Python Code Documentation

Introduction

This code is used to perform object detection and tracking using the ZED stereo camera and the ZED SDK.
The code opens a ZED camera, sets the configuration parameters, and enables positional tracking and object detection modules. It then enters a loop where it grabs images from the camera, retrieves objects using object detection, and updates the 3D view and tracking view if the GUI is enabled. The loop continues until the program is closed or the end of the SVO file is reached (if playback mode is enabled).

Usage

To run the code, use the following command:

python3 code.py --input_svo_file [path_to_svo_file] --ip_address [IP_address] --resolution [resolution] --disable_gui --enable_batching_reid

- --input_svo_file: Path to an .svo file (optional).
- --ip_address: IP Address for live streaming (optional).
- --resolution: Camera resolution. Can be HD2K, HD1200, HD1080, HD720, SVGA, or VGA (optional).
- --disable_gui: Flag to disable the GUI and increase detection performances (optional).
- --enable_batching_reid: Flag to enable batching re identification and display difference on ID tracking in the console (optional).

Code Structure

The code is organized as follows:

1. Import necessary libraries and modules.
2. Check if the code is running on Jetson (Nvidia Jetson platform).
3. Define the main function.
4. Call the parse_args function to parse command line arguments.
5. Open the ZED camera and set configuration parameters.
6. Enable positional tracking and object detection modules.

7. Setup the GUI if enabled.
8. Define a loop to grab images, retrieve objects, and update the views.
9. Handle key presses to control the GUI.
10. Clean up and exit.

Functions

main

The main function is the entry point of the code. It opens the ZED camera, sets the configuration parameters, enables positional tracking and object detection modules, and enters a loop to grab images, retrieve objects, and update the views. It also handles key presses to control the GUI and exits when the program is closed or the end of the SVO file is reached.

parse_args

The parse_args function is used to parse the command line arguments. It takes an `init` parameter which is an initialized InitParameters object. It checks if the `input_svo_file` or `ip_address` arguments are provided and sets the corresponding values in the `init` object. It also sets the camera resolution based on the `resolution` argument.

printHelp

The printHelp function is used to print a help message to the console. It lists the hotkeys for controlling the GUI.

Variables

- `is_jetson`: A boolean variable to indicate if the code is running on Jetson.
- `zed`: An instance of the sl.Camera() class to interface with the ZED camera.
- `init_params`: An instance of the sl.InitParameters() class to store the camera configuration parameters.
- `is_playback`: A boolean variable to indicate if an SVO is used.
- `status`: A variable to store the status of the camera open operation.
- `positional_tracking_parameters`: An instance of the sl.PositionalTrackingParameters() class to store the positional tracking parameters.

- batch_parameters: An instance of the sl.BatchParameters() class to store the batch parameters for object detection.
- obj_param: An instance of the sl.ObjectDetectionParameters() class to store the object detection parameters.
- detection_confidence: The default detection threshold for object detection.
- detection_parameters_rt: An instance of the sl.ObjectDetectionRuntimeParameters() class to store the runtime parameters for object detection.
- quit_bool: A boolean variable to indicate if the program should quit.
- image_aspect_ratio: The aspect ratio of the camera image.
- requested_low_res_w: The requested width of the low resolution image for display.
- display_resolution: An instance of the sl.Resolution() class to store the resolution of the display.
- image_scale: A list to store the scaling factors for the camera image and the tracking view.
- image_left_ocv: An OpenCV numpy array to store the left camera image with a background color.
- camera_config: An instance of the sl.CameraConfiguration() class to store the camera configuration.
- tracks_resolution: An instance of the sl.Resolution() class to store the resolution of the tracking view.
- track_view_generator: An instance of the cv_viewer.TrackingViewer() class to generate the tracking view.
- image_track_ocv: An OpenCV numpy array to store the tracking view.
- global_image: An OpenCV numpy array to store the combined camera image and tracking view.
- viewer: An instance of the gl.GLViewer() class to display the 3D view.
- pc_resolution: An instance of the sl.Resolution() class to store the resolution of the point cloud.
- point_cloud: An instance of the sl.Mat() class to store the point cloud data.
- objects: An instance of the sl.Objects() class to store the detected objects.

- image_left: An instance of the sl.Mat() class to store the left camera image.
- cam_w_pose: An instance of the sl.Pose() class to store the camera world pose.
- viewer_available: A boolean variable to indicate if the gl_viewer is available.
- id_counter: A dictionary to store the ID counters for objects.
- runtime_parameters: An instance of the sl.RuntimeParameters() class to store the runtime parameters for grabbing images.
- window_name: The window name for the GUI.
- gl_viewer_available: A boolean variable to indicate if the gl_viewer is available.

Imports:

- cv2 module from OpenCV library is imported for computer vision operations.
- numpy module is imported to use arrays and numerical operations.
- math module is imported for mathematical operations.
- deque class from collections module is imported to create a queue data structure.
- pyzed.sl module is imported for using the ZED camera.

TrackPointState class:

- Constants representing the state of a track point:
- OK: 0
- PREDICTED: 1
- OFF: 2

OBJECT_CLASS class:

- Constants representing different classes of objects:
- PERSON: 0
- VEHICLE: 1

- LAST: 2

TrackPoint class:

- Represents a point in a track.
- Attributes:
- x, y, z: The coordinates of the point.
- tracking_state: The state of tracking for this point.
- timestamp: The timestamp at which the point was tracked.
- Methods:
- get_xyz(): Returns the coordinates of the point as an array multiplied by 1000.

Tracklet class:

- Represents a track of an object.
- Attributes:
- id: The ID of the tracklet.
- label: The label of the tracklet.
- object_type: The type of object being tracked.
- is_alive: A flag indicating if the tracklet is still active.
- last_detected_timestamp: The timestamp at which the object was last detected.
- recovery_cpt: Counter for recovery.
- recovery_length: Length of the recovery period.
- positions: A list of TrackPoint objects representing the positions of the object over time.
- positions_to_draw: A list of TrackPoint objects to be drawn.
- tracking_state: The state of tracking for the object.
- Methods:

- addDetectedPoint(obj, timestamp, smoothing_window_size): Adds a detected point to the tracklet.

Class TrackingViewer

init(self, res, fps_, D_max, duration)

- Purpose: initializes a TrackingViewer object with the given parameters
- Parameters:
  -
  - res: the resolution of the window (type: object)
  - fps_: the frames per second (type: int)
  - D_max: the maximum distance (type: float)
  - duration: the duration (type: int)

generate_view(self, objects, image_left_ocv, img_scale, current_camera_pose, tracking_view, tracking_enabled)

- Purpose: generates the view of the tracking objects
- Parameters:
  -
  - objects: the list of objects to be tracked (type: object)
  - image_left_ocv: the left image (type: object)
  - img_scale: the scale of the image (type: tuple)
  - current_camera_pose: the pose of the camera (type: object)
  - tracking_view: the view of the tracking (type: object)
  - tracking_enabled: a boolean indicating if tracking is enabled (type: bool)

render_2D(self, left_display, img_scale, objects, render_mask, isTrackingON)

- Purpose: renders the 2D projection of the objects
- Parameters:
  -
  - left_display: the left image (type: object)
  - img_scale: the scale of the image (type: tuple)
  - objects: the list of objects to be rendered (type: list)
  - render_mask: a boolean indicating if the mask should be rendered (type: bool)
  - isTrackingON: a boolean indicating if tracking is enabled (type: bool)

zoomIn(self)

- Purpose: zooms in the view

zoomOut(self)

- Purpose: zooms out the view

addToTracklets(self, objects)

- Purpose: adds the detected objects to the tracklets
- Parameters:
  -
  - objects: the list of detected objects (type: object)

detectUnchangedTrack(self, current_timestamp)

- Purpose: detects the unchanged tracks and marks them as inactive
- Parameters:
  -
  - current_timestamp: the current timestamp (type: int)

pruneOldPoints(self, ts)

- Purpose: prunes the old track points
- Parameters:
  -
  - ts: the timestamp (type: int)

set_camera_calibration(self, calib)

- Purpose: sets the camera calibration for the viewer
- Parameters:
  -
  - calib: the camera calibration data (type: object)

computeFOV(self)

- Purpose: computes the field of view (FOV) of the camera

zoom(self, factor)

- Purpose: zooms the view by the given factor
- Parameters:
  - factor: the zoom factor (type: float)

drawScale(self, tracking_view)

- Purpose: draws the scale on the tracking view
- Parameters:
  - tracking_view: the tracking view (type: object)

drawTracklets(self, tracking_view, current_camera_pose)

- Purpose: draws the tracklets on the tracking view
- Parameters:
  - tracking_view: the tracking view (type: object)
  - current_camera_pose: the current pose of the camera (type: object)

generateBackground(self)

- Purpose: generates the background image for the tracking view

drawCamera(self)

- Purpose: draws the camera on the background image

drawHotkeys(self)

- Purpose: draws the hotkeys on the background image

to_cv_point(self, x, z)

- Purpose: converts the 3D point (x, z) to the corresponding point on the image plane
- Parameters:
  - x: the x-coordinate of the 3D point (type: float or object)
  - z: the z-coordinate of the 3D point (type: float or object)

- Returns: the corresponding point on the image plane (type: list or np.ndarray)

renderObject(i, isTrackingON)

- Purpose: renders the given object based on the tracking state and tracking mode
- Parameters:
  -
  - i: the object to be rendered (type: object)
  - isTrackingON: a boolean indicating if tracking is enabled (type: bool)
- Returns: a boolean indicating if the object should be rendered (type: bool)

generateColorID_u(idx)

- Purpose: generates a unique color based on the given index
- Parameters:
  -
  - idx: the index (type: int)
- Returns: the unique color (type: tuple)

cvt(pt, scale)

- Purpose: converts the point from the image scale to the original scale
- Parameters:
  -
  - pt: the point to be converted (type: object)
  - scale: the scale of the image (type: tuple)
- Returns: the converted point (type: np.ndarray)

drawVerticalLine(display, start_pt, end_pt, clr, thickness)

- Purpose: draws a vertical line on the display
- Parameters:
  -
  - display: the display image (type: object)
  - start_pt: the starting point of the line (type: tuple)
  - end_pt: the ending point of the line (type: tuple)

    - clr: the color of the line (type: tuple)
    - thickness: the thickness of the line (type: int)

getImagePosition(bounding_box_image, img_scale)

- Purpose: gets the image position of the object

- Parameters:
    - bounding_box_image: the bounding box of the object in the image (type: list)
    - img_scale: the scale of the image (type: tuple)

- Returns: the image position of the object (type: np.ndarray)

LineIterator

Produces an array that consists of the coordinates and intensities of each pixel in a line between two points.

Parameters

- img: the image being processed

- P1: a numpy array that consists of the coordinate of the first point (x,y)

- P2: a numpy array that consists of the coordinate of the second point (x,y)

Returns

- it: a numpy array that consists of the coordinates and intensities of each pixel in the radii (shape: [numPixels, 3], row = [x,y,intensity])

Simple3DObject

__init__(self, _is_static, pts_size = 3, clr_size = 3)

This function initializes a Simple3DObject object.

Parameters

- _is_static (bool): Indicates whether the object is static or dynamic.

- pts_size (int): The size of the points array (default is 3).

- clr_size (int): The size of the colors array (default is 3).

add_pt(self, _pts)

This function adds points to the object's vertices array.

Parameters

- _pts (list): A list of points to be added to the vertices array.

add_clr(self, _clrs)

This function adds colors to the object's colors array.

Parameters

- _clrs (list): A list of colors to be added to the colors array.

add_point_clr(self, _pt, _clr)

This function adds a point and its color to the object's vertices and colors arrays.

Parameters

- _pt (list): The point to be added to the vertices array.
- _clr (list): The color to be added to the colors array.

add_line(self, _p1, _p2, _clr)

This function adds a line segment to the object's vertices and colors arrays.

Parameters

- _p1 (list): The first point of the line segment.
- _p2 (list): The second point of the line segment.
- _clr (list): The color of the line segment.

addFace(self, p1, p2, p3, clr)

This function adds a triangle face to the object's vertices and colors arrays.

Parameters

- p1 (list): The first point of the face.
- p2 (list): The second point of the face.
- p3 (list): The third point of the face.
- clr (list): The color of the face.

add_full_edges(self, _pts, _clr)

This function adds the full set of edges to the object's vertices and colors arrays.

Parameters

- _pts (list): A list of points representing the edges.
- _clr (list): The color of the edges.

__add_single_vertical_line(self, _top_pt, _bottom_pt, _clr)

This function adds a single vertical line to the object's vertices and colors arrays.

Parameters

- _top_pt (list): The top point of the line.
- _bottom_pt (list): The bottom point of the line.
- _clr (list): The color of the line.

add_vertical_edges(self, _pts, _clr)

This function adds vertical edges to the object's vertices and colors arrays.

Parameters

- _pts (list): A list of points representing the edges.
- _clr (list): The color of the edges.

add_top_face(self, _pts, _clr)

This function adds the top face to the object's vertices and colors arrays.

Parameters

- _pts (list): A list of points representing the face.
- _clr (list): The color of the face.

__add_quad(self, _quad_pts, _alpha1, _alpha2, _clr)

This function adds a quad to the object's vertices and colors arrays.

Parameters

- _quad_pts (list): A list of points representing the quad.
- _alpha1 (float): The first alpha parameter for the quad color.
- _alpha2 (float): The second alpha parameter for the quad color.
- _clr (list): The color of the quad.

add_vertical_faces(self, _pts, _clr)

This function adds vertical faces to the object's vertices and colors arrays.

Parameters

- _pts (list): A list of points representing the faces.
- _clr (list): The color of the faces.

push_to_GPU(self)

This function pushes the object's data to the GPU.

init(self, res)

This function initializes the object with the specified resolution.

Parameters

- res (Resolution): The resolution of the object.

setPoints(self, pc)

This function sets the points data for the object.

Parameters

- pc (PointCloud): The point cloud data.

clear(self)

This function clears the object's vertices, colors, and indices arrays.

set_drawing_type(self, _type)

This function sets the drawing type for the object.

Parameters

- _type (GLint): The drawing type.

draw(self)

This function draws the object.

Class GLViewer

The GLViewer class provides a graphical user interface for viewing objects in a 3D space. It uses OpenGL to render the objects and allows for interaction with the mouse and keyboard.

Methods

- __init__(self): Initializes the GLViewer object and sets initial values for its attributes.
- init(self, camera_model, res, is_tracking_on): Initializes the GLViewer with the specified camera model, resolution, and tracking mode.
- is_available(self): Checks if the GLViewer is available for use.
- render_object(self, _object_data): Determines if an object should be rendered based on its tracking state and the current tracking mode.
- updateData(self, pc, _objs): Updates the GLViewer data with new point cloud and object data.
- create_bbox_rendering(self, _bbox, _bbox_clr): Creates the rendering for a bounding box.
- idle(self): Callback function for idle state.
- exit(self): Exits the GLViewer.
- close_func(self): Callback function for closing the window.
- keyPressedCallback(self, key, x, y): Callback function for keyboard key press events.
- on_mouse(self,*args,**kwargs): Callback function for mouse button events.
- on_mousemove(self,*args,**kwargs): Callback function for mouse motion events.
- on_resize(self,Width,Height): Callback function for window resize events.
- draw_callback(self): Callback function for drawing the scene.
- update(self): Updates the GLViewer based on mouse and keyboard input.
- draw(self): Draws the scene using OpenGL.

CameraGL

A class that represents a camera in 3D space for graphical applications.

Constructor

__init__(self)

- Initializes a new instance of the CameraGL class.
- Sets the default values for the camera's properties.

Properties

- ORIGINAL_FORWARD: A sl.Translation object representing the original forward direction of the camera.
- ORIGINAL_UP: A sl.Translation object representing the original up direction of the camera.
- ORIGINAL_RIGHT: A sl.Translation object representing the original right direction of the camera.
- znear: A float representing the near clipping plane of the camera.
- zfar: A float representing the far clipping plane of the camera.
- horizontalFOV: A float representing the horizontal field of view of the camera.
- orientation_: A sl.Orientation object representing the orientation of the camera.
- position_: A sl.Translation object representing the position of the camera.
- forward_: A sl.Translation object representing the forward direction of the camera.
- up_: A sl.Translation object representing the up direction of the camera.
- right_: A sl.Translation object representing the right direction of the camera.
- vertical_: A sl.Translation object representing the vertical direction of the camera.
- vpMatrix_: A sl.Matrix4f object representing the view projection matrix of the camera.
- offset_: A sl.Translation object representing the offset of the camera.
- projection_: A sl.Matrix4f object representing the projection matrix of the camera.

Methods

update(self)

- Updates the camera's view projection matrix based on its current properties.

setProjection(self, im_ratio)

- Sets the projection matrix of the camera based on the given image ratio.

Parameters

- im_ratio: A float representing the image ratio.

getViewProjectionMatrix(self)

- Gets the view projection matrix of the camera.

Returns

- A list of floats representing the elements of the view projection matrix.

getViewProjectionMatrixRT(self, tr)

- Gets the view projection matrix of the camera transformed by the given transformation matrix.

Parameters

- tr: A sl.Translation object representing the transformation.

Returns

- A list of floats representing the elements of the transformed view projection matrix.

setDirection(self, dir, vert)

- Sets the direction of the camera.

Parameters

- dir: A sl.Translation object representing the direction.
- vert: A sl.Translation object representing the vertical direction.

translate(self, t)

- Translates the camera.

Parameters

- t: A sl.Translation object representing the translation.

setPosition(self, p)

- Sets the position of the camera.

Parameters

- p: A sl.Translation object representing the position.

rotate(self, r)

- Rotates the camera.

Parameters

- r: A sl.Rotation object representing the rotation.

setRotation(self, r)

- Sets the rotation of the camera.

Parameters

- r: A sl.Rotation object representing the rotation.

updateVectors(self)

- Updates the camera's forward, up, and right vectors based on its current orientation.

CSharp Code Documentation

Class: Utils

The Utils class provides various utility methods for working with OpenCV and ZED Mat.

SLMat2CVMat Method

public static OpenCvSharp.Mat SLMat2CVMat(ref sl.Mat zedmat, MAT_TYPE zedmattype)

Creates an OpenCV version of a ZED Mat.

Parameters

- zedmat : ref sl.Mat - Source ZED Mat.
- zedmattype : MAT_TYPE - Type of ZED Mat - data type and channel number.

Returns

- OpenCvSharp.Mat - The converted OpenCV Mat.

SLMatType2CVMatType Method

private static int SLMatType2CVMatType(MAT_TYPE zedmattype)

Returns the OpenCV type that corresponds to a given ZED Mat type.

Parameters

- zedmattype : MAT_TYPE - The ZED Mat type.

Returns

- int - The corresponding OpenCV Mat type.

getMilliseconds Method

public static ulong getMilliseconds(ulong ts_ns)

Converts nanoseconds to milliseconds.

Parameters

- ts_ns : ulong - The timestamp in nanoseconds.

Returns

- ulong - The timestamp in milliseconds.

drawVerticalLine Method

public static void drawVerticalLine(ref OpenCvSharp.Mat left_display, Point start_pt, Point end_pt, Scalar clr, int thickness)

Draws a vertical line on an OpenCV Mat.

Parameters

- left_display : ref OpenCvSharp.Mat - The OpenCV Mat to draw on.
- start_pt : Point - The starting point of the line.
- end_pt : Point - The ending point of the line.
- clr : Scalar - The color of the line.
- thickness : int - The thickness of the line.

cvt Method

public static Point cvt(Vector2 pt, sl.float2 scale)

Converts a vector point from ZED camera coordinates to OpenCV coordinates.

Parameters

- pt : Vector2 - The point in ZED camera coordinates.
- scale : sl.float2 - The scale factor.

Returns

- Point - The point in OpenCV coordinates.

generateColorID Method

public static sl.float4 generateColorID(int idx)

Generates a color based on the given index.

Parameters

- idx : int - The index.

Returns

- sl.float4 - The generated color.

generateColorID_u Method

public static OpenCvSharp.Scalar generateColorID_u(int idx)

Generates a color based on the given index.

Parameters

- idx : int - The index.

Returns

- OpenCvSharp.Scalar - The generated color.

generateColorClass Method

public static float4 generateColorClass(int idx)

Generates a color based on the given index.

Parameters

- idx : int - The index.

Returns

- float4 - The generated color.

generateColorClass_u Method

public static OpenCvSharp.Scalar generateColorClass_u(int idx)

Generates a color based on the given index.

Parameters

- idx : int - The index.

Returns

- OpenCvSharp.Scalar - The generated color.

renderObject Method

public static bool renderObject(ObjectData i, bool showOnlyOK)

Determines if an object should be rendered based on its tracking state and the showOnlyOK flag.

Parameters

- i : ObjectData - The object data.
- showOnlyOK : bool - Flag indicating whether to show only objects in OK state.

Returns

- bool - True if the object should be rendered, False otherwise.

_applyFading Method

public static byte _applyFading(double val, float current_alpha, double current_clr)

Applies fading to a color component.

Parameters

- val : double - The color component value.

- current_alpha : float - The current alpha value.
- current_clr : double - The current color value.

Returns

- byte - The faded color component.

applyFading Method

public static Vec4b applyFading(Scalar val, float current_alpha, Scalar current_clr)

Applies fading to an OpenCV scalar.

Parameters

- val : Scalar - The scalar value.
- current_alpha : float - The current alpha value.
- current_clr : Scalar - The current color value.

Returns

- Vec4b - The faded scalar value.

TrackPointState

An enumeration representing the state of a tracked point. It has the following values:

- **OK**: The point is successfully tracked.
- **PREDICTED**: The point is predicted but not confirmed.
- **OFF**: The point is no longer being tracked.

TrackPoint

A struct representing a tracked point in 3D space. It has the following fields:

- **x**: The x-coordinate of the point.
- **y**: The y-coordinate of the point.
- **z**: The z-coordinate of the point.
- **timestamp**: The timestamp of the point.

- **tracking_state**: The state of the point.

Tracklet

A class representing a tracklet, which is a sequence of consecutive tracked points for an object. It has the following fields:

- **id**: The identifier of the tracklet.
- **positions**: A list of tracked points in the tracklet.
- **positions_to_draw**: A list of tracked points to be visualized (may be different than actual points when smoothing the track).
- **tracking_state**: The state of the tracklet.
- **object_type**: The type of the object being tracked.
- **last_detected_timestamp**: The timestamp of the last detected point in the tracklet.
- **recovery_cpt**: Counter used for recovery when a tracklet is temporarily lost.
- **is_alive**: Indicates whether the tracklet is considered alive.
- **recovery_length**: The number of consecutive frames required to recover a lost tracklet.

The class also has methods for adding a detected point to the tracklet and initializing a tracklet with an initial detected point.

TrackingViewer

A class responsible for generating the tracking view. It has the following fields:

- **x_min**: The minimum x-coordinate of the objects to be shown.
- **x_max**: The maximum x-coordinate of the objects to be shown.
- **x_step**: The conversion factor from world position to pixel coordinates in the x-direction.
- **z_step**: The conversion factor from world position to pixel coordinates in the z-direction.
- **z_min**: The minimum z-coordinate of the objects to be shown.
- **window_width**: The width of the tracking view window.

- **window_height**: The height of the tracking view window.
- **tracklets**: A list of tracklets to be visualized in the tracking view.
- **history_duration**: The duration of the track history to be shown (in nanoseconds).
- **min_length_to_draw**: The minimum length of a tracklet to be drawn.
- **background**: The background image of the tracking view.
- **has_background_ready**: Indicates whether the background image is ready.
- **background_color**: The background color of the tracking view.
- **fov_color**: The color of the field of view.
- **camera_offset**: The offset of the camera view in the tracking view.
- **camera_calibration**: The calibration parameters of the camera used for the tracking.
- **fov**: The field of view of the camera.
- **do_smooth**: Indicates whether to smooth the tracklets.
- **smoothing_window_size**: The size of the smoothing window.

The class also has methods for generating the tracking view, setting the camera calibration, and zooming in/out.

Methods

render_2D

A static method that renders the 2D bounding boxes of objects on the left image. It takes as input the left image, the image scale, the detected objects, a flag indicating whether to render the object masks, and a flag indicating whether the tracking is enabled. The method draws the 2D bounding boxes and labels of the detected objects on the left image.

generate_view

This method generates the tracking view by overlaying the 3D bounding boxes of the detected objects on the background image. It takes as input the detected objects, the current camera pose, the tracking view image, and a flag indicating whether the tracking is enabled. The method first transforms the positions of the detected objects from camera coordinates to world coordinates. It then adds new points to the tracklets, removes old points, and detects unchanged tracklets. Finally, it draws all tracklets or positions on the tracking view depending on the tracking mode.

setCameraCalibration

This method sets the camera calibration parameters. It takes as input the calibration parameters.

zoomIn

This method zooms in the tracking view.

zoomOut

This method zooms out the tracking view.

drawTracklets

This method draws all tracklets on the tracking view. It iterates through all tracklets and draws their positions or bounding boxes on the tracking view.

drawPosition

This method draws the positions of the detected objects on the tracking view. It iterates through all detected objects and draws their positions or bounding boxes on the tracking view.

drawScale

This method draws the scale of the tracking view. It draws a line representing 1 meter and adds ticks and text labels indicating the scale.

addToTracklets(ref sl.Objects objects)

This method is used to add objects to the tracklets list. It takes a reference to a sl.Objects object as a parameter.

Parameters

- objects: A reference to a sl.Objects object that contains information about the detected objects.

detectUnchangedTrack(ulong current_timestamp)

This method is used to detect and remove tracks that have not been updated for a certain duration. It takes the current timestamp as a parameter.

Parameters

- current_timestamp: The current timestamp.

pruneOldPoints(ulong ts)

This method is used to prune old points from the tracklets list. It removes points that have timestamps older than a certain duration. It takes the current timestamp as a parameter.

Parameters

- ts: The current timestamp.

computeFOV()

This method is used to compute the field of view (FOV) based on the camera calibration parameters.

zoom(float factor)

This method is used to zoom the view by scaling the minimum and maximum values of the x and z coordinates. It takes a zoom factor as a parameter.

Parameters

- factor: The zoom factor.

generateBackground()

This method is used to generate the background for the visualization. It draws the camera and hotkeys information on the background.

drawCamera()

This method is used to draw the camera shape on the background. It also draws the field of view (FOV) based on the camera calibration parameters.

drawHotkeys()

This method is used to draw the hotkeys information on the background.

toCVPoint(double x, double z)

This method is used to convert a 3D position (x, z) to a 2D point on the OpenCV image. It takes the x and z coordinates as parameters and returns the corresponding 2D point.

Parameters

- x: The x coordinate of the 3D position.
- z: The z coordinate of the 3D position.

toCVPoint(Vector3 position, sl.Pose pose)

This method is used to convert a 3D position (Vector3) to a 2D point on the OpenCV image, based on the camera pose. It takes the 3D position and the camera pose as parameters and returns the corresponding 2D point.

Parameters

- position: The 3D position.
- pose: The camera pose.

toCVPoint(TrackPoint position, sl.Pose pose)

This method is used to convert a TrackPoint object to a 2D point on the OpenCV image, based on the camera pose. It takes the TrackPoint object and the camera pose as parameters and returns the corresponding 2D point.

Parameters

- position: The TrackPoint object.
- pose: The camera pose.

GLViewer class

This class represents a viewer for rendering objects in OpenGL.

Constructors

- GLViewer(): Initializes a new instance of the GLViewer class. It creates the necessary objects for rendering.

Methods

- isAvailable(): bool: Checks if the GLViewer is available for use.
- init(param: CameraParameters, isTrackingON: bool): Initializes the GLViewer with the camera parameters and tracking mode.

- update(image: Mat, objects: Objects, pose: sl.Pose): Updates the GLViewer with the latest image, object data, and camera pose.
- render(): Renders the objects in the GLViewer.
- keyEventFunction(e: NativeWindowKeyEventArgs): Handles key events in the GLViewer.
- mouseEventFunction(e: NativeWindowMouseEventArgs): Handles mouse events in the GLViewer.
- resizeCallback(width: int, height: int): Handles resize events in the GLViewer.
- computeMouseMotion(x: int, y: int): Computes the mouse motion based on the current and previous mouse positions.
- draw(): Draws the objects in the GLViewer.
- generateColorClass(idx: int): sl.float4: Generates a color based on the class index.
- getColorClass(idx: int): float4: Gets the color for a specific class index.
- renderObject(i: ObjectData, showOnlyOK: bool) -> bool: Checks if an object should be rendered based on its tracking state and the showOnlyOK flag.
- setRenderCameraProjection(camParams: CameraParameters, znear: float, zfar: float): Sets the camera projection matrix for rendering.

int getIdx(BODY_38_PARTS part)

This function takes a BODY_38_PARTS enum as input and returns its corresponding index as an integer.

void clearInputs()

This function sets the mouseMotion_ array to zero.

public void exit()

This function checks if the currentInstance variable is not null and sets the available variable to false.

bool available

A boolean variable that determines whether the application is available or not.

*bool isTrackingON_*

A boolean variable that determines if tracking is turned on.

*int[] mouseCurrentPosition_*

An array of integers that stores the current position of the mouse.

*int[] mouseMotion_*

An array of integers that stores the motion of the mouse.

*int[] previousMouseMotion_*

An array of integers that stores the previous motion of the mouse.

const float MOUSE_R_SENSITIVITY

A constant float variable that represents the sensitivity of the mouse rotation.

const float MOUSE_T_SENSITIVITY

A constant float variable that represents the sensitivity of the mouse translation.

const float MOUSE_UZ_SENSITIVITY

A constant float variable that represents the sensitivity of the mouse zooming in.

const float MOUSE_DZ_SENSITIVITY

A constant float variable that represents the sensitivity of the mouse zooming out.

*CameraGL camera_*

An instance of the CameraGL class.

*Matrix4x4 projection_*

A 4x4 matrix that represents the projection matrix.

PointCloud pointCloud

An instance of the PointCloud class.

ShaderData shaderBbox

An instance of the ShaderData class for rendering bounding boxes.

ShaderData shaderLine

An instance of the ShaderData class for rendering lines.

List<ObjectClassName> objectsName

A list of ObjectClassName objects.

Matrix4x4 cam_pose

A 4x4 matrix that represents the camera pose.

Simple3DObject BBox_edges

An instance of the Simple3DObject class for rendering bounding box edges.

Simple3DObject BBox_faces

An instance of the Simple3DObject class for rendering bounding box faces.

Simple3DObject floor_grid

An instance of the Simple3DObject class for rendering a floor grid.

Simple3DObject frustum

An instance of the Simple3DObject class for rendering the frustum.

GLViewer currentInstance

An instance of the GLViewer class.

---

class ImageHandler
public ImageHandler(Resolution res)

The constructor for the ImageHandler class that takes a Resolution object as input.

public void initialize()

This function initializes the OpenGL buffers, shaders, and textures for rendering images.

public void pushNewImage(Mat zedImage)

This function updates the texture with the current zedImage.

public void draw()

This function draws the image using the OpenGL shaders and textures.

public void close()

This function closes and deletes the image texture.

private int texID

An integer variable that represents the texture ID.

private uint imageTex

A unsigned integer variable that represents the image texture.

private ShaderData shaderImage

An instance of the ShaderData class for rendering images.

private Resolution resolution

A Resolution object that represents the resolution of the images.

private uint quad_vb

An unsigned integer variable that represents the vertex buffer ID.

---

class PointCloud
public PointCloud()

The default constructor for the PointCloud class.

void close()

This function closes and deletes the point cloud data.

public void initialize(Resolution res)

This function initializes the point cloud by creating and binding the vertex buffer object and compiling the shaders.

public void pushNewPC(Mat matXYZRGBA)

This function updates the point cloud data with the current XYZRGBA matrix.

public void draw(Matrix4x4 vp)

This function draws the point cloud using the supplied view projection matrix.

uint bufferGLID_

An unsigned integer variable that represents the vertex buffer ID.

Mat mat_

A Mat object that represents the point cloud data.

ShaderData shader

An instance of the ShaderData class for rendering the point cloud.

---

class CameraGL
public CameraGL()

The default constructor for the CameraGL class.

public CameraGL(Vector3 position, Vector3 direction, Vector3 vertical)

The constructor for the CameraGL class that takes the position, direction, and vertical vectors as input.

public void update()

This function updates the camera by checking the orientation of the vertical vector and updating the view matrix accordingly.

public void setProjection(float horizontalFOV, float verticalFOV, float znear, float zfar)

This function sets the projection matrix of the camera based on the specified horizontal and vertical field of view, near and far clipping planes.

public Matrix4x4 getViewProjectionMatrix()

This function returns the view projection matrix of the camera.

public float getHorizontalFOV()

This function returns the horizontal field of view of the camera.

public float getVerticalFOV()

This function returns the vertical field of view of the camera.

public void setOffsetFromPosition(Vector3 o)

This function sets the offset from the camera position.

public Vector3 getOffsetFromPosition()

This function returns the offset from the camera position.

public void setDirection(Vector3 direction, Vector3 vertical)

This function sets the camera direction and vertical vectors.

public void translate(Vector3 t)

This function translates the camera by the specified translation vector.

public void setPosition(Vector3 p)

This function sets the camera position.

public void rotate(Quaternion rot)

This function rotates the camera by the specified rotation quaternion.

public void rotate(Matrix4x4 m)

This function rotates the camera by the specified rotation matrix.

public void setRotation(Quaternion rot)

This function sets the camera rotation to the specified rotation quaternion.

public void setRotation(Matrix4x4 m)

This function sets the camera rotation to the specified rotation matrix.

public Vector3 getPosition()

This function returns the camera position.

public Vector3 getForward()

This function returns the camera forward vector.

public Vector3 getRight()

This function returns the camera right vector.

public Vector3 getUp()

This function returns the camera up vector.

public Vector3 getVertical()

This function returns the camera vertical vector.

public float getZNear()

This function returns the near clipping plane of the camera.

public float getZFar()

This function returns the far clipping plane of the camera.

Code Transformation Documentation

This documentation describes the code transformation for the given code snippet.

Main Class

Properties

- projection_: Represents the projection matrix.

Type

- Public
- Matrix4x4

Fields

- offset_: Represents the offset vector.
- position_: Represents the position vector.
- forward_: Represents the forward direction vector.
- up_: Represents the up direction vector.
- right_: Represents the right direction vector.
- vertical_: Represents the vertical direction vector.
- rotation_: Represents the rotation quaternion.

- view_: Represents the view matrix.
- vpMatrix_: Represents the view-projection matrix.
- horizontalFieldOfView_: Represents the horizontal field of view.
- verticalFieldOfView_: Represents the vertical field of view.
- znear_: Represents the near clipping plane.
- zfar_: Represents the far clipping plane.

Type

- Private
- Vector3 (For offset_, position_, forward_, up_, right_, vertical_)
- Quaternion (For rotation_)
- Matrix4x4 (For view_, vpMatrix_)
- float (For horizontalFieldOfView_, verticalFieldOfView_, znear_, zfar_)

Methods

transform()

This method applies the transformation matrix and rotation to the view matrix. It also transposes the transformation matrix and calculates the inverted view matrix.

- Parameters:
  -
  - transformation: Represents the transformation matrix.
  - rotation_: Represents the rotation quaternion.

updateVPMatrix()

This method updates the view-projection matrix (vpMatrix_) by multiplying the projection matrix (projection_) and view matrix (view_).

- No parameters

Shader Class

Fields

- IMAGE_VERTEX_SHADER: Represents the vertex shader for images.

- IMAGE_FRAGMENT_SHADER: Represents the fragment shader for images.
- POINTCLOUD_VERTEX_SHADER: Represents the vertex shader for point clouds.
- POINTCLOUD_FRAGMENT_SHADER: Represents the fragment shader for point clouds.
- VERTEX_SHADER: Represents the vertex shader.
- FRAGMENT_SHADER: Represents the fragment shader.
- vertexId_: Represents the vertex shader id.
- fragmentId_: Represents the fragment shader id.
- programId_: Represents the program id for the shaders.

Type

- Public
- string[] (For IMAGE_VERTEX_SHADER, IMAGE_FRAGMENT_SHADER, POINTCLOUD_VERTEX_SHADER, POINTCLOUD_FRAGMENT_SHADER, VERTEX_SHADER, FRAGMENT_SHADER)
- uint (For vertexId_, fragmentId_, programId_)

Methods

Shader (Constructor)

Initializes and compiles the shader program by attaching and linking the vertex and fragment shaders.

- Parameters:
  -
    - vs: Represents the vertex shader source code.
    - fs: Represents the fragment shader source code.

getProgramId()

Returns the program id of the shader.

- No parameters

Simple3DObject Class

Fields

- isStatic_: Represents if the object is static or not.
- vaoID_: Represents the vertex array object id.
- shader: Represents the shader data for the object.
- vertices_: Represents the list of vertices.
- colors_: Represents the list of colors.
- indices_: Represents the list of indices.
- normals_: Represents the list of normals.
- is_init: Represents if the object is initialized or not.

Type

- Private
- bool (For isStatic_, is_init)
- uint (For vaoID_)
- ShaderData (For shader)
- List<float> (For vertices_, colors_)
- List<uint> (For indices_)
- List<float> (For normals_)

Methods

Simple3DObject (Constructor)

Initializes a new instance of the Simple3DObject class.

- Parameters:
  -
  - isStatic: Represents if the object is static or not.

init()

Initializes the object by setting up the shader program and initializing the lists for vertices, colors, indices, and normals.

- No parameters

isInit()

Checks if the object is initialized.

- No parameters

addPt()

Adds a point to the list of vertices.

- Parameters:
  -
  - pt: Represents the point to be added.

addClr()

Adds a color to the list of colors.

- Parameters:
  -
  - clr: Represents the color to be added.

addNormal()

Adds a normal to the list of normals.

- Parameters:
  -
  - normal: Represents the normal to be added.

addBBox()

Adds a bounding box to the object.

- Parameters:
  -
  - pts: Represents the list of points defining the bounding box.
  - clr: Represents the color of the bounding box.

addPoint()

Adds a point to the object.

- Parameters:
  -

    - pt: Represents the point to be added.
    - clr: Represents the color of the point.

addLine()

Adds a line to the object.

- Parameters:
    -
    - p1: Represents the starting point of the line.
    - p2: Represents the ending point of the line.
    - clr: Represents the color of the line.

addTriangle()

Adds a triangle to the object.

- Parameters:
    -
    - p1: Represents the first point of the triangle.
    - p2: Represents the second point of the triangle.
    - p3: Represents the third point of the triangle.
    - clr: Represents the color of the triangle.

addFullEdges()

Adds full edges to the object.

- Parameters:
    -
    - pts: Represents the list of points defining the edges.
    - clr: Represents the color of the edges.

addSingleVerticalLine()

Adds a single vertical line to the object.

- Parameters:
    -
    - top_pt: Represents the top point of the line.
    - bot_pt: Represents the bottom point of the line.
    - clr: Represents the color of the line.

addVerticalEdges()

Adds vertical edges to the object.

- Parameters:
    -
    - pts: Represents the list of points defining the edges.
    - clr: Represents the color of the edges.

addTopFace()

Adds the top face of a cube to the object.

- Parameters:
    -
    - pts: Represents the list of points defining the face.
    - clr: Represents the color of the face.

addQuad()

Adds a quad to the object.

- Parameters:
    -
    - quad_pts: Represents the list of points defining the quad.
    - alpha1: Represents the alpha value for the first half of the quad.
    - alpha2: Represents the alpha value for the second half of the quad.
    - clr: Represents the color of the quad.

addVerticalFaces()

Adds vertical faces to the object.

- Parameters:
    -
    - pts: Represents the list of points defining the faces.
    - clr: Represents the color of the faces.

Bottom quads

This code defines a class with several methods to create and manipulate 3D objects in the OpenGL graphics library. The class, called "BottomQuads", has the following methods:

addQuad

This method takes a list of vertices coordinates, an alpha value, and a color as input parameters. It adds a quad (a four-sided polygon) to the 3D object using the provided vertices coordinates, alpha value, and color.

createFrustum

This method creates a frustum shape, which is the part of a cone or pyramid that remains after cutting off the top part with a plane parallel to the base. It takes a CameraParameters object as input and uses its properties to create the frustum shape in 3D space.

addCylinder

This method creates a cylinder shape. It takes a start position, end position, and a color as input parameters. The start and end positions define the two ends of the cylinder, and the color parameter determines the color of the cylinder.

addSphere

This method creates a sphere shape. It takes a position and a color as input parameters. The position parameter defines the center of the sphere, and the color parameter determines the color of the sphere.

pushToGPU

This method pushes the vertices, colors, indices, and normals of the 3D object to the GPU memory. It binds the vertex array object (VAO), vertex buffer objects (VBOs), and buffers the data to the GPU.

clear

This method clears the vertices, colors, indices, and normals of the 3D object.

setDrawingType

This method sets the drawing type for the 3D object. The drawing type can be specified as points, lines, triangles, or quads.

draw

This method draws the 3D object on the screen using the specified drawing type.

In addition to the methods, the class also has several member variables, such as "vertices_", "colors_", "indices_", and "normals_", which store the data of the 3D object. The class also has variables for grid size, position, rotation, and shader data.

Main Window Class

This class represents the main window of the application. It initializes the camera, enables tracking and object detection, and creates an OpenGL window to display the captured 3D point cloud and detected objects.

PROPERTIES

- `USE_BATCHING`: A boolean flag to enable or disable the batch option in the object detection module. Batching system allows to reconstruct trajectories from the object detection module by adding Re-Identification / Appareance matching. Use with caution if image retention is activated.
- `isTrackingON`: A boolean flag to indicate whether object tracking is enabled or not.
- `isPlayback`: A boolean flag to indicate whether the camera is in playback mode or not.
- `viewer`: An instance of the `GLViewer` class used to display the captured 3D point cloud and detected objects in an OpenGL window.
- `zedCamera`: An instance of the `Camera` class representing the ZED camera.
- `obj_runtime_parameters`: An instance of the `ObjectDetectionRuntimeParameters` class representing the runtime parameters for object detection.
- `runtimeParameters`: An instance of the `RuntimeParameters` class representing the runtime parameters for camera capturing.
- `batchParameters`: An instance of the `BatchParameters` class representing the parameters for the batching system.
- `pointCloud`, `imageLeft`: Instances of the `sl.Mat` class representing the 3D point cloud and left image captured by the camera.
- `imageRenderLeft`, `imageTrackOcv`, `imageLeftOcv`, `globalImage`: Instances of the `OpenCvSharp.Mat` class representing the rendered left image, tracking view, left image, and global image respectively.
- `imgScale`: An instance of the `sl.float2` class representing the scale factor for the image display.
- `camWorldPose`, `camCameraPose`: Instances of the `Pose` class representing the camera pose in world and camera reference frames respectively.
- `pcRes`, `displayRes`: Instances of the `Resolution` class representing the resolution of the point cloud and display respectively.
- `objects`: An instance of the `Objects` class representing the detected objects.
- `maxDepthDistance`: The maximum depth distance for the camera.
- `trackViewGenerator`: An instance of the `TrackingViewer` class used to generate the tracking view.
- `batchHandler`: An instance of the `BatchSystemHandler` class used to handle the batching system.
- `detection_confidence`: An integer representing the detection confidence threshold for object detection.

- `window_name`: A string representing the name of the window.

CONSTRUCTOR

MainWindow(string[] args)

The constructor initializes the camera, enables tracking and object detection, sets the initial parameters, and creates an OpenGL window.

METHODS

CreateWindow()

Creates an OpenGL window using the `NativeWindow` class provided by the `OpenGL.CoreUI` namespace.

NativeWindow_ContextCreated(object sender, NativeWindowEventArgs e)

Callback method called when the OpenGL context is created. It initializes the OpenGL settings and the viewer.

NativeWindow_Render(object sender, NativeWindowEventArgs e)

Callback method called when the OpenGL window needs to be rendered. It grabs the camera frames, retrieves the objects, updates the views, and renders the OpenGL window.

NativeWindow_MouseEvent(object sender, NativeWindowMouseEventArgs e)

Callback method called when a mouse event occurs in the OpenGL window. It updates the viewer's mouse motion.

NativeWindow_Resize(object sender, EventArgs e)

Callback method called when the OpenGL window is resized. It resizes the viewer.

close()

Closes the application by disabling tracking and object detection, closing the camera, and freeing the allocated memory.

parseArgs(string[] args, ref sl.InitParameters param)

This function is used to parse the command line arguments and set the appropriate values in the `param` object.

Parameters

- `args`: An array of command line arguments.

- `param`: A reference to the `sl.InitParameters` object to be updated based on the command line arguments.

Example Usage

```
sl.InitParameters parameters = new sl.InitParameters();
parseArgs(args, ref parameters);
```

Usage in Command Line

```
$ myApp.exe path/to/file.svo
$ myApp.exe 192.168.0.1
$ myApp.exe HD2K
$ myApp.exe HD1080
$ myApp.exe HD720
$ myApp.exe VGA
```

Code Explanation

- The function first checks if the length of `args` is greater than 0 and if the first argument ends with ".svo". If both conditions are true, it sets the `inputType` of `param` to `SVO`, assigns the first argument to `pathSVO`, sets `isPlayback` to `true`, and displays a message indicating that SVO File input is being used.
- If the first condition is true but the second condition is false, it checks if the first argument is a valid IP address. If it is, it sets the `inputType` of `param` to `STREAM`, assigns the IP address to `ipStream`, and displays a message indicating that Stream input is being used.
- If both conditions are false, it checks the value of the first argument to determine the desired camera resolution. It sets the `resolution` of `param` based on the value and displays a corresponding message.
- If none of the conditions are true, the function does nothing.

CPP Code Documentation

```
# GLViewer.cpp

### Constants:

- `FADED_RENDERING`: Flag for faded rendering.
- `grid_size`: Size of the floor grid.

### Global Variables:

- `VERTEX_SHADER`: Vertex shader source code.
- `FRAGMENT_SHADER`: Fragment shader source code.

#### Functions:
```

- `addVert(Simple3DObject &obj, float i_f, float limit, float height, sl::float4 &clr)`: Helper function for adding vertices to a 3D object.
- `createFrustum(sl::CameraParameters param)`: Creates a 3D object representing the camera frustum.
- `CloseFunc()`: Callback function for closing the window.
- `GLViewer()`: Constructor for the `GLViewer` class.
- `~GLViewer()`: Destructor for the `GLViewer` class.
- `exit()`: Closes the viewer and stops rendering.
- `isAvailable()`: Checks if the viewer is available for rendering.
- `init(int argc, char **argv, sl::CameraParameters &param, bool isTrackingON)`: Initializes the viewer with the specified camera parameters and tracking state.
- `render()`: Renders the current frame.
- `updateData(sl::Mat &matXYZRGBA, std::vector<sl::ObjectData> &objs, sl::Transform& pose)`: Updates the point cloud data and object detections to be rendered.
- `createBboxRendering(std::vector<sl::float3> &bbox, sl::float4 bbox_clr)`: Creates the rendering of a bounding box with the specified coordinates and color.
- `createIDRendering(sl::float3 & center, sl::float4 clr, unsigned int id)`: Creates the rendering of an object label with the specified position, color, and ID.
- `update()`: Updates the camera and point cloud buffer.
- `draw()`: Draws the scene with the current camera view and object renderings.
- `clearInputs()`: Clears the input states.
- `drawCallback()`: Callback function for rendering the frame.
- `mouseButtonCallback(int button, int state, int x, int y)`: Callback function for mouse button events.
- `mouseMotionCallback(int x, int y)`: Callback function for mouse motion events.
- `reshapeCallback(int width, int height)`: Callback function for window reshape events.
- `keyPressedCallback(unsigned char c, int x, int y)`: Callback function for keyboard key press events.
- `keyReleasedCallback(unsigned char c, int x, int y)`: Callback function for keyboard key release events.

GLVIEWER::IDLE()
Function that is called when the OpenGL window is idle. It posts a redisplay event to the window.

SIMPLE3DOBJECT::SIMPLE3DOBJECT(SL::TRANSLATION POSITION, BOOL ISSTATIC)
Constructor function for the `Simple3DObject` class. Initializes the object's variables and sets the position and static status.

SIMPLE3DOBJECT::~SIMPLE3DOBJECT()
Destructor function for the `Simple3DObject` class. Deletes the object's vertex arrays and buffers.

SIMPLE3DOBJECT::ADDBBOX(STD::VECTOR<SL::FLOAT3> &PTS, SL::FLOAT4 CLR)
Adds a bounding box to the `Simple3DObject`. Takes a vector of 3D coordinates and a color as input.

SIMPLE3DOBJECT::ADDPT(SL::FLOAT3 PT)
Adds a point to the `Simple3DObject`. Takes a 3D coordinate as input.

SIMPLE3DOBJECT::ADDCLR(SL::FLOAT4 CLR)

Adds a color to the Simple3DObject. Takes a color as input.

SIMPLE3DOBJECT::ADDPOINT(SL::FLOAT3 PT, SL::FLOAT4 CLR)
Adds a point with a color to the Simple3DObject. Takes a 3D coordinate and a color as input.

SIMPLE3DOBJECT::ADDLINE(SL::FLOAT3 P1, SL::FLOAT3 P2, SL::FLOAT4 CLR)
Adds a line segment with a color to the Simple3DObject. Takes 2 3D coordinates and a color as input.

SIMPLE3DOBJECT::ADDTRIANGLE(SL::FLOAT3 P1, SL::FLOAT3 P2, SL::FLOAT3 P3, SL::FLOAT4 CLR)
Adds a triangle with a color to the Simple3DObject. Takes 3 3D coordinates and a color as input.

SIMPLE3DOBJECT::ADDFULLEDGES(STD::VECTOR<SL::FLOAT3> &PTS, SL::FLOAT4 CLR)
Adds full edges to the Simple3DObject. Takes a vector of 3D coordinates and a color as input.

SIMPLE3DOBJECT::ADDVERTICALEDGES(STD::VECTOR<SL::FLOAT3> &PTS, SL::FLOAT4 CLR)
Adds vertical edges to the Simple3DObject. Takes a vector of 3D coordinates and a color as input.

SIMPLE3DOBJECT::ADDTOPFACE(STD::VECTOR<SL::FLOAT3> &PTS, SL::FLOAT4 CLR)
Adds the top face to the Simple3DObject. Takes a vector of 3D coordinates and a color as input.

SIMPLE3DOBJECT::ADDVERTICALFACES(STD::VECTOR<SL::FLOAT3> &PTS, SL::FLOAT4 CLR)
Adds vertical faces to the Simple3DObject. Takes a vector of 3D coordinates and a color as input.

SIMPLE3DOBJECT::PUSHTOGPU()
Pushes the Simple3DObject data to the GPU. If the object is not static or has not been initialized, it generates and binds vertex arrays and buffers.

SIMPLE3DOBJECT::CLEAR()
Clears the Simple3DObject data. Removes all vertices, colors, and indices.

SIMPLE3DOBJECT::SETDRAWINGTYPE(GLENUM TYPE)
Sets the drawing type for the Simple3DObject. Takes a GLenum as input.

SIMPLE3DOBJECT::DRAW()

Draws the Simple3DObject. Binds the vertex array and calls glDrawElements with the drawing type and number of indices.

SIMPLE3DOBJECT::TRANSLATE(CONST SL::TRANSLATION& T)
Translates the Simple3DObject by a given translation vector. Takes a translation vector as input.

SIMPLE3DOBJECT::SETPOSITION(CONST SL::TRANSLATION& P)
Sets the position of the Simple3DObject. Takes a translation vector as input.

SIMPLE3DOBJECT::SETRT(CONST SL::TRANSFORM& MRT)
Sets the rotation and translation of the Simple3DObject using a 4x4 transformation matrix. Takes a transformation matrix as input.

SIMPLE3DOBJECT::ROTATE(CONST SL::ORIENTATION& ROT)
Rotates the Simple3DObject by a given orientation. Takes an orientation as input.

SIMPLE3DOBJECT::ROTATE(CONST SL::ROTATION& M)
Rotates the Simple3DObject by a given rotation matrix. Takes a rotation matrix as input.

SIMPLE3DOBJECT::SETROTATION(CONST SL::ORIENTATION& ROT)
Sets the rotation of the Simple3DObject using an orientation. Takes an orientation as input.

SIMPLE3DOBJECT::SETROTATION(CONST SL::ROTATION& M)
Sets the rotation of the Simple3DObject using a rotation matrix. Takes a rotation matrix as input.

CONST SL::TRANSLATION& SIMPLE3DOBJECT::GETPOSITION()
Gets the position of the Simple3DObject. Returns a translation vector.

SL::TRANSFORM SIMPLE3DOBJECT::GETMODELMATRIX()
Gets the model matrix of the Simple3DObject. Returns a 4x4 transformation matrix.

SHADER::SHADER(GLCHAR *VS, GLCHAR* FS)
Constructor function for the Shader class. Compiles the vertex and fragment shaders and creates the shader program.

SHADER::~SHADER()
Destructor function for the Shader class. Deletes the vertex and fragment shaders and the shader program.

GLUINT SHADER::GETPROGRAMID()
Gets the shader program ID. Returns a GLuint.

BOOL SHADER::COMPILE(GLUINT &SHADERID, GLENUM TYPE, GLCHAR* SRC)
Compiles the shader source code. Takes the shader ID, type, and source code as input. Returns a bool indicating success or failure.

POINTCLOUD::POINTCLOUD()
Constructor function for the `PointCloud` class. Initializes the `hasNewPCL_` flag.

POINTCLOUD::~POINTCLOUD()
Destructor function for the `PointCloud` class. Calls the `close` function.

VOID POINTCLOUD::CLOSE()
Cleans up the `PointCloud` object. Frees the GPU memory, unmaps the buffer, and deletes the OpenGL buffer.

PointCloud Class

INITIALIZE(RES: RESOLUTION)
This method initializes the point cloud by generating a buffer object using OpenGL functions and registering it with CUDA for writing. It also initializes the shader program and allocates memory for the point cloud data on the GPU.

Parameters

- `res`: A `Resolution` object specifying the resolution of the point cloud.

PUSHNEWPC(MATXYZRGBA: MAT)
This method updates the point cloud data with a new frame of XYZRGBA values. It checks if the GPU memory has been initialized and sets the new data.

Parameters

- `matXYZRGBA`: A `Mat` object containing the XYZRGBA values for the new frame.

UPDATE()
This method updates the point cloud data on the GPU if there is new data available. It copies the data from the GPU memory to the mapped buffer.

DRAW(VP: TRANSFORM)
This method renders the point cloud using OpenGL functions. It binds the shader program, sets the MVP matrix uniform, and draws the point cloud as a set of points.

Parameters

- `vp`: A `Transform` object specifying the view-projection matrix.

CameraGL Class

CAMERAGL(POSITION: TRANSLATION, DIRECTION: TRANSLATION, VERTICAL: TRANSLATION)
This constructor initializes the camera with a position, direction, and vertical vector. It sets the camera's position and calculates the view matrix. It also sets the camera's projection matrix using the specified field of view and near/far planes.

Parameters

- `position`: A `Translation` object specifying the camera's position.
- `direction`: A `Translation` object specifying the camera's direction.
- `vertical`: A `Translation` object specifying the camera's vertical vector.

UPDATE()
This method updates the camera's view matrix and view-projection matrix. It checks if the vertical vector and up vector are in the same direction and adjusts the vertical vector if necessary.

SETPROJECTION(HORIZONTALFOV: FLOAT, VERTICALFOV: FLOAT, ZNEAR: FLOAT, ZFAR: FLOAT)
This method sets the camera's projection matrix using the specified field of view and near/far planes. It calculates the projection matrix based on the specified FOV angles and near/far planes.

Parameters

- `horizontalFOV`: A float specifying the horizontal field of view angle in degrees.
- `verticalFOV`: A float specifying the vertical field of view angle in degrees.
- `znear`: A float specifying the near plane distance.
- `zfar`: A float specifying the far plane distance.

GETVIEWPROJECTIONMATRIX() -> TRANSFORM
This method returns the camera's view-projection matrix.

GETHORIZONTALFOV() -> FLOAT
This method returns the camera's horizontal field of view angle in degrees.

GETVERTICALFOV() -> FLOAT
This method returns the camera's vertical field of view angle in degrees.

SETOFFSETFROMPOSITION(O: TRANSLATION)
This method sets the camera's offset from its position.

Parameters

- o: A Translation object specifying the camera's offset from its position.

GETOFFSETFROMPOSITION() -> TRANSLATION

This method returns the camera's offset from its position.

SETDIRECTION(DIRECTION: TRANSLATION, VERTICAL: TRANSLATION)

This method sets the camera's direction and vertical vectors. It normalizes the direction vector and calculates the rotation matrix based on the original forward vector and the normalized direction vector.

Parameters

- direction: A Translation object specifying the camera's direction.
- vertical: A Translation object specifying the camera's vertical vector.

TRANSLATE(T: TRANSLATION)

This method translates the camera's position by the specified translation vector.

Parameters

- t: A Translation object specifying the translation vector.

SETPOSITION(P: TRANSLATION)

This method sets the camera's position.

Parameters

- p: A Translation object specifying the camera's position.

ROTATE(ROT: ORIENTATION)

This method rotates the camera by the specified orientation.

Parameters

- rot: An Orientation object specifying the rotation.

ROTATE(M: ROTATION)

This method rotates the camera by the specified rotation matrix.

Parameters

- m: A Rotation object specifying the rotation matrix.

SETROTATION(ROT: ORIENTATION)

This method sets the camera's rotation.

Parameters

- rot: An Orientation object specifying the rotation.

SETROTATION(M: ROTATION)

This method sets the camera's rotation using the specified rotation matrix.

Parameters

- `m`: A `Rotation` object specifying the rotation matrix.

GETPOSITION() -> TRANSLATION
This method returns the camera's position.

GETFORWARD() -> TRANSLATION
This method returns the camera's forward vector.

GETRIGHT() -> TRANSLATION
This method returns the camera's right vector.

GETUP() -> TRANSLATION
This method returns the camera's up vector.

GETVERTICAL() -> TRANSLATION
This method returns the camera's vertical vector.

GETZNEAR() -> FLOAT
This method returns the camera's near plane distance.

GETZFAR() -> FLOAT
This method returns the camera's far plane distance.

UPDATEVECTORS()
This method updates the camera's forward, up, and right vectors based on the current rotation.

UPDATEVIEW()
This method updates the camera's view matrix based on the current position and rotation.

UPDATEVPMATRIX()
This method updates the camera's view-projection matrix by multiplying the projection matrix with the view matrix.

Main.cpp

Object Detection with Re-Identification Documentation

INTRODUCTION
This code example demonstrates how to perform object detection with possible re-identification on a 2D display using the ZED 3D camera. The code detects objects in real time and displays the results on a 2D display. The code also includes a GUI for visualizing the detected objects and their trajectories.

SETUP

- Include the necessary dependencies:
  -

    - sl/Camera.hpp: ZED camera library
    - iostream: Standard input/output library
    - fstream: File input/output library
    - deque: Double-ended queue library
- Define constants and flags:
    -
        - ENABLE_GUI: Flag to enable or disable the GUI (Graphical User Interface)
        - ENABLE_BATCHING_REID: Flag to enable or disable object re-identification using batch processing
- Import necessary namespaces:
    -
        - std: Standard namespace for C++
        - sl: Namespace for ZED camera library
- Define a boolean variable is_playback to check if the camera is in playback mode or not.
- Define the function print to print messages with optional error codes.
- Define the function parseArgs to parse command line arguments and set the initialization parameters for the ZED camera.
- Define the function printHelp to print a helpful message with the hotkeys for the GUI.

MAIN FUNCTION

- Define a boolean variable isJetson to check if the code is running on a Jetson platform.
- Create a ZED camera object zed.
- Set the ZED camera initialization parameters init_parameters:
    -
        - Set the depth mode to ULTRA for high-quality depth sensing.
        - Set the maximum depth distance to 10 meters.
        - Set the coordinate system to RIGHT_HANDED_Y_UP to match the OpenGL coordinate system.
        - Set the SDK verbosity to 1 to display additional information during runtime.
        - Parse the command line arguments using the parseArgs function to configure the ZED camera parameters.

- Open the ZED camera using the `open` function and check if the camera was successfully opened.
- Retrieve the camera configuration and calibration parameters.
- Enable positional tracking using the `enablePositionalTracking` function.
- Print a message indicating the object detection module is being loaded.
- Define the object detection parameters `detection_parameters`:
    -
        - Set the `enable_tracking` flag to true to enable object tracking.
        - Set the `enable_segmentation` flag to false to disable object segmentation.
        - Set the `detection_model` to `MULTI_CLASS_BOX_FAST` for fast object detection (for Jetson platforms) or `MULTI_CLASS_BOX_ACCURATE` for accurate object detection (for non-Jetson platforms).
        - Optionally, enable object re-identification using batch processing by setting `ENABLE_BATCHING_REID` flag to 1 and configuring the batch parameters.
- Enable object detection using the `enableObjectDetection` function and check if it was successfully enabled.
- Set the detection confidence threshold for all object classes.
- Create necessary variables and objects for the GUI (if `ENABLE_GUI` flag is set to 1):
    -
        - Create a window for the 2D view tracking.
        - Create a trackbar to adjust the detection confidence threshold.
        - Define variables for the image resolution and scale.
        - Create a global image to store the image and tracks view.
        - Retrieve references to the image and tracks view parts.
        - Initialize an `sl::Mat` object from the OpenCV image reference.
        - Create an `sl::Resolution` object for the point cloud resolution.
        - Retrieve the camera calibration parameters.
        - Create a point cloud and a GL viewer object.
        - Display the help message with the GUI hotkeys.

- Define a `std::map` variable `id_counter` to keep track of the IDs of detected objects.
- Define the runtime parameters for object detection.
- Define `Objects` variable `objects` to store the detected objects.
- Enter the main loop to perform object detection and visualization until the program is closed or the end of playback is reached (if in playback mode):
    -
        - Update the detection confidence threshold based on the trackbar value.
        - Retrieve the objects using the `retrieveObjects` function.
            -
                - If the objects are successfully retrieved, proceed with visualization and processing.
        - Update the GUI and check for keyboard input events (if `ENABLE_GUI` flag is set to 1).
        - Check if object re-identification batch processing is available and print the detected and re-ID'd object IDs.
- Clean up and exit the program:
    -
        - Clean up resources used by the GUI (if `ENABLE_GUI` flag is set to 1).
        - Free memory used by the point cloud and image (if `ENABLE_GUI` flag is set to 1).
        - Disable object detection and close the ZED camera.

OTHER FUNCTIONS

- The `print` function prints messages with an optional error code and additional suffix message.
- The `parseArgs` function parses command line arguments and sets the initialization parameters for the ZED camera.
- The `printHelp` function prints a message with the hotkeys for the GUI.

CONCLUSION

This code example demonstrates how to perform object detection with possible re-identification using the ZED camera. The code showcases the functionality of the ZED camera library and provides a GUI for visualizing the detected objects and their trajectories. The code can be used as a starting point for developing applications that require real-time object detection and tracking.

TrackingViewer.cpp

Parameters

- objects (type: sl::Objects&): The list of tracked objects.
- tracking_view (type: cv::Mat&): The matrix representing the tracking view.
- current_camera_pose (type: sl::Pose): The current pose of the camera.

Code

```
for (auto obj : objects.object_list) {
sl::float4 generated_color_sl = getColorClass((int) obj.label)* 255.0f;
cv::Scalar generated_color(generated_color_sl.x, generated_color_sl.y, generated_color_sl.z, 255);

switch (obj.label) { case sl::OBJECT_CLASS::PERSON: cv::circle(tracking_view, toCVPoint(obj.position, current_camera_pose), 5, generated_color, 5); break; case sl::OBJECT_CLASS::VEHICLE: { if (!obj.bounding_box.empty()) { cv::Point2i rect_center = toCVPoint(obj.position, current_camera_pose); int square_size = 10; cv::Point2i top_left_corner = rect_center - cv::Point2i(square_size, square_size * 2); cv::Point2i right_bottom_corner = rect_center + cv::Point2i(square_size, square_size * 2);
#if (defined(CV_VERSION_EPOCH) && CV_VERSION_EPOCH == 2)
cv::rectangle(tracking_view, top_left_corner, right_bottom_corner, generated_color, -1);
#else
cv::rectangle(tracking_view, top_left_corner, right_bottom_corner, generated_color, cv::FILLED);
#endif
}
break;
}
case sl::OBJECT_CLASS::LAST:
break;
default:
break;
}
}
```

Description

- Creates a color based on the label of the object.
- Based on the label, either a circle or a rectangle is drawn at the position of the object on the tracking view matrix.

TRACKINGVIEWER::DRAWSCALE

This function is responsible for drawing the scale on the tracking view.

Parameters

- `tracking_view` (type: cv::Mat&): The matrix representing the tracking view.

Code

```
int one_meter_horizontal = static_cast<int>(1000.f / x_step + .5f);
cv::Point2i st_pt(25, window_height - 50);
cv::Point2i end_pt(25 + one_meter_horizontal, window_height - 50);
int thickness = 1;

cv::line(tracking_view, st_pt, end_pt, cv::Scalar(0, 0, 0, 255), thickness);

cv::line(tracking_view, st_pt + cv::Point2i(0, -3), st_pt + cv::Point2i(0, 3), cv::Scalar(0, 0, 0, 255), thickness);
cv::line(tracking_view, end_pt + cv::Point2i(0, -3), end_pt + cv::Point2i(0, 3), cv::Scalar(0, 0, 0, 255),
thickness);

cv::putText(tracking_view, "1m", end_pt + cv::Point2i(5, 5), 1, 1.0, cv::Scalar(0, 0, 0, 255), 1);
```

Description

- Determines the size of one meter on the tracking view based on the x step.
- Draws a line representing the scale.
- Adds ticks at the starting and ending points of the line.
- Adds text indicating the length of the scale.

TRACKINGVIEWER::GENERATEBACKGROUND

This function generates the background for the tracking view by drawing the camera and hotkeys information.

Code

```
drawCamera();
drawHotkeys();
has_background_ready = true;
```

Description

- Calls the `drawCamera` function to draw the camera on the background.
- Calls the `drawHotkeys` function to draw the hotkeys information on the background.

TRACKINGVIEWER::DRAWCAMERA

This function is responsible for drawing the camera on the background of the tracking view.

Code

```
cv::Scalar camera_color(255, 117, 44, 255);

int camera_size = 10;
```

```
int camera_height = window_height - camera_offset;
cv::Point2i camera_left_pt(window_width / 2 - camera_size / 2, camera_height);
cv::Point2i camera_right_pt(window_width / 2 + camera_size / 2, camera_height);

std::vector<cv::Point2i> camera_pts{
cv::Point2i(window_width / 2 - camera_size, camera_height),
cv::Point2i(window_width / 2 + camera_size, camera_height),
cv::Point2i(window_width / 2 + camera_size, camera_height + camera_size / 2),
cv::Point2i(window_width / 2 - camera_size, camera_height + camera_size / 2)};
cv::fillConvexPoly(background, camera_pts, camera_color);

if (fov < 0.0f)
computeFOV();

float z_at_x_max = x_max / tan(fov / 2.0f);
cv::Point2i left_intersection_pt = toCVPoint(x_min, -z_at_x_max), right_intersection_pt = toCVPoint(x_max,
-z_at_x_max);

uchar clr[4] = {static_cast<uchar>(camera_color(0)), static_cast<uchar>(camera_color(1)),
static_cast<uchar>(camera_color(2)), static_cast<uchar>(camera_color(3))};

cv::LineIterator left_line_it(background, camera_left_pt, left_intersection_pt, 8);
for (int i = 0; i < left_line_it.count; ++i, ++left_line_it) {
cv::Point2i current_pos = left_line_it.pos();
if (i % 5 == 0 || i % 5 == 1) {
(*left_line_it)[0] = clr[0]; (*left_line_it)[1] = clr[1];
(*left_line_it)[2] = clr[2]; (*left_line_it)[3] = clr[3];
}

for (int r = 0; r < current_pos.y; ++r) { float ratio = float(r) / camera_height; background.at<cv::Vec4b>(r,
current_pos.x) = applyFading(background_color, ratio, fov_color); }
}

cv::LineIterator right_line_it(background, camera_right_pt, right_intersection_pt, 8);
for (int i = 0; i < right_line_it.count; ++i, ++right_line_it) {
cv::Point2i current_pos = right_line_it.pos();
if (i % 5 == 0 || i % 5 == 1) {
(*right_line_it)[0] = clr[0]; (*right_line_it)[1] = clr[1];
(*right_line_it)[2] = clr[2];
}

for (int r = 0; r < current_pos.y; ++r) { float ratio = float(r) / camera_height; background.at<cv::Vec4b>(r,
current_pos.x) = applyFading(background_color, ratio, fov_color); }
}

for (int c = window_width / 2 - camera_size / 2; c <= window_width / 2 + camera_size / 2; ++c) {
for (int r = 0; r < camera_height; ++r) {
float ratio = float(r) / camera_height;
background.at<cv::Vec4b>(r, c) = applyFading(background_color, ratio, fov_color);
}
}
```

Description

- Sets the color and size of the camera.
- Determines the coordinates of the points that form the camera shape.
- Fills the shape of the camera with the specified color.
- Computes the field of view (FOV) if not already computed.
- Calculates the intersection points of the FOV with the window borders.
- Draws the FOV on the background matrix using a dotted line.
- Fills the area inside the camera shape with fading effect.
- Adds fading effect to the lines representing the FOV.

TRACKINGVIEWER::DRAWHOTKEYS

This function is responsible for drawing the hotkeys information on the background of the tracking view.

Code

```
cv::Scalar hotkeys_clr(0, 0, 0, 255);
cv::putText( background, "Press 'i' to zoom in", cv::Point2i(25, window_height - 25), 1,
1.0, hotkeys_clr, 1 );
cv::putText( background, "Press 'o' to zoom out", cv::Point2i(25, window_height - 15), 1,
1.0, hotkeys_clr, 1 );
```

Description

- Sets the color of the text for the hotkeys information.
- Draws the text indicating the hotkey for zooming in.
- Draws the text indicating the hotkey for zooming out.

UTILS: FROM 3D WORLD TO 2D TRACK VIEW

This section contains utility functions for converting 3D coordinates to 2D coordinates on the track view.

TrackingViewer::toCVPoint (double x, double z)

This function converts 3D coordinates to 2D coordinates on the track view.

Parameters

- `x` (type: double): The x coordinate in 3D world.
- `z` (type: double): The z coordinate in 3D world.

Returns

- The 2D coordinates on the track view.

TrackingViewer::toCVPoint (sl::float3 position, sl::Pose pose)

This function converts 3D coordinates with pose information to 2D coordinates on the track view.

Parameters

- `position` (type: sl::float3): The 3D position of the object.
- `pose` (type: sl::Pose): The pose of the camera.

Returns

- The 2D coordinates on the track view.

TrackingViewer::toCVPoint (TrackPoint position, sl::Pose pose)

This function converts a TrackPoint object with pose information to 2D coordinates on the track view.

Parameters

- `position` (type: TrackPoint): The position of the object.
- `pose` (type: sl::Pose): The pose of the camera.

Returns

- The 2D coordinates on the track view.

TrackingViewer::toCVPoint (TrackPoint position)

This function converts a TrackPoint object to 2D coordinates on the track view.

Parameters

- `position` (type: TrackPoint): The position of the object.

Returns

- The 2D coordinates on the track view.