Parallel Tracking and Multiple Mapping (PTAMM) Manual

Source Code Release v1.3

Robert Castle

Robotics Research Group
Department of Engineering Science
University of Oxford

November 2010

http://www.robots.ox.ac.uk/ActiveVision

1 Introduction

This software is an implementation of the method described in the paper *Video-rate Localization in Multiple Maps for Wearable Augmented Reality* by Robert Castle, Georg Klein and David Murray, which appeared in the proceedings of the IEEE International Symposium on Wearable Computers (ISWC) 2008.

This release is aimed at experienced software developers and/or researchers familiar with implementing real-time vision algorithms in C++ on the platform of their choice.

PTAMM is based on the PTAM code (forked from r112) and it is recommended that you gain familiarity with PTAM before delving into PTAMM.

Questions? Email `ptam@robots.ox.ac.uk` or visit the blog at `http://ewokrampage.wordpress.com`.

2 PTAMM's Enhancements Over PTAM

PTAMM provides the following enhancements:

- Multiple maps with automatic switching
- Saving and loading to and from disk
- Video Output
- New game architecture to add your own games.
- Two new games

3 Requirements

3.1 Supported Operating Systems

The code was developed on x86/x86-64 Linux, and it is the recommended OS for use with PTAMM. PTAMM will also compile and run under Mac OSX (using X11 for the display) and Microsoft Windows. Linux provides the best performance and experience, followed by OSX, once bent to your will. Windows will make you cry.

3.2 Processor Requirements

The software runs at least two processing intensive threads at the same time and so needs at least a dual-core machine. Intel Core 2 Duo processors 2.4GHz+ are fine.

3.3 Graphics

The software requires accelerated OpenGL graphics output. It has been written and tested only with nVidia cards: this is primarily of concern under Linux, where the use of an nVidia card and the proprietary nVidia display drivers are highly recommended. Since the Linux code compiles directly against the nVidia driver's GL headers, use of a different GL driver may require some modifications to the code.

3.4 Video Input

The software requires a video camera with a wide-angle lens, capable of $640 \times 480 \times 30$Hz video capture and an appropriate driver installation (which is supported by libCVD.) Only monochrome capture is needed for tracking, colour images are used only for the AR display. A further discussion of video input follows later in this file.

3.5 Video Output

This is only supported under Linux, as the method uses FIFOs to pass out the video data. To be able to save videos mencoder must be installed. Under Linux mencoder can usually be installed from your distributions package repositories or downloaded from `http://www.mplayerhq.hu`. Under Ubuntu do:

```
$> sudo apt-get install mencoder
```

Videos are saved to the current directory and automatically named using the current date and time in the format **YYYY-MM-DD_hh-mm-ss.avi**, e.g. **2009-12-15_21-31-38.avi**.

3.6 Libraries

The software has four principal dependencies:

1. TooN - a header library for linear algebra.
2. libCVD - a library for image handling, video capture and computer vision.
3. Gvars3 - a runtime configuration/scripting library, this is a subproject of libCVD.
4. lib3ds - a library for handling 3DS model files.

The first three above are written by members of the Cambridge Machine Intelligence lab and are licensed under the LGPL. Current versions are available from Savannah via CVS:

- `http://savannah.nongnu.org/projects/toon` for TooN.
- `http://savannah.nongnu.org/projects/libcvd` for libCVD and GVars.

The latest version of these libraries can be obtained via CVS and ssh:

```
$> export CVS_RSH=ssh
$> cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/toon co TooN
$> cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/libcvd co libcvd
$> cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/libcvd co gvars3
```

It should be noted, however, that the libraries change rapidly. To ensure compatibility, it may be useful to download the libraries corresponding to a time at which they were known to be compatible. To do so, use the following commands:

```
$> export CVS_RSH=ssh
$> cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/toon co -D "Mon
    May 11 16:29:26 BST 2009" TooN
$> cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/libcvd co -D "Mon
    May 11 16:29:26 BST 2009" libcvd
$> cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/libcvd co -D "Mon
    May 11 16:29:26 BST 2009" gvars3
```

The lib3ds library can be downloaded from `http://www.lib3ds.org`. The version used during development was `lib3ds-20080909.zip`.

4 Installation - Linux

4.1 Installation of the dependencies

The four dependent libraries are all compiled and installed using the familiar `./configure; make; make install` system, however the following points are worth noting. On Linux, the following libraries (and their devel versions) are required:

- blas,
- lapack,
- libgfortran,
- ncurses,
- libreadline,
- libdc1394 (and maybe libraw1394) for Firewire capture,
- libpng.
- Optionally libtiff and libjpeg.

The order of installation should be

1. TooN
2. libCVD
3. GVars3
4. lib3ds

TooN installation is trivial, since it's only a set of headers. Just do

```
$> ./configure; make; make install
```

For libCVD, I recommend the following configure options:

```
$> export CXXFLAGS=-D_REENTRANT
$> ./configure --without-ffmpeg
```

If the compiler hangs on one of the FAST detectors these can be disabled with `./configure disablefast7` for example.

Documentation can be generated (if doxygen is installed) by running

```
$> make docs
```

For GVars3, I recommend the following configure options:

```
$> ./configure --disable-widgets
```

Installing lib3ds should also be straight forward with a

```
$> ./configure; make; make install
```


4.2 Compiling the Software

The source code is in the PTAMM directory. The first step is to copy the appropriate platform build files to the PTAMM source directory. E.g. for Linux, copy all the files from `PTAMM/Build/Linux` to `PTAMM`. The Makefile can then be edited to reference any custom include or linker paths which might be necessary (depending on where the dependencies were installed.)

The second step, for Linux, is to set up the correct video source. Two files are provided, **VideoSource_Linux_DV.cc** and **VideoSource_Linux_V4L.cc**, which work with the Unibrain Fire-i and the Logitech Quickcam Pro 5000 respectively. The DV version is compiled by default; edit the Makefile to switch to the V4L version instead, if needed. Other cameras may require manual editing of the video input files, e.g. to change the videobuffer's colourspace. Other video source classes are available with libCVD.

Finally, if a custom video source not supported by libCVD is required, the code for it will have to be put into some **VideoSource_XYZ.cc** file (the interface for this file is very simple.)

If you do not want to use the lib3ds based AR game, then the Makefile can be simply modified to remove the associated files, and lib3ds will then not be required.

The software can then be compiled with the command

```
$> make
```

This builds two target executables: PTAMM and CameraCalibrator.

5 Installation - Apple Mac OSX

Installation of PTAM, and thus PTAMM, on OSX is a little more complicated than on Linux. If you are using 10.5 I believe it is all very straight forward. However, if you are using 10.6 with its default 64-bit building then it requires a little taming. These instructions are for 10.6 only, but you should be able to see what to do for 10.5.

The install instructions assume that you already have a build system set up on OSX, i.e. you have XCode, and have a way of getting open source software onto your machine, such as with MacPorts or Fink. The installation procedure is very similar to that of Linux, however, there are a few gotchas, that you will need to deal with. The step-by-step install guide at the end details these.

5.1 Installation of the dependencies

The installation of the dependencies is virtually the same as for Linux. If you are using Mac Ports or Fink it should be straightforward to get the required dependencies. Make sure you are building universal builds, so that you have 32-bit versions along with the default 64-bit versions.

The installation of the Cambridge libraries should be straight forward on 10.5 or earlier. However on 10.6 there are issues with getting the libraries to build 32-bit versions. To aid you with this a bash script has been included with PTAMM to make sure the configure scripts for libCVD, GVars, and lib3ds set up up a 32-bit 10.5 build. To use this bash script, copy the `configure-10.5-32bit` file from the `PTAMM/Build/OSX` directory into the libCVD, GVars, and lib3ds directories, and then run that instead of configure,

```
$> ./configure-10.5-32bit; make; make install
```


5.2 Compiling the Software

The source code is in the PTAMM directory. The first step is to copy the appropriate platform build files to the PTAMM source directory. E.g. for OSX, copy all the files from `PTAMM/Build/OSX` to `PTAMM`. The Makefile can then be edited to reference any custom include or linker paths which might be necessary (depending on where the dependencies were installed.)

If you do not want to use the lib3ds based AR game, then the Makefile can be simply modified to remove the associated files, and lib3ds will then not be required.

The software can then be compiled with the command

```
$> make
```

This builds two target executables: PTAMM and CameraCalibrator.

6 Installation - Microsoft Windows

Installation of PTAM, and thus PTAMM, on Windows is rather more temperamental than on Linux. Some people have no problems installing PTAM on Windows and others have great difficulty, but at different points. Here a summary is provided and at the end of the document a step by step installation is shown.

This summary and the example installation at the end have been tested on 32-bit Windows XP Media Centre Edition with all updates applied. Visual Studio 2005 is used along with Windows SDK 6.1.

6.1 Installation of the dependencies

Before installation of PTAM, PTAMM or the Cambridge libraries the following dependencies are required. The files used are shown in brackets :

- Lapack and BLAS - available pre-compiled from `http://www.fi.muni.cz/~xsvobod2/misc/lapack` (lapack-MT-release.zip)
- pthreads from `http://sourceware.org/pthreads-win32` (pthreads-w32-2-8-0-release.exe)
- GLEW from `http://glew.sourceforge.net` (glew-1.5.5-win32.zip)
- CMU1394 camera driver from `http://www.cs.cmu.edu/~iwan/1394` (1394camera645.exe)
- libjpeg for win32 from `http://gnuwin32.sourceforge.net/packages/jpeg.htm` (jpeg-6b-4.exe)
- lib3ds from `http://sourceforge.net/projects/lib3ds` (lib3ds-20080909.zip),
- libpng from `http://gnuwin32.sourceforge.net/packages/libpng.htm` (libpng-1.2.37-setup.exe)
- zlib from `http://zlib.net` (zlib125.zip)

WARNING: The CMU 1394 drivers *only* work on 32-bit Windows. I tried to get PTAM running on 64-bit Windows 7 Professional with a Unibrain Firewire camera, but no luck. Advice welcome.

LibCVD, TooN, and GVars3 can be obtained from CVS as above. However this install guide uses different versions, as they were found to work on the machines used with fewer issues. Finding the right combination of the Cambridge libraries for your machine is the largest hurdle. To compile and install, libCVD and GVars3 come with msdev project files, TooN is a group of headers which can be copied directly into your include directory.

To be able to use the 3ds Models Game, the preprocessor definition `ENABLE_MODELS_GAME` needs to be declared in the PTAMM project properties (as well as installing lib3ds). In the PTAMM project properties go to *Configuration Properties* > *C/C++* > *Preprocessor*. Then append `ENABLE_MODELS_GAME` to the Preprocessor Definitions entry

The order of installation should be

1. TooN 2 from `http://mi.eng.cam.ac.uk/~er258/cvd/toon.html` (TooN-2.0.beta7.zip)
2. libCVD from `http://mi.eng.cam.ac.uk/~er258/cvd/index.html` (libcvd-20100511.zip)
3. GVars3 from `http://mi.eng.cam.ac.uk/~er258/cvd/gvars3.html` (gvars3-20090421.tar.gz)
4. lib3ds

6.2 Compiling the Software

The source code is in the PTAMM directory. The first step is to copy the appropriate platform build files to the PTAMM source directory. For Windows, copy all the files from `PTAMM/Build/Win32` to `PTAMM`. Open the `PTAMM.sln` file. The solution contains projects to build the Camera Calibrator and PTAMM. It will have the CMU video interface (`VideoSource_Win32_CMU1394.cc`) already in the project. If you want to use a different video interface then you will need to add it to the project. Edit the project properties to reflect the location of your lib and include dirs, and possibly the names of the libraries you are linking against.

7 Running The Software

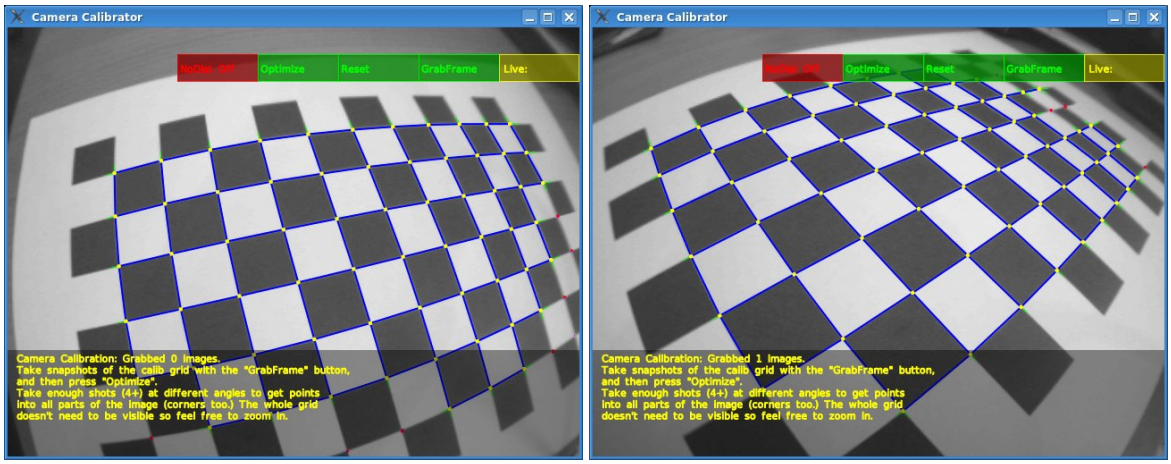
7.1 Calibrating the Camera

CameraCalibrator should be run first to obtain a camera calibration (and to verify that video input is in fact working.) This requires the user to point the camera at a checker board calibration pattern; any checker board of any size will do, a sample is included as **calib_pattern.pdf**.

The camera calibrator attempts to find square corners in the image, and then to link these together. This is indicated by fragments of squares appearing in the image. The calibrator is not very robust; If no squares are detected, some things to try would be:

- Modify camera settings to remove any sharpening; Sharpening artifacts break the calibrator, and also reduce performance of the tracker.
- Adjust the calibrator's settings, for example increase the value of `CameraCalibrator.BlurSigma`

When the camera is in a pose in which some portions of the grid are detected, the user should press the 'GrabFrame' button to add a snapshot of that frame to the optimiser. After



(a) **(b)**

Figure 1: Camera Calibration.

a few frames from different poses have been added, pressing 'Optimize' button iteratively calculates the camera parameters. When the user is satisfied with convergence (the RMS error should be no more than around 0.3 pixels) pressing 'Save' stores the camera calibration in a file **camera.cfg**. Fig. 1 shows the **CameraCalibrator** in use.

7.2 Running the Tracker

Once a calibration has been stored, invoking **PTAMM** runs the tracker. At the start, the tracker requires the user to provide a stereo pair to initialise the map. The user does this by pointing the camera at an angle to a planar (or near-planar) surface to be augmented, pressing spacebar, slowly translating the camera to provide a baseline, and pressing spacebar again to complete the stereo pair. At this point a map is created and the tracker runs. Augmented graphics can be shown once the tracker is running by first pressing the 'Draw AR' toggle button, then selecting a game from the Demos menu. Fig. 2 shows PTAMM running.

If there appear to be problems with the map not being expanded or not being tracked well, the most likely culprit is a lack of baseline, i.e. the camera was not translated enough. A stereo initialisation with only rotation does not provide a baseline and cannot ever work.

N.b. you don't need the calibration grid for the tracker! Any planar or near-planar textured scene will do.

7.3 General Use of GVars

Both programs rely on GVars for a console user interface. In the terminal window, a user may inspect a list of all tweakable variables by typing

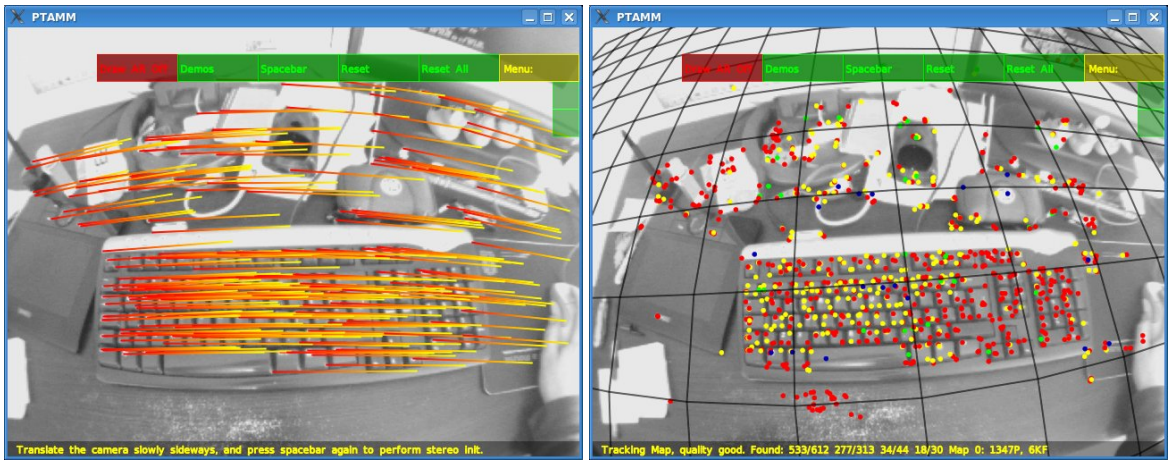
```
> gvarlist
```

or for a more complete list

```
> gvarlist a
```

and then modify variables by typing

```
> Variable_Name = new_value
```

(a) Initial tracking **(b)** Map created and tracking

Figure 2: PTAMM running.

This allows the user to tweak a number of the program's features; for example, if the quality of video input is dubious, typing `DrawFASTCorners=1` in the tracker allows the user to inspect the response of the FAST corner detector.

8 Video Sources

The software was developed with a Unibrain Fire-i colour camera, using a 2.1mm M12 (board-mount) wide-angle lens. It also runs well with a Logitech Quickcam Pro 5000 camera, modified to use the same 2.1mm M12 lens.

Wide-angle lenses are important for natural feature trackers and SLAM systems, and the software's performance using a zoomier lens is likely to be compromised. It will require more keyframes, robustness to rotation will decrease, and relocalizer performance will drop.

How wide is wide-angle? The first number in **camera.cfg** is a normalized horizontal focal length (fx): for our wide-angle lenses $\text{fx} = 0.57$. Up to $\text{fx} \leq 1.0$ the system would be expected to run fine, beyond that performance will likely be degraded.

Independent of OS, it is important to turn down in-camera sharpening to the point that no sharpening artifacts appear! Sharpening artifacts produce spurious corners from image noise, move the location of real corners, and break scale and rotation invariance. Expect poor performance if running on an oversharpened image. For example, on the Unibrain Fire-i, turn sharpness down from the default 80 to 25.

Linux Fire-i notes: Firewire with libCVD has been tested against the old-style (pre-juju) Firewire stack, using either the libDC-1 or libDC-2 series. If your distribution ships with Firewire support in the form of the experimental juju stack (e.g. Fedora 8 and 9) you may experience video input lock-ups, you should install packages for the old Firewire stack instead.

Linux Logitech Quickcam notes: the Logitech Quickcam pro 5000 is supported by the Linux UVC driver and can be used with a `CVD::V4LBuffer<yuv422>`.

MacOS X notes: Video input properties are reset every time the software is run, and so settings will have to be adjusted every time (e.g. turning down in-camera sharpening.)

Windows notes: If attempting to use on Windows, note that we have experienced poor performance in conjunction with DSVL: this seems to be a threading issue, as the video source

also uses a thread, which can get starved. Using the Unibrain or Point Grey APIs as appropriate would be preferable. Note also that YUV >RGB >greyscale produces notable artifacts as opposed to direct YUV >greyscale. At the moment, only a CMU1394 interface is included, this works fine.

9 New Features in PTAMM

The changes to the base PTAM code vary from new classes, to changes to the some of the PTAM classes. A diff of PTAM r112 and this will show what has been changed.

9.1 Multiple Maps

The user can now create multiple maps around an environment, and have the system automatically switch between them as they move from one to the other. See `http://www.robots.ox.ac.uk/~bob/software` for example usage. The automatic switching is done using the relocalizer to detect which map the camera is now in. The relocalizer now searches all maps for matching keyframes. If a different map is required it issues a GVar command `SwitchMap $mapnum` which is picked up by `System`. `System` then orchestrates the switching to the new map.

Creating a new map causes `System` to move all elements (tracker, map maker, map viewer) to the new map. Additionally maps can be individually locked from editing (new keyframe addition, point addition and removal, no bundle adjustments). The map viewer can be used to view the current map and automatically switches when the camera moves to a new map. It can also used to browse the maps. Users can reset all of the maps taking the system back to its original starting status, or reset only the current map. Maps can also be deleted. Fig. 3 shows an example of multiple maps and recovery.

9.2 Saving and Loading

Users can now save the state of a map to disk, and then later reload it. Maps are saved in an open format using XML for the map data, and PNG images for the keyframes. Only the salient data is saved, some is regenerated on loading (such as FAST corners).

A new `MapSerializer` thread is used to serialize the maps. This utilizes tinyXML[1] (included with the source code) and libCVD. A new class to handle thread synchronization issues, `MapLockManager`, is added to the `Map` class. This is used to enable the serializer to gain complete control of a map, so other threads cannot corrupt the data during saving or loading. All threads now have to register to each map and unregister when leaving. `System` and `MapMaker` check for a complete lock on a map before continuing their operations. If a map is locked they will signal acknowledgement and wait until the map is unlocked. The key advantage is that multiple maps can be saved in the background, with only a brief pause in operations in the current map during saving.

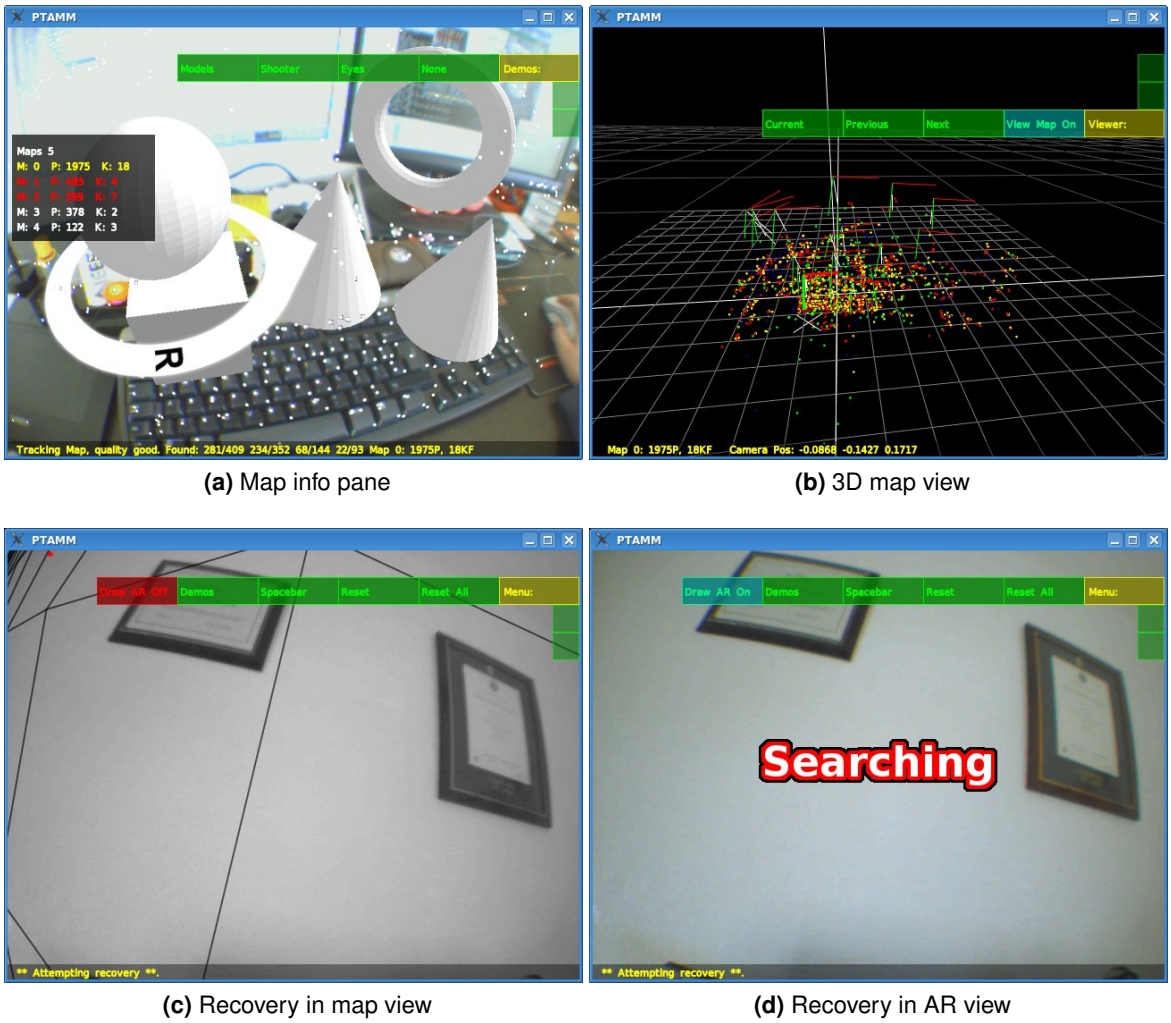
New source files:

MapSerializer.h - The map serialization header.

MapSerializer.cc - The map serialization implementation.

MapLockManager.h - The map lock manager header for thread control.

[1] `http://www.grinninglizard.com/tinyxml`



(a) Map info pane **(b)** 3D map view

(c) Recovery in map view **(d)** Recovery in AR view

Figure 3: Multiple maps and recovery.

MapLockManager.cc - The map lock manager implementation.
tinyxml.h - Main TinyXML header for reading/writing XML.
tinyxml.cc - TimyXML implementation.
tinyxmlerror.cc - TinyXML's error handler.
tinyxmlparser.h - TinyXML's parser header.
tinyxmlparser.cc - TinyXML's parser implementation.
MD5.h - MD5 hash generation header
MD5.cc - MD5 hash generation implementation
MD5Wrapper.h - A wrapper class header for MD5 hash generation
MD5Wrapper.cc - Wrapper implementation

Changes:

1. Locking, unlocking, and lock checking mechanisms added to `System` and `MapMaker`.

2. `Keyframe` changed so that it contains camera data.
3. `MapMaker` changed so that it uses the keyframe camera and not a global camera.
4. `System` changed to add new functions for saving and loading maps.
5. `ATanCamera` changed to allow camera creation from known parameters.

9.3 New Game Architecture

Users can now add their own games more easily by deriving from an abstract `Game` class. The three included games demonstrate how to use this new architecture. To add a new game to PTAMM changes need to be made in the following places.

Makefile - Add your new source files to the compile list, e.g. `MyGame.o`.

Games.h - Add the main game header file here. e.g. `#include "MyGame.h"`

Games.cc - Add your game using the templates provided in the two TODO sections.

New Source files:

Game.h - The abstract base class for all games

Games.h - A header file for including the each game's header and some miscellaneous functions.

Games.cc - Implementation of the miscellaneous functions.

Changes:

1. `EyeGame` class changed to use the new Game class.
2. `ARDriver` changed to load and render the specified game, pass mouse clicks and button presses, and update the state of the game.

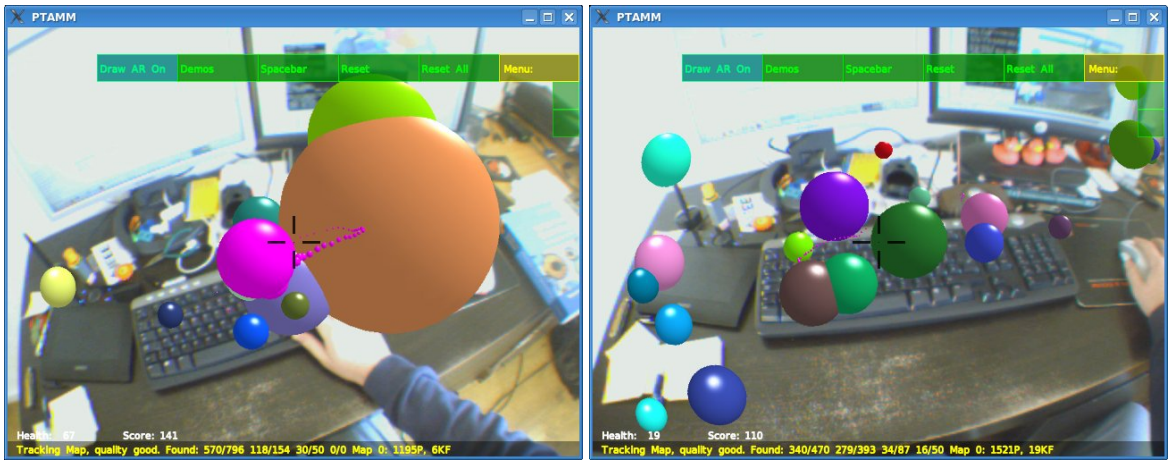
NOTE: The original PTAM code is not well designed for user interaction. To keep as much of the code the same, the resulting design is not the best. The main drawback is that only some user input is sent to the `GLWindow2` class, and it only uses it for the menus. This class has been modified to allow various commands to be passed to games using GVars. This is not good, but a major rewrite is required for a better gaming architecture, and that is not the main purpose of PTAM and PTAMM.

9.3.1 Shooter Game

This simple game allows users to shoot expanding spheres by pointing their camera at them pressing the Return key. If the spheres hit you, you will loose health. Fig. 4 shows some screen shots from the game. This game demonstrates how to:

- use key presses.
- draw 2D elements.
- load and save a game.
- update a game state using `Advance()`.

Game files:



(a) (b)

Figure 4: The shooter game.

ShooterGame.h - The main header file for the game.
ShooterGame.cc - The implementation of the game.
ShooterGameTarget.h - Target class header file (the sphere).
ShooterGameTarget.cc - Target class implementation.

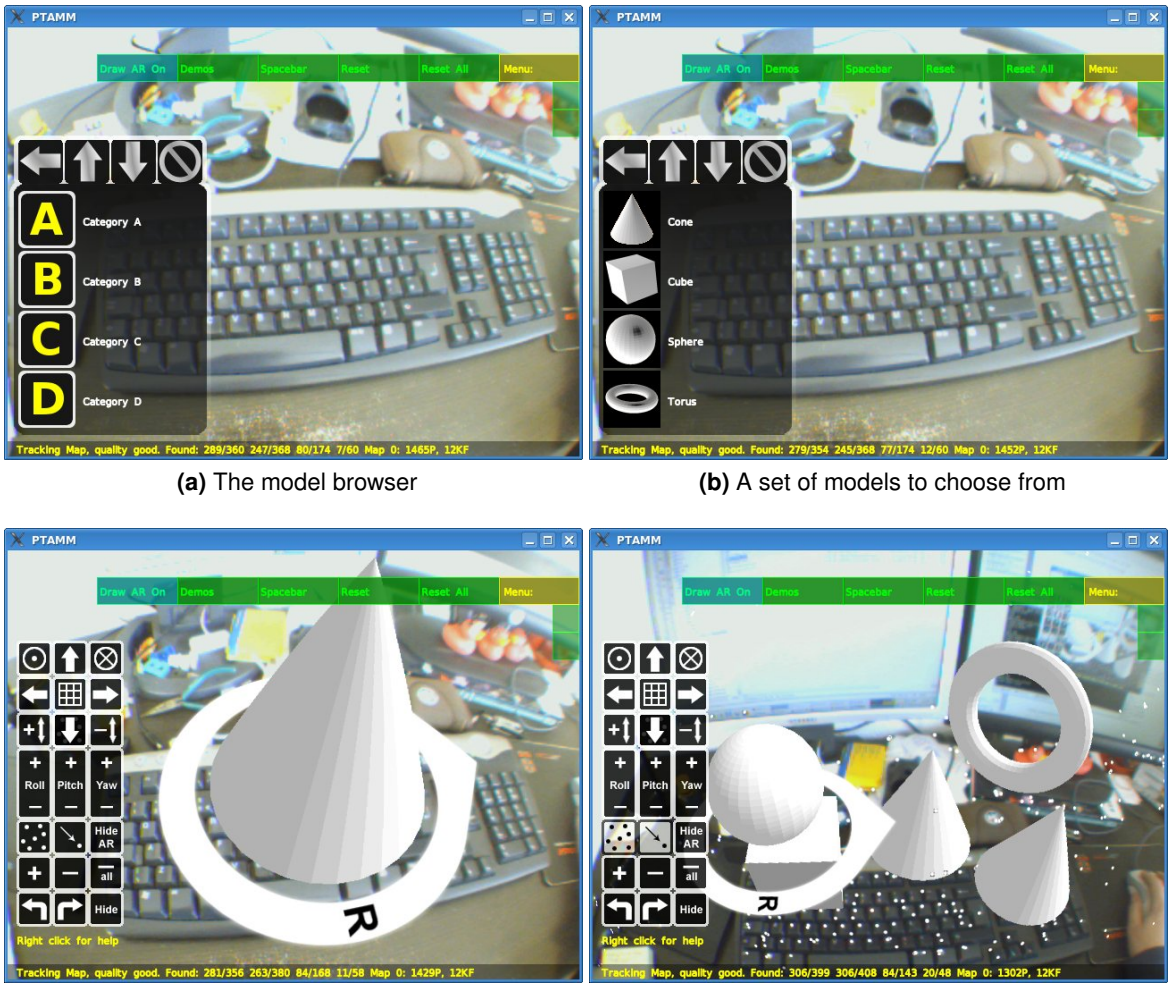
9.3.2 Models Game

This is a much more complex game, and allows the user to load a database of 3DS models, scroll through them, and place them in the world where they can be scaled and moved around. Fig. 5 shows some screen shots from the game. This game demonstrates how to:

- use mouse clicks.
- draw 2D elements.
- load and save a game.
- perform delayed loading.

Game files:

ModelsGame.h - The main header file for the game.
ModelsGame.cc - The implementation of the game.
ModelsGameData.h - The game data class header.
ModelsGameData.cc - The game data class implementation.
ModelBrowser.h - Browser interface class header.
ModelBrowser.cc - Browser interface class implementation.
ModelControls.h - Controls interface class header.
ModelControls.cc - Controls interface class implementation.
MGButton.h - Control interface button class header.
MGButton.cc - Control interface button class implementation.
Model3ds.h - A 3DS model class header.
Model3ds.cc - A 3DS model class implementation.



(a) The model browser **(b)** A set of models to choose from

(c) A model is added **(d)** Many models are added

Figure 5: The models game.

9.3.3 Model Database

To add your own models to the game the file **ARData/Models/models.xml** needs to be modified, or an alternative one specified before the game is started with the GVar `ModelsGame.XMLFile`. The basic database that is provided is listed below.

```
<?xml version="1.0" ?>
<PTAMM_Model_DB version="1.0">
  <Category name="Category A" image="Cat_A/A.png">
    <Model name="Cone" image="cone.png" dir="Cat_A/Cone" file="cone.3ds" roll="0"
        pitch="0" yaw="0" />
    <Model name="Cube" image="cube.png" dir="Cat_A/Cube" file="cube.3ds" roll="0"
        pitch="0" yaw="0" />
    <Model name="Sphere" image="sphere.png" dir="Cat_A/Sphere" file="sphere.3ds"
        roll="0" pitch="0" yaw="0" />
    <Model name="Torus" image="torus.png" dir="Cat_A/Torus" file="torus.3ds" roll="
        0" pitch="0" yaw="0" />
  </Category>
  <Category name="Category B" image="Cat_B/B.png">
  </Category>
```



```
  <Category name="Category C" image="Cat_C/C.png">
  </Category>
  <Category name="Category D" image="Cat_D/D.png">
  </Category>
  <Category name="Category E" image="Cat_E/E.png">
  </Category>
</PTAMM_Model_DB>
```

Four models are provided. Each category has a name and, optionally, an image. Each model in a category has a name, an optional image, the location of the model's directory relative to the database, the name of the model file, and an initial rotation of the model. The initial rotation is required as some models are not aligned facing along the x-axis. The angles are in degrees. The icons must be 64×64 pixels, and be in a format readable by libCVD.

10 The Source Code

The main documentation of the code is in the code's comments. What follows here is an overview of the tracking system. The tracker runs from `main.cc`, which spawns a `System` class which starts two main threads:

1. The `Tracker` class thread, driven by the `System` class event loop, which performs real-time tracking and video I/O and graphical display.
2. The `MapMaker` class thread updates the map in the background.

Both threads access a common `Map` class data structure. The `Map` class contains two main arrays: a vector of `MapPoint` structs, and a vector of `KeyFrame` structs. Together these make up the map. Each map also contains a pointer to an optional game derived from the base `Game` class.

The bulk of the functionality is contained within the classes `Tracker` and `MapMaker`, which also make use of auxiliary classes. The `Tracker` uses the `Relocaliser` class to recover from failure, and to switch maps; `MapMaker` uses the `Bundle` class to perform bundle adjustment and the `HomographyInit` class to bootstrap the map. Many bits of code use `ATANCamera` class, which is a model of the camera's intrinsic parameters.

Further, the `ARDriver` class provides distorted rendering and compositing to run AR games.

11 Commands

Most of the commands the user will need to access are available in the menus, however there are some further commands that are only accessible from the command line.

11.1 Window Menus

The menu system has more options than the PTAM one. The new options are detailed below, and Fig. 6 shows screenshots of the various menus.

Menu:

Reset All Resets PTAMM back to its initial state.

Reset Resets the current map. The map must be unlocked.

Spacebar Press this or the spacebar to start a map.

Demos Demos sub-menu.

- **None** No AR game set.
- **Eyes** The Eye game.
- **Shooter** The shooter game.
- **Models** The 3DS model game.

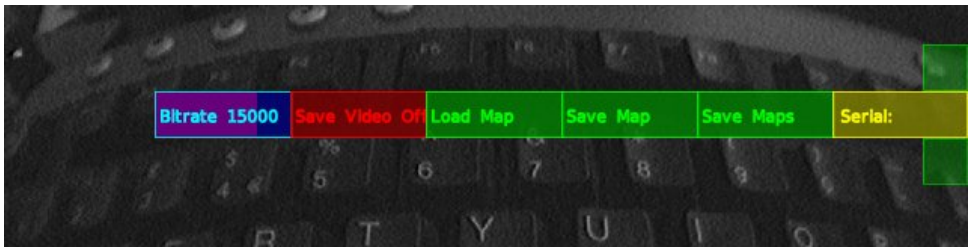
Draw AR Off/On Draw AR toggle.

Maps:

New Map Start a new map.

(a) Main Menu

(b) AR Demos Menu



(c) Serialization Menu

Figure 6: The menus.

Serialize Serialization sub-menu.

Save Maps Save all maps to disk using default names.

Save Map Save the current map to disk using the default name.

Load Map Load a map from disk using the default name.

Save Video Off/On Save the view out to an AVI video file.

Bitrate 15000 Slider to set the video bitrate - default value is 15kbps. It must be set before video is output for the first time.

Lock Map Off/On Lock a map from new keyframe addition, map point. addition/deletion, and bundle adjustment.

Delete Map Delete the current map.

Map Info Off/On Display map information.

Viewer:

View Map Off/On Show the 3D map viewer.

Next View the next map in the list.

Previous View the previous map in the list.

Current Switch back to the current map.

11.2 Console Commands

Map serialization has the majority of the new console commands. These allow much more advanced usage than the buttons do. Loading and saving both use the current map to either load to or save from. The default naming scheme is **map000000**, so a map with `mapNum = 5` would be saved to **map000005**. The following can be done using the console commands:

Action	Dir Name / Map Num	Result
LoadMap		Loads a map using the current map number (same as button)
LoadMap	mapNum	Load a specified map number
LoadMap	path/to/dirname	Load a specified map directory
SaveMap		Save the current map (same as button)
SaveMap	mapNum	Save the map number mapNum to disk
SaveMap	path/to/dirname	Save map mapNum to the path specified
SaveMaps		Save all maps (same as button)
SaveMaps	path/to/basename	Save all maps to the base path specified

Other commands of interest are:

DeleteMap mapNum Delete the map specified. If no number then the current map is deleted.

LockMap = 1/0 Lock the map toggle (1 = locked, 0 = unlocked)

MapInfo = 1/0 Draw the map info screen

NewMap Create a new map

NextMap View the next map in the list

PrevMap View the previous map in the list

CurrentMap Switch back to the current map

Reset Resets the current map. The map must be unlocked.

ResetAll Resets PTAMM back to its initial state

The games also have further console commands, and variables that may be of interest to the user.

12 Other Changes

- PTAMM has been placed in its own namespace, so as not to interfere with your code.
- When AR is being drawn and the tracker is lost, an image is displayed telling the user.
- Doxygen comments are being added to the PTAMM source. They mainly exist on functions that have been modified from PTAM, and on new functions.

13 An Example Linux Install Process

This section lists the commands used to build PTAMM from scratch. The build was done on a clean install of Ubuntu 9.04 32-bit with gcc 4.3.3 and kernel 2.6.28-11-generic. All sources are placed in `$HOME/Src`, and all libraries and headers are installed to `$HOME/local/lib` and `$HOME/local/include` respectively.

Firstly, download lib3ds from `http://sourceforge.net/projects/lib3ds/files`, and unzip to the source dir. The version used is `lib3ds-20080909.zip`.

```
cd ~
mkdir Src
mkdir -p local/include
mkdir -p local/lib
cd Src
unzip ../Desktop/lib3ds-20080909.zip
sudo apt-get install build-essential
sudo apt-get install cvs liblapack3gf liblapack-dev libraw1394-dev libdc1394-22
    libdc1394-22-dev libtiff4 libtiff4-dev freeglut3 freeglut3-dev coriander libpng-
    dev libreadline5-dev mencoder
cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/toon co -D "Mon May
    11 16:29:26 BST 2009" TooN
cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/libcvd co -D "Mon May
     11 16:29:26 BST 2009" libcvd
cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/libcvd co -D "Mon May
     11 16:29:26 BST 2009" gvars3
cd TooN/
./configure --prefix=$HOME/local
make install
cd ../libcvd/
export CXXFLAGS=-D_REENTRANT
./configure --without-ffmpeg --prefix=$HOME/local
make -j3
make install
cd ../gvars3/
./configure --disable-widgets --prefix=$HOME/local
make -j3
make install
cd ../lib3ds-20080909/
./configure --prefix=$HOME/local
make -j3
make install
cd ../
unzip ../Desktop/ptamm.zip
cd PTAMM
cp Build/Linux/* .
gedit Makefile
make
```

The changes that were made to the Makefile are to specify the include and link path, i.e. the following replacements were made:

- `MY_CUSTOM_INCLUDE_PATH` replaced with `/$(HOME)/local/include`
- `MY_CUSTOM_LINK_PATH` replaced with `/$(HOME)/local/lib`

changing


```
COMPILEFLAGS = -I MY_CUSTOM_INCLUDE_PATH -D_LINUX -D_REENTRANT -Wall  -O3 -march=
    nocona -msse3 -fno-strict-aliasing
LINKFLAGS = -L MY_CUSTOM_LINK_PATH -lGVars3 -lcvd $(3DSLIB)
```

to

```
COMPILEFLAGS = -I $(HOME)/local/include -D_LINUX -D_REENTRANT -Wall  -O3 -march=
    nocona -msse3 -fno-strict-aliasing
LINKFLAGS = -L $(HOME)/local/lib -lGVars3 -lcvd $(3DSLIB)
```

If you are using a Firewire camera then before running the CameraCalibrator or PTAMM issuing the following command may be necessary:

```
sudo chmod a+rwX -R /dev/*1394
```


14 An Example OSX Install Process

This section lists the commands used to build PTAMM from scratch. The build was done on 64-bit OSX 10.6 (Snow Leopard). All sources are placed in `$HOME/Src`, and all libraries and headers are installed to `$HOME/local/lib` and `$HOME/local/include` respectively.

Eariler, it was advocated that using the Cambridge libraries from a particular date was best, however if you run into issues grab one of the latest release snapshots. In general do not grab the latest CVS source, as that can be broken in interesting ways.

For this Mac OSX build, using the dated CVS source resulted in the following error when building libCVD

```
cvd_src/i686/yuv411_to_stuff_MMX_64.C: In function  void CVD::ColourSpace::
    yuv411_to_rgb_y(const unsigned char*, int, unsigned char*, unsigned char*)   :
    cvd_src/i686/yuv411_to_stuff_MMX_64.C:246: error: bp cannot be used in asm here
```

Newer releases do not have this issue. However, there are some hurdles to overcome.

The files used in this build are as follows:

- `TooN-2.0.beta7.zip`
- `libcvd-20100511.zip`
- `gvars3-20090421.tar.gz`
- `lib3ds-20080909.zip`
- `ptamm.zip`

During this build I use MacPorts[2] to grab the missing dependencies.

First of all make sure MacPorts is set up correctly to build 32-bit versions, as well as 64-bit versions. Edit `/opt/local/etc/macports/variants.conf` and add `+universal` at the end.

```
sudo port -v selfupdate
sudo port install libdc1394 doxygen readline
cd ~
mkdir -p local/include
mkdir -p local/lib
```

Doxygen brings in a lot of stuff that you need such as jpeg, png, and some x11 stuff. Now edit `.bash_profile` in your home directory add the following lines, so that your `local` directory is known about

```
export CPATH=$CPATH:$HOME/local/include
export LIBRARY_PATH=$LIBRARY_PATH:$HOME/local/lib
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$HOME/local/lib
```

You can just call these lines in your terminal, but you will need to do it for each terminal session.

Now onto building the dependencies. The following assumes that the files are in your `Src` directory already

```
cd Src
unzip ptamm.zip
unzip lib3ds-20080909.zip
tar xvf TooN-2.0.beta7.tar
tar xvf libcvd-20100511.tar
tar xvf gvars3-20090421.tar
cd TooN-2.0.beta7/
```

[2] `http://www.macports.org/install.php` - v1.9.1 for Snow Leopard

Here, we need to edit the TooN `SymEigen.h` file to fix a bug. Add `#include <algorithm>` to the top and find and replace all occurancies of `swap` with `std::swap`. Strangely this is only required for PTAMM, PTAM works without this change.

Continuing:

```
./configure --prefix=$HOME/local
make install
cd ../libcvd-20100511
```

Here, we need to make two edits. First edit the `cvd/gl_helpers.h` file to make it look in the right place for the OpenGL headers. Change

```
#include <GL/gl.h>
#include <GL/glu.h>
```

to

```
#ifdef _OSX
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
#include <GL/gl.h>
#include <GL/glu.h>
#endif
```

This change is a little suspect, as I am sure the libCVD uses the X11 GL headers (the ones in GL/), and PTAM and PTAMM use the ones in the OSX location (OpenGL/). It works, but it is not good.

Second, edit `cvd_src/Linux/dvbuffer3_dc1394v2.cc` and on lines 129 and 156 add the letters "LLU" to the end of the long hex numbers, i.e.

```
if (guid==0x814436200006075) { return GBRG; }
if (guid==0x814436200006075) {
```

becomes

```
if (guid==0x814436200006075LLU) { return GBRG; }
if (guid==0x814436200006075LLU) {
```

Before we configure, we need the special bash script to force 10.5 32-bit building. This is supplied with PTAMM.

```
cp ../PTAMM/Build/OSX/configure-10.5-32bit .
export CXXFLAGS=-D_REENTRANT
./configure-10.5-32bit --without-ffmpeg --prefix=$HOME/local --x-libraries=/usr/X11
    /lib --x-includes=/usr/X11/include
make -j3
make install
```

If you are not sure if libCVD has all the right options selected after running the configure, then compare it to my output below:

```
Options:
inline_asm dc1394v2 qtbuffer videodisplay tr1_shared_ptr toon lapack pthread png
    jpeg tiff glob v4lbuffer
Missing options for darwin10.4.1:
assembler ffmpeg memalign posix_rt v4lbuffer
Dodgy things:
missing_c99_feenableexcept no_posix_memalign
SIMD support:
mmx mmxext sse sse2 sse3
Missing SIMD support for i386-apple-darwin10.4.1:
```


Continuing with GVars:

```
cd ../gvars3/
cp ../PTAMM/Build/OSX/configure-10.5-32bit .
./configure-10.5-32bit --disable-widgets --prefix=$HOME/local
make -j3
make install
cd ../lib3ds-20080909/
cp ../PTAMM/Build/OSX/configure-10.5-32bit .
./configure-10.5-32bit --prefix=$HOME/local
make -j3
make install
cd ../PTAMM
cp Build/OSX/* .
vim Makefile
make
```

The changes that were made to the Makefile are to specify the include and link path, i.e. the following replacements were made:

- `MY_CUSTOM_INCLUDE_PATH` replaced with `$(HOME)/local/include`
- `MY_CUSTOM_LINK_PATH` replaced with `$(HOME)/local/lib`

There should now be a CameraCalibrator and PTAMM executable in the directory.

This has been tested on a Core 2 Duo MacBook Pro and an iMac Quad Core i5. PTAMM runs great on the MacBook, but fails to run on the iMac exiting after the QT screen with the cryptic error message: ”Bus error”. No idea why.

15 An Example Windows Install Process

This example installation has been tested on 32-bit Windows XP Media Centre Edition with all updates applied. Visual Studio 2005 is used along with Windows SDK 6.1.

During this install, we will first build PTAM and then PTAMM.

This installation may not be optimal or the best way, but it worked. If you have any advice or comments please email `ptam@robots.ox.ac.uk` or post them on the blog at `http://ewokrampage.wordpress.com`.

Firstly download, extract or install, the required dependencies, here I used the following:

- `TooN-2.0.beta7.zip`
- `libcvd-20100511.zip`
- `gvars3-20090421.tar.gz`
- `pthreads-w32-2-8-0-release.exe`
- `lib3ds-20080909.zip`
- `lapack-MT-release.zip`
- `glew-1.5.5-win32.zip`
- `1394camera645.exe` (I selected to install the additional install options)
- `jpeg-6b-4.exe`
- `libpng-1.2.37-setup.exe`
- `zlib125.zip`
- `PTAM.zip`
- `ptamm.zip`

Then create the following directory structure:

```
($SRCDIR)
 |
 +---gvars3
 +---include
 +---lib
 +---lib3ds
 +---libcvd
 +---PTAM
 +---PTAMM
 \---zlib125
```

`include` and `lib` with hold the header files and `.lib` files respectively, and the others are the extracted sources that need to be built.

Now copy the headers and lib files from the dependencies into the `include` and `lib` directories. Also copy `TooN` into the `include` directory. This is what my directories contained:

```
+---include
|   |   1394camapi.h
|   |   1394Camera.h
|   |   1394CameraControl.h
|   |   1394CameraControlSize.h
|   |   1394CameraControlStrobe.h
|   |   1394CameraControlTrigger.h
|   |   1394common.h
|   |   jconfig.h
|   |   jerror.h
|   |   jmorecfg.h
|   |   jpeglib.h
|   |   png.h
|   |   pngconf.h
|   |   pthread.h
|   |   sched.h
|   |   semaphore.h
|   +---GL
|   |       glew.h
|   |       glxew.h
|   |       wglew.h
|   +---libpng12
|   |       png.h
|   |       pngconf.h
|   |
|   \---TooN
|          (lots of toon files and directories)
|
+---lib
|       1394camera.lib
|       1394camerad.lib
|       blas_win32_MT.lib
|       glew32.lib
|       glew32s.lib
|       jpeg-bcc.lib
|       jpeg.lib
|       lapack_win32_MT.lib
|       libpng-bcc.lib
|       libpng.lib
|       pthreadVC2.lib
|       pthreadVCE2.lib
|       pthreadVSE2.lib
```

Copy the following dlls into the `PTAM` and `PTAMM` directories. They should be in the same place as the executables, i.e. in the Debug and Release directories. It will be easiest to copy these in once you have tried to build PTAM or PTAMM, and Visual Studio has made the directories for you. (Or you can be naughty and just stick them in `C:/Windows/System32`.)

```
glew32.dll
blas_win32_MT.dll
lapack_win32_MT.dll
pthreadGC2.dll
pthreadGCE2.dll
pthreadVC2.dll
pthreadVCE2.dll
pthreadVSE2.dll
```


If you used the installers then these should already be in `C:/Windows/System32`, and there is no need to put them in the `PTAM` and `PTAMM` directories:

```
1394camera.dll
1394camerad.dll
```

15.1 Build zlib

For PNG support, zlib is needed. It does not come with a VS2005 solution file, so one needs to be made. The following steps show how the library can be built and installed.

1. File >New >Project from existing code
2. Project type: Visual C++
3. Locate the dir
4. Give the project a name
5. Untick "Add subfolders"
6. Project type is "Static library (LIB) project"
7. Once out of the wizard, build a Release build.
8. Copy `zlib.lib` to `lib`
9. Copy `zlib.h` and `zconf.h` to `include`

15.2 Configure TooN

TooN may need configuring, by editing or creating `include/TooN/internals/config.hh`. This is not well documented, and the contents of this file seems to change with every iteration of TooN. It should contain only the following line:

```
#define TOON_USE_LAPACK 1
```

15.3 Build libCVD

Open the `libcvd.sln` solution file in `libcvd/build/vc2005`. Set the build to "Release" then edit the project properties.

Set the additional include paths to:

```
..\..
..\..\..
..\..\..\inlcude
```

Set the additional libs path to:

```
..\..\..\libs
```

To enable PNG support, open `config.h` and make sure `CVD_HAVE_PNG` will be defined:

```
#ifndef CVD_DISABLE_PNG
    #define CVD_HAVE_PNG 1
#endif
```

Then make sure that `png.cc` is included in the solution file. If not add it. It should be under *Source Files >pnm_src*

Then build the solution. Repeat for the Debug build if required.

Copy the resulting `libcvd.lib` (and `libcvdd.lib` if you built the debug build) from `libcvd/lib` to your `lib` directory. Then copy `libcvd/cvd` to `include/cvd`.

15.4 Build GVars3

Unless you want to use FLTK with GVars for some other project I suggest building GVars without FLTK support. This is detailed below.

Open the `gvars3.sln` solution file in `gvars3/build/vc2005`.

Remove the following files from the project:

```
GUI_Fltk2.h
GUI_Fltk2.cc
```

Create a `config.h` file in `gvars3/gvars3` and place the following in it:

```
#ifndef GVARS3_INCLUDE_CONFIG_H
#define GVARS3_INCLUDE_CONFIG_H
#define GVARS3_HAVE_TOON 1
#endif
```

Set the build to "Release" then edit the project properties. Set the additional include and lib paths to the same as the ones used above in libCVD, and remove the FLTK path.

Repeat this for gvars3-headless, and for the Debug build if you require it.

Now build the solution. Repeat for the Debug build if required.

Copy the resulting `gvars3.lib` and `gvars3-headless.lib` (and `gvars3d.lib` and `gvars3-headlessd.lib` if you built the debug build) from `gvars3/lib` to your `lib` directory. Then copy `gvars3/gvars3` to `include/gvars3`.

15.4.1 GVars3 Compilation Issues

This is where I ran into serious problems with the Cambridge libraries.

If you get this error:

```
error C2064: term does not evaluate to a function taking 0 arguments gvars3.h 258
```

edit `gvars3.h` and change line 258 from:

```
template<class T> static T& get(const std::string& name, const T& default_val=
    DefaultValue<T>::val(), int flags=0);
```

to:

```
template<class T> static T& get(const std::string& name, const T& default_val=T(),
    int flags=0);
```

15.5 Build PTAM

Copy all of the files in `PTAM/Build/Win32` to `PTAM`, and then open the solution file. Set build as "Release" and edit the project properties to add the additional include and lib dirs:

```
../include
../lib
```

NOTE: Rather than copying the headers and libs from the libCVD and GVars directories each time you build them, you could add the libCVD and GVar3 directories to the PTAM project properties.

15.5.1 PTAM Compilation Issues

Here are the issues that I ran into compiling PTAM.

If there is a build error about ambiguity of sqrt in `TooN::Internal::ComputeSymEigen` on line 181 change:

```
ev =  makeVector(A_plus_B, -A_plus_B/2 + A_minus_B * sqrt(3)/2, -A_plus_B/2 -
    A_minus_B * sqrt(3)/2) - Ones * a/3;
```

to:

```
ev =  makeVector(A_plus_B, -A_plus_B/2 + A_minus_B * sqrt(3.0)/2.0, -A_plus_B/2 -
    A_minus_B * sqrt(3.0)/2.0) - Ones * a/3;
```

If the compiler complains that is cannot find `blas_win32.lib` or `lapack_win32.lib` then check their names in the lib folder. They may be `blas_win32_MT.lib` and `lapack_win32_MT.lib`. If so change the name of libs that the CameraCalibrator and PTAM projects are looking for.

15.6 Build PTAMM

If you have got this far then PTAM should be working and the rest should be painless. First of all we need to build the lib3ds library.

1. Open `lib3ds.sln` and build a "Release" build.
2. Copy `lib3ds-2_0.lib` to `lib`
3. Copy `lib3ds-2_0.dll` to the PTAMM Release and Debug directories (or `c:/Windows/System32`)
4. Copy `lib3ds.h` to `include`

Now onto PTAMM. Copy all of the files in `PTAMM/Build/Win32` to `PTAMM`, and then open the solution file. Set build as "Release" and build the solution.

If you have placed files in different directories or have libs that have different names, edit the project properties and make the appropriate modifications. Remember to add the preprocessor definition `ENABLE_MODELS_GAME` if you want to build the 3ds Models game.

You should now have a working build of PTAMM on Windows.

16 Acknowledgements

Many thanks to Simon Hadfield (`http://www.computing.surrey.ac.uk/personal/pg/S.Hadfield`) for providing the required modifications to PTAMM to make it build under Windows, and supplying the solution to why GVars would not build under Windows. Thank you to Damian Stewart for the bash script for OSX configuration, and thank you to Adam Clarkson (`http://ajclarkson.wordpress.com`) for help on the OSX build.

17 Troubleshooting and Updates

See the PTAM Blog (`http://ewokrampage.wordpress.com/` for troubleshooting tips and PTAM and PTAMM update news.

