```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

using std::string;

SegListTraversal::SegListTraversal(SQTimeAlignment   const *timeAlignmentInfo,
                                   SQTA_ResampResult const *resamplingInfo,
                                   SQSignal const *refSig, SQSignal const *degSig,
                                   int frameSize, int frameStep)
: mTAinfo               (NULL),
  mResampInfo           (NULL),
  mDetailedSegList      (NULL),
  mSegList              (NULL),
  mUnusedDegSegments    (NULL),
  mRef                  (NULL),
  mDeg                  (NULL),
  mSignalPosRef         (-1),
  mSignalPosDeg         (-1),
  mTASearch_bestFrmPos  (-1),
  mJumpedToNewSeg       (false),
  mSegType              (TA_SEG_PAUSE),
  mRefMissingInDeg      (false),
  mDegMissingInRef      (false)
{
    OPTTRY
    {
        if (timeAlignmentInfo == NULL || resamplingInfo == NULL || refSig == NULL ||
            degSig == NULL || frameSize < 2 || frameStep < 1 || frameStep > frameSize ||
            refSig->SamplingFreq() != degSig->SamplingFreq())
            OPTTHROW ((string("ERROR in SegListTraversal: Invalid constructor
arguments.\n")));

        if (timeAlignmentInfo->MergedSegments() == NULL ||
            timeAlignmentInfo->UnusedDegSegments() == NULL ||
            timeAlignmentInfo->MergedSegments()->size() < 1)
            OPTTHROW ((string("ERROR in SegListTraversal: SQTimeAlignment object
contains invalid data.\n")));

        sqStorage = new SQStorage();

        //Init time alignment member vars
        mTAinfo            = timeAlignmentInfo;
        mResampInfo        = resamplingInfo;
        mDetailedSegList   = timeAlignmentInfo->Segments();
        mSegList           = new TA_SegList(*timeAlignmentInfo->MergedSegments());
        mUnusedDegSegments = new TA_SegList(*timeAlignmentInfo->UnusedDegSegments());

        //Init signal and frame member vars
        mRef = refSig;
        mDeg = degSig;
        mFrameSize = frameSize;
        mFrameStep = frameStep;

        //Determine the beginning and end positions for time alignment traversal

        mStartRefPos = 0;
        mEndRefPos   = mRef->NrOfSamples()-1;

        PreprocessSegList();
        FindStartingSegments();
    }
    OPTCATCH( (...))
    {
        delete mSegList;
        delete mUnusedDegSegments;
        sqStorage->ClearAllItemsWhoseIDStartsWith("SegListTraversal_");
        OPTTHROW ((string("ERROR in SegListTraversal constructor.\n")));
    }
}
SegListTraversal::~SegListTraversal()
{
    delete mSegList;
    delete mUnusedDegSegments;
    sqStorage->ClearAllItemsWhoseIDStartsWith("SegListTraversal_");
```

```
    delete sqStorage;

    sqStorage = 0;
    mSegList = mUnusedDegSegments = NULL;
}

bool SegListTraversal::JumpedToNewSeg() const
{
    return mJumpedToNewSeg;
}

int SegListTraversal::SignalPosRef() const
{
    return mFinalSignalPosRef;
}

int SegListTraversal::SignalPosDeg() const
{
    return mFinalSignalPosDeg;
}

XFLOAT SegListTraversal::SegReliability() const
{
    return mSegReliability;
}

TA_SEG_TYPE SegListTraversal::SegType() const
{
    return mSegType;
}

bool SegListTraversal::RefMissingInDeg() const
{
    return mRefMissingInDeg;
}

bool SegListTraversal::DegMissingInRef() const
{
    return mDegMissingInRef;
}

XFLOAT SegListTraversal::OverallMatchQuality() const
{
    return mTAinfo->MatchQuality();
}

XFLOAT SegListTraversal::ResamplingFac() const
{
    return mResampInfo->fMeanResamplingFac;
}

void SegListTraversal::MoveToNextFramePair()
{
    if (!mRefMissingInDeg)
        mSignalPosDeg += mFrameStep;

    else
        mSignalPosRef += mFrameStep;
}

int SegListTraversal::FullTraversal(int &curRefPos,    int &curDegPos,
                                    int &minSearchPos, int &maxSearchPos)
{
    OPTTRY
    {
        int const minContFrmLen
            = (int)(mFrameSize - mFrameStep/2);

        //Have we reached the end of the current segment? -> find the next valid one.
        while (mCurSegs[mActiveList] != mEndOfLists[mActiveList] &&

               (SkipThisMissingSegment(mCurSegs[mActiveList], mActiveList) ||
                mSignalPosDeg + minContFrmLen >
                 mCurSegs[mActiveList]->degPos + mCurSegs[mActiveList]->segLen ||
                (mCurSegs[mActiveList]->segLen < mFrameSize &&
                 mCurSegs[mActiveList]->segType != TA_SEG_MISSING) ||
```

```
                (mRefMissingInDeg &&

                 mSignalPosRef + minContFrmLen > mCurSegs[mActiveList]->refPos +
mCurSegs[mActiveList]->segLen)))
            mCurSegs[mActiveList]++;

        //Leave if both iterators are the list ends or beyond the active signal.
        if ((mCurSegs[UNUSED_DEG_LIST] == mEndOfLists[UNUSED_DEG_LIST] ||
             mSignalPosDeg + minContFrmLen >
              mEndRefPos + mCurSegs[UNUSED_DEG_LIST]->degPos -
mCurSegs[UNUSED_DEG_LIST]->refPos)
             &&
            (mCurSegs[SEGLIST] == mEndOfLists[SEGLIST] ||
             mSignalPosDeg + minContFrmLen >
              mEndRefPos + mCurSegs[SEGLIST]->degPos - mCurSegs[SEGLIST]->refPos))
            return LEAVE_LOOP;

        //Select which segment of the two lists to use.
        while (mCurSegs[SEGLIST] != mEndOfLists[SEGLIST] &&
               mCurSegs[UNUSED_DEG_LIST] != mEndOfLists[UNUSED_DEG_LIST])
        {
            if (mCurSegs[SEGLIST]->degPos >= mCurSegs[UNUSED_DEG_LIST]->degPos)
                if (SkipThisMissingSegment(mCurSegs[UNUSED_DEG_LIST], UNUSED_DEG_LIST)
||
                    mCurSegs[UNUSED_DEG_LIST]->degPos +
mCurSegs[UNUSED_DEG_LIST]->segLen <= mSignalPosDeg)
                    mCurSegs[UNUSED_DEG_LIST]++;
                else
                    { mActiveList = UNUSED_DEG_LIST; break; }
            else
                if (SkipThisMissingSegment(mCurSegs[SEGLIST], SEGLIST) ||
                    (mCurSegs[SEGLIST]->segLen  <  mFrameSize &&
                     mCurSegs[SEGLIST]->segType != TA_SEG_MISSING) ||
                    mCurSegs[SEGLIST]->degPos + mCurSegs[SEGLIST]->segLen <=
mSignalPosDeg)
                    mCurSegs[SEGLIST]++;
                else
                    { mActiveList = SEGLIST; break; }
        }

        if (mCurSegs[mActiveList] == mEndOfLists[mActiveList])
        {
            mActiveList = (mActiveList+1)%2;
            while (mCurSegs[mActiveList] != mEndOfLists[mActiveList] &&
                   (SkipThisMissingSegment(mCurSegs[mActiveList], mActiveList) ||
                    (mCurSegs[mActiveList]->segLen  <  mFrameSize &&
                     mCurSegs[mActiveList]->segType != TA_SEG_MISSING)))
                mCurSegs[mActiveList]++;
            if (mCurSegs[mActiveList] == mEndOfLists[mActiveList])
                return LEAVE_LOOP;
        }

        //Get current position in the deg signal, and associated segment info.
        if (mSignalPosDeg < 0)
            if (mActiveList == SEGLIST)
                mSignalPosDeg = mStartRefPos + mCurSegs[mActiveList]->degPos -
mCurSegs[mActiveList]->refPos;
            else
                mSignalPosDeg = mCurSegs[mActiveList]->degPos;

        mSegType = mCurSegs[mActiveList]->segType;
        if ((mSegType == TA_SEG_MISSING && mActiveList == SEGLIST) &&
            !mRefMissingInDeg)
        {
            mSignalPosRef = mCurSegs[mActiveList]->refPos;
            mTASearch_bestFrmPos = -100000;
        }
        else if (!(mSegType == TA_SEG_MISSING && mActiveList == SEGLIST) &&
                  mRefMissingInDeg)
            mTASearch_bestFrmPos = -100000;

        mRefMissingInDeg = mSegType == TA_SEG_MISSING && mActiveList == SEGLIST;
        mDegMissingInRef = mActiveList == UNUSED_DEG_LIST;

        if (mSignalPosDeg < mCurSegs[mActiveList]->degPos)
        {
```

```
            mSignalPosDeg   = mCurSegs[mActiveList]->degPos;
            mJumpedToNewSeg = true;
        }
        else
            mJumpedToNewSeg = false;

        //Handle segments with imprecise matching of ref and deg signal,
        //e.g. missing or additional utterances, or matching based on guess.
        bool handledRefFrame = false, handledDegFrame = false;
        minSearchPos = maxSearchPos = -1;
        if (mSegType == TA_SEG_MISSING)
            HandleMissingSegment(mTASearch_bestFrmPos,
                                 minSearchPos, maxSearchPos,
                                 handledRefFrame, handledDegFrame);
        else if (mSegType != TA_SEG_MATCHED)
            HandleGuessedSegment(mTASearch_bestFrmPos,
                                 minSearchPos, maxSearchPos,
                                 handledRefFrame, handledDegFrame);
        else
            mSegReliability = 1.0f;

        if (!handledRefFrame && !mRefMissingInDeg)
            mSignalPosRef = mSignalPosDeg -
                (mCurSegs[mActiveList]->degPos - mCurSegs[mActiveList]->refPos);

        //Now determine the remaining ref/deg frame position, depending
        //on the value of the handledRefFrame/handledDegFrame flags.

        //Reached last frame of current segment -> allow a smaller
        //frame step than mFrameStep for the last frame.
        if (mActiveList == SEGLIST && !handledRefFrame &&
            mSignalPosRef + mFrameSize > mCurSegs[mActiveList]->refPos +
mCurSegs[mActiveList]->segLen)
        {
            int lastFramePos = max(mCurSegs[mActiveList]->refPos +
mCurSegs[mActiveList]->segLen
                                   - mFrameSize, 0);

            handledRefFrame = true;
            mFinalSignalPosRef = lastFramePos;
            if (!handledDegFrame)
            {
                int curShift = limit(mCurSegs[mActiveList]->degPos -
mCurSegs[mActiveList]->refPos,
                                     0 - lastFramePos,
(int)mDeg->NrOfSamples()-lastFramePos-mFrameSize);
                handledDegFrame = true;
                mFinalSignalPosDeg = lastFramePos + curShift;
            }

            mCurSegs[mActiveList]++;
        }
        else if (mActiveList == UNUSED_DEG_LIST &&
                 mSignalPosDeg + mFrameSize > mCurSegs[mActiveList]->degPos +
mCurSegs[mActiveList]->segLen)
        {
            int lastFramePos = max(mCurSegs[mActiveList]->degPos +
mCurSegs[mActiveList]->segLen
                                   - mFrameSize, 0);

            if (!handledDegFrame)
            {
                handledDegFrame = true;
                mFinalSignalPosDeg = lastFramePos;
            }

            mCurSegs[mActiveList]++;

            if (!handledRefFrame)
                OPTTHROW(( string("HandleMissingSegment did not copy the ref
segment!")));
        }
        else //'standard case'
        {
            if (mActiveList == SEGLIST)
            {
```

```
                if (!handledRefFrame)
                {
                    handledRefFrame = true;
                    mFinalSignalPosRef = mSignalPosRef;
                }
                if (!handledDegFrame &&
                    mSignalPosDeg + mFrameSize > mCurSegs[mActiveList]->degPos +
mCurSegs[mActiveList]->segLen)
                {
                    int lastFramePos = max(mCurSegs[mActiveList]->degPos +
mCurSegs[mActiveList]->segLen
                                           - mFrameSize, 0);
                    handledDegFrame = true;
                    mFinalSignalPosDeg = lastFramePos;
                    mCurSegs[mActiveList]++;
                }
                else if (!handledDegFrame)
                {
                    handledDegFrame = true;
                    mFinalSignalPosDeg = mSignalPosDeg;
                }

            }
            else if (!handledDegFrame)
            {
                handledDegFrame = true;
                mFinalSignalPosDeg = mSignalPosDeg;
            }
        }

        curRefPos = mFinalSignalPosRef;
        curDegPos = mFinalSignalPosDeg;
        if (minSearchPos < 0)
        {
            if (mSegType == TA_SEG_MISSING)
                OPTTHROW(( string("Min search position not set inside
HandleMissingSegment(). This should never happen!")));
            minSearchPos = mFinalSignalPosRef;
        }
        if (maxSearchPos < 0)
        {
            if (mSegType == TA_SEG_MISSING)
                OPTTHROW(( string("Max search position not set
insidTHROW((dleMissingSegment(). This should never happen!")));
            maxSearchPos = mFinalSignalPosRef;
        }

        return SQ_NO_ERRORS;
    }

    OPTCATCH((string errorMsg))
    {
        OPTTHROW(( string("ERROR in FullTraversal: " + errorMsg + "\n")));
    }
}

void SegListTraversal::PreprocessSegList()
{
    if (mSegList == NULL || mSegList->size() == 0 || mFrameSize < 2 ||
        mDeg == NULL || mDeg->NrOfSamples() < mFrameSize)
        OPTTHROW(( string("ERROR in PreprocessSegList: Invalid segList, frame size or
deg signal.\n")));

    //Set the most probable deg frame starting position for all ref frames missing in
the deg signal
    for (int i = 0; i < (int)mSegList->size(); i++)
    {
        if ((*mSegList)[i].segType == TA_SEG_MISSING)
        {
            (*mSegList)[i].degPos += (*mSegList)[i].segLen/2 - mFrameSize/2;
            (*mSegList)[i].degPos =  limit((*mSegList)[i].degPos, 0,
(int)mDeg->NrOfSamples()-mFrameSize);
        }
    }
}
```

```
void SegListTraversal::FindStartingSegments()
{
    //Position ourselves at the beginning of the segments lists
    mCurSegs   [SEGLIST]         = mSegList->begin();
    mCurSegs   [UNUSED_DEG_LIST] = mUnusedDegSegments->begin();
    mEndOfLists[SEGLIST]         = mSegList->end();
    mEndOfLists[UNUSED_DEG_LIST] = mUnusedDegSegments->end();
    mActiveList                  = SEGLIST;

    //Find starting segment
    while(mCurSegs[SEGLIST] != mEndOfLists[SEGLIST] &&
          ((mCurSegs[SEGLIST]->segType != TA_SEG_MISSING &&
            mCurSegs[SEGLIST]->segLen   < mFrameSize) ||
           mCurSegs[SEGLIST]->refPos + mCurSegs[SEGLIST]->segLen <= mStartRefPos))
        mCurSegs[SEGLIST]++;

    //Find first valid segment
    while(mCurSegs[UNUSED_DEG_LIST] != mEndOfLists[UNUSED_DEG_LIST] &&
          mCurSegs[UNUSED_DEG_LIST]->refPos + mCurSegs[UNUSED_DEG_LIST]->segLen <=
mStartRefPos)
        mCurSegs[UNUSED_DEG_LIST]++;
}

struct SkipThisMissingSegmentStruct:SQStorageSkeletonClass
{
    int minSegLenRef;
    int minSegLenDeg;
    int firstSpeechActRef;
    int firstSpeechActDeg;
    TA_SegList::const_iterator prevSegRef;
    TA_SegList::const_iterator prevSegDeg;

    SkipThisMissingSegmentStruct(int argNum, va_list *argList)
    {
        if (argNum != 0)
            OPTTHROW(( string("ERROR in SkipThisMissingSegmentStrTHROW((onstructor:
Invalid arguments.\n")));

        minSegLenRef = 0;
        minSegLenDeg = 0;

        firstSpeechActRef = -1;
        firstSpeechActDeg = -1;
    }
};
bool SegListTraversal::SkipThisMissingSegment(TA_SegList::iterator const seg, int
listIdx)
{
    int const STMS_SEGLEN_TOLERANCE =
        min(round(0.005f * mRef->SamplingFreq()), mFrameSize);
    int const STMS_MIN_PAUSE_LEN    =
        round(0.3f*mRef->SamplingFreq());
    int const STMS_MIN_SEGLEN       =
        min(round(20e-3f*mRef->SamplingFreq()), mFrameSize);

    SkipThisMissingSegmentStruct *varsToStore = NULL;
    sqStorage->GetOrStore("SegListTraversal_SkipThisMissingSegmentStruct", &varsToStore,
0);

    if (varsToStore->firstSpeechActDeg < 0)
    {
        varsToStore->prevSegRef = mSegList->end();
        varsToStore->prevSegDeg = mUnusedDegSegments->end();

        if (mCurSegs[mActiveList]->segType == TA_SEG_GUESSED ||
            mCurSegs[mActiveList]->segType == TA_SEG_MATCHED)
        {
            varsToStore->firstSpeechActRef = mCurSegs[mActiveList]->refPos;
            varsToStore->firstSpeechActDeg = mCurSegs[mActiveList]->degPos;
        }
    }

    //Reset counters between sentences
    if (varsToStore->firstSpeechActDeg >= 0 &&
        seg->segType == TA_SEG_PAUSE && seg->segLen >= STMS_MIN_PAUSE_LEN)
    {
```

```
        varsToStore->firstSpeechActRef = seg->refPos + seg->segLen;
        varsToStore->firstSpeechActDeg = seg->degPos + seg->segLen;
        varsToStore->prevSegRef        = mSegList->end();
        varsToStore->prevSegDeg        = mUnusedDegSegments->end();
        varsToStore->minSegLenRef = varsToStore->minSegLenDeg = 0;
    }

    if (seg->segType != TA_SEG_MISSING)
        return false;

    if (seg->segLen < STMS_MIN_SEGLEN) //Segment too short to matter at all.
        return true;

    return false;
}

void SegListTraversal::HandleMissingSegment(int &TASearch_bestFrmPos,
                                            int &minSearchPos, int &maxSearchPos,
                                            bool &handledRefFrame, bool
&handledDegFrame)
{
    if (mCurSegs[mActiveList] == mEndOfLists[mActiveList] ||
        (handledRefFrame && handledDegFrame) || mSegType != TA_SEG_MISSING)
        OPTTHROW(( string("ERROR in HandleMissingSegment: Invalid input
arguments.\n")));

    mSegReliability = 0.0f; //signal parts not occurring in the other signal always have
reliability 0.

    if (mRefMissingInDeg)
    {
        if ((mCurSegs[SEGLIST]+1) != mEndOfLists[SEGLIST]) //haven't reached the end of
the deg signal
        {
            int TASearch_curFrmPos;

            TASearch_curFrmPos  = max(mCurSegs[SEGLIST]->degPos + mFrameSize/2 -
mFrameSize, TASearch_bestFrmPos);
            TASearch_curFrmPos  = limit(TASearch_curFrmPos, 0,
(int)mDeg->NrOfSamples()-mFrameSize);
            TASearch_bestFrmPos = TASearch_curFrmPos; //init to first tried position

            minSearchPos       = TASearch_curFrmPos;
            maxSearchPos       = min((int)mDeg->NrOfSamples()-mFrameSize,
                                     (mCurSegs[SEGLIST]+1)->degPos);
            mFinalSignalPosDeg = mCurSegs[SEGLIST]->degPos;
            handledDegFrame = true;
        }
        else //Ref segment with signal that doesn't occur in deg, at the end of the deg
signal:
        {
            int degSigPos;
            degSigPos = max(mCurSegs[mActiveList]->degPos, TASearch_bestFrmPos);
            degSigPos = max(degSigPos, mSignalPosDeg);
            degSigPos = min(degSigPos, (int)mDeg->NrOfSamples() - mFrameSize);

            minSearchPos = maxSearchPos = mFinalSignalPosDeg = degSigPos;
            handledDegFrame = true;
        }
    }

    else if (mCurSegs[SEGLIST] != mEndOfLists[SEGLIST]) //we're not at the end of the
ref signal
    {
        int TASearch_curFrmPos;

        maxSearchPos = (int)mRef->NrOfSamples() - mFrameSize;
        int i;
        for (i = 0;
             i < (int)mDetailedSegList->size() &&
             ((*mDetailedSegList)[i].segType != TA_SEG_MATCHED ||
              (*mDetailedSegList)[i].degPos   < mSignalPosDeg);
             i++);
        if (i != (int)mDetailedSegList->size())
            if
(mDeg->VADprofile()[mDeg->SamplePosToFrameNum(mSignalPosDeg+mFrameSize/2)]
```

```
== SQ_VAD_NO_SPEECH &&
                ((i > 0 && (*mDetailedSegList)[i-1].segType == TA_SEG_PAUSE) ||
                 (i > 1 && (*mDetailedSegList)[i-2].segType == TA_SEG_PAUSE)))
                maxSearchPos = limit((*mDetailedSegList)[i].refPos - mFrameSize,
                                     mSignalPosRef, maxSearchPos);
            else
                maxSearchPos = min((*mDetailedSegList)[i].refPos,
                                     maxSearchPos);
        else
            maxSearchPos = min(mCurSegs[SEGLIST]->refPos + round(0.200f *
mRef->SamplingFreq()),  //just make sure we don't jump too far
                               maxSearchPos);
        if (maxSearchPos < mSignalPosRef)
            OPTTHROW ((string("ERROR in HandleMissingSegment: Internal error.\n")));

        TASearch_curFrmPos  = mSignalPosRef;
        TASearch_curFrmPos  = max(mSignalPosRef, TASearch_bestFrmPos);
        TASearch_curFrmPos  = limit(TASearch_curFrmPos, 0, maxSearchPos);
        TASearch_bestFrmPos = TASearch_curFrmPos;

        minSearchPos       = TASearch_curFrmPos;
        mFinalSignalPosRef = max(mCurSegs[SEGLIST]->refPos - mFrameSize/2,
TASearch_curFrmPos);
        mSignalPosRef      = TASearch_curFrmPos;
        handledRefFrame = true;
    }

    //Deg segment with signal that doesn't occur in ref, after the end of the ref
signal:
    else
    {
        int bestMissingFrmPos = mCurSegs[mActiveList]->refPos +
mCurSegs[mActiveList]->segLen/2 - mFrameSize/2;
        bestMissingFrmPos = max(bestMissingFrmPos, mSignalPosRef); //Don't go backwards
        bestMissingFrmPos = min(bestMissingFrmPos, (int)mRef->NrOfSamples()-mFrameSize);

        minSearchPos = maxSearchPos = bestMissingFrmPos;
        mFinalSignalPosRef = bestMissingFrmPos;
        mSignalPosRef      = bestMissingFrmPos;
        handledRefFrame = true;
    }
}

void SegListTraversal::HandleGuessedSegment(int &TASearch_bestFrmPos,
                                            int &minSearchPos, int &maxSearchPos,
                                            bool &handledRefFrame, bool
&handledDegFrame)
{
    if ((handledRefFrame && handledDegFrame) || mActiveList != SEGLIST)
        OPTTHROW(( string("ERROR in HandleGuessedSegment: Invalid input
arguments.\n")));

    int   const TA_SEARCH_MIN_INCREMENT =
        mSegType == TA_SEG_PAUSE ? 0 :
        mFrameSize / 10;

    int minShift, maxShift, guessedShift, minRefPos, maxRefPos, TASearch_curFrmPos;

    //Determine the ref signal range to try out
    if (!HandleGuessedSegment_determineTAsearchRange(minShift,
                                                     maxShift,
                                                     guessedShift,
                                                     minRefPos,
                                                     maxRefPos))
        return;

    TASearch_curFrmPos  = mSignalPosDeg - maxShift;
    TASearch_bestFrmPos = max(minRefPos, TASearch_bestFrmPos + TA_SEARCH_MIN_INCREMENT);
    TASearch_curFrmPos  = max(TASearch_curFrmPos, TASearch_bestFrmPos);
    TASearch_curFrmPos  = limit(TASearch_curFrmPos, minRefPos, maxRefPos);

    minSearchPos       = max(minRefPos, TASearch_bestFrmPos);
    maxSearchPos       = min(maxRefPos, mSignalPosDeg - minShift);
    mFinalSignalPosRef = limit(mSignalPosDeg - guessedShift, minSearchPos,
maxSearchPos);
    mSignalPosRef      = TASearch_curFrmPos;
```

```
    handledRefFrame    = true;
}

bool SegListTraversal::HandleGuessedSegment_determineTAsearchRange(int &minShift, int
&maxShift,
                                                                   int &guessedShift,
                                                                   int &minRefPos, int
&maxRefPos)
{
    if (mCurSegs[mActiveList] == mEndOfLists[mActiveList] ||
        mSegType == TA_SEG_MISSING || mSegType == TA_SEG_MATCHED ||
        mActiveList != SEGLIST || mDetailedSegList == NULL || mDetailedSegList->size()
== 0)
        OPTTHROW(( string("ERROR in HandleGuessedSegment_determineTAsearchRange: Invalid
input arguments.\n")));

    int   const MIN_CONT_FRM_LEN        =
        (int)(mFrameSize - mFrameStep/2);
    int   const TA_SEARCH_STEP          =
        mFrameSize / 10;
    int   const MIN_PAUSE_LEN           =
        round(0.333f*mRef->SamplingFreq());

    //Try various ref frame positions based on info from adjacent TA_SEG_MATCHED
segments.

    int i;
    TA_segStruct const *nextMatchedSeg = NULL, *prevMatchedSeg = NULL, *curMatchedSeg =
NULL;
    for (i = 0;
         i < (int)mDetailedSegList->size() &&
         ((*mDetailedSegList)[i].segType != TA_SEG_MATCHED ||
          (*mDetailedSegList)[i].degPos + (*mDetailedSegList)[i].segLen <=
mSignalPosDeg);
         i++);
    if (i != (int)mDetailedSegList->size())
        nextMatchedSeg = &(*mDetailedSegList)[i];

    for (i = i >= 0 ? i-1 : (int)mDetailedSegList->size()-1;
         i >= 0 &&
         (*mDetailedSegList)[i].segType != TA_SEG_MATCHED;
         i--);
    if (i >= 0)
        prevMatchedSeg = &(*mDetailedSegList)[i];

    if (prevMatchedSeg != NULL &&
        mSignalPosDeg >= prevMatchedSeg->degPos &&
        mSignalPosDeg <  prevMatchedSeg->degPos + prevMatchedSeg->segLen)
        OPTTHROW(( string("I thought this never happened?! What the heck!\n")));

    else if (nextMatchedSeg != NULL &&
        mSignalPosDeg >= nextMatchedSeg->degPos &&
        mSignalPosDeg <  nextMatchedSeg->degPos + nextMatchedSeg->segLen)
        curMatchedSeg = nextMatchedSeg;

    if (curMatchedSeg != NULL &&
        mSignalPosDeg                    >= curMatchedSeg->degPos &&
        mSignalPosDeg + MIN_CONT_FRM_LEN <= curMatchedSeg->degPos +
curMatchedSeg->segLen)
    {
        mSegReliability = 1.0f;
        return false;
    }

    if (curMatchedSeg != NULL)
    {
        prevMatchedSeg = nextMatchedSeg;
        for (i = 0;
             i < (int)mDetailedSegList->size() &&
             ((*mDetailedSegList)[i].segType != TA_SEG_MATCHED ||
              (*mDetailedSegList)[i].degPos  <  prevMatchedSeg->degPos +
prevMatchedSeg->segLen);
             i++);
        if (i != (int)mDetailedSegList->size())
            nextMatchedSeg = &(*mDetailedSegList)[i];
        else
```

```
            nextMatchedSeg = NULL;
    }

    bool pauseSegBefore     = mCurSegs[SEGLIST] != mSegList->begin() &&
                              (mCurSegs[SEGLIST]-1)->segType == TA_SEG_PAUSE &&
                              (mCurSegs[SEGLIST]-1)->segLen >= MIN_PAUSE_LEN;
    bool pauseSegAfterwards = (mCurSegs[SEGLIST]+1) != mEndOfLists[SEGLIST] &&
                              (mCurSegs[SEGLIST]+1)->segType == TA_SEG_PAUSE &&
                              (mCurSegs[SEGLIST]+1)->segLen >= MIN_PAUSE_LEN;
    pauseSegBefore = pauseSegBefore &&
        (prevMatchedSeg == NULL ||
         prevMatchedSeg->refPos + prevMatchedSeg->segLen <=
(mCurSegs[SEGLIST]-1)->refPos);
    pauseSegAfterwards = pauseSegAfterwards &&
        (nextMatchedSeg == NULL ||
         nextMatchedSeg->refPos >= (mCurSegs[SEGLIST]+1)->refPos +
(mCurSegs[SEGLIST]+1)->segLen);

    //Determine the maximum and minimum shifts, based on the adjacent matched segments.
    minShift = maxShift = guessedShift = mCurSegs[SEGLIST]->degPos -
mCurSegs[SEGLIST]->refPos;

    //Use the full delay spread as search range of there is a pause segment just before
or after.
    if (pauseSegAfterwards || pauseSegBefore)
    {
        minShift = mTAinfo->MinDelay();
        maxShift = mTAinfo->MaxDelay();
    }
    else
    {
        if (nextMatchedSeg != NULL)
        {
            minShift = min(minShift, nextMatchedSeg->degPos - nextMatchedSeg->refPos);
            maxShift = max(maxShift, nextMatchedSeg->degPos - nextMatchedSeg->refPos);
        }
        if (prevMatchedSeg != NULL)
        {
            minShift = min(minShift, prevMatchedSeg->degPos - prevMatchedSeg->refPos);
            maxShift = max(maxShift, prevMatchedSeg->degPos - prevMatchedSeg->refPos);
        }

        TA_segStruct const *nextMatchedSegAfterCurMergedSeg = NULL,
                           *prevMatchedSegAfterCurMergedSeg = NULL;
        for (i = 0;
             i < (int)mDetailedSegList->size() &&
             ((*mDetailedSegList)[i].segType != TA_SEG_MATCHED ||
              (*mDetailedSegList)[i].degPos <
               mCurSegs[SEGLIST]->degPos + mCurSegs[SEGLIST]->segLen);
             i++);
        if (i != (int)mDetailedSegList->size())
            nextMatchedSegAfterCurMergedSeg = &(*mDetailedSegList)[i];
        for (i = i >= 0 ? i-1 : (int)mDetailedSegList->size()-1;
             i >= 0 &&
             ((*mDetailedSegList)[i].segType != TA_SEG_MATCHED ||
              (*mDetailedSegList)[i].degPos + (*mDetailedSegList)[i].segLen >
               mCurSegs[SEGLIST]->degPos);
             i--);
        if (i >= 0)
            prevMatchedSegAfterCurMergedSeg = &(*mDetailedSegList)[i];

        if (nextMatchedSegAfterCurMergedSeg != NULL)
        {
            minShift = min(minShift, nextMatchedSegAfterCurMergedSeg->degPos -
nextMatchedSegAfterCurMergedSeg->refPos);
            maxShift = max(maxShift, nextMatchedSegAfterCurMergedSeg->degPos -
nextMatchedSegAfterCurMergedSeg->refPos);
        }
        if (prevMatchedSegAfterCurMergedSeg != NULL)
        {
            minShift = min(minShift, prevMatchedSegAfterCurMergedSeg->degPos -
prevMatchedSegAfterCurMergedSeg->refPos);
            maxShift = max(maxShift, prevMatchedSegAfterCurMergedSeg->degPos -
prevMatchedSegAfterCurMergedSeg->refPos);
        }
    }
```

```
    if (!pauseSegBefore && curMatchedSeg == NULL && mSegType != TA_SEG_PAUSE &&
        prevMatchedSeg == NULL && nextMatchedSeg != NULL)
    {
        for (i = mDetailedSegList->findInsLocIdx(nextMatchedSeg->refPos) - 1;
             i >= 0 && (*mDetailedSegList)[i].segType != TA_SEG_MATCHED; i--);
        if (i < 0)
        {
            minShift = min(minShift, (*mDetailedSegList)[0].degPos -
(*mDetailedSegList)[0].refPos);
            maxShift = max(maxShift, (*mDetailedSegList)[0].degPos -
(*mDetailedSegList)[0].refPos);
        }
    }
    if (!pauseSegAfterwards && curMatchedSeg == NULL && mSegType != TA_SEG_PAUSE &&
        nextMatchedSeg == NULL && prevMatchedSeg != NULL)
    {
        for (i = mDetailedSegList->findInsLocIdx(prevMatchedSeg->refPos) + 1;
             i < (int)mDetailedSegList->size() && (*mDetailedSegList)[i].segType !=
TA_SEG_MATCHED; i++);
        if (i == mDetailedSegList->size())
        {
            minShift = min(minShift, (*mDetailedSegList)[i-1].degPos -
(*mDetailedSegList)[i-1].refPos);
            maxShift = max(maxShift, (*mDetailedSegList)[i-1].degPos -
(*mDetailedSegList)[i-1].refPos);
        }
    }

    for (i = 0;
         i < (int)mDetailedSegList->size() &&
         (*mDetailedSegList)[i].degPos + (*mDetailedSegList)[i].segLen <= mSignalPosDeg;
         i++);
    if (i != (int)mDetailedSegList->size() && (*mDetailedSegList)[i].segType ==
TA_SEG_GUESSED)
        mSegReliability = (*mDetailedSegList)[i].reliability;
    else
        mSegReliability = 0.0f;

    if (maxShift - minShift < TA_SEARCH_STEP)
        return false; //No room for different ref frame positions, let FullTraversal()
handle this.

    //Avoid intruding into adjacent matched segments.
    minRefPos = 0;
    maxRefPos = mRef->NrOfSamples() - mFrameSize;
    if (prevMatchedSeg != NULL)
        minRefPos =
            max(minRefPos,
                  max(prevMatchedSeg->refPos,
                        prevMatchedSeg->refPos + prevMatchedSeg->segLen - mFrameSize));
    if (nextMatchedSeg != NULL)
        maxRefPos = min(maxRefPos,
                          max(minRefPos, nextMatchedSeg->refPos));
    if (maxRefPos < minRefPos)
        OPTTHROW(( string("ERROR in HandleGuessedSegment_determineTAsearchRange: Invalid
adjacent matched segments.\n")));

    return true;
}
```