# Documentation of Source Code for Endoscopic Surface Normal and Shape Reconstruction Utilizing Specular Isophotes

### Folder 'ISBI2024':

Source codes for the estimation of mesh principal curvatures and directions. It serves for comparing the principal curvature ratio, and principal curvature directions estimated at specular brightest points from 2D images, with those computed from triangle meshes. **It assumes that 2D/3D correspondences are available**.

## Inputs and file formats:

1. r un specular_curvature.py : given a triangle mesh, it computes its principal curvatures and directions at each mesh vertex.
2. p rincipal curvatures are used as inputs to run the script "curvature.py" which deal with blender simulated colon-like data. The text file used as input contains 2D/3D correspondences. It is structured as follows; each line gives the vertex index, columns 3 to 6 represent the ground truth normal at that vertex, while columns 6 to 8 give the corresponding pixel coordinates in the image. For liver data, we provide the script "curvature_liver.py", for which, the input 2D/3D correspondences are encoded within a .mat file that can be loaded using the function `load_dot_mat_file` function as defined in the script library.py. For both cases, inputs are: 1. 2D image path, 2. camera intrinsics, 3. mesh path, 4. 2D/3D correspondences path, and 5. principal curvatures and directions already computed from the 3D model. Please find examples illustrating the structure of the input 2D/3D correspondences in the folder /ISBI2024/2D_3D_correspondences.

## Outputs:

"curvature.py" and "curvature_liver.py" provide histogram errors for estimated normals, principal curvature, and directions at specular brightest points in a comparison with their values computed from the corresponding 3D model.

### Folder 'ICRA2024':

The script 'ICRA2024_synthetic_data.py' processes a set of images to estimate normals and centroids of specular blobs based on elliptical contour fitting.

## Inputs:

1. **I ntrinsic Matrix** K (camera calibration).
2. **I mage Sequence Path** `data_path` (folder containing image set).
3. **C ontour Detection Level** and `dist_tolerence` for filtering contours based on size.

## Main Functions:

1. `robust_centroid_detection`: Detects contours in images and finds centroids of useful contours.
2. `normal_estimation_direct`: Estimates normals directly using the intrinsic matrix and centroid coordinates (sightline based method).
3. `fit_ellipse_lstsq`: Fits an ellipse to contour points for robust centroid positioning.
4. `Zisserman_method`: Provides dual normal solutions from ellipse fitting for normal disambiguation.

## Outputs:

1. **Excel Files** with estimated normals from centroid coordinates per image.
2. **Error Plots**: Visualizes angular errors for method comparison.

This setup provides robust normals and centroid estimates for specularities across images in the specified folder. These estimated normals were used as inputs to boost the NRSfM method, applied to colon images.

The script 'ICRA2024_colonoscopy.py' is a modified version which can deal with colonoscopy data. Contrary to synthetic data for which specularity segmentation is performed using a simple threshold, this script assumes the specularity masks to be provided as inputs. Moreover, angular error and eccentricity thresholding is applied with conditions allowing only those specularities with specific angular errors (≤1.0 degree) and eccentricity values (≤0.95) within the specified tolerance.

### Folder 'dependency_depth_refinement':

The script depth_refinement.py is structured as follows:

## Inputs

- **Images**: Gray-scale image input for normal estimation and depth refinement.
- **Specular Mask**: A binary mask indicating specular regions in the image.
- **Camera Intrinsics (K)**: Camera matrix for direct normal estimation.
- **Contour Points (Cx, Cy)**: Coordinates for known boundaries used in normal estimation.
- **Dilation Parameters**: Radius for dilation of masks.

## Main Functions

1. **Direct Normal Estimation**:
    - Computes normals from given pixel coordinates using camera intrinsics.

2. **Local Planarity Refinement**:
    - Fills in normal estimates based on local planarity assumptions to create a comparative baseline for evaluation.
3. **Iterative Relaxation**:
    - Refines normal and depth maps by iteratively solving the heat equation, leveraging the previously estimated normals.
4. **Masking and Dilation**:
    - Functions for creating foreground/background masks and performing morphological dilation.
5. **Error Analysis**:
    - Computes and visualizes angular errors and RMSE between refined normals and ground truth.

## Outputs

- **Refined Normals**: Updated normal vectors after applying the iterative relaxation method.
- **Refined Depth Map**: Depth map refined using the improved normals.
- **Visualizations**: Plots showing angular error, RMSE, and comparisons between methods.
- **Error Statistics**: Quantitative metrics (min, max, mean, std) of angular error and RMSE.

### Folder 'ISBI2023-IJCARS2023':

The provided code 'normal_reconstruction.py' processes images to detect contours, fit ellipses to these contours, and calculate normal vectors for the surfaces represented in the input images. The workflow is designed to analyze the geometric characteristics of the surfaces captured in the images.

**Main Inputs:**

1. **Image Path (`path`)**: The file path to the image that will be processed.
2. **Threshold Level (`level`)**: A value used for contour detection to filter out noise and enhance significant features.
3. **Isovalue (`isovalue`)**: Used to create a specular mask that distinguishes highlights in the image.
4. **Camera Intrinsic Matrix (`K`)**.

**Outputs:**

1. **Contour Data**: Detected contours derived from the input image, representing significant shapes or features.
2. **Fitted Ellipses**: Parameters of ellipses fitted to the contours, which describe the shape and orientation of detected features.

3. **Normal Vectors**: Calculated normal vectors for each fitted ellipse, indicating the orientation of the surface at those points.
4. **Excel File**: Contains detailed information about the fitted ellipses and their associated normals, useful for further analysis and documentation.
5. **Visual Plots**: Graphical representations of contours, ellipses, and normals overlaid on the grayscale image for visualization purposes.

**Principal Functions:**

- **Contour Detection and Fitting**: Functions such as `robust_elliptic_contours_thresholding` detect contours in the grayscale image, while functions like `fit_ellipse_Fitzgibbon` and `fit_ellipse_lstsq` fit ellipses to these contours.
- **Normal Vector Calculation**: Functions like `Zisserman_method` calculate normal vectors from the fitted ellipses, while `normal_disambiguiting` selects the appropriate normal based on specific conditions.
- **Visualization**: Various plotting functions visualize the contours, fitted ellipses, and normal vectors to facilitate understanding of the results.

**Workflow:**

The overall workflow begins by reading and processing the input image to detect and extract contours. The grayscale version of the image is then analyzed to fit ellipses to these contours, and the associated normal vectors are calculated. The results are visualized through plots and saved into an Excel file for documentation and future reference.

Unet_depth

## Folder ‘dependency_unet_depth’:

The script uncertainty_visualisation.py performs a series of operations to estimate depth and uncertainty maps from input images, utilizing a deep learning approach. It serves also for visualization of vector fields representing estimated normals at specular brightest points.

**Inputs:**

1. **Images:**
    - Original image and its downsampled version, which serve as the base for depth estimation.
2. **Brightest Points (BPs):**
    - A set of coordinates representing key features in the image, loaded from a `.npy` file. These points are crucial for depth and normal calculations.
3. **Camera Parameters:**
    - Intrinsic camera matrix (`K`) and distortion coefficients (`D`) are defined to rectify the images for accurate 3D reconstruction.
4. **Depth Maps:**

- Smooth depth maps are generated or provided for further processing, including interpolation at the brightest points.

**Outputs:**

1. **Depth Maps:**
   - Predicted depth maps based on the input images.
2. **Uncertainty Maps:**
   - Corresponding uncertainty maps indicating the confidence of the depth estimates.
3. **Point Cloud Representation:**
   - A reconstructed point cloud based on the brightest points, their depth, and normals.
4. **Visualizations:**
   - 3D visualizations of the reconstructed points and their associated normals, which can be plotted using different libraries (e.g., Matplotlib, Plotly, Mayavi).

**Key Functions:**

1. **resize_image_and_points:**
   - Resizes images and corresponding point coordinates to specified dimensions.
2. **get_interpolated_points_and_intensities:**
   - Interpolates depth values at the brightest points for better depth estimation.
3. **plot_vector_field_manual:**
   - Plots the 3D vector field of normals and brightest points, allowing for visual interpretation of spatial features.
4. **U-Net Function for Depth and Uncertainty Estimation:**
   - The U-Net architecture employs an encoder-decoder structure to predict depth and uncertainty maps from images, leveraging skip connections to maintain spatial accuracy while incorporating contextual information.