Software architecture document for project iSLAM (indoor SLAM)

0. Authors

- Kozlov Kirill (22215)
- Shalygin Igor (22215)
- Tsoy Anton (22215)

1. Goals and limitations

1.1. Key functional requirements

Use-Cases:

- 3D map construction - main use case with getting 3D points using video source
- Calibration - getting intrinsic camera parameters necessary for 3D map construction using video or photos with chessboard
- Single visualization - case when user wants to observe map constructed before

1.2. Non-functional requirements

- The software is guaranteed to be deployable on Windows Subsystem for Linux (WSL2), Ubuntu 22 (and probably other Linux distributions)
- The software must visualize three-dimensional objects obtained from photos/video and save their coordinates in the TXT file.

1.3. Architectural goals

- Clear usage ways
- Easy integration methods
- Acceptable accuracy of 3D reconstruction (at least we want to recognize some objects…)
- The best video processing performance will be with a powerful video card from Nvidia

1.4. Additional goals, restrictions and preferences

- Most of the team members don't know how easy it is to get depressed doing Computer Vision. So it would be good to understand it during the current project.

2. Goals analysis

Main part of the project is a simultaneous localization and mapping algorithm (SLAM) for indoor domain. To produce fast enough algorithm it looks good to use C++ with OpenCV framework

2.0. Calibration

At first, we need to get the camera's intrinsic parameters, as they are needed in further rotation

and transition estimation. These parameters are combined in a calibration matrix. The user must specify a path to this matrix in the configuration file.

If a user doesn't have this matrix he can obtain it using our additional calibration module based on the built-in OpenCV calibration pipeline.

1. It consumes set of photos or video with simple chessboard and finds its corners to estimate intrinsic parameters (this is possible because ideal corners positions are known)
2. Estimated calibration matrix and additional intrinsic parameters are saved to path specified in configuration JSON with XML format
3. So next user can execute our main module

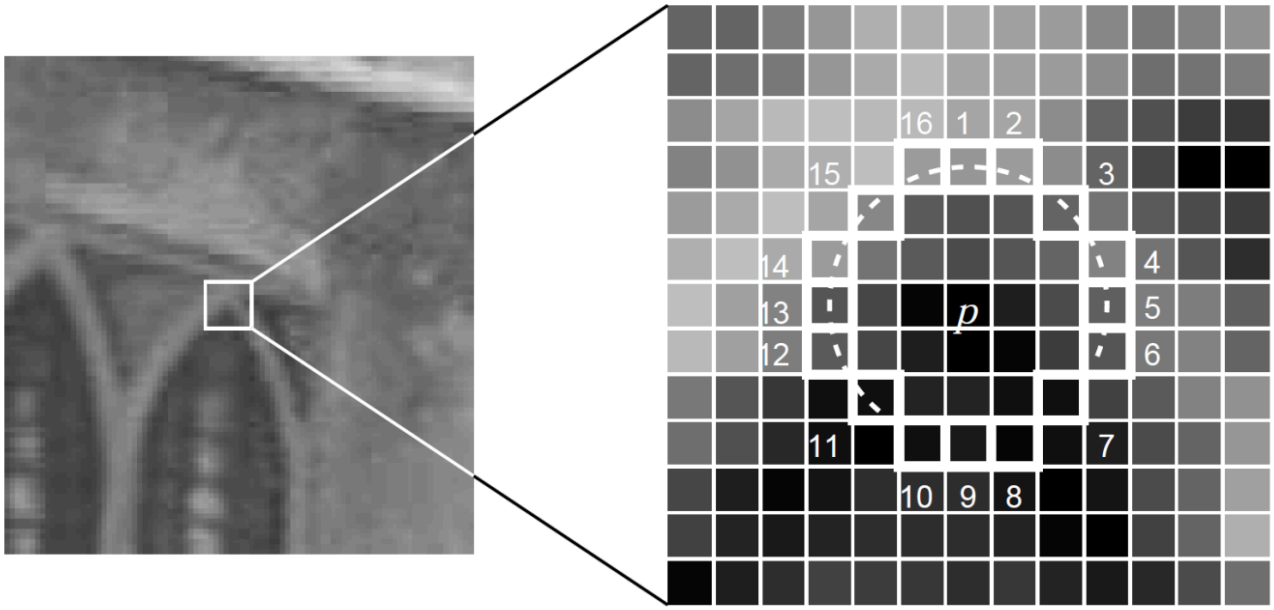
2.1. Indoor SLAM

As a first reference we've tried to learn Large-Scale Direct Monocular SLAM but this article was too complex and hard to understand for CV beginners so our next and main reference became "Abiel Aguilar-González, Miguel Arias-Estrada. Dense mapping for monocular-SLAM" with some modifications which will be described below.

2.1.1. Feature extraction

A special feature extraction module for the further matching part. This module takes part in the main module. At each iteration of the loop, we take a specific frame from the sequence, then its key points are extracted.

For feature extraction we use the FAST algorithm (Features from Accelerated Segment Test). It provides low processing time and computational requirements. We use the implementation of the FAST algorithm from the OpenCV standard library.



2.1.2. Key points correspondence finding

Module for finding corresponding features in two previously extracted key points sets for the Rt matrix estimation and triangulation.

At the beginning, we used a feature tracking algorithm taken from “Abiel Aguilar-González, Miguel Arias-Estrada. Dense mapping for monocular-SLAM” article. But it had significant speed and feature count problems. So we moved to OpenCV's CUDA FLANN based matching algorithm, which uses SIFT descriptor extractor. With this structure we can acquire high features count and improved working time.

Matcher	+	-
ORB with BF	fast	Works not correct with rotation and similar features
SIFT with BF	Good for rotation	Works not correct with similar features
SIFT with FLANN	Good for many similar features and rotations.	

2.1.3. Transition estimation

This step differs for the first two frames and for further ones.

2.1.3.1. The first frames’ pair

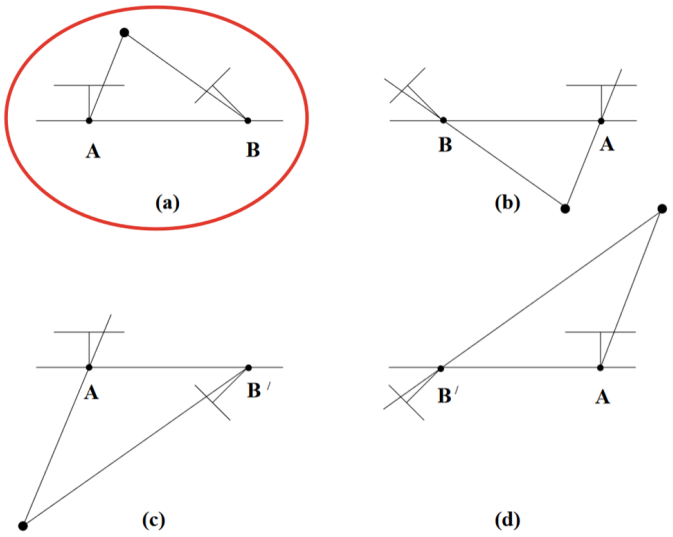
For the first two frames estimation is established by the RANSAC algorithm provided with OpenCV Camera Calibration and 3D Reconstruction module.

1. findEssentialMat solves this equation with corresponding 2D points and calibration matrix K to find essential matrix E:

$$[p_{2i}; 1]^T K^{-T} E K^{-1} [p_{1i}; 1] = 0$$

2. recoverPose gets 4 variants of transition using singular values decomposition and finds best one using triangulation (triangulation will be explained below):

$$[R_1, t], [R_1, -t], [R_2, t], [R_2, -t].$$

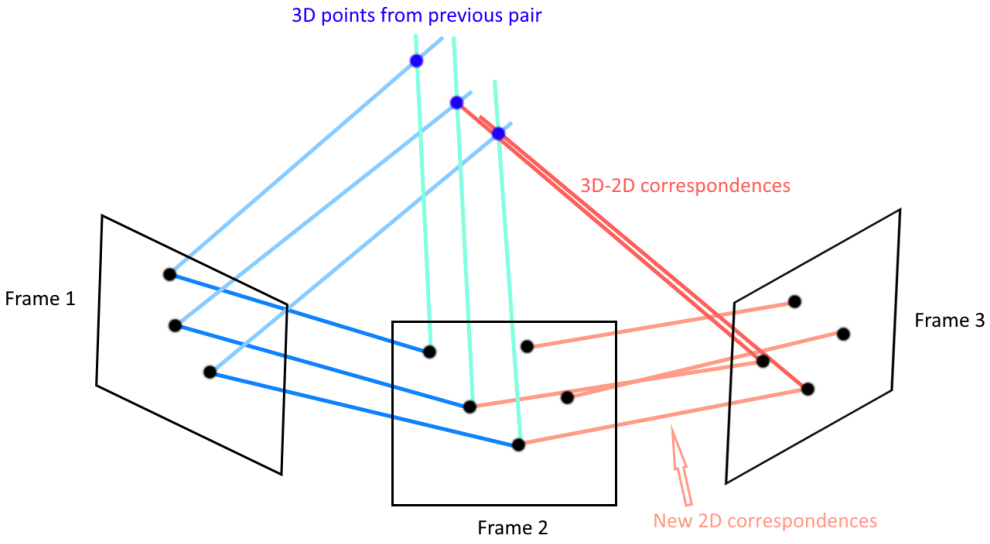


This simple way was suitable but not accurate enough for long chain of frames because it gave only translation vectors with unit length that may not reflect the actual movement.

2.1.3.2. Further frames' pairs

Due to reasons described above we decided to use the perspective-n-point pose computation algorithm.

1. 3D-2D point's correspondences are obtained from the previous estimations:



2. These correspondences are passed to `solvePnPRansac` function which returns translation between frame 2 and 3

2.1.4. 3D points triangulation

Knowing corresponding 2D points from two frames and translation between them we can place these points in 3D space using simple linear triangulation method described in section III.E of our main article or in chapter 12.2 (pg. 312) of book "Multiple View Geometry in Computer Vision" by Richard Hartley and Andrew Zisserman.

OpenCV has a built-in function for this purpose but we decided to rewrite it to remove some redundant checks and get a more effective version.

2.1.5. Additional accuracy optimizations

For bad matching/transition estimation cases the algorithm must work as accurately as possible. We implemented 2 decisions to solve this problem

2.1.5.1. Restartable main cycle for robustness

This is the step-by-step description of the main loop also named "main module":

1) Initialization of necessary data structures with zero global position.
2) Processing of first pair frames. Here we get the spatial point data from the first pair of frames.
3) Cycling processing of the new frames:

a) Filling batch frames to the specified size. Only frames with sufficient number of keypoints are batch-filled, the rest are skipped. We took the best frame from this batch. The rest of the frames from the batch are not used.
b) Identify which three-dimensional points from those calculated for previous frames correspond to the features for the new frame.
c) Calculating the camera rotation matrix and motion vector between the last two frames.
d) Reconstruct 3D points by triangulation using calibration, rotations, transitions and corresponding points in 2 vectors.
e) Push new spatial points of the current frame into the global data structure.

In situations where there is no frame in the new batch of images that matches well with the previous frames, the main loop interrupts and restarts with the refined initial global position so new 3D points will be placed correctly enough.

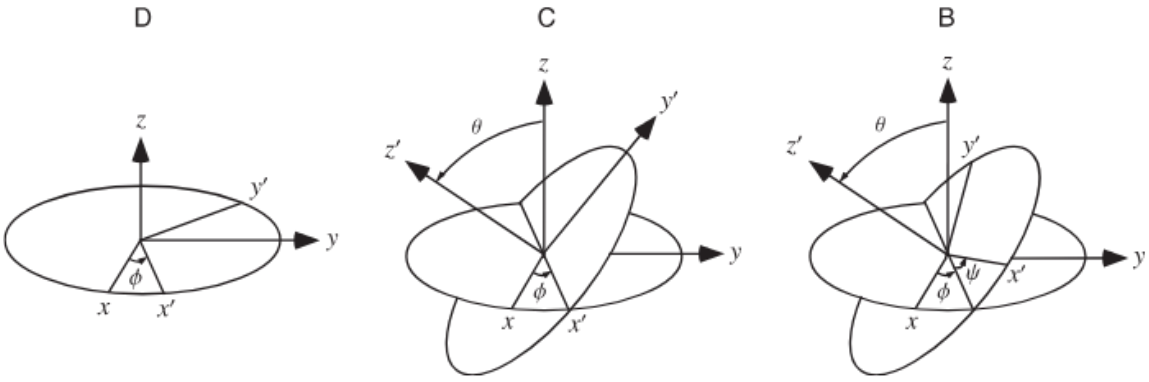
2.1.5.2. Bundle adjustment

When we have a set of frames with translations, corresponding points and its 3D points projections we can refine translations and spatial points using maximum likelihood estimator, e. g. bundle adjustment.

For implementation an additional library was used, Ceres solver. Theoretically, this step should give us improved accuracy. But, unfortunately, selection of the loss function and its parameters turned out to be too difficult so the accuracy was getting worse. Maybe some automated selection algorithm will be able to find the best function and parameters, but we actually don't know how to define the loss function.

2.2. Visualization

To see the results of main program computations, we've added a 3D visualization module, which mostly relies on opencv's built-in viz submodule. The function of our module is a 3d visualization of triangulated points. The main improvement we added to the basic viz is an opportunity of movement. To achieve this, we use the camera's Affine matrix, estimate the euler angles and move the camera towards, judging by the current viewpoint.



3.1.2.1. Utilities evolution

- Pybullet:

This Python engine allowed user to visualize points, scale them and give us a free will of movement, but it turned out to be a bad decision for our project due to the low performance

- Open3D

Also a python module, has a very good performance, especially with the point clouds, and has built-in tools to “mesh” the clouds. But we put off this solution, due to points scale problem: our points had too small coordinates to get an appropriate result

- Viz

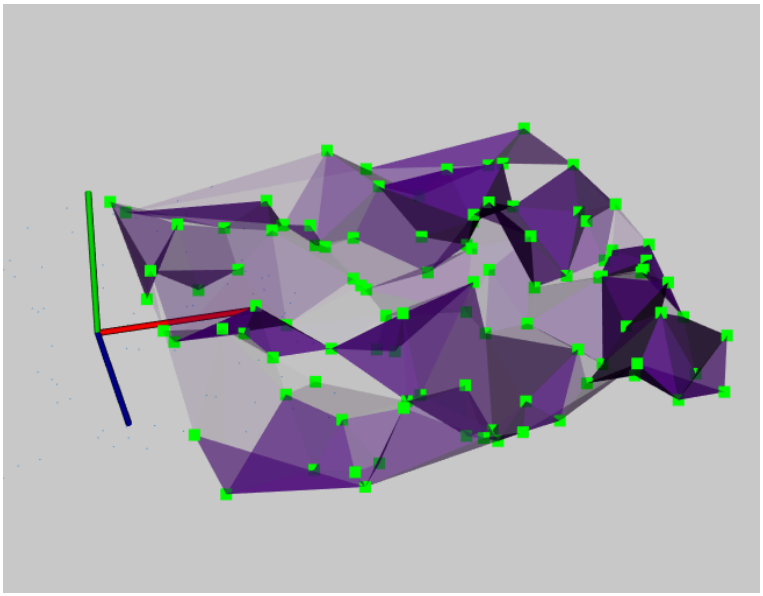
OpenCV’s built-in module,so we don’t need to think about compatibility. Also it gives us an opportunity to add colors for points and good performance, but it doesn’t have any movement except rotation around certain points and also there is no tool for “meshing” the point cloud.

3.1.2.2. Triangulation

Another visualization utility module consisting of three submodules:

1. First one finds the ‘best fitting plane’ for a set of 3d points using OpenCV’s built-in SVD. ‘Best fitting plane’ is a plane with the least sum of squared distance to each point.
2. Second projects given 3d points on an acquired best fitting plane.
3. 2D Delaunay triangulation module, which uses Bowyer-Watson’s triangulation algorithm described in “Rebay, S. Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm.” article.

Using these 3 submodules as one, we can build a mesh on a given 3d points



3. Solution description

3.1. Modules and subsystems

By this link you can observe the comprehensive diagram of our modules and their communications. Next, we will describe some of the components very briefly, because the previous part, together with the diagram, gives an exhaustive description

3.1.1. Indoor SLAM

Main algorithm needs to have calibration and JSON configuration, then, main cycle can be launched.

3.1.1.1. Feature extraction

Algorithm loads frames and detects key points by significant visual parameters.

3.1.1.2. Feature matching

Algorithm compares sets of key points extracted from two frames and tries to find correspondences

3.1.1.3. Translation estimation

Corresponding points and camera calibration matrix are used for estimation of rotation and translation between frames

3.1.1.4. 3D points triangulation

When transtation data is obtained it will be used to transform 2D points to 3D..

3.1.1.5. Bundle adjustment

When points and transitions from several frames are obtained additional accuracy optimization named bundle adjustment can be done

3.1.1.6. Data saving

Final 3D points are passed to global data structure and saved to files

3.1.1.7. Iteration end

Main cycle processes the next batch of frames or breaks weather frames are over.

3.1.2. Visualization

This module can be launched independently or after the main cycle. In the second case data loaded to the global structure from files. When global structure with points and additional data is available visual triangulation can be performed. After that, user can observe points or triangles.

3.2. Deployment

1. Install additional dependencies and tools (**located outside of project's directory**):

```
sudo apt update
sudo apt install cmake libtbb2 g++ wget unzip ffmpeg libgtk2.0-dev
libavformat-dev libavcodec-dev libavutil-dev libswscale-dev libtbb-dev
libjpeg-dev libpng-dev libtiff-dev
sudo apt install libvtk7-dev
sudo apt install build-essential
sudo apt install cmake git libgtk2.0-dev pkg-config libavcodec-dev
libavformat-dev libswscale-dev
sudo apt install libeigen3-dev libgflags-dev libgoogle-glog-dev
libatlas-base-dev libsuitesparse-
```

2. Install Ceres-solver

```
sudo apt-get install libeigen3-dev libgflags-dev libgoogle-glog-dev

wget http://ceres-solver.org/ceres-solver-2.2.0.tar.gz
tar zxf ceres-solver-2.2.0.tar.gz
mkdir ceres-bin
cd ceres-bin
cmake ../ceres-solver-2.2.0

make -j3
make test
sudo make install

# Run next command to test ceres
./bin/simple_bundle_adjuster ../ceres-solver-2.2.0/data/problem-16-22106-pre.txt
```

3. *(Optional)* Install CUDA packages:

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-keyrin
g_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt-get update
sudo apt-get -y install cuda-toolkit-12-3
```

4. Build OpenCV:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=../opencv_contrib-4.8.0/modules/ \
-D BUILD_SHARED_LIBS=ON \
-D BUILD_opencv_sfm=ON \
-D OPENCV_ENABLE_NONFREE=ON \
-D BUILD_SHARED_LIBS=ON \
-D BUILD_TESTS=ON \
-D OPENCV_GENERATE_PKGCONFIG=ON \
```

```
-D BUILD_EXAMPLES=ON \
-D WITH_QT=ON \
-D WITH_GTK=ON \
-D WITH_OPENGL=ON \
-D WITH_FFMPEG=ON \
-D WITH_TBB=ON \
-D WITH_V4L=ON \
-D WITH_VTK=ON \
../opencv-4.8.0/
# Make sure FFMPEG and its modules marked "YES"
```

a. For CUDA version use this build options (Specify version for **`-D CUDA_ARCH_BIN=<version>`** regarding to your hardware using this site):

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=../opencv_contrib-4.8.0/modules/ \
-D BUILD_SHARED_LIBS=ON \
-D BUILD_opencv_sfm=ON \
-D OPENCV_ENABLE_NONFREE=ON \
-D BUILD_SHARED_LIBS=ON \
-D BUILD_TESTS=ON \
-D OPENCV_GENERATE_PKGCONFIG=ON \
-D BUILD_EXAMPLES=ON \
-D WITH_QT=ON \
-D WITH_GTK=ON \
-D WITH_OPENGL=ON \
-D WITH_FFMPEG=ON \
-D WITH_TBB=ON \
-D WITH_V4L=ON \
-D WITH_VTK=ON \
-D ENABLE_FAST_MATH=1 \
-D CUDA_FAST_MATH=1 \
-D WITH_CUBLAS=1 \
-D WITH_CUDA=ON \
-D BUILD_opencv_cudacodec=OFF \
-D WITH_CUDNN=ON \
-D OPENCV_DNN_CUDA=OFF \
-D CUDA_ARCH_BIN=<version> \
-D WITH_GSTREAMER=ON \
../opencv-4.8.0/

# Make sure FFMPEG, its and CUDA (not cuDNN) modules marked "YES"
```

5. Install built files:

```
make -j8  # Number of jobs can be specified
sudo make install
```

6. Install nlohmann-json

a. **`sudo apt install nlohmann-json3-dev`**
b. Create JSON file with configurations' parameters (nlohmann supports comments in JSON):

```
{
  "onlyViz": true,

  "calibrate": false,
  "visualCalibration": true,
  "calibrationPath": "./config/samsung-hv-2.xml",

  "usePhotosCycle": false,
  //  "photosPathPattern":
"/mnt/c/Users/bakug/YandexDisk/NSU/private/2-1/PAK/static/photos/samsung-room-new/_
(*).JPG",
  "photosPathPattern":
"/mnt/c/Users/bakug/YandexDisk/NSU/private/2-1/PAK/static/photos/samsung-4to3-tumbochk
a/*.JPG",
  // "videoSourcePath" has an effect when "usePhotosCycle" is false
  "videoSourcePath":
"/mnt/c/Users/bakug/YandexDisk/NSU/private/2-1/PAK/static/samsung-new-tumbochka-fhd.MP
4",

  // If "onlyViz" is true program searches files with data in this folder
  "outputDataDir": "./data/video_report/table-photos-CPU",

  "threadsCount": 1,

  "useUndistortion": false,

  "requiredExtractedPointsCount": 10000,
  "featureExtractingThreshold": 1,

  "framesBatchSize": 210,
  "skipFramesFromBatchHead": 0,
  "useFirstFitInBatch": true,

  "requiredMatchedPointsCount": 500,

  // This block has an effect when "useFeatureTracker" is false
  "useFM-SIFT-FLANN": true, // NORM_L2 in CUDA
  "useFM-SIFT-BF": false, // NORM_L1 in CUDA
  "useFM-ORB": false, // NORM_HAMMING in CUDA

  "knnMatcherDistance": 0.7,

  // Now this parameters are used only for the first pair of frames
  "RPUseRANSAC": true,
  "RPRANSACProb": 0.999,
```

```
  "RPRANSACThreshold": 5.0,
  "RPDistanceThreshold": 200.0,


  "useBundleAdjustment": false,
  // Next parameter also specifes max frames datas cnt which will be uploaded to
global data at the one moment
  "BAMaxFramesCnt": 8,
  "BAThreadsCnt": 12,

  // Priority of loss functions' flags is from top to bottom
  "BAUseTrivialLossFunction": false,

  "BAUseHuberLossFunction": true,
  "BAHuberLossFunctionParameter": 4.0,

  "BAUseCauchyLossFunction": false,
  "BACauchyLossFunctionParameter": 4.0,

  "BAUseArctanLossFunction": false,
  "BAArctanLossFunctionParameter": 2.0,

  "BAUseTukeyLossFunction": false,
  "BATukeyLossFunctionParameter": 4.0
}
```

7. Now you can build and execute our project:
    a. Go to project's directory
    b. Build a project (one of variants):
        i. **`cmake . -B build`**
        ii. **`cmake . -B build -D USE_CUDA=YES`**
    c. Collect binary file - **`make -C build -j8`**
    d. So now you can execute **`./build/slam-indoor-code <path/to/config.json>`**

4. Key architectural elements

4.1. Configuration module

Almost all components of the main module use a lot of static defined parameters (paths, threshold for extractor, required count of extracted and matched points etc.) so we developed specific convenient configuration module to obtain any of them everywhere using the nlohmann-json library.

1. Configuration system consists of producer class and header with config field's enum and map where every enumeration corresponds to type and JSON field
2. When program starts it loads specified JSON and checks existence and type's correctness

for all fields described in config header

3. If there are no errors we can call producer's method in other modules with enum value to get configuration's parameters

4.2. Data structures for main cycle

In the main loop operation, the following data structures are used for easy data transfer to other modules.

4.2.1. Media sources structure

This structure is designed to handle both video and photo data sources, allowing the same functions to process either type of media with no need of overriding.

4.2.2. Structure containing conditions for data processing

This structure is needed to determine distortion coefficients of camera, size of frame's batch, count of frames in the batch's head which won't be checked, threshold for feature extraction, required number of extracted points in frame, type of descriptor extractor and matcher, required number of matched points - in general, parameters loaded from configuration module and their derivatives

4.2.3. Structure containing specific frame data

In general, this structure provides the information needed to match objects, define geometric transformations between frames and visualize the point cloud.

4.2.4. Structure for storing main module's results

This structure stores the data that will be received after processing of all significant frames. All data from this structure will be used for visualization

5. Platform

Operating Systems: Ubuntu 22, WSL2

Packages' list can be observed in deployment section

Language: C++11 (compiler g++)

Libraries: OpenCV 4.8 with contrib modules, Ceres-solver 2.2.0, nlohmann-json

For CUDA: GPU developed by NVIDIA (you can see all list here)