Python Code Documentation

Apolline Boyer

August 1, 2024

Contents

1 Classes and Functions

1.1 Class `StereoCalibration`

- **Description**: This class manages the parameters and processes of stereo calibration for two cameras.
- **Attributes**:
  - `cam_mats`: Dictionary of camera matrices (intrinsic parameters) for the left and right cameras.
  - `dist_coefs`: Dictionary of distortion coefficients for the left and right cameras.
  - `rot_mat`: Rotation matrix.
  - `trans_vec`: Translation vector.
  - `e_mat`: Essential matrix.
  - `f_mat`: Fundamental matrix.
  - `rect_trans`: Dictionary of rectification transformations for the left and right cameras.
  - `proj_mats`: Dictionary of projection matrices for the left and right cameras.
  - `disp_to_depth_mat`: Disparity-to-depth conversion matrix.
  - `valid_boxes`: Dictionary of valid pixel bounding boxes for the left and right cameras.
  - `undistortion_map`: Dictionary of undistortion maps for the left and right cameras.
  - `rectification_map`: Dictionary of rectification maps for the left and right cameras.

1.1.1 Method `__str__`

- **Description**: Returns a string representation of the class attributes.
- **Arguments**: None.
- **Returns**: **str** - String representation of the object's attributes.
- **Usage Example**:

```
print(calibration)
```

1.1.2 Method `__init__`

- **Description**: Initializes the calibration parameters for stereo cameras.
- **Arguments**: None.
- **Returns**: None.

- **Usage Example**:

```
calibration = StereoCalibration()
```

1.1.3 Method `save_data`

- **Description**: Saves the calibration parameters to .npy and .csv files.
- **Arguments**: None.
- **Returns**: None.
- **Usage Example**:

```
calibration.save_data()
```

1.1.4 Method `rectify`

- **Description**: Rectifies stereo images using undistortion and rectification maps.
- **Arguments**:
  - `frames`: **list** - List of left and right camera images.
- **Returns**: **list** - List of rectified images.
- **Usage Example**:

```
rectified_frames = calibration.rectify(frames)
```

1.1.5 Method `load_data`

- **Description**: Loads calibration parameters from .npy files in a specified directory.
- **Arguments**:
  - `directory`: **str** - Directory containing the parameter files.
- **Returns**: None.
- **Usage Example**:

```
calibration.load_data('calibration_data')
```

1.2 Class `Calibrator`

- **Description**: Manages the calibration process for a pair of stereo cameras.
- **Attributes**:
  - `image_count`: Number of calibration images used.
  - `row`: Number of internal corners in the rows of the calibration pattern.
  - `column`: Number of internal corners in the columns of the calibration pattern.
  - `square_size`: Size of the calibration pattern squares in cm.
  - `image_size`: Size of calibration images in pixels.
  - `corner_coordinates`: 3D coordinates of the calibration pattern corners.
  - `object_points`: List of coordinates of real corners found in each image.
  - `image_points`: Dictionary of coordinates of corners found in the images for the left and right cameras.

1.2.1 Method `__init__`

- **Description**: Initializes the parameters for camera calibration.
- **Arguments**:
  - `row`: **int** - Number of internal corners in the rows of the pattern.
  - `column`: **int** - Number of internal corners in the columns of the pattern.
  - `square_size`: **float** - Size of the pattern squares in cm.
  - `image_size`: **tuple** - Size of calibration images in pixels.
- **Returns**: None.
- **Usage Example**:

```
calibrator = Calibrator(row=6, column=9, square_size=2.5,
    image_size=(1920, 1080))
```

1.2.2 Method `corner_detect`

- **Description**: Detects the corners of the calibration pattern in a pair of images.
- **Arguments**:
  - `image_pair`: **tuple** - Tuple containing the left and right images.
- **Returns**: None.
- **Usage Example**:

```
calibrator.corner_detect((left_image, right_image))
```

1.2.3 Method `calibrate_camera`

- **Description**: Calibrates both cameras and determines their related matrices.
- **Arguments**: None.
- **Returns**: **StereoCalibration** - Instance of the `StereoCalibration` class with calibrated parameters.
- **Usage Example**:

```
calibration = calibrator.calibrate_camera()
```

1.2.4 Method `calibration_process`

- **Description**: Performs the calibration process by reading the calibration images and calling the calibration method.
- **Arguments**:
  - `nbr_photo`: **int** - Number of calibration images to process.
  - `image_folder`: **str** - Directory containing the calibration images.
- **Returns**: **StereoCalibration** - Instance of the `StereoCalibration` class with the calibrated parameters.
- **Usage example**:

```
calibration = calibrator.calibration_process(nbr_photo=20,
    image_folder='calibration_images')
```

1.3 Class `DualCameraCapture`

- **Description**: Captures stereo images using two cameras and provides tools to display and validate these images.
- **Attributes**:
  - `left_cam_id`: ID of the left camera.
  - `right_cam_id`: ID of the right camera.
  - `preview_size`: Size of the preview of the captured images.
  - `preview_type`: Type of preview.
  - `capture_delay`: Delay before capturing the image.
  - `interval`: Interval between image captures.

1.3.1 Method `__init__`

- **Description**: Initializes the parameters for capturing images with two cameras.
- **Arguments**:
  - `left_cam_id`: **int** - ID of the left camera.
  - `right_cam_id`: **int** - ID of the right camera.
  - `preview_size`: **tuple** - Size of the preview.
  - `preview_type`: **Preview** - Type of preview.
  - `capture_delay`: **float** - Delay before capturing the image.
  - `interval`: **float** - Interval between image captures.
- **Returns**: None.
- **Usage example**:

```
capture = DualCameraCapture(left_cam_id=0, right_cam_id=1,
    preview_size=(800, 600), capture_delay=2, interval=5)
```

1.3.2 Method `capture_and_save_image`

- **Description**: Captures and saves an image from the specified camera.
- **Arguments**:
  - `picam_id`: **int** - ID of the camera to use.
  - `filename`: **str** - Filename to save the image.
- **Returns**: None.
- **Usage example**:

```
capture.capture_and_save_image(picam_id=0, filename='left_image.png
    ')
```

1.3.3 Method `display_images`

- **Description**: Displays the captured images from the specified files.
- **Arguments**:
  - `left_filename`: **str** - Filename of the left image.
  - `right_filename`: **str** - Filename of the right image.
- **Returns**: None.
- **Usage example**:

```
capture.display_images(left_filename='left_image.png',
    right_filename='right_image.png')
```

1.3.4 Method `validate_images`

- **Description**: Validates if the captured images are acceptable.
- **Arguments**: None.
- **Returns**: **bool** - Returns True if the images are acceptable, otherwise False.
- **Usage example**:

```
is_valid = capture.validate_images()
```

1.3.5 Method `capture_images`

- **Description**: Captures a specified number of image pairs and saves them in the specified folder.
- **Arguments**:
  - `nbr_photos`: **int** - Number of image pairs to capture.
  - `image_folder`: **str** - Folder to save the images.
- **Returns**: None.
- **Usage example**:

```
capture.capture_images(nbr_photos=10, image_folder='captured_images')
```

1.4 Class `StereoVision`

- **Description**: This class manages stereo image capture, disparity and depth map calculation, as well as result processing and display.
- **Attributes**:
  - `cam_capture`: Instance of `DualCameraCapture` used for capturing images.
  - `baseline`: Distance between the cameras (in meters).
  - `focal`: Focal length of the camera.
  - `block_size`: Block size for stereo matching.
  - `P1`: Weight for cost regularization.
  - `P2`: Weight for cost regularization.
  - `min_disp`: Minimum disparity to consider.
  - `max_disp`: Maximum disparity to consider.
  - `uniqueRatio`: Uniqueness ratio for stereo matching.
  - `speckleWindowSize`: Window size for speckle filtering.
  - `speckleRange`: Range of values for speckle filtering.

- `disp12MaxDiff`: Maximum difference between left and right disparities.
- `stop_event`: Event to stop processes.
- `n`: Counter for the number of saved images.
- `images`: Dictionary to store captured and rectified images.
- `disparity`: Calculated disparity map.
- `disparity_normalized`: Normalized disparity map for display.
- `depth`: Calculated depth map.

1.4.1 Method `__init__`

- **Description**: Initializes the parameters for stereo vision.
- **Arguments**:
  - `cam_capture`: **DualCameraCapture** - Instance of the class for capturing images.
  - `baseline`: **float** - Distance between the cameras.
  - `focal`: **float** - Focal length of the camera.
  - `block_size`: **int** - Block size for stereo matching.
  - `P1`: **int** - Weight for cost regularization.
  - `P2`: **int** - Weight for cost regularization.
  - `min_disp`: **int** - Minimum disparity to consider.
  - `max_disp`: **int** - Maximum disparity to consider.
  - `uniqueRatio`: **int** - Uniqueness ratio for stereo matching.
  - `speckleWindowSize`: **int** - Window size for speckle filtering.
  - `speckleRange`: **int** - Range of values for speckle filtering.
  - `disp12MaxDiff`: **int** - Maximum difference between left and right disparities.
- **Returns**: None.
- **Usage example**:

```
stereo_vision = StereoVision(
    cam_capture=DualCameraCapture(left_cam_id=0, right_cam_id=1,
       preview_size=(800, 600)),
    baseline=0.06,
    focal=1300,
    block_size=15,
    P1=150,
    P2=64,
    min_disp=-16,
    max_disp=128,
    uniqueRatio=4,
    speckleWindowSize=200,
    speckleRange=4,
    disp12MaxDiff=0
)
```

1.4.2 Method `stereo_taking`

- **Description**: Captures and rectifies stereo images.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
stereo_vision.stereo_taking()
```

1.4.3 Method `save_images`

- **Description**: Saves the images and the normalized disparity map.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
stereo_vision.save_images()
```

1.4.4 Method `depth_map_calcul`

- **Description**: Calculates the disparity map from the rectified images.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
stereo_vision.depth_map_calcul()
```

1.4.5 Method `depth_calcul`

- **Description**: Calculates the depth for each pixel from the disparity map.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
stereo_vision.depth_calcul()
```

1.4.6 Method `process_stereo`

- **Description**: Processes the depth map using `DepthMapProcessor`.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
stereo_vision.process_stereo()
```

1.4.7 Method `capture_and_compute`

- **Description**: Captures images, calculates the disparity map and depth, and then puts the results in a queue.
- **Arguments**:
  - `queue`: **Queue** - Queue for passing results between processes.
- **Returns**: None.
- **Usage example**:

```
queue = Queue()
stereo_vision.capture_and_compute(queue)
```

1.4.8 Method `depth_map_display`

- **Description**: Displays the disparity map and depth from the results in the queue.
- **Arguments**:
  - `queue`: **Queue** - Queue for obtaining calculated results.
- **Returns**: None.
- **Usage example**:

```
queue = Queue()
stereo_vision.depth_map_display(queue)
```

1.4.9 Method `process_and_display`

- **Description**: Creates processes for capturing and computing images, as well as displaying the results.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
stereo_vision.process_and_display()
```

1.5 Class `TofCamera`

- **Description**: This class handles capturing images from a ToF camera, processing depth and amplitude data, and displaying the results.
- **Attributes**:
  - `cam`: Instance of `ArducamCamera` for the ToF camera.
  - `max_distance`: Maximum measurable distance by the camera (in meters).
  - `frame`: Current frame captured by the camera.
  - `amplitude_buf`: Buffer for amplitude data.
  - `depth_buf`: Buffer for depth data.
  - `depth_normalized`: Normalized depth map for display.
  - `result_image`: Resulting image after processing.
  - `n`: Counter for saved image names.

1.5.1 Method `__init__`

- **Description**: Initializes the ToF camera with the maximum distance parameters.
- **Arguments**:
  - `max_distance`: **float** - Maximum measurable distance by the camera (in meters).
- **Returns**: None.
- **Usage example**:

```
tof_camera = TofCamera(max_distance=4)
```

1.5.2 Method `process_frame`

- **Description**: Processes the captured frame to produce a resulting image by combining depth and amplitude data.
- **Arguments**: None.
- **Returns**: **np.ndarray** - Resulting image after processing.
- **Usage example**:

```
result_image = tof_camera.process_frame()
```

1.5.3 Method `capture_image`

- **Description**: Saves the resulting image under the name `tof{n}.png`.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
tof_camera.capture_image()
```

1.5.4 Method `process_tof`

- **Description**: Processes the depth map using `DepthMapProcessor` to analyze and extract contours.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
tof_camera.process_tof()
```

1.5.5 Method `continuous_display`

- **Description**: Captures and displays images continuously from the ToF camera, with options to save and process images.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
tof_camera.continuous_display()
```

1.5.6 Method `cleanup`

- **Description**: Stops and closes the camera, and destroys all OpenCV windows.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
tof_camera.cleanup()
```

1.5.7 Method `get_depth_buf`

- **Description**: Returns the current depth buffer.
- **Arguments**: None.
- **Returns**: **np.ndarray** - Depth buffer.
- **Usage example**:

```
depth_buf = tof_camera.get_depth_buf()
```

1.5.8 Method `get_depth_normalized`

- **Description**: Returns the normalized depth map.
- **Arguments**: None.
- **Returns**: **np.ndarray** - Normalized depth map.
- **Usage example**:

```
depth_normalized = tof_camera.get_depth_normalized()
```

1.6 Class `DepthMapProcessor`

- **Description**: This class handles processing of depth and disparity maps, including segmentation, calculating average amplitudes, and drawing contours.
- **Attributes**:
  - `depth_map_original`: Original depth map.
  - `depth_map_normalized`: Normalized disparity map.
  - `pixel_min`: Minimum number of non-zero pixels required to consider a segment (default 15000).
  - `min_contour_area`: Minimum area for contours to be considered (default 10).
  - `thresholds`: List of thresholds for disparity segmentation.
  - `kernel_size`: Kernel size for morphological operations (default 5).
  - `dilate_iterations`: Number of iterations for dilation (default 1).
  - `erode_iterations`: Number of iterations for erosion (default 2).
  - `segmented_image`: Segmented image after applying thresholds.
  - `contours`: List of found contours.
  - `mean_amplitudes`: Dictionary of average amplitudes for each contour.

1.6.1 Method `__init__`

- **Description**: Initializes the `DepthMapProcessor` class with the provided parameters.
- **Arguments**:
  - `depth_map`: **np.ndarray** - Original depth map.
  - `disparity`: **np.ndarray** - Normalized disparity map.
  - `pixel_min`: **int** - Minimum number of non-zero pixels required to consider a segment (default 15000).
  - `min_contour_area`: **int** - Minimum area for contours to be considered (default 10).
  - `thresholds`: **list** - List of thresholds for disparity segmentation.
  - `kernel_size`: **int** - Kernel size for morphological operations (default 5).
  - `dilate_iterations`: **int** - Number of iterations for dilation (default 1).
  - `erode_iterations`: **int** - Number of iterations for erosion (default 2).
- **Returns**: None.
- **Usage example**:

```
depth_map = np.zeros((480, 640))  # Example depth map
disparity = np.zeros((480, 640))  # Example disparity map
processor = DepthMapProcessor(depth_map, disparity)
```

1.6.2 Method `apply_morphological_operations`

- **Description**: Applies morphological operations (dilation and erosion) to the specified image.
- **Arguments**:
  - `image`: **np.ndarray** - Image to process.
- **Returns**: **np.ndarray** - Image after applying morphological operations.
- **Usage example**:

```
processed_image = processor.apply_morphological_operations(image)
```

1.6.3 Method `calculate_mean_amplitude`

- **Description**: Calculates the average amplitude for each specified contour.
- **Arguments**:
  - `contours`: **list** - List of contours found in the image.

- **Returns**: **dict** - Dictionary of average amplitudes for each contour.
- **Usage example**:

```
mean_amplitudes = processor.calculate_mean_amplitude(contours)
```

1.6.4 Method `find_and_draw_contours`

- **Description**: Finds and draws contours in the processed image.
- **Arguments**:
  - `processed_image`: **np.ndarray** - Image after morphological operations.
- **Returns**: **np.ndarray** - Image with drawn contours.
- **Usage example**:

```
image_with_contours = processor.find_and_draw_contours(
    processed_image)
```

1.6.5 Method `process_contour`

- **Description**: Processes contours by applying morphological operations, finding and drawing contours, and calculating average amplitudes for the found contours.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
processor.process_contour()
```

1.6.6 Method `process_disparity_image`

- **Description**: Processes the disparity image by segmenting it according to the defined thresholds, and then applying contour processing on each segment.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
processor.process_disparity_image()
```

1.7 Utility Functions

1.7.1 Function `calculate_histogram`

- **Description**: Calculates the histogram of pixel values in the image.
- **Arguments**:
  - `image`: **np.ndarray** - Image to analyze.
- **Returns**: **np.ndarray** - Histogram of pixel values.
- **Usage example**:

```
hist = calculate_histogram(image)
```

1.7.2 Function `count_non_zero_pixels_from_histogram`

- **Description**: Counts the number of non-zero pixels from the pixel value histogram.
- **Arguments**:
  - `hist`: **np.ndarray** - Histogram of pixel values.
- **Returns**: **int** - Total number of non-zero pixels.
- **Usage example**:

```
non_zero_count = count_non_zero_pixels_from_histogram(hist)
```

1.7.3 Function `plot_histogram`

- **Description**: Plots and displays the histogram of pixel values.
- **Arguments**:
  - `title`: **str** - Title of the plot.
  - `hist`: **np.ndarray** - Histogram of pixel values.
- **Returns**: None.
- **Usage example**:

```
plot_histogram("Pixel Histogram", hist)
```

2 Functions

2.1 Function `folder_create`

- **Description**: Checks if a folder exists; if not, it creates one.
- **Arguments**:
  - `folder`: **str** - Path of the folder to check/create.
- **Returns**: None.
- **Usage example**:

```
folder_create('path/to/folder')
```

2.2 Function `file_create`

- **Description**: Creates a file of the specified type in a given folder (optional).
- **Arguments**:
  - `data`: **np.ndarray**, **list**, or other - Data to save in the file.
  - `file_name`: **str** - Name of the file to create (without extension).
  - `file_type`: **str** - Type of file to create ('csv', 'image', 'npy', etc.).
  - `folder_name`: **str** (optional) - Folder in which to create the file.
- **Returns**: None.
- **Usage example**:

```
file_create(data, 'example_image', 'png', 'path/to/folder')
file_create(data, 'example_array', 'npy')
file_create(data, [['header1', 'header2'], [1, 2]], 'csv', 'path/to
    /folder')
```

2.3 Function `show_image`

- **Description**: Displays an image with a specified colormap.
- **Arguments**:
  - `title`: **str** - Title of the display window.
  - `image`: **np.ndarray** - Image to display.
  - `cmap`: **str** (optional) - OpenCV colormap to apply ('gray', 'jet', 'rainbow', etc.).
- **Returns**: None.
- **Usage example**:

```
show_image('Displayed Image', image, 'jet')
```

2.3.1 List of Colormaps

- **Available Colormaps**:
  - `autumn`
  - `bone`
  - `jet`
  - `winter`
  - `rainbow`
  - `ocean`
  - `summer`
  - `spring`
  - `cool`
  - `hsv`
  - `pink`
  - `hot`
  - `parula`
  - `magma`
  - `inferno`
  - `plasma`
  - `viridis`
  - `cividis`
  - `twilight`
  - `twilight_shifted`
  - `turbo`
  - `deepgreen`

3 Camera and Process Management

3.1 Function `calibrate_cameras`

- **Description**: Calibrates the cameras by taking pictures of a checkerboard and using a calibration process.
- **Arguments**:
  - `cam_capture`: Instance of `DualCameraCapture` for capturing images.
- **Returns**: None.
- **Usage example**:

```
calibrate_cameras(cam_capture)
```

3.2 Function `run_tof_camera`

- **Description**: Function to run the ToF camera continuously and update a queue with depth data.
- **Arguments**:
  - `camera_queue`: Queue to store camera data.
- **Returns**: None.
- **Usage example**:

```
run_tof_camera(camera_queue)
```

3.3 Function `run_stereo_vision`

- **Description**: Function to run stereo vision and return the disparity and depth results.
- **Arguments**: None.
- **Returns**: Tuple containing normalized disparity and depth.
- **Usage example**:

```
disparity_normalized, depth = run_stereo_vision()
```

3.4 Function `terminate_processes`

- **Description**: Terminates the given processes.
- **Arguments**:
  - `processes`: List of processes to terminate.
- **Returns**: None.
- **Usage example**:

```
terminate_processes(processes)
```

3.5 Function `kill_zombie_processes`

- **Description**: Terminates the zombie processes detected on the system.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
kill_zombie_processes()
```

3.6 Function `clean_temp_dirs`

- **Description**: Cleans the specified temporary directories.
- **Arguments**:
  - `directories`: List of directories to clean.
- **Returns**: None.
- **Usage example**:

```
clean_temp_dirs(['/tmp', '/var/tmp'])
```

3.7 Function `cleanup`

- **Description**: Performs system cleanup and frees memory.
- **Arguments**: None.
- **Returns**: None.
- **Usage example**:

```
cleanup()
```

4 Main Script Execution

```
if __name__ == "__main__":
    cleanup()
    folder_create('data')
    folder_create('image')
    folder_create('corner')

    calib_choice = input("Do you want to calibrate the cameras (y/n)? ")
        .strip().lower()

    if calib_choice == "y":
        cam_capture = DualCameraCapture(left_cam_id=2, right_cam_id=1,
            preview_size=(840, 820))
        calibrate_cameras(cam_capture)
    elif calib_choice == "n":
        print("Cameras will not be calibrated.")
    else:
        print("Invalid choice. Please enter 'y' or 'n'.")
        exit(1)

    # Initialize queues for inter-process communication
    camera_queue = multiprocessing.Queue()

    # Create processes
    tof_process = multiprocessing.Process(target=run_tof_camera, args=(
        camera_queue,))
```

```
    stereo_process = multiprocessing.Process(target=run_stereo_vision)

    # Start processes
    tof_process.start()
    stereo_process.start()

    processes = [tof_process, stereo_process]

    # Wait for processes to finish
    try:
        # Keep processes alive
        while True:
            sleep(1)
    except KeyboardInterrupt:
        print("Interruption detected. Stopping processes...")
    finally:
        # Stop processes
        terminate_processes(processes)
        print("All processes have been stopped.")
        cleanup()
        sys.exit(0)
```