Georgia Tech Gesture Toolkit
$\mathbf{GT^2k}$

Tracy Westeyn, Helene Brashear, Amin Atrash, and Thad Starner

December 13, 2003

Contents

Part I

Toolkit Overview

Chapter 1

Introduction

The Georgia Tech Gesture Toolkit (**GT^{2}k**) is a toolkit designed to simplify the task of creating a prototype system that uses gesture recognition. Hidden Markov models (HMMs) are commonly used for gesture recognition; however, most available toolkits are geared towards speech recognition and are difficult for the casual user to adapt. The Georgia Tech Gesture Toolkit provides tools to interface to Cambridge University's HMM Toolkit, HTK [3, 1], that allows the user to concentrate on building the application rather than the intricacies of the toolkit itself.

GT^{2}k provides tools to assist with:

- data preparation
- HMM creation and training
- HMM validation
- recognition

The user is expected to provide the components of the system that interface with **GT^{2}k** :

- data generation
- results interpreter

The relationship between **GT^{2}k** and the external system is illustrated in Fig 1.1.

Fig. 1.1 GT^{2}k interaction with application components.

User Provided Components

In order for **GT^{2}k** to recognize gestures, it must first be trained on known data. This training data, and later, data to be interpreted as gestures, must be provided by the user. Once **GT^{2}k** is trained, it will perform gesture recognition. The results can then be acted on by the application designer. In order for **GT^{2}k** to best use the input data, it should be processed to find salient characteristics, or features. An example feature is the size and shape of a ball being tracked by a camera.

GT^{2}k Tools

Once data has been gathered, and features have been extracted from the data, **GT^{2}k** must be trained. The first step is to create an HMM layout that the training will be based on. **GT^{2}k** provides a tool to assist in the creation of the HMM. Next, **GT^{2}k** must be informed what gestures should be recognized. This is specified in the form of a simple grammar. The training process then involves labeling examples from the training set of data with the gestures that they belong to. **GT^{2}k** provides automatic and assisted labeling, depending on the data. After the data has been labeled, **GT^{2}k** automatically trains the HMM, preparing it for recognition.

Once the HMM has been trained, **GT^{2}k** validates the model using one of several methods, such as cross–validation or leave–one–out validation. This validation process gives an indicator of how accurate the model is likely to be when using real data. If the user is not satisfied with the results, it is a simple matter to try a different HMM layout to see if it improves results.

When training is completed to the user's satisfaction, **GT^{2}k** is ready to perform gesture recognition. Data is provided in the same form as the training data, and **GT^{2}k** will return the most likely (or n most likely) gestures. The user can then act on these results in the desired manner.

Chapter 2

Quick Start Guide

This chapter serves as a quick introduction to **GT^{2}k** . Please note that most directories have a `README` file containing useful information.

After installing **GT^{2}k** (see the included `INSTALL` file), you can use the `utils/new_project.sh` script to start a new project. The script requires three arguments: the location of the new project, the size of the feature vector you will be using, and where you installed **GT^{2}k** (such as `/usr/local/gt2k`). For example, if you wanted to start a new project in `/home/me/my_gt2k_proj` that will use a feature vector that is 24 elements long, you could run:

```
/usr/local/gt2k/utils/new_project.sh /home/me/my_gt2k_proj 24 /usr/local/gt2k
```

This command will create the directory `/home/me/my_gt2k_proj` and copy over various useful files and directories. Some of these subdirectories will be empty. These will be populated with project specific files generated by **GT^{2}k** during the data preparation and model training processes. Each project directory contains a copy of the template directory README file. This will provide some guidance on how to adapt the general project directory to the specifics of your project.

In your project directory, in the `scripts/` subdirectory is a file called `options.sh`. You can edit this file to set the options for your project. To help you understand how to edit the file and run training and testing with your data, we have provided example projects (in the `examples/` directory) and tutorials (in the `tutorials/` directory).

Chapter 3

Overview

GT^{2}k provides a user with tools for preparation, training, validation, and recognition using Hidden Markov Models for gesture–based applications. Preparation requires that the user design gesture models, determine an appropriate grammar, and provide labeled examples of the gestures to be trained. Training uses information from the preparation phase to train models of each gesture. Validation evaluates the potential performance of the overall system. Recognition uses the trained models to classify new data. At this point, **GT^{2}k** assumes data is being provided by a *Data Generator*, such as a camera, microphone, or accelerometer, in the form of a feature vector. The resulting **GT^{2}k** classification is then handled by a *Results Interpreter* as appropriate for the application.

Preparation

The preparation phase provides **GT^{2}k** with set of initial gesture models, a semantic interpretation of the data, and examples of each gesture for training.

Designing Gesture Models

Each gesture is modeled using a separate HMM. Hidden Markov models are specified by a topology which includes a set of states and transitions. These transitions define the connections between the states and the likelihood of following that connection. HMM training can update the probabilities of transitions but does not typically add or remove states from the topology. Thus, designing the gesture models involves some insight into the structure of the data and may require some experimentation to determine the best topology. **GT^{2}k** provides tools allowing novice users to automatically generate models, while still providing experienced users with the capabilities to craft models which incorporate domain-specific knowledge. Details of HMMs may be found in Rabiner's HMM tutorial [2]. Visualization tools are also provided to aid in model construction.

Specifying a Grammar

In the simplest case, recognition can be performed on one gesture at a time. This technique is known as *isolated* gesture recognition. However, sometimes it is necessary to perform *continuous* recognition on a sequence of gestures within a contiguous block of data. Knowledge of the possible sequences of gestures can be presented to **GT^{2}k** in the form of a rule–based or stochastic grammar. Grammars allow **GT^{2}k** to leverage knowledge about the structure of data, which aids in continuous recognition by using previously classified gestures to constrain the current gesture classification. Grammars also allow users to define complex gestures as a sequence of simpler gestures.

Data Collection and Annotation

Sensing devices are used to gather data about activities in the environment. Common sensing devices include cameras, accelerometers, microphones, laser range finders, and motion capture devices. These sensors typically return raw data measurements of the observable environment. This raw

data can be used directly for recognition or processed to extract the significant features as deemed appropriate for the task. This data is stored as numerical vectors, known as *feature vectors*, and form the data set over which **GT^{2}k** operates. The range of values and the length of the vector is dependent on the application. A typical gesture example would appear as a sequence of these feature vectors. For instance, video-tracked gestures may return the position of the hand at each video frame as an (x, y) coordinate. This results in a feature vector of length two with two real values corresponding to (x, y) position. A gesture which completes in 29 frames would be represented with 29 feature vectors.

In order for **GT^{2}k** to properly understand the data it receives, the data must be annotated by the user. This requires the user to specify which gestures appear in each of the training examples.

Training

The major contribution of **GT^{2}k** is the abstraction of the training process. Once the preparation phase is complete, training of the model requires only that the user select a training validation method and configure a few method–specific parameters. The training process is automated, returning results and models which can later be used for recognition in various systems. This abstraction allows users to avoid the details of the underlying algorithms. **GT^{2}k** provides a few default training/validation methods, however user–defined methods can easily be integrated.

Training/validation methods provide quantitative feedback concerning the training process. Such methods typically require that data collected for training be separated into two sets, a *training set* and a *validation set*. The training set is the set of data used to train the models, and the validation set is used to measure the performance of the trained models on unseen, yet known data. Evaluation of the model using the validation set helps gauge how well the model will generalize to new data. It also helps determine if overfitting occurs during the training process. Overfitting results in improved performance over the training data but a decline in generalization, and thus a decrease in performance over new data.

Two standard training/validation techniques provided by **GT^{2}k** are *cross–validation* and *leave–one–out validation*. Cross-validation randomly selects a predetermined percentage of the data (typically 66.6%) as the training set. The remaining data (typically 33.3%) acts as the validation set. Leave–one–out validation selects one data example as the validation set and uses the remainder of the data as the training set. The training/validation phase is repeated for every permutation of the data set with one element "left out" of the training set. The results of each iteration are then tallied to compute overall statistics of the models' performance.

Validation and System Performance

The training/validation methods provide a quantitative measure of the system's performance based on the accuracy of recognition. The **GT^{2}k** metric for accuracy is the standard definition that incorporates substitution, insertion, and deletion errors. Substitution errors occur when the system incorrectly classifies a gesture. Insertion errors occur when the system hallucinates the occurrence of a gesture. Deletion errors arise when the system fails to recognize the occurrence of a gesture within a sequence of gestures. If we let S represent substitutions errors, I represent insertion errors, D represent deletion errors, and N represent the total number of examples, then accuracy is defined as:

$$Accuracy = \frac{N - S - D - I}{N}.$$

It should be noted that insertion and deletion errors can only occur during continuous recognition. When recognizing gestures in isolation, the values for D and I will always equal zero.

System performance is reported in the form of a confusion matrix. The matrix reports the ground–truth gesture versus the gesture as classified by the system.

Recognition Application

Models generated during the training phase can be used for recognition of new data. As with the training phase, the underlying algorithms have been abstracted. Once an unclassified gesture is

received, **GT^{2}k** calculates the likelihood of each model. The actual probabilities are calculated using the Viterbi algorithm. This information can be used by the results interpreter as deemed appropriate by the application, for example, taking the most likely gesture or considering the probability of each gesture for making a decision.

Once a model of each of the gestures has been trained, it can be used independently of the training system. For our fictitious example of recognizing gestures for controlling ground vehicles, the individual models could be embedded in an autonomous robot. When the traffic controller gestures towards the robot, **GT^{2}k** receives features from the robot's sensors, calculates the probability of each model given the features, and returns a list of the most likely gestures issued by the traffic controller. The robot acts accordingly based on this data.

Part II

Using GT^{2}k

Chapter 4

Data Preparation

The preparation phase provides **GT^{2}k** with set of initial gesture models, a semantic interpretation of the data (a grammar), and examples of each gesture for training (annotated data). This chapter will discuss the tools provided to assist the user with the data preparation process.

4.1 Designing Gesture Models

Each gesture is modeled using a separate Hidden Markov Model (HMM). The HMM model parameters (i.e.: the topology, observation probabilities, transition probabilities) are specified using the HTK HMM Definition File format. For users unfamiliar with this format (or unfamiliar with HMMs in general), tools are provided to aid in the creation of the model definition files. A visualization tool is also provided to ensure that the topology was correctly specified.

4.1.1 HMM Generation Tool

Name:	`gen_hmmdef.pl`
Location:	`gt2k/tools/hmm_generator`
Detailed Usage:	`gt2k/tools/hmm_generator/README`
Purpose:	generates HMM definition files

Generation of an HMM definition file requires three values from the user: the number of states in the topology, the number of observations per state (elements per feature vector), and the number of states that can be reached from any given state (number of neighboring states that can be skipped). These values are specified as command line options. Fig 4.1 shows a command line for the generation of a 5–state HMM with 4 observations per state. Fig 4.2 shows the resulting HMM definition file. It should be noted that **GT^{2}k** assumes that each HMM has an initial and final non–emitting state. This means that to create a model with three emitting states, a five–state model must actually be specified.

The user is not required to specify transition probabilities, as they are automatically generated by the `gen_hmmdef` program. The generated transition matrix assume that all transitions are equally probable. For most projects, this assumption is sufficient for an initial topology. However, if the user has more detailed knowledge about the initial topology of the model, the HMM definition can be edited to reflect this information. The <TransP> section of the `gen_hmmdef` output specifies the transition matrix for the HMM. Each row represents a "current" state of the HMM, and each column represents a state that the model can transition to from the "current" state. Figure 4.3 shows a visualization of the transition matrix in Figure 4.2.

```
gen_hmmdef.pl -n5 -v4 -s1
```

Fig. 4.1 Example usage of `gen_hmmdef`

```
<BeginHMM>
    <VecSize> 4
    <NumStates> 5 <nullID><USER>
    <State> 2 <NumMixes> 1
       <Mixture> 1 1.0
       <Mean> 4
          0.0 0.0 0.0 0.0
       <Variance> 4
          1.0 1.0 1.0 1.0
    <State> 3 <NumMixes> 1
       <Mixture> 1 1.0
       <Mean> 4
          0.0 0.0 0.0 0.0
       <Variance> 4
          1.0 1.0 1.0 1.0
    <State> 3 <NumMixes> 1
       <Mixture> 1 1.0
       <Mean> 4
          0.0 0.0 0.0 0.0
       <Variance> 4
          1.0 1.0 1.0 1.0
    <TransP> 5
       0.000e+00 1.000e+00 0.000e+00 0.000e+00 0.000e+00
       0.000e+00 3.333e-01 3.333e-01 3.333e-01 0.000e+00
       0.000e+00 0.000e+00 3.333e-01 3.333e-01 3.333e-01
       0.000e+00 0.000e+00 0.000e+00 5.000e-01 5.000e-01
       0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
<EndHMM>
```

Fig. 4.2 Example HMM Definition file

4.1.2 HMM Visualization Tool

Name:	`hv.sh`
Location:	`gt2k/tools/hv`
Detailed Usage:	`gt2k/tools/hv/README`
Purpose:	Visualize HMM definition files

The HMM visualization tool can be used to graphically represent the topology and transition probabilities of an HMM definition file. Fig 4.3 is the graphical representation of the HMM definition file shown in Fig 4.2. It is a five state HMM where each of the three emitting states can skip its neighbor.

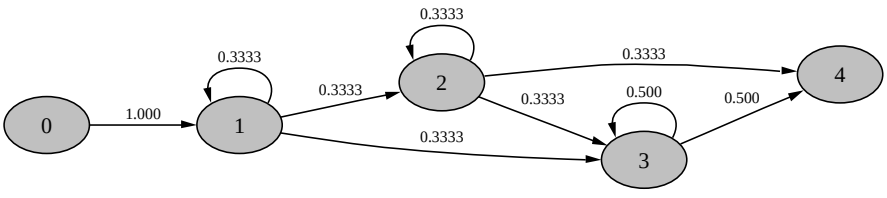

Fig. 4.3 Visualization of Fig 4.2

4.2 Specifying a Grammar

In order to perform training and recognition, **GT²k** must be supplied with a grammar. This grammar can be automatically generated or the user may manually specify it. In the case of automatic generation, the grammar will allow **GT²k** to perform isolated recognition; that is, one gesture at a time. If continuous recognition is desired, better results may be obtained by manually specifying a grammar that more closely matches the structure of the data to be recognized. A partial grammar for continuous recognition is shown in Figure 4.4; for the entire sample, please see the file `examples/acceleglove/strictgrammar` in the **GT²k** directory.

A full description of the grammar file's format may be found in the HTK reference manual Section 17.15; however, we will briefly describe some of the more important features here. The grammar is specified using variables and commands. Each variable, or category, is denoted with a '`$`' symbol, as in "`$noun`". Commands represent gestures to recognize, and are simple text strings such as "`adult`". Each category may consist of one or more commands, each separated with the '|' (pipe) character. Sentences are specified using parenthesis. For isolated recognition, a sentence is as simple as "`( $gesture )`"; for continuous recognition, a sentence can be more complicated (as in Figure 4.4). Using angled brackets around a symbol or variable, such as "`< $word >`" denotes that the enclosed symbol may appear any number of times.

```
$noun       =  adult | america | arch | area | badge | boy | ... ;
$pronoun    =  he | i | they | we | you ;
$verb       =  apologize | appreciate | believe | collect | ... ;
$adjective  =  bald | black | brown | deaf | dizzy | english ... ;
$anynoun    =  $noun | $pronoun ;

( start_sentence $anynoun $verb $adjective $noun end_sentence)
```

Fig. 4.4 Sample grammar specification for continuous recognition

4.3 Data Collection and Annotation

In order to train and use an HMM with **GT²k** , data must be collected. First, some data to be used for training the HMM should be gathered. Later, **GT²k** can perform recognition on new data collected by the same methods. Typical sources of data include sensors such video cameras, accelerometers, and microphones.

Before **GT²k** can use the collected data, salient features should be extracted from the data. While **GT²k** can accept some forms of raw data (such as an entire frame of video), better performance is usually realized by computing features over the data. A simple example of a feature would be the (x, y) location of a ball being tracked in a frame of video. The feature vector for each sample of data should be the same length; in the ball example, each frame of video would have a feature vector of length two.

Once features have been extracted from the data that will be used for training, they should be stored in files, with one feature vector per line. The structure of the files is generally up to the user; however, it is often the case that the easiest structure to work with is one gesture per file. Regardless of the data structure chosen, a Master Label File (MLF) must be created. This file allows **GT²k** to associate the data for gestures with labels defining what gestures that data represents. If data is stored with one gesture per file, the `gen_mlf.sh` script may be used to create the MLF. This script is located in the `examples/gestpan/scripts` directory in the **GT²k** distribution. Otherwise, the MLF must be created by the user. Unfortunately, the MLF format is quite complicated, but powerful. For full details, please consult Section 6.3 of the HTK documentation and the tutorials found in the `tutorials/` directory of the **GT²k** distribution. There is also a customized `gen_mlf.sh` script in the `examples/acceleglove/scripts` directory that deals with a more complicated example.

Chapter 5

Training and Recognition

Once data has been prepared, the HMM is ready to be trained. Training is fairly simple from the user's perspective, but may take a long time, depending on the complexity of the project. Once training is complete, the model may be used for recognition. This chapter will discuss the tools provided to support training and recognition.

5.0.1 Training

The training process is initiated using the `train.sh` script. This script was copied into the `scripts/` directory of your projects directory by the `new_project.sh` script. `train.sh` is invoked with one argument, which is the options file `scripts/options.sh`. After training completes, a confusion matrix will be printed describing the model's performance using the specified validation method. The files describing the models will be left in a set of directories under `models/`. Each directory will have a name of the form `hmm?.?`. For cross–validation, the final model will be stored in `hmm0.3/newMacros`. When using leave–one–out validation, the first numeral in the directory name will represent the sample that was left out. In order to decide which model to use, look at the various results.mlf files and find one that classified correctly. Then use the model file (`newMacros`) from the corresponding directory ending in `.3`. This will not be necessary in future releases.

Training Options

The `scripts/options.sh` file contains all of the settings that will dictate how your HMM is trained. The options are described here, and in the file itself. Most of the defaults should be okay, but please read over them at least once to be sure.

- `PRJ`
  Defines the location of the project. This can either be set to an absolute pathname or `‘pwd‘` if `scripts/train.sh` is run from the main project directory.
- `SCRIPTS_DIR`
  Defines the location of the `scripts` directory. Generally this is assumed to be inside the directory specified for `PRJ` above.
- `UTIL_DIR`
  Defines the location of the **GT^{2}k** `utils` directory. The default location is `/usr/local/gt2k`, but may have been changed during installation.
- `VECTOR_LENGTH`
  Specifies the length of the feature vector used for this data. This should be the same number as was specified with the `gen_hmmdef.pl` script.
- `MIN_VARIANCE`
  This is the minimum variance that each state should be allowed to have during training. This is to prevent certain issues with HTK, and should generally be left at the default.

- HMM_TOPOLOGY_DIR
  Defines the location of the **hmmdefs** directory. Generally this is assumed to be inside the directory specified for PRJ above.

- HMM_LOCATION
  Specifies the file that contains the topology for the HMM to train. This is initially set to "<none>", so make sure to change it.

- INITIALIZE_HMM
  Set to "yes" or "no". Defines whether to have HTK initialize model parameters or not. Only set to noif you have a previously–initialized HMM from another source.

- GEN_TRAIN_TEST
  Set to "yes" or "no". If set to yes, **GT^{2}k** will automatically generate new testing and training sets each time training is run. If you have defined your own testing and training sets, or wish to reuse a previously generated set, set this to no.

- TRAIN_TEST_VALIDATION
  Set to "CROSS" or "LEAVE_ONE_OUT" to select the type of training/testing to perform. Cross–validation uses 66% of the data as training data, and the remaining 33% as data for testing. Leave–one–out validation uses all off the data except for one sample as training data, and the remaining one as test data. It does this for every possible permutation, and can take a very long time.

- DATAFILES_LIST
  Specifies a file containing a list of all the data files that will be divided into training and testing sets.

- GRAMMARFILE
  Specifies a file containing the grammar definition for the project.

- TOKENS
  A file containing the commands defined in the grammar, one per line.

- MLF_LOCATION
  Defines the location of the Master Label File.

- GEN_EXT_FILES
  Set to "yes" or "no". Specifies whether or not .ext files should be generate. Leave set to yesunless they have already been generated.

- EXT_DIR
  Specifies which directory HTK–specific files will be placed in.

- GEN_GRAMMAR
  Set to "yes" or "no". Specifies whether or not to automatically generate a grammar based on the data. If you have manually specified a grammar, set to no.

- OUTPUT_MLF
  Specifies the file that will contain the results of recognition.

- LOG_RESULTS
  Specifies the file that will contain the results of the validation process.

- AUTO_ESTIMATE
  Set to "yes" or "no". Specifies whether or not to allow HTK to estimate gesture boundaries for data with multiple gestures per file. This can be useful if you're unsure of the exact boundaries between gestures. Setting this to nowill speed up training.

- HMM_TEMP_DIR
  Specifies the directory to use for storage of intermediate models during training.

- `TRACE_LEVEL`
  Set to 1—7. Specifies level of debugging for HTK. Level 7 provides the most debugging output.

Recognition

Once the HMM has been trained, new data can be used for recognition. The script `recognize.sh`, found in the `scripts/` directory in your project directory can be used for this purpose. `recognize.sh` requires four arguments: the name of a data file to recognize, the name of a file to store the recognition results in, the name of the options file, and the name of the trained model. The trained model is the one that was chosen at the end of the training process.

The output of the recognizer script will be in the form of a Master Label File. Several possible gestures will be listed, ranked by a likelihood score. The topmost gesture is the most likely classification.

Bibliography

[1] HTK Hidden Markov Model Toolkit home page. http://htk.eng.cam.ac.uk/.

[2] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[3] S. Young. *HTK: Hidden Markov Model Toolkit V1.5.* Cambridge Univ. Eng. Dept. Speech Group and Entropic Research Lab. Inc., Washington DC, 1993.