3DMCAP Documentation

Release 1.0

Thiago T. Santos <thiago.santos@embrapa.br>

March 22, 2018

INTRODUCTION

3DMCAP is an application for three-dimensional reconstruction of objects from digital images. It allows an ordinary camera and a computer to operate as *a simple 3-D scanner*. The application assists the user on imaging and computing a cloud of 3-D points, sampling the objects surfaces in three dimensions. Its purpose within Embrapa is the 3-D reconstruction of plants for purposes of automatic measurement in phenotyping and precision agriculture [1].

You can see what the application is able to do in:

Santos, T. T., Bassoi, L. H., Oldoni, H., & Martins, R. L. (2017). *Automatic grape bunch detection in vineyards based on affordable 3D phenotyping using a consumer webcam*. In J. G. A. Barbedo, M. F. Moura, L. A. S. Romani, T. T. Santos, & D. P. Drucker (Eds.), Anais do XI Congresso Brasileiro de Agroinformática (SBIAgro 2017) (pp. 89–98). Campinas: Unicamp. PDF

Comments and contributions

Comments, issues and any feedback can be send using the 3DMCAP repository at GitHub[1].

[1] Visit https://github.com/thsant/3dmcap.

CHAPTER

ONE

INSTALLATION

1.1 Running using Docker

The easiest way to install and run 3DMCAP is using our *Docker image*, thsant/3dmcap. If you have Docker running in a Ubuntu Linux host and a camera connected as `/dev/video0`, the following command should install and run 3DMCAP for you:

```
$ docker run -it --rm -e DISPLAY=$DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
--device /dev/video0 \
-v $HOME/.3dmcap:/home/demeter/.3dmcap -v $HOME/workdir:/home/demeter/workdir \
thsant/3dmcap capture.py
```

Note the command above assumes you have, in your home directory, a directory `.3dmcap`, containing the configuration file (see Section 2.3 for details), and a `workdir` directory for saving the results produced by the application in the Docker container.

If you have problems loading the graphical interface, try to execute

```
$ xhost +local:root
```

before the docker run. After the execution, run

```
$ xhost -local:root
```

to return the access controls.

1.1.1 Errors regarding NVIDIA drivers and OpenGL

If you are facing crashes presenting the following error message:

```
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
```

then you are having issues regarding NVIDIA drivers and OpenGL from Docker. In this case, consider NVIDIA Docker as a workaround.

1.2 Building 3DMCAP from sources

For users that prefer build the entire system in their own hosts, this is the detailed building process.

1.2.1 Dependencies

The software depends on:

- Eigen 3;
- OpenCV with Python 2.7 support;
- Our modified version of ORB_SLAM2. It includes DBoW2 and g2o as the original ORB_SLAM2[2].

The Python components depend on:

- numpy
- scipy
- Pillow
- wxPython
- ObjectListView
- traits
- traitsui
- mayavi
- PyYAML

3DMCAP have beend tested in **Ubuntu 16.04** and **16.10**.

1.2.2 Building

We assume `/usr/local` as the install directory.

```
INSTALL_PREFIX=/usr/local
```

Add Eigen (needed by OpenCV and g2o), build-essential and cmake:

```
apt-get install -y build-essential libeigen3-dev cmake
```

Add OpenCV 3.4 dependencies:

```
apt-get install -y git libgtk2.0-dev pkg-config libavcodec-dev \
libavformat-dev libswscale-dev
apt-get install -y python-dev python-numpy libtbb2 libtbb-dev \
libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

Get the OpenCV 3.4.1 sources and extract the files to `/usr/local/src`. Then build OpenCV:

```
cd /usr/local/src/opencv-3.4.1
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=$INSTALL_PREFIX ..
make -j4
make install
```

Check if the Python module is OK:

```
python -c 'import cv2; print cv2.__version__'
```

Install Pangolin:

```
apt-get install -y libglew-dev
cd /usr/local/src
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
mkdir build
cd build
cmake -D CMAKE_INSTALL_PREFIX=$INSTALL_PREFIX ..
make -j4
make install
```

Install our modified ORB-SLAM2 version:

```
apt-get install -y python-pip
pip install cython
cd /usr/local/src
git clone https://github.com/thsant/ORB_SLAM2.git
cd ORB_SLAM2
./build.sh
cp lib/libORB_SLAM2.so /usr/local/lib
cp Thirdparty/DBoW2/lib/libDBoW2.so /usr/local/lib
cp Thirdparty/g2o/lib/libg2o.so /usr/local/lib
cp python/slam.so /usr/local/lib/python2.7/dist-packages/
mkdir /usr/local/share/3dmcap
cp Vocabulary/ORBvoc.txt /usr/local/share/3dmcap/
```

Install PMVS. We recommend pmoulon's version at GitHub:

```
cd /usr/local/src
git clone https://github.com/pmoulon/CMVS-PMVS.git
cd CMVS-PMVS/program
mkdir build
cd build/
cmake -D CMAKE_INSTALL_PREFIX=$INSTALL_PREFIX ..
make -j4
make install
```

Add other 3DMCAP dependencies:

```
apt-get install -y python-wxgtk3.0 python-vtk python-tk v4l-utils
```

Finally, get 3DMCAP code:

```
cd /usr/local/src
git clone https://github.com/thsant/3dmcap.git
cd 3dmcap
pip install -r requirements.txt
```

Configure the environment:

```
cd /usr/local/src/3dmcap
cp -r dmcap/ /usr/local/lib/python2.7/dist-packages
cp -r ./resources/* /usr/local/share/3dmcap
cp ./dmcap/camcal.py ./dmcap/capture.py /usr/local/bin
```

Edit the **3dmcap.cfg** file and save it to your `$HOME/.3dmcap` directory. You can run **capture.py** to start 3DMCAP.

CHAPTER TWO

BEFORE STARTING IMAGE ACQUISITION

Before using 3DMCAP for your image-based reconstruction, you must complete a few preliminary steps:

1. Pick a USB camera supported by Video4Linux.
2. Disable camera autofocus functionality, if it is present.
3. Calibrate the camera - you can use our *camcal.py* utility, included in 3DMCAP distribution.
4. Print the *scaling pattern*, if you wish that 3DMCAP to transform your point cloud to a specific scale (millimeters or inches, for example).
5. Edit your *configuration file*.

In normal conditions, you should perform this procedure a single time if you intend to use the same camera several times.

2.1 Camera selection and calibration

You should use an *USB high-definition camera* that is supported by Video4Linux. We have used the Logitech HD Webcam c920, but other devices should also work.

After connecting your camera, you should *set the focus to infinity* and *turn autofocus off*. Changing focus turn the visual odometry (the estimation of the camera location) a lot harder and the ORB-SLAM2 system (the visual odometry system used by 3DMCAP) will not work properly. You can use the `v4l2-ctl` tool accomplish this step at the Linux shell:

```
$ v4l2-ctl -d /dev/video1 -c focus_auto=0
$ v4l2-ctl -d /dev/video1 -c focus_absolute=0
```

The example above assumes your USB camera is connected as `/dev/video1`. After that, you can use the utility `camcal.py`, included in the 3DMCAP software, to calibrate the camera. If 1920×1080 is the maximum resolution your camera is able to support, you should run `camcal.py` as the following:

```
$ camcal.py --device /dev/video1 --fwidth 1920 --fheight 1080
```

or, if you are using the Docker container:

```
$ docker run -it --rm -e DISPLAY=$DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
--device /dev/video1 -v /tmp:/home/demeter/workdir:rw \
-v /home/thiago/.3dmcap:/home/demeter/.3dmcap \
thsant/3dmcap camcal.py --device /dev/video1 --fwidth 1920 --fheight 1080
```

You should print the *chessboard pattern* available in the *resources directory* and use the application to capture images of it from multiples views, as seen in Figure 2.1.

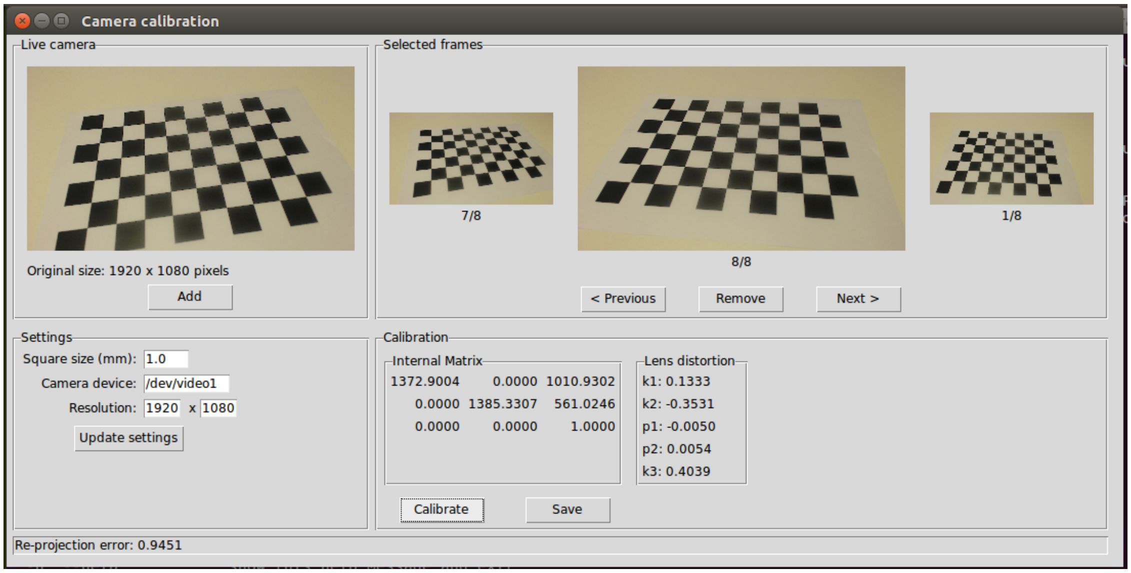

Fig. 2.1: The camera calibration tool `camcal.py`.

Take a dozen images or more, then click the *Calibrate* button. Then, save the values to a file using the *Save* button for further usage.

2.2 Printing the scaling pattern

The *scaling pattern* (available in the *resources directory*) is a sheet containing easily detectable markers presenting a known size (Figure 2.2). Ideally, it should be printed and then *laminated* (or fixed on a planar surface, as a table for example), forming a rigid planar tablet. 3DMCAP will use this pattern to scale the point cloud to a proper measurement unit and also rotate the cloud to a standard orientation. If you put the scaling pattern in the ground, 3DMCAP can give you a oriented model where the Z axis points upward.

2.3 The configuration file

The configuration file provides essential information that 3DMCAP needs to work properly. The application looks for a configuration file in four different locations, using the first file it found in the following order:

1. `$HOME/.3dmcap/3dmcap.cfg`
2. `/etc/3dmcap.cfg`
3. `/usr/local/share/3dmcap/3dmcap.cfg`
4. `/usr/share/3dmcap/3dmcap.cfg`

We recommend users employ the first option, creating a `.3dmcap` directory in their home directories and placing a `3dmcap.cfg` file there. Below we show an example of a working `3dmcap.cfg` file:

```
[camera]
width=1920
height=1080

[general]
```

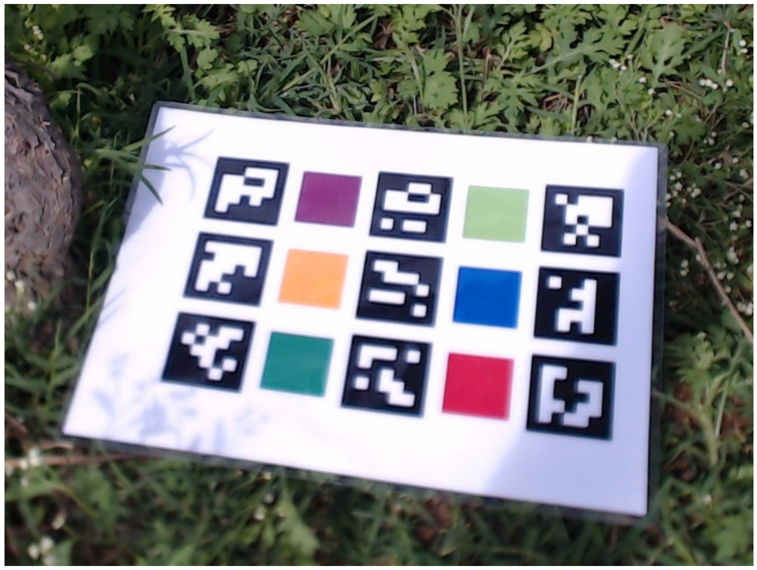

Fig. 2.2: The scaling pattern.

```
resources_path=/usr/local/share/3dmcap
ref_distance_mm=51.5

[orbslam]
config_fpath=/usr/local/share/3dmcap/Logitech-C920.yaml
```

The *camera* section provides the desired frame resolution. Remember digital cameras support several different resolutions and we recommend the bigger one able to work at 30 Hz. The values for width and height **must be the same** used in the camera calibration step. The *general* section informs the path for the resources directory containing essential files for the application. Also in this section we have `ref_distance_mm`, where you must provide the distance observed between two adjacent markers in your printed scaling pattern. Different printing configurations can create different patterns, so it is important you measure your final scaling pattern and set this value properly (Figure 2.3).

Finally, the *orbslam* section defines the path to the ORB-SLAM2 YAML file containing the camera calibration and other parameters needed by ORB-SLAM2 system. You should edit the camera calibration values, inserting the values you got using `camcal.py`. Again, you will find an example in the resources directory.

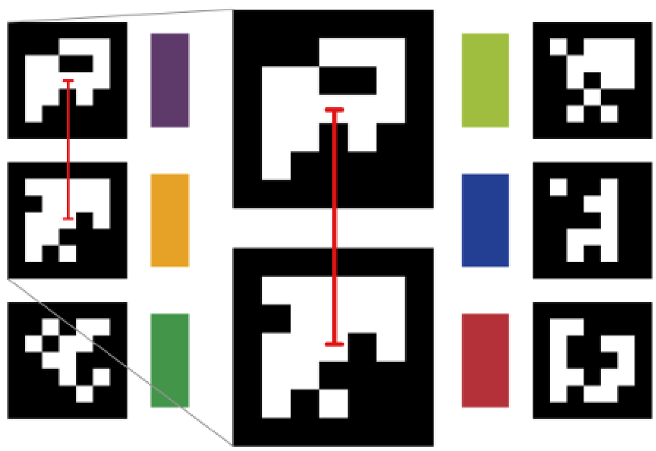

Fig. 2.3: The scaling pattern reference distance is the distance between the centers of two adjacent black-and-white squares.

CHAPTER THREE

THE IMAGE ACQUISITION STEP

The application is started running `capture.py`:

```
$ capture.py
```

Using Docker, you should employ:

```
docker run -it --rm \
-e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix \
--device /dev/video0 --device /dev/video1 \
-v /tmp:/home/demeter/workdir:rw \
-v /home/thiago/.3dmcap:/home/demeter/.3dmcap
thsant/3dmcap capture.py
```

Consider the following arguments:

- `-e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix` to make the Docker container use the host X Windows system;
- `--device /dev/video0 --device /dev/video1` to make the cameras in the host available to the container;
- `-v /tmp:/home/demeter/workdir:rw` to map the host `/tmp` directory to the container `$HOME/workdir`; and
- `-v /home/thiago/.3dmcap:/home/demeter/.3dmcap` to map the configuration directory in the host to the container.

You can change the host working directory or the host configuration directory to values that make more sense to your personal workflow. Considering the configuration is properly set, you should see the main application screen as shown in Figure 3.1.

The first step is to use the *Settings* menu to select the camera to be employed, as shown in Figure 3.2.

Image acquisition is started pressing *Start new acquisition*. The software will spent a few seconds loading the *visual words* data and then the video frames will be displayed in the camera frame panel.

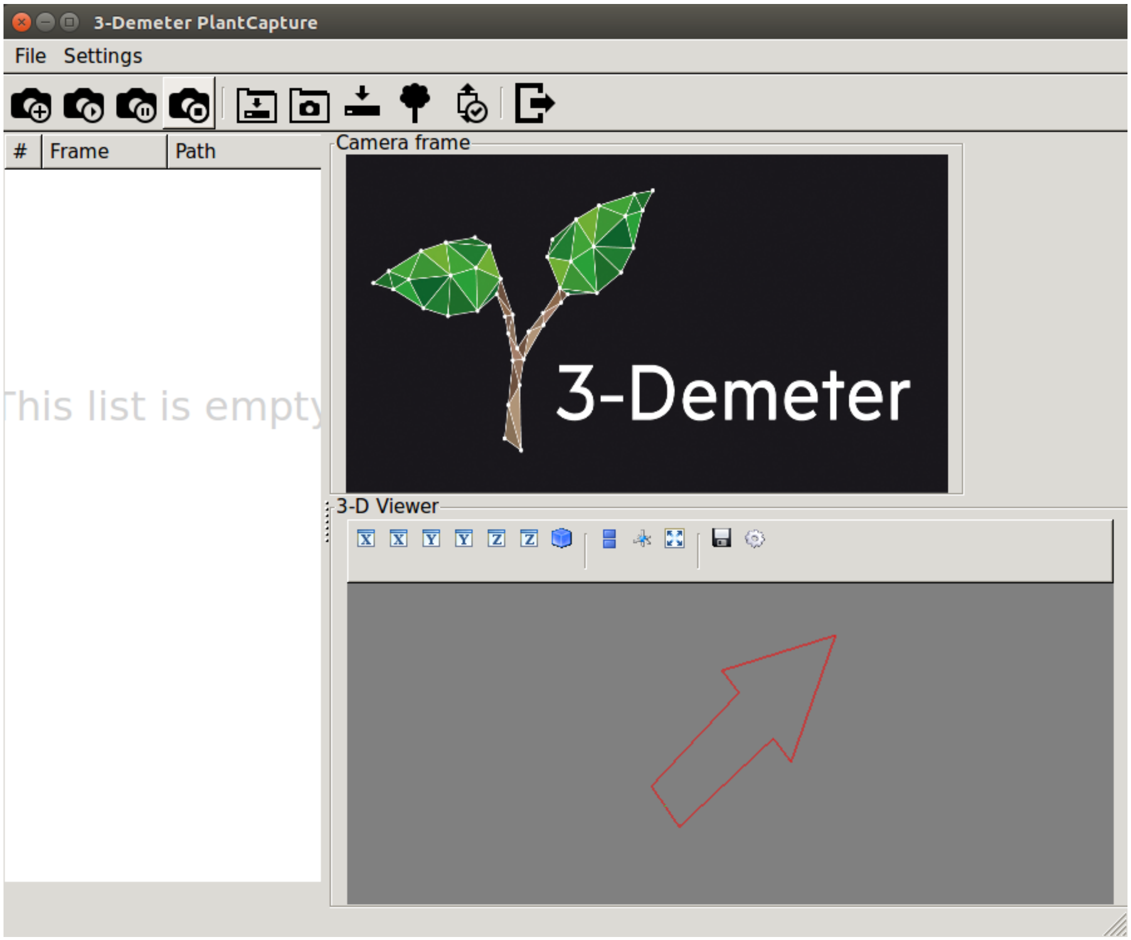

Fig. 3.1: The `capture` application main screen.

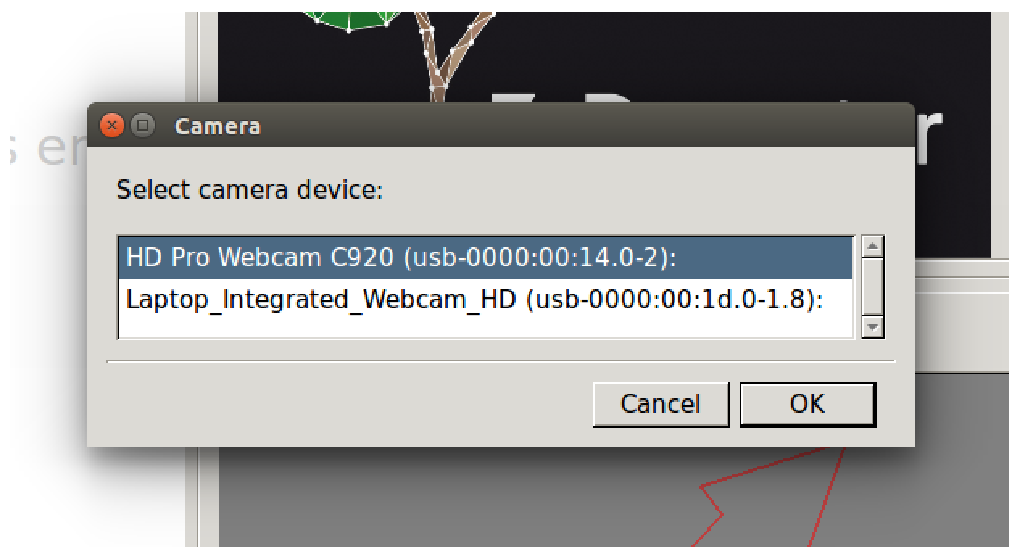

Fig. 3.2: The camera selection dialog.

.

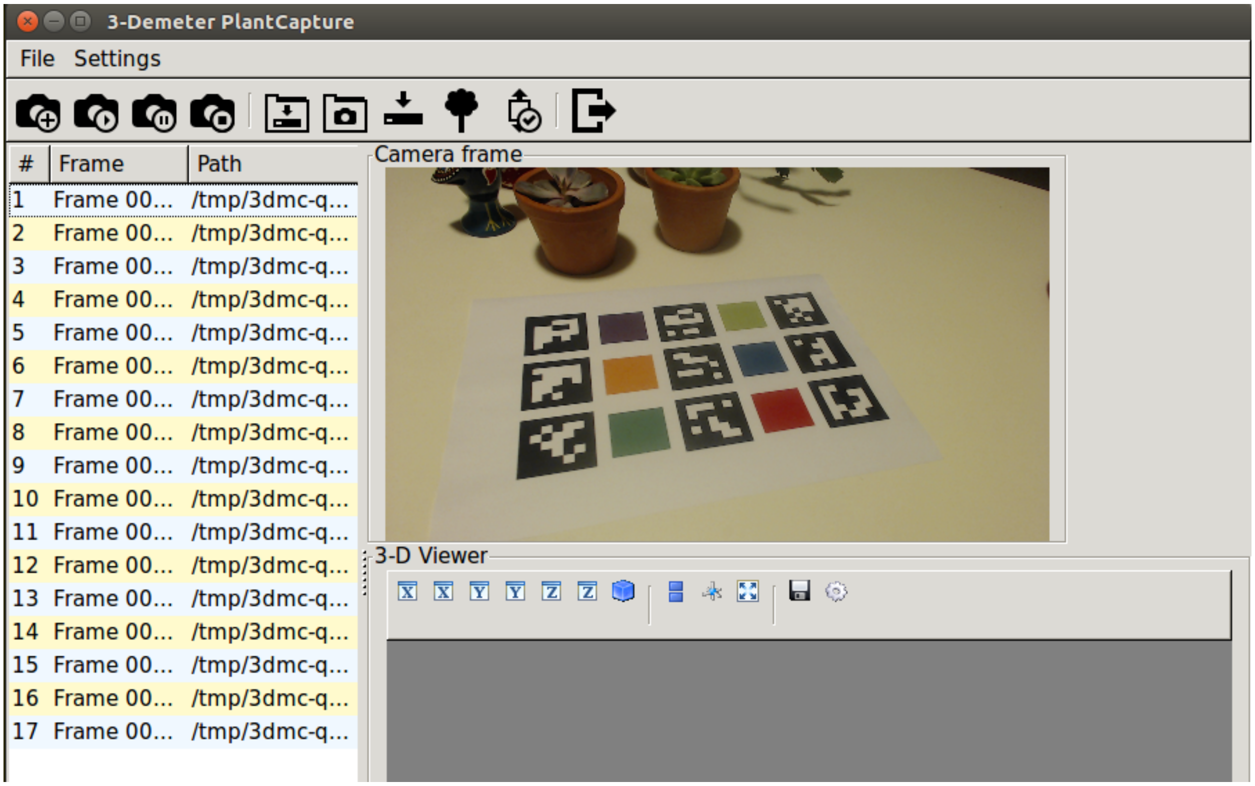

Fig. 3.3: The capture process. The current frame shows the scaling pattern. However, it is not necessary to have the pattern visible in every frame.

3.1 3-D map initialization

A very important step is the *map initialization*, when the visual odometry system finds the first 3-D points in the scene by stereo vision. The further localization and mapping routines will rely in this initial 3-D map. To get a good initial map, consider these tips at the **very beginning** of the image acquisition:

- translates the camera slowly from left to right;
- look for textures and salient points, avoid homogeneous surfaces;
- avoid point the camera to a single plane, look for regions displaying objects in different planes.

Once the map is initialized, frames will start to appear in the list at the left of the application window. After that, you can move the camera with more freedom, employing rotations and approximations. However, continue to avoid very fast camera movements. In the case the tracking is lost, move the camera near to a previously visited location: ORB-SLAM will then perform *relocalization*. You can use the pause and resume buttons and the relocalization feature to take a break in long acquisitions steps.

Don't forget to take a few frames of the scaling pattern. There is *no need* to make the pattern visible in every frame, a few frames are sufficient for further scaling and rotation (Figure 3.3).

Pressing *Finish acquistion* will stop the acquisition procedure. You can use the *Save capture files to...* button to save the data to your prefered path.

CHAPTER

FOUR

MULTIPLE VIEW STEREO WITH PMVS

The *Export files to the MVS subsystem* button will create the files needed by PMVS to perform the multiple view stereo step. After that, you can use *Start 3-D reconstruction* to start PMVS [3] (Figure 4.1). You could also run PMVS directly from the shell other time, avoiding the 3DMCAP interface or employing a PMVS instance running in a more powerful machine, just using the files exported by 3DMCAP.

PMVS will create a point cloud (Figure 4.2), stored as a PLY file in `pmvs/models/3dmc-3dmodel.cfg.ply` at the working directory.

4.1 Scaling

Normalize scale and orientation is optional and depends on good images of the scaling pattern.

A successful scaling will produce a PLY file in `pmvs/models/3dmc-3dmodel.norm.ply`. Assuming that `ref_distance_mm` was properly set and 3DMCAP was able to identify the scaling pattern, users will be able to perform measurements using the point cloud from this PLY in millimeter scale. The normalization step will also orient the 3-D scene, making the scaling pattern the origin of the 3-D space and making the Z-axis to point upward, orthogonal to the pattern plate.

4.2 Exploring your point cloud using Meshlab

You can explore your point clouds in different applications. For example, Meshlab[1] is a great tool used to explore and manipulate 3-D models (Figure 4.3). Meshlab provides tools for editing, measuring, polygonal meshing and surface fitting, allowing further analysis based in the clouds created using 3DMCAP.

[1] See http://www.meshlab.net.

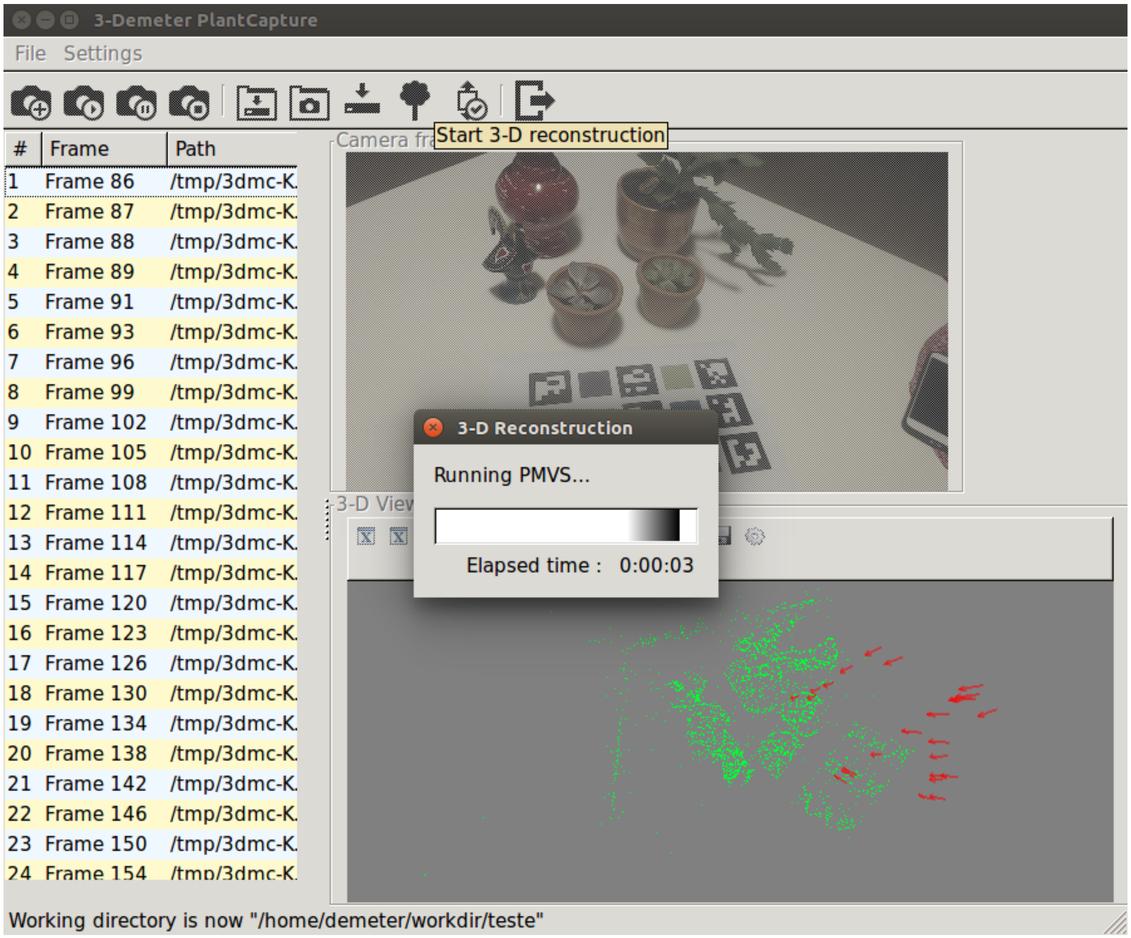

Fig. 4.1: The multiple view stereo step.

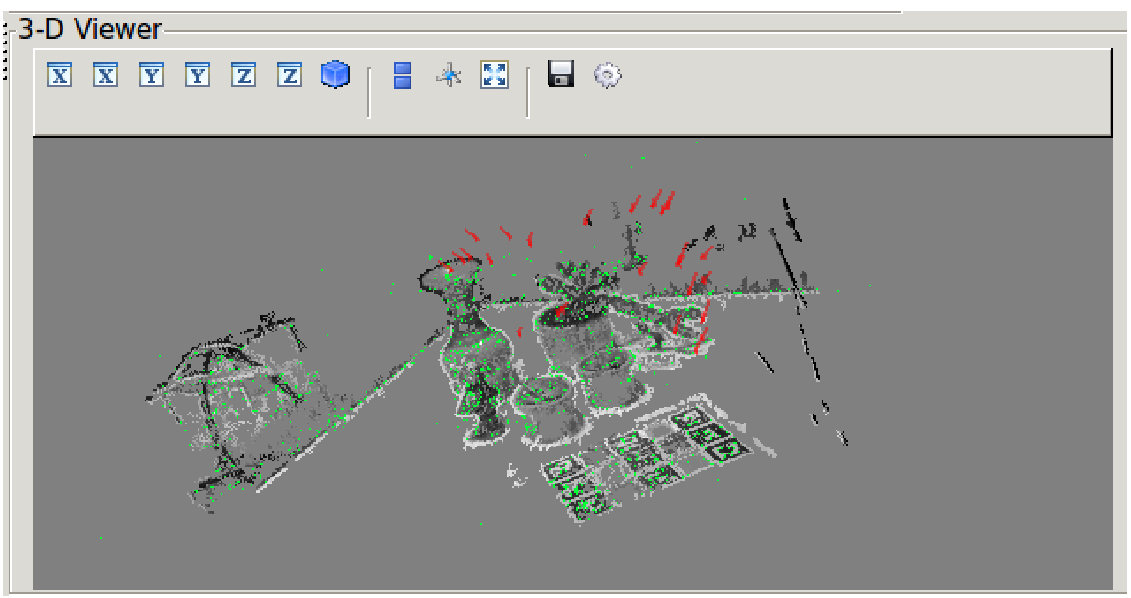

Fig. 4.2: Dense point cloud shown in the application 3-D viewer.

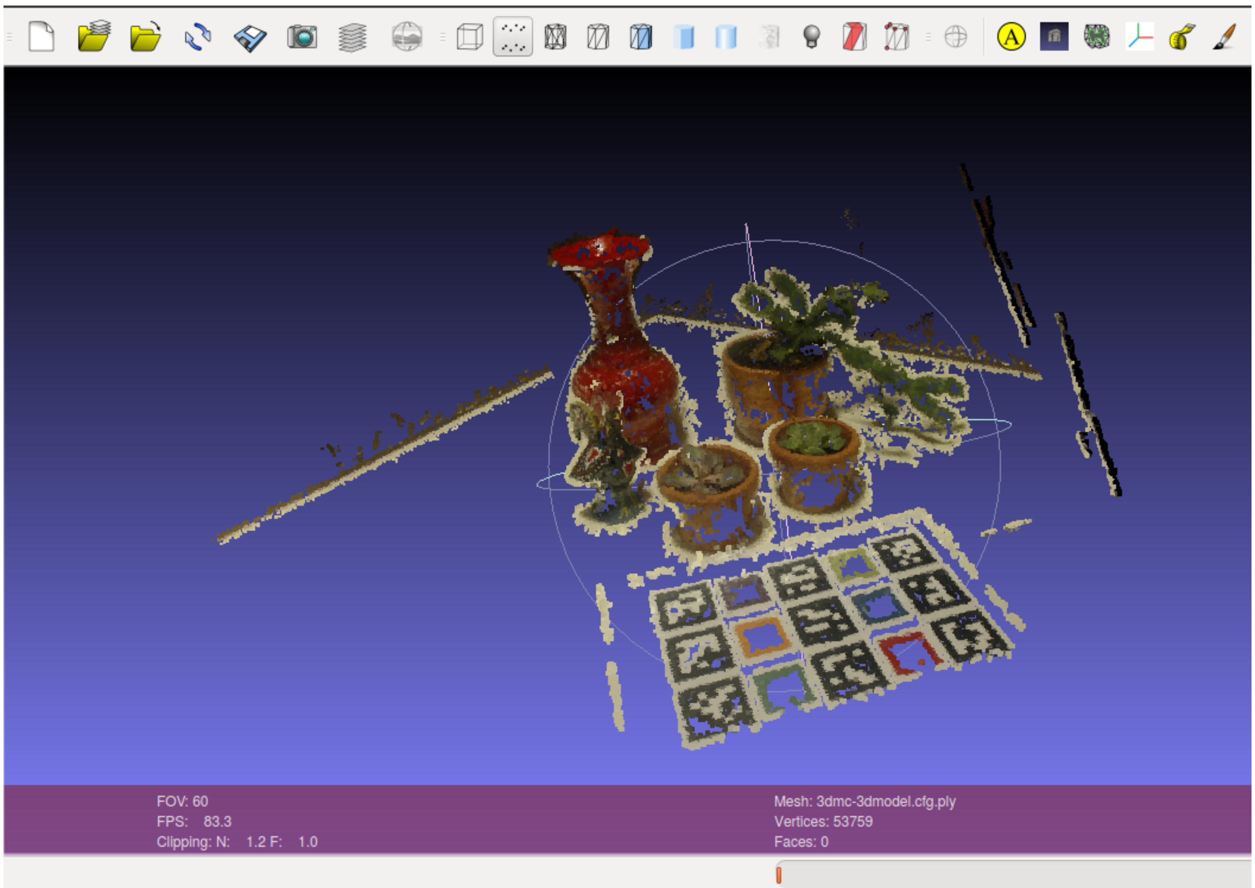

Fig. 4.3: Point cloud as shown in Meshlab.

[1] T. T. Santos, L. H. Bassoi, H. Oldoni, and R. L. Martins, "Automatic grape bunch detection in vineyards based on affordable 3D phenotyping using a consumer webcam," in *Anais do XI Congresso Brasileiro de Agroinformática (SBIAgro 2017)* (J. G. A. Barbedo, M. F. Moura, L. A. S. Romani, T. T. Santos, and D. P. Drucker, eds.), no. October, (Campinas), pp. 89–98, Unicamp, 2017.

[2] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, Oct 2015.

[3] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1362–1376, Aug 2010.