Quick-Start Guide

Created 26 July, 2013 -- Adam Harmat
Updated 2 March, 2014 -- Michael Tribou

MCPTAM was originally developed on Ubuntu 10.04 LTS with ROS Diamondback, but has since switched to Ubuntu 12.04 LTS with ROS Hydro. It might work with other versions of Ubuntu and ROS but this has not been tested, so it is recommended that 12.04 LTS and Hydro be used. The following instructions assume you have these two requirements met.

A list of required ROS packages is given below:

```
$ sudo apt-get install ros-hydro-image-transport ros-hydro-image-transport-plugins ros-hydro-pcl-ros ros-hydro-libg2o
```

1. Prerequisites Installation

Since MCPTAM is an extension of PTAM, we use the same image, math, and GUI libraries, namely libCVD, TooN, and GVars3. Installing libCVD, TooN and GVars3 the same process as for PTAM. The installation order matters and must follow:

1. TooN
2. libCVD
3. GVars3

1.1 Install TooN

Get the latest version of TooN from the website (http://www.edwardrosten.com/cvd/toon.html). MCPTAM was developed with TooN v2.1 but also works with TooN v2.2. Installation is the usual configure, make, make install. There isn't really anything to make as it's just a bunch of headers. Follow the installation instructions on the target website.

```
$ ./configure
$ make
$ sudo make install
```

1.2 Install libCVD

Get the latest version of libCVD from the website (http://www.edwardrosten.com/cvd/). MCPTAM was developed with release 20121025. Installation is the usual procedure of configure, make, make install. Follow the installation instructions on the target website. The following options are recommended for configure:

```
$ export CXXFLAGS=-D_REENTRANT
$ ./configure --without-ffmpeg
$ make
$ sudo make install
```

1.3 Install GVars3

Get the latest version of GVars3 from the website (http://www.edwardrosten.com/cvd/gvars3.html). MCPTAM was developed with version 3.0. Installation is the usual procedure of configure, make, make install. Follow the installation instructions on the target website. The following options are recommended for configure:

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

2. MCPTAM Installation

2.1 Building MCPTAM

(It is assumed that you are using ROS Hydro and the Catkin build system)

Set up the ROS environment.

```
$ source /opt/ros/hydro/setup.bash
```

Create a Catkin workspace for the package or use one you've defined previously.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src/
$ catkin_init_workspace
```

Set up your Catkin workspace.

```
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

Download the MCPTAM source code into the src/ folder of the workspace.

```
$ cd ~/catkin_ws/src/
$ svn co <url> mcptam
```

Compile the MCPTAM source code with catkin_make.

```
$ cd ~/catkin_ws/
$ catkin_make
```

3. Camera Intrinsic Calibration

To calibrate the intrinsic parameters of the cameras for the Taylor camera model, run the camera_calibrator program in the mcptam package.

Create a folder in the MCPTAM path for saving these intrinsic calibrations.

```
$ mkdir ~/catkin_ws/mcptam/calibrations/
```

An example launch file launch/camera_calibrator_usb.launch is given below:

```
<launch>

<!-- Camera Node -->
<node name="$(arg camera_name)" pkg="uvc_camera" type="uvc_camera_node" output="screen" ns="$(arg
camera_name)">
  <param name="device" type="string" value="$(arg device)" />
  <param name="camera_info_url" type="string" value="package://mcptam/calibrations/$(arg camera_name).yaml" />
</node>

<!-- camera_calibrator node -->
<node name="camera_calibrator" pkg="mcptam" type="camera_calibrator" output="screen">
```

```
  <param name="cam_name" type="string" value="$(arg camera_name)"/>
  <param name="image_transport" type="string" value="compressed"/>
  <param name="image_topic" type="string" value="image_raw" />
  <param name="info_topic" type="string" value="camera_info"/>
</node>

</launch>
```

This launch file needs to be run with the name of the camera as the command-line parameter.

```
$ roslaunch mcptam camera_calibrator.launch camera_name:=camera1 device:=/dev/video0
```

This will launch the calibrator using the camera at location /dev/video0 with the camera name camera1.

This camera_calibrator node will look for images on the topic /<camera_name>/image_raw and will attempt to save the calibrated parameters at the end with a call to the ROS service server at /<camera_name>/set_camera_info. Note that the default image topic is always image_raw for any of the programs in the mcptam package, so the above <param> tag was not necessary, it is there just as an example.

The intrinsic parameters of the cameras are stored in corresponding calibrations/<camera_name>.yaml files.

When running camera_calibrator, you need to point the camera at a checkerboard (of any size/shape) and when the checkerboard is detected, press [Spacebar] to save the image. Collect 10-20 images from different viewpoints and then hit the Optimize button. When you are satisfied with how low the reprojection error has become or it has stopped changing, press the Save button to save the calibration to the yaml file.

Run the camera_calibrator node for each of the cameras in your cluster. Make sure that each has a unique <camera_name>, which will identify it in the rest of the MCPTAM system.

4. Camera Extrinsic Calibration

Once all the camera intrinsic parameters are calibrated, it's time to run the pose_calibrator to get the extrinsic parameters (translation and orientation of the cameras within the camera cluster). Create a folder for the extrinsic calibrations of the component cameras.

```
$ mkdir ~/catkin_ws/src/mcptam/poses/
```

To run the pose_calibrator, an example pose_calibrator.launch file is below:

```
<launch>

<!-- pose_calibrator node -->
<node name="pose_calibrator" pkg="mcptam" type="pose_calibrator" clear_params="true" output="screen">
  <rosparam command="load" file="$(find mcptam)/groups/$(arg group_name).yaml" />

  <remap from="reset" to="pose_calibrator/reset" />

  <param name="image_transport" type="string" value="compressed"/>
  <param name="pattern_width" type="int" value="6"/>
  <param name="pattern_height" type="int" value="8"/>
  <param name="square_size" type="double" value="0.024"/>
  <param name="finder_max_ssd_per_pixel" type="int" value="500" />
  <param name="kf_distance_mean_diff_fraction" type="double" value="0.0" />
  <param name="kf_adaptive_thresh" type="bool" value="false" />

  <param name="pose_out_file" type="string" value="$(find mcptam)/poses/poses.dat" />
</node>

</launch>
```

This launches the pose_calibrator node using the camera group specified in the file groups/<group_name>.yaml. The groups of cameras that compose the cluster are specified in this file. A camera cluster consists of camera groups. Camera groups are made of individual component cameras that are to be synchronized in time. For example, to specify a camera cluster composed of two groups with two cameras each, the yaml file would be written:

```
---
cam_group_list: [[camera1,camera2],[camera3,camera4]]
```

where camera1, camera2, camera3, and camera4 are the names of the four cameras in the cluster. As of writing, MCPTAM only fully supports clusters with a single camera group. Therefore, create the groups folder,

```
$ mkdir ~/catkin_ws/src/mcptam/groups/
```

and the appropriate group.yaml file.

```
---
cam_group_list: [[camera1,camera2,camera3,camera4]]
```

The pose_calibrator.launch file will remap the reset service to pose_calibrator/reset (needed for internal implementation reasons). Setting the image_transport parameter to compressed means that we are subscribing to compressed images on the topic cameraXYZ/image_raw/compressed. The pattern_width and pattern_height parameters set the width and height of the checkerboard (*number of inside corners*, NOT the number of squares). The square_size parameter is the edge length of one checkerboard square in meters. The kf_adaptive_thresh parameter dictates whether we are using adaptive of fixed threshold for extracting image features. For calibration in a nice indoor environment it's best to use fixed thresholds. The pose_out_file parameter specifies the file to which the poses will be written. In this example, the optimized poses for the cameras are stored in the file poses/poses.dat. By default, the image topic will be image_raw and the intrinsic calibration topic will be camera_info.

To perform the actual calibration, the launch file can be run using:

```
$ roslaunch mcptam pose_calibrator.launch group_name:=group
```

Once running, point the first camera (image outlined in bright blue) at the checkerboard until it is found, then press [Spacebar]. Slowly translate the camera while keeping the checkerboard in view until the second KeyFrame is added to the map. During this initial translation the tracking is sensitive because it is tracking very few points (the checkerboard corners). After the second KeyFrame is added, a lot more points (including natural point features in the surrounding environment) get generated and tracking becomes more stable. Move the camera rig so that the second camera is now facing the checkerboard while the first camera builds and maintains track of its world map, then press [Spacebar] to bring the second camera into the map generated by the first camera. It should be initialized into the same map as the first and you should see coloured dots indicating found features pop up on the second camera's image right away. Continue this process until all cameras have been initialized, then move the rig around some more to collect extra data.

Once all of the cameras are added into the same world map, it is important to have them undergo motion for which the translations and rotations around the environment are as large as possible to sufficiently constrain the calibration and produce accurate results. Ensure that the camera cluster is rotated about all of the coordinate axes and remember that the calibration sensitivity goes up as you get closer to the point features in the environment, so be sure to operate close to the point features in the environment.

When satisfied with the collected motion, press the Optimize button to calibrate the inter-camera poses. Allow the optimization to run for a good amount of time until the parameters stabilize and the residual error is sufficiently small. Once achieved, press the Save Calib button to write them out to the poses.dat file, as well as embedding the extrinsic calibration parameters into the camera's camera_info structure using the camera's set_camera_info service. This will write the parameters out to the individual camera calibration .yaml files.

MCPTAM uses some of the extra matrices in the camera_info message to transmit the extrinsic calibration parameters alongside the intrinsic parameters and camera images. The rotation of a camera's coordinate frame with respect to the cluster coordinate frame is found in the rectification_matrix and the translation is found in the projection_matrix.

```
sensor_msgs::CameraInfo infoMsg;

Rotation = [ infoMsg.R[0] infoMsg.R[1] infoMsg.R[2] ]
           [ infoMsg.R[3] infoMsg.R[4] infoMsg.R[5] ]
           [ infoMsg.R[6] infoMsg.R[7] infoMsg.R[8] ]
```

```
translation = [ infoMsg.P[3]  ]
              [ infoMsg.P[7]  ]
              [ infoMsg.P[11] ]
```

As a result, the camera_info messages used with MCPTAM in this way are incompatible with other ROS nodes which make use of these fields in the camera_info messages. It is therefore useful to store the camera .yaml files in the local mcptam/calibrations folder.

You are now ready to use MCPTAM.

5. Running MCPTAM

Here is an example launch file for the mcptam node.

```
<launch>

<!-- mcptam node -->
<node name="mcptam" pkg="mcptam" type="mcptam" clear_params="true" output="screen">
  <rosparam command="load" file="$(find mcptam)/groups/$(arg group_name).yaml" />
  <remap from="reset" to="mcptam/reset" />
  <param name="image_transport" type="string" value="compressed"/>

  <param name="get_pose_separately" type="bool" value="false" />

  <param name="mkf_distance_mean_diff_fraction" type="double" value="0.5" />
  <param name="mm_max_scaled_mkf_dist" type="double" value="0.3" />
  <param name="mm_outlier_multiplier" type="double" value="1.1" />
  <param name="mm_init_cov_thresh" type="double" value="1.0" />
</node>

</launch>
```

This file will launch the mcptam node and load the camera groups from the groups/<group_name>.yaml file. The parameter get_pose_separately being false specifies that the camera_info messages from the cameras will contain the extrinsic calibration parameters in the messages themselves.

Another example of a launch file is presented, which loads the extrinsic calibration parameters from a file instead of from the camera_info messages.

```
<launch>

<!-- mcptam node -->
<node name="mcptam" pkg="mcptam" type="mcptam" clear_params="true" output="screen">
  <rosparam command="load" file="$(find mcptam)/groups/$(arg group_name).yaml" />
  <remap from="reset" to="mcptam/reset" />
  <param name="image_transport" type="string" value="compressed"/>

  <param name="get_pose_separately" type="bool" value="false" />
  <param name="camera_pose_file" type="string" value="$(find mcptam)/poses/poses.dat" />

  <param name="mkf_distance_mean_diff_fraction" type="double" value="0.5" />
  <param name="mm_max_scaled_mkf_dist" type="double" value="0.3" />
  <param name="mm_outlier_multiplier" type="double" value="1.1" />
  <param name="mm_init_cov_thresh" type="double" value="1.0" />
</node>

</launch>
```

The extrinsic calibration is loaded from the file poses/poses.dat and overwrites the pose extracted from the camera_info message for each camera.

Once the mcptam node is running, press [Spacebar] to start tracking. If you are subscribing to any camera groups with more than one camera in it, MCPTAM will wait until it receives data from one of these multi-camera groups before initializing using the stereo effect. If there is no field-of-view overlap between these images, MCPTAM will initialize immediately but requires sufficient motion (both translation and rotation) to converge to an accurate solution in map size and shape. Note that asynchronous operation of a cluster composed of several camera groups is a bit touchy and hasn't been as well tested as synchronous operation of several cameras or monocular operation.