# Parallel Tracking and Multiple Mapping (PTAMM) Manual

Daniel Palomino

February 18, 2019

# 1 Introduction

This software is an implementation of the method described in the paper Parallel tracking and mapping for small AR workspaces by Robert Castle, Georg Klein and David Murray, which appeared in the proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR) 2007. (Klein & Murray, 2007) (http://www.robots.ox.ac.uk/ gk/PTAM/)

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.

# 2 PTAMM's Enhancements Over PTAM

1. Multiple maps with automatic switching
2. Saving and loading to and from disk.
3. Video Output.
4. New game architecture to add your own games.
5. 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.

## 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. This release was tested over Intel Core i7 @ 3.4GHz but 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 × 480 × 30Hz 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. 2019-01-11 06-31-38.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 GIT:

```
$> git clone https://github.com/edrosten/TooN
$> git clone https://github.com/edrosten/libcvd.git
$> git clone https://github.com/edrosten/gvars.git
```

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:

1. blas,
2. lapack,
3. libgfortran,
4. ncurses,
5. libreadline,
6. libdc1394 (and maybe libraw1394) for Firewire capture,
7. libpng.
8. 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
```

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

Next, just do:

```
$> make
$> make install
```

For GVars3, I recommend the following configure options:

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

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, for Linux systems is not needed, all files are in theirs correct location. 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: *VideoSource_Linux_V4L.cc*.

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 Running The Software

### 5.1 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:

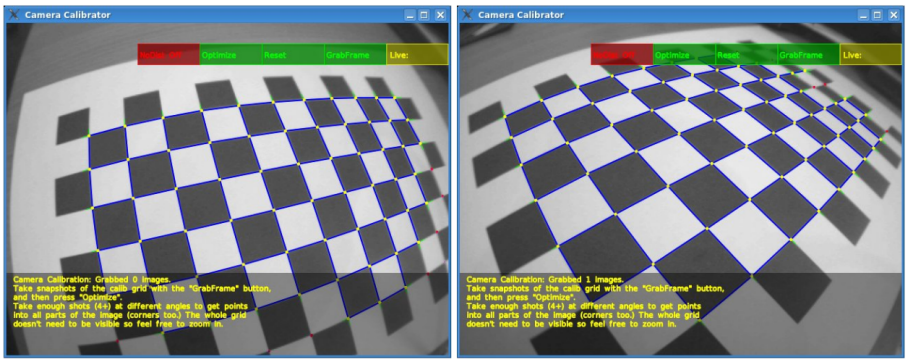
1. Modify camera settings to remove any sharpening; Sharpening artifacts break the calibrator, and also reduce performance of the tracker.

2. 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 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.




(a) (b)

Figure 1: Camera Calibration.

## 5.2 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. 4 shows PTAMM running.

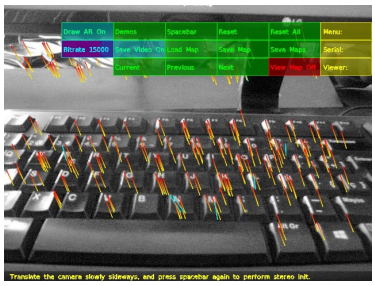
If there appear to be problems with the map not being expanded or not being tracked well, the most likely culprit is a lack of baseline, i.e. the camera was not translated enough. A stereo initialisation with only rotation does not provide a baseline and cannot ever work.

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

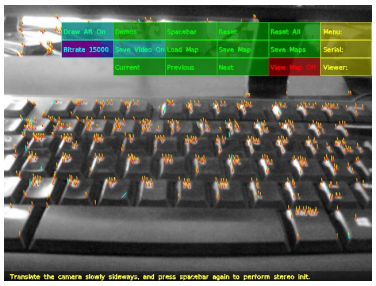
Steps for use the software:

1. Point the Camera at planar scene and press spacebar to start for initial map.

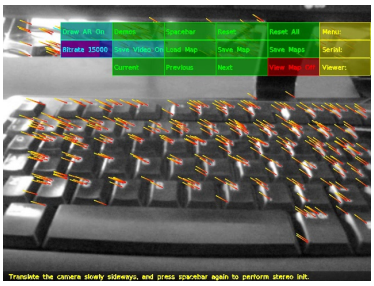
(a) Point the camera at planar scene



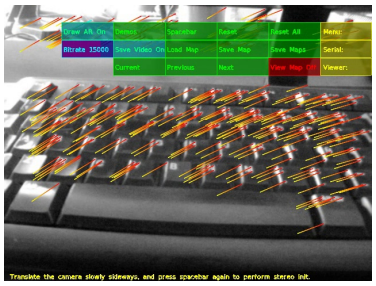
(b) Press Spacebar and move the camera



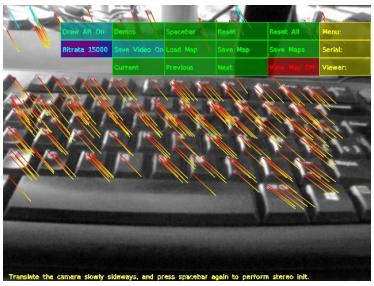
(c) Keep moving the camera



(d) Keep moving the camera



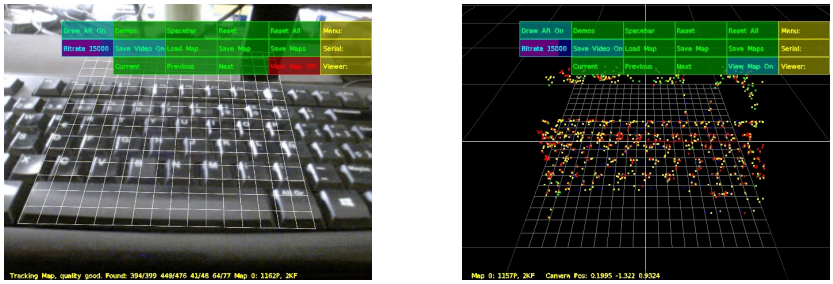
(e) Keep moving the camera



(f) Keep moving the camera

Figure 2: Steps for map creation

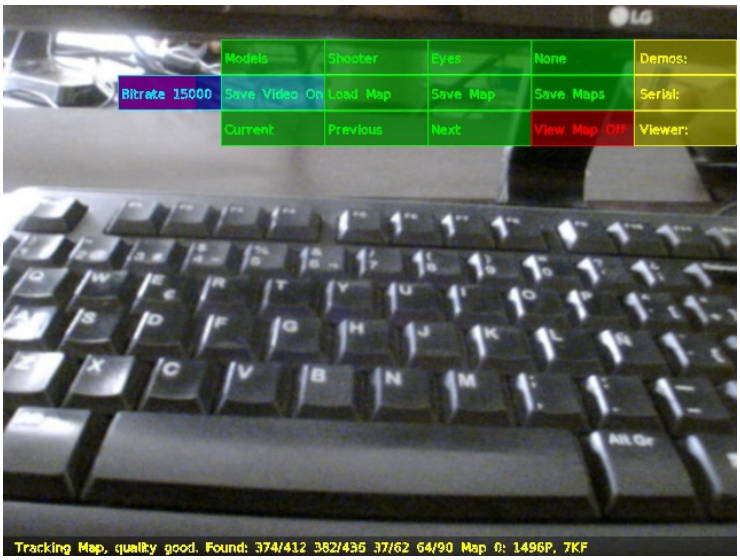
2. Visualize the map created.



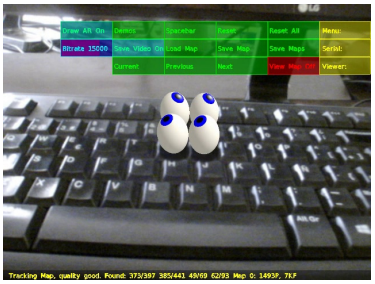
(a) Planar grid created (b) Press View Map button

Figure 3: Visualize the map created

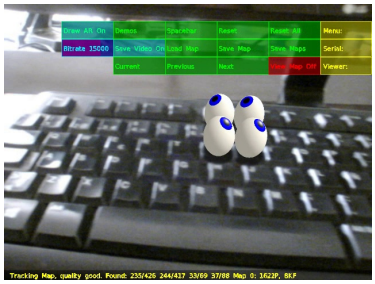
3. Choose the AR model.




(a) Press the Draw AR button



(b) Select the AR model



(c) Move the camera or the base

Figure 4: Visualize the AR model selected

# References

Klein, G., & Murray, D. (2007, 01). Parallel tracking and mapping for small ar workspaces. *ISMAR. IEEE*, 225-234.