Swept-plane Scanner: Team Plan

Team Details

Team members: Dylan, Arnav, Wisoo, Kaihong
Team slack channel in ES 143 workspace (includes instructors): #team-turtle
URL for live version of this google doc:
https://docs.google.com/document/d/1dSvfeDLauBAf2gaQ67dufuJZYOoouH81dcelnETRUJ8/edit?usp=sharing
Deepnote link:

Team communication plan

- *We will work in a single Deepnote project. We will use the slack channel to communicate with each other. When we complete our tasks, we will let one another know. Dylan will take your tasks if re-allocations are needed.*
- *Do not edit .ipynb files that you have not been assigned to. If anyone has suggestions, communicate through Slack & leave comments (e.g "# please fix line ~") on the relevant .ipynb for the author.*
- *The master notebook is named "swept_plane.ipynb". (https://deepnote.com/project/57541c7c-0d5c-4e50-9836-695e7044e86d#%2FHarvard_3d_scanner_team_turtle%2Fswept_plane.ipynb)*

Responsibilities

What	Who (Main + Assistant)	When (completion deadline)
Coordinate python functions and maintain master jupyter notebook. (Teammates with github experience are good for this task.)	Wisoo	Friday 10/23 (Complete)
Capture initial test dataset: calibration images (20) and shadow video. Report calibration reprojection error. ZIP and post to dropbox. Share link in Slack. (Teammates who are handy and careful are good for this task.)	Dylan	Sunday 10/25
Manually create object and plane masks. For each scanning dataset, manually draw a pixel-mask that indicates the object region, and two rectangular	Kaihong	Monday 10/25

pixels masks that indicate the unoccluded regions of the horizontal and vertical AprilBoards. Store these masks as a binary (0,1) HxW arrays that accompany the corresponding shadow video. (Teammates with experience with paint/editing programs like photoshop are good for this task.) Three masks: one for object, one for top board, one for bottom board.		
calibrateCamera() [easy]	Arnav	Tuesday 10/26
computeTwoPlanes() [hard]	Wisoo	Wednesday 10/28
processVideo() [hard]	Dylan	Wednesday 10/28
computeShadowPlanes() [moderate]	Arnav + Dylan	Thursday 10/29
computeObjectPoints() [moderate]	Arnav + Dylan	Thursday 10/29
visualizeObjectPoints() [moderate]	Kaihong	Thursday 10/29

Python functions

calibrateCamera() [easy]

Purpose: Estimate camera calibration (calMatrix (K) and distCoeffs). Needs to be run once for each scanning dataset. Lightly modified version of 7_CalibrateCamera.ipynb.
Input:
- ZIP archive with calibration images
- ES 143 AprilBoard datafile (AprilBoards.pickle)

Output:
- Camera calibration (calMatrix (K) and distCoeffs)

Verify correctness by:
- *Project points back and see if they're the same. And we can read the RMSE error data.*

computeTwoPlanes() [hard]

Purpose:
1) Estimate horizontal and vertical planes (\Pi_v, \Pi_h) from one image of the setup. Needs to be run once for each scanning dataset (each video). Use detect_aprilboard() function from 7_CalibrateCamera.ipynb. Also use openCV function cv2.solvePnP().

Input:
- Camera calibration: `calMatrix` and `distCoeffs` `(outputs from calibrateCamera()`
- Image of the two AprilBoards setup with object to be scanned

- ES 143 AprilBoard datafile (AprilBoards.pickle)

Output:

- `plane_h, plane_v` -- Two homogeneous 4-vectors corresponding to planes (\Pi_v, \Pi_h) in the camera-centered coordinate system.

Verify correctness by:

- *Using the plane normals \Pi_v and \Pi_h, visualize the intersection line of the two planes and see if it matches the join of the two boards on the image.*

processVideo() [hard]

Purpose: Detect the shadow edges in a shadow video. Needs to be run once for each scanning dataset. Follow the algorithm (and perhaps reference Matlab code) on the scanner documentation page. Also use line-fitting code from previous ES 143 line-fitting assignments.

Input:

- Video file (HxWxT=time)

Output:

- `front_edge_times --` HxW array with each element storing the time (in units of frame number) at which the leading shadow edge crosses that pixel.
    - Optionally, a second HxW array with each element storing the time (in units of frame number) at which the *trailing* shadow edge crosses that pixel.
    - Entries in these arrays that are NaN or (-1) indicate that no shadow edge was detected at the pixel.
- `image_lines --` length-T list of 2x3 arrays. The ith 2x3 array contains the homogeneous coordinates of the two image lines (as rows) on the horizontal and vertical planes in the ith video frame.

Verify correctness by:

- *Create a new video where the shadow edge is highlighted.*

computeShadowPlanes() [moderate]

Purpose: Estimate shadow plane for each video frame. Uses SVD and the math from 7 | Pre-session B.

Input:

- Length-T list (T = number of video frames) `image_lines`
- Planes `plane_h, plane_v`
- Camera calibration: `calMatrix` and `distCoeffs`

Output:

- `shadow_planes` -- length-T list of homogeneous 4-vectors, with the ith list element storing the coordinates of the shadow plane (in the camera-centered coordinate system) for the ith video frame

Verify correctness by:

- *Visualize shadow line, or check that error is close to 0 from SVD.*
- *If you use the minimal number of points to constrain problem, there should be an exact match.*

computeObjectPoints() [moderate]

Purpose: Estimate 3D point coordinates for every pixel within the object region of a dataset. Uses back-projection of image points and line-plane intersection (see 7 | Class Session A).
Input:

- `shadow_planes`
- Camera calibration: `calMatrix` and `distCoeffs`
- `object_mask` -- a HxW array with binary (0,1) entries indicating which pixels correspond to the object. Drawn manually, once per dataset.

Output:

- `point_cloud` -- a HxWx3 array with (X,Y,Z) point coordinates at each object pixel. Values of Nan or -1 indicate pixels that are outside of the object mask.

Verify correctness by:

- *Use visualizeObjectPoints() and see if it looks right.*

visualizeObjectPoints() [moderate]

Purpose: Display the point cloud interactively using plotly, matplotlib, or another point-cloud library.
Input:

- `point_cloud`

Output:

- None

Verify correctness by:

- *Try it on something we know, like the checkerboard boxes. Also try it on our scanned objects and see if they look right.*