Implementation of a Relational Database as an Aid to Automatic Target Recognition

Christopher C. Frost
Huntsville High School
AMCOM MRDEC
Mentor: Steven Vanstone

August 4, 1999

Abstract

The integration of a Relational Database and Automatic Target Recognition (ATR) systems is presented. The ability to select ideal target references from a comprehensive relational database for the creation of filters will enhance the ability of an ATR system to detect and classify targets. The process of identification and classification of targets to be utilized in the creation of filters, design and implementation of the relational database, construction of tools to populate the database through extraction of current references, and creation of example code illustrating the ability to integrate existing ATR systems with the designed database are presented and discussed.

1 Introduction

Automatic Target Recognition (ATR) systems detect and identify targets through pattern matching techniques, such as space-invariant transforms[1]/circular harmonic functions[2] or synthetic discriminant functions[3] (generalizations of matched spatial filters[4]). ATR enables such tasks as target acquisition for missile guidance systems, matching fingerprints or retina scans with their respective owners, and robotic delivery of items ordered in a restaurant to customers.

However, a common problem with ATR systems has to do with filters being created through manual selection of files and specific references therein. Not only is this method time consuming, but more importantly, often leads to poorly optimized filters with regard to the fact that more ideal references exist elsewhere for this targeting instance. The goal of this project is to develop a system capable of

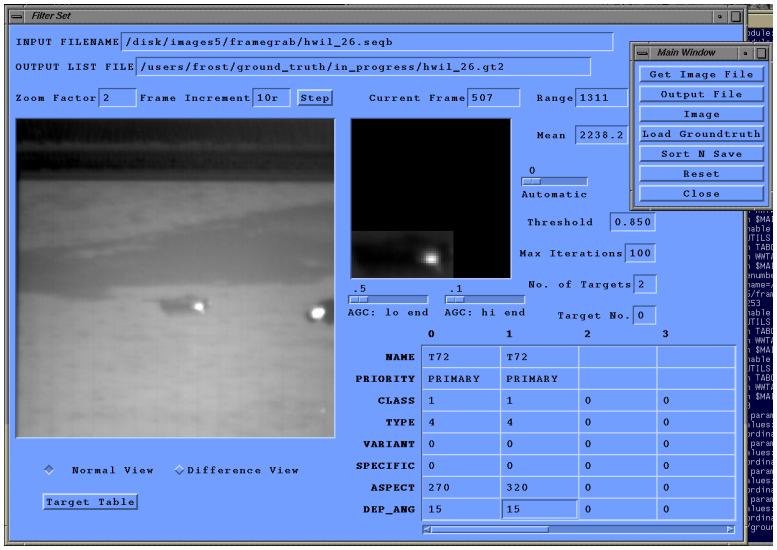

Figure 1: Screenshot from `cfl_gt2`, the main program used for ground-truthing.

giving an ATR system access to a much more comprehensive selection of references, selected according to various parameters determined by the ATR system, yielding an increase in ability to locate and identify targets.

This implementation is centered around the requirements of target acquisition for the Image and Signal Processing Directive of Future Missile Technology Initiative (FMTI), allowing the ATR system to query the database to request target references meeting any conditions present in the database (such as distance from target, elevation, etc.)

2 Ground-truthing Image Sequences

Ground-truthing is the process of manually analyzing sequences of recorded data and identifying target types, specifying target signature outlines, and pinpointing consistent aimpoints for each target. This information is the basis of the filters used for acquisition by ATR systems. The first step to be taken had to be ground-truthing (a screenshot of which can be found on page 2, Figure 2), so that there would be data from which programs could extract, and to serve as references from which filters could be created.

3 Designing the Database

To serve as the database server, three systems were considered: Oracle[1], PostgreSQL[2], and MySQL[3]. I evaluated each with regard to portability, speed, available language bindings, and maturity.

Oracle is known for its ability to handle databases of enormous size, certainly much more gracefully than the other possibilities and is a very mature product. However, it is only available on a select few platforms and the supported language bindings number few, ruling it out for the current time. PostgreSQL was considered next, and was also a very mature product (with its beginning in Berkeley in the mid-eighties). PostgreSQL, primarily because of it being open source, is very portable, and has a wide array of language bindings. It is also one of the very few Relation Database Management Systems (RDBMSs) offering an Object-Relational interface, enabling many new possibilities in the area of tables and relations. However, PostgreSQL still has a way to come in speed and was thus not an ideal choice. Next was considered MySQL, a derivative of mSQL. Also open source, this RDBMS is very portable, has a wide array of language bindings, is a fairly mature product, and has shown itself to be extremely fast. Thus, MySQL was selected to serve as the database backend for this project.

In order for this database to meet the needs of the Image and Signal Processing Directive I consulted those who worked on the gt2 format and created filters from ground-truthed data on various aspects of the information to be stored, as well as personally analyzing code which handles this data. After this work the planning of the layout for the database was began. For this project there would be many differing classes of targets (and types of targets, and variants of types, and finally specifics of variants), so a table (Table 1; Appendix C.1, Page 26) holding generic information for each specific was designed. A table for each target id (which is the reference string unique to each target, in the format "`class_type_variant_specific`") was constructed to facilitate the large number of references to be stored in this database. (Table 2; Appendix C.3, Page 26) Because of space limitations the database stores paths and filenames (indexed by a unique id) in a separate table (Table 3; Appendix C.2, Page 26), letting the filter generator retrieve the image sequence from the indicated location rather than querying the database for the actual image (though the infrastructure for this possibility has been designed (Table 4; Appendix C.4, Page 27)). Relationships among these tables are shown in Table 5.

[1] `http://www.oracle.com`
[2] `http://www.postgresql.org`
[3] `http://www.mysql.com`

Table 1: Target table properties

FIELD	TYPE	USE
id	unsigned integer, not null, auto incremented, primary key	used as a reference and unique identifier
name	variable length character	stores name of target
class	unsigned integer	class (tank, apc, et)
type	unsigned integer	type of class (T72, BMP)
variant	unsigned integer	variant of type
specific	unsigned integer	specific of variant
priority	unsigned integer	priority of target (for allocating resources)

Table 2: Frame table properties

FIELD	TYPE	USE
id	unsigned integer, not null, auto incremented, primary key	used as a reference and unique identifier
time_record	HHMMSS	time at which the sequence was recorded
time_gt	YYYYMMDDHHMMSS	date and time reference was first ground-truthed
time_modified	YYYYMMDDHHMMSS	date and time of last modification
aspect	floating double precision	aspect of target with regard to camera
depression	floating double precision	angle of depression
sequence_id	unsigned integer	reference to sequence_location.id
findex	unsigned integer	index number found in gt2 format
frame	unsigned integer	frame number
aim_x	unsigned integer	X coordinate aimpoint
aim_y	unsigned integer	Y coordinate aimpoint
ul_x	unsigned integer	upper left X coordinate for patch

Table 2: Frame table properties (*continued*)

FIELD	TYPE	USE
ul_y	unsigned integer	upper left Y coordinate for patch
range	unsigned integer	range to target (in meters)
patch_min	floating double precision	minimum pixel value of all patch pixels
patch_max	floating double precision	maximum pixel value of all patch pixels
patch_mean	floating double precision	mean pixel value of all patch pixels
b_mean	floating double precision	mean background pixel value
patch_edge_-offset_top	integer	for use in cad overlay (for creating now-rectangular patches)
patch_edge_-offset_-bottom	integer	for use in cad overlay
patch_edge_-offset_left	integer	for use in cad overlay
patch_edge_-offset_right	integer	for use in cad overlay
scm	floating double precision	signal to cluster ratio

Table 3: Sequence location table properties

FIELD	TYPE	USE
id	unsigned integer, not null, auto incremented, primary key	used as a reference and unique identifier
file	text	filename and path of file location
bytes_pixel	unsigned integer	bytes per pixel for sequence

Table 4: Image table properties

FIELD	TYPE	USE
id	unsigned integer, not null, auto incremented, primary key	used as a reference and unique identifier
frame_table	variable length character	to which table the frame_id belongs
frame_id	unsigned integer	to which frame we belong

Table 5: Table relationships

target.id (one)		→	frame_t_<id> (many)
sequence_location.id (one)		→	frame_t_<id> (many)
image.id (one)		→	frame_t_<id> (many)
image.frame_table	and	→	frame_t_<id>.id (one)
image.frame_id (one)			

4 Communication with the Database

To demonstrate interfacing code written in C with the MySQL database, an example program was written illustrating the ability to connect to the database server, select databases, perform select and insert queries, loop through the results of queries, free query memory, etc. The source to this program can be found in Appendix A (Page 10).

This program is composed of three main functions: `main()` which initializes the connection and calls the other two functions to perform queries, `select_funct()`, which performs a query passed to it via an argument and then returns data, and `update_funct()`, which performs data-modifying queries (such as updates, inserts, or dropping).

The program begins by calling `mysql_connect()` with the `mysql`, `host`, `user`, and `password` arguments. If `host` is null it is assumed to be localhost, if `user` is null it is assumed to be the current user, and if `password` is null assume no password. Then information on the connection type, id of the connection instance, and version of the server are given (showing that we are truly connected). The database on which we will be operating is then selected using the `mysql_select_db()` call.

Once this has been accomplished, `main()` calls `select_funct()` with a query asking for the `id`, `name`, and `other_info` fields from the `epoch` table. `select_funct()` then submits the query and stores its results. `mysql_num_fields()` is then called for use in the loop which cycles through the rows of data returned. A for statement then loops through each column, retrieving and printing cells of data. Once this process has completed, `mysql_free_result()` is called to free the memory used by the `mysql_store_result()` call holding the returned data of the query. Assuming this query function finishes without running into a catastrophic error, control is returned to `main()` which then calls `update_funct()` with an insert query. `update_funct()` submits the query, obtains the number of rows affected through `mysql_affected_rows()`, and exits. Another select query is then performed to verify that a new record has been inserted, and finally a delete query is sent deleting the row which was inserted.

5 Loading Gt2 File Formatted Data

Now that the database had been designed, a tool needed to be written capable of parsing files in the gt2 format and loading this ground-truthed information into the database and appropriate tables. For this project the language Python[4] was selected for its simple syntax, elegant and clean semantics, powerful methods of conveying ideas, exception handling, dynamic typing and binding, orthogonal structure, stability, portability, and extensibility and flexibility. The source to this program can be found in Appendix B (Page 16).

The program begins with the `get_files()` function, checking to see whether the program is a module being called from another program or from the command line. If it was called from the command line the system arguments are checked and input and output files are stored. The `process_file()` function is called, which stores the output of `get_known_targets()` into the variable `known_targets`. `get_known_targets()` works by connecting to the database, storing the query (querying for class, type, variant, and specific from the table target), and returning the output from the query. Once this list has been returned "gt2import.py" steps through the current input file line by line (thereby allowing files of extreme length to be processed).

Upon encountering the first line the bytes per pixel value is found by calling `get_bytes_pixel()`, which opens the file again and searches for the first reference, then parsing it and extracting its bytes per pixel value. Then the sequence location is stored and `load_sequence_id(bytes_pixel)` is called, which connects to the database, checks to make sure this sequence has not already been entered into

[4] `http://www.python.org`

the database, and inserts the needed information into the `sequence_location` table (Appendix C.2) if this is a new sequence location. Control is handed back to `process_file()` and the time at which this sequence was recorded is stored.

The second line of the gt2 format contains the number of targets in the file, so this value is stored as `number_targets` for later use. Starting on line three the number of references for each target is stored, so `number_targets` lines are read and parsed for the number of references for each target. These values are then stored in a dictionary for use in parsing the main sections of this file.

In lines four through four plus `number_targets` target specific information is stored. "gt2import.py" iterates through `number_targets`, reading each line and storing the values contained on that line in a list which is then transformed into a dictionary. A consistency check is also done (through `check_consistency()`) to help ensure the file format is valid. In the gt2 format, priority levels are stored as words ("PRIMARY," "SECONDARY," ...) but the designed database uses numerals to sort priority, so this value is converted. The class, type, variant, and specific values are then stored as a string into the list `target_ids`, to enable checking. The current target code is checked against the list to which was just added and if there is no match "gt2import.py" proceeds and queries the database to see if this target is entered or not. If not, all needed target information is stored into the database through an `INSERT` query, if the target is already present, this process is skipped.

We have now entered the "main" section of the gt2 file, where reference data is stored. "gt2import.py" iterates through `list_targets`, checking to see whether it should append or overwrite each time. The temporary output file is opened, however many lines were indicated earlier are read, checked for consistency, and stored in a dictionary. The output is then formatted as a tab delimited file and written to the output file. After this if EOF is encountered "gt2import.py" breaks, else it continues to the next line. Once the current target's references have been read and written, the output file is closed, current directory is stored, and the `load_frame_rows()` call is made, loading the output data into the database (the `LOAD` command is significantly faster than inserting rows individually). Once the data is loaded the temporary output file is erased and the next target is processed.

6 Results

The utility "gtimport.py" successfully converts data currently in the gt2 format for use in the designed database, allowing all ground-truthed data to be loaded into this database. Communication with an external database within C was proven to be relatively easy, as demonstrated by mysql_example.c (Appendix A). Finally, newly ground-truthed sequences were easily added.

7 Conclusions

This project has shown that integration a relational database with ATR systems is possible, that current routines can be made more flexible through the ability to abstract existing tools from file formats, and that database integration can provide more accurate filters thereby improving ATR Systems' ability to detect and identify targets.

8 Areas for Further Study

The two primary areas of study to follow up on this research deal with extending current ATR Systems to access the designed database using the example communication code and extending "gt2import.py" to parse a wider array of formats.

9 Acknowledgments

I would like to thank my mentor, Steven Vanstone, for his guidance and knowledge, Tom Branch for his help in debugging mysql_example.c, Jonathan Mills whose idea was the basis of this project, Michael Beatty for his advice, Victoria Stanfield for her help with the MySQL C bindings, and Joseph Robertson for advice on "gt2import.py."

A mysql_example.c

```
/*
 * This code shows how one would connect to a MySQL database, do select
 * statments, parse the output generated by these statments, insert new
 * information into the database, and drop information that may no longer be
 * needed.
 *
 * Originally written in 1999 by Chris Frost (<chris at frostnet.advicom.net>)
 * under the Science and Engineering Apprenticeship Program.
 *
 * $Id: mysql_example.c,v 1.4 1999/07/29 20:43:38 frost Stab $
 */

#include <stdio.h>
#include <mysql.h>
#include <errmsg.h>

/* Define 'DEBUG' for some verbose output */
/* #define DEBUG */

int select_funct(MYSQL mysql, const char *query)
{
        /*-------------------------------*
         * Let's do some defining for the needed values *
         *-------------------------------*/
        int query_result; /* Value returned by mysql_query */
        MYSQL_RES query_out, *query_out_ptr; /* Output from query */
        int query_out_rows, *query_out_rows_ptr;
        int i; /* For loop to display contents of num_query_columns[ ] */
        unsigned int num_query_columns;
        unsigned int num_rows_affected;
        MYSQL_ROW row;
        /* Length of a returned field. we have no idea how long it is, but 100
         * sounds like it may work (hmm, this isn't ugly) */
        /* Sample code simply stored this value into an unreferenced pointer */
        unsigned int lengths[100], *lengths_ptr;

        /* Associate pointers */
        query_out_rows_ptr = &query_out_rows;
        query_out_ptr = &query_out;
        lengths_ptr = lengths;


        /* Perform the query on the database, use mysql_real_query if there
         * might be binary data returned (mysql_query interprets '\0' as the
         * end of the query string). Also, mysql_real_query is faster
```

```
 * since it does not call 'strlen()' on the query string. */
if (query_result = mysql_query(&mysql, query))
{
        fprintf(stderr, "ERROR running query:\n%s\n\n",
                        mysql_error(&mysql));                                   50
        return (EOF);
}
else
{
        #ifdef DEBUG
        printf("Debug: query sent to database.\n");
        #endif
}

/* Store query results */                                                       60
if (!(query_out_ptr = mysql_store_result(&mysql)))
{
        fprintf(stderr, "ERROR getting query results:\n%s\n\n",
                        mysql_error(&mysql));
        return (EOF);
}
else if (query_out_ptr) /* There are rows */
{
        #ifdef DEBUG
        printf("Debug: query seems to have succceeded\n");                      70
        #endif
        printf("\nOutput from query '%s' follows.\n", query);
        printf("-------------------------------------------------\n");
        num_query_columns = mysql_num_fields(query_out_ptr);
        while (row = mysql_fetch_row(query_out_ptr))
        {
                lengths_ptr = mysql_fetch_lengths(query_out_ptr);
                /* Number of columns is one if only one total, but
                 * since we start at 0 (one less), we only do a less
                 * than and not an equals */                                      80
                for (i = 0; i < (int) num_query_columns; i++)
                {
                        /* Let's line everything up nicely */
                        if (i > 0)
                                printf("\t");

                        /* Suggested to use lengths_ptr[i] instead of
                         * lenghts[i]. Not sure why . . . */
                        printf("%.*s", (int) lengths_ptr[i], row[i] ? row[i] : "NULL");
                }                                                               90
                printf("\n"); /* Return character for above printf,
                                        * this way we can print out multiple
                                        * columns */
        }
```

```
                printf("-------------------------------------------------\n");
                printf("End output from query.\n\n");

                /* Number of returned columns and rows */
                query_out_rows= mysql_num_rows(query_out_ptr);
                #ifdef DEBUG                                                    100
                printf("Debug: %lu rows returned.\n", query_out_rows_ptr);
                if (num_query_columns > 1)
                        printf("Debug: %u columns returned.\n",
                                num_query_columns);
                else if (num_query_columns = 1)
                        printf("Debug: %u column returned.\n",
                                num_query_columns);
                else
                        printf("Error: Number of columns returned from query
                                is %u, less than 1\n", num_query_columns);      110

                printf("Debug: info on last query: %s\n", mysql_info(&mysql));
                #endif
        }
        else /* mysql_store_result() returned nothing; should it have? */
        {
                if (mysql_num_fields(&mysql) == 0)
                {
                        /* Query does not return data
                         * (it was not a SELECT) */                            120
                        num_rows_affected = mysql_affected_rows(&mysql);
                        printf("Number of rows affected by query: %d\n",
                                num_rows_affected);
                }
                else /* mysql_store_result() should have returned data */
                {
                        fprintf(stderr,"Query Error: %s\n",mysql_error(&mysql));

                        /* If we get a number, what does that mean? */
                        #ifdef DEBUG                                            130
                        printf("Debug: %d", mysql_num_fields(&mysql));
                        #endif
                        return (EOF);
                }
        }

        /* Free the memory used by mysql_store_result() */
        mysql_free_result(query_out_ptr);
}
                                                                                140

int update_funct(MYSQL mysql, const char *query)
{
```

```
/*_____________________________*
 * Let's do some defining for the needed values *
 *_____________________________*/
int query_result; /* Value returned by mysql_query */
MYSQL_RES query_out, *query_out_ptr; /* Output from query */
int query_out_rows, *query_out_rows_ptr;
unsigned int num_rows_affected;                                    150
/* Length of a returned field. we have no idea how long it is, but 100
 * sounds like it may work (hmm, this isn't ugly) */
/* Sample code simply stored this value into an unreferenced pointer */
unsigned int lengths[100], *lengths_ptr;

/* Associate pointers */
query_out_ptr = &query_out;
lengths_ptr = lengths;

                                                                   160
/* Perform the query on the database, use mysql_real_query if there
 * might be binary data returned (mysql_query interprets '\0' as the
 * end of the query string). Also, mysql_real_query is faster
 * since it does not call 'strlen()' on the query string. */
if (query_result = mysql_query(&mysql, query))
{
        fprintf(stderr, "ERROR running query:\n%s\n\n",
                        mysql_error(&mysql));
        return (EOF);
}                                                                  170
else
{
        #ifdef DEBUG
        printf("Debug: query sent to database.\n");
        #endif
}

if (mysql_num_fields(&mysql) == 0)
{
        /* Query does not return data                              180
         * (it was not a SELECT) */
        num_rows_affected = mysql_affected_rows(&mysql);
        printf("Number of rows affected by query: %d\n", num_rows_affected);
}
else /* mysql_store_result() should have returned data */
{
        fprintf(stderr,"Query Error: %s\n",mysql_error(&mysql));

        /* If we get a number, what does that mean? */
        #ifdef DEBUG                                               190
        printf("Debug: %d", mysql_num_fields(&mysql));
        #endif
```

```
        return (EOF);
    }
}


int main()
{
    /*————————————————————————*                                          200
     * Let's do some defining for the needed values *
     *————————————————————————*/
    MYSQL mysql;
    char host[] = "host",  /* If NULL assumed to be localhost, may be
                            * ip address or hostname */
        user[]= "user",     /* MySQL login name, if null current user */
        password[]="password",/* If null no passowrd, do not encrypt */
        db[] = "database";       /* Database to which we will connect */
    unsigned int port;          /* If not zero will use for tcp/ip
                                                          * connection */  210
    const char unix_socket, *unix_socket_ptr;
    unsigned int client_flag;   /* Usually [0|null], but can set various
                                * client flags, such as compress, ODBC,
                                * and a few other things */
    const char query_select[] = "SELECT id, name, other_info FROM epoch";
    const char query_update[] = "INSERT INTO epoch (name, other_info)
        values ('M60','sand-colored paint')";
    const char query_delete[] = "DELETE FROM epoch WHERE name = \"M60\"";

    /* Associate pointers */                                               220
    unix_socket_ptr = &unix_socket;

    /*————————————————————*
     * Now lets init and connect to mysql *
     *————————————————————*/
    printf("Connecting to MySQL server...\n");
    if (!mysql_connect(&mysql, host, user, password))
    {
        fprintf(stderr, "\nERROR connecting to database:\n%s\n\n",
                        mysql_error(&mysql));                              230
        return (EOF);
    }

    /* Various bits of info */
    printf("Connection Info: %s\n", mysql_get_host_info(&mysql));
    printf("Connection id: %d\n",mysql_thread_id(&mysql));
    printf("Server Version: %s\n\n", mysql_get_server_info(&mysql));

    if (mysql_select_db(&mysql, db))
    {                                                                      240
        fprintf(stderr, "\nERROR selecting database:\n%s\n\n",
```

```
                        mysql_error(&mysql));
        return (EOF);
}


/*———————————————————————*
 * Now that we're connected to the database, execute our queries *
 * ———————————————————————*/

/* Do a select query */
printf("Executing query:\n\"%s\"\n", query_select);
if (select_funct(mysql, query_select) == -1)
{
        fprintf(stderr, "\nERROR in select_funct, exiting prematurely.\n\n");
        return (EOF);
}

/* Do an insert query */
printf("\nExecuting query:\n\"%s\"\n", query_update);
if (update_funct(mysql, query_update) == -1)
{
        fprintf(stderr, "\nERROR in update_funct, exiting prematurely.\n");
        return (EOF);
}

/* Do another select query, showing info has been inserted */
printf("\n\nExecuting query:\n\"%s\"\n", query_select);
if (select_funct(mysql, query_select) == -1)
{
        fprintf(stderr,"\nERROR in select_funct, exiting prematurely.\n\n");
        return (EOF);
}

/* Do an update query to erase all references of M60's */
printf("\nExecuting query:\n\"%s\"\n", query_delete);
if (!(update_funct(mysql, query_delete)))
{
        fprintf(stderr,"\nERROR in update_funct, exiting prematurely.\n\n");
        return (EOF);
}


/*———————————————————*
 * All done, close connection to MySQL *
 *———————————————————*/
#ifdef DEBUG
printf("\nDebug: closing MySQL connection.\n");
#endif
mysql_close(&mysql);
```

```
        printf("\n"); /* give the user an extra space, it looks nice */
        return 0;
}
```

B gt2import.py

```
\begin{verbatim}
#! /usr/bin/env python
# Usage: gt2import [file(s) in] (must not include path)
# $Id: gt2import.py,v 1.5 1999/07/26 19:01:43 frost Stab $

"""Takes ground truth formated file and outputs file for import todatabase

Usage: gt2import [file(s) in] (must not include path)

About:
Data converter, takes file stored in ground truth version two file format,
parses the file, performs various queries to gather information on the current
target, and does what is necessary to load the frame info into the database.

Originally written in 1999 by Chris Frost (<chris@frostnet.advicom.net>) under
the Science and Engineering Apprenticeship Program.

This program assumes that the first data entires bit depth is the same for the
entire sequence; the depression and aspect information in the header of
input files is ignored; and, does not check to see if data being loaded into
the current frame table has been loaded before.
"""

from string import split, atoi, upper
import sys, posix, os
from MySQL import *

def get_files():
        """Associate from which and to which files we should read and write"""
        global infile, base_outfile, outfile_target, outfile_sequence_location

        # If we are ran from the command line, look for paramaters
        if __name__ == '__main__':
                iteration = 2
                while iteration <= len(sys.argv):
                        infile = sys.argv[iteration - 1]
                        base_outfile = infile + '.'
                        outfile_target = base_outfile + 'target.insert'

                        print 'Processing', infile
```

```
                process_file()

                iteration = iteration + 1
        # If no command line arguments, ask what file we should read
        if len(sys.argv) == 1:
                infile=input('File from which we should import data? ')
                base_outfile = infile + '.'
                outfile_target = base_outfile + 'target.insert'
                outfile_sequence_location = base_outfile + \                    50
                'sequence_location.insert'

                print 'Processing', infile

                process_file()

def check_consistency(line, filename, keywords, values):
        """Check number of keywords versus number of values to check format"""
        if keywords != values:
                print ""                                                        60
                print "+-------------------------==========-",
                        print "-------------------"
                print "| Error: This file is not gt2 compliant."
                print "| "
                print "| In line",line,"of",filename,"there were",values
                print "| entries while I was expecting", keywords, "."
                print "| Please manually inspect this file and correct it."
                print "+-------------------------------------------------",
                        print "-------------------"
                print ""                                                        70
                raise FileConsistencyError, 'Please manually inspect this file and correct it.'

def get_bytes_pixel():
        """Opens a copy of current input and reads first data row's b/p entry"""
        input = open(infile, 'r')
        line = input.readline()
        line = input.readline()
        values = split(line)
        number_targets = atoi(values[0])
                                                                                80
        # Skip over references per target
        iteration = 1
        while iteration <= number_targets:
                line = input.readline()
                iteration = iteration + 1

        # Go forward one and skip fields names
        line = input.readline()
        line = input.readline()
```

```
                                                                        90
        # Skip over generic taget info
        iteration = 1
        while iteration <= number_targets:
                line = input.readline()
                iteration = iteration + 1

        # Skip over field summary and field names
        line = input.readline()
        line = input.readline()
                                                                        100
        values = split(line)
        bytes_pixel = values[19]

        input.close()

        return bytes_pixel

def process_file():
        """Step through current infile line by line and parse"""
                                                                        110
        input = open(infile, 'r')

        tab = ' '
        line = ' '
        current_file_line = 0
        list_targets = []
        targets_references = []
        global sequence_file
        print '\nRetrieving list of known targets.'
        known_targets = get_known_targets()                             120
        print 'List retrieved and saved.'

        # Loop through until reaching EOF
        while line != '':
                line = input.readline()
                current_file_line = current_file_line + 1

                # Seqence filename, path, and time recorded
                if current_file_line == 1:
                        print "\nParsing Header Information . . ."       130

                        bytes_pixel = get_bytes_pixel()

                        values = split(line)
                        sequence_file = values[0]
                        load_sequence_id(bytes_pixel)

                        sequence_id = get_sequence_id()
```

```
        # Extract time sequence was recorded from sequence file
        # this assumes the time is the next to last '_' entry
        values = split(line, '_')
        time_recorded = values[-2]

# Number of targets
elif current_file_line == 2:
        values = split(line)
        number_targets = atoi(values[0])
        iteration = 0
        while iteration <= atoi(values[0]) - 1:
                list_targets[0:0] = [iteration]
                iteration = iteration + 1
        list_targets.reverse()

# Number of references for each target
elif current_file_line == 3:
        target_ref_dict = {}
        iteration = 1
        for iteration in list_targets:
                values = split(line)
                targets_references[0:0] = [atoi(values[0])]

                line = input.readline()
                current_file_line = current_file_line + 1
                iteration = iteration + 1

        targets_references.reverse()

        check_consistency(current_file_line, infile,
        len(list_targets), len(targets_references))

        # Create dictionary of target number and # of references
        # Robertson suggested using this instead of map (below)
        for iteration in list_targets:
                target_ref_dict[list_targets \
                [list_targets.index(iteration)]] \
                = targets_references[list_targets.index \
                (iteration)]

        #for target_num, references_num in \
        #map(None, list_targets, targets_references):
                #target_ref_dict[target_num] = references_num

        for iteration in list_targets:
                print 'Target', iteration, 'has', \
                target_ref_dict[iteration], 'references.'
```

```
# Target specific header info
elif current_file_line == 4 + number_targets:
        iteration = 1                                                    190
        target_ids = []
        targets_proc = []

        while iteration <= number_targets:
                values = split(line)
                keywords = ['target', 'class', 'type',
                'variant', 'specific', 'aspect', 'depression',
                'priority', 'name']

                check_consistency(current_file_line, infile,             200
                len(values), len(keywords))

                target_id_dict = {}
                for value, keyword in \
                map(None, values, keywords):
                        target_id_dict[keyword] = value

                # Convert priority to integer
                target_id_dict['priority'] = \
                        upper(target_id_dict['priority'])                210
                if target_id_dict['priority'] == 'PRIMARY':
                        target_id_dict['priority'] = '1'
                elif target_id_dict['priority'] == 'SECONDARY':
                        target_id_dict['priority'] = '2'
                elif target_id_dict['priority'] == 'TERTIARY':
                        target_id_dict['priority'] = '3'
                elif target_id_dict['priority'] == \
                                                'QUATERNARY':
                        target_id_dict['priority'] = '4'
                else:                                                    220
                        raise FileConsistencyError, 'This file has a target
                        of lower priority than quaternary,
                        which I do not currently know how
                        to deal with.  Please correct me
                        to process this file'

                # Save target identification string to list
                target_ids[0:0] = [target_id_dict['class']+\
                '_' + target_id_dict['type'] + '_' + \
                target_id_dict['variant'] + '_' + \                      230
                target_id_dict['specific']]

                target_code = \
                [atoi(target_id_dict['class']),
                atoi(target_id_dict['type']),
                atoi(target_id_dict['variant']),
```

```
                atoi(target_id_dict['specific'])]

                # If we have already processed an  identical
                # target, break                                          240
                if target_code in targets_proc:
                        print 'Target "', target_code, \
                        '" was processed earlier during this session,
                                skipping.'
                        break
                else:
                        targets_proc[-1:-1]= [target_code]

                # Check to see if we know about this target
                if not target_code in known_targets:                   250
                        print '\n-------------------------'
                        print 'Target Code:', target_code
                        print 'New target encountered. Saving target
                                summary information and inserting'
                        print 'into database for future use.'
                        print ""

                        insert_new_target_info(\
                        target_id_dict['name'],
                        target_id_dict['class'],                       260
                        target_id_dict['type'],
                        target_id_dict['variant'],
                        target_id_dict['specific'],
                        target_id_dict['priority'])
                        print 'Database now updated, resuming operations.'
                        print '-------------------------\n'

                iteration = iteration + 1
                line = input.readline()
                current_file_line = current_file_line + 1              270

        target_ids.reverse()

        input.readline() # Skip over row containing fields names

else: # we are in the main 'data' of the file
        print "\nParsing and Loading Data Rows . . ."

        keywords = ['target', 'index', 'frame', 'range',
        'ul_x', 'ul_y', 'aim_x', 'aim_y', 'size_x', 'size_y',          280
        'patch_max', 'patch_min', 'patch_mean', 'time_gt',
        'time_modified', 'aspect', 'depression',
        'scm', 'b_mean', 'bytes_pixel', 'patch_edge_offset_top',
        'patch_edge_offset_bottom', 'patch_edge_offset_left',
        'patch_edge_offset_right']
```

```
frame_row_dict = {}
target_id_last = 0
id = '\N' # auto_incremented by MySQL
                                                                    290
for current_target in list_targets:
        print 'Current target:', current_target, \
        '(with', number_targets - current_target -1, \
        'remaining)'

        outfile = base_outfile + 'frame' + '-' + \
        target_ids[current_target] + '.insert'

        if target_ids[current_target] == target_id_last:
                output = open(outfile, 'a')                         300
        else:
                output = open(outfile, 'w')

        target_id_last = target_ids[current_target]

        cycle = 1

        while cycle <= target_ref_dict[current_target]:
                print target_ref_dict[current_target] \
                - cycle,                                            310

                values = split(line)

                check_consistency(current_file_line, \
                infile, len(keywords), len(values))

                for iteration in keywords:
                        frame_row_dict[keywords \
                        [keywords.index(iteration)]] \
                        = values[keywords.index \                   320
                        (iteration)]

                output.write(id + tab + \
                time_recorded + tab + \
                frame_row_dict['time_gt'] + tab + \
                frame_row_dict['time_modified'] + tab +\
                frame_row_dict['aspect'] + tab + \
                frame_row_dict['depression'] + tab + \
                sequence_id + tab + \
                frame_row_dict['index'] + \                         330
                tab + frame_row_dict['frame'] + \
                tab + frame_row_dict['aim_x'] + tab + \
                frame_row_dict['aim_y'] + tab + \
                frame_row_dict['ul_x'] + tab + \
```

```
                frame_row_dict['ul_y'] + tab + \
                frame_row_dict['size_x'] + tab + \
                frame_row_dict['size_y'] + tab + \
                frame_row_dict['range'] + tab + \
                frame_row_dict['patch_min'] + tab + \
                frame_row_dict['patch_max'] + tab + \                    340
                frame_row_dict['patch_mean'] + tab + \
                frame_row_dict['b_mean'] + tab + \
                frame_row_dict['patch_edge_offset_top']\
                + tab + \
                frame_row_dict\
                ['patch_edge_offset_bottom'] + tab + \
                frame_row_dict\
                ['patch_edge_offset_left'] + tab + \
                frame_row_dict\
                ['patch_edge_offset_right'] + tab + \                    350
                frame_row_dict['scm'] + '\n')

                line = input.readline()
                current_file_line= current_file_line+ 1
                # Ugly, can we do better?
                if line == '':
                        break
                cycle = cycle + 1
        print '\n'
        output.close()                                                   360

        # MySQL wants the full path to the file it will
        # be loading
        full_path_outfile = os.getcwd() + '/' + outfile

        load_frame_rows(full_path_outfile, \
        target_ids[current_target])

        # Now that we have loaded the data, erase the
        # intermediate file                                              370
        string = 'rm -f ' + outfile
        print string
        posix.system(string)
        print 'File loaded and intermediate file erased, continuing.\n'
print '+-------------------------------------------------+'
print '|--- File parsing and loading complete ---|'
print '+-------------------------------------------------+',
input.close()

def get_known_targets():                                                 380
    """Retrieve list of known targets"""
    DBH = connect_database()
    print 'Connection Info:', DBH.hostinfo()
```

```
        query = 'SELECT class, type, variant, specific FROM target'
        query_results = DBH.do(query)
        known_targets = query_results

        return known_targets
                                                                        390
def load_frame_rows(frame_filename, target_id):
        """Load rows of data in db from outputed file"""
        DBH = connect_database()
        print 'Connection Info:', DBH.hostinfo()

        query = 'LOAD DATA INFILE "' + frame_filename + '" INTO TABLE
                frame_' \ + target_id
        print query

        STH = DBH.query(query)                                          400

def insert_new_target_info(name, tclass, type, variant, specific, priority):
        """Insert new target info and create appropriate frame table"""
        DBH = connect_database()
        print 'Connection Info:', DBH.hostinfo()

        # Insert generic target info into table target
        query = \
        'INSERT INTO target (name, class, type, variant, specific, priority) VALUES ('\
        + '\"' + name + '\"' + ', ' + tclass + ', ' + type + ', ' + variant \          410
        + ', ' + specific + ', ' + priority + ')'
        print 'Query:\n', query, '\n'

        STH = DBH.query(query)

        # Create a frame target table, will fail if table already exists
        print 'Creating new frame table for this target type...'
        table_attributes = '( id int unsigned not null auto_increment, primary key (id),
        time_record time not null, index time_record_index (time_record),
        time_gt datetime, time_modified timestamp, aspect double, depression double,   420
        sequence_id mediumint unsigned, findex mediumint unsigned, frame smallint
        unsigned, aim_x smallint unsigned, aim_y smallint unsigned, ul_x smallint
        unsigned, ul_y smallint unsigned, size_x smallint unsigned, size_y smallint
        unsigned, range smallint unsigned, patch_min double, patch_max double,
        patch_mean double, b_mean double,  patch_edge_offset_top smallint,
        patch_edge_offset_bottom smallint, patch_edge_offset_left smallint,
        patch_edge_offset_right smallint, scm double )'
        query = 'CREATE TABLE frame_' + tclass + '_' + type + '_' + \
        variant + '_' + specific + '\n' + table_attributes
        print query, '\n'                                               430

        STH = DBH.query(query)
```

```
def load_sequence_id(bytes_pixel):
        """Insert sequence filename into database"""
        DBH = connect_database()
        print 'Connection Info:', DBH.hostinfo()

        # Check to make sure this file doesn't already have an entry
        query = 'SELECT file FROM sequence_location WHERE file ="' + \                  440
                sequence_file +'"'
        query_results = DBH.do(query)
        if query_results:
                # If this sequence file already exists don't insert it again
                pass
        else:
                query = 'INSERT INTO sequence_location (file, bytes_pixel) VALUES \
                (' + '\'' + sequence_file + '\'' + ', ' + bytes_pixel + ')'
                print 'Query:\n', query, '\n'
                                                                                       450
                STH = DBH.query(query)

def get_sequence_id():
        """Retrieves sequence identification number"""
        DBH = connect_database()
        print 'Connection Info:', DBH.hostinfo()

        query = 'SELECT id FROM sequence_location WHERE file = "' + \
                sequence_file + '"'
        print 'Query:\n', query                                                        460
        query_results = DBH.do(query)
        print 'Sequence ids that match our current file:', query_results
        sequence_id = query_results[0]
        # Turn our query results into a single number, not a list
        sequence_id_number = sequence_id[0]
        return sequence_id_number

def connect_database():
        """Connect to the database and return DBH"""
        database = 'database'                                                          470
        host = 'localhost'
        user = 'user'
        password = 'password'
        print 'Connecting to', database, 'database, hosted on', host, '. . .'
        DBH = connect(host, user, password) #()->localhost,this user,no passwd

        DBH.selectdb(database)

        return DBH
                                                                                       480
get_files()
```

\end{verbatim}

C SQL

C.1 Target table

```
CREATE TABLE target /* generic target information */
(
id smallint unsigned not null auto_increment, /* unique identifier, 2^16-1 */
primary key (id), /* setup primary key */
name varchar(80), /* name of target, 80 characters max (gt2 limit) */
class int unsigned, /* class number, 2^32-1 (gt2 limit) */
type int unsigned, /* type of class, 2^32-1 (gt2 limit) */
variant int unsigned, /* variant of type, 2^32-1 (gt2 limit) */
specific int unsigned, /* specific of variant, 2^32-1 */
priority tinyint /* priority of variant, 2^8-1 (+/-) */
);
```

C.2 Sequence location table

```
CREATE TABLE sequence_location /* table which holds actual filename for seqs */
(
id mediumint unsigned not null auto_increment, /* unique identifier, 2^24-1*/
primary key (id), /* setup primary key */
file tinytext, /*filename and path as text, limited to 2^16-1 ascii characters*/
bytes_pixel tinyint unsigned /* bytes per pixel, 2^8-1 */
);
```

C.3 Frame table

```
CREATE TABLE frame_<id> /* info on each frame, with a seperate table for each
        * unique target.id */
(
id int unsigned not null auto_increment, /* unique identifier, 2^32-1 */
primary key (id), /* setup primary key */
time_record time not null, /* time sequence was recorded */
index time_record_index (time_record), /*index time_record for speedy searches*/
time_gt datetime, /* time frame was first ground truthed */
time_modified timestamp, /* time frame last modified */
aspect double, /* aspect of target, double precision */
depression double, /* angle of depression of target */ /* CHECK: in
                * sequence_location? */
```

```
sequence_id mediumint unsigned, /* reference to sequence_location.id */
findex mediumint unsigned,/* index in gt2, uses findex b/c of keywords, 2^24-1*/
frame smallint unsigned, /* frame in sequence, 2^16-1 */
aim_x smallint unsigned, /* X aimpoint, 2^16-1 */
aim_y smallint unsigned, /* Y aimpoint, 2^16-1 */
ul_x smallint unsigned, /* upper left X coordinate for patch, 2^16-1 */
ul_y smallint unsigned, /* upper left Y coordinate for patch, 2^16-1 */
size_x smallint unsigned, /* size of patch along X, 2^16-1 */                    20
size_y smallint unsigned, /* size of patch along Y, 2^16-1 */
range smallint unsigned, /* range to target in meters, 2^16-1 */
patch_min double, /* minimum value of patch pixel values */
patch_max double, /* maximum value of patch pixel values */
patch_mean double, /* mean of patch pixel values */
b_mean double, /* mean of background (non-patch) pixel values */
patch_edge_offset_top smallint, /* for use in cad overlay */
patch_edge_offset_bottom smallint, /* for use in cad overlay */
patch_edge_offset_left smallint, /* for use in cad overlay */
patch_edge_offset_right smallint, /* for use in cad overlay */                   30
scm double /* signal to cluter ratio */
);
```

C.4 Image table

```
CREATE TABLE image /* holds actual images, for possible future use */
(
id integer unsigned not null auto_increment, /* unique identifier, 2^32-1 */
primary key (id), /* setup primary key */
frame_table varchar(100), /* to which table the frame_id belongs.
        * format: frame_t<id> */
frame_id integer unsigned, /* store to which frame we belong, do we need this?*/
image mediumblob /* binary image, max 2^24-1 bytes */
);
```

References

[1] D. Casasent and D. Psaltis, "Position, rotation, and scale invariant optical correlation," *Applied Optics*, vol. 15, p. 1795, 1976.

[2] Y.N. Hsu and H.H. Arsenault, "Optical pattern recognition using the circular harmonic expansion," *Applied Optics*, vol. 21, p. 1699, 1982.

[3] D. Casasent, "Unified synthetic discriminant function computational formulation," *Applied Optics*, vol. 23, p. 1620, 1984.

[4] A.B. VanderLugt, “Signal detection by complex matched spatial filtering,” *IEEE Trans. Inf. Theory*, vol. IT-10, p. 139, 1964.

[5] K. King, *C Programming: A Modern Approach*. W.W. Norton & Company, 1996.

[6] B. W. Kernighan and D. M. Ritchie, *C Programming Language*. Prentice Hall, 2 ed., 1988.

[7] S. McConnell, *Code Complete*. Microsoft Press, 1993.

[8] S. Navathe, S. B. Navathe and R. Elmasri, *Fundamentals of Database Systems*. Addision-Wesley Pub Co, 2 ed., 1994.

[9] M. Lutz, *Programming Python*. O’Reilly & Associates, 1996.