = MaxIndex - SecA.Start + SecB.Start;
            *AvgDelayInFrames = -MaxIndex;
        }
        else
        {
            ;
            *AvgDelayInFrames = -10000;
            *CorrAtMax = -1;
        }

    }

    return rc;
}

//Get an initial estimate of the delay at each reparse point.
//This estimate is used as a starting point and refined with every alignment iteration.
//The degraded signal is searched in the ref signal.
bool CTempAlignment::GetInitialDelayInSamplesForOneDownsamplingStep(int Point, SEGMENT*
pSegment, int DownsamplingStep, int* FoundDelayInSamples, OTA_FLOAT* RAtDelay, int
MaxInitialDelay)
{
    bool rc = true;

    mProcessData.Init(2, (float)DownsamplingStep);

    if (mProcessData.mpLogFile)
        ;

    MaxInitialDelay /= mProcessData.mStepSize;

    rc = mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH,
pSegment);
    if (rc)
    {
        int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize;

        int NextIndex=0;
        if (rc)
        {
            const int NumSegmentFactors = 2;
            const int NumWindowShifts = 2;
            int f, i, j, FoundDelay;
            OTA_FLOAT CorrAtMax=0;
            int FeatureIndex=0;
            OTA_FLOAT* Correlations = new
OTA_FLOAT[NumSegmentFactors*NumWindowShifts*mpFeatureList->mNumFeatures];
            int* Delays = new
int[NumSegmentFactors*NumWindowShifts*mpFeatureList->mNumFeatures];
            int* Windows = new int[NumWindowShifts];
            int* Segments = new int[NumSegmentFactors];

            int
WindowShiftUnit=(int)((0.7*mProcessData.mSamplerate)/mProcessData.mStepSize
);

            for (j=0; j<NumWindowShifts; j++)
            {

                Windows[j] =
```

```
(int)((0.1*mProcessData.mSamplerate)/mProcessData.mStepSize) +
j*WindowShiftUnit;

                for (i=0; i<NumSegmentFactors; i++)
                {
                    OTA_FLOAT SegmentFactor=0.5*(i+1);
                    Segments[i] =
(int)(mProcessData.mP.mTAPara.CoarseAlignSegmentLengthInMs*SegmentF
actor);

                    for (f=0; f<mpFeatureList->mNumFeatures; f++)
                    {

                        if (GetCoarseAvgDelayAtStart(f, 0, Windows[j], Windows[j],
Segments[i], MaxInitialDelay, &FoundDelay, &CorrAtMax, 0, 1))

                        {
                            Correlations[NextIndex] = CorrAtMax;
                            Delays[NextIndex] = FoundDelay;
                        }
                        else
                        {
                            Correlations[NextIndex] = 0;
                            Delays[NextIndex] = 0;
                        }
                        NextIndex++;
                    }
                }
            }

            int BestFeature = 0;
            OTA_FLOAT BestR = Correlations[0];
            int Offset = (pSegment[0].Start - pSegment[1].Start) /
mProcessData.mStepSize;;
            for (f=0; f<NextIndex; f++)
            {

                if (BestR<Correlations[f]+0.03)
                {
                    if (fabs(Correlations[f]-BestR) > 0.03 || abs(Delays[f]+Offset)<
abs(Delays[BestFeature]+Offset))
                    {
                        BestFeature = f;
                        BestR = Correlations[f];
                    }
                }

            }
            //Convert the delay from frames to samples and relate it to the degraded
signal
            *FoundDelayInSamples = -Delays[BestFeature]*mProcessData.mStepSize;
            *RAtDelay = BestR;

        }
    }
    else
    {
        ;
        ;
        *FoundDelayInSamples = 0;
        *RAtDelay = -1;
    }

    return rc;
}

//Search through both input signals and identify the initial delay.
//The first OffsetInSamples samples are skipped (but may be used for negative delays)
long CTempAlignment::GetInitialDelayInSamples(int Point, SEGMENT* pSegments, int*
pStartSampleRef, int *pStartSampleDeg, int* pActiveFrameFlags, OTA_FLOAT* pReliability,
int *WorstResolutionInSamples, int SectionOffset, int MaxInitialDelay)
{

    bool rc=true;
    int i;
    int Delays[2];
```

```
    OTA_FLOAT Rs[2];
    *WorstResolutionInSamples = -1;
    for (i=0; i<2; i++)
    {
        Delays[i] = 0; Rs[i] = 0;
        GetInitialDelayInSamplesForOneDownsamplingStep(Point, pSegments, i+1,
&Delays[i], &Rs[i], MaxInitialDelay);
        if (mProcessData.mStepSize>*WorstResolutionInSamples)
            *WorstResolutionInSamples = mProcessData.mStepSize;
    }
    int BestIteration=0;
    int BestDelay = Delays[0];
    OTA_FLOAT BestR = Rs[0];
    for (i=1; i<2; i++)
    {

        if (BestR<Rs[i]+0.03)
        {
            if (fabs(Rs[i]-BestR) > 0.03 || abs(Delays[i]+SectionOffset)<
abs(BestDelay+SectionOffset))
            {
                BestDelay = Delays[i];
                BestR = Rs[i];
                BestIteration = i;
            }
        }

    }

    if (BestR<0.6)
    {
        BestDelay = 0;
        BestIteration = -1;
        if (mProcessData.mpLogFile)
            ;
    }
    else if (abs(BestDelay)>MSecondsToSamples(200) && BestR<0.8)
    {
        BestDelay = 0;
        BestIteration = -1;
        if (mProcessData.mpLogFile)
            ;
    }

    *pReliability = BestR;
    if (BestIteration<0) *pReliability-=0.2;

    if (mProcessData.mpLogFile)
        ;

    return BestDelay;
}

//For each reparse point identify the delay in samples
//Here we also set the ref start point and do also refine both startpoints.
int CTempAlignment::GetInitialDelaysInSamples(REPARSE_POINT* ParsePoints, int
NumParsePoints, int* pActiveFrameFlags, int OverallDelayEstimate, OTA_FLOAT
OverallDelayEstimateReliability, int *AccuracyInSamples)
{
    int Point;
    int LastDelayInSamples=0;
    SEGMENT Segments[2];

    for (Point=0; Point<NumParsePoints; Point++)
    {
        ;

        if (ParsePoints[Point].Reliability >
mProcessData.mP.mTAPara.mCorrForSkippingInitialDelaySearch)

        {
            ParsePoints[Point].DelayInSamples = -ParsePoints[Point].Deg.Start +
ParsePoints[Point].Ref.Start;
            ;
        }
        else
```

```
        {
            Segments[1].Start = ParsePoints[Point].Deg.Start;
            Segments[1].Start = (((0) > ((((mppSignals[1]->mSignalLength) <
(Segments[1].Start)) ? (mppSignals[1]->mSignalLength) :
(Segments[1].Start)))) ? (0) : ((((mppSignals[1]->mSignalLength) <
(Segments[1].Start)) ? (mppSignals[1]->mSignalLength) :
(Segments[1].Start))));
            Segments[1].End   = Segments[1].Start + 4*mProcessData.mSamplerate;

            int MaxInitialDelay = 0;

            Segments[0].Start = ParsePoints[Point].Ref.Start -
600/1000*mProcessData.mSamplerate;
            Segments[0].Start = (((0) >
(Segments[0].Start-0.1*mProcessData.mSamplerate)) ? (0) :
(Segments[0].Start-0.1*mProcessData.mSamplerate));

            if (Point && Segments[0].Start <
ParsePoints[Point-1].Deg.End+ParsePoints[Point-1].DelayInSamples)
                Segments[0].Start = (((0) >
(ParsePoints[Point-1].Deg.End+ParsePoints[Point-1].DelayInSamples)) ?
(0) :
(ParsePoints[Point-1].Deg.End+ParsePoints[Point-1].DelayInSamples));

            MaxInitialDelay = (OTA_FLOAT)600*mProcessData.mSamplerate/1000.0;

            Segments[0].End = Segments[0].Start +
(600+5000)/1000*mProcessData.mSamplerate;;

            Segments[0].End = (((Segments[0].End) < (mppSignals[0]->mSignalLength-1)) ?
(Segments[0].End) : (mppSignals[0]->mSignalLength-1));
            Segments[1].End = (((Segments[1].End) < (mppSignals[1]->mSignalLength-1)) ?
(Segments[1].End) : (mppSignals[1]->mSignalLength-1));

            OTA_FLOAT Reliability;
            int DelayOffset = Segments[0].Start - Segments[1].Start;
            long DelayInSamples = GetInitialDelayInSamples(Point, Segments,
&ParsePoints[Point].Ref.Start, &ParsePoints[Point].Deg.Start,
pActiveFrameFlags, &Reliability, AccuracyInSamples, DelayOffset,
MaxInitialDelay);

            DelayInSamples += DelayOffset;

            if (Reliability>ParsePoints[Point].Reliability)
            {
                ParsePoints[Point].DelayInSamples = DelayInSamples;
                ParsePoints[Point].Reliability = Reliability;
                ;
            }
            else ;

        }

    }

    return NumParsePoints;
}
#pragma endregion

#pragma region FIND USED SECTION

//If one of the two signals is much longer, it may also contain more active sections
than the other.
//Here we are looking for a section which gives a good overall match and we will
discard all others
//afterwards.
//Returns the detected start sample of the active part of the ref signal (minus some
guard intervall).
int CTempAlignment::FindUsedSectionOfRefSignal()
{
    int SectionStart = 0;
    const int HistoLen = 1;
    const int HistoShift = 0;
    SECTION SecA, SecB;
    mProcessData.Init(2, 1.0);
```

```
    if (mProcessData.mpLogFile)
        ;

    if (mppSignals[0]->mSignalLength>2*mppSignals[1]->mSignalLength)
    {
        ;
        mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH);
        int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize;
        int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize;
        OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector;
        OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector;

        int DegDelay;
        OTA_FLOAT CorrAll;
        SecA.Start = 0; SecA.End = NumFFramesDeg;
        SecB.Start = 0; SecB.End = NumFFramesRef;
        if (SecA.End-SecA.Start>=MSecondsToFrames(1000))
        {
            DegDelay = FindSectionAInSectionB(&SecA, &SecB, pDeg, NumFFramesDeg, pRef,
NumFFramesRef, &CorrAll, HistoLen, HistoShift);
            DegDelay = DegDelay - SecA.Start+SecB.Start;
            if (CorrAll>0.8)
                SectionStart = (((0) > (DegDelay-MSecondsToFrames(300))) ? (0) :
(DegDelay-MSecondsToFrames(300)));
            else ;
        }
    }
    else if (mppSignals[1]->mSignalLength>2*mppSignals[0]->mSignalLength)
    {
        ;
        mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH);
        int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize;
        int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize;
        OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector;
        OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector;

        int DegDelay;
        OTA_FLOAT CorrAll;
        SecA.Start = 0; SecA.End = NumFFramesRef;
        SecB.Start = 0; SecB.End = NumFFramesDeg;
        if (SecA.End-SecA.Start>=MSecondsToFrames(1000))
        {
            DegDelay = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef, pDeg,
NumFFramesDeg, &CorrAll, HistoLen, HistoShift);
            DegDelay = DegDelay - SecA.Start+SecB.Start;
            if (CorrAll>0.8)
                SectionStart = -(((0) > (DegDelay-MSecondsToFrames(300))) ? (0) :
(DegDelay-MSecondsToFrames(300)));
            else ;
        }
    }
    else ;

    SectionStart = FramesToSamples(SectionStart);
    ;
    return SectionStart;
}

#pragma endregion

#pragma region OVERALL DELAY ESTIMATION

//Estimate the overall delay. This may be completely off, but in most cases it is a
good starting point.
//Especially for low SNR values this may help identifying the correct reparse sections.
//Also, if the correlation is high enough, we may skip/speed up some later alignment
steps.
//
//the end of the last active ref section. The deg signal is used entirely.
//For the first half, the ref signal starts with the first active section and ranges
till the middle of the signal,
//the deg section starts at 0 and ends at 75% of the signal length.
//For the second half, the ref signal starts in the middle and ends after the last
active section. The deg
//section begins at 25% signal length and ends at the end.
//
```

```
bool CTempAlignment::EstimateOverallDelaySimpleLimits(int*
OverallDelayEstimateInSamples,      OTA_FLOAT* OverallDelayEstimateReliability,
                                                    int*
OverallDelayEstimateInSamples1st,
OTA_FLOAT*
OverallDelayEstimateReliability1s
t,
                                                    int*
OverallDelayEstimateInSamples2nd,
OTA_FLOAT*
OverallDelayEstimateReliability2n
d,
                                                    int* Resolution, bool
IncreasedResolution)
{
    return EstimateOverallDelaySimpleLimits(OverallDelayEstimateInSamples,
OverallDelayEstimateReliability, OverallDelayEstimateInSamples1st,
OverallDelayEstimateReliability1st,
                                        OverallDelayEstimateInSamples2nd,
OverallDelayEstimateReliability2nd,
Resolution, IncreasedResolution, 0, 0);
}

bool CTempAlignment::EstimateOverallDelaySimpleLimits(int*
OverallDelayEstimateInSamples,      OTA_FLOAT* OverallDelayEstimateReliability,
                                                    int*
OverallDelayEstimateInSamples1st,
OTA_FLOAT*
OverallDelayEstimateReliability1s
t,
                                                    int*
OverallDelayEstimateInSamples2nd,
OTA_FLOAT*
OverallDelayEstimateReliability2n
d,
                                                    int* Resolution, bool
IncreasedResolution, int
InitialEstimate, int
InitialEstimateResolution)
{
    int HistoLen = 10;
    int HistoShift = 1;
    int SignalPartDelayIterations = 2;

    int SearchRange = InitialEstimateResolution;

    SECTION SecA, SecB;
    if (IncreasedResolution)
    {
        mProcessData.Init(2, 1.0/128.);
        HistoLen = 1;
        HistoShift = 0;
        SignalPartDelayIterations = 1;
    }
    else
    {
        mProcessData.Init(2, 1.0);
        HistoLen = 10;
        HistoShift = 1;
        SignalPartDelayIterations = 2;

        InitialEstimateResolution = 0;
        InitialEstimate = 0;
    }

    if (mProcessData.mpLogFile)
        ;

    mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH);
    int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize;
    int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize;
    OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector;
    OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector;
    *Resolution = mProcessData.mStepSize;

    int StartFrameRef = mpActiveFrameDetection->GetStartFrame(0, 0,
```

```
mProcessData.mStepSize, 0);
    int LastFrameRef  = mpActiveFrameDetection->GetLastActiveFrame(0, 0,
mProcessData.mStepSize, 0);
    int StartFrameDeg = 0;
    int LastFrameDeg  = NumFFramesDeg;

    if (InitialEstimateResolution>0)
    {
        StartFrameDeg = StartFrameRef + (-InitialEstimate - SearchRange) / *Resolution;
        StartFrameDeg = (((0) > (StartFrameDeg)) ? (0) : (StartFrameDeg));
        LastFrameDeg  = LastFrameRef  + (-InitialEstimate + SearchRange) / *Resolution;

        LastFrameDeg = (((LastFrameDeg) < (NumFFramesDeg-1)) ? (LastFrameDeg) :
(NumFFramesDeg-1));

        if (LastFrameRef-StartFrameRef>LastFrameDeg-StartFrameDeg)
            LastFrameRef = LastFrameDeg - SearchRange / *Resolution;
    }

    int ms50InFrames=MSecondsToFrames(25);

    int DegDelay;
    OTA_FLOAT CorrAll;
    SecA.Start = StartFrameRef; SecA.End = LastFrameRef;
    SecB.Start = StartFrameDeg; SecB.End = LastFrameDeg;
    if (SecA.End-SecA.Start<MSecondsToFrames(500))
    {
        ;
        *OverallDelayEstimateInSamples = 0;
        *OverallDelayEstimateReliability = 0;
        return false;
    }

    ;
    DegDelay = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef, pDeg,
NumFFramesDeg, &CorrAll, HistoLen, HistoShift);
    DegDelay = DegDelay - StartFrameRef + StartFrameDeg;

    int DelayTemp;
    OTA_FLOAT CorrTemp;

    int SecBStartForIteration[5];
    int EndOffset = LastFrameRef-LastFrameDeg;

    int Delay1st = 0;
    OTA_FLOAT Corr1st = 0;

    DelayTemp = 0;
    CorrTemp = 0;

    SecBStartForIteration[0] = (((0) > (StartFrameDeg - ms50InFrames)) ? (0) :
(StartFrameDeg - ms50InFrames));
    SecBStartForIteration[1] = (((0) > (StartFrameRef + DegDelay - ms50InFrames)) ? (0)
: (StartFrameRef + DegDelay - ms50InFrames));

    SecA.End = NumFFramesRef/2;
    if (InitialEstimateResolution>0)
    {
        SecB.End = (int)(SecA.End - EndOffset);
    }
    else
    {
        SecB.End = (int)(NumFFramesDeg*0.75);
    }

    for(int iteration = 0; iteration < SignalPartDelayIterations; iteration++)
    {
        SecB.Start = SecBStartForIteration[iteration];
        if (SecB.Start<SecB.End && SecA.Start<SecA.End)
        {
            if ((OTA_FLOAT)(SecA.End-SecA.Start) /
(OTA_FLOAT)(SecB.End-SecB.Start)>0.99)
                SecA.End = SecA.Start + (int)(0.99*(OTA_FLOAT)(SecB.End-SecB.Start));
            if (SecA.End-SecA.Start<MSecondsToFrames(500))
            {
                ;
```

```
            }
            else
            {
                ;
                DelayTemp = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef,
pDeg, NumFFramesDeg, &CorrTemp, HistoLen);
                DelayTemp += -SecA.Start + SecB.Start;
                ;
                if(Corr1st < CorrTemp)
                {
                    Corr1st = CorrTemp;
                    Delay1st = DelayTemp;
                }
            }
        }
    }

    int Delay2nd = 0;
    OTA_FLOAT Corr2nd = 0;

    DelayTemp = 0;
    CorrTemp = 0;

    if (InitialEstimateResolution>0)
    {
        int StartOffset = StartFrameRef - StartFrameDeg;
        StartFrameRef = (int)(0.5*NumFFramesRef);
        StartFrameDeg = StartFrameRef - StartOffset;
        SecB.End = LastFrameDeg;

    }
    else
    {
        StartFrameRef = (int)(0.5*NumFFramesRef);
        StartFrameDeg = (int)(0.25*NumFFramesDeg);
        SecB.End = NumFFramesDeg;
    }

    SecA.Start = StartFrameRef;
    SecA.End = LastFrameRef;

    SecBStartForIteration[0] = (((0) > (StartFrameDeg)) ? (0) : (StartFrameDeg));
    SecBStartForIteration[1] = (((0) > (StartFrameRef + DegDelay - ms50InFrames)) ? (0)
: (StartFrameRef + DegDelay - ms50InFrames));

    for(int iteration = 0; iteration < SignalPartDelayIterations; iteration++)
    {
        SecB.Start = SecBStartForIteration[iteration];
        if (SecB.Start<SecB.End && SecA.Start<SecA.End)
        {
            if ((OTA_FLOAT)(SecA.End-SecA.Start) /
(OTA_FLOAT)(SecB.End-SecB.Start)>0.99)
                SecA.End = SecA.Start + 0.99*(OTA_FLOAT)(SecB.End-SecB.Start);

            if (SecA.End-SecA.Start<MSecondsToFrames(500))
            {
                ;
            }
            else
            {
                ;
                DelayTemp = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef,
pDeg, NumFFramesDeg, &CorrTemp, HistoLen);
                DelayTemp += -SecA.Start + SecB.Start;
                ;
                if(Corr2nd < CorrTemp)
                {
                    Corr2nd = CorrTemp;
                    Delay2nd = DelayTemp;
                }
            }
        }
    }

    int Tolerance = ms50InFrames;
    int MaxLength = MSecondsToFrames(15000);
```

```
    if (LastFrameRef>MaxLength && CorrAll<0.75)
        Tolerance = MSecondsToFrames(10);

    if (abs(DegDelay-Delay1st)>Tolerance)
        *OverallDelayEstimateReliability = 0;
    else if (abs(DegDelay-Delay2nd)>Tolerance)
        *OverallDelayEstimateReliability = 0;
    else
        *OverallDelayEstimateReliability = CorrAll;

    if (CorrAll>0.94)
        *OverallDelayEstimateReliability = CorrAll;

    *OverallDelayEstimateInSamples = FramesToSamples( -DegDelay);

    *OverallDelayEstimateReliability1st = Corr1st;
    *OverallDelayEstimateInSamples1st = FramesToSamples( -Delay1st);

    *OverallDelayEstimateReliability2nd = Corr2nd;
    *OverallDelayEstimateInSamples2nd = FramesToSamples( -Delay2nd);

    bool DelayIsReliableandConst=false;
    const OTA_FLOAT MinCorr = 0.5;
    if (Delay1st==Delay2nd && Delay1st==DegDelay && CorrAll>MinCorr && Corr1st>MinCorr
&& Corr2nd>MinCorr)
        DelayIsReliableandConst = true,

    return DelayIsReliableandConst;
}

#pragma endregion

#pragma region  IDENTIFY AND ALLOCATE REPARSE POINTS

int CTempAlignment::GetNearestStart(int Signal, int StartSample, int OffsetInSamples,
bool IgnoreActivityFlags)
{

    assert(StartSample>=0);
    assert(StartSample+OffsetInSamples>=0);

    int NumMacroFramesRef = mpActiveFrameDetection->GetMaxFrames(Signal, 0);
    int RefStartFrame;

    if (!IgnoreActivityFlags)
    {
        int* pActiveFrameFlagsRef = new int[NumMacroFramesRef];

        NumMacroFramesRef = mpActiveFrameDetection->GetActiveFrameFlags(Signal, 0,
mProcessData.mStepSize, pActiveFrameFlagsRef, NumMacroFramesRef);

        RefStartFrame = (StartSample+OffsetInSamples) / mProcessData.mStepSize;
        RefStartFrame = (((RefStartFrame) < (NumMacroFramesRef)) ? (RefStartFrame) :
(NumMacroFramesRef));
        if (RefStartFrame==0) RefStartFrame = 1;
        if (!pActiveFrameFlagsRef[RefStartFrame] )
        {
            while (RefStartFrame<NumMacroFramesRef &&
!pActiveFrameFlagsRef[RefStartFrame])
                RefStartFrame++;

            while (RefStartFrame && !pActiveFrameFlagsRef[RefStartFrame])
                RefStartFrame--;

            int LastEnd = RefStartFrame;
            while (LastEnd && !pActiveFrameFlagsRef[LastEnd])
                LastEnd--;
            RefStartFrame = LastEnd + (RefStartFrame-LastEnd)/2;
        }
        else
        {
            while (RefStartFrame && pActiveFrameFlagsRef[RefStartFrame])
                RefStartFrame--;

            int LastEnd = RefStartFrame;
            while (LastEnd && !pActiveFrameFlagsRef[LastEnd])
```

```
                LastEnd--;
            RefStartFrame = LastEnd + (RefStartFrame-LastEnd)/2;

        }

        if (RefStartFrame==NumMacroFramesRef)
        {
            RefStartFrame = (((0) > (StartSample-OffsetInSamples)) ? (0) :
(StartSample-OffsetInSamples))/mProcessData.mStepSize;
            if (RefStartFrame>=NumMacroFramesRef)
                RefStartFrame = StartSample / mProcessData.mStepSize;

            if (RefStartFrame>=NumMacroFramesRef)
                RefStartFrame = 0;
        }

        delete[] pActiveFrameFlagsRef;
    }
    else
    {
        RefStartFrame = (((0) > (StartSample-OffsetInSamples)) ? (0) :
(StartSample-OffsetInSamples))/mProcessData.mStepSize;
        if (RefStartFrame>=NumMacroFramesRef)
            RefStartFrame = 0;
    }

    if (RefStartFrame>1) RefStartFrame-= 2;
    else RefStartFrame = 0;

    return mpActiveFrameDetection->GetStartSample(Signal, 0,
RefStartFrame*mProcessData.mStepSize);
}

//Do a coarse localisation of reparse points by searching for the next longer active
section after
//an initial inactive section.
int CTempAlignment::SearchActiveSegments(REPARSE_POINT* ParsePoints, int
MaxParsePoints, int* pActiveFrameFlags)
{
    int NumParsePointsDetected=0;
    int i=0;

    while (i<mNumMacroFrames && !pActiveFrameFlags[i])  i++;
    ParsePoints[NumParsePointsDetected].Deg.Start = FramesToSamples(i);

    while(i<mNumMacroFrames)
    {
        i = GetNextPauseStartFrameIndex(pActiveFrameFlags, i, mNumMacroFrames);

        ParsePoints[NumParsePointsDetected++].Deg.End = FramesToSamples(i);

        while (i<mNumMacroFrames && !pActiveFrameFlags[i])  i++;
        ParsePoints[NumParsePointsDetected].Deg.Start = FramesToSamples(i);
    }

    if (!NumParsePointsDetected)
    {
        NumParsePointsDetected = 1;
        ParsePoints[0].Deg.End = FramesToSamples(mNumMacroFrames);
    }

    return NumParsePointsDetected;
}

//Do a coarse search for potential endpoints of the reparse points.
void CTempAlignment::SearchInactiveSegments(REPARSE_POINT* ReparsePoints, int
MaxReparsePoints, int* pActiveFrameFlags, bool WorkOnDegSignal)
{
    int MinPauseDurationInFrames = MSecondsToFrames(500);
    for (int r=0; r<MaxReparsePoints; r++)
    {
        if (WorkOnDegSignal)
        {
            int End = SamplesToFrames(ReparsePoints[r].Deg.Start);
            while (End<mNumMacroFrames && !pActiveFrameFlags[End]) End++;
            End = GetNextPauseStartFrameIndex(pActiveFrameFlags, End, mNumMacroFrames);
```

```
            ReparsePoints[r].Deg.End = FramesToSamples(End);
        }
        else
        {
            int End = SamplesToFrames(ReparsePoints[r].Ref.Start);
            while (End<mNumMacroFrames && !pActiveFrameFlags[End]) End++;

            End = GetNextPauseStartFrameIndex(pActiveFrameFlags, End, mNumMacroFrames);
            ReparsePoints[r].Ref.End = FramesToSamples(End);
        }
    }
}

int CTempAlignment::GetNextPauseStartFrameIndex(int* pVec, int StartIdx, int VecLen)
{
    int MinPauseDurationInFrames = MSecondsToFrames(500);
    bool EndFound=false;
    int End = StartIdx;

    while (!EndFound && End<VecLen)
    {
        while (End<mNumMacroFrames && pVec[End]) End++;
        int RequiredEnd=End+MinPauseDurationInFrames;
        while (End<mNumMacroFrames && !pVec[End] && RequiredEnd) RequiredEnd--;
        if (!RequiredEnd)
            EndFound=true;
        else
            if (End<mNumMacroFrames)
                End += MinPauseDurationInFrames-RequiredEnd;
    }
    return End;
}

inline void DeleteReparsePoint(int IndexToDelete, REPARSE_POINT*ReparsePoints, int*
Num)
{
    for (int d=IndexToDelete; d<*Num-1; d++)
        ReparsePoints[d] = ReparsePoints[d+1];
    *Num = *Num-1;
}

inline void DuplicateReparsePoint(int IndexToDup, REPARSE_POINT*ReparsePoints, int*
Num)
{
    for (int d=*Num; d>=IndexToDup; d--)
        ReparsePoints[d+1] = ReparsePoints[d];
    *Num = *Num+1;
}

//Return 1 if a match could be found, -1 if the sections were modified, 0 if nothing
could be done
int CTempAlignment::FindMatchingSection(REPARSE_POINT* ReparsePointsRef, int
NumReparsePointsRef, REPARSE_POINT* ReparsePointsDeg, int NumReparsePointsDeg,
OTA_FLOAT SNRdB, int SectionIndex, int* pRefPointsRemaining, int* pDegPointsRemaining)
{
    int rc=0;
    int d, r;
    r=SectionIndex;

    if (r<NumReparsePointsDeg)
    {
        int LengthRef = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start;
        int LengthDeg = ReparsePointsDeg[r].Deg.End - ReparsePointsDeg[r].Deg.Start;
        if (abs(LengthDeg-LengthRef)<MSecondsToSamples(200) || r==NumReparsePointsRef)
        {
            ;
            rc = 1;
        }
        else if (LengthRef>LengthDeg)
        {
            int ms200 = MSecondsToSamples(200);
            int ms250 = MSecondsToSamples(250);
            int ms275 = MSecondsToSamples(275);
            int ms350 = MSecondsToSamples(350);

            int DistanceForCombination = ms250;
```

```
            if (SNRdB<8.0) DistanceForCombination *=2;

            int StartDiff =
ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r].Deg.Start;
            int LengthOfTwo=0;
            int LengthOfThree=0;
            if (r<NumReparsePointsDeg-1)
                LengthOfTwo = ReparsePointsDeg[r+1].Deg.End -
ReparsePointsDeg[r].Deg.Start;
            if (r<NumReparsePointsDeg-2)
                LengthOfThree = ReparsePointsDeg[r+2].Deg.End -
ReparsePointsDeg[r].Deg.Start;

            if (r<NumReparsePointsDeg-1 &&
(OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>2.0 && LengthDeg<ms250 &&
                    abs(StartDiff) -
abs(ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r+1].Deg.Start)
>ms200
                    )
            {
                ;
                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg);
            }

            else if (abs(LengthOfTwo-LengthRef)<DistanceForCombination)
            {
                if (r<NumReparsePointsDeg-1 && abs(ReparsePointsDeg[r+2].Deg.End -
ReparsePointsDeg[r+1].Deg.Start-LengthRef)<ms250
                    && abs(StartDiff) >
abs(ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r+1].Deg.Start)
)
                {
                    ;
                    DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg);
                }
                else
                {
                    ;
                    ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r+1].Deg.End;
                    ReparsePointsDeg[r].IsVirtualPoint = true;
                    DeleteReparsePoint(r+1, ReparsePointsDeg, &NumReparsePointsDeg);
                }
            }
            else if (r<NumReparsePointsDeg-2 && abs(LengthOfThree-LengthRef)<ms200)
            {
                ;
                ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r+2].Deg.End;
                ReparsePointsDeg[r].IsVirtualPoint = true;
                for (d=r+3; d<NumReparsePointsDeg; d++)
                    ReparsePointsDeg[d-2] = ReparsePointsDeg[d];
                NumReparsePointsDeg-=2;
            }

            else if (r<NumReparsePointsDeg-1 &&
(OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>2.0 && LengthDeg<ms200
                &&
abs(ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r+1].Deg.Start)<MSec
ondsToSamples(1000) )
            {
                ;
                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg);
            }
            else if (r<NumReparsePointsDeg-1 &&
(OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>2.0
                                        && abs(ReparsePointsDeg[r+1].Deg.End -
ReparsePointsDeg[r+1].Deg.Start-LengthRef)<ms20
0)
            {
                ;
                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg);
            }
            else if (r==NumReparsePointsDeg-1 ||
ReparsePointsDeg[r+1].Deg.Start-ReparsePointsDeg[r].Deg.End>MSecondsToSampl
es(1000))
            {
```

```
                if (1 && ReparsePointsDeg[r].Deg.End>1 &&
abs(ReparsePointsDeg[r].Deg.Start-ReparsePointsRef[r].Ref.Start) >
abs(ReparsePointsDeg[r].Deg.End-ReparsePointsRef[r].Ref.End)
                    && ( (r>0)? ReparsePointsDeg[r].Deg.Start >
ReparsePointsDeg[r-1].Deg.End+MSecondsToFrames(50) : 1 ) )
                {
                    ;
                    ReparsePointsDeg[r].Deg.Start = ReparsePointsDeg[r].Deg.End-
LengthRef;
                    ReparsePointsDeg[r].Deg.Start = (((ReparsePointsDeg[r].Deg.Start) >
(0)) ? (ReparsePointsDeg[r].Deg.Start) : (0));

                }
                else
                {
                    ;
                    ReparsePointsDeg[r].Deg.End =
ReparsePointsDeg[r].Deg.Start+LengthRef;
                }
            }
            else if (0 && r<NumReparsePointsDeg-1 &&
(OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>5)
            {
                ;
                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg);
            }
            else
            {

                if (1 && r<NumReparsePointsDeg-1 && r>0
                    && ms350 < abs(ReparsePointsRef[r-1].Ref.Start -
ReparsePointsDeg[r-1].Deg.Start -(ReparsePointsRef[r].Ref.Start -
ReparsePointsDeg[r].Deg.Start)) )
                {
                    ;
                    DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg);
                }
                else
                {
                    if (abs(LengthOfTwo-LengthRef)<DistanceForCombination*1.5)
                    {
                        ;
                        int a = (LengthRef - LengthOfTwo) / 2;
                        ReparsePointsDeg[r].Deg.Start = (((0) >
(ReparsePointsDeg[r].Deg.Start+a)) ? (0) :
(ReparsePointsDeg[r].Deg.Start+a));
                        ReparsePointsDeg[r].Deg.End   = ReparsePointsDeg[r].Deg.Start +
LengthRef;
                        DeleteReparsePoint(r+1, ReparsePointsDeg,
&NumReparsePointsDeg);
                    }
                    else if (1 && ReparsePointsDeg[r].Deg.Start>1 &&
abs(ReparsePointsDeg[r].Deg.Start-ReparsePointsRef[r].Ref.Start) >
abs(ReparsePointsDeg[r].Deg.End-ReparsePointsRef[r].Ref.End) )
                    {
                        ;
                        ReparsePointsDeg[r].Deg.Start = ReparsePointsDeg[r].Deg.End-
LengthRef;
                        ReparsePointsDeg[r].Deg.Start =
(((ReparsePointsDeg[r].Deg.Start) > (0)) ?
(ReparsePointsDeg[r].Deg.Start) : (0));

                    }
                    else
                    {
                        ;
                        ReparsePointsDeg[r].Deg.End =
ReparsePointsDeg[r].Deg.Start+LengthRef;
                    }
                }
            }
            rc = -1;
        }
        else

        {
```

```
            int ms200 = MSecondsToSamples(200);
            int ms250 = MSecondsToSamples(250);
            int ms350 = MSecondsToSamples(350);
            int StartDiff =
ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r].Deg.Start;
            int LengthOfTwo=0;
            int LengthOfThree=0;
            if (r<NumReparsePointsRef-1)
                LengthOfTwo = ReparsePointsRef[r+1].Ref.End -
ReparsePointsRef[r].Ref.Start;
            if (r<NumReparsePointsRef-2)
                LengthOfTwo = ReparsePointsRef[r+2].Ref.End -
ReparsePointsRef[r].Ref.Start;

            if (abs(LengthOfTwo-LengthDeg)<ms250)
            {
                if (r<NumReparsePointsRef-1 && abs(ReparsePointsRef[r+2].Ref.End -
ReparsePointsRef[r+1].Ref.Start-LengthDeg)<ms250
                     && abs(StartDiff) >
abs(ReparsePointsDeg[r].Deg.Start-ReparsePointsRef[r+1].Ref.Start)
)
                {
                    ;
                    DeleteReparsePoint(r, ReparsePointsRef, &NumReparsePointsRef);
                }
                else
                {
                    ;
                    ReparsePointsRef[r].Ref.End = ReparsePointsRef[r+1].Ref.End;
                    ReparsePointsRef[r].IsVirtualPoint = true;
                    DeleteReparsePoint(r+1, ReparsePointsRef, &NumReparsePointsRef);
                }
            }
            else if (r<NumReparsePointsRef-2 && abs(LengthOfThree-LengthDeg)<ms200)
            {
                ;
                ReparsePointsRef[r].Ref.End = ReparsePointsRef[r+2].Ref.End;
                ReparsePointsRef[r].IsVirtualPoint = true;
                for (d=r+2; d<NumReparsePointsDeg; d++)
                    ReparsePointsRef[d] = ReparsePointsRef[d+2];
                NumReparsePointsRef-=2;
            }

            else
            {
                int LengthOfTwoRef=0;
                if (r<NumReparsePointsRef-1)
                    LengthOfTwoRef = ReparsePointsRef[r+1].Ref.End -
ReparsePointsRef[r].Ref.Start;
                bool IsGoodFit = (abs(LengthOfTwoRef-LengthDeg)<ms250);
                if (NumReparsePointsDeg<NumReparsePointsRef || IsGoodFit)
                {
                    ;
                    for (d=NumReparsePointsDeg; d>r; d--)
                        ReparsePointsDeg[d] = ReparsePointsDeg[d-1];
                    ReparsePointsDeg[r+1].IsVirtualPoint = true;
                    int PauselLenRef =
ReparsePointsRef[r+1].Ref.Start-ReparsePointsRef[r].Ref.End;
                    ReparsePointsDeg[r].Deg.End =
ReparsePointsDeg[r].Deg.Start+LengthRef;
                    ReparsePointsDeg[r+1].Deg.Start =
ReparsePointsDeg[r].Deg.End+PauselLenRef;
                    if (ReparsePointsDeg[r+1].Deg.Start>=ReparsePointsDeg[r+1].Deg.End)
                        ReparsePointsDeg[r+1].Deg.Start = ReparsePointsDeg[r].Deg.End
+1;
                    NumReparsePointsDeg++;
                }
                else
                {
                    ;
                    ReparsePointsDeg[r].Deg.End =
ReparsePointsDeg[r].Deg.Start+LengthRef;
                }
            }
            rc = -1;
        }
```

```
    }
    else
    {
        int LengthRef = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start;

        if (LengthRef<MSecondsToSamples(80))
        {
            ;
            DeleteReparsePoint(r, ReparsePointsRef, &NumReparsePointsRef);
            rc = -1;
        }
        else if (LengthRef<MSecondsToSamples(500))
        {
            ;
            int AddLen = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r-1].Ref.End;
            ReparsePointsRef[r].Ref.End = ReparsePointsRef[r-1].Ref.End;
            ReparsePointsDeg[r-1].Ref.End += AddLen;
            NumReparsePointsRef--;
            rc = -1;
        }
        else
        {
            ;
            int Offset = 0;
            if (r) Offset = ReparsePointsDeg[r-1].Deg.Start -
ReparsePointsRef[r-1].Ref.Start;
            ReparsePointsDeg[r].Deg.Start = ReparsePointsRef[r].Ref.Start + Offset;
            ReparsePointsDeg[r].Deg.End = ReparsePointsRef[r].Ref.End + Offset;
            ReparsePointsDeg[r].Reliability = -1;
            NumReparsePointsDeg++;
            rc = -1;
        }
    }

    //Combine any sections which may be overlapping now
    while (r<NumReparsePointsDeg-1 &&
ReparsePointsDeg[r+1].Deg.Start<ReparsePointsDeg[r].Deg.End)
    {
        ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r+1].Deg.End;
        DeleteReparsePoint(r+1, ReparsePointsDeg, &NumReparsePointsDeg);
    }

    while (r<NumReparsePointsRef-1 &&
ReparsePointsRef[r+1].Ref.Start<ReparsePointsRef[r].Ref.End)
    {
        ReparsePointsRef[r].Ref.End = ReparsePointsRef[r+1].Ref.End;
        DeleteReparsePoint(r+1, ReparsePointsRef, &NumReparsePointsRef);
    }

    *pDegPointsRemaining = NumReparsePointsDeg;
    *pRefPointsRemaining = NumReparsePointsRef;
    return rc;
}

void CTempAlignment::FindMatchingSectionAInBWithCorrelation(REPARSE_POINT*
ReparsePointsA, int NumReparsePointsA, CFeatureVector* pVecA, REPARSE_POINT*
ReparsePointsB, int NumReparsePointsB, CFeatureVector* pVecB, int r, int* pDelay,
OTA_FLOAT* pReliability)
{
    int Delay;
    OTA_FLOAT Reliability;
    SECTION SectionA, SectionB;

    ;
    int ALen = ReparsePointsA[r].Ref.Len();
    SectionA.Start = (int)(SamplesToFrames(ReparsePointsA[r].Ref.Start+0.05*ALen));
    SectionA.End   = (int)(SamplesToFrames(ReparsePointsA[r].Ref.End-0.05*ALen));
    ALen = FramesToSamples(SectionA.Len());

    SectionB.Start = (((0) > ((int)SamplesToFrames(ReparsePointsB[r].Deg.Start -
0.1*ALen))) ? (0) : ((int)SamplesToFrames(ReparsePointsB[r].Deg.Start -
0.1*ALen)));
    SectionB.End = SamplesToFrames(ReparsePointsB[r].Deg.End);

    if(SectionA.Len() > 0 && SectionB.Len() > 0)
    {
```

```
        ;
        int MaxInactivity = (int)SamplesToFrames(0.2*ALen);
        int InactivityCount=0;
        while ( SectionB.End<pVecB->mSize && InactivityCount<MaxInactivity)
        {
            while (SectionB.End<pVecB->mSize && pVecB->mpVector[SectionB.End]>500)
SectionB.End++;

            InactivityCount=0;
            while (SectionB.End<pVecB->mSize && pVecB->mpVector[SectionB.End]<=500 &&
InactivityCount++<MaxInactivity) SectionB.End++;
        }

        //If the gap between this and the next active B section is less than 700ms,
        //A check is made whether the next deg section shall be included in the search
as well.
        ;
        if (r<NumReparsePointsB-1 &&
ReparsePointsB[r+1].Deg.Start-ReparsePointsB[r].Deg.End<MSecondsToSamples(700))
        {
            int NextStop = SamplesToFrames(ReparsePointsB[r+1].Deg.End);
            if (NextStop-SectionB.Start > SectionA.End-SectionA.Start)
                SectionB.End = (((SectionB.End) <
(SamplesToFrames(ReparsePointsB[r+1].Deg.End))) ? (SectionB.End) :
(SamplesToFrames(ReparsePointsB[r+1].Deg.End)));
            if (SectionB.End>pVecB->mSize)  SectionB.End = pVecB->mSize;
        }
        else SectionB.End = (((SectionB.End) <
(SamplesToFrames(ReparsePointsB[r].Deg.End)+MSecondsToFrames(500))) ?
(SectionB.End) :
(SamplesToFrames(ReparsePointsB[r].Deg.End)+MSecondsToFrames(500)));

        int AFrames = pVecA->mSize;

        if (SectionA.End-SectionA.Start>1 && AFrames / (SectionA.End-SectionA.Start)<15
&& SectionB.End-SectionB.Start>32 &&
            SectionB.End>SectionB.Start && SectionA.End>SectionA.Start)
        {
            //If the ref section is longer than the deg section, then extend ref and
reverse the search.
            if (SectionB.End-SectionB.Start<SectionA.End-SectionA.Start)
            {
                ;
                SectionB.End = SamplesToFrames(ReparsePointsB[r].Deg.End);
                SectionA.Start = SamplesToFrames(ReparsePointsA[r].Ref.Start);
                SectionA.End   = SamplesToFrames(ReparsePointsA[r].Ref.End);
                Delay = SectionB.Start - SectionA.Start;
                Delay = FindSectionAInSectionB(&SectionB, &SectionA, pVecB, pVecA,
&Reliability, 10);
                Delay = Delay -SectionB.Start +SectionA.Start;
                Delay = -Delay;
            }
            else
            {
                ;
                Delay = SectionA.Start - SectionB.Start;
                Delay = FindSectionAInSectionB(&SectionA, &SectionB, pVecA, pVecB,
&Reliability, 10);
                Delay = Delay -SectionA.Start +SectionB.Start;
            }
        }
        else
        {
            Reliability=0;
            Delay = 0;
        }
    }
    else
    {
        ;
        Reliability=0;
        Delay = 0;
    }
    *pReliability = Reliability;
    *pDelay = Delay;
}
```

```
REPARSE_POINT CTempAlignment::AllocateSectionWithCorrelationBasedOnRefInfo(int Offset,
int Len, int r, REPARSE_POINT* ReparsePointsRef, int NumReparsePointsRef,
CFeatureVector* pVecRefLin, REPARSE_POINT* ReparsePointsDeg, int NumReparsePointsDeg,
CFeatureVector* pVecDegLin, OTA_FLOAT SNRdB, OTA_FLOAT NoiseLevel, bool*
pSectionAllocated, bool* pSectionAccepted)
{
    REPARSE_POINT Corr1ReparsePointDeg;
    OTA_FLOAT     Corr1Reliability=-1;
    int           Corr1Delay;
    bool          Corr1SectionAllocated=false;
    bool          Corr1SectionAccepted=false;

    CFeatureVector* pRef = pVecRefLin;
    CFeatureVector* pDeg = pVecDegLin;
    int VecLenRefSamples = FramesToSamples(pRef->mSize);
    int VecLenDegSamples = FramesToSamples(pDeg->mSize);

    Corr1ReparsePointDeg.DelayInSamples = 0;
    Corr1ReparsePointDeg.Reliability = -1;

    SECTION SecRef = ReparsePointsRef[r].Ref;
    SECTION SecDeg;
    if (r<NumReparsePointsRef)
    {
        ;

        Corr1ReparsePointDeg.Ref = ReparsePointsRef[r].Ref;
        if (r<NumReparsePointsDeg)
        {
            Corr1ReparsePointDeg = ReparsePointsDeg[r];
        }
        else
        {
            Corr1ReparsePointDeg.Deg.Start = 0;
            if (r)  Corr1ReparsePointDeg.Deg.Start = ReparsePointsDeg[r-1].Deg.End;
            Corr1ReparsePointDeg.Deg.End = VecLenDegSamples;
        }

        SecRef.Start += Offset;
        if (Len>0) SecRef.End = SecRef.Start+Len;

        SecDeg = SecRef;

        SecDeg.Start -= mOverallDelayEstimate;
        SecDeg.Start = ((((((SecDeg.Start) < (0)) ? (SecDeg.Start) : (0))) >
(SecDeg.Start - MSecondsToSamples(700))) ? ((((SecDeg.Start) < (0)) ?
(SecDeg.Start) : (0))) : (SecDeg.Start - MSecondsToSamples(700)));

        if (r<NumReparsePointsRef-1)
        {
            int k;
            SecDeg.End = ReparsePointsRef[r+1].Ref.Start - mOverallDelayEstimate;
            if (ReparsePointsRef[r+1].Ref.Start - ReparsePointsRef[r].Ref.End<
MSecondsToSamples(500))
                SecDeg.End += MSecondsToSamples(500);

            int End = ReparsePointsRef[r].Ref.End - mOverallDelayEstimate;
            for (k=r; k<NumReparsePointsDeg && ReparsePointsDeg[k].Deg.Start<End; k++)
;

            if (k>0 && k<NumReparsePointsDeg &&
SecDeg.End>ReparsePointsDeg[k].Deg.Start)
                SecDeg.End = ReparsePointsDeg[k-1].Deg.End +
(ReparsePointsDeg[k].Deg.Start-ReparsePointsDeg[k-1].Deg.End)/2;
        }
        else SecDeg.End = VecLenDegSamples;

        if (SecDeg.Start<0)
        {
            SecRef.Start -= SecDeg.Start;

            if (SecRef.Len() < SamplesToMSeconds(1000))
                SecRef.End -= SecDeg.Start;
            SecDeg.End -= SecDeg.Start;
            SecDeg.Start = 0;
```

```
        }

        int TooLong = SecDeg.End-FramesToSamples(pDeg->mSize);
        if (TooLong>0)
        {
            SecRef.End -= TooLong;
            SecRef.Start -= TooLong;
            SecDeg.End += TooLong;
            SecDeg.Start -= TooLong;
        }

        if (SecRef.Start<0) SecRef.Start = 0;
        if (SecRef.End>VecLenRefSamples) SecRef.End = VecLenRefSamples;
        if (SecDeg.Start<0) SecDeg.Start = 0;
        if (SecDeg.End>VecLenDegSamples) SecDeg.End = VecLenDegSamples;

        if (r==0) SecDeg.Start = 0;
        if (r==NumReparsePointsRef-1) SecDeg.End = VecLenDegSamples;

        if (SecDeg.Len()<SecRef.Len())
            ;

        int HistoShiftFrames=4;
        int HistoShiftSamples = FramesToSamples(HistoShiftFrames);

        int HistoLen=12;
        int lenghtIndicator;
        lenghtIndicator = (SamplesToMSeconds(VecLenRefSamples)/2800);

        if (lenghtIndicator>4)  HistoLen = (8 + lenghtIndicator);
        if (HistoLen>18) HistoLen=18;

        int MaxSpaceForHistogram =
(VecLenRefSamples-SecRef.Len()-SecRef.Start)/HistoShiftSamples;
        MaxSpaceForHistogram = (((MaxSpaceForHistogram) <
((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples)) ?
(MaxSpaceForHistogram) :
((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples));

        if (MaxSpaceForHistogram<HistoLen)
        {
            int TotalShiftSamples = HistoShiftSamples * HistoLen;
            if(SecRef.Start>=TotalShiftSamples) SecRef.Start -= TotalShiftSamples;
            SecRef.End -= TotalShiftSamples;
            if(SecDeg.Start>=TotalShiftSamples) SecDeg.Start -= TotalShiftSamples;
            SecDeg.End -= TotalShiftSamples;

            HistoLen = (((HistoLen) <
((VecLenRefSamples-SecRef.Len()-SecRef.Start)/HistoShiftSamples)) ?
(HistoLen) :
((VecLenRefSamples-SecRef.Len()-SecRef.Start)/HistoShiftSamples));
            HistoLen = (((HistoLen) <
((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples)) ?
(HistoLen) :
((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples));
        }

        ;
        ;
        int DelayOffset = (SecRef.Start-SecDeg.Start);

        Corr1Delay = FindDelayStrict((MAT_HANDLE)mProcessData.mpMathlibHandle,

                    pRef->mpVector+SamplesToFrames(SecRef.Start),
SamplesToFrames(SecRef.Len()),
                    pDeg->mpVector+SamplesToFrames(SecDeg.Start),
SamplesToFrames(SecDeg.Len()),
                    HistoLen, HistoShiftFrames, 0, &Corr1Reliability);
        Corr1Delay = FramesToSamples(Corr1Delay);
        Corr1Delay = -Corr1Delay+DelayOffset;
        ;
    }

    SecDeg.Start = ReparsePointsRef[r].Ref.Start - Corr1Delay;
    SecDeg.End = ReparsePointsRef[r].Ref.End - Corr1Delay;
```

```
    if (SecDeg.Start<0)
    {
        SecDeg.Start = 0;
    }
    if (SecDeg.End>FramesToSamples(pDeg->mSize))
    {
        SecDeg.End = FramesToSamples(pDeg->mSize);
    }
    if (SecRef.End>FramesToSamples(pRef->mSize))
    {
        assert(-1);
        SecDeg.End -= SecRef.End - FramesToSamples(pRef->mSize);
    }

    if (SecRef.End>SecRef.Start && SecDeg.End>SecDeg.Start)
    {
        ;
        Corr1ReparsePointDeg.Deg = SecDeg;
        Corr1ReparsePointDeg.Ref = ReparsePointsRef[r].Ref;
        Corr1ReparsePointDeg.Reliability = Corr1Reliability;
        Corr1SectionAllocated = true;
    }

    bool DegInfoOk=false;
    if (r<NumReparsePointsDeg &&
abs(ReparsePointsDeg[r].Deg.Len()-ReparsePointsRef[r].Ref.Len())<MSecondsToSamples(
250) )
        DegInfoOk = true;

    if (Corr1SectionAllocated )
    {
        int Offset;
        if (r<NumReparsePointsDeg && DegInfoOk)
            Offset = Corr1ReparsePointDeg.Deg.Start-ReparsePointsDeg[r].Deg.Start;
        else
        {
            if (r)
            {
                int DelayChange = ReparsePointsDeg[r-1].DelayInSamples-Corr1Delay;
                int StartWithOldDelay = Corr1ReparsePointDeg.Deg.Start+DelayChange;
                int OldPause = StartWithOldDelay - ReparsePointsDeg[r-1].Deg.End;
                Offset = (int)(-DelayChange*0.5);

                Offset = MSecondsToSamples(50);
            }
            else Offset = 0;
        }
        Corr1ReparsePointDeg.Deg.Start -= Offset;

        Corr1ReparsePointDeg.Ref.Start -= Offset;

        if ( (SNRdB<10.0 && Corr1Reliability>=0.5) || Corr1Reliability>0.6)
            Corr1SectionAccepted=true;
    }

    Corr1ReparsePointDeg.Reliability = Corr1Reliability;
    Corr1ReparsePointDeg.DelayInSamples = Corr1Delay;

    *pSectionAccepted = Corr1SectionAccepted;
    *pSectionAllocated = Corr1SectionAllocated;
    return Corr1ReparsePointDeg;
}

REPARSE_POINT CTempAlignment::AllocateSectionWithCorrelationBasedOnVADInfo(int r,
REPARSE_POINT* ReparsePointsRef, int NumReparsePointsRef, CFeatureVector* pVecRef,
REPARSE_POINT* ReparsePointsDeg, int NumReparsePointsDeg, CFeatureVector* pVecDeg,
OTA_FLOAT SNRdB, bool* pSectionAllocated, bool* pSectionAccepted)
{
    REPARSE_POINT Corr2ReparsePointDeg;
    OTA_FLOAT     Corr2Reliability=-1;
    int           Corr2Delay=0;
    bool          Corr2SectionAllocated=false;
    bool          Corr2SectionAccepted=false;

    if (r<NumReparsePointsDeg)
    {
```

```
        Corr2ReparsePointDeg = ReparsePointsDeg[r];
        Corr2ReparsePointDeg.DelayInSamples = 0;
        Corr2ReparsePointDeg.Reliability = -1;
    }

    if (r<NumReparsePointsRef && r<NumReparsePointsDeg)
    {

        ;
        int RefLen = ReparsePointsRef[r].Ref.End-ReparsePointsRef[r].Ref.Start;
        int DegLen = ReparsePointsDeg[r].Deg.End-ReparsePointsDeg[r].Deg.Start;
        if (0.9*RefLen < DegLen)
        {
            ;
            FindMatchingSectionAInBWithCorrelation(ReparsePointsRef,
NumReparsePointsRef, pVecRef, ReparsePointsDeg, NumReparsePointsDeg,
pVecDeg, r, &Corr2Delay, &Corr2Reliability);
            Corr2Delay = -FramesToSamples(Corr2Delay);
            ;
        }
        else
        {
            ;
            FindMatchingSectionAInBWithCorrelation(ReparsePointsDeg,
NumReparsePointsDeg, pVecDeg, ReparsePointsRef, NumReparsePointsRef,
pVecRef, r, &Corr2Delay, &Corr2Reliability);
            Corr2Delay = FramesToSamples(Corr2Delay);
            Corr2Delay = Corr2Delay;
            ;
        }

        int DegStart = ReparsePointsRef[r].Ref.Start - Corr2Delay;
        if (DegStart<0)
        {
            if (ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start <
-DegStart)
            {
                ;
                Corr2ReparsePointDeg.Deg.Start = 0;
                Corr2ReparsePointDeg.Deg.End   = ReparsePointsRef[r].Ref.End   -
Corr2Delay;
                Corr2ReparsePointDeg.Ref       = ReparsePointsRef[r].Ref;
                Corr2ReparsePointDeg.Ref.Start += -DegStart;
                Corr2ReparsePointDeg.Reliability = Corr2Reliability;
                Corr2SectionAllocated = true;
            }
            else
            {
                Corr2SectionAllocated = false;
            }
        }
        else
        {
            ;
            Corr2ReparsePointDeg.Deg.Start = ReparsePointsRef[r].Ref.Start -
Corr2Delay;
            Corr2ReparsePointDeg.Deg.End   = ReparsePointsRef[r].Ref.End   -
Corr2Delay;
            Corr2ReparsePointDeg.Ref       = ReparsePointsRef[r].Ref;
            Corr2ReparsePointDeg.Reliability = Corr2Reliability;
            Corr2SectionAllocated = true;
        }

        int Offset = Corr2ReparsePointDeg.Deg.Start-ReparsePointsDeg[r].Deg.Start;

        if (r>0 && ReparsePointsRef[r].Ref.Start>ReparsePointsRef[r].Ref.End)

        {
            Corr2ReparsePointDeg.Deg.Start -= Offset;

            Corr2ReparsePointDeg.Ref.Start -= Offset;
        }

        if (Corr2SectionAllocated && (SNRdB<10.0 && Corr2Reliability>=0.4) ||
Corr2Reliability>=0.8)
        {
```

```
            Corr2SectionAccepted = true;
        }
    }
    else Corr2Reliability=0;

    Corr2ReparsePointDeg.Reliability = Corr2Reliability;
    Corr2ReparsePointDeg.DelayInSamples = Corr2Delay;

    *pSectionAccepted = Corr2SectionAccepted;
    *pSectionAllocated = Corr2SectionAllocated;
    return Corr2ReparsePointDeg;
}

void CTempAlignment::CheckSectionLimits(REPARSE_POINT* ReparsePointsDeg, int
NumReparsePointsRef, int *NumReparsePointsDeg)
{
    ;
    for (int r=0; r<NumReparsePointsRef; r++)
    {
        if (ReparsePointsDeg[r].Deg.End>mppSignals[1]->mSignalLength)
        {
            ;
            ReparsePointsDeg[r].Deg.End=mppSignals[1]->mSignalLength;
        }
        if (ReparsePointsDeg[r].Ref.End>mppSignals[0]->mSignalLength)
        {
            ;
            ReparsePointsDeg[r].Ref.End=mppSignals[0]->mSignalLength;
        }

        if (ReparsePointsDeg[r].Deg.Start<0)
        {
            ;
            ReparsePointsDeg[r].Ref.Start -= ReparsePointsDeg[r].Deg.Start;
            ReparsePointsDeg[r].Deg.Start = 0;
        }

        if (ReparsePointsDeg[r].Deg.End<ReparsePointsDeg[r].Deg.Start)
        {
            ;
            DeleteReparsePoint(r, ReparsePointsDeg, NumReparsePointsDeg);
            NumReparsePointsRef--;
            continue;
        }
        if (ReparsePointsDeg[r].Ref.Start<0)
        {
            ;
            if (-ReparsePointsDeg[r].Ref.Start<MSecondsToSamples(300))
                ReparsePointsDeg[r].Deg.Start -= ReparsePointsDeg[r].Ref.Start;
            ReparsePointsDeg[r].Ref.Start = 0;
        }
        if (ReparsePointsDeg[r].Ref.End<ReparsePointsDeg[r].Ref.Start)
        {
            ;
            DeleteReparsePoint(r, ReparsePointsDeg, NumReparsePointsDeg);
            NumReparsePointsRef--;
            continue;
        }

    }
}

void CTempAlignment::AllocateSectionsFromDelayEstimate(REPARSE_POINT* ReparsePointsDeg,
REPARSE_POINT* ReparsePointsRef, int NumReparsePointsRef, int *NumReparsePointsDeg, int
DelayEstimate, OTA_FLOAT DelayEstimateReliability)
{
    int i;
    ;

    for (i=0; i<NumReparsePointsRef; i++)
    {
        ReparsePointsDeg[i].Deg.Start = ReparsePointsRef[i].Ref.Start-DelayEstimate;
        ReparsePointsDeg[i].Deg.End   = ReparsePointsRef[i].Ref.End-DelayEstimate;
        ReparsePointsDeg[i].Ref = ReparsePointsRef[i].Ref;
        ReparsePointsDeg[i].Reliability = DelayEstimateReliability;
        if (ReparsePointsDeg[i].Deg.Start<0)
```

```
        {
            ReparsePointsDeg[i].Ref.Start -= ReparsePointsDeg[i].Deg.Start;
            ReparsePointsDeg[i].Deg.Start = 0;
            if (ReparsePointsDeg[i].Ref.Start>=ReparsePointsDeg[i].Ref.End)
            {
                ;
                DeleteReparsePoint(i, ReparsePointsRef, &NumReparsePointsRef);
                i--;
            }
        }
    }
    *NumReparsePointsDeg = NumReparsePointsRef;
}

//Identify the position of all reparse points.
//Only the deg startpoint is located roughly, the ref startpoint is set later
//during the delay search when we know the delay of each preceeding active section.
//This will not yet fill in the Delay element of the REPARSE_POINT structs.
//Returns the number of found reparse points.
int CTempAlignment::IdentifyReparsePoints(REPARSE_POINT* ReparsePointsDeg, int
MaxParsePoints, int* pActiveFrameFlagsDeg, int OverallDelayEstimate, OTA_FLOAT
OverallDelayEstimateReliability,
                                          int OverallDelayEstimate1st, OTA_FLOAT
OverallDelayEstimateReliability1st, int
OverallDelayEstimate2nd, OTA_FLOAT
OverallDelayEstimateReliability2nd)
{
    bool Done = false;
    int i;
    int NumReparsePointsDeg=-1;
    int OriginalNumReparsePointsDeg=-1;
    bool TakeRefPointsOnly = false;
    int OriginalDegSectionLen[100];

    ;

    for (i=0; i<MaxParsePoints; i++)
        ReparsePointsDeg[i].Reliability = -1;

    mProcessData.Init(1, 1.0);

    OTA_FLOAT SigLevel, NoiseLevel, NoiseThreshold;
    mpActiveFrameDetection->GetLevels(1, 0, mProcessData.mStepSize, &NoiseLevel,
&SigLevel, &NoiseThreshold);
    OTA_FLOAT SNRdB = 10*log10(SigLevel / NoiseLevel);
    ;

    ;

    REPARSE_POINT* ReparsePointsRef = new REPARSE_POINT[MaxParsePoints];
    int *pActiveFrameFlagsRef = new int[mNumMacroFrames];
    mpActiveFrameDetection->GetActiveFrameFlags(0, 0, mProcessData.mStepSize,
pActiveFrameFlagsRef, mNumMacroFrames);
    int NumReparsePointsRef = SearchActiveSegments(ReparsePointsRef, MaxParsePoints,
pActiveFrameFlagsRef);
    for (i=0; i<NumReparsePointsRef; i++)
        ReparsePointsRef[i].Ref = ReparsePointsRef[i].Deg;

    if (OverallDelayEstimateReliability>0.6)

    {
        AllocateSectionsFromDelayEstimate(ReparsePointsDeg, ReparsePointsRef,
NumReparsePointsRef, &NumReparsePointsDeg, OverallDelayEstimate,
OverallDelayEstimateReliability);
        OriginalNumReparsePointsDeg = NumReparsePointsDeg;
        for (i=0; i<NumReparsePointsDeg; i++)
            OriginalDegSectionLen[i] = ReparsePointsDeg[i].Deg.Len();
    }

    //Do a search for active segments in the deg signal. These may be very inaccurate
due to the poor signal quality.
    if (!Done)
    {
        ;
        NumReparsePointsDeg = SearchActiveSegments(ReparsePointsDeg, 100,
pActiveFrameFlagsDeg);
```

```
        for (i=0; i<NumReparsePointsDeg; i++)
            ReparsePointsDeg[i].Deg.Start = ReparsePointsDeg[i].Deg.Start;

        for (i=0; i<NumReparsePointsDeg; i++)
            ReparsePointsDeg[i].IsVirtualPoint = false;

    }

    if (!Done)
    {
        mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH);

        if (SNRdB<11)
        {
            ;

            int LastStart = ReparsePointsDeg[NumReparsePointsDeg-1].Deg.Start;
            for (i=0; i<NumReparsePointsDeg; i++)
                ReparsePointsDeg[i].Deg.Start = GetNearestStart(1,
ReparsePointsDeg[i].Deg.Start+MSecondsToFrames(250), 0,
false)-MSecondsToFrames(250);

            SearchInactiveSegments(ReparsePointsDeg, NumReparsePointsDeg,
pActiveFrameFlagsDeg, true);

            for (i=0; i<NumReparsePointsDeg; i++)
            {
                while (i<NumReparsePointsDeg-1 &&
ReparsePointsDeg[i+1].Deg.Start<ReparsePointsDeg[i].Deg.End)
                {
                    ReparsePointsDeg[i].Deg.End = ReparsePointsDeg[i+1].Deg.End;
                    DeleteReparsePoint(i+1, ReparsePointsDeg, &NumReparsePointsDeg);
                }
            }

            if (ReparsePointsDeg[NumReparsePointsDeg-1].Deg.End <
ReparsePointsDeg[NumReparsePointsDeg-1].Deg.Start)
                ReparsePointsDeg[NumReparsePointsDeg-1].Deg.Start = LastStart;
        }

        for (i=0; i<NumReparsePointsDeg; i++)
        {
            ReparsePointsDeg[i].IsVirtualPoint = false;
            ReparsePointsDeg[i].Ref = ReparsePointsDeg[i].Deg;
        }
        for (i=0; i<NumReparsePointsRef; i++)
        {
            ReparsePointsRef[i].Deg = ReparsePointsRef[i].Ref;
        }

        OriginalNumReparsePointsDeg = NumReparsePointsDeg;
        for (i=0; i<NumReparsePointsDeg; i++)
            OriginalDegSectionLen[i] = ReparsePointsDeg[i].Deg.Len();

        //Assign ref and deg segments to each other
        ;

        {
            int LastPoint=-1;
            int LoopCount=0;
            int r=0;
            OTA_FLOAT Reliability, SectionCorrelation=0;
            int Delay;
            CFeatureVector* pVecRefLog = mpFeatureList->GetFVector(0, 0, 0, 1);
            CFeatureVector* pVecDegLog = mpFeatureList->GetFVector(0, 1, 0, 1);
            CFeatureVector* pVecRefLin = mpFeatureList->GetFVector(0, 0, 0, 0);
            CFeatureVector* pVecDegLin = mpFeatureList->GetFVector(0, 1, 0, 0);
            for (r=0; r<NumReparsePointsRef && NumReparsePointsDeg>0; r++)
            {

                ;

                //See if we find a nice match based on the VAD info of both signals. If
that is the case, just
                //use this allocation.
                REPARSE_POINT VAD1ReparsePoint;
```

```
                bool          VAD1SectionAccepted=false;
                if (1 && r<NumReparsePointsDeg)
                {
                    OTA_FLOAT     VAD1Reliability=-1;
                    int           VAD1Delay;
                    const int MaxLenDiffSamples=MSecondsToSamples(120);
                    int LengthRef = ReparsePointsRef[r].Ref.End -
ReparsePointsRef[r].Ref.Start;
                    int LengthDeg = ReparsePointsDeg[r].Deg.End -
ReparsePointsDeg[r].Deg.Start;
                    if (abs(LengthDeg-LengthRef)<MaxLenDiffSamples || SNRdB>35.0 &&
abs(LengthDeg-LengthRef)<4*MaxLenDiffSamples)
                    {
                        ;
                        VAD1ReparsePoint.Deg = ReparsePointsDeg[r].Deg;
                        VAD1ReparsePoint.Ref = ReparsePointsRef[r].Ref;
                        VAD1SectionAccepted = true;

                        int DelayOffset =
(VAD1ReparsePoint.Ref.Start-VAD1ReparsePoint.Deg.Start);
                        SECTION SecA = VAD1ReparsePoint.Ref;
                        SECTION SecB = VAD1ReparsePoint.Deg;
                        SecA.Start = SamplesToFrames(VAD1ReparsePoint.Ref.Start);
                        SecA.End = SamplesToFrames(VAD1ReparsePoint.Ref.End);
                        SecB.Start = SamplesToFrames(VAD1ReparsePoint.Deg.Start);
                        SecB.End= SamplesToFrames(VAD1ReparsePoint.Deg.End);

                        if (SecA.Len()>4)
                        {
                            int Unit = SamplesToFrames(MaxLenDiffSamples);
                            SecB.Start -= Unit;
                            SecB.End   += 2*Unit;
                            SecB.Start = (((0) > (SecB.Start)) ? (0) : (SecB.Start));

                            int Correction = 1;
                            if (SecA.End-SecA.Start>4*Unit+50)
                            {
                                SecA.Start += 2*Unit;
                                SecA.End   -= 2*Unit;
                                Correction *= 2;
                            }
                            VAD1Delay  = FindSectionAInSectionB(&SecA, &SecB,
pVecRefLog, pVecDegLog, &VAD1Reliability, 1, 1);
                            VAD1Delay -= Correction*Unit;

                        }
                        else
                        {
                            if (abs(SecA.Len()-SecB.Len())<1)
                            {
                                VAD1Delay = SecB.Start-SecA.Start;
                                VAD1Reliability = 0.95;
                            }
                            else
                            {
                                VAD1Delay = SecB.Start-SecA.Start;
                                VAD1Reliability = 0.7;
                            }
                        }

                        VAD1Delay = FramesToSamples(VAD1Delay);
                        VAD1Delay = -VAD1Delay+DelayOffset;
                        VAD1ReparsePoint.Ref.Start = VAD1ReparsePoint.Deg.Start +
VAD1Delay;
                        VAD1ReparsePoint.Ref.End = VAD1ReparsePoint.Deg.End +
VAD1Delay;
                        VAD1ReparsePoint.Reliability = VAD1Reliability;
                        ;
                    }
                }

                //Try finding the ref section in the deg section by using the ref VAD
info, correlation and the global delay
                //estimate as starting points. This works for most cases where the
delay between the first and
                //the second half of the sequence did not vary significantly.
```

```
                REPARSE_POINT Corr1ReparsePointDeg;
                bool          Corr1SectionAllocated=false;
                bool          Corr1SectionAccepted=false;
                int           Corr1Delay;
                OTA_FLOAT     Corr1Reliability=-1;
                Corr1ReparsePointDeg = AllocateSectionWithCorrelationBasedOnRefInfo(0,
0, r, ReparsePointsRef, NumReparsePointsRef, pVecRefLin,
ReparsePointsDeg, NumReparsePointsDeg, pVecDegLin, SNRdB, NoiseLevel,
&Corr1SectionAllocated, &Corr1SectionAccepted);
                Corr1Reliability = Corr1ReparsePointDeg.Reliability;
                Corr1Delay = Corr1ReparsePointDeg.DelayInSamples;

                if (Corr1Reliability<0.65 &&
ReparsePointsRef[r].Ref.Len()>MSecondsToSamples(500))

                {
                    REPARSE_POINT Corr11ReparsePointDeg;
                    bool          Corr11SectionAllocated=false;
                    bool          Corr11SectionAccepted=false;
                    int           Corr11Delay;
                    OTA_FLOAT     Corr11Reliability=-1;
                    int Offset = (int)(0.25*Corr1ReparsePointDeg.Ref.Len());
                    int Len = 0;

                    if (SNRdB<5)
                    {
                        int CenterPos=0;
                        double MaxEnergy=0;
                        const int RefLen =
SamplesToFrames(ReparsePointsRef[r].Ref.Len());
                        MaxEnergy =
matMaxExt(pVecRefLin->mpVector+SamplesToFrames(ReparsePointsRef
[r].Ref.Start), RefLen, &CenterPos);
                        Len = 0.25 * ReparsePointsRef[r].Ref.Len();
                        Offset = FramesToSamples((((0) > (CenterPos-Len/2)) ? (0) :
(CenterPos-Len/2)));
                    }

                    Corr11ReparsePointDeg =
AllocateSectionWithCorrelationBasedOnRefInfo(Offset, Len, r,
ReparsePointsRef, NumReparsePointsRef, pVecRefLin,
ReparsePointsDeg, NumReparsePointsDeg, pVecDegLin, SNRdB,
NoiseLevel, &Corr11SectionAllocated, &Corr11SectionAccepted);
                    Corr11Reliability = Corr11ReparsePointDeg.Reliability;
                    Corr11Delay = Corr11ReparsePointDeg.DelayInSamples;

                    if (Corr11Reliability>Corr1Reliability)
                    {
                        Corr1ReparsePointDeg = Corr11ReparsePointDeg;
                        Corr1Reliability = Corr11Reliability;
                        Corr1Delay = Corr11Delay;
                        Corr1SectionAllocated = Corr11SectionAllocated;
                        Corr1SectionAccepted = Corr11SectionAccepted;
                    }
                }

                //Try finding the ref section in the deg section by using the
correlation and the VAD info
                //as starting points. If this results in a better correlation for the
section, then take that.
                //This method is successful if there are large delay differences
between sections, but the
                //VAD worked reliably on the degraded signal.
                REPARSE_POINT Corr2ReparsePointDeg;
                OTA_FLOAT     Corr2Reliability=-1;
                int           Corr2Delay;
                bool          Corr2SectionAllocated=false;
                bool          Corr2SectionAccepted=false;
                Corr2ReparsePointDeg = AllocateSectionWithCorrelationBasedOnVADInfo(r,
ReparsePointsRef, NumReparsePointsRef, pVecRefLog, ReparsePointsDeg,
NumReparsePointsDeg, pVecDegLog, SNRdB, &Corr2SectionAllocated,
&Corr2SectionAccepted);
                Corr2Reliability = Corr2ReparsePointDeg.Reliability;
                Corr2Delay = Corr2ReparsePointDeg.DelayInSamples;

                bool SectionAllocated=false;
```

```
                {
                    int VersionToUse=-1;
                    if (Corr1SectionAllocated && Corr2SectionAllocated)
                    {
                        if (Corr1Reliability>=Corr2Reliability) VersionToUse = 1;
                        else                                    VersionToUse = 2;
                    }
                    else if (Corr1SectionAllocated)             VersionToUse = 1;
                    else if (Corr2SectionAllocated)             VersionToUse = 2;
                    else if (Corr1Reliability>=Corr2Reliability) VersionToUse = 1;
                    else                                         VersionToUse = 2;

                    switch(VersionToUse)
                    {
                        case 1:
                        {
                            ;

                            if(mProcessData.mpLogFile)
                            {
                                if(r < NumReparsePointsRef-1)
                                    fprintf(mProcessData.mpLogFile, "Reliability %.10f,
NumReparsePointsRef %d,
ReparsePointsDeg[r].Deg.Len() %d,
ReparsePointsRef[r].Ref.Len() %d,
ReparsePointsRef[r+1].Ref.Len() %d\n",
                                                                    Corr1Reliability,
NumReparsePointsRef,
ReparsePointsDeg[r].Deg
.Len(),
ReparsePointsRef[r].Ref
.Len(),
ReparsePointsRef[r+1].R
ef.Len());
                                else
                                    fprintf(mProcessData.mpLogFile, "Reliability %.10f,
NumReparsePointsRef %d,
ReparsePointsDeg[r].Deg.Len() %d,
ReparsePointsRef[r].Ref.Len() %d\n",
                                                                    Corr1Reliability,
NumReparsePointsRef,
ReparsePointsDeg[r].Deg
.Len(),
ReparsePointsRef[r].Ref
.Len());

                            }
                            if (r<NumReparsePointsRef-1 && Corr1Reliability > 0.85
                                && ReparsePointsDeg[r].Deg.Len() >
ReparsePointsRef[r].Ref.Len() +
ReparsePointsRef[r+1].Ref.Len())
                            {
                                ;
                                DuplicateReparsePoint(r, ReparsePointsDeg,
&NumReparsePointsDeg);
                                ReparsePointsDeg[r].Deg.End =
ReparsePointsDeg[r].Deg.Start +
ReparsePointsRef[r].Ref.Len();
                                ReparsePointsDeg[r+1].Deg.Start=ReparsePointsDeg[r].Deg
.End + ReparsePointsRef[r+1].Ref.Start -
ReparsePointsRef[r].Ref.End;
                            }

                            int VLen1 = ReparsePointsDeg[r].Deg.Len() +
ReparsePointsDeg[r+1].Deg.Len();
                            int VLen2 = ReparsePointsRef[r].Ref.Len();

                            if (r<NumReparsePointsDeg-1 &&
                                ReparsePointsRef[r].Ref.Len() >
ReparsePointsDeg[r].Deg.Len() +
ReparsePointsDeg[r+1].Deg.Len() &&
(ReparsePointsDeg[r+1].Deg.Start-ReparsePointsDeg[r].De
g.End)<MSecondsToSamples(500))

                            {
                                ;
```

```
                                DuplicateReparsePoint(r, ReparsePointsRef,
&NumReparsePointsRef);
                                ReparsePointsRef[r].Ref.End =
ReparsePointsRef[r].Ref.Start +
ReparsePointsDeg[r].Deg.Len();
                                ReparsePointsRef[r+1].Ref.Start=ReparsePointsRef[r].Ref
.End + 1;
                                Corr1ReparsePointDeg.Deg.End =
Corr1ReparsePointDeg.Deg.Start +
ReparsePointsDeg[r].Deg.Len();
                                Corr1ReparsePointDeg.Ref.End =
Corr1ReparsePointDeg.Ref.Start +
ReparsePointsDeg[r].Ref.Len();
                            }

                            Reliability = Corr1Reliability;
                            Delay = SamplesToFrames(Corr1Delay);
                            ReparsePointsDeg[r] = Corr1ReparsePointDeg;

                            if (r>=NumReparsePointsDeg)
                            {
                                NumReparsePointsDeg++;
                            }
                            if (Corr1SectionAccepted)
                            {
                                SectionAllocated = true;

                                ;
                            }

                            break;
                        }
                        case 2:
                        {
                            ;

                            int DegLen = ReparsePointsDeg[r].Deg.Len();
                            int NewLen = Corr2ReparsePointDeg.Ref.Len();
                            if (DegLen-NewLen > MSecondsToSamples(500))
                            {

                                if (ReparsePointsDeg[r].Deg.Len() >
ReparsePointsRef[r].Ref.Len() +
ReparsePointsRef[r+1].Ref.Len())

                                {
                                    ;
                                    DuplicateReparsePoint(r, ReparsePointsDeg,
&NumReparsePointsDeg);

                                    ReparsePointsDeg[r+1].Deg.Start =
Corr2ReparsePointDeg.Deg.End + 1;

                                }
                            }
                            Reliability = Corr2Reliability;
                            Delay = SamplesToFrames(Corr2Delay);
                            ReparsePointsDeg[r] = Corr2ReparsePointDeg;
                            if (Corr2SectionAccepted)
                            {
                                SectionAllocated = true;
                                ;
                            }

                            if (r>=NumReparsePointsDeg)
                            {
                                NumReparsePointsDeg++;
                            }

                            break;
                        }
                        default:
                        {
                            ;
                            Reliability = 0;
                            Delay = 0;
```

```
                            SectionAllocated = false;
                        }
                    }
                }

                //If all of the above failed we need to guess the alignment from the
VAD info only.
                //This is just a last resort!
                //Since this may destroy some of the real reparse points and a
different allocation method might be chosen later,
                //we must operate on a copy of the data.

                REPARSE_POINT ReparsePointVAD;
                REPARSE_POINT CopyOfReparsePointsDeg[100];
                int CopyOfNumReparsePointsDeg=NumReparsePointsDeg;
                bool FoundVADMatch=false;

                for (i=0; i<NumReparsePointsDeg; i++)
                    CopyOfReparsePointsDeg[i] = ReparsePointsDeg[i];

                if (!SectionAllocated)
                {
                    ;
                    int Res;
                    int LoopCount=0;
                    do
                    {
                        Res = FindMatchingSection(ReparsePointsRef,
NumReparsePointsRef, CopyOfReparsePointsDeg,
CopyOfNumReparsePointsDeg, SNRdB, r, &NumReparsePointsRef,
&CopyOfNumReparsePointsDeg);
                        if (Res==1)
                        {
                            ReparsePointVAD = CopyOfReparsePointsDeg[r];
                            ReparsePointVAD.Ref = ReparsePointsRef[r].Ref;
                            ReparsePointVAD.Reliability = Reliability=0;

                            int RefStart = SamplesToFrames(ReparsePointVAD.Ref.Start);
                            int DegStart = SamplesToFrames(ReparsePointVAD.Deg.Start);
                            int DegEnd = SamplesToFrames(ReparsePointVAD.Deg.End);
                            int Len = (((DegEnd-DegStart) <
(pVecRefLog->mSize-RefStart)) ? (DegEnd-DegStart) :
(pVecRefLog->mSize-RefStart));
                            Len = (((Len) < (pVecDegLog->mSize-DegStart)) ? (Len) :
(pVecDegLog->mSize-DegStart));
                            if (Len>0)
                                ReparsePointVAD.Reliability =
matPearsonCorrelation(pVecDegLog->mpVector+DegStart,
pVecRefLog->mpVector+RefStart, Len);
                            else ReparsePointVAD.Reliability = -1;
                            FoundVADMatch = true;
                        }
                        else ReparsePointVAD.Reliability = -1;

                        LoopCount++;
                        if (LoopCount>20)
                        {
                            ;
                            exit(1);
                        }
                    } while (Res<0 && r<NumReparsePointsRef);

                    ;

                }

                if (!SectionAllocated)
                {
                    bool UseOverallDelay =
(OverallDelayEstimateReliability>ReparsePointsDeg[r].Reliability &&
ReparsePointsDeg[r].Reliability>ReparsePointVAD.Reliability) ? true
: false;

                    ;
                    ;
                    ;
```

```
                    bool UseVADMatchDelay = FoundVADMatch && !UseOverallDelay &&
ReparsePointVAD.Reliability>ReparsePointsDeg[r].Reliability ? true
: false;

                    bool EmergencySolution = false;
                    if (1 && ReparsePointsDeg[r].Reliability < 0.2 &&
ReparsePointVAD.Reliability < 0.2 &&
OverallDelayEstimateReliability < 0.2)
                    {
                        int ReparseSectionLiesinHalf = -1;
                        OTA_FLOAT OverallDelayEstimateReliabilityThisHalf;
                        int OverallDelayEstimateThisHalf;

                        if (ReparsePointsDeg[r].Deg.End <=
mppSignals[1]->mSignalLength/2)
                        {
                            ReparseSectionLiesinHalf = 0;
                            OverallDelayEstimateReliabilityThisHalf =
OverallDelayEstimateReliability1st;
                            OverallDelayEstimateThisHalf = OverallDelayEstimate1st;
                        }
                        else if(ReparsePointsDeg[r].Deg.Start >
mppSignals[1]->mSignalLength/2)
                        {
                            ReparseSectionLiesinHalf = 1;
                            OverallDelayEstimateReliabilityThisHalf =
OverallDelayEstimateReliability2nd;
                            OverallDelayEstimateThisHalf = OverallDelayEstimate2nd;
                        }

                        if(ReparseSectionLiesinHalf != -1 &&
OverallDelayEstimateReliabilityThisHalf > 0.5)
                        {
                            ;
                            ReparsePointsDeg[r].Deg.Start =
ReparsePointsRef[r].Ref.Start -
OverallDelayEstimateThisHalf;
                            ReparsePointsDeg[r].Deg.End   = ReparsePointsRef[r].Ref.End
- OverallDelayEstimateThisHalf;
                            ReparsePointsDeg[r].Ref = ReparsePointsRef[r].Ref;
                            ReparsePointsDeg[r].Reliability =
OverallDelayEstimateReliabilityThisHalf;
                            if (r>=NumReparsePointsDeg) NumReparsePointsDeg++;
                            SectionAllocated = true;
                            UseOverallDelay = false;
                            UseVADMatchDelay = false;
                            EmergencySolution = true;
                        }
                    }

                    //If the section was not allocated so far, choose the best possible
approach now.
                    //We have three possible delays now:
                    //1. The overalDelayEstimate
                    //2. The delay from the section correlation (this is already used
by default)
                    //3. The delay from VAD matching

                    if(!EmergencySolution)
                    {
                        if (UseOverallDelay)
                        {
                            ;
                            ReparsePointsDeg[r].Deg.Start =
ReparsePointsRef[r].Ref.Start-OverallDelayEstimate;
                            ReparsePointsDeg[r].Deg.End   =
ReparsePointsRef[r].Ref.End-OverallDelayEstimate;
                            ReparsePointsDeg[r].Ref = ReparsePointsRef[r].Ref;
                            ReparsePointsDeg[r].Reliability =
OverallDelayEstimateReliability;
                            if (r>=NumReparsePointsDeg) NumReparsePointsDeg++;
                            SectionAllocated = true;
                        }

                        if (UseVADMatchDelay)
                        {
```

```
                            ;

                            for (i=0; i<CopyOfNumReparsePointsDeg; i++)
                                ReparsePointsDeg[i] = CopyOfReparsePointsDeg[i];
                            NumReparsePointsDeg = CopyOfNumReparsePointsDeg;

                            ReparsePointsDeg[r].Deg = ReparsePointVAD.Deg;
                            ReparsePointsDeg[r].Ref = ReparsePointVAD.Ref;
                            ReparsePointsDeg[r].Reliability =
ReparsePointVAD.Reliability;
                            if (r>=NumReparsePointsDeg) NumReparsePointsDeg++;
                            SectionAllocated = true;
                        }

                        if (!UseOverallDelay && !UseVADMatchDelay)
                        {
                            ;
                            if (r>=NumReparsePointsDeg) NumReparsePointsDeg++;
                            SectionAllocated = true;
                        }
                    }
                }

                if (ReparsePointsDeg[r].Reliability<0.8 && VAD1SectionAccepted)
                {
                    ;
                    for (i=0; i<CopyOfNumReparsePointsDeg; i++)
                        ReparsePointsDeg[i] = CopyOfReparsePointsDeg[i];
                    NumReparsePointsDeg = CopyOfNumReparsePointsDeg;
                    ReparsePointsDeg[r] = VAD1ReparsePoint;
                    if (r>=NumReparsePointsDeg) NumReparsePointsDeg++;
                    SectionAllocated = true;
                }

                if (SectionAllocated)
                {
                    for (i=r; i<NumReparsePointsDeg-1; i++)
                    {
                        if
(ReparsePointsDeg[i].Deg.End>=ReparsePointsDeg[i+1].Deg.Start)
                        {
                            ;
                            DeleteReparsePoint(i+1, ReparsePointsDeg,
&NumReparsePointsDeg);
                            i--;
                        }
                    }
                }

            }
        }

        //Clean up found sections.

        if (NumReparsePointsDeg<=0)
        {
            ;
            for (i=0; i<NumReparsePointsRef; i++)
            {
                ReparsePointsDeg[i].Deg = ReparsePointsRef[i].Ref;
                ReparsePointsDeg[i].Ref = ReparsePointsRef[i].Ref;
            }
        }

        //Check for proper limits etc.
        //In theory the following checks should not be required...
        CheckSectionLimits(ReparsePointsDeg, NumReparsePointsRef,
&NumReparsePointsDeg);
        if (NumReparsePointsRef<=0)
        {
            ;
            ReparsePointsDeg[0].Ref.Start = ReparsePointsDeg[0].Deg.Start = 0;
            ReparsePointsDeg[0].Ref.End   = ReparsePointsDeg[0].Deg.End   =
FramesToSamples((((mpFeatureList->GetFVector(0, 0, 0, 0)->mSize) <
(mpFeatureList->GetFVector(0, 1, 0, 0)->mSize)) ?
(mpFeatureList->GetFVector(0, 0, 0, 0)->mSize) :
```

```
(mpFeatureList->GetFVector(0, 1, 0, 0)->mSize)));
            NumReparsePointsRef = NumReparsePointsDeg = 1;
        }
        else
        {

            bool LowSectionCorrelation=true;
            for (i=0; i<NumReparsePointsRef; i++)
                if (ReparsePointsDeg[i].Reliability>0.57)
                    LowSectionCorrelation = false;

            bool BadLengthRatio=true;
            int TotalOriginalLen=0;
            int TotalFinalLen=0;
            for (i=0; i<OriginalNumReparsePointsDeg; i++)
            {
                TotalOriginalLen += OriginalDegSectionLen[i];
                TotalFinalLen    += ReparsePointsDeg[i].Deg.Len();
            }
            OTA_FLOAT Ratio = (OTA_FLOAT)TotalFinalLen /
(OTA_FLOAT)TotalOriginalLen+0.1;
            if (Ratio<4 && Ratio>0.25)
                BadLengthRatio = false;

            bool BadSectionRatio=true;
            OTA_FLOAT SectionRatio =
(OTA_FLOAT)NumReparsePointsDeg/(OTA_FLOAT)MaxParsePoints;
            if (SectionRatio<2 && SectionRatio>0.5)
                BadSectionRatio = false;

            if (LowSectionCorrelation && BadLengthRatio)
            {
                ;

                if (OverallDelayEstimateReliability>0)
                    AllocateSectionsFromDelayEstimate(ReparsePointsDeg,
ReparsePointsRef, NumReparsePointsRef, &NumReparsePointsDeg,
OverallDelayEstimate, OverallDelayEstimateReliability);
                else
                    AllocateSectionsFromDelayEstimate(ReparsePointsDeg,
ReparsePointsRef, NumReparsePointsRef, &NumReparsePointsDeg, 0, 0);

                CheckSectionLimits(ReparsePointsDeg, NumReparsePointsRef,
&NumReparsePointsDeg);
            }
        }

        NumReparsePointsDeg = NumReparsePointsRef;

        Done = true;
    }

    for (int r=0; r<NumReparsePointsDeg; r++)
        ReparsePointsDeg[r].DelayInSamples = ReparsePointsDeg[r].Ref.Start -
ReparsePointsDeg[r].Deg.Start;

    return NumReparsePointsDeg;
}

#pragma endregion

#pragma region Prealignment
//Fill the initial delay vector with the average delay at the current feature frame
resolution,
//but only one for each macro frame. Delay changes are placed in the middle between two
reparse points.
void CTempAlignment::ReparseSections2DelayVector(OTA_FLOAT* ReliabilityPerFrame, int*
pSearchRangePerMacroFrameLow, int* pSearchRangePerMacroFrameHigh, int DelayResolution)
{
    int r, NextFrame;
    CProcessData IterationData = mProcessData;
    IterationData.Init(1, 1.0);
    int ReparsePointStepSize = IterationData.mStepSize;

    for (r=0; r<mNumReparsePoints-1; r++)
    {
```

```
        int StartMacroFrame = (mpReparsePoints[r].Deg.Start+ReparsePointStepSize/2) /
ReparsePointStepSize;

        int LastMacroFrame  = (mpReparsePoints[r].Deg.End+ReparsePointStepSize/2) /
ReparsePointStepSize;

        int StartOfNextMacroFrame =
(mpReparsePoints[r+1].Deg.Start+ReparsePointStepSize/2) / ReparsePointStepSize;

        int DelayDiff = mpReparsePoints[r+1].DelayInSamples -
mpReparsePoints[r].DelayInSamples;
        int NewDelayFrame=0;
        NewDelayFrame = ((mpReparsePoints[r+1].Deg.Start - mpReparsePoints[r].Deg.End +
DelayDiff)/2 + mpReparsePoints[r].Deg.End) / ReparsePointStepSize;
        NewDelayFrame = (((NewDelayFrame) > (mpReparsePoints[r].Deg.End /
ReparsePointStepSize)) ? (NewDelayFrame) : (mpReparsePoints[r].Deg.End /
ReparsePointStepSize));

        if (r==0)
        {
            int Delay = mpReparsePoints[0].DelayInSamples;
            OTA_FLOAT Reliability = 0;
            for (NextFrame=0; NextFrame<StartMacroFrame && NextFrame<mNumMacroFrames;
NextFrame++)
            {
                mpDelayInSamplesPerFrame[NextFrame] = Delay;
                mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] =
Reliability;
                pSearchRangePerMacroFrameHigh[NextFrame] =
mpReparsePoints[0].MaxDelayVarInSamples;
                pSearchRangePerMacroFrameLow[NextFrame] =
mpReparsePoints[0].MinDelayVarInSamples;
            }
        }
        int Delay = mpReparsePoints[r].DelayInSamples;
        OTA_FLOAT Reliability = mpReparsePoints[r].Reliability;
        for (NextFrame=StartMacroFrame; NextFrame<LastMacroFrame &&
NextFrame<mNumMacroFrames; NextFrame++)
        {
            mpDelayInSamplesPerFrame[NextFrame] = Delay;
            mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] =
Reliability;
            pSearchRangePerMacroFrameHigh[NextFrame] =
mpReparsePoints[r].MaxDelayVarInSamples;
            pSearchRangePerMacroFrameLow[NextFrame] =
mpReparsePoints[r].MinDelayVarInSamples;
        }

        if (mpReparsePoints[r].Reliability >
mProcessData.mP.mTAPara.mCorrForSkippingInitialDelaySearch)
        {
            for (int j=StartMacroFrame; j<StartMacroFrame+2 && j<LastMacroFrame &&
j<mNumMacroFrames; j++)
            {
                pSearchRangePerMacroFrameHigh[j] =  4*DelayResolution;
                pSearchRangePerMacroFrameLow[j]  = -4*DelayResolution;
            }
        }

        Reliability = mpReparsePoints[r].Reliability;
        for (; NextFrame<NewDelayFrame && NextFrame<mNumMacroFrames; NextFrame++)
        {
            mpDelayInSamplesPerFrame[NextFrame] = Delay;
            mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] =
Reliability;
            pSearchRangePerMacroFrameHigh[NextFrame] =
mpReparsePoints[r].MaxDelayVarInSamples;
            pSearchRangePerMacroFrameLow[NextFrame] =
mpReparsePoints[r].MinDelayVarInSamples;
        }

        Delay = mpReparsePoints[r+1].DelayInSamples;
        Reliability = mpReparsePoints[r+1].Reliability;
        for (; NextFrame<StartOfNextMacroFrame && NextFrame<mNumMacroFrames;
NextFrame++)
        {
```

```
            mpDelayInSamplesPerFrame[NextFrame] = Delay;
            mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] =
Reliability;
            pSearchRangePerMacroFrameHigh[NextFrame] =
mpReparsePoints[r+1].MaxDelayVarInSamples;
            pSearchRangePerMacroFrameLow[NextFrame] =
mpReparsePoints[r+1].MinDelayVarInSamples;
        }
    }

    int LastStartFrame = mNumReparsePoints>1 ?
mpReparsePoints[mNumReparsePoints-1].Deg.Start / ReparsePointStepSize : 0;
    LastStartFrame = (((mNumMacroFrames) < ((((0) > (LastStartFrame)) ? (0) :
(LastStartFrame)))) ? (mNumMacroFrames) : ((((0) > (LastStartFrame)) ? (0) :
(LastStartFrame))));
    for (NextFrame=LastStartFrame; NextFrame<mNumMacroFrames; NextFrame++)
    {
        mpDelayInSamplesPerFrame[NextFrame] =
mpReparsePoints[mNumReparsePoints-1].DelayInSamples;
        mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] =
mpReparsePoints[mNumReparsePoints-1].Reliability;
        pSearchRangePerMacroFrameHigh[NextFrame] =
mpReparsePoints[mNumReparsePoints-1].MaxDelayVarInSamples;
        pSearchRangePerMacroFrameLow[NextFrame] =
mpReparsePoints[mNumReparsePoints-1].MinDelayVarInSamples;
    }
}

//Get a list of the active frames and make sure that all sections between reparse
points are marked as inactive.
//This list is operating at the macro frame rate. MARGIN additional frames on either
side of active segments are
//set to inactive in order to avoid delay decissions based on frames which may show
some border effects.

void CTempAlignment::SetActiveFrameFlags(int Stepsize, int* pActiveFrameFlags)
{
    int i, r;
    mpActiveFrameDetection->GetActiveFrameFlags(1, 0, Stepsize, pActiveFrameFlags,
mNumMacroFrames);
    for (i=0; i<mNumMacroFrames && i<mpReparsePoints[0].Deg.Start/Stepsize+0; i++)
        pActiveFrameFlags[i] = 0;
    for (r=1; r<mNumReparsePoints; r++)
        for (i=(((0) > (mpReparsePoints[r-1].Deg.End/Stepsize-0)) ? (0) :
(mpReparsePoints[r-1].Deg.End/Stepsize-0)); i<mNumMacroFrames &&
i<mpReparsePoints[r].Deg.Start/Stepsize+0; i++)
            pActiveFrameFlags[i] = 0;
    for (i=(((0) > (mpReparsePoints[mNumReparsePoints-1].Deg.End/Stepsize-0)) ? (0) :
(mpReparsePoints[mNumReparsePoints-1].Deg.End/Stepsize-0)); i<mNumMacroFrames; i++)
        pActiveFrameFlags[i] = 0;

    int SumFlags = matSum(pActiveFrameFlags, mNumMacroFrames);
    if (SumFlags==0)
        for (r=1; r<mNumReparsePoints; r++)
            matbSet(1, pActiveFrameFlags+mpReparsePoints[r].Deg.Start/Stepsize,
(mpReparsePoints[r].Deg.End-mpReparsePoints[r].Deg.Start)/Stepsize);

}

//Do the fast prealignment method. This replaces:
//- Overall delay estimation
//- Reparse point identification
//- Initial delay search
//In addition this drasticall limits the searchrange required for the coarse alignment
//and thus speeds up processing dramatically!
bool CTempAlignment::SectionsFromSQPrealignment(void **pPAHandle)
{
    bool rc = true;
    int i, j, r;

    assert(((CAudioSignal*)mppSignals[0])->mSampleRate ==
((CAudioSignal*)mppSignals[1])->mSampleRate);
    assert(((CAudioSignal*)mppSignals[0])->mNumChannels == 1 &&
           ((CAudioSignal*)mppSignals[0])->mNumChannels == 1);

    int             pa_sampleRate = (int) ((CAudioSignal*)mppSignals[0])->mSampleRate;
```

```
    TA_SegList const *pa_segList   = NULL;

    *pPAHandle = PreAlignment_Init(
        mppSignals[0]->GetDataVector(0), mppSignals[0]->mSignalLength,
        mppSignals[1]->GetDataVector(0), mppSignals[1]->mSignalLength,
        pa_sampleRate, 2, mProcessData.mpLogFile, mProcessData.mpMathlibHandle);
    if (*pPAHandle == NULL)
        return false;

    pa_segList = PreAlignment_GetSegList(*pPAHandle);
    if (pa_segList == NULL)
    {
        PreAlignment_Free(pPAHandle);
        return false;
    }

    OTA_FLOAT maxPauseLen = (OTA_FLOAT)0.1 /
sqrt(((((((PreAlignment_GetDegSNR(*pPAHandle) / (OTA_FLOAT)40.0) >
((OTA_FLOAT)0.25)) ? (PreAlignment_GetDegSNR(*pPAHandle) / (OTA_FLOAT)40.0) :
((OTA_FLOAT)0.25))) < ((OTA_FLOAT)1.0)) ? ((((PreAlignment_GetDegSNR(*pPAHandle) /
(OTA_FLOAT)40.0) > ((OTA_FLOAT)0.25)) ? (PreAlignment_GetDegSNR(*pPAHandle) /
(OTA_FLOAT)40.0) : ((OTA_FLOAT)0.25))) : ((OTA_FLOAT)1.0)));

    r = 0;
    bool  inSpeechSeg    = false;
    int   consecLen=0, consecPauseLen, consecLenMatched;
    XFLOAT avgWeightedDelay=0.0, avgWeightedRlblt=0.0;
    int   IndexOfLongestSegment = -1;
    for (i = 0; i < (int)pa_segList->size() && r < 100; i++)
    {
        if (!inSpeechSeg && pa_segList->at(i).segType == TA_SEG_PAUSE)
            continue;

        if (inSpeechSeg && pa_segList->at(i).segType == TA_SEG_PAUSE)
        {
            for (j = i, consecPauseLen = 0;
                 j < (int)pa_segList->size() && pa_segList->at(j).segType ==
TA_SEG_PAUSE;
                 consecPauseLen += pa_segList->at(j).segLen, j++);
            if (j < (int)pa_segList->size() && consecPauseLen <
(int)(pa_sampleRate*maxPauseLen + (OTA_FLOAT)0.5))
            {
                i = j-1;
                continue;
            }
        }

        if (pa_segList->at(i).segType == TA_SEG_PAUSE)
        {

            if (pa_segList->at(IndexOfLongestSegment).segType == TA_SEG_MATCHED &&
pa_segList->at(IndexOfLongestSegment).segLen > (consecLen>>1))
            {
                mpReparsePoints[r].DelayInSamples =
pa_segList->at(IndexOfLongestSegment).refPos -
pa_segList->at(IndexOfLongestSegment).degPos;
                mpReparsePoints[r].Reliability    =
pa_segList->at(IndexOfLongestSegment).reliability;
            }
            else
            {
                mpReparsePoints[r].DelayInSamples = RINT(avgWeightedDelay / consecLen);
                mpReparsePoints[r].Reliability    = avgWeightedRlblt / consecLen;
            }

            mpReparsePoints[r].MinDelayVarInSamples =
-mpReparsePoints[r].DelayInSamples +
mpReparsePoints[r].MinDelayVarInSamples;
            mpReparsePoints[r].MaxDelayVarInSamples =
mpReparsePoints[r].MaxDelayVarInSamples -
mpReparsePoints[r].DelayInSamples;

            mpReparsePoints[r].Deg.End = pa_segList->at(i).degPos - 1;
            mpReparsePoints[r].Ref.End = pa_segList->at(i).refPos - 1;

            r++;
```

```
            inSpeechSeg = false;
        }
        else
        {
            if (!inSpeechSeg)
            {
                mpReparsePoints[r].Ref.Start = pa_segList->at(i).refPos;
                mpReparsePoints[r].Deg.Start = pa_segList->at(i).degPos;
                mpReparsePoints[r].DelayInSamples = pa_segList->at(i).refPos -
pa_segList->at(i).degPos;
                mpReparsePoints[r].MinDelayVarInSamples =
mpReparsePoints[r].DelayInSamples;
                mpReparsePoints[r].MaxDelayVarInSamples =
mpReparsePoints[r].DelayInSamples;
                consecLenMatched = consecLen = 0;
                avgWeightedDelay = avgWeightedRlblt = 0.0f;
                IndexOfLongestSegment = i;
                inSpeechSeg = true;
            }
            else if (pa_segList->at(i).segType != TA_SEG_MISSING)
            {
                int NewDelayStart = pa_segList->at(i).refPos -
pa_segList->at(i).degPos;
                int NewDelayEnd   = NewDelayStart;
                mpReparsePoints[r].MinDelayVarInSamples =
(((mpReparsePoints[r].MinDelayVarInSamples) < ((((NewDelayStart) <
(NewDelayEnd)) ? (NewDelayStart) : (NewDelayEnd)))) ?
(mpReparsePoints[r].MinDelayVarInSamples) : ((((NewDelayStart) <
(NewDelayEnd)) ? (NewDelayStart) : (NewDelayEnd))));
                mpReparsePoints[r].MaxDelayVarInSamples =
(((mpReparsePoints[r].MaxDelayVarInSamples) > ((((NewDelayStart) >
(NewDelayEnd)) ? (NewDelayStart) : (NewDelayEnd)))) ?
(mpReparsePoints[r].MaxDelayVarInSamples) : ((((NewDelayStart) >
(NewDelayEnd)) ? (NewDelayStart) : (NewDelayEnd))));
            }

            if (pa_segList->at(i).segType != TA_SEG_MISSING)
            {
                consecLen        += pa_segList->at(i).segLen;
                avgWeightedDelay += pa_segList->at(i).segLen *
(pa_segList->at(i).refPos - pa_segList->at(i).degPos);
                avgWeightedRlblt += pa_segList->at(i).segLen *
pa_segList->at(i).reliability;
                if (pa_segList->at(i).segLen >
pa_segList->at(IndexOfLongestSegment).segLen)
                    IndexOfLongestSegment = i;
            }
        }
    }

    if (inSpeechSeg)
    {
        for (j = i-1; j >= 0 && pa_segList->at(j).segType == TA_SEG_MISSING; j--);

        if (pa_segList->at(IndexOfLongestSegment).segType == TA_SEG_MATCHED &&
pa_segList->at(IndexOfLongestSegment).segLen > (consecLen>>1))
        {
            mpReparsePoints[r].DelayInSamples =
pa_segList->at(IndexOfLongestSegment).refPos -
pa_segList->at(IndexOfLongestSegment).degPos;
            mpReparsePoints[r].Reliability    =
pa_segList->at(IndexOfLongestSegment).reliability;
        }
        else
        {
            mpReparsePoints[r].DelayInSamples = RINT(avgWeightedDelay / consecLen);
            mpReparsePoints[r].Reliability    = avgWeightedRlblt / consecLen;
        }

        mpReparsePoints[r].MinDelayVarInSamples = -mpReparsePoints[r].DelayInSamples +
mpReparsePoints[r].MinDelayVarInSamples;
        mpReparsePoints[r].MaxDelayVarInSamples =
mpReparsePoints[r].MaxDelayVarInSamples - mpReparsePoints[r].DelayInSamples;

        mpReparsePoints[r].Deg.End = pa_segList->at(j).degPos +
pa_segList->at(j).segLen - 1;
```

```
        mpReparsePoints[r].Deg.End = (((mppSignals[1]->mSignalLength-1) <
(mpReparsePoints[r].Deg.End)) ? (mppSignals[1]->mSignalLength-1) :
(mpReparsePoints[r].Deg.End));
        mpReparsePoints[r].Ref.End = pa_segList->at(i-1).refPos +
pa_segList->at(i-1).segLen - 1;

        r++;
    }

    if (r <= 0)
    {
        PreAlignment_Free(pPAHandle);
        return false;
    }
    else
    {

        mNumReparsePoints = r;
    }

    return rc;
}

bool CTempAlignment::ResetSectionData()
{
    bool rc = true;

    mNumReparsePoints = 1;
    for (int i = 0; i < 100; i++)
    {
        mpReparsePoints[i].DelayInSamples = 0;
        mpReparsePoints[i].Deg.Start = 0;
        mpReparsePoints[i].Ref.Start = 0;
        mpReparsePoints[i].Deg.End = mppSignals[1]->mSignalLength;
        mpReparsePoints[i].Ref.End = mppSignals[0]->mSignalLength;
    }

    return rc;
}

//Derive the initial delay vector etc. directly from the segment lists created by
SectionsFromSQPrealignment().
//this is an alternative to ReparseSections2DelayVector(), but only available when
SectionsFromSQPrealignment()
//was called before!
bool CTempAlignment::SegmentList2DelayVector(void** pPAHandle, void** pPA_vec, float
DelayVecStepsize, int* pActiveFrameFlags, OTA_FLOAT* ReliabilityPerFrame, int*
pSearchRangePerMacroFrameLow, int* pSearchRangePerMacroFrameHigh)
{
    bool rc = true;

    int i, j, k, f;

    if (pPAHandle == NULL || pPA_vec == NULL)
        return false;

    TraversalVecType const *pa_vec = (TraversalVecType const*)(*pPA_vec);
    if (pa_vec == NULL)
    {
        PreAlignment_Free(pPAHandle);
        return false;
    }

    i = f = 0;
    for (; i < (int)pa_vec->size() && pa_vec->at(i).type == MISSING_SPEECH; i++);
    if (i >= (int)pa_vec->size())
    {
        PreAlignment_Free(pPAHandle);
        *pPA_vec = NULL;
        return false;
    }

    int pa_sampleRate = (int) ((CAudioSignal*)mppSignals[0])->mSampleRate;
    int pa_delayInSamples, curDegPos = 0, firstSpeechPos, lastSpeechPos;
    firstSpeechPos = pa_vec->at(i).refPos;
    for (j = 0; j <= (int)pa_vec->size()-1 && (pa_vec->at(j).type == PAUSE ||
```

```
pa_vec->at(j).type == INSERTED_SIG); j++)
        firstSpeechPos = pa_vec->at(j).refPos;
    lastSpeechPos  = pa_vec->back().refPos;
    for (j = (int)pa_vec->size()-1; j >= 0 && (pa_vec->at(j).type == PAUSE ||
pa_vec->at(j).type == INSERTED_SIG); j--)
        lastSpeechPos  = pa_vec->at(j).refPos;

    //Add frames for extra leading silence in deg for which there is no info in pa_vec.
    if (pa_vec->at(i).degPos != 0)
    {
        pa_delayInSamples = pa_vec->at(i).refPos - pa_vec->at(i).degPos;
        while (f < mNumMacroFrames && curDegPos < pa_vec->at(i).degPos)
        {
            pActiveFrameFlags  [f] = 0;
            ReliabilityPerFrame[f] = (OTA_FLOAT)0;
            pSearchRangePerMacroFrameLow [f] = 0;
            pSearchRangePerMacroFrameHigh[f] = firstSpeechPos -
(curDegPos+pa_delayInSamples);
            mpDelayInSamplesPerFrame[f] = pa_delayInSamples;
            curDegPos += mMacroFrameSize;
            f++;
        }
    }

    int firstPauseRefPos = -1, lastPauseRefPos = -1, insertedSigAct = -1,
missingRefStart = -1, missingRefEnd = -1,
        minConsecMinPos = -1, maxConsecMaxPos = -1;
    for (; f < mNumMacroFrames && i < (int)pa_vec->size(); i++)
    {
        if (pa_vec->at(i).type == MISSING_SPEECH)
        {
            for (j = i; j < (int)pa_vec->size() && pa_vec->at(j).type ==
MISSING_SPEECH; j++);
            j--;
            missingRefStart = pa_vec->at(i).refPos;
            missingRefEnd   = pa_vec->at(j).refPos + mMacroFrameSize;

            if (f > 0 && pActiveFrameFlags[f-1] == 1 && i > 0 &&
                (pa_vec->at(i-1).type == SEARCHABLE_SPEECH || pa_vec->at(i-1).type ==
FIXED_SPEECH) &&
                pa_vec->at(i-1).maxPos + mMacroFrameSize < missingRefEnd)
            {
                pSearchRangePerMacroFrameHigh[f-1] =
(((pSearchRangePerMacroFrameHigh[f-1]) >
((missingRefEnd-mMacroFrameSize) - pa_vec->at(i-1).refPos)) ?
(pSearchRangePerMacroFrameHigh[f-1]) : ((missingRefEnd-mMacroFrameSize)
- pa_vec->at(i-1).refPos));
                ReliabilityPerFrame          [f-1] = (((ReliabilityPerFrame[f-1]) <
(0.75f)) ? (ReliabilityPerFrame[f-1]) : (0.75f));
            }

            firstPauseRefPos = lastPauseRefPos = -1;
            i = j;
            continue;
        }

        ReliabilityPerFrame[f] = pa_vec->at(i).reliability;
        pActiveFrameFlags  [f] = pa_vec->at(i).degActivity && pa_vec->at(i).type !=
PAUSE ? 1 : 0;

        if (pa_vec->at(i).type == INSERTED_SIG)
        {
            if (insertedSigAct < 0)
            {
                minConsecMinPos = lastSpeechPos, maxConsecMaxPos = firstSpeechPos;
                for (j = i; j > 0                    && pa_vec->at(j).type ==
INSERTED_SIG; j--)
                    minConsecMinPos = (((minConsecMinPos) < (pa_vec->at(j).minPos)) ?
(minConsecMinPos) : (pa_vec->at(j).minPos));
                for (k = i; k < (int)pa_vec->size()-1 && pa_vec->at(k).type ==
INSERTED_SIG; k++)
                    maxConsecMaxPos = (((maxConsecMaxPos) > (pa_vec->at(k).maxPos)) ?
(maxConsecMaxPos) : (pa_vec->at(k).maxPos));

                if (pa_vec->at(j).type != PAUSE && pa_vec->at(k).type != PAUSE)
                {
```

```
                        pActiveFrameFlags[f] = insertedSigAct = 1;
                }
                else
                    pActiveFrameFlags[f] = insertedSigAct = 0;
            }
            else
                pActiveFrameFlags[f] = insertedSigAct;

            if (pa_vec->at(i).maxPos < firstSpeechPos || pa_vec->at(i).minPos >
lastSpeechPos)
                pActiveFrameFlags[f] = 0;
        }
        else
            insertedSigAct = -1;

        //Set activity to 1 at utterance boundaries after a pause, as the delay may
have changed.
        if (pa_vec->at(i).type == SEARCHABLE_SPEECH && pActiveFrameFlags[f] == 0)
        {
            for (j = i; j > 0                       && pa_vec->at(j).type ==
SEARCHABLE_SPEECH; j--);
            for (k = i; k < (int)pa_vec->size()-1 && pa_vec->at(k).type ==
SEARCHABLE_SPEECH; k++);
            if (pa_vec->at(j).type == PAUSE || pa_vec->at(k).type == PAUSE)
                pActiveFrameFlags[f] = 1;
        }

        if (pa_vec->at(i).type != PAUSE)
        {
            firstPauseRefPos = lastPauseRefPos = -1;
            int minRefPos = pa_vec->at(i).minPos, maxRefPos = pa_vec->at(i).maxPos;
            if (missingRefStart >= 0 && missingRefEnd >= 0)
            {
                minRefPos = (((minRefPos) < (missingRefStart)) ? (minRefPos) :
(missingRefStart));
                ReliabilityPerFrame[f] = (((ReliabilityPerFrame[f]) < (0.75f)) ?
(ReliabilityPerFrame[f]) : (0.75f));
            }
            else if (insertedSigAct >= 0)
            {
                minRefPos = (((minRefPos) < (minConsecMinPos)) ? (minRefPos) :
(minConsecMinPos));
                maxRefPos = (((maxRefPos) > (maxConsecMaxPos)) ? (maxRefPos) :
(maxConsecMaxPos));
            }
            minRefPos = (((minRefPos) < (pa_vec->at(i).refPos)) ? (minRefPos) :
(pa_vec->at(i).refPos));
            maxRefPos = (((maxRefPos) > (pa_vec->at(i).refPos)) ? (maxRefPos) :
(pa_vec->at(i).refPos));
            pSearchRangePerMacroFrameLow [f] = -pa_vec->at(i).refPos + minRefPos;
            pSearchRangePerMacroFrameHigh[f] =  maxRefPos - pa_vec->at(i).refPos;

            if (ReliabilityPerFrame[f]==0 && pSearchRangePerMacroFrameHigh[f]==0)
                pSearchRangePerMacroFrameHigh[f] += mProcessData.mStepSize;

            if (ReliabilityPerFrame[f]==0 && pSearchRangePerMacroFrameLow[f]==0)
                pSearchRangePerMacroFrameLow[f] -= mProcessData.mStepSize;

        }
        else
        {
            if (firstPauseRefPos < 0)
            {
                firstPauseRefPos = pa_vec->at(i).refPos;
                for (int j = i; j < (int)pa_vec->size() && pa_vec->at(j).type == PAUSE;
j++)
                    lastPauseRefPos = pa_vec->at(j).refPos;
            }
            pSearchRangePerMacroFrameLow [f] = -pa_vec->at(i).refPos +
firstPauseRefPos;
            pSearchRangePerMacroFrameHigh[f] = lastPauseRefPos       -
pa_vec->at(i).refPos;
        }

        mpDelayInSamplesPerFrame[f] = pa_vec->at(i).refPos - pa_vec->at(i).degPos;
```

```
        f++;

        missingRefStart = missingRefEnd = -1;
    }

    //Add frames for extra trailing silence in deg for which there is no info in
pa_vec.
    for (i--; i >= 0 && pa_vec->at(i).type == MISSING_SPEECH; i--);
    if (pa_vec->at(i).degPos + mMacroFrameSize <=
((CAudioSignal*)mppSignals[1])->mSignalLength - mMacroFrameSize)
    {
        curDegPos         = pa_vec->at(i).degPos + mMacroFrameSize;
        pa_delayInSamples = mpDelayInSamplesPerFrame[f-1];
        while (f < mNumMacroFrames && curDegPos + mMacroFrameSize <=
((CAudioSignal*)mppSignals[1])->mSignalLength)
        {
            pActiveFrameFlags            [f] = 0;
            ReliabilityPerFrame          [f] = (OTA_FLOAT)0;
            pSearchRangePerMacroFrameLow [f] = -(curDegPos+pa_delayInSamples) +
lastSpeechPos;
            pSearchRangePerMacroFrameHigh[f] = 0;
            mpDelayInSamplesPerFrame     [f] = pa_delayInSamples;
            curDegPos += mMacroFrameSize;
            f++;
        }
    }

    for (--f, i = f+1; i < mNumMacroFrames; i++)
    {
        pActiveFrameFlags            [i] = pActiveFrameFlags            [f];
        ReliabilityPerFrame          [i] = (OTA_FLOAT)0;
        pSearchRangePerMacroFrameLow [i] = pSearchRangePerMacroFrameLow [f];
        pSearchRangePerMacroFrameHigh[i] = pSearchRangePerMacroFrameHigh[f];
        mpDelayInSamplesPerFrame     [i] = mpDelayInSamplesPerFrame     [f];
    }

    int FPAResolution = ceil((OTA_FLOAT)mProcessData.mSamplerate / 8000.0);
    for (int i=0; i<mNumMacroFrames; i++)
    {
        pSearchRangePerMacroFrameLow [i] = (((pSearchRangePerMacroFrameLow [i]) <
(-FPAResolution)) ? (pSearchRangePerMacroFrameLow [i]) : (-FPAResolution));
        pSearchRangePerMacroFrameHigh[i] = (((pSearchRangePerMacroFrameHigh[i]) >
(+FPAResolution)) ? (pSearchRangePerMacroFrameHigh[i]) : (+FPAResolution));
    }

    int       OverallDelayEstimate;
    int       OverallDelayEstimate1st;
    int       OverallDelayEstimate2nd;
    OTA_FLOAT OverallDelayEstimateReliability;
    OTA_FLOAT OverallDelayEstimateReliability1st;
    OTA_FLOAT OverallDelayEstimateReliability2nd;
    int WorstResolutionInSamples2;

    EstimateOverallDelaySimpleLimits(&OverallDelayEstimate,
&OverallDelayEstimateReliability, &OverallDelayEstimate1st,
&OverallDelayEstimateReliability1st,
                                     &OverallDelayEstimate2nd,
&OverallDelayEstimateReliability2nd,
&WorstResolutionInSamples2, false);
    if (abs(OverallDelayEstimate-OverallDelayEstimate1st)<=WorstResolutionInSamples2 &&
            abs(OverallDelayEstimate-OverallDelayEstimate2nd)<=WorstResolutionInSample
s2 &&
            OverallDelayEstimateReliability>0.85)
    {
        int InitialDelay = OverallDelayEstimate;
        if (EstimateOverallDelaySimpleLimits(&OverallDelayEstimate,
&OverallDelayEstimateReliability, &OverallDelayEstimate1st,
&OverallDelayEstimateReliability1st,
                                         &OverallDelayEstimate2nd,
&OverallDelayEstimateReliability2nd,
&WorstResolutionInSamples2, true, InitialDelay,
WorstResolutionInSamples2) &&
OverallDelayEstimateReliability>0.90)
        {
            for (int f=0; f<mNumMacroFrames; f++)
```

```
            {
                pSearchRangePerMacroFrameLow [f] = -(((4) >
(WorstResolutionInSamples2)) ? (4) : (WorstResolutionInSamples2));
                pSearchRangePerMacroFrameHigh[f] =  (((4) >
(WorstResolutionInSamples2)) ? (4) : (WorstResolutionInSamples2));
                mpDelayInSamplesPerFrame     [f] =  OverallDelayEstimate;
            }
        }
    }

    return rc;
}

OTA_FLOAT CTempAlignment::DetectTAtimeDrift(void const *pPA_vec, OTA_FLOAT
frameStepInSec, int frameStepInSamples)
{

    if (pPA_vec == NULL)
        return (OTA_FLOAT)-1.0;

    TraversalVecType const *pa_vec = (TraversalVecType const*)pPA_vec;

    int vecLen = (int)pa_vec->size();
    if (vecLen <= 0 || frameStepInSec <= 0.0f || frameStepInSamples <= 0)
        return (OTA_FLOAT)-1.0;

    int   const DTD_MIN_NUM_FRAMES     = (((RINT((OTA_FLOAT)1.5 / frameStepInSec)) >
(50)) ? (RINT((OTA_FLOAT)1.5 / frameStepInSec)) : (50));
    OTA_FLOAT const DTD_MIN_TIMEDRIFT      = (OTA_FLOAT)0.05;
    OTA_FLOAT const DTD_MAX_RESIDUAL_SQERR = (OTA_FLOAT)1.5;

    OTA_FLOAT timeDrift = 0.0;
    std::auto_ptr<OTA_FLOAT> timePosRef(new OTA_FLOAT[vecLen]), timePosDeg(new
OTA_FLOAT[vecLen]);
    bool awaitingNewSeg = true;
    int i, cnt, refOffset = 0, degOffset = 0, end = 0;

    for (i = 1, cnt = 0; i < vecLen; i++)
    {
        if (pa_vec->at(i).type == MISSING_SPEECH || pa_vec->at(i).type == INSERTED_SIG
||
            pa_vec->at(i).type == PAUSE || !pa_vec->at(i).degActivity)
        {
            if (!awaitingNewSeg)
                end = i-1;

            awaitingNewSeg = true;
            continue;
        }

        if (awaitingNewSeg)
        {
            refOffset += pa_vec->at(i-1).refPos - pa_vec->at(end).refPos;
            degOffset += pa_vec->at(i-1).degPos - pa_vec->at(end).degPos;
            end = i;
            awaitingNewSeg = false;
        }
        if (!awaitingNewSeg)
        {
            timePosRef.get()[cnt] = (pa_vec->at(i).refPos - refOffset) /
(OTA_FLOAT)(frameStepInSamples);
            timePosDeg.get()[cnt] = (pa_vec->at(i).degPos - degOffset) /
(OTA_FLOAT)(frameStepInSamples);
            cnt++;
        }
    }

    if (cnt < DTD_MIN_NUM_FRAMES)
        return (OTA_FLOAT)0.0;

    OTA_FLOAT covariance = 0.0, variance = 0.0;
    OTA_FLOAT meanDeg, meanRef, residualSQerr;
    meanRef = matMean(timePosRef.get(), cnt);
    meanDeg = matMean(timePosDeg.get(), cnt);
    for (int i = 0; i < cnt; i++)
    {
```

```
        covariance += (timePosRef.get()[i] - meanRef) * (timePosDeg.get()[i] -
meanDeg);
        variance   += pow(timePosRef.get()[i] - meanRef, 2);
    }
    covariance /= (OTA_FLOAT)cnt;
    variance   /= (OTA_FLOAT)cnt;
    OTA_FLOAT slopeDiff = (covariance+1e-10f)/(variance+1e-10f);

    matbMpy1(slopeDiff, timePosRef.get(), cnt);
    meanRef = matMean(timePosRef.get(), cnt);

    matbAdd1(meanDeg-meanRef, timePosRef.get(), cnt);

    matbSub3(timePosRef.get(), timePosDeg.get(), timePosRef.get(), cnt);
    matbAbs1(timePosRef.get(),                                      cnt);
    residualSQerr = 0.0;
    for (int i = 0; i < cnt; i++)
        residualSQerr += timePosRef.get()[i] * timePosRef.get()[i];
    residualSQerr /= (OTA_FLOAT)cnt;

    OTA_FLOAT slopeFac, errFac;
    slopeFac = (fabs(slopeDiff - 1) - DTD_MIN_TIMEDRIFT) / DTD_MIN_TIMEDRIFT;
    errFac   = sqrt((((1.0f - (residualSQerr / DTD_MAX_RESIDUAL_SQERR)) > (0.0f)) ?
(1.0f - (residualSQerr / DTD_MAX_RESIDUAL_SQERR)) : (0.0f)));
    slopeFac = ((((((slopeFac) > (0.0f)) ? (slopeFac) : (0.0f))) < (1.0f)) ?
((((slopeFac) > (0.0f)) ? (slopeFac) : (0.0f))) : (1.0f));
    errFac   = ((((((errFac) > (0.0f)) ? (errFac) : (0.0f))) < (1.0f)) ? ((((errFac) >
(0.0f)) ? (errFac) : (0.0f))) : (1.0f));

    return ((((((slopeFac*errFac) > ((OTA_FLOAT)0.0)) ? (slopeFac*errFac) :
((OTA_FLOAT)0.0))) < ((OTA_FLOAT)1.0)) ? ((((slopeFac*errFac) > ((OTA_FLOAT)0.0)) ?
(slopeFac*errFac) : ((OTA_FLOAT)0.0))) : ((OTA_FLOAT)1.0));

}

bool CTempAlignment::RunPrealignment(int* pActiveFrameFlags, OTA_FLOAT*
ReliabilityPerFrame, int* pSearchRangePerMacroFrameLow, int
*pSearchRangePerMacroFrameHigh, int TArunIndex, bool &doRevertToOPTprealignment)
{
    bool rc=true;
    long MaxDelayVecLen = mppSignals[1]->mSignalLength;

    void *PAHandle = NULL;
    TraversalVecType const *pa_vec = NULL;

    if (!doRevertToOPTprealignment)
    {
        rc = SectionsFromSQPrealignment(&PAHandle);
        fflush(mProcessData.mpLogFile);

        OTA_FLOAT landmarkPA_matchQual = PreAlignment_GetMatchQuality(PAHandle);
        ;

        doRevertToOPTprealignment  = landmarkPA_matchQual < (OTA_FLOAT)0.75;

    }

    if (!doRevertToOPTprealignment)
    {
        int   pa_sampleRate       = (int) ((CAudioSignal*)mppSignals[0])->mSampleRate;
        OTA_FLOAT pa_frameLengthInSec = mMacroFrameSize / (OTA_FLOAT)pa_sampleRate;
        pa_vec = PreAlignment_Traverse(PAHandle, pa_frameLengthInSec,
pa_frameLengthInSec);
        if (pa_vec == NULL)
        {
            PreAlignment_Free(&PAHandle);
            doRevertToOPTprealignment = true;
        }

        OTA_FLOAT timeDriftFac = DetectTAtimeDrift((void*)pa_vec, pa_frameLengthInSec,
mMacroFrameSize);

        if (timeDriftFac > (OTA_FLOAT)0.30)
            doRevertToOPTprealignment = true;

        ;
```

```
    }

    if (!doRevertToOPTprealignment)
    {
        rc = SegmentList2DelayVector(&PAHandle, (void**)&pa_vec, 1, pActiveFrameFlags,
ReliabilityPerFrame, pSearchRangePerMacroFrameLow,
pSearchRangePerMacroFrameHigh);
        fflush(mProcessData.mpLogFile);

            int i;
            for (i=0; i<mNumMacroFrames && !pActiveFrameFlags[i]; i++);

            if (i<mNumMacroFrames)
            {
                int Delay = mpDelayInSamplesPerFrame[i];
                OTA_FLOAT Reliability = 0;
                int SL = pSearchRangePerMacroFrameLow[i];
                int SH = pSearchRangePerMacroFrameHigh[i];
                for (int j=0; j<i; j++)
                {
                    mpDelayInSamplesPerFrame[j] = Delay;
                    ReliabilityPerFrame[j] = Reliability;
                    pSearchRangePerMacroFrameLow[j] = SL;
                    pSearchRangePerMacroFrameHigh[j] = SH;
                }
            }

        PreAlignment_Free(&PAHandle);
        pa_vec   = NULL;
        PAHandle = NULL;
    }
    else
    {
        PreAlignment_Free(&PAHandle);
        PAHandle = NULL;
        pa_vec   = NULL;

        ;
        ResetSectionData();

        int WorstResolutionInSamples=0;

        {
            int StartFrame=0;
            int* pActiveFrameFlagsRef = new int[MaxDelayVecLen];
            int NumRefFrames=mpActiveFrameDetection->GetActiveFrameFlags(0, 0,
mProcessData.mStepSize, pActiveFrameFlagsRef, MaxDelayVecLen);
            while (StartFrame<NumRefFrames && pActiveFrameFlagsRef[StartFrame]<=0)
StartFrame++;
            GetDelayLimits(FramesToSamples(StartFrame),
&mProcessData.mMaxStaticDelayInMs, &mProcessData.mMinStaticDelayInMs);
            delete[] pActiveFrameFlagsRef;
        }

        bool
HighReliableInitialDelay=EstimateOverallDelaySimpleLimits(&mOverallDelayEstimat
e, &mOverallDelayEstimateReliability, &mOverallDelayEstimate1st,
&mOverallDelayEstimateReliability1st,
                                        &mOverallDelayEstimate2nd,
&mOverallDelayEstimateReliability2nd,
&WorstResolutionInSamples, false);
        if (HighReliableInitialDelay)
        {

            int       OverallDelayEstimate;
            int       OverallDelayEstimate1st;
            int       OverallDelayEstimate2nd;
            OTA_FLOAT OverallDelayEstimateReliability;
            OTA_FLOAT OverallDelayEstimateReliability1st;
            OTA_FLOAT OverallDelayEstimateReliability2nd;
            int WorstResolutionInSamples2;
            if (0 && EstimateOverallDelaySimpleLimits(&OverallDelayEstimate,
&OverallDelayEstimateReliability, &OverallDelayEstimate1st,
&OverallDelayEstimateReliability1st,
                                        &OverallDelayEstimate2nd,
&OverallDelayEstimateReliability2nd,
```

```
&WorstResolutionInSamples2, true))
            {
                mOverallDelayEstimate = OverallDelayEstimate;
                mOverallDelayEstimate1st = OverallDelayEstimate1st;
                mOverallDelayEstimate2nd = OverallDelayEstimate2nd;
                mOverallDelayEstimateReliability = OverallDelayEstimateReliability;
                mOverallDelayEstimateReliability1st =
OverallDelayEstimateReliability1st;
                mOverallDelayEstimateReliability2nd =
OverallDelayEstimateReliability2nd;
                WorstResolutionInSamples = WorstResolutionInSamples2;
            }
        }

        mNumReparsePoints = IdentifyReparsePoints(mpReparsePoints, 100,
pActiveFrameFlags, mOverallDelayEstimate, mOverallDelayEstimateReliability,
                                                  mOverallDelayEstimate1st,
mOverallDelayEstimateReliability1st
, mOverallDelayEstimate2nd,
mOverallDelayEstimateReliability2nd
);

        ;
        GetInitialDelaysInSamples(mpReparsePoints, mNumReparsePoints,
pActiveFrameFlags, mOverallDelayEstimate, mOverallDelayEstimateReliability,
&WorstResolutionInSamples);
        mStartSampleRef = mpReparsePoints[0].Ref.Start;
        mStartSampleDeg = mpReparsePoints[0].Deg.Start;

        SetActiveFrameFlags(mMacroFrameSize, pActiveFrameFlags);

        for (int r=0; r<mNumReparsePoints; r++)
        {

            if (mpReparsePoints[r].Reliability>0.95 && HighReliableInitialDelay)

            {
                mpReparsePoints[r].MinDelayVarInSamples = -2*WorstResolutionInSamples;
                mpReparsePoints[r].MaxDelayVarInSamples = 2*WorstResolutionInSamples;
            }
            else
            {
                mpReparsePoints[r].MinDelayVarInSamples =
mProcessData.mMinLowVarDelayInSamples;
                mpReparsePoints[r].MaxDelayVarInSamples =
mProcessData.mMaxHighVarDelayInSamples;
            }

        }

        ReparseSections2DelayVector(ReliabilityPerFrame, pSearchRangePerMacroFrameLow,
pSearchRangePerMacroFrameHigh, WorstResolutionInSamples);

    }

    return rc;
}
#pragma endregion

#pragma region  Coarse Alignment

//Input:
    //&IterationData
    // pActiveFrameFlags,  DelayVec
    //FrameWithLastValidDelay
    //Path, ReliabilityPerFrame

//Output:
//  ReliabilityPerFrame, DelayVec

void CTempAlignment::LogDelayVec(const char* Title, int FeatureLength, long* DelayVec,
OTA_FLOAT* ReliabilityPerFrame, int* pActiveFrameFlags)
{

}
```

```
void CTempAlignment::CoarseAlignmentLog(FILE* pLogFile, long* DelayVec, int
FeatureLength, int* Path, CProcessData* pIterationData, int* pRelativeDelayPerFrame,
OTA_FLOAT* ReliabilityPerFrame)
{

}

long CTempAlignment::modifyActiveFrameFlags(unsigned int numSimpleAnalysisFrames, int*
pActiveFrameFlags, int* pActiveFrameFlagsSimplified)
{
    bool activeBlock = false;
    int activeBlockLength = 0;
    matibZero(pActiveFrameFlagsSimplified, mNumMacroFrames);

    long startFrameSimplified = -1;
    for(int frame = 0; frame < mNumMacroFrames; frame++)
    {
        if(activeBlock)
        {
            if(pActiveFrameFlags[frame])
            {
                if(activeBlockLength < numSimpleAnalysisFrames)
                    pActiveFrameFlagsSimplified[frame] = 1;
                activeBlockLength++;
            }
            else
            {

                activeBlock = false;
                activeBlockLength = 0;
            }
        }
        else
            if(pActiveFrameFlags[frame])
            {
                activeBlock = true;
                if(startFrameSimplified == -1)
                    startFrameSimplified = frame;
                pActiveFrameFlagsSimplified[frame] = 1;
                activeBlockLength++;
            }
    }
    return startFrameSimplified;
}

bool CTempAlignment::CoarseAlignmentFirstRun2(CProcessData* pIterationData, int*
pActiveFrameFlags, int StartFrame, int NumFrames, int* DelayVecOffset, long* DelayVec,
OTA_FLOAT* ReliabilityPerFrame)
{
    int d;

    int AlertThresholdP =
(int)(0.95*(pIterationData->mMaxHighVarDelay-pIterationData->mMinLowVarDelay));
    int AlertThresholdM =
(int)(0.05*(pIterationData->mMaxHighVarDelay-pIterationData->mMinLowVarDelay));
    bool Alert=false;
    int LastGoodFrame=-1;

    for (d=0; d<NumFrames-1; d++)
    {
        if (pActiveFrameFlags[d]
             && (DelayVecOffset[d]>AlertThresholdP ||
DelayVecOffset[d]<AlertThresholdM)
             && (DelayVecOffset[d+1]>AlertThresholdP ||
DelayVecOffset[d+1]<AlertThresholdM)
             && ReliabilityPerFrame[d]>0.7)

        {
            LastGoodFrame = d-1;
            Alert=true;
            break;
        }
    }

    if (Alert && LastGoodFrame>=0 && pActiveFrameFlags[LastGoodFrame])
    {
```

```
        int UseDelay = DelayVec[LastGoodFrame];
        d = LastGoodFrame;
        while (d<NumFrames && pActiveFrameFlags[d])
            DelayVec[d++] = UseDelay;
    }

    return Alert;
}

int CTempAlignment::CoarseAlignmentNewWindowSize(CProcessData* pIterationData, int
Loop, int DegStep, int* pSearchRangeLow, int* pSearchRangeHigh, long* DelayVec, long*
pDelayInSamples, PLOT_VECTOR* pVecs, int* pNumVecs)
{
    int MinLowVarDelayInFF;
    int MaxHighVarDelayInFF;
    int LastWindowSize = pIterationData->mWindowSize;

    pIterationData->mMinLowVarDelayInSamples  = -LastWindowSize*2;
    pIterationData->mMaxHighVarDelayInSamples =  LastWindowSize*2;

    ;
    OTA_FLOAT IterationRatio =
(pIterationData->mpWindowSize[Loop-1]-pIterationData->mpOverlap[Loop-1]) /
(pIterationData->mpWindowSize[Loop]-pIterationData->mpOverlap[Loop]);

    pIterationData->Init(Loop, 1.0);
    DegStep *= IterationRatio;

    MinLowVarDelayInFF = pIterationData->mMinLowVarDelayInSamples / DegStep;
    MaxHighVarDelayInFF = pIterationData->mMaxHighVarDelayInSamples / DegStep;

    OTA_FLOAT StepF = (OTA_FLOAT)pIterationData->mStepSize;
    ;
    for (int d=0; d<mNumMacroFrames; d++)
    {
        DelayVec[d] *= IterationRatio;

        mCAIntermediate.pSearchRangeLow[d] *= IterationRatio;
        mCAIntermediate.pSearchRangeHigh[d] *= IterationRatio;

        int SearchRangeInSamplesIn = mCAIntermediate.pSearchRangeHighIn[d] -
mCAIntermediate.pSearchRangeLowIn[d];
        if (SearchRangeInSamplesIn<StepF)
        {
            int SearchRangeInSamples = (mCAIntermediate.pSearchRangeHigh[d] -
mCAIntermediate.pSearchRangeLow[d])* StepF;
            if (SearchRangeInSamples>SearchRangeInSamplesIn)
            {
                if (DelayVec[d]*StepF<mCAIntermediate.pSearchRangeLowIn[d] ||
DelayVec[d]*StepF>mCAIntermediate.pSearchRangeHighIn[d])
                    DelayVec[d] = (long)floor(pDelayInSamples[d] / StepF);

                if (mCAIntermediate.pSearchRangeHighIn[d]!=0 &&
mCAIntermediate.pSearchRangeLowIn[d]!=0)
                {
                    int SearchRangeShift = pDelayInSamples[d]-DelayVec[d]*StepF;
                    mCAIntermediate.pSearchRangeLow[d]  =
mCAIntermediate.pSearchRangeLowIn[d] + SearchRangeShift;
                    mCAIntermediate.pSearchRangeLow[d]  =
(((mCAIntermediate.pSearchRangeLow[d]) < (0)) ?
(mCAIntermediate.pSearchRangeLow[d]) : (0));
                    mCAIntermediate.pSearchRangeHigh[d] =
mCAIntermediate.pSearchRangeHighIn[d] + SearchRangeShift;
                    mCAIntermediate.pSearchRangeLow[d] =
(int)floor(mCAIntermediate.pSearchRangeLow[d] /StepF);
                    mCAIntermediate.pSearchRangeHigh[d] = (int)ceil
(mCAIntermediate.pSearchRangeHigh[d]/StepF);
                }
            }
        }

    }

    return DegStep;
}
```

```
void CTempAlignment::CoarseAlignmentReduceSearchRangeAtSectionEnd(CProcessData*
pIterationData)
{
    int Frame=0;
    int NumConstFramesBeforePause = PIMSecondsToFrames(150);
    int ReducedSearchRange  = PIMSecondsToFrames(32);

    int* pActiveFrameFlags=mCAIntermediate.pActiveFrameFlags;
    int IsActiveSection = pActiveFrameFlags[Frame];
    for (Frame=0; Frame<mNumMacroFrames; Frame++)
    {
        if (IsActiveSection)
        {
            if(!pActiveFrameFlags[Frame])
            {
                int LastDelay = mCAIntermediate.pDelayVec[(((0) >
(Frame-NumConstFramesBeforePause)) ? (0) :
(Frame-NumConstFramesBeforePause))];

                for (int j=Frame-1; j>=Frame-NumConstFramesBeforePause && j>=0 &&
pActiveFrameFlags[j];j++)
                {

                    int CenterIndex = -mCAIntermediate.pDelayVec[j] + LastDelay;
                    mCAIntermediate.pSearchRangeLow[j]  =
(((mCAIntermediate.pSearchRangeLow[j]) >
(CenterIndex-ReducedSearchRange)) ?
(mCAIntermediate.pSearchRangeLow[j]) :
(CenterIndex-ReducedSearchRange));
                    mCAIntermediate.pSearchRangeHigh[j] =
(((mCAIntermediate.pSearchRangeHigh[j]) <
(CenterIndex+ReducedSearchRange)) ?
(mCAIntermediate.pSearchRangeHigh[j]) :
(CenterIndex+ReducedSearchRange));

                    mCAIntermediate.pOptionsApplied[j] |=
APPL_REDUCED_SEARCHRANGE_AT_SECTION_END;
                }
            }
        }

        IsActiveSection = pActiveFrameFlags[Frame];
    }
}

bool CTempAlignment::CoarseAlignment(CProcessData* pIterationData, int*
pActiveFrameFlags, int StartFrame, int* pSearchRangeLow, int* pSearchRangeHigh, long*
DelayVec, OTA_FLOAT* ReliabilityPerFrame, bool doRevertToOPTprealignment, PLOT_VECTOR*
pVecs, int* pNumVecs)
{
    ;

    bool rc = true;
    int StartPlotIteration=mProcessData.mStartPlotIteration;
    int LastPlotIteration =mProcessData.mLastPlotIteration;
    bool EnablePlotting=mProcessData.mEnablePlotting;
    int i, f, d;
    OTA_FLIST_TYPE FeatureListType = OTA_FLTYPE_COARSE_ALIGN;

    long* pDelayInSamples = (long*)matMalloc(sizeof(long)*mNumMacroFrames);
    memcpy(pDelayInSamples, DelayVec, sizeof(long)*mNumMacroFrames);
    mCAIntermediate.pDelayVec = DelayVec;
    mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags;
    int *Path = mCAIntermediate.pOptOffset;

    memcpy(mCAIntermediate.pSearchRangeLow,  pSearchRangeLow,
sizeof(int)*mNumMacroFrames);
    memcpy(mCAIntermediate.pSearchRangeHigh, pSearchRangeHigh,
sizeof(int)*mNumMacroFrames);

    mCAIntermediate.pSearchRangeLowIn = pSearchRangeLow;
    mCAIntermediate.pSearchRangeHighIn = pSearchRangeHigh;

    int Loop = 3;
    pIterationData->Init(Loop, 1.0);
    int DegStep = (pIterationData->mpWindowSize[1]-pIterationData->mpOverlap[1]) /
```

```
(pIterationData->mpWindowSize[Loop]-pIterationData->mpOverlap[Loop]);

    float StepF = (float)pIterationData->mStepSize;

    for (d=0; d<mNumMacroFrames; d++)
    {
        DelayVec[d] = (long)floor(pDelayInSamples[d] / StepF);

        if (mCAIntermediate.pSearchRangeHigh[d]!=0 &&
mCAIntermediate.pSearchRangeLow[d]!=0)
        {
            int SearchRangeShift =
pDelayInSamples[d]-DelayVec[d]*pIterationData->mStepSize;
            mCAIntermediate.pSearchRangeLow[d] += SearchRangeShift;
            mCAIntermediate.pSearchRangeLow[d] = (((mCAIntermediate.pSearchRangeLow[d])
< (0)) ? (mCAIntermediate.pSearchRangeLow[d]) : (0));
            mCAIntermediate.pSearchRangeHigh[d] += SearchRangeShift;
        }

        mCAIntermediate.pSearchRangeLow[d] =
(int)floor(mCAIntermediate.pSearchRangeLow[d] /StepF);
        mCAIntermediate.pSearchRangeHigh[d] = (int)ceil
(mCAIntermediate.pSearchRangeHigh[d]/StepF);
    }

    int* FrameWithLastValidDelay = new int[mNumMacroFrames];
    for (d=0; d<mNumMacroFrames; d++)
        FrameWithLastValidDelay[d] = d;
    mCAIntermediate.pFrameWithLastValidDelay = FrameWithLastValidDelay;

    long MinDelay = matMin(DelayVec, mNumMacroFrames);
    long MaxDelay = matMax(DelayVec, mNumMacroFrames);
    long startFrameSimplified = -1;
    int *pActiveFrameFlagsSimplified = 0;
    OTA_FLOAT MinReliability = matMin(ReliabilityPerFrame, mNumMacroFrames);
    bool simplified = false;

    unsigned int numSimpleAnalysisFrames = (unsigned
int)(400*(double)(pIterationData->mSamplerate)/1000.0/(double)mMacroFrameSize);

    if (MinDelay==MaxDelay && MinReliability > 0.9)
    {
        ;
        simplified = true;
        FeatureListType = OTA_FLTYPE_COARSE_ALIGN_SIMPLIFIED;

        pActiveFrameFlagsSimplified = new int[mNumMacroFrames];

        startFrameSimplified = modifyActiveFrameFlags(numSimpleAnalysisFrames,
pActiveFrameFlags, pActiveFrameFlagsSimplified);
        mCAIntermediate.pActiveFrameFlags = pActiveFrameFlagsSimplified;
    }
    else
        mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags;

    OTA_FLOAT*  pDelayVecBackup=new OTA_FLOAT[mNumMacroFrames];
    int*        pFrameWithLastValidDelayBackup=new int[mNumMacroFrames];
    bool AllIterationsDone=false;
    bool FirstRun=true;
    bool GoToNextIteration=true;
    bool RecalculateFeatures=true;
    bool ReprocessCorrelationMatrix=false;
    int DelayLimitsExceeded = 0;
    while(rc && !AllIterationsDone)
    {
        ;
        mCAIntermediate.CurrentLoop = Loop;
        GoToNextIteration=true;

        for (i=0; i<mNumMacroFrames; i++)
            pDelayVecBackup[i] = DelayVec[i];
        for (i=0; i<mNumMacroFrames; i++)
            pFrameWithLastValidDelayBackup[i] = FrameWithLastValidDelay[i];

        GetPitchVector(1, 0, mCAIntermediate.pPitchVec, mNumMacroFrames,
mMacroFrameSize);
```

```
        if (RecalculateFeatures)
        {
            ;
            rc = mpFeatureList->Create(mppSignals, pIterationData, FeatureListType);
            ReprocessCorrelationMatrix = true;
        }

        if (ReprocessCorrelationMatrix)
        {
            for (int i=0; i<mNumMacroFrames; i++)
                mCAIntermediate.pOptionsApplied[i]=0;

            ;
            if (rc) rc = mpDelaySearch->CreateMatrix(mpFeatureList, &mCAIntermediate,
pIterationData, mNumMacroFrames, DegStep);

            mProcessData.mP.mViterbi.UseRelDistance = true;

            //Combine all matrices to the matrix[0]. CombineMatricesAndFeatures()
should be overloaded by the
            //specific implementation. The default version does nothing.
            //After this method was called, the correlation matrix for feature 0 left
            //is the only one which is evaluated.
            if (rc) rc = mpDelaySearch->CombineMatricesAndFeatures(StartFrame, DegStep,
&mCAIntermediate);
        }

        int FeatureLength = mpDelaySearch->mpCorrMatrix[0][0].mNumMacroFrames;

        //If combining the matrices decided that some frames have invalid delays, then
correct
        //the delay of those frames to match the delay of the last valid frame.
        //CombineMatricesAndFeatures() must have set the correlation vectors of those
        //frames to ...0 0 0 0 0 0 0 0 0... This will force the Viterbi algorithm
        //to keep the delay constant.
        //WARNING: This shifts delay jumps during inactive phases to the beginning of
        //the next active section!

        int* pRelativeDelayPerFrame = mCAIntermediate.pRelativeDelayPerFrame;
        pRelativeDelayPerFrame[0] = 0;
        for (f=1; f<mNumMacroFrames; f++)
            pRelativeDelayPerFrame[f] = DelayVec[f] - DelayVec[f-1];

        if(1 || mCAIntermediate.CurrentLoop==3)
            CoarseAlignmentReduceSearchRangeAtSectionEnd(pIterationData);

        if (rc) rc = mpDelaySearch->CalcOptimumPath(DegStep, &mCAIntermediate, 0, 0,
ReliabilityPerFrame);

        OTA_FLOAT MinReliability=1.0;
        for (i=0; i<FeatureLength; i++)
        {
            mpReliabilityPerFrame[i] = ReliabilityPerFrame[i];
            if (ReliabilityPerFrame[i]>0 && ReliabilityPerFrame[i]<MinReliability)
                MinReliability=ReliabilityPerFrame[i];
        }

        CoarseAlignmentLog(pLogFile, DelayVec, FeatureLength, Path, pIterationData,
pRelativeDelayPerFrame, ReliabilityPerFrame);

        for (d=0; d<FeatureLength; d++)
            DelayVec[d] = DelayVec[d] + Path[d] + pIterationData->mMinLowVarDelay;

        mpDelaySearch->CleanupPath(&mCAIntermediate, DelayVec, FeatureLength,
FrameWithLastValidDelay, DegStep);
        LogDelayVec("DelayVec per frame in CA after CleanupPath()", FeatureLength,
DelayVec, ReliabilityPerFrame, pActiveFrameFlags);

        if (FirstRun)
        {
            bool DoAgain = CoarseAlignmentFirstRun2(pIterationData,
mCAIntermediate.pActiveFrameFlags, StartFrame, FeatureLength, Path,
DelayVec, ReliabilityPerFrame);
            if (DoAgain && !simplified)
            {
```

```
                GoToNextIteration = false;
                RecalculateFeatures = false;
                ReprocessCorrelationMatrix=true;
                DelayLimitsExceeded++;
            }
            else GoToNextIteration = true;
        }

        if (GoToNextIteration)
        {
            ;
            long MinDelay = matMin(DelayVec, mNumMacroFrames);
            long MaxDelay = matMax(DelayVec, mNumMacroFrames);

            if ((MaxDelay-MinDelay)>1 || MinReliability < 0.9)
            {
                if(simplified)
                {
                    ;

                    for (int i=0; i<mNumMacroFrames; i++)
                        DelayVec[i] = (long)pDelayVecBackup[i];
                    for (int i=0; i<mNumMacroFrames; i++)
                        FrameWithLastValidDelay[i] = pFrameWithLastValidDelayBackup[i];

                    FeatureListType = OTA_FLTYPE_COARSE_ALIGN;
                    GoToNextIteration = false;
                    RecalculateFeatures = true;
                    simplified = false;
                    mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags;
                }
            }
        }

        //Report large delay variations
        if (GoToNextIteration && mProcessData.mpLogFile)
        {
            ;
            ;
            for (d=1; d<FeatureLength; d++)
            {
                int DelayDifferenceInFrames = DelayVec[d]-DelayVec[d-1];
                if (abs(DelayDifferenceInFrames)>20)
                    ;
            }
        }

        if (true)
        {
            if (!GoToNextIteration)
            {
                if (FirstRun && DelayLimitsExceeded>1)
                {
                    ;

                    pIterationData->mMinLowVarDelayInSamples *= 2;
                    pIterationData->mMaxHighVarDelayInSamples *= 2;

                    if (pIterationData->mMinLowVarDelayInSamples <
-((int)(0.3*pIterationData->mSamplerate)))
                    {
                        pIterationData->mMinLowVarDelayInSamples =
-((int)(0.3*pIterationData->mSamplerate));
                        GoToNextIteration=true;
                    }
                    if (pIterationData->mMaxHighVarDelayInSamples >
((int)(0.3*pIterationData->mSamplerate)))
                    {
                        pIterationData->mMaxHighVarDelayInSamples =
((int)(0.3*pIterationData->mSamplerate));
                        GoToNextIteration=true;
                    }
                    pIterationData->Init(Loop, 1.0);
                }
            }
        }
```

```
        //*********************************************************
        //Choose the next iteration parameters or terminate the loop
        if (GoToNextIteration)
        {
            int MinLowVarDelayInFF;
            int MaxHighVarDelayInFF;
            int LastWindowSize = pIterationData->mWindowSize;

            pIterationData->mMinLowVarDelayInSamples  = -LastWindowSize*2;
            pIterationData->mMaxHighVarDelayInSamples =  LastWindowSize*2;

            if (mProcessData.mEnablePlotting)
                SetVecInfoTimeSeries(&pVecs[(*pNumVecs)++], mpDelayInSamplesPerFrame,
mNumMacroFrames, 0,0, (OTA_FLOAT)pIterationData->mStepSize, "Delay
After CA Downsampling %d", pIterationData->mStepSize);

            if (++Loop<8 && pIterationData->mpWindowSize[Loop]>0)
            {
                DegStep = CoarseAlignmentNewWindowSize(pIterationData, Loop, DegStep,
pSearchRangeLow, pSearchRangeHigh, DelayVec, pDelayInSamples, pVecs,
pNumVecs);
                RecalculateFeatures = true;
            }
            else
            {
                AllIterationsDone = true;
                ;
            }
            FirstRun = false;
        }
    }

    {
        OTA_FLOAT StepF = (OTA_FLOAT)pIterationData->mStepSize;
        for (int d=0; d<mNumMacroFrames; d++)
        {
            if (pSearchRangeLow[d]==0 && pSearchRangeHigh[d]==0)
                DelayVec[d] = (long)floor(pDelayInSamples[d] / StepF);

            pSearchRangeLow[d] = (((pSearchRangeLow[d]) >
(pIterationData->mMinLowVarDelayInSamples)) ? (pSearchRangeLow[d]) :
(pIterationData->mMinLowVarDelayInSamples));
            pSearchRangeHigh[d] = (((pSearchRangeHigh[d]) <
(pIterationData->mMaxHighVarDelayInSamples)) ? (pSearchRangeHigh[d]) :
(pIterationData->mMaxHighVarDelayInSamples));
        }
    }

    delete[] pDelayVecBackup;
    delete[] pFrameWithLastValidDelayBackup;

    if(pActiveFrameFlagsSimplified)
        delete[] pActiveFrameFlagsSimplified;

    if (simplified)
    {
        ;
        int RefFrame = startFrameSimplified + numSimpleAnalysisFrames/2;
        for (d = startFrameSimplified + numSimpleAnalysisFrames; d < mNumMacroFrames;
d++)
        {
            mpReliabilityPerFrame[d] = mpReliabilityPerFrame[RefFrame];
            DelayVec[d] = DelayVec[RefFrame];
            ReliabilityPerFrame[d] = ReliabilityPerFrame[RefFrame];
        }
    }

    ;
    ;
    for (i=0; i<mNumMacroFrames; i++)
        DelayVec[i] = DelayVec[i] * pIterationData->mStepSize;

    if (pDelayInSamples) matFree(pDelayInSamples);

    ;
```

```
    return rc;
}
#pragma endregion

void CTempAlignment::DetermineInvalidSections()
{
    mpResults->mpIgnoreFlags = (int*)matMalloc(sizeof(int)*mNumMacroFrames);
    matbSet(0, mpResults->mpIgnoreFlags, mNumMacroFrames);

    int* pFlags = mpResults->mpIgnoreFlags;
    int* pActiveFrameFlags = mFAIntermediate.pActiveFrameFlags;

    const int SignificantChange = MSecondsToSamples(50);
    const int SmallChange = MSecondsToSamples(10);
    for (int i=1; i<mNumMacroFrames; i++)
    {
        if (i>1 && pActiveFrameFlags[i] && pActiveFrameFlags[i-1] &&
mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]>0)
        {
            int FramesToSearch =
(mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]) /
mProcessData.mStepSize+1;
            for (int k=(((0) > (i-FramesToSearch)) ? (0) : (i-FramesToSearch)); k<i;
k++)
                pFlags[k] |= 1;
        }

        if (i>1 && pActiveFrameFlags[i] && pActiveFrameFlags[i-1]
            &&
mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]>SignificantChange
)
        {
            int FramesToSearch =
(mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]) /
mProcessData.mStepSize+1;
            for (int k=(((0) > (i-FramesToSearch)) ? (0) : (i-FramesToSearch)); k<i;
k++)
            {
                pFlags[k] |= 2;
                if (pActiveFrameFlags[k] && mFAIntermediate.pReliabilityPerFrame[k]<0.7
&&
abs(int(mpDelayInSamplesPerFrame[k]-mpDelayInSamplesPerFrame[i-1]))<Sma
llChange
                    &&
mFAIntermediate.pRefEnergy[k+mpDelayInSamplesPerFrame[k]/mProcessDa
ta.mStepSize]>10.0*mFAIntermediate.pDegEnergy[k])
                    pFlags[k] |= 4;
            }
        }
    }

    for (int i=1; i<mNumMacroFrames; i++)
    {
        if (pFlags[i] & 4)
        {
            int DelayToLastGood = mpDelayInSamplesPerFrame[i]-mProcessData.mStepSize;
            while (i<mNumMacroFrames && (pFlags[i] & 4))
            {
                mpDelayInSamplesPerFrame[i++] = DelayToLastGood;
                DelayToLastGood -= mProcessData.mStepSize;
            }
        }
    }

}

bool CTempAlignment::Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)
{

    ;

    char UsedDefines[4096];
    sprintf(UsedDefines, "\nRepository Info:\n");
    strcat(UsedDefines, "\tRev.  ");
    strcat(UsedDefines, POLQA_SVN_REVISION);
```

```
    strcat(UsedDefines, "\n\tMod.  ");
    strcat(UsedDefines, POLQA_SVN_MODIFIED);
    strcat(UsedDefines, "\n\tRange ");
    strcat(UsedDefines, POLQA_SVN_RANGE);
    strcat(UsedDefines, "\n\Global defines used (read from version.h):\n");
    ;

    MAT_HANDLE  mh=(MAT_HANDLE)mProcessData.mpMathlibHandle;
    TACheckTimeMatInit(mh, 2);

    long ClockCycles=0;
    double TimeDiffIdentifyReparsePoints=-1, TimeDiffInitalDelay=-1,
TimeDiffCoarseAlignment=-1, TimeDiffFineAlignment=-1, TimeDiffCleanup=-1;

    int StartPlotIteration=mProcessData.mStartPlotIteration;
    int LastPlotIteration =mProcessData.mLastPlotIteration;
    bool EnablePlotting=mProcessData.mEnablePlotting;

    bool rc = true;
    int i, r, d;

    int NumSpareFrames=0;

    mNumMacroFrames=0;
    mProcessData.Init(1, 1);
    mMacroFrameSize = mProcessData.mStepSize;

    unsigned long TriggerPoint = 0;
    int TriggerPointInFrames = 0;
    mpResults->FirstRefSample = 0;
    mpResults->FirstDegSample = 0;

    long MaxDelayVecLen = mppSignals[1]->mSignalLength+TriggerPoint;

    mNumReparsePoints = 1;
    if (mpReparsePoints) delete[] mpReparsePoints;
    mpReparsePoints = new REPARSE_POINT[100];

    OTA_FLOAT RelativeSamplerateDifference = 0;
    int *pActiveFrameFlags = 0;

    mOverallDelayEstimate=0;
    mOverallDelayEstimateReliability = -1;
    if (mpDelayInSamplesPerFrame) delete[] mpDelayInSamplesPerFrame;
    mpDelayInSamplesPerFrame = 0;
    if (mpReliabilityPerFrame) delete[] mpReliabilityPerFrame;
    mpReliabilityPerFrame = 0;

    mStartOffset = FindUsedSectionOfRefSignal();

    int StartOffsetInFrames = abs(mStartOffset) / mMacroFrameSize;

    if (mStartOffset)
    {
        if (mStartOffset<0)
            mppSignals[1]->SetOffset(-mStartOffset);
        else
            mppSignals[0]->SetOffset(mStartOffset);

        NumSpareFrames += StartOffsetInFrames;
    }

    ResetSectionData();

    OTA_FLOAT ERef = mppSignals[0]->GetEnergy(0);
    OTA_FLOAT EDeg = mppSignals[1]->GetEnergy(0);

    OTA_FLOAT Factor = ERef / EDeg;
    if (Factor<0.1) Factor = 0.1;
    if (Factor>10.0) Factor = 10.0;

    mppSignals[1]->Amplify(0, sqrt(Factor));

    pActiveFrameFlags = (int*)matCalloc(MaxDelayVecLen, sizeof(int));

    mpActiveFrameDetection->Init(&mProcessData);
```

```
    mpActiveFrameDetection->Start(mppSignals);
    mProcessData.Init(1, 1);
    mNumMacroFrames = mpActiveFrameDetection->GetActiveFrameFlags(1, 0,
mProcessData.mStepSize, pActiveFrameFlags, MaxDelayVecLen);
    ;

    mpFeatureList2->Create(mppSignals, &mProcessData, OTA_ENERGYPERFRAME);

    mpDelayInSamplesPerFrame = (long*)matCalloc((mNumMacroFrames+NumSpareFrames),
sizeof(long));

    mpReliabilityPerFrame = (OTA_FLOAT*)matCalloc((mNumMacroFrames+NumSpareFrames),
sizeof(OTA_FLOAT));

    OTA_FLOAT* ReliabilityPerFrame =
(OTA_FLOAT*)matCalloc((mNumMacroFrames+NumSpareFrames), sizeof(OTA_FLOAT));

    int* pSearchRangePerMacroFrameLow =
(int*)matCalloc((mNumMacroFrames+NumSpareFrames), sizeof(int));
    int* pSearchRangePerMacroFrameHigh =
(int*)matCalloc((mNumMacroFrames+NumSpareFrames), sizeof(int));

    bool doRevertToOPTprealignment = TArunIndex > 0;
    if(mppSignals[1]->mSignalLength/mProcessData.mSamplerate > MAX_SPEECH_DURATION)
doRevertToOPTprealignment=true;
    if(mppSignals[0]->mSignalLength/mProcessData.mSamplerate > MAX_SPEECH_DURATION)
doRevertToOPTprealignment=true;

    int StartFrame = 0;

    rc = RunPrealignment(pActiveFrameFlags, ReliabilityPerFrame,
pSearchRangePerMacroFrameLow, pSearchRangePerMacroFrameHigh, TArunIndex,
doRevertToOPTprealignment);

    if (rc)
    {
        TACheckTimeMatEval(mh, 2, &ClockCycles, &TimeDiffInitalDelay);
        TACheckTimeMatInit(mh, 2);

        StartFrame = (((0) > (mpReparsePoints[0].Deg.Start / mMacroFrameSize)) ? (0) :
(mpReparsePoints[0].Deg.Start / mMacroFrameSize));

    }
    else
    {

    }

    //The initial delay is known now, mpDelayInSamplesPerFrame is filled with it,
    //pActiveFrameFlags contains for each macro frame an indication
    //whether the frame is active or not and StartFrame points to the first
    //really active frame. IterationData contains the window parameters
    //etc. for the following coarse alignment and the schedule for the
    //iterative increase of the delay resolution.
    //mpDelayInSamplesPerFrame must now be refined and ReliabilityPerFrame must be set
    //to the correlation found for each frame.
    //The coarse alignment will shift delay jumps during silence to the beginning of
    //the next active section. This must be corrected later in order to avoid problems
    //when converting deg delays to ref delays. Otherwise it will result in lost or
    //repeated frames.

    CProcessData IterationData = mProcessData;

    if (rc)

    {
        mCAIntermediate.Init(mNumMacroFrames);
        mCAIntermediate.AvgEnergyRef = ERef;
        mCAIntermediate.AvgEnergyDeg = EDeg;

        int Size = mpFeatureList2->GetFVector(0, 0, 0)->mSize;
        int MinSize = (((Size) < (mNumMacroFrames)) ? (Size) : (mNumMacroFrames));
        matbCopy(mpFeatureList2->GetFVector(0, 0, 0)->mpVector,
mCAIntermediate.pRefEnergy, MinSize);
        if (MinSize<mNumMacroFrames)
            matbSet(0.0, mCAIntermediate.pRefEnergy+MinSize, mNumMacroFrames-MinSize);
```

```
        Size = mpFeatureList2->GetFVector(0, 1, 0)->mSize;
        MinSize = (((Size) < (mNumMacroFrames)) ? (Size) : (mNumMacroFrames));
        matbCopy(mpFeatureList2->GetFVector(0, 1, 0)->mpVector,
mCAIntermediate.pDegEnergy, MinSize);
        if (MinSize<mNumMacroFrames)
            matbSet(0.0, mCAIntermediate.pDegEnergy+MinSize, mNumMacroFrames-MinSize);

        rc = CoarseAlignment(&IterationData, pActiveFrameFlags, StartFrame,
pSearchRangePerMacroFrameLow, pSearchRangePerMacroFrameHigh,
mpDelayInSamplesPerFrame, ReliabilityPerFrame, doRevertToOPTprealignment,
pVecs, &NumVecs);

;

        if (rc)
        {

            //Shift delay changes from the start of active sections to the center of
inactive intervalls.
            //To avoid problems with transition effects at the start of the section,
the delay of the second
            //frame of the section is taken.
            ;
            ;
            mProcessData.Init(1, 1.0);
            for (r=1; r<mNumReparsePoints; r++)
            {
                int DegEndSample = mpReparsePoints[r-1].Deg.End;
                int RefEndSample = mpReparsePoints[r-1].Ref.End;

                int DegStartSample = mpReparsePoints[r].Deg.Start;
                int RefStartSample = mpReparsePoints[r].Ref.Start;

                int CheckFrame = SamplesToFrames(DegStartSample);
                if (CheckFrame<mNumMacroFrames-1 && CheckFrame>1)
                {
                    int DelayAfter = mpDelayInSamplesPerFrame[CheckFrame+1];

                    int ChangePosDeg;
                    if (RefStartSample-RefEndSample<DegStartSample-DegEndSample)
                    {
                        ChangePosDeg = (((0) > (RefEndSample +
(int)(0.5*(RefStartSample-RefEndSample)) - DelayAfter)) ? (0) :
(RefEndSample + (int)(0.5*(RefStartSample-RefEndSample)) -
DelayAfter));

                        ;
                    }
                    else
                    {
                        ChangePosDeg = DegEndSample +
(int)(0.5*(DegStartSample-DegEndSample));
                        ;
                    }

                    for (i=SamplesToFrames(ChangePosDeg);
i<SamplesToFrames(DegStartSample)+1 && i<mNumMacroFrames; i++)
                        mpDelayInSamplesPerFrame[i] = DelayAfter;
                }
            }

    if (rc)
    {
        OTA_FLOAT LagToSamples=1;

        long* pDelayInSamplesPerFrameAfterCA = (long*)matMalloc(mNumMacroFrames *
sizeof(long));
        for (i=0; i<mNumMacroFrames; i++)
            pDelayInSamplesPerFrameAfterCA[i] = mpDelayInSamplesPerFrame[i];

        ;
        mProcessData.Init(1, 1.0);

        mFAIntermediate.Init(mNumMacroFrames);
        mFAIntermediate.AvgEnergyRef = ERef;
```

```
        mFAIntermediate.AvgEnergyDeg = EDeg;
        mFAIntermediate.pDelayVec = pDelayInSamplesPerFrameAfterCA;
        mFAIntermediate.pPitchVec = mCAIntermediate.pPitchVec;
        mFAIntermediate.pDegEnergy = mCAIntermediate.pDegEnergy;
        mFAIntermediate.pRefEnergy = mCAIntermediate.pRefEnergy;
        mFAIntermediate.pActiveFrameFlags = mCAIntermediate.pActiveFrameFlags;
        mFAIntermediate.pReliabilityPerFrame = ReliabilityPerFrame;
        mFAIntermediate.pSearchRangeLow = pSearchRangePerMacroFrameLow;
        mFAIntermediate.pSearchRangeHigh = pSearchRangePerMacroFrameHigh;
        mFAIntermediate.pConstDelayMarker = mCAIntermediate.pConstDelayMarker;

        if (Control & 0x1)
        {
            ;
            bool IsSpecialAlignment =
mProcessData.mDelayFineAlignCorrlen!=mProcessData.mSRDetectFineAlignCorrlen
;

            mpDelaySearch->FineAlign(&mFAIntermediate, mppSignals,
mpActiveFrameDetection, pDelayInSamplesPerFrameAfterCA,
ReliabilityPerFrame, &mNumMacroFrames, mProcessData.mStepSize,
2*IterationData.mWindowSize, mProcessData.mSRDetectFineAlignCorrlen,
IsSpecialAlignment?FA_FOR_SRDETECTION:0);

            LagToSamples = 1;

            OTA_FLOAT* partialSRperFrame = (OTA_FLOAT*)matCalloc(mNumMacroFrames,
sizeof(OTA_FLOAT));
            int numPartialSR = -1;
            OTA_FLOAT RelativeSamplerateDifference_linear =
GetSampleRateRatio_linear(mFAIntermediate.pActiveFrameFlags,pDelayInSamples
PerFrameAfterCA, mNumMacroFrames, mProcessData.mStepSize,
(int)LagToSamples, EnablePlotting, mFAIntermediate.pUsedForSRDet,
partialSRperFrame, &numPartialSR);
            RelativeSamplerateDifference = RelativeSamplerateDifference_linear;
            matFree(partialSRperFrame);

            ;

        }

        if (Control & 0x2 && mProcessData.mDelayFineAlignCorrlen>0
            &&
(RelativeSamplerateDifference<1+mProcessData.mMaxToleratedRelativeSamplerat
eDifference
            &&
RelativeSamplerateDifference>1-mProcessData.mMaxToleratedRelativeSamplerate
Difference
            || RelativeSamplerateDifference==-1 ))
        {
            if
(mProcessData.mDelayFineAlignCorrlen==mProcessData.mSRDetectFineAlignCorrle
n)
            {
                for (int i=0; i<mNumMacroFrames; i++)
                    mpDelayInSamplesPerFrame[i] = pDelayInSamplesPerFrameAfterCA[i];
            }
            else
            {
                ;

                mpDelaySearch->FineAlign(&mFAIntermediate, mppSignals,
mpActiveFrameDetection, mpDelayInSamplesPerFrame, ReliabilityPerFrame,
&mNumMacroFrames, mProcessData.mStepSize, 2*IterationData.mWindowSize,
mProcessData.mSRDetectFineAlignCorrlen);

            }
            LagToSamples = 1;
        }

        TACheckTimeMatEval(mh, 2, &ClockCycles, &TimeDiffFineAlignment);
        TACheckTimeMatInit(mh, 2);

        OTA_FLOAT avgReliability  = (OTA_FLOAT)0.0;
        int   numActiveFrames = 0;
        for (i=0; i<mNumMacroFrames; i++)
```

```
        {
            mpReliabilityPerFrame[i] = ReliabilityPerFrame[i];
            if (pActiveFrameFlags[i])
            {
                avgReliability += ReliabilityPerFrame[i];
                numActiveFrames++;
            }
        }
        avgReliability /= (((numActiveFrames) > (1)) ? (numActiveFrames) : (1));
        if (mpResults)
            mpResults->mAvgReliability = avgReliability;

        ;

        mProcessData.Init(1, 1.0);

        if (pResult)
        {
            if(mpResults)
            {
                mpResults->mNumUtterances = 0;
                mpResults->mpDelayUtterance = 0;
                mpResults->mpStartSampleUtterance = 0;
                mpResults->mpStopSampleUtterance = 0;

                mpResults->mNumFrames = mNumMacroFrames;

                DetermineInvalidSections();

                if (Control & 0x4)
                    CreateUtteranceVectorsRef(&mpResults->mNumUtterances,
&mpResults->mpStartSampleUtterance,
&mpResults->mpStopSampleUtterance, &mpResults->mpDelayUtterance,
mpReparsePoints, mNumReparsePoints);
                else if (Control & 0x8)
                    CreateUtteranceVectorsDeg(&mpResults->mNumUtterances,
&mpResults->mpStartSampleUtterance,
&mpResults->mpStopSampleUtterance, &mpResults->mpDelayUtterance,
mpReparsePoints, mNumReparsePoints);

                if (mStartOffset)
                {
                    if (Control & 0x4)
                    {
                        for (i=0; i<mpResults->mNumUtterances; i++)
                            mpResults->mpDelayUtterance[i] -= mStartOffset;
                        if (mStartOffset>0)
                        {
                            for (i=0; i<mpResults->mNumUtterances; i++)
                            {
                                mpResults->mpStartSampleUtterance[i] += mStartOffset;
                                mpResults->mpStopSampleUtterance[i] += mStartOffset;
                            }
                            for (i=0; i<mNumReparsePoints; i++)
                            {
                                mpReparsePoints[i].Deg.Start -= mStartOffset;
                                mpReparsePoints[i].Deg.End -= mStartOffset;
                            }
                        }
                        else
                        {
                            for (i=0; i<mNumReparsePoints; i++)
                            {
                                mpReparsePoints[i].Deg.Start -= mStartOffset;
                                mpReparsePoints[i].Deg.End -= mStartOffset;
                            }
                        }
                    }
                    else if (Control & 0x8)
                    {
                        for (i=0; i<mpResults->mNumUtterances; i++)
                            mpResults->mpDelayUtterance[i] += mStartOffset;
                        if (mStartOffset<0)
                        {
                            for (i=0; i<mpResults->mNumUtterances; i++)
                            {
```

```
                            mpResults->mpStartSampleUtterance[i] -= mStartOffset;
                            mpResults->mpStopSampleUtterance[i] -= mStartOffset;
                        }

                        for (i=0; i<mNumReparsePoints; i++)
                        {
                            mpReparsePoints[i].Deg.Start -= mStartOffset;
                            mpReparsePoints[i].Deg.End -= mStartOffset;
                        }
                    }
                    else
                    {
                        for (i=0; i<mNumReparsePoints; i++)
                        {
                            mpReparsePoints[i].Ref.Start += mStartOffset;
                            mpReparsePoints[i].Ref.End += mStartOffset;
                        }
                    }
                }

                if (mStartOffset<0)
                {
                    int StartFrame = (-mStartOffset) / mProcessData.mStepSize;
                    for (i=mNumMacroFrames-1; i>=0; i--)
                    {
                        mpDelayInSamplesPerFrame[i+StartFrame] =
mpDelayInSamplesPerFrame[i] + mStartOffset;
                        mpReliabilityPerFrame[i+StartFrame] =
mpReliabilityPerFrame[i];
                        pActiveFrameFlags[i+StartFrame] = pActiveFrameFlags[i];
                    }
                    for (i=0; i<StartFrame; i++)
                    {
                        mpDelayInSamplesPerFrame[i] =
mpDelayInSamplesPerFrame[StartFrame];
                        mpReliabilityPerFrame[i] =
mpReliabilityPerFrame[StartFrame];
                        pActiveFrameFlags[i] = pActiveFrameFlags[StartFrame];
                    }
                    mNumMacroFrames += StartFrame;
                    mpResults->mNumFrames = mNumMacroFrames;
                }
                else
                {
                    for (i=0; i<mNumMacroFrames; i++)
                        mpDelayInSamplesPerFrame[i] += mStartOffset;
                }

            }

            if (Control & 0x1)
                mpResults->mRelSamplerateDev = RelativeSamplerateDifference;
            else
                mpResults->mRelSamplerateDev = 1.0;
            mpResults->mResolutionInSamples = mProcessData.mStepSize;
            mpResults->mStepsize = mProcessData.mStepSize;

            mpResults->mpRefSections = new SECTION[mNumReparsePoints];
            mpResults->mpDegSections = new SECTION[mNumReparsePoints];
            for (i=0; i<mNumReparsePoints; i++)
            {
                mpResults->mpRefSections[i] = mpReparsePoints[i].Ref;
                mpResults->mpDegSections[i] = mpReparsePoints[i].Deg;
            }
            mpResults->mNumSections = mNumReparsePoints;

            OTA_FLOAT tempDegNoiseLevel, tempDegSignalLevel, tempRefNoiseLevel,
tempRefSignalLevel;
            OTA_FLOAT tempDegNoiseThreshold,tempRefNoiseThreshold;
            mpActiveFrameDetection->GetLevels(0, 0, 1, &tempRefNoiseLevel,
&tempRefSignalLevel, &tempRefNoiseThreshold);
            mpActiveFrameDetection->GetLevels(1, 0, 1, &tempDegNoiseLevel,
&tempDegSignalLevel, &tempDegNoiseThreshold);

            mpResults->mAslFrames = pResult->mAslFrames;
            mpResults->mAslFramelength = pResult->mAslFramelength;
```

```
                mpResults->mpAslActiveFrameFlags =
(int*)matMalloc(mpResults->mAslFrames*sizeof(int));
                mpResults->mAslFrames = mpActiveFrameDetection->GetActiveFrameFlags(0,
0, mpResults->mAslFramelength, mpResults->mpAslActiveFrameFlags,
mpResults->mAslFrames, 0);

                mpResults->mSNRRefdB = 10*log10(tempRefSignalLevel/tempRefNoiseLevel);
                mpResults->mSNRDegdB = 10*log10(tempDegSignalLevel/tempDegNoiseLevel);
                mpResults->mNoiseLevelRef = tempRefNoiseLevel;
                mpResults->mNoiseLevelDeg = tempDegNoiseLevel;
                mpResults->mSignalLevelRef = tempRefSignalLevel;
                mpResults->mSignalLevelDeg = tempDegSignalLevel;
                mpResults->mNoiseThresholdRef = tempRefNoiseThreshold;
                mpResults->mNoiseThresholdDeg = tempDegNoiseThreshold;

                mpResults->mpDelay            = (long*)matMalloc(mNumMacroFrames *
sizeof(long));
                mpResults->mpReliability      = (OTA_FLOAT*)matMalloc(mNumMacroFrames *
sizeof(OTA_FLOAT));
                mpResults->mpActiveFrameFlags = (int*)matMalloc(mNumMacroFrames *
sizeof(int));
                for (i=0; i<mNumMacroFrames; i++)
                {
                    mpResults->mpDelay           [i] = mpDelayInSamplesPerFrame[i];
                }
                matbCopy(pActiveFrameFlags, mpResults->mpActiveFrameFlags,
mNumMacroFrames);
                matbCopy(mpReliabilityPerFrame, mpResults->mpReliability,
mNumMacroFrames);
            }
            else rc = false;
        }
        if(pDelayInSamplesPerFrameAfterCA)
            matFree(pDelayInSamplesPerFrameAfterCA);
    }
    else
    {

    }

    if(ReliabilityPerFrame)
        matFree(ReliabilityPerFrame);
    if(pActiveFrameFlags)
        matFree(pActiveFrameFlags);
    if(pSearchRangePerMacroFrameLow)
        matFree(pSearchRangePerMacroFrameLow);
    if(pSearchRangePerMacroFrameHigh)
        matFree(pSearchRangePerMacroFrameHigh);

    TACheckTimeMatEval(mh, 2, &ClockCycles, &TimeDiffCleanup);

    if (mpResults)
    {
        mpResults->mTimeDiffs[0] = 0;
        mpResults->mTimeDiffs[1] = TimeDiffInitalDelay;
        mpResults->mTimeDiffs[2] = TimeDiffCoarseAlignment;
        mpResults->mTimeDiffs[3] = TimeDiffFineAlignment;
        mpResults->mTimeDiffs[4] = TimeDiffCleanup;
    }

    if (pResult && mpResults)
        pResult->CopyFrom(mpResults);

    return rc;
}

int CTempAlignment::GetPitchFrameSize()
{
    return mpResults->mPitchFrameSize;
}

void inline CombineFirstTwoUtterancesRef(int NumUtterancesLeft, int* pStartUtt, int*
pStopUtt, int* pDelayUtt, bool* pIsInsideActiveSection)
{
    pStopUtt[0] = pStopUtt[1];
    for (int i=1; i<NumUtterancesLeft-1; i++)
```

```
    {
        pStartUtt[i] = pStartUtt[i+1];
        pStopUtt[i] = pStopUtt[i+1];
        pDelayUtt[i] = pDelayUtt[i+1];
        pIsInsideActiveSection[i] = pIsInsideActiveSection[i+1];
    }
}

void inline CombineFirstTwoUtterancesDeg(int NumUtterancesLeft, int* pStartUtt, int*
pStopUtt, int* pDelayUtt)
{
    pStopUtt[0] = pStopUtt[1];
    for (int i=1; i<NumUtterancesLeft-1; i++)
    {
        pStartUtt[i] = pStartUtt[i+1];
        pStopUtt[i] = pStopUtt[i+1];
        pDelayUtt[i] = pDelayUtt[i+1];
    }
}

//Create utterance vectors and Set the utterance information.
//This requires "reversing" the delay information since the calculated delay is the
delay
//of the reference signal, but we need the delay of the degraded signal.
//The three vectors must be destroyed by the calling routine!
void CTempAlignment::CreateUtteranceVectorsRef(int*  pNumUtterances, int**
ppStartSampleUtterance, int** ppStopSampleUtterance, int** ppDelayUtterance,
REPARSE_POINT* ReparsePoints, int NumReparsePoints)
{

}

//Create utterance vectors and Set the utterance information.
//This version does NOT reverse the delay information!.
//The three vectors are allocated here, but must be destroyed by the calling routine!
void CTempAlignment::CreateUtteranceVectorsDeg(int*  pNumUtterances, int**
ppStartSampleUtterance, int** ppStopSampleUtterance, int** ppDelayUtterance,
REPARSE_POINT* ReparsePoints, int NumReparsePoints)
{
    int fdeg;

    int NumberOfUtterances = 1;
    for (fdeg=1; fdeg<mNumMacroFrames; fdeg++)
        if (mpDelayInSamplesPerFrame[fdeg] != mpDelayInSamplesPerFrame[fdeg-1])
            NumberOfUtterances++;
    ;

    int* pStartSampleUtterance = (int*)matMalloc((NumberOfUtterances+5) * sizeof(int));
    int* pStopSampleUtterance = (int*)matMalloc((NumberOfUtterances+5) * sizeof(int));

    int* pDelayUtterance = (int*)matMalloc((NumberOfUtterances+5) * sizeof(int));

    int utt=0;

    NumberOfUtterances = 0;
    pStartSampleUtterance[0] = 0;
    pDelayUtterance[0] = mpDelayInSamplesPerFrame[0];
    for (fdeg=1; fdeg<mNumMacroFrames; fdeg++)
    {
        if (mpDelayInSamplesPerFrame[fdeg] != mpDelayInSamplesPerFrame[fdeg-1])
        {
            pStopSampleUtterance[NumberOfUtterances] = fdeg*mProcessData.mStepSize - 1;
            NumberOfUtterances++;
            pStartSampleUtterance[NumberOfUtterances] = fdeg*mProcessData.mStepSize;
            pDelayUtterance[NumberOfUtterances] = mpDelayInSamplesPerFrame[fdeg];
        }
    }
    pStopSampleUtterance[NumberOfUtterances] = fdeg*mProcessData.mStepSize - 1;
    NumberOfUtterances++;

    ;

    int LastUsedReparsePoint=0;
    for (utt=0; utt<NumberOfUtterances-1; utt++)
    {
        if (pStartSampleUtterance[utt+1]-pStopSampleUtterance[utt] <
```

```
0.1*mProcessData.mSamplerate)
        {
            bool IsInsideActiveSection = true;
            while(LastUsedReparsePoint<NumReparsePoints-1 &&
pStartSampleUtterance[utt+1]>ReparsePoints[LastUsedReparsePoint].Deg.End)
                LastUsedReparsePoint++;
            if
(pStartSampleUtterance[utt+1]<ReparsePoints[LastUsedReparsePoint].Deg.Start
)
                IsInsideActiveSection = false;

            if ((!IsInsideActiveSection &&
                    abs(pDelayUtterance[utt]-pDelayUtterance[utt+1])<0.015*mProcessData
.mSamplerate)
                ||
abs(pDelayUtterance[utt]-pDelayUtterance[utt+1])<0.0003*mProcessData.mS
amplerate
                )

            {
                int Delay;
                int Len1 = pStopSampleUtterance[utt] - pStartSampleUtterance[utt];
                int Len2 = pStopSampleUtterance[utt+1] - pStartSampleUtterance[utt+1];
                if (Len2>Len1)
                    Delay = pDelayUtterance[utt+1];
                else
                    Delay = pDelayUtterance[utt];

                CombineFirstTwoUtterancesDeg(NumberOfUtterances-utt,
pStartSampleUtterance+utt, pStopSampleUtterance+utt,
pDelayUtterance+utt);
                pDelayUtterance[utt] = Delay;

                NumberOfUtterances--;
                utt--;
            }
        }
    }
    ;

    SECTION SecA, SecB;
    int MaxLagSamples = (int)(0.001F*(float)mProcessData.mSamplerate);
    int NumSamples = mNumMacroFrames*mProcessData.mStepSize;
    CFeatureVector RefSig;
    mppSignals[0]->GetAsFeatureVector(&RefSig, 0);
    CFeatureVector DegSig;
    mppSignals[1]->GetAsFeatureVector(&DegSig, 0);
    for (utt=0; utt<NumberOfUtterances; utt++)
    {
        SecA.Start = pStartSampleUtterance[utt];
        SecA.End = pStopSampleUtterance[utt];
        SecB.Start = pStartSampleUtterance[utt]-pDelayUtterance[utt]-MaxLagSamples;
        SecB.End = pStopSampleUtterance[utt]-pDelayUtterance[utt]+MaxLagSamples;

        if (SecA.Len()<0.5*(float)mProcessData.mSamplerate)
            continue;
        if (SecA.Start<0 || SecA.End>RefSig.mSize)
            continue;
        if (SecB.Start<0 || SecB.End>DegSig.mSize)
            continue;
        int Delay = FindSectionAInSectionB(&SecA, &SecB, &RefSig, &DegSig, 0, 1, 1,
2*MaxLagSamples);
        Delay = -(Delay -SecA.Start +SecB.Start);
        if (Delay!=pDelayUtterance[utt])
        {
            ;
            pDelayUtterance[utt] = Delay;
        }
    }

    *pNumUtterances = NumberOfUtterances;
    *ppStartSampleUtterance = pStartSampleUtterance;
    *ppStopSampleUtterance = pStopSampleUtterance;
    *ppDelayUtterance = pDelayUtterance;
}
```

```
}
```