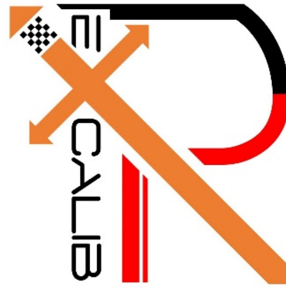

16-682 - MRSD Project II | ILR #09

Individual Lab Report #09 |March 23, 2017

SAMBUDDHA SARKAR

Team G

eXcalibR

Huan-Yang Chang
Man-ning Chen
Sambuddha Sarkar
Siddharth Raina
Yiqing Cai

1. INDIVIDUAL PROGRESS

1.1 Overview

In this ILR I don't many visuals or results to show as most of the work has been on the back-end of blender's python API, writing custom scripts to include functionality and ease of use required for validation of the geometric calibration algorithm.
The virtual environment is being modeled in an open source platform: Blender 3D 7.68a. It is a Maya based platform and is programmable by Python 3.

Topics covered have been listed below for a quick overview.

1.2 Mapping the geometric locations on the checkered pattern
1.3 Blender Render Pipeline Status

1.2 Mapping the geometric locations on the checkered pattern

The task of physically relating the coordinates of the checker pattern on the calibration target to the world frame and/or reference points is crucial to validate the geometric calibration algorithm. Though it may seem to be an easy, to assign and compute the transformations of the different faces of the icosahedron with respect to a reference (world frame or local frame), it isn't when it comes to making it part of a user-friendly pipeline. The image in Fig.1.2.1 shows a snapshot of some of the faces of the calibration target with different local frames of reference.

The process of marking the geometric locations of the face patterns involves the assignment of a local reference frame on the calibration target (any one vertex of the target), then assigning each face of the target a local reference frame and finally relating all these frames to the world frame. The reference frame for each individual face is set according to some parameters relating to the geometry of the calibration target itself. The Z axis is aligned along the face normal, the X-Y axes are chosen such that each face's geometric pattern can be described using just on single description file.

The description file output (with custom .dsc, .calib extensions) which has been generated using the bpy module I have written and developed has been shown below in Fig. 1.2.2. The Fig. 1.2.3 gives an infographic on what each parameter means (like the April tag information of each face.).

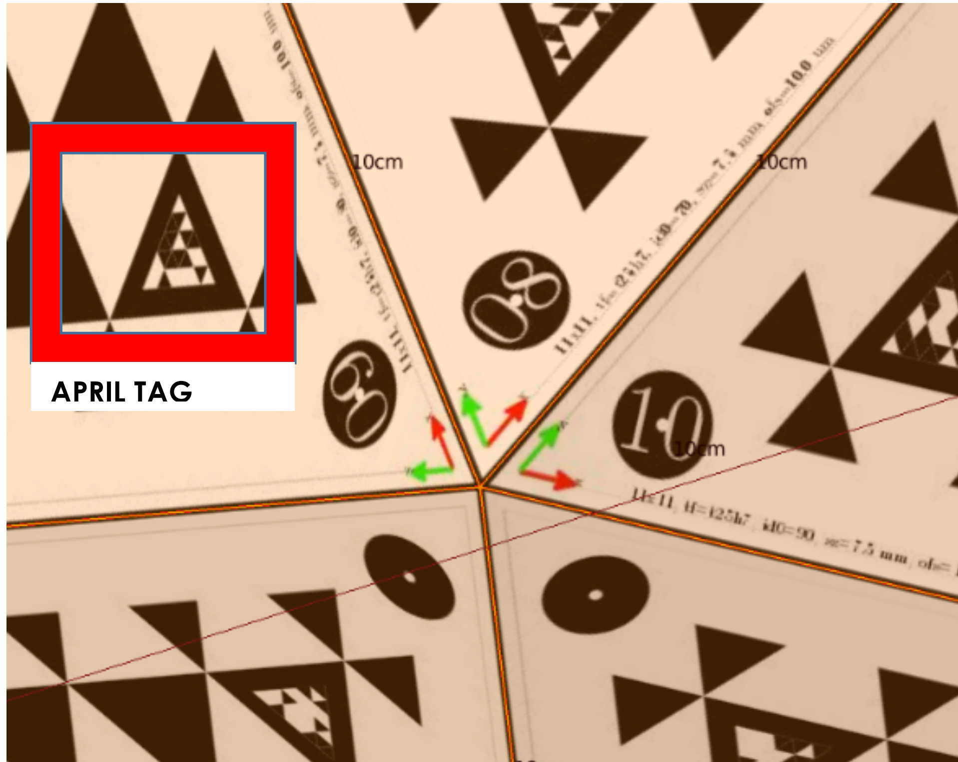

Fig. 1.2.1, Snapshot of pattern.

File Edit Selection Find View Goto Tools Project Preferences Help

icosahedron.calib | icosahedron.dsc | icosahedron_v2.calib

```
1   0,13,0,7.500000
2   t25h7,1.500000
3   -1,0,10,5.000000,7.216878,0.000000
4   9,0,9,4.625000,6.567359,0.000000
5   -1,1,9,5.375000,6.567359,0.000000
6   -1,0,8,4.250000,5.917840,0.000000
7   -1,1,8,5.000000,5.917840,0.000000
8   -1,2,8,5.750000,5.917840,0.000000
9   -1,0,7,3.875000,5.268321,0.000000
10  -1,1,7,4.625000,5.268321,0.000000
11  -1,2,7,5.375000,5.268321,0.000000
12  -1,3,7,6.125000,5.268321,0.000000
```

Fig. 1.2.2, Descriptor Output.

The first 2 lines in the descriptor output correspond to the face identification. The rest are the coordinates of the corners of the calibration pattern in local frame of reference of each face of the target. Fig. 1.2.3 elaborates with an infographic.

Padding: (X & Y local axis)
Print preset: Pattern printing information relating to tolerances in terms of microns
ID: Local ID of vertices of pattern for validation recall
(X, Y, Z): Locations of vertices in local XYZ coordinates

1. Face ID, #Padding, #Print-Preset
2. #April Tag ID, #Print-Preset
3. ID-V1, ID-V2, ID-V3, X,Y,Z

```
1   0,13,0,7.500000
2   t25h7,1.500000
3   -1,0,10,5.000000,7.216878,0.000000
```

Fig. 1.2.3, Legend of Information.

1.3 Blender Render pipeline

The blender rendering pipeline (Fig 1.3.1) is being developed to aide semi-autonomous generation of image data sets for the geometric calibration algorithm. The calibration pipeline illustrated is self-explanatory. The pipeline loads in camera data (intrinsics & extrinsics), object data (calibration target), required configuration and the number of the cameras and the render settings. It spews out Images and other mesh data using the cycles render engine and bpy module. The current progress is shown in Fig. 1.3.2.

The light coloration is to imply that the task has been inherited since the last PR.

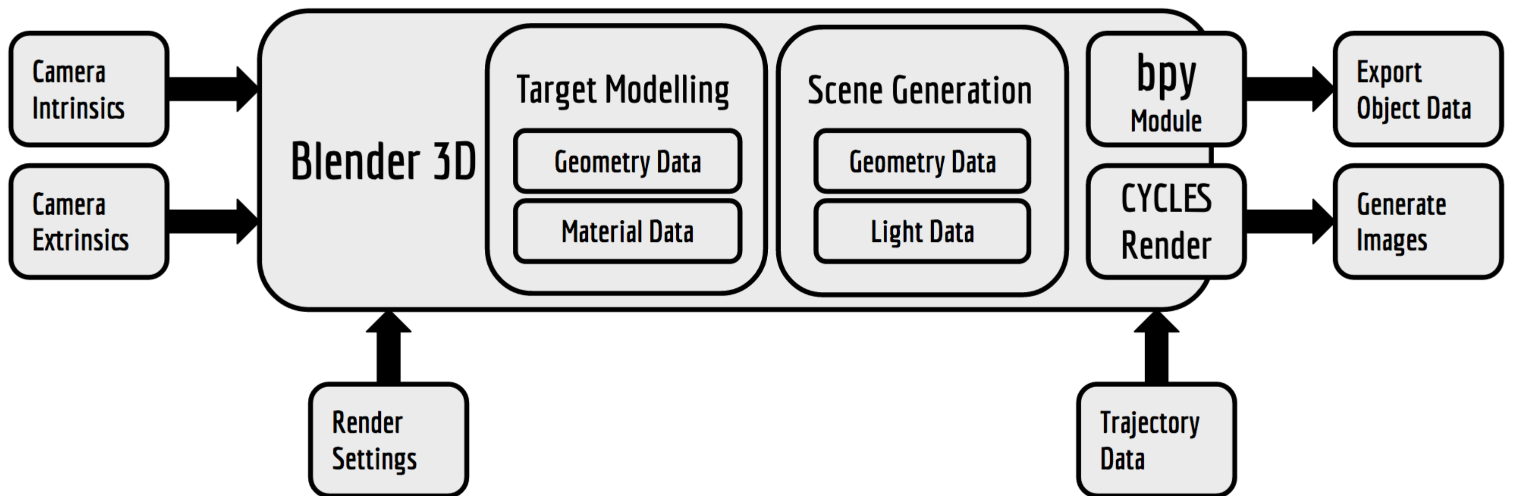

Fig. 1.3.1, Overall Pipeline.

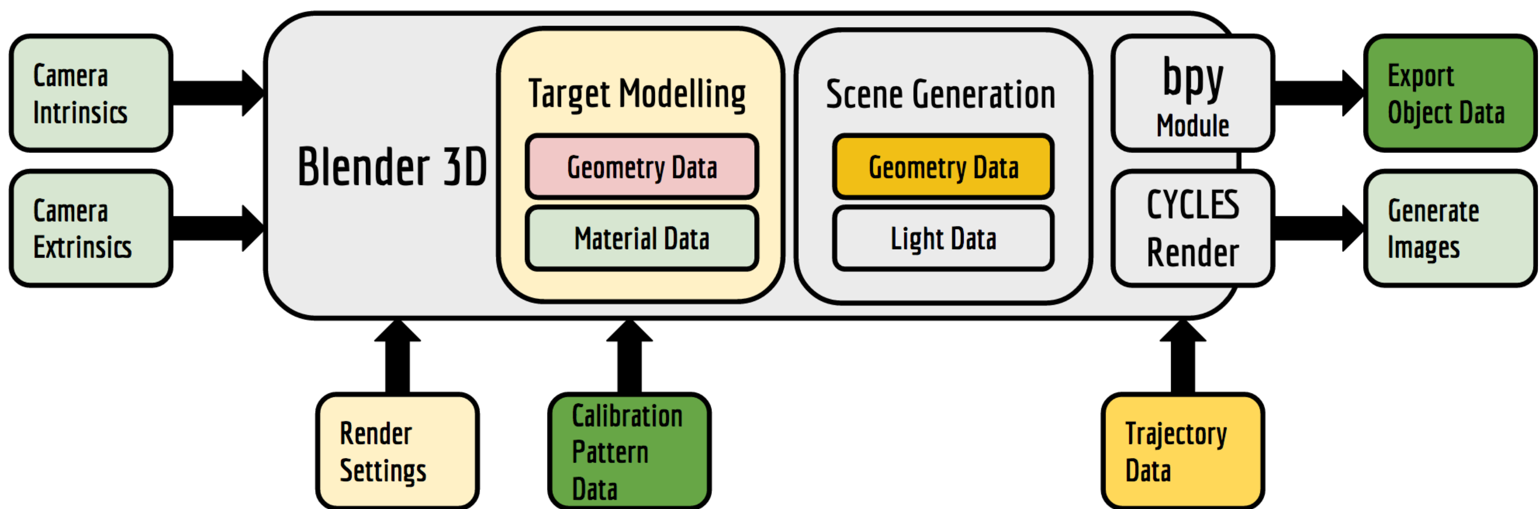

Fig. 1.3.2, Current progress on the pipeline.

2. CHALLENGES

The main challenge is the extraction of crucial data from the blender environment using the python API. It is time consuming as I have to go through a lot of documentation to figure out how data was being stored and handled. Thus accessing, modifying and creating data inside the blender environment using scripts is a long process. It so happened that I accidentally corrupted a native blender filesystem relating to the libraries which stores the 3D mesh data, and in doing so corrupted all my back-ups in one stroke.

3. TEAM WORK

The project work was divided among the team members and the task was assigned according to the strengths of the team members. The task division has been listed below in Table 3. The divided tasks can be completed in parallel; hence others can pitch in when some team members fall behind in their work.

Team Member	Task
Huan-Yang Chang	Path optimization for ABB Robot arm.
Siddarth Raina	Shot Noise and PRNU correction.
Man-nig Chen	Color Calibration: Mapping function.
Yiqing Cai	Multiple Camera Model: Scoring heuristics for the original path.
Sambuddha Sarkar	Blender Render pipeline and pattern mapping.

Table 3, Task Division.

4. FUTURE PLANS

My future plans until the next progress report is to complete the porting modules for import and exporting geometry data in and out of blender. Most of the work is being conducted in the back-end of blender using the blender-python API (bpy) so the results may not be tangible representations of the actual amount of work being done.