
Roboy Vision Module Documentation

Release 0.0.

Sep 22, 2017

Usage and Installation

1	What Roboy Vision can do:
----------	----------------------------------

3

This project's goal is to provide Roboy with extensive vision capabilities. This means to recognize, localize and classify objects in its environment as well as to provide data for localization to be processed by other modules. The input will be a realsense camera device, the output should be high-level data about Roboy's environment provided using ROS messages and services.

The most import task in Vision for human interaction is to detect and recognize faces, which is why this was considered the highest priority of this project. The current main tasks of this project are:

- Identification of Roboy Team Members
- Pose estimation of a detected face and Roboy Motor Control
- Tracking of detected objects
- Person Talking detection
- Mood Recognition
- Gender Recognition
- Remebering faces online
- Age classification
- Scene and object classification

What Roboy Vision can do:

- Face detection.
- Speaker detection.
- Object detection.
- Multitracking.

Relevant Background Information and Pre-Requisites

Our approach to tackle the given tasks in Vision is to use machine learning methods. Therefore a basic understanding of machine learning, specifically also deep Neural Networks and Convolutional Neural Networks will be necessary.

The following links are to be seen as suggestions for getting started on machine learning:

- Crash Course on Deep Learning in the form of Youtube tutorials: [DeepLearning.tv](#)
- Closer Look at the implementation of Neural Networks: [The Foundations of deep learning](#)
- An introduction to Convolutional Neural Networks (CNNs): [Deep learning in Computer vision](#)
- The machine learning framework used for implementation: [Tensorflow](#)
- Furthermore a basic understanding of simple machine learning approaches like Regression, Tree Learning, K-Nearest-Neighbours (KNN), Support Vector Machines (SVMs), Gaussian Models, Eigenfaces, etc. will be helpful.

The papers currently used for implementation should be understood:

- [Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks](#)
- [FaceNet: A Unified Embedding for Face Recognition and Clustering](#)
- [DLIB: Facial landmarks and face recognition](#)
- ‘You Only Look Once: Unified, Real-Time Object Detection <<https://pjreddie.com/media/files/papers/yolo.pdf>>’_

Furthermore there are plans to extend the implementation using this paper:

- [An All-In-One Convolutional Neural Network for Face Analysis](#)

Contents

Installation

Running all sub modules in realtime requires Ubuntu 16.04 with Kernel version 4.4.x. For getting started a jupyter notebook installation using anaconda will be sufficient. Tutorials in form of jupyter notebooks are provided.

Anaconda

We recommend the use of Anaconda. This allows all python libraries to only be installed in a virtual environment which then won't have any influence on other programs you are running. We will create a virtual environment using python 3.

- Download Anaconda from <https://www.continuum.io/downloads#linux>:
- Install Anaconda:

```
bash ~/Downloads/Anaconda3-4.3.0-Linux-x86_64.sh
```

- Enter 'yes' when prompted with the following question:

```
Do you wish the installer to prepend the Anaconda install location to PATH in your
/home/name/.bashrc ? [yes|no]
```

- Restart the terminal.
- Create a Conda Environment with the name "roboy" and python 3:

```
conda create --name roboy python=3
```

- To work on the created environment it has to be activated:

```
source activate roboy
```

- When you want to leave the environment you have to use:

```
source deactivate
```

Dependencies

Now you should be working in your virtual environment. We then will install all requirements. We are working with python 3, because of tensorflow requirements.

- First clone the Vision repository and run the setup script to install most of the necessary dependencies:

```
cd ~/
git clone https://github.com/Roboy/Vision

cd ~/Vision
chmod +x setup.sh
sudo ./setup.sh
```


- Download Cuda from <https://developer.nvidia.com/cuda-downloads>
- Install Cuda with instructions from <http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#axzz4rHIEa0GY>

Build

To build doxygen documentation offline for viewing you can run:

```
cd ~/Vision
sphinx-build -b html ./docs ./build/docs
```

Please download the files

- SharedLibs
- StaticLibs

from <https://drive.google.com/drive/folders/0B0cOyLVrawK5TFJhdGJvNE9wNzg>

Compiling opencv from source (MacOS)

Note: This is required only if you want to work with multiple object/face tracking. This step is needed as this Multiple tracking is part of opencv_contrib module which needs to be compiled along with opencv, as it doesn't get shipped with opencv. The following instructions are for Mac.

First you will need:

1. Mac OSX 10.12
2. XCode
3. Command Line Tools (This is done from inside XCode)
4. CMake(<http://www.cmake.org/download/>)

Step 1: Download openCV and unzip it somewhere on your computer. Create two new folders inside of the openCV directory, one called StaticLibs and the other SharedLibs.

Step 2a: Build the Static Libraries with Terminal. To build the libraries in Terminal.

- Open CMake.
- Click Browse Source and navigate to your openCV folder.
- Click Browse Build and navigate to your StaticLib Folder.
- Click the configure button. You will be asked how you would like to generate the files. Choose Unix-Makefile from the Drop Down menu and Click OK. CMake will perform some tests and return a set of red boxes appear in the CMake Window.

You will need to uncheck and add to the following options.

- Uncheck BUILD_SHARED_LIBS
- Uncheck BUILD_TESTS
- Add an SDK path to CMAKE_OSX_SYSROOT, it will look something like this “/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.9.sdk”. (NOTE: make sure your version of SDK is used here)
- Add x86_64 to CMAKE_OSX_ARCHITECTURES, this tells it to compile against the current system

- Uncheck WITH_1394
- Uncheck WITH_FFMPEG

Click Configure again, then Click Generate.

When the application has finished generating, Open Terminal and type the following commands.

- `cd <path/to/your/opencv/staticlibs/folder/>`
- `make` (This will take awhile)
- `sudo make install`

Enter your password. This will install the static libraries on your computer.

Step 2c: Build the Shared Libraries with Terminal.

- Open CMake.
- Click Browse Source and navigate to your openCV folder.
- Click Browse Build and navigate to your SharedLib Folder.
- Click the configure button. You will be asked how you would like to generate the files. Choose Unix-Makefile from the Drop Down menu and Click OK. CMake will perform some tests and return a set of red boxes appear in the CMake Window.

You will need to uncheck and add to the following options.

- Check BUILD_SHARED_LIBS
- Uncheck BUILD_TESTS
- Add an SDK path to CMAKE_OSX_SYSROOT, it will look something like this “/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.9.sdk”.
- Add x86_64 to CMAKE_OSX_ARCHITECTURES, this tells it to compile against the current system
- Uncheck WITH_1394
- Uncheck WITH_FFMPEG
- Click Configure again, then Click Generate.

When the application has finished generating, Open Terminal.

- `cd <path/to/your/opencv/SharedLibs/folder/>`
- `make` (This will take awhile)
- `sudo make install`

You should see the libraries build in the shared and static libraries folders.

- `cd /Users/<Username>/<path-to-installation>/StaticLibs/lib/python3`
- `ls -s cv2.cpython-36m-darwin.so cv2.so`

The above step would help in creating a symbolic link so you can use it with python.

Compiling opencv from source (Linux)

You will need the following packages:

- GCC 4.4.x or later
- CMake 2.6 or higher

- Git
- GTK+2.x or higher, including headers (libgtk2.0-dev)
- pkg-config
- Python 2.6 or later and Numpy 1.5 or later with developer packages (python-dev, python-numpy)
- ffmpeg or libav development packages: libavcodec-dev, libavformat-dev, libswscale-dev
- [optional] libtbb2 libtbb-dev
- [optional] libdc1394 2.x
- [optional] libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev

Step 1: The packages can be installed using Terminal as follows:

```
[compiler] sudo apt-get install build-essential
```

```
[required] sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
```

```
[optional] sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

Step 2: Get the latest stable version of OpenCV from <https://sourceforge.net/projects/opencvlibrary/>

2a: Download the source tarball and unpack it.

2b: In terminal, cd into the working directory followed by cloning the OpenCV repository:

```
cd ~/<my_working_directory>

git clone https://github.com/opencv/opencv.git
```

Step 3: Building OpenCV from source using CMake:

3a: Create a temporary directory, here denoted as <cmake_binary_dir>, where you want to put the generated Makefiles, project files as well the object files and output binaries.

3b: Enter the <cmake_binary_dir> and type:

```
cmake <path to the OpenCV source directory>
```

Step 4: Enter the created temporary directory (<cmake_binary_dir>) and proceed with:

```
make

sudo make install
```

Running the Vision module

Finally, to run the entire module, run the 'RoboyVision.py' script in the 'src' folder:

```
python RoboyVision.py
```

Getting started

Tutorials

As a start run the jupyter notebooks in tutorials section:

```
source activate roboy
cd ~/Vision/tutorials
jupyter notebook
```

There is four different tutorials:

- **Face_Detection** tutorial will show you how to run a face detection on an image or webcam input using the MTCNN neural network. Additionally also DLib face detector is used.
- **Facenet_Embeddings** tutorial shows how to calculate the 128D embeddings given a face using facenet. There exist 2 versions of this tutorial. One is using MTCNN for face detection, the other one using DLib. This tutorial provides the functionality to calculate and save embeddings on a database of pictures, where all pictures are stored in a folder structure, with the folder name being the person in the picture.
- **Classifier_Training** uses embeddings calculated in the previous tutorial to train a classifier to distinguish between the classes in these embeddings. Currently only SVM and a binary Tree have been implemented.
- **Face_Recognition** tutorial shows how to run the classification on an image or webcam input. It demonstrates this using KNN and the classifiers trained in the previous tutorial.

Real Time - How to run Vision module.

For running face detection in real time you can run the script:

```
source activate roboy
cd ~/Vision/src
python RoboyVision.py
```

Context

The Vision package receives ZED stereo camera input which is then processed internally. Other than that, it should also receive data from Roboy motor control about Roboy's current position. This can later be used to calculate the relative and absolute positions of detected objects.

The main output of the Vision module will be detected objects and some object properties (e.g. detected face, name of person, mood, gender, ...). The receiver for this will be Roboy's Dialog and Memory modules. This is illustrated in the following context overview:

Solution Strategy

Basic decisions for Vision Package:

- Separation of different tasks into sub-modules (Face Detection, Object Detection, Object tracking, face recognition, mood recognition, age estimation, scene classification, ...)
- Highest priority on face detection and face pose estimation. Recognition of people as second priority. All other properties concerning people with lower priority and general object detection with least importance.
- Face detection using this approach: [Joint Face Detection and Alignment using MTCNNs](#). Good real time performance, other modules to be built on top.
- Face embeddings using [FaceNet](#). These embeddings can be used for recognition.

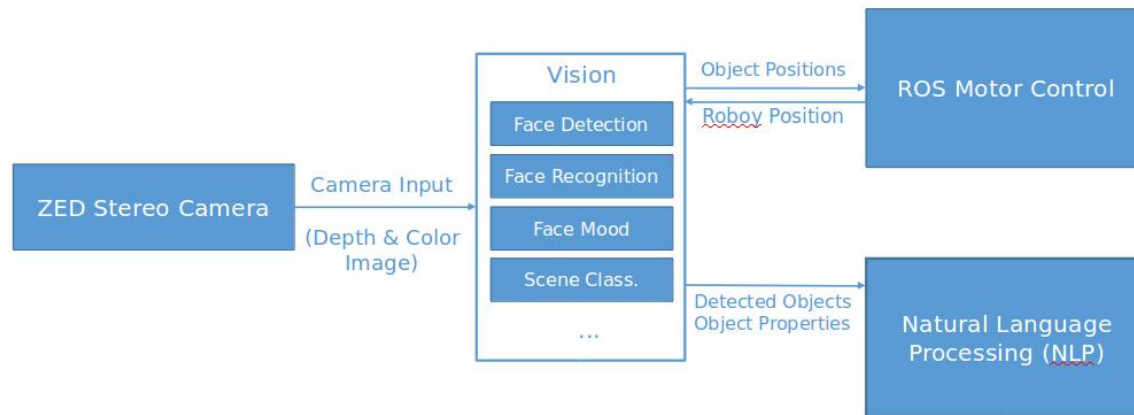


Fig. 1.1: **Context diagram** - shows the birds eye view of the vision system (black box) described by this architecture within the ecosystem it is to be placed in. Shows orbit level interfaces on the user interaction and component scope.

- Speaker detection using facial landmarks from [DLIB](#)
- Object recognition using [YOLO](#)

Current implementation:

- **RoboyVision as main, handling all sub-modules:**
 - **Face Detection** using Facenet for calculating embeddings for a given face and SVM for classification. SVM currently trained on pictures of LFW (labelled Faces in the Wild) dataset, using Roboy Team members as next step. Sends coordinates to **Tracker** and facial landmarks to **Speaker Detection**
 - **Speaker Detection** using DLIB's facial landmarks to calculate specific mouth parameters (width, lip distance) of each face to determine, whether a person is speaking
 - **ROS services** are handled by RoboyVision via websocket
 - **Object recognition** is implemented based on YOLO
 - **Tracking objects/faces** running in realtime. This implementation is based on the MIL(Visual Tracking with Online Multiple Instance Learning). Also part of the Opencv_contrib module.

Read about the Public Interfaces (ROS) [here](#)

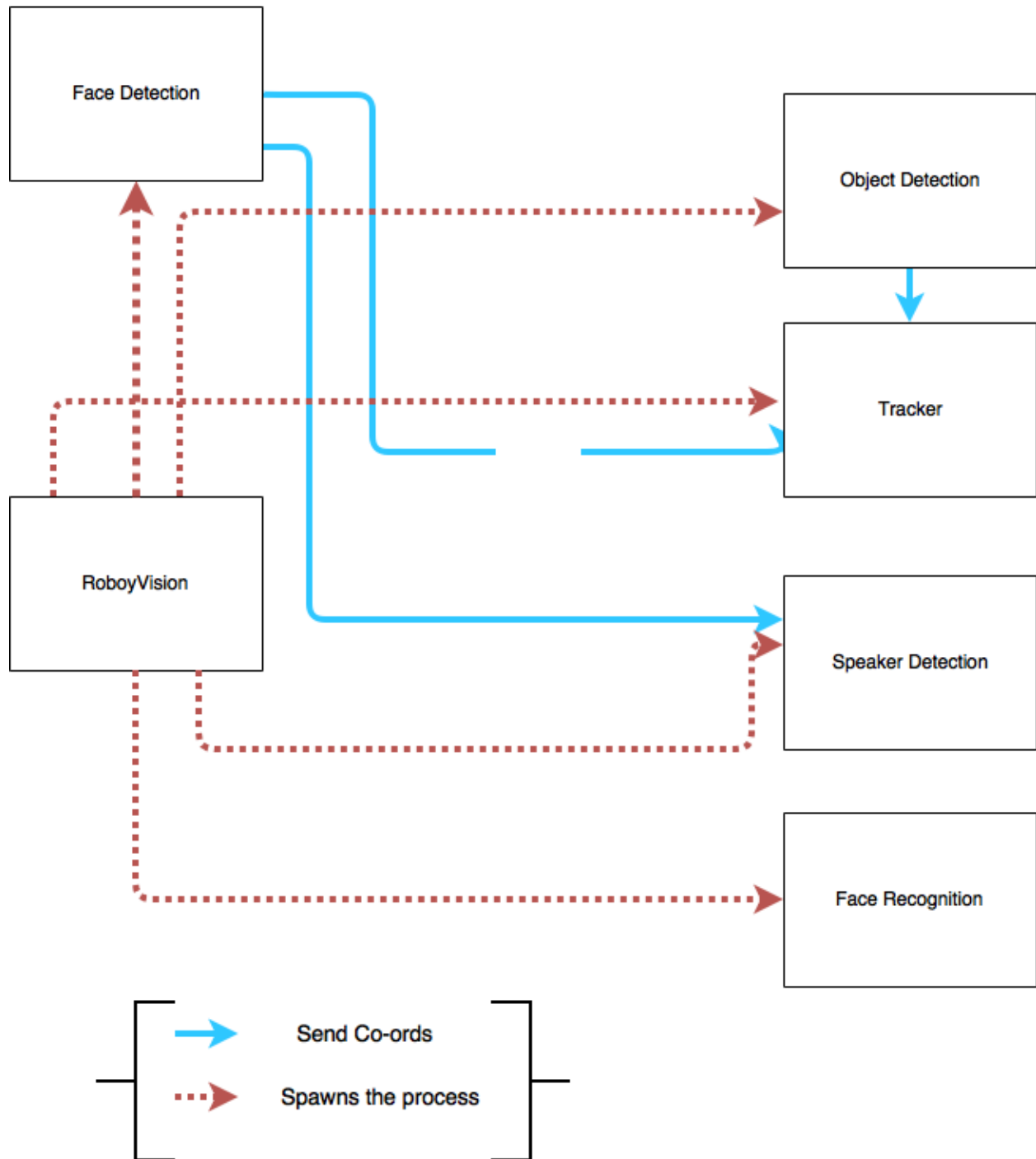
Plan for this semester with priorities in red (5 being highest priority):

Future plans on current implementation: * Improve tracking by implementing the GOTURN algorithm.

Architecture of the current System:



Vision Architecture



Public Interfaces(ROS)

Interfaces to other modules will be realized using ROS Communication. Currently 5 interfaces have been designed for communication, although not all have been fully implemented due to the not fully assembled Roboy to test it on. Due to version clashes and Rospy not being available for Python 3, the ROS Communication is implemented using websocket.

- **FaceCoordinates message:** For each recognized face in the current frame, this message publishes the id, a boolean indicating whether the person is speaking and the 3D position (depth from ZED camera still to be implemented):

```
# returns: int32 id, bool speaking, float32 x, float32 y, float32 z  
  
rostopic echo /roboy/cognition/vision/FaceCoordinates
```

- **NewFacialFeatures message:** For each unrecognized face in the current frame, this message publishes the facial features as a 128-dimensional vector and a boolean indicating whether the person is speaking:

```
# returns: bool speaking, float64[128] ff  
  
rostopic echo /roboy/cognition/vision/NewFacialFeatures
```

- **DescribeScene service:** Service called to list all objects detected in the current frame, ordered from left to right:

```
# argument:  
# returns: String[] objects_detected  
  
rosservice call /roboy/cognition/vision/DescribeScnee
```

- **FindObject service:** Service called to find an object in the current frame. Given an object type, the position of the object is returned:

```
# argument: String type  
# returns: bool found, float32 x, float32 y, float32 z  
  
rosservice call /roboy/cognition/vision/FindObject *type*
```

- **LookAtSpeaker service:** Service called to turn towards to closest speaking face. Due to Roboy not being assembled yet, this service hasn't been implemented yet:

```
# argument:  
# returns: bool turned  
  
rosservice call /roboy/cognition/vision/LookAtSpeaker
```

A more detailed documentation of the ROS Communication as well as future plans can be viewed here: <https://devanthro.atlassian.net/wiki/spaces/HT/pages/76402960/ROS+services+messages+overview>

Architecture Constraints

Hardware Constraints

Operating System Constraints

Programming Constraints

Constraint Name	Description
Python 3	Tensorflow v1.0 required for facenet implementation which uses Python 3
Python 2	ROS requires Python 2 still

Conventions

We follow the coding guidelines:

Lan- guage	Guideline	Tools
Python	https://www.python.org/dev/peps/pep-0008/	pep-format: https://github.com/google/yapf
C++	http://wiki.ros.org/CppStyleGuide	clang-format: https://github.com/davetcoleman/roscpp_code_format

Libraries and external Software

Name	URL/Author	License	Description
MTCNN face detection & alignment	https://github.com/kpzhang93/MTCNN_face_detection_alignment	MIT license	Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Neural Networks
Facenet	https://github.com/davidsandberg/facenet	MIT License	Face recognition using Tensorflow
Object Detection	https://github.com/pjreddie/darknet	GNU General Public License	Object Detection Using YOLO9000
arc42	http://www.arc42.de/template/	Creative Commons Attribution license.	Template for documenting and developing software
DLIB	https://github.com/davisking/dlib	Boost Software License	ML toolkit, mainly used for face detection and facial landmarks
YOLO	https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection		Real-time object recognition system

Presentations

This section lists presentations of Roboy Vision Team.

Midterm/ Final Presentations

- [Midterm Presentation WS 2016/17](#)
- [Final Presentation WS 2016/17](#)
- [Midterm Presentation SS 2017](#)
- [Final Presentation SS 2017](#)

Other Presentations

- [RoadMap 2017](#)

API

struct `augment_args`

Public Members

`int w`

`int h`

`float scale`

`float rad`

`float dx`

`float dy`

`float aspect`

class `BOX`

Inherits from Structure

Private Static Attributes

```
list ObjectRecognition.BOX._fields_ = [("x", c_float), ("y", c_float), ("w", c_float), ("h", c_float)]
```

struct `box`

Public Members

`float x`

`float y`

`float w`

`float h`

class `BOX`

Inherits from Structure

Private Static Attributes

```
list FaceDetect.BOX._fields_ = [("x", c_float), ("y", c_float), ("w", c_float), ("h", c_float)]
```

struct `box_label`

Public Members

int **id**
float **x**
float **y**
float **w**
float **h**
float **left**
float **right**
float **top**
float **bottom**

struct CalibrationParameters

#include <Core.hpp> Intrinsic parameters of each cameras and extrinsic (translation and rotation).

Note The calibration/rectification process, called during *sl::Camera::open*, is using the raw parameters defined in the SNXXX.conf file, where XXX is the ZED Serial Number. Those values may be adjusted or not by the Self-Calibration to get a proper image alignment. After *sl::Camera::open* is done (with or without Self-Calibration activated) success, most of the stereo parameters (except Baseline of course) should be 0 or very close to 0. It means that images after rectification process (given by *retrieveImage()*) are aligned as if they were taken by a “perfect” stereo camera, defined by the new *CalibrationParameters*.

Public Members

sl::float3 **R**
Rotation (using Rodrigues’ transformation) between the two sensors. Defined as ‘tilt’, ‘convergence’ and ‘roll’.

sl::float3 **T**
Translation between the two sensors. T.x is the distance between the two cameras (baseline) in the *sl::UNIT* chosen during *sl::Camera::open* (mm, cm, meters, inches...).

CameraParameters **left_cam**
Intrinsic parameters of the left camera

CameraParameters **right_cam**
Intrinsic parameters of the right camera

class Camera

#include <Camera.hpp> The main class to use the ZED camera.

Camera infos

void **setSVOPosition** (int *frame_number*)
Sets the position of the SVO file to a desired frame.

Note Works only if the camera is open in SVO playback mode.

Parameters

- *frame_number*: : the number of the desired frame to be decoded.

int **getSVOPosition** ()

Returns the current position of the SVO file.

Return The current position in the SVO file as int (-1 if the SDK is not reading a SVO).

Note Works only if the camera is open in SVO reading mode.

int **getSVONumberOfFrames** ()

Returns the number of frames in the SVO file.

Return The total number of frames in the SVO file (-1 if the SDK is not reading a SVO).

Note Works only if the camera is open in SVO reading mode.

void **setCameraSettings** (*CAMERA_SETTINGS* settings, int value, bool use_default = false)

Sets the value to the corresponding sl::CAMERA_SETTINGS (Gain, brightness, hue, exposure...).

Note Works only if the camera is open in live mode.

Parameters

- settings: : enum for the control mode.
- value: : value to set for the corresponding control.
- use_default: : will set default (or automatic) value if set to true (value (int) will not be taken into account).

int **getCameraSettings** (*CAMERA_SETTINGS* setting)

Returns the current value to the corresponding sl::CAMERA_SETTINGS (Gain, brightness, hue, exposure...).

Return The current value for the corresponding control (-1 if something wrong happened).

Note Works only if the camera is open in live mode.

Parameters

- setting: : enum for the control mode.

float **getCameraFPS** ()

Returns the current FPS of the camera.

Return The current FPS (or recorded FPS for SVO). Return -1.f if something goes wrong.

void **setCameraFPS** (int *desired_fps*)

Sets a new frame rate for the camera, or the closest available frame rate.

Note Works only if the camera is open in live mode.

Parameters

- desired_fps: : the new desired frame rate.

float **getCurrentFPS** ()

Returns the current FPS of the application/callback. It is based on the difference of camera timestamps between two successful *grab()*.

Return The current FPS of the application (if grab leads the application) or callback (if ZED is called in a thread)

unsigned long long **getCameraTimestamp** ()

Returns the timestamp at the time the frame has been extracted from USB stream. (should be called after a *grab()*).

Return The timestamp of the frame grab in ns. -1 if not available (SVO file without compression).

Note SVO file from SDK 1.0.0 (with compression) contains the camera timestamp for each frame.

unsigned long long **getCurrentTimestamp** ()

Returns the current timestamp at the time the function is called. Can be compared to the camera *sl::Camera::getCameraTimestamp* for synchronization. Use this function to compare the current timestamp and the camera timestamp, since they have the same reference (Computer start time).

Return The current timestamp in ns.

unsigned int **getFrameDroppedCount** ()

Returns the number of frame dropped since *sl::Camera::grab* has been called for the first time. Based on camera timestamp and FPS comparison.

Return The number of frame dropped since first *sl::Camera::grab* call.

CameraInformation **getCameraInformation** ()

Returns camera informations (calibration parameters, serial number and current firmware version).

Return *CameraInformation* containing the calibration parameters of the ZED, as well as serial number and firmware version It also returns the ZED Serial Number (as uint) (Live or SVO) and the ZED Firmware version (as uint), 0 if the ZED is not connected.

Self calibration

SELF_CALIBRATION_STATE **getSelfCalibrationState** ()

Returns the current status of the self-calibration.

Return A status code given informations about the self calibration status. For more details see *sl::SELF_CALIBRATION_STATE*.

void **resetSelfCalibration** ()

Resets the self camera calibration. This function can be called at any time AFTER the *sl::Camera::open* function has been called. It will reset and calculate again correction for misalignment, convergence and color mismatch. It can be called after changing camera parameters without needing to restart your executable.

if no problem was encountered, the camera will use new parameters. Otherwise, it will be the old ones.

Tracking

ERROR_CODE **enableTracking** (*TrackingParameters* tracking_params = TrackingParameters ())

Initializes and start the tracking processes.

Return `sl::ERROR_CODE_FAILURE` if the `sl::TrackingParameters::area_file_path` file wasn't found, `sl::SUCCESS` otherwise.

Warning The area localization is a beta feature, the behavior might change in the future.

Parameters

- `tracking_params`: Structure of `sl::TrackingParameters`, which defines specific parameters for tracking. default : Leave it empty to get best default parameters or create your own structure to change tracking parameters according to `sl::TrackingParameters` documentation.

`sl::TRACKING_STATE getPosition (sl::Pose &camera_pose, REFERENCE_FRAME reference_frame = sl::REFERENCE_FRAME_WORLD)`

Fills the position of the camera frame in the world frame and return the current state of the Tracker.

Extract *Rotation* Matrix : `camera_pose.getRotation()`; Extract *Translation* Vector: `camera_pose.getTranslation()`; Convert to *Orientation* / quaternion : `camera_pose.getOrientation()`;

Note The camera frame is positioned at the back of the left eye of the ZED.

Return The current state of the tracking process.

Parameters

- `camera_pose`: (out) : the pose containing the position of the camera (path or position) and other information (timestamp, confidence)
- `reference_frame`: : defines the reference from which you want the pose to be expressed.

`sl::AREA_EXPORT_STATE getAreaExportState ()`

Returns the state of exportation of the area database (spatial memory).

Return The current state of the exportation of the area file.

void **disableTracking** (`sl::String area_file_path = ""`)

Disables motion tracking.

Warning This feature is still in beta, you might encounter reloading issues. Please also note that the '.area' database depends on the depth map `sl::SENSING_MODE` chosen during the recording. The same mode must be used to reload the database.

Note The saving is done asynchronously, the state can be get by `getAreaExportState()`.

Parameters

- `area_file_path`: (optional) : if set, save the spatial database in a '.area' file. `areaFilePath` is the name and path of the database, e.g. : "path/to/file/myArea1.area".

void **resetTracking** (`sl::Transform &path`)

Resets the tracking, re-initializes the path with the transformation matrix given.

Note Please note that this function will also flush the area database built / loaded.

Spatial Mapping

`ERROR_CODE enableSpatialMapping (SpatialMappingParameters spatial_mapping_parameters = SpatialMappingParameters ())`

Initializes and starts the spatial mapping processes. The spatial mapping will create a geometric representation of the scene based on both tracking data and 3D point clouds. The resulting output is a `sl::Mesh` and

can be obtained by the `sl::Camera::extractWholeMesh` function or with `sl::Camera::retrieveMeshAsync` after calling `sl::Camera::requestMeshAsync`.

Return `sl::SUCCESS` if everything went fine, `sl::ERROR_CODE_FAILURE` otherwise

Warning The tracking needs to be enabled to create a map

Warning The performance greatly depends on the input parameters. If the mapping framerate is too slow in live mode, consider using a SVO file, or choose a coarser mesh resolution

Note This features is using host memory (RAM) to store the 3D map, the maximum amount of available memory allowed can be tweaked using the [SpatialMappingParameters](#).

Parameters

- `spatial_mapping_parameters`: : the structure containing all the specific parameters for the spatial mapping. default : Leave it empty to get best default parameters or initialize it from a preset. For more informations, checkout the [sl::SpatialMappingParameters](#) documentation.

void **pauseSpatialMapping** (bool *status*)

Switches the pause status of the data integration mechanism for the spatial mapping.

Parameters

- `status`: : if true, the integration is paused. If false, the spatial mapping is resumed.

[SPATIAL_MAPPING_STATE](#) **getSpatialMappingState** ()

Returns the current spatial mapping state.

Return `status` The current state of the spatial mapping process

[ERROR_CODE](#) **extractWholeMesh** (*sl::Mesh &mesh*)

Extracts the current mesh from the spatial mapping process.

Note This function will return when the mesh has been created or updated. This is therefore a blocking function. You should either call it in a thread or at the end of the mapping process. Calling this function in the grab loop will block the depth and tracking computation and therefore gives bad results.

Return `sl::SUCCESS` if the mesh is filled and available, otherwise `sl::ERROR_CODE_FAILURE`.

Parameters

- `mesh`: (out) : The mesh to be filled.

void **requestMeshAsync** ()

Starts the mesh generation process in a non blocking thread from the spatial mapping process.

Note Only one mesh generation can be done at a time, consequently while the previous launch is not done every call will be ignored.

[ERROR_CODE](#) **getMeshRequestStatusAsync** ()

Returns the mesh generation status, useful to after calling `requestMeshAsync`.

Return `sl::SUCCESS` if the mesh is ready and not yet retrieved, otherwise `sl::ERROR_CODE_FAILURE`.

[ERROR_CODE](#) **retrieveMeshAsync** (*sl::Mesh &mesh*)

Retrieves the generated mesh after calling `requestMeshAsync`.

Return `sl::SUCCESS` if the mesh is retrieved, otherwise `sl::ERROR_CODE_FAILURE`.

Parameters

- `mesh`: (out) : The mesh to be filled.

void **disableSpatialMapping** ()

Disables the Spatial Mapping process. All the spatial mapping functions are disabled, mesh cannot be retrieved after this call.

Recorder

ERROR_CODE **enableRecording** (*sl::String* `video_filename`, *SVO_COMPRESSION_MODE* `compression_mode = SVO_COMPRESSION_MODE_LOSSLESS`)

Creates a file for recording the current frames.

Warning This function can be called multiple times during ZED lifetime, but if `video_filename` is already existing, the file will be erased.

Return an `sl::ERROR_CODE` that defines if file was successfully created and can be filled with images.
* `sl::SUCCESS` if file can be filled * `sl::ERROR_CODE_SVO_RECORDING_ERROR` if something wrong happens.

Parameters

- `video_filename`: : can be a *.svo file or a *.avi file (detected by the suffix name provided).
- `compression_mode`: : can be one of the `sl::SVO_COMPRESSION_MODE` enum.

sl::RecordingState **record** ()

Records the current frame provided by *grab()* into the file.

Warning *grab()* must be called before *record()* to take the last frame available. Otherwise, it will be the last grabbed frame.

Return The recording state structure, for more details see *sl::RecordingState*.

void **disableRecording** ()

Disables the recording and closes the generated file.

Public Functions

Camera ()

Default constructor which creates an empty *Camera*.

~Camera ()

Camera destructor.

void **close** ()

Closes the camera and free the memory. *Camera::open* can then be called again to reset the camera if needed.

ERROR_CODE **open** (*InitParameters* `init_parameters = InitParameters ()`)

Opens the ZED camera in the desired mode (live/SVO), sets all the defined parameters, checks hardware requirements and launch internal self calibration.

Return An error code given informations about the internal process, if SUCCESS is returned, the camera is ready to use. Every other code indicates an error and the program should be stopped. For more details see `sl::ERROR_CODE`.

Parameters

- `init_parameters`: : a structure containing all the individual parameters

bool **isOpened** ()

Tests if the camera is opened and running.

Return true if the ZED is already setup, otherwise false.

ERROR_CODE **grab** (*RuntimeParameters* `rt_parameters` = `RuntimeParameters ()`)

Grabs a new image, rectifies it and computes the disparity map and optionally the depth map. The grabbing function is typically called in the main loop.

Return An `sl::SUCCESS` if no problem was encountered, `sl::ERROR_CODE_NOT_A_NEW_FRAME` otherwise if something wrong happens

Parameters

- `rt_parameters`: : a structure containing all the individual parameters.

ERROR_CODE **retrieveImage** (*Mat* &`mat`, *VIEW* `view` = `VIEW_LEFT`, *MEM* `type` = `MEM_CPU`)

Downloads the rectified image from the device and returns the CPU buffer. The retrieve function should be called after the function *Camera::grab*.

Return SUCCESS if the method succeeded, `ERROR_CODE_FAILURE` if an error occurred.

Parameters

- `mat`: : the *Mat* to store the image.
- `view`: : defines the image side wanted (see `sl::VIEW`)
- `type`: : the memory type desired. `sl::MEM_CPU` by default.

ERROR_CODE **retrieveMeasure** (*Mat* &`mat`, *MEASURE* `measure` = `MEASURE_DEPTH`, *MEM* `type` = `MEM_CPU`)

Downloads the measure (disparity, depth or confidence of disparity) from the device and returns the CPU buffer. The retrieve function should be called after the function *Camera::grab*.

Return SUCCESS if the method succeeded, `ERROR_CODE_FAILURE` if an error occurred.

Parameters

- `mat`: : the *Mat* to store the measures.
- `measure`: : defines the type wanted, such as disparity map, depth map or the confidence (see `sl::MEASURE`)
- `type`: : the memory type desired. `sl::MEM_CPU` by default.

void **setConfidenceThreshold** (int `conf_threshold_value`)

Sets a threshold for the disparity map confidence (and by extension the depth map). The function should be called before *Camera::grab* to be taken into account.

Parameters

- `conf_threshold_value`: : a value in [1,100]. A lower value means more confidence and precision (but less density), an upper value reduces the filtering (more density, less certainty). Other value means no filtering.

int **getConfidenceThreshold** ()

Returns the current confidence threshold value apply to the disparity map (and by extension the depth map).

Return The current threshold value between 0 and 100.

CUcontext **getCUDAContext** ()

Returns the CUDA context used for all the computation.

Return The CUDA context created by the inner process.

Resolution **getResolution** ()

Returns the current image size.

Return The image resolution.

void **setDepthMaxRangeValue** (float *depth_max_range*)

Sets the maximum distance of depth/disparity estimation (all values after this limit will be reported as TOO_FAR value).

Parameters

- `depth_max_range`: : maximum distance in the defined sl::UNIT.

float **getDepthMaxRangeValue** ()

Returns the current maximum distance of depth/disparity estimation.

Return The current maximum distance that can be computed in the defined sl::UNIT.

float **getDepthMinRangeValue** ()

Returns the closest measurable distance by the camera, according to the camera and the depth map parameters.

Return The minimum distance that can be computed in the defined sl::UNIT.

Public Static Functions

static *sl::String* **getSDKVersion** ()

Returns the version of the currently installed ZED SDK.

Return The ZED SDK version as a string with the following format : MAJOR.MINOR.PATCH

static int **isZEDconnected** ()

Checks if ZED cameras are connected, can be called before instantiating a *Camera* object.

Return The number of connected ZED.

Warning On Windows, only one ZED is accessible so this function will return 1 even if multiple ZED are connected.

static *sl::ERROR_CODE* **sticktoCPUCore** (int *cpu_core*)

Sticks the calling thread to a specific CPU core. This function is only available for Jetson TK1 and TX1.

Return *sl::SUCCESS* if stick is OK, otherwise status error.

Warning Function only available for Jetson. On other platform, result will be always 0 and no operations are performed.

Parameters

- *cpuCore*: : int that defines the core the thread must be run on. could be between 0 and 3. (cpu0,cpu1,cpu2,cpu3).

Private Functions

ERROR_CODE **openCamera** (bool)

bool **nextImage** (bool)

int **initMemory** ()

bool **initRectifier** ()

Private Members

CameraMemberHandler *h = 0

bool **opened** = false

struct CameraInformation

#include <Core.hpp> *Camera* specific parameters.

Public Members

CalibrationParameters **calibration_parameters**

Intrinsic and Extrinsic stereo parameters for rectified images (default).

CalibrationParameters **calibration_parameters_raw**

Intrinsic and Extrinsic stereo parameters for original images (unrectified).

unsigned int **serial_number** = 0

camera dependent serial number.

unsigned int **firmware_version** = 0

current firmware version of the camera.

struct CameraParameters

#include <Core.hpp> Intrinsic parameters of a camera.

Note Similar to the *CalibrationParameters*, those parameters are taken from the settings file (SNXXX.conf) and are modified during the *sl::Camera::open* call (with or without Self-Calibration). Those parameters given after *sl::Camera::open* call, represent the “new camera matrix” that fits/defines each image taken after rectification (through *retrieveImage*).

Note fx, fy, cx, cy must be the same for Left and Right *Camera* once *sl::Camera::open* has been called. Since distortion is corrected during rectification, distortion should not be considered after *sl::Camera::open* call.

Public Functions

void **SetUp** (float *focal_x*, float *focal_y*, float *center_x*, float *center_y*)
Setups the parameter of a camera.

Parameters

- *focal_x*: : horizontal focal length.
- *focal_y*: : vertical focal length.
- *focal_x*: : horizontal optical center.
- *focal_x*: : vertical optical center.

Public Members

float **fx**
Focal length in pixels along x axis.

float **fy**
Focal length in pixels along y axis.

float **cx**
Optical center along x axis, defined in pixels (usually close to width/2).

float **cy**
Optical center along y axis, defined in pixels (usually close to height/2).

double **disto**[5]
Distortion factor : [k_1, k_2, p_1, p_2, k_3]. Radial (k_1, k_2, k_3) and Tangential (p_1, p_2) distortion.

float **v_fov**
Vertical field of view after stereo rectification, in degrees.

float **h_fov**
Horizontal field of view after stereo rectification, in degrees.

float **d_fov**
Diagonal field of view after stereo rectification, in degrees.

Resolution **image_size**
size in pixels of the images given by the camera.

struct data

Public Members

int **w**

int **h**

matrix **X**

matrix **y**

```
int shallow
int *num_boxes
box **boxes
```

```
struct dbox
```

Public Members

```
float dx
float dy
float dw
float dh
```

```
struct detection_info
```

Public Members

```
char name[32]
int left
int right
int top
int bottom
float prob
```

```
struct float_pair
```

Public Members

```
float *x
float *y
```

```
class IMAGE
    Inherits from Structure
```

Private Static Attributes

```
list FaceDetect.IMAGE._fields_ = [("w", c_int), ("h", c_int), ("c", c_int), ("data", POINTER(c_float))]
```

```
class IMAGE
    Inherits from Structure
```

Private Static Attributes

```
list ObjectRecognition.IMAGE._fields_ = [("w", c_int), ("h", c_int), ("c", c_int), ("data", POINTER(c_float))]
```

```
class IMAGE
    Inherits from Structure
```

Private Static Attributes

```
list darknet.IMAGE._fields_ = [("w", c_int), ("h", c_int), ("c", c_int), ("data", POINTER(c_float))]
```

```
struct image
```

Public Members

```
int w
```

```
int h
```

```
int c
```

```
float *data
```

```
class ImageClass
```

Public Functions

```
__init__(self self, name name, image_paths image_paths)
```

```
__str__(self self)
```

```
__len__(self self)
```

Public Members

```
name
```

```
image_paths
```

```
struct Indice
```

```
#include <Mesh.hpp> Store the index per faces of the associated vertices/normals/texture coordinates.
```

Note Contains data only after you call *Mesh::applyTexture()*.

Public Members

```
sl::uint3 v_vn_ind  
vertices and normals indices.
```

```
sl::uint3 uv_ind  
texture coordinates indices.
```

```
struct InfoOption
```

Public Members

```
std::string svo_path
```

```
bool recordingMode = false
```

```
std::string output_path
```

```
bool computeDisparity = false
```

bool **videoMode** = false

class **InitParameters**

#include <Camera.hpp> Parameters for ZED initialization.

A default constructor is enable.

Public Functions

InitParameters (*RESOLUTION* camera_resolution_ = RESOLUTION_HD720, int camera_fps_ = 0, int camera_linux_id_ = 0, *sl::String* svo_input_filename_ = *sl::String*(), bool svo_real_time_mode_ = false, *DEPTH_MODE* depth_mode_ = DEPTH_MODE_PERFORMANCE, *UNIT* coordinate_units_ = UNIT_MILLIMETER, *COORDINATE_SYSTEM* coordinate_system_ = COORDINATE_SYSTEM_IMAGE, bool sdk_verbose_ = false, int sdk_gpu_id_ = -1, float depth_minimum_distance_ = -1., bool camera_disable_self_calib_ = false, bool camera_image_flip_ = false, int camera_buffer_count_linux_ = 4)

Default constructor, set all parameters to their default and optimized values.

bool **save** (*sl::String* filename)

Saves the current bunch of parameters into a file.

Return true if file was successfully saved, otherwise false.

Parameters

- filename: : the path to the file in which the parameters will be stored.

bool **load** (*sl::String* filename)

Loads the values of the parameters contained in a file.

Return true if the file was successfully loaded, otherwise false.

Parameters

- filename: : the path to the file from which the parameters will be loaded.

Public Members

RESOLUTION **camera_resolution**

Define the chosen ZED resolution default : RESOLUTION_HD720.

int **camera_fps**

Requested FPS for this resolution.

set as 0 will choose the default FPS for this resolution (see User guide). default : 0

int **camera_linux_id**

ONLY for LINUX : if multiple ZEDs are connected, it will choose the first zed listed (if zed_linux_id=0), the second listed (if zed_linux_id=1), ...

Each ZED will create its own memory (CPU and GPU), therefore the number of ZED available will depend on the configuration of your computer. Currently not available for Windows default : 0

sl::String **svo_input_filename**

Path with filename to the recorded SVO file.

bool **svo_real_time_mode**

When enabled the timestamp is taken as reference to determine the reading framerate.

This mode simulates the live camera and consequently skipped frames if the computation framerate is too slow. default : false

UNIT **coordinate_units**

Define the unit for all the metric values (depth, point cloud, tracking).

default : sl::UNIT::UNIT_MILLIMETER

COORDINATE_SYSTEM **coordinate_system**

Define the coordinate system of the world frame (and the camera frame as well).

This defines the order of the axis of the coordinate system. see COORDINATE_SYSTEM for more information. default : COORDINATE_SYSTEM::COORDINATE_SYSTEM_IMAGE

DEPTH_MODE **depth_mode**

Defines the quality of the depth map, affects the level of details and also the computation time.

default : DEPTH_MODE::DEPTH_MODE_PERFORMANCE

float **depth_minimum_distance**

Specify the minimum depth information that will be computed, in the sl::UNIT you previously define.

default : 70cm (-1)

Warning The computation time is affected by the value, exponentially. The closer it gets the longer the disparity search will take. In case of limited computation power, consider increasing the value.

Note This value is used to calculate the disparity range estimation (in pixels). Due to metric to pixel conversion, a small difference may occur with the value returned by getDepthMinRange().

int **camera_image_flip**

Defines if the image are horizontally flipped.

default : 0

bool **camera_disable_self_calib**

If set to true, it will disable self-calibration and take the optional calibration parameters without optimizing them.

It is advised to leave it as false, so that calibration parameters can be optimized. default : false

int **camera_buffer_count_linux**

Set the number of buffers in the internal grabbing process.

decrease this number may reduce latency but can also produce more corrupted frames. default: 4

Warning Linux Only, this parameter has no effect on Windows.

bool **sdk_verbose**

If set to true, it will output some information about the current status of initialization.

default : false

int **sdk_gpu_id**

Defines the graphics card on which the computation will be done.

The default value search the more powerful (most CUDA cores) usable GPU. default : -1

struct **kvp**

Public Members

char ***key**

char ***val**

int **used**

struct layer

Public Members

LAYER_TYPE **type**

ACTIVATION **activation**

COST_TYPE **cost_type**

void (***forward**) (struct *layer*, struct *network*)

void (***backward**) (struct *layer*, struct *network*)

void (***update**) (struct *layer*, *update_args*)

void (***forward_gpu**) (struct *layer*, struct *network*)

void (***backward_gpu**) (struct *layer*, struct *network*)

void (***update_gpu**) (struct *layer*, *update_args*)

int **batch_normalize**

int **shortcut**

int **batch**

int **forced**

int **flipped**

int **inputs**

int **outputs**

int **nweights**

int **nbiases**

int **extra**

int **truths**

int **h**

int **w**

int **c**

int **out_h**

int **out_w**

int **out_c**

int **n**

int **max_boxes**

`int groups`
`int size`
`int side`
`int stride`
`int reverse`
`int flatten`
`int spatial`
`int pad`
`int sqrt`
`int flip`
`int index`
`int binary`
`int xnor`
`int steps`
`int hidden`
`int truth`
`float smooth`
`float dot`
`float angle`
`float jitter`
`float saturation`
`float exposure`
`float shift`
`float ratio`
`float learning_rate_scale`
`int softmax`
`int classes`
`int coords`
`int background`
`int rescore`
`int objectness`
`int does_cost`
`int joint`
`int noadjust`
`int reorg`
`int log`

int **tanh**
float **alpha**
float **beta**
float **kappa**
float **coord_scale**
float **object_scale**
float **noobject_scale**
float **mask_scale**
float **class_scale**
int **bias_match**
int **random**
float **thresh**
int **classfix**
int **absolute**
int **onlyforward**
int **stopbackward**
int **dontload**
int **dontloadscases**
float **temperature**
float **probability**
float **scale**
char ***cweights**
int ***indexes**
int ***input_layers**
int ***input_sizes**
int ***map**
float ***rand**
float ***cost**
float ***state**
float ***prev_state**
float ***forgot_state**
float ***forgot_delta**
float ***state_delta**
float ***combine_cpu**
float ***combine_delta_cpu**
float ***concat**

```
float *concat_delta
float *binary_weights
float *biases
float *bias_updates
float *scales
float *scale_updates
float *weights
float *weight_updates
float *delta
float *output
float *squared
float *norms
float *spatial_mean
float *mean
float *variance
float *mean_delta
float *variance_delta
float *rolling_mean
float *rolling_variance
float *x
float *x_norm
float *m
float *v
float *bias_m
float *bias_v
float *scale_m
float *scale_v
float *z_cpu
float *r_cpu
float *h_cpu
float *prev_state_cpu
float *temp_cpu
float *temp2_cpu
float *temp3_cpu
float *dh_cpu
float *hh_cpu
```

```
float *prev_cell_cpu
float *cell_cpu
float *f_cpu
float *i_cpu
float *g_cpu
float *o_cpu
float *c_cpu
float *dc_cpu
float *binary_input
struct layer *input_layer
struct layer *self_layer
struct layer *output_layer
struct layer *reset_layer
struct layer *update_layer
struct layer *state_layer
struct layer *input_gate_layer
struct layer *state_gate_layer
struct layer *input_save_layer
struct layer *state_save_layer
struct layer *input_state_layer
struct layer *state_state_layer
struct layer *input_z_layer
struct layer *state_z_layer
struct layer *input_r_layer
struct layer *state_r_layer
struct layer *input_h_layer
struct layer *state_h_layer
struct layer *wz
struct layer *uz
struct layer *wr
struct layer *ur
struct layer *wh
struct layer *uh
struct layer *uo
struct layer *wo
struct layer *uf
```

```
struct layer *wf
struct layer *ui
struct layer *wi
struct layer *ug
struct layer *wg
tree *softmax_tree
size_t workspace_size
```

```
struct list
```

Public Members

```
int size
node *front
node *back
```

```
struct load_args
```

Public Members

```
int threads
char **paths
char *path
int n
int m
char **labels
int h
int w
int out_w
int out_h
int nh
int nw
int num_boxes
int min
int max
int size
int classes
int background
int scale
```

```

int center
int coords
float jitter
float angle
float aspect
float saturation
float exposure
float hue
data *d
image *im
image *resized
data_type type
tree *hierarchy

```

class **Mat**

#include <Core.hpp> The *Mat* class can handle multiple matrix format from 1 to 4 channels, with different value types (float or uchar), and can be stored CPU and/or GPU side. *sl::Mat* is defined in a row-major order: - It means that, in the image buffer, the entire first row is stored first, followed by the entire second row, and so on. The CPU and GPU buffer aren't automatically synchronized for performance reasons, you can use *Mat::updateCPUfromGPU* / *Mat::updateGPUfromCPU* to do it. If you are using the GPU side of the *Mat* object, you need to make sure to call *sl::Mat::free()* before destroying the *sl::Camera* object. The destruction of the *sl::Camera* object delete the CUDA context needed to free the GPU *Mat* memory.

Public Functions

Mat ()

empty *Mat* default constructor.

Mat (size_t width, size_t height, *MAT_TYPE* mat_type, *MEM* memory_type = MEM_CPU)
Mat constructor.

Parameters

- width: : width of the matrix in pixels.
- height: : height of the matrix in pixels.
- mat_type: : the type of the matrix (*sl::MAT_TYPE_32F_C1*, *sl::MAT_TYPE_8U_C4*...)
- memory_type: : defines where the buffer will be stored. (*sl::MEM_CPU* and/or *sl::MEM_GPU*). This function directly allocates the requested memory. It calls *Mat::alloc*.

Mat (size_t width, size_t height, *MAT_TYPE* mat_type, *sl::uchar1* *ptr, size_t step, *MEM* memory_type = MEM_CPU)
Mat constructor from an existing data pointer.

Parameters

- width: : width of the matrix in pixels.

- `height`: : height of the matrix in pixels.
- `mat_type`: : the type of the matrix (`sl::MAT_TYPE_32F_C1`, `sl::MAT_TYPE_8U_C4...`)
- `ptr`: : pointer to the data array. (CPU or GPU).
- `step`: : step of the data array. (the Bytes size of one pixel row)
- `memory_type`: : defines where the buffer will be stored. (`sl::MEM_CPU` and/or `sl::MEM_GPU`). This function doesn't allocate the memory.

Mat (`size_t width`, `size_t height`, *MAT_TYPE* `mat_type`, *sl::uchar1* `*ptr_cpu`, `size_t step_cpu`, *sl::uchar1* `*ptr_gpu`, `size_t step_gpu`)
Mat constructor from two existing data pointers, CPU and GPU.

Parameters

- `width`: : width of the matrix in pixels.
- `height`: : height of the matrix in pixels.
- `mat_type`: : the type of the matrix (`sl::MAT_TYPE_32F_C1`, `sl::MAT_TYPE_8U_C4...`)
- `ptr_cpu`: : CPU pointer to the data array.
- `step_cpu`: : step of the CPU data array. (the Bytes size of one pixel row)
- `ptr_gpu`: : GPU pointer to the data array.
- `step_gpu`: : step of the GPU data array. (the Bytes size of one pixel row) This function doesn't allocate the memory.

Mat (*sl::Resolution* `resolution`, *MAT_TYPE* `mat_type`, *MEM* `memory_type` = `MEM_CPU`)
Mat constructor.

Parameters

- `resolution`: : the size of the matrix in pixels.
- `mat_type`: : the type of the matrix (`sl::MAT_TYPE_32F_C1`, `sl::MAT_TYPE_8U_C4...`)
- `memory_type`: : defines where the buffer will be stored. (`sl::MEM_CPU` and/or `sl::MEM_GPU`). This function directly allocates the requested memory. It calls *Mat::alloc*.

Mat (*sl::Resolution* `resolution`, *MAT_TYPE* `mat_type`, *sl::uchar1* `*ptr`, `size_t step`, *MEM* `memory_type` = `MEM_CPU`)
Mat constructor from an existing data pointer.

Parameters

- `resolution`: : the size of the matrix in pixels.
- `mat_type`: : the type of the matrix (`sl::MAT_TYPE_32F_C1`, `sl::MAT_TYPE_8U_C4...`)
- `ptr`: : pointer to the data array. (CPU or GPU).
- `step`: : step of the data array. (the Bytes size of one pixel row)
- `memory_type`: : defines where the buffer will be stored. (`sl::MEM_CPU` and/or `sl::MEM_GPU`). This function doesn't allocate the memory.

Mat (*sl::Resolution* resolution, *MAT_TYPE* mat_type, *sl::uchar1* *ptr_cpu, size_t step_cpu, *sl::uchar1* *ptr_gpu, size_t step_gpu)
Mat constructor from two existing data pointers, CPU and GPU.

Parameters

- resolution: : the size of the matrix in pixels.
- mat_type: : the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
- ptr_cpu: : CPU pointer to the data array.
- step_cpu: : step of the CPU data array. (the Bytes size of one pixel row)
- ptr_gpu: : GPU pointer to the data array.
- step_gpu: : step of the GPU data array. (the Bytes size of one pixel row) This function doesn't allocate the memory.

Mat (const *sl::Mat* &mat)
Mat constructor by copy (deep copy).

Parameters

- mat: : the reference to the *sl::Mat* to copy. This function allocates and duplicates the data

void **alloc** (size_t width, size_t height, *MAT_TYPE* mat_type, *MEM* memory_type = MEM_CPU)
 Allocates the *Mat* memory.

Warning It erases previously allocated memory.

Parameters

- width: : width of the matrix in pixels
- height: : height of the matrix in pixels
- mat_type: : the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
- memory_type: : defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU).

void **alloc** (*sl::Resolution* resolution, *MAT_TYPE* mat_type, *MEM* memory_type = MEM_CPU)
 Allocates the *Mat* memory.

Warning It erases previously allocated memory.

Parameters

- resolution: : the size of the matrix in pixels.
- mat_type: : the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
- memory_type: : defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU).

~Mat ()
Mat destructor. This function calls *Mat::free* to release owned memory.

void **free** (*MEM* memory_type = MEM_CPU|MEM_GPU)
 Free the owned memory.

Parameters

- `memory_type`: : specify whether you want to free the `sl::MEM_CPU` and/or `sl::MEM_GPU` memory.

Mat &operator= (const *Mat* &that)

Performs a shallow copy. This function doesn't copy the data array, it only copies the pointer.

Return The new *sl::Mat* object which point to the same data as that.

Parameters

- `that`: : the *sl::Mat* to be copied.

ERROR_CODE updateCPUfromGPU ()

Downloads data from DEVICE (GPU) to HOST (CPU), if possible.

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Note If no CPU or GPU memory are available for this *Mat*, some are directly allocated.

Note If verbose sets, you have informations in case of failure.

ERROR_CODE updateGPUfromCPU ()

Uploads data from HOST (CPU) to DEVICE (GPU), if possible.

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Note If no CPU or GPU memory are available for this *Mat*, some are directly allocated.

Note If verbose sets, you have informations in case of failure.

ERROR_CODE copyTo (*Mat* &dst, *COPY_TYPE* cpyType = COPY_TYPE_CPU_CPU) const

Copies data an other *Mat* (deep copy).

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Note If the destination is not allocated or has a not a compatible `sl::MAT_TYPE` or *sl::Resolution*, current memory is freed and new memory is directly allocated.

Parameters

- `dst`: : the *Mat* where the data will be copied.
- `cpyType`: : specify the memories that will be used for the copy.

ERROR_CODE setFrom (const *Mat* &src, *COPY_TYPE* cpyType = COPY_TYPE_CPU_CPU)

Copies data from an other *Mat* (deep copy).

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Note If the current *Mat* is not allocated or has a not a compatible `sl::MAT_TYPE` or *sl::Resolution* with the source, current memory is freed and new memory is directly allocated.

Parameters

- `src`: : the *Mat* where the data will be copied from.
- `cpyType`: : specify the memories that will be used for the update.

***ERROR_CODE* read (const char *filePath)**

Reads an image from a file (only if `sl::MEM_CPU` is available on the current `sl::Mat`).

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Note Supported `sl::MAT_TYPE` are : `sl::MAT_TYPE_8U_C1`, `sl::MAT_TYPE_8U_C3` and `sl::MAT_TYPE_8U_C4`. input files format are PNG and JPEG. verbose sets, you have informations in case of failure.

Parameters

- `filePath`: : file path including the name and extension.

***ERROR_CODE* write (const char *filePath)**

Writes the `sl::Mat` (only if `sl::MEM_CPU` is available) into a file as an image.

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Note Supported `sl::MAT_TYPE` are : `sl::MAT_TYPE_8U_C1`, `sl::MAT_TYPE_8U_C3` and `sl::MAT_TYPE_8U_C4`. output files format are PNG and JPEG. verbose sets, you have informations in case of failure.

Parameters

- `filePath`: : file path including the name and extension.

template <typename T>***ERROR_CODE* setTo (T value, *MEM* memory_type = MEM_CPU)**

Fills the `Mat` with the given value.

Note This function is templated for `sl::uchar1`, `sl::uchar2`, `sl::uchar3`, `sl::uchar4`, `sl::float1`, `sl::float2`, `sl::float3`, `sl::float4`.

Parameters

- `value`: : the value to be copied all over the matrix.
- `memory_type`: : defines which buffer to fill, CPU and/or GPU. This function overwrite all the matrix.

template <typename N>***ERROR_CODE* setValue (size_t x, size_t y, N value, *MEM* memory_type = MEM_CPU)**

Sets a value to a specific point in the matrix.

Return `sl::SUCCESS` if everything went well, `sl::ERROR_CODE_FAILURE` otherwise.

Warning Not efficient for `sl::MEM_GPU`, use it on sparse data.

Note This function is templated for `sl::uchar1`, `sl::uchar2`, `sl::uchar3`, `sl::uchar4`, `sl::float1`, `sl::float2`, `sl::float3`, `sl::float4`.

Parameters

- `x`: : specify the column.
- `y`: : specify the row.
- `value`: : the value to be set.
- `memory_type`: : defines which memory will be updated.

template <typename N>

ERROR_CODE **getValue** (size_t x, size_t y, N *value, *MEM* memory_type = MEM_CPU)

Returns the value of a specific point in the matrix.

Return sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Warning Not efficient for sl::MEM_GPU, use it on sparse data.

Note This function is templated for sl::uchar1, sl::uchar2, sl::uchar3, sl::uchar4, sl::float1, sl::float2, sl::float3, sl::float4.

Parameters

- x: : specify the column
- y: : specify the row
- memory_type: : defines which memory should be read.

size_t **getWidth** () const

brief Returns the width of the matrix.

Return The width of the matrix in pixels.

size_t **getHeight** () const

brief Returns the height of the matrix.

Return The height of the matrix in pixels.

Resolution **getResolution** () const

brief Returns the height of the matrix.

Return The height of the matrix in pixels.

size_t **getChannels** () const

brief Returns the number of values stored in one pixel.

Return The number of values in a pixel.

MAT_TYPE **getDataType** () const

brief Returns the format of the matrix.

Return The format of the current *Mat*.

MEM **getMemoryType** () const

brief Returns the type of memory (CPU and/or GPU).

Return The type of allocated memory.

template <typename N>

N ***getPtr** (*MEM* memory_type = MEM_CPU)

brief Returns the CPU or GPU data pointer.

Return The pointer of the *Mat* data.

Parameters

- memory_type: : specify whether you want sl::MEM_CPU or sl::MEM_GPU step.

size_t **getStepBytes** (*MEM* memory_type = MEM_CPU)

brief Returns the memory step in Bytes (the Bytes size of one pixel row).

Return The step in bytes of the specified memory.

Parameters

- `memory_type`: : specify whether you want `sl::MEM_CPU` or `sl::MEM_GPU` step.

template <typename N>

`size_t` **getStep** (*MEM* `memory_type` = `MEM_CPU`)

brief Returns the memory step in number of elements (the number of values in one pixel row).

Return The step in number of elements.

Parameters

- `memory_type`: : specify whether you want `sl::MEM_CPU` or `sl::MEM_GPU` step.

`size_t` **getStep** (*MEM* `memory_type` = `MEM_CPU`)

brief Returns the memory step in number of elements (the number of values in one pixel row).

Return The step in number of elements.

Parameters

- `memory_type`: : specify whether you want `sl::MEM_CPU` or `sl::MEM_GPU` step.

`size_t` **getPixelBytes** ()

brief Returns the size in bytes of one pixel.

Return The size in bytes of a pixel.

`size_t` **getWidthBytes** ()

brief Returns the size in bytes of a row.

Return The size in bytes of a row.

sl::String **getInfos** ()

brief Return the informations about the *Mat* into a *sl::String*.

Return A string containing the *Mat* informations.

`bool` **isInit** ()

brief Defines whether the *Mat* is initialized or not.

Return True if current *Mat* has been allocated (by the constructor or therefore).

`bool` **isMemoryOwner** ()

brief Returns whether the *Mat* is the owner of the memory it access. If not, the memory won't be freed if the *Mat* is destroyed.

Return True if the *Mat* is owning its memory, else false.

`void` **clone** (`const` *Mat* &*src*)

Duplicates *Mat* by copy (deep copy).

Parameters

- `src`: : the reference to the *Mat* to copy. This function copies the data array(s), it mark the new *Mat* as the memory owner.

Public Members

sl::String **name**

`bool` **verbose** = false

Private Members

Resolution **size**

size_t **channels** = 0

size_t **step_gpu** = 0

size_t **step_cpu** = 0

size_t **pixel_bytes** = 0

MAT_TYPE **data_type**

MEM **mem_type** = sl::MEM_CPU

uchar1 ***ptr_cpu** = NULL

uchar1 ***ptr_gpu** = NULL

uchar1 ***ptr_internal** = NULL

bool **init** = false

bool **memory_owner** = false

struct matrix

Public Members

int **rows**

int **cols**

float ****vals**

class Matrix3f

#include <types.hpp> The class *Matrix3f* represent a generic three-dimensional matrix.

sl::Matrix3f is defined in a row-major order: - It means that, in the value buffer, the entire first row is stored first, followed by the entire second row, and so on. - you can access the data with the 'r' ptr or by element name as : r00, r01, r02 | 0 1 2 | r10, r11, r12 <-> r1 3 4 5 | r20, r21, r21 | 6 7 8 |

Subclassed by *sl::Rotation*

Public Functions

Matrix3f()

Matrix3f default constructor.

Matrix3f(float data[])

Matrix3f copy constructor (deep copy).

Matrix3f(const *Matrix3f* &mat)

brief *Matrix3f* copy constructor (deep copy).

Parameters

- rotation: : the *Matrix3f* to copy.

Matrix3f **operator***(const *Matrix3f* &mat) **const**

brief Gives the result of the multiplication between two *Matrix3f*

Matrix3f **operator*** (const double &*scalar*) const
 brief Gives the result of the multiplication between a *Matrix3f* and a scalar.

bool **operator==** (const *Matrix3f* &*mat*) const
 brief Test two *Matrix3f* equality.

bool **operator!=** (const *Matrix3f* &*mat*) const
 brief Test two *Matrix3f* inequality.

float &**operator()** (int *u*, int *v*)
 Gets access to a specific point in the *Matrix3f* (read / write).

Return The value at the *u*, *v* coordinates.

Parameters

- *u*: : specify the row
- *v*: : specify the column

void **inverse** ()
 Sets the *Matrix3f* to its inverse.

void **transpose** ()
 Sets the *RotationArray* to its transpose.

void **setIdentity** ()
 Sets the *Matrix3f* to identity.

void **setZeros** ()
 Sets the *Matrix3f* to zero.

sl::String **getInfos** ()
 Return the components of the *Matrix3f* in a sl::String.

Return A sl::String containing the components of the current *Matix3f*.

Public Members

float **r00**

float **r01**

float **r02**

float **r10**

float **r11**

float **r12**

float **r20**

float **r21**

float **r22**

float **r**[nbElem]

union **sl::Matrix3f::@1 sl::Matrix3f::@2**

sl::String **matrix_name**
Name of the matrix (optional).

Public Static Functions

static *Matrix3f* **inverse** (**const** *Matrix3f* &rotation)
Returns the inverse of a *Matrix3f*.

Return The inverse of the given *Matrix3f*

Parameters

- rotation: : the *Matrix3f* to compute the inverse from.

static *Matrix3f* **transpose** (**const** *Matrix3f* &rotation)
Returns the transpose of a *Matrix3f*.

Return The transpose of the given *Matrix3f*

Parameters

- rotation: : the *Matrix3f* to compute the transpose from.

static *Matrix3f* **identity** ()
Creates an identity *Matrix3f*.

Return A *Matrix3f* set to identity.

static *Matrix3f* **zeros** ()
Creates a *Matrix3f* filled with zeros.

Return A *Matrix3f* set to zero.

Public Static Attributes

const int nbElem = 9

class **Matrix4f**

#include <types.hpp> The class *Matrix4f* represent a generic fourth-dimensional matrix.

sl::Matrix4f is defined in a row-major order: - It means that, in the value buffer, the entire first row is stored first, followed by the entire second row, and so on. - you can access the data by the 'm' ptr or by the element name as : r00, r01, r02, tx | 0 1 2 3 | r10, r11, r12, ty <-> m1 4 5 6 7 | r20, r21, r22, tz | 8 9 10 11 | m30, m31, m32, m33 | 12 13 14 15 |

Subclassed by *sl::Transform*

Public Functions

Matrix4f ()
Matrix4f default constructor.

Matrix4f (float data[])
Matrix4f copy constructor (deep copy).

Matrix4f (const *Matrix4f* &mat)

brief *Matrix4f* copy constructor (deep copy).

Parameters

- rotation: : the *Matrix4f* to copy.

Matrix4f **operator*** (const *Matrix4f* &mat) const

brief Gives the result of the multiplication between two *Matrix4f*.

Matrix4f **operator*** (const double &scalar) const

brief Gives the result of the multiplication between a *Matrix4f* and a scalar.

bool **operator==** (const *Matrix4f* &mat) const

brief Test two *Matrix4f* equality.

bool **operator!=** (const *Matrix4f* &mat) const

brief Test two *Matrix4f* inequality.

float &**operator** () (int u, int v)

Gets access to a specific point in the *Matrix4f* (read / write).

Return The value at the u, v coordinates.

Parameters

- u: : specify the row.
- v: : specify the column.

ERROR_CODE **inverse** ()

Sets the *Matrix4f* to its inverse.

Return SUCCESS if the inverse has been computed, *ERROR_CODE_FAILURE* is not (det = 0).

void **transpose** ()

Sets the *Matrix4f* to its transpose.

void **setIdentity** ()

Sets the *Matrix4f* to identity.

void **setZeros** ()

Sets the *Matrix4f* to zero.

ERROR_CODE **setSubMatrix3f** (sl::*Matrix3f* input, int row = 0, int column = 0)

Sets a 3x3 Matrix inside the *Matrix4f*.

Note Can be used to set the rotation matrix when the matrix4f is a pose or an isometric matrix.

Return SUCCESS if everything went well, *ERROR_CODE_FAILURE* otherwise.

Parameters

- sl::*Matrix3f*: : sub matrix to put inside the *Matrix4f*.
- row: : index of the row to start the 3x3 block. Must be 0 or 1.
- column: : index of the column to start the 3x3 block. Must be 0 or 1.

ERROR_CODE **setSubVector3f** (*sl::Vector3<float> input*, int *column* = 3)

Sets a 3x1 Vector inside the *Matrix4f* at the specified column index.

Note Can be used to set the Translation/Position matrix when the matrix4f is a pose or an isometry.

Return SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

Parameters

- *sl::Vector3*: : sub vector to put inside the *Matrix4f*.
- *column*: : index of the column to start the 3x3 block. By default, it is the last column (translation for a *Pose*).

ERROR_CODE **setSubVector4f** (*sl::Vector4<float> input*, int *column* = 3)

Sets a 4x1 Vector inside the *Matrix4f* at the specified column index.

Note Can be used to set the Translation/Position matrix when the matrix4f is a pose or an isometry.

Return SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

Parameters

- *sl::Vector4*: : sub vector to put inside the *Matrix4f*.
- *column*: : index of the column to start the 3x3 block. By default, it is the last column (translation for a *Pose*).

sl::String **getInfos** ()

Return the components of the *Matrix4f* in a *sl::String*.

Return A *sl::String* containing the components of the current *Matrix4f*.

Public Members

float **r00**

float **r01**

float **r02**

float **tx**

float **r10**

float **r11**

float **r12**

float **ty**

float **r20**

float **r21**

float **r22**

float **tz**

float **m30**

float **m31**

```
float m32
float m33
float m[nbElem]
union s1::Matrix4f::@5 s1::Matrix4f::@6
sl::String matrix_name
    Name of the matrix (optional).
```

Public Static Functions

static *Matrix4f* inverse (const *Matrix4f* &mat)
Creates the inverse of a *Matrix4f*.

Return The inverse of the given *Matrix4f*.

Parameters

- rotation: : the *Matrix4f* to compute the inverse from.

static *Matrix4f* transpose (const *Matrix4f* &mat)
Creates the transpose of a *Matrix4f*.

Return The transpose of the given *Matrix4f*.

Parameters

- rotation: : the *Matrix4f* to compute the transpose from.

static *Matrix4f* identity ()
Creates an identity *Matrix4f*.

Return A *Matrix4f* set to identity.

static *Matrix4f* zeros ()
Creates a *Matrix4f* filled with zeros.

Return A *Matrix4f* set to zero.

Public Static Attributes

```
const int nbElem = 16
```

class Mesh
#include <Mesh.hpp> A mesh contains the geometric data of the scene computed by the spatial mapping.

Public Functions

Mesh ()
Default constructor which creates an empty *Mesh*.

~Mesh ()
Mesh destructor.

bool **filter** (*MeshFilterParameters* params = MeshFilterParameters ())

Filters the mesh according to the given parameters.

Return True if the filtering was successful, false otherwise.

Note The filtering is a costly operation but the resulting mesh is significantly lighter and less noisy, the parameters can be tweaked to get a mesh that fit better the final application. For instance a collision mesh will need to have a coarser, more decimated resolution.

Parameters

- params: : defines the filtering parameters, for more info checkout the [sl::MeshFilterParameters](#) documentation.

bool **applyTexture** (*MESH_TEXTURE_FORMAT* texture_format = MESH_TEXTURE_RGB)

Applies texture to the mesh.

Return True if the texturing was successful, false otherwise.

Warning SpatialMappingParams::saveTextureData must be set as true when enabling the spatial mapping to be able to apply the textures.

Warning The mesh should be filtered before calling this function since [Mesh::filter](#) will erased the textures, the texturing is also significantly slower on non-filtered meshes.

bool **save** (*std::string* filename, *MESH_FILE_FORMAT* type = MESH_FILE_OBJ)

Saves the mesh into a file.

Return True if the file was successfully saved, false otherwise.

Note Only the sl::MESH_FILE_OBJ support the textures recording

Parameters

- filename: : the path and filename of the mesh
- type: : defines the file type (extension)

bool **load** (*const std::string* filename)

Loads the mesh from a file.

Parameters

- filename: : the path and filename of the mesh.

Return Value

- true: if the loading was successful, false otherwise.

void **clear** ()

Clears all the mesh data (empty the vectors).

Public Members

std::vector<sl::float3> **vertices**

Vertices are defined by a 3D point {x,y,z}.

std::vector<sl::uint3> **triangles**

Triangles (or faces) contains the index of its three vertices.

It corresponds to the 3 vertices of the triangle {v1, v2, v3}.

std::vector<sl::float3> **normals**

Normals are defined by three componant, {nx, ny, nz}.

Normals are defined for each vertices (*Mesh::vertices* and *Mesh::normals* are the same size).

std::vector<sl::float2> **uv**

Texture coordinates defines 2D points on a texture.

std::vector<std::vector<Indice>> **material_indices**

Store the list of vertices index affected to each texture image.

The first vector has the same size as *Mesh::textures*, the size of the second vector represents the number of faces associated to this texture. By running over the second vector you can access the vertices/normals and texture coordinates of the current texture.

Note Contains data only after you call *Mesh::applyTexture()*.

std::vector<Texture> **textures**

List of textures images.

Private Members

sl::TextureImagePool **im_pool**

sl::CameraParameters **cam_param**

float **min_d**

float **max_d**

bool **welded**

size_t **memory** = 0

class **MeshFilterParameters**

#include <Mesh.hpp> Parameters for the optional filtering step of a *sl::Mesh*.

A default constructor is enabled and set to its default parameters.

Note Parameters can be user adjusted.

Public Types

enum **FILTER**

Values:

FILTER_LOW

Soft decimation and smoothing.

FILTER_MEDIUM

Decimate the number of faces and apply a soft smooth.

FILTER_HIGH

Drastically reduce the number of faces.

Public Functions

MeshFilterParameters (*FILTER* *filtering_* = FILTER_LOW)

Default constructor, set all parameters to their default and optimized values.

void **set** (*FILTER* *filtering_* = FILTER_LOW)

Sets the filtering intensity.

Parameters

- *filtering_*: : the desired *sl::MeshFilterParameters::FILTER*.

bool **save** (*sl::String* *filename*)

Saves the current bunch of parameters into a file.

Return true if the file was successfully saved, otherwise false.

Parameters

- *filename*: : the path to the file in which the parameters will be stored.

bool **load** (*sl::String* *filename*)

Loads the values of the parameters contained in a file.

Return true if the file was successfully loaded, otherwise false.

Parameters

- *filename*: : the path to the file from which the parameters will be loaded.

Public Members

FILTER **filtering** = FILTER::FILTER_LOW

class **METADATA**

Inherits from Structure

Private Static Attributes

list **ObjectRecognition.METADATA._fields_** = [("classes", c_int), ("names", POINTER(c_char_p))]

struct **metadata**

Public Members

int **classes**

char ****names**

class **METADATA**

Inherits from Structure

Private Static Attributes

```
list FaceDetect.METADATA._fields_ = [("classes", c_int), ("names", POINTER(c_char_p))]
```

class **METADATA**

Inherits from Structure

Private Static Attributes

```
list darknet.METADATA._fields_ = [("classes", c_int), ("names", POINTER(c_char_p))]
```

struct **moves**

Public Members

char ****data**

int **n**

struct **network**

Public Members

int **n**

int **batch**

size_t ***seen**

int ***t**

float **epoch**

int **subdivisions**

layer ***layers**

float ***output**

learning_rate_policy **policy**

float **learning_rate**

float **momentum**

float **decay**

float **gamma**

float **scale**

float **power**

int **time_steps**

int **step**

int **max_batches**

float ***scales**

int ***steps**

```
int num_steps
int burn_in
int adam
float B1
float B2
float eps
int inputs
int outputs
int truths
int notruth
int h
int w
int c
int max_crop
int min_crop
int center
float angle
float aspect
float exposure
float saturation
float hue
int gpu_index
tree *hierarchy
float *input
float *truth
float *delta
float *workspace
int train
int index
float *cost
```

struct node

Public Members

```
void *val
struct node *next
struct node *prev
```


class Orientation

#include <Core.hpp> The class *Orientation* is designed to contains orientation data from the tracking. *sl::Orientation* is a vector defined as [ox, oy, oz, ow].

Inherits from Vector4< float >

Public Functions**Orientation()**

empty *Orientation* default constructor.

Orientation(const Orientation &orientation)

brief *Orientation* copy constructor (deep copy).

Parameters

- orientation: : the *Orientation* to copy.

Orientation(const Vector4<float> &in)

brief *Orientation* copy constructor (deep copy).

Parameters

- in: : the vector to copy.

Orientation(const Rotation &rotation)

brief *Orientation* constructor from an *Rotation*. It converts the *Rotation* representation to the *Orientation* one.

Parameters

- rotation: : the *Rotation* to be used.

Orientation(const Translation &tr1, const Translation &tr2)

brief *Orientation* constructor from a vector represented by two *Translation*.

Parameters

- tr1: : the first point of the vector.
- tr2: : the second point of the vector.

float operator() (int x)

Returns the value at specific position in the *Orientation*.

Return The value at the x position.

Parameters

- x: : the position of the value

Orientation operator* (const Orientation &orientation) const

brief Multiplication operator by an *Orientation*.

Return The current orientation after being multiplied by the other orientation.

Parameters

- orientation: : the orientation.

void setRotation(const Rotation &rotation)

Sets the orientation from a *Rotation*.

Parameters

- `rotation`: : the *Rotation* to be used.

Rotation **getRotation () const**

Returns the current orientation as a *Rotation*.

Return The rotation computed from the orientation data.

void **setIdentity ()**

Sets the current *Orientation* to identity.

void **setZeros ()**

Fills the current *Orientation* with zeros.

void **normalise ()**

Normalizes the current *Orientation*.

Public Static Functions

static *Orientation* identity ()

Creates an *Orientation* initialized to identity.

Return An identity *Orientation*.

static *Orientation* zeros ()

Creates an *Orientation* filled with zeros.

Return An *Orientation* filled with zeros.

static *Orientation* normalise (const *Orientation* &orient)

Creates the normalized version of an existing *Orientation*.

Return The normalized version of the *Orientation*.

Parameters

- `orient`: : the *Orientation* to be used.

class Pose

#include <Camera.hpp> The class *Pose* contains the Motion tracking data which gives the position and orientation of the ZED in space (orientation/quaternion, rotation matrix, translation/position) and other connected values (timestamp, confidence).

Public Functions

Pose ()

Default constructor which creates an empty *Pose*.

Pose (const *Pose* &pose)

Pose constructor with deep copy.

Pose (**const** *sl::Transform* &*pose_data*, unsigned long long *mtimestamp* = 0, int *mconfidence* = 0)
Pose constructor with deep copy.

~Pose ()
Pose destructor.

sl::Translation **getTranslation** ()
 Returns the camera translation from the pose.

Return The translation vector of the ZED position.

sl::Orientation **getOrientation** ()
 Returns the camera orientation from the pose.

Return The orientation vector of the ZED position.

sl::Rotation **getRotation** ()
 brief Returns the camera rotation (3x3) from the pose.

Return The rotation matrix of the ZED position.

sl::Vector3<float> **getRotationVector** ()
 Returns the camera rotation (3x1) of the pose.

Return The rotation vector of the ZED position.

Public Members

bool **valid**
 boolean that indicates if tracking is activated or not. You should check that first if something wrong.

unsigned long long **timestamp**
 Timestamp of the pose. This timestamp should be compared with the camera timestamp for synchronization.

sl::Transform **pose_data**
 4x4 Matrix which contains the rotation (3x3) and the translation. *Orientation* is extracted from this transform as well.

int **pose_confidence**
 Confidence/Quality of the pose estimation for the target frame A confidence metric of the tracking [0-100], 0 means that the tracking is lost, 100 means that the tracking can be fully trusted.

struct RecordingState
#include <defines.hpp> Recording structure that contains information about SVO.

Public Members

bool **status**
 status of current frame. May be true for success or false if frame could not be written in the SVO file.

double **current_compression_time**
 compression time for the current frame in ms.

double **current_compression_ratio**
 compression ratio (% of raw size) for the current frame.

double **average_compression_time**

average compression time in ms since beginning of recording.

double **average_compression_ratio**

compression ratio (% of raw size) since beginning of recording.

struct Resolution

#include <Core.hpp> Width and height of an array.

Public Functions

Resolution (size_t w_ = 0, size_t h_ = 0)

size_t **area** ()

Returns the area of the image.

Return The number of pixels of the array.

bool **operator==** (const *Resolution* &that) const

Tests if the given *sl::Resolution* has the same properties.

Return True if the sizes matches.

bool **operator!=** (const *Resolution* &that) const

Tests if the given *sl::Resolution* has different properties.

Return True if the sizes are not equal.

Public Members

size_t **width**

array width in pixels

size_t **height**

array height in pixels

class Rotation

#include <Core.hpp> The class *Rotation* is designed to contains rotation data from the tracking.

It inherits from the generic *sl::Matrix3f*

Inherits from *sl::Matrix3f*

Public Functions

Rotation ()

empty *Rotation* default constructor.

Rotation (const *Rotation* &rotation)

brief *Rotation* copy constructor (deep copy).

Parameters

- rotation: : the *Rotation* to copy.

Rotation (const *Matrix3f* &mat)
brief *Rotation* copy constructor (deep copy).

Parameters

- mat: : the mat to copy.

Rotation (const *Orientation* &orientation)
brief *Rotation* constructor from an *Orientation*. It converts the *Orientation* representation to the *Rotation* one.

Parameters

- orientation: : the *Orientation* to be used.

Rotation (const float angle, const *Translation* &axis)
brief Creates a *Rotation* representing the 3D rotation of angle around an arbitrary 3D axis.

Parameters

- angle: : the rotation angle in rad.
- axis: : the 3D axis to rotate around.

void **setOrientation** (const *Orientation* &orientation)
Sets the *Rotation* from an *Orientation*.

Parameters

- orientation: : the *Orientation* containing the rotation to set.

Orientation **getOrientation** ()
Returns the *Orientation* corresponding to the current *Rotation*.

Return The rotation of the current orientation.

sl::Vector3<float> **getRotationVector** ()
Returns the rotation vector (Rx,Ry,Rz) corresponding to the current *Rotation* (using Rodrigues' transformation).

Return The rotation vector .

void **setRotationVector** (const *sl::Vector3<float>* &vec_rot)
Sets the *Rotation* from a rotation vector (using Rodrigues' transformation).

Parameters

- vec_rot: : the *Rotation* Vector.

class RuntimeParameters
#include <Camera.hpp> Contains all the *Camera::grab()* parameters.

Public Functions

RuntimeParameters (*SENSING_MODE* *sensing_mode_* = SENSING_MODE::SENSING_MODE_STANDARD, bool *enable_depth_* = true, bool *enable_point_cloud_* = true, bool *move_point_cloud_to_world_frame_* = false)

Default constructor, set all parameters to their default and optimized values.

bool **save** (*sl::String filename*)

Saves the current bunch of parameters into a file.

Return true if the file was successfully saved, otherwise false.

Parameters

- *filename*: : the path to the file in which the parameters will be stored.

bool **load** (*sl::String filename*)

Loads the values of the parameters contained in a file.

Return true if the file was successfully loaded, otherwise false.

Parameters

- *filename*: : the path to the file from which the parameters will be loaded.

Public Members

SENSING_MODE **sensing_mode**

Defines the type of disparity map, more info : *sl::SENSING_MODE* definition.

bool **enable_depth**

Defines if the depth map should be computed.

If false, only the images are available.

bool **enable_point_cloud**

Defines if the point cloud should be computed (including XYZRGBA).

bool **move_point_cloud_to_world_frame**

Apply the current pose to the point cloud.

It means that values of point cloud will be defined in world frame (as opposite to the camera frame).

struct section

Public Members

char ***type**

list ***options**

struct size_params

Public Members

int **batch**

int **inputs**

int **h**

int **w**

int **c**

int **index**

int **time_steps**

network **net**

struct **sortable_bbox**

Public Members

int **index**

int **sortable_bbox::class**

float ****probs**

network **net**

char ***filename**

int **classes**

float **elo**

float ***elos**

class **SpatialMappingParameters**

#include <Camera.hpp> Parameters for ZED scanning initialization.

A default constructor is enabled and set to its default parameters.

Note Parameters can be user adjusted.

Public Types

enum **RESOLUTION**

List the spatial mapping resolution presets.

Values:

RESOLUTION_HIGH

Create a detail geometry, requires lots of memory.

RESOLUTION_MEDIUM

Smalls variations in the geometry will disappear, useful for big object

RESOLUTION_LOW

Keeps only huge variations of the geometry , useful outdoor.

enum RANGE

List the spatial mapping depth range presets.

Values:

RANGE_NEAR

Only depth close to the camera will be used by the spatial mapping.

RANGE_MEDIUM

Medium depth range.

RANGE_FAR

Takes into account objects that are far, useful outdoor.

typedef *std::pair*<float, float> **interval**

Public Functions

SpatialMappingParameters (*RESOLUTION* resolution = RESOLUTION_HIGH, *RANGE* range = RANGE_MEDIUM, int max_memory_usage_ = 2048, bool save_texture_ = true)

Default constructor, set all parameters to their default and optimized values.

void **set** (*RESOLUTION* resolution = RESOLUTION_HIGH)

Sets the resolution corresponding to the given `sl::SpatialMappingParameters::RESOLUTION` preset.

Parameters

- resolution: : the desired `sl::SpatialMappingParameters::RESOLUTION`.

void **set** (*RANGE* range = RANGE_MEDIUM)

Sets the maximum value of depth corresponding to the given `sl::SpatialMappingParameters::RANGE` presets.

Parameters

- range: : the desired `sl::SpatialMappingParameters::RANGE`.

bool **save** (*sl::String* filename)

Saves the current bunch of parameters into a file.

Return true if the file was successfully saved, otherwise false.

Parameters

- filename: : the path to the file in which the parameters will be stored.

bool **load** (*sl::String* filename)

Loads the values of the parameters contained in a file.

Return true if the file was successfully loaded, otherwise false.

Parameters

- filename: : the path to the file from which the parameters will be loaded.

Public Members

int **max_memory_usage** = 2048

The maximum CPU memory (in mega bytes) allocated for the meshing process (will fit your configuration in any case).

bool **save_texture** = true

Set to true if you want be able to apply texture to your mesh after its creation.

Note This option will take more memory.

const *interval* **allowed_min** = std::make_pair(0.3f, 10.f)

The minimal depth value allowed by the spatial mapping.

const *interval* **allowed_max** = std::make_pair(2.f, 20.f)

The maximal depth value allowed by the spatial mapping.

interval **range_meter** = std::make_pair(0.7f, 5.f)

Depth integration range in meters. `range_meter.first` should fit `interval::allowed_min`. `range_meter.first` will be set to `sl::Camera::getDepthMinRangeValue` if you do not change it. `range_meter.second` should fit `interval::allowed_max`.

const *interval* **allowed_resolution** = std::make_pair(0.01f, 0.2f)

The resolutions allowed by the spatial mapping.

float **resolution_meter** = 0.03f

Spatial mapping resolution in meters, should fit `interval::allowed_resolution`.

Public Static Functions

static float **get** (*RESOLUTION* resolution = RESOLUTION_HIGH)

Return the resolution corresponding to the given `sl::SpatialMappingParameters::RESOLUTION` preset.

Return The resolution in meter.

Parameters

- `resolution`: : the desired `sl::SpatialMappingParameters::RESOLUTION`.

static float **get** (*RANGE* range = RANGE_MEDIUM)

Return the maximum value of depth corresponding to the given `sl::SpatialMappingParameters::RANGE` presets.

Return The maximum value of depth.

Parameters

- `range`: : the desired `sl::SpatialMappingParameters::RANGE`.

struct **stbi_io_callbacks**

Public Members

int (***read**) (void *user, char *data, int size)

void (***skip**) (void *user, int n)

```
int (*eof) (void *user)
```

struct Texture

#include <Mesh.hpp> Contains information about texture images associated to the *Mesh*.

Public Members

std::string **name**

The name of the file in which the texture is saved.

sl::Mat **texture**

A *sl::Mat* that contains the data of the texture.

unsigned int **indice_gl**

useful for OpenGL binding reference (value not set by the SDK).

class TrackingParameters

#include <Camera.hpp> Parameters for ZED tracking initialization.

A default constructor is enabled and set to its default parameters.

Note Parameters can be user adjusted.

Public Functions

TrackingParameters (*sl::Transform* *init_pos* = *sl::Transform* (), bool *_enable_memory* = true, *sl::String* *_area_path* = *sl::String*())

Default constructor, set all parameters to their default and optimized values.

bool **save** (*sl::String* *filename*)

Saves the current bunch of parameters into a file.

Return true if the file was successfully saved, otherwise false.

Parameters

- *filename*: : the path to the file in which the parameters will be stored.

bool **load** (*sl::String* *filename*)

Loads the values of the parameters contained in a file.

Return true if the file was successfully loaded, otherwise false.

Parameters

- *filename*: : the path to the file from which the parameters will be loaded.

Public Members

sl::Transform **initial_world_transform**

Position of the camera in the world frame when camera is started.

By default it should be identity.

Note The camera frame (defines the reference frame for the camera) is by default positioned at the world frame when tracking is started. Use this *sl::Transform* to place the camera frame in the world frame.
default : Identity matrix

bool **enable_spatial_memory**

This mode enables the camera to learn and remember its surroundings.

This helps correct motion tracking drift and position different cameras relative to each other in space.

Warning : This mode requires few resources to run and greatly improves tracking accuracy. We recommend to leave it on by default. default : true

sl::String **area_file_path**

Area localization mode can record and load (if areaFilePath is specified) a file that describes the surroundings.

Note Loading an area file will start a searching phase during which the camera will try to position itself in the previously learned area.

Warning : The area file describes a specific location. If you are using an area file describing a different location, the tracking function will continuously search for a position and may not find a correct one.

Warning The '.area' file can only be used with the same depth mode (sl::MODE) as the one used during area recording. default : NULL

class Transform

#include <Core.hpp> The class *Transform* contains a 4x4 matrix that specifically contains a rotation 3x3 and a 3x1 translation.

It then contains the orientation as well. It can be used to create any type of Matrix4x4 or *sl::Matrix4f* that must be specifically used for handling a rotation and position information (OpenGL, Tracking...) It inherits from the generic *sl::Matrix4f*

Inherits from *sl::Matrix4f*

Public Functions

Transform()

brief *Transform* default constructor.

Transform(const Transform &motion)

brief *Transform* copy constructor (deep copy).

Parameters

- motion: : the *Transform* to copy.

Transform(const Matrix4f &mat)

brief *Transform* copy constructor (deep copy).

Parameters

- mat: : the *Matrix4f* to copy.

Transform(const Rotation &rotation, const Translation &translation)

brief *Transform* constructor from a *Rotation* and a *Translation*.

Parameters

- rotation: : the *Rotation* to copy.
- translation: : the *Translation* to copy.

Transform(const Orientation &orientation, const Translation &translation)

brief *Transform* constructor from an *Orientation* and a *Translation*.

Parameters

- `orientation`: : the *Orientation* to copy.
- `translation`: : the *Translation* to copy.

void **setRotation** (const *Rotation* &rotation)
Sets the rotation of the current *Transform* from an *Rotation*.

Parameters

- `rotation`: : the *Rotation* to be used.

Rotation **getRotation** () const
Returns the *Rotation* of the current *Transform*.

Return The *Rotation* created from the *Transform* values.

Warning The given *Rotation* contains a copy of the *Transform* values. Not references.

void **setTranslation** (const *Translation* &translation)
Sets the translation of the current *Transform* from an *Translation*.

Parameters

- `translation`: : the *Translation* to be used.

Translation **getTranslation** () const
Returns the *Translation* of the current *Transform*.

Return The *Translation* created from the *Transform* values.

Warning The given *Translation* contains a copy of the *Transform* values. Not references.

void **setOrientation** (const *Orientation* &orientation)
Sets the orientation of the current *Transform* from an *Orientation*.

Parameters

- `orientation`: : the *Orientation* to be used.

Orientation **getOrientation** () const
Returns the *Orientation* of the current *Transform*.

Return The *Orientation* created from the *Transform* values.

Warning The given *Orientation* contains a copy of the *Transform* values. Not references.

sl::Vector3<float> **getRotationVector** ()
Returns the vector *Rotation* (3x1) of the *Transform*.

Return The rotation value for each axis (rx,ry,rz).

void **setRotationVector** (const *sl::Vector3<float>* &vec_rot)
Sets the *Rotation* 3x3 of the *Transform* with a 3x1 rotation vector (using Rodrigues' transformation).

Parameters

- `vec_rot`: : vector that contains the rotation value for each axis (rx,ry,rz).

class **Translation**

#include <Core.hpp> The class *Translation* is designed to contains translation data from the tracking. *sl::Translation* is a vector as [tx, ty, tz]. You can access the data with the 't' ptr or by element name as : tx, ty, tz <-> | 0 1 2 |.

Inherits from `Vector3< float >`

Public Functions

Translation()

empty *Translation* default constructor.

Translation(const Translation &translation)

brief *Translation* copy constructor (deep copy).

Parameters

- `translation`: : the *Translation* to copy.

Translation(float t1, float t2, float t3)

brief *Translation* constructor.

Parameters

- `t1`: : the x translation.
- `t2`: : the y translation.
- `t3`: : the z translation.

Translation(Vector3<float> in)

brief *Translation* constructor.

Parameters

- `in`: : vector.

Translation **operator*** (const *Orientation* &mat) const

brief Multiplication operator by an *Orientation*.

Return The current *Translation* after being multiplied by the orientation.

Parameters

- `mat`: : *Orientation*.

void **normalize()**

Normalizes the current translation.

float &**operator()** (int x)

Get the value at specific position in the *Translation*.

Return The value at the x position.

Parameters

- `x`: : the position of the value

Public Static Functions

static *Translation* **normalize** (**const** *Translation* &*tr*)

brief Get the normalized version of a given *Translation*.

Return An other *Translation* object, which is equal to tr.normalize.

Parameters

- *tr*: : the *Translation* to be used.

struct tree

Public Members

int ***leaf**

int **n**

int ***parent**

int ***child**

int ***group**

char ****name**

int **groups**

int ***group_size**

int ***group_offset**

struct update_args

Public Members

int **batch**

float **learning_rate**

float **momentum**

float **decay**

int **adam**

float **B1**

float **B2**

float **eps**

int **t**

struct yolo_obj

Public Members

char **darknet_path**[1024]

char ****names**

float **nms**

box ***boxes**

float ****probs**

network **net**

namespace **ctypes**

namespace **darknet**

Functions

load_meta (*ff*)

load_net (*cfg cfg, weights weights*)

load_img (*ff*)

letterbox_img (*im im, w w, h h*)

predict (*net net, im im*)

classify (*net net, meta meta, im im*)

detect (*net net, meta meta, im im*)

Variables

lib = CDLL("/home/pjreddie/documents/darknet/libdarknet.so", RTLD_GLOBAL)

argtypes

restype

net = load_net("cfg/densenet.cfg", "/home/pjreddie/trained/densenet201.weights")

im = load_img("data/wolf.jpg")

meta = load_meta("cfg/imagenet1k.data")

r = classify(net, meta, im)

namespace **DescribeSceneSrv**

Functions

service_callback ()

namespace **example**

Variables

```
string example.darknet_path = './darknet'
string example.datacfg = 'cfg/coco.data'
string example.cfgfile = 'cfg/tiny-yolo.cfg'
string example.weightfile = './tiny-yolo.weights'
string example.filename = "/home/roboy/outputRoboy.mp4"
float example.thresh = 0.24
float example.hier_thresh = 0.5
cam = cv2.VideoCapture(filename)
ret_val
img = img.transpose(2,0,1)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
outVideo = cv2.VideoWriter('outputRoboySkyfall.mp4',fourcc, 20.0, (800,533))
ok
frame = imutils.resize(img, width=800)
c
h
w
data = img.ravel()/255.0
outputs = pyyolo.detect(w, h, c, data, thresh, hier_thresh)
tuple example.p1 = (output['left'],output['top'])
tuple example.p2 = (output['right'],output['bottom'])
```

namespace face_detection

This module processes realsense camera input and runs face detection, alignment and pose estimation.

Module Tasks:

- Main loop to process realsense camera input
- Run Face Detection based MTCNN for Joint Face Detection & Alignment for each frame
- Track Face over frames
- Calculate Face Position in 3D coordinates
- Send ROS msg containing face area, key points and face pose and unique ID for each detected face

Current Workarounds:

- Tracking not implemented (no unique face id provided)
- 3D coordinates not implemented (face region used as distance measure)
- Function for Face Recognition also implemented in this module for simplicity (to be put into another module)
- No ROS communication

Functions

detect_face_and_landmarks_mtcnn (*img img*)

Function to do face detection and alignment on an image.

Run face detection on the full input image using a MTCNN for Joint Detection and Alignment from here: https://github.com/pangyupo/mxnet_mtcnn_face_detection

Return Bounding boxes bb and landmark points for eyes, nose and mouth edges.

Parameters

- *img*: The RGB image

align_face_mtcnn (*img img, bb bb*)

Function to align detected faces.

The current implementation crops the picture given a face region. We do not use actual alignment because performance increase for face recognition is marginal and only slows down realtime performance as is also argued here: <https://github.com/davidsandberg/facenet/issues/93>

Return Returns the cropped face region.

Parameters

- *img*: The RGB image
- *bb*: The bounding box of a face as tuple (x1, y1, x2, y2)

draw_rects (*img img, bbs bbs, resize_factor resize_factor = 1*)

Function to draw bounding boxes in a picture.

Given an image, the bounding boxes for the corresponding face regions are drawn. Additionally a *resize_factor* is used if the bounding boxes were calculated on a scaled version of the input image. Default value of the *resize_factor* is 1, meaning bounding boxes were calculated on the same image size.

Return Image overlayed with the bounding boxes.

Parameters

- *img*: The RGB image
- *bbs*: An array of bounding boxes of a face as tuple (x1, y1, x2, y2) factor to scale up bounding box size if calculated on different picture scale.

draw_landmarks (*img img, points points, resize_factor resize_factor*)

Function to draw feature points in a picture.

Given an image, the feature points for the corresponding faces are drawn. Additionally a *resize_factor* is used if the feature points were calculated on a scaled version of the input image. Default value of the *resize_factor* is 1, meaning the feature points were calculated on the same image size.

Return Image overlayed with the feature points

Parameters

- *img*: The RGB image
- *points*: An array containing arrays of feature points of a face factor to scale up bounding box size if calculated on different picture scale.

get_closest_face (*bbs bbs*)

Returns the closest face of all detected faces.

Current implementation uses bounding box size to compare proximity

Return The array index of the biggest bounding box.

Parameters

- `bbs`: An array of bounding boxes of a face as tuple (x1, y1, x2, y2).

face_detected (*bbs bbs*)

Checks whether a face is visible within certain distance.

Current implementation uses bounding box to check for proximity. Key value defined in `FACE_AREA`.

Return True if a face is close enough, False otherwise

Parameters

- `bbs`: An array of bounding boxes of a face as tuple (x1, y1, x2, y2).

recognize_face (*face_img face_img, session session, classifier classifier*)

Identifies a face using Facenet.

TODO: To be moved into other module The function calculates the 128D embeddings of a given face using facenet in this implementation: <https://github.com/davidsandberg/facenet> Then the embeddings are run through a SVM classifier to identify the face. The cropped image of the face region.

Return Return the name of the face.

Parameters

- `session`: The tensorflow session with FaceNet already loaded
- `classifier`: The SVM classifier already loaded

load_model (*model_dir model_dir, model_meta model_meta, model_content model_content*)

Function to load a tensorflow model.

TODO: To be moved into other module

Return Returns a tensorflow session

Parameters

- `model_dir`: model directory
- `model_meta`: meta file
- `model_content`: checkpoint file

get_model_filenames (*model_dir model_dir*)

Helper Function to load a tensorflow model.

TODO: To be moved into other module The function finds the meta_file and checkpoint within a given path

Return Returns meta_file and checkpoint

Parameters

- `model_dir`: Path where the model is stored

Variables

```
int face_detection.EXPECT_SIZE = 160
```

```
int face_detection.FACE_AREA = 1500
```

```

int face_detection.x_pixel = 640
    Entry Point to run face detection.

    Loads all data and processes realsense camera input in a loop.

int face_detection.y_pixel = 480

float face_detection.resize_factor = 0.5

bool face_detection.face_nearby = False

int face_detection.no_face_detect_counter = 0

dev = pyrs.Device( device_id=0, streams=[ pyrs.ColourStream(width=x_pixel, height=y_pixel, fps=30), pyrs.DepthStream() ])

sess = tf.Session(config=tf.ConfigProto(log_device_placement=False))

pnet
rnet
onet

int face_detection.minsize = 20

list face_detection.threshold = [0.6, 0.7, 0.7]

float face_detection.factor = 0.709

string face_detection.tree_model = "models/Tree/own.mod"

string face_detection.svm_model = "models/SVM/svm_lfw.mod"

clf = pickle.load(open(tree_model, "rb"))

string face_detection.model_dir = 'models/facenet'

meta_file
ckpt_file

session = load_model(model_dir, meta_file, ckpt_file)

graph = tf.get_default_graph()

image_batch = graph.get_tensor_by_name("input:0")

phase_train_placeholder = graph.get_tensor_by_name("phase_train:0")

embeddings = graph.get_tensor_by_name("embeddings:0")

c = cv2.cvtColor(dev.colour, cv2.COLOR_RGB2BGR)

int face_detection.d = dev.depth*dev.depth_scale*1000

img = cv2.resize(c, (int(resize_factor * x_pixel), int( resize_factor * y_pixel)))

d_img = cv2.resize(d, (int(resize_factor * x_pixel), int( resize_factor * y_pixel)))

total_boxes

points

start_recognize_face = false

draw = draw_rects(c.copy(), total_boxes, resize_factor)

```

```
namespace FaceDetect
1. Face is detected using Dlib library
2. 68 landmarks are located on the face and circles are drawn over the landmarks.
3. The position(rect) of the face is sent to the main process through the RectQueue.
4. Frame queue is as a future reference to send data to the main process
```

Functions

StartDetection (*CameraQueue CameraQueue, FrameQueue FrameQueue, RectQueue RectQueue, FacepointQueue FacepointQueue, SpeakerQueue SpeakerQueue*)

sample (*probs probs*)

c_array (*ctype ctype, values values*)

detect (*net net, meta meta, image image, thresh thresh = .5, hier_thresh hier_thresh = .5, nms nms = .45*)

draw_results (*res res, img img*)

Initialize ()

detectObjects (*frame frame, detect_net detect_net, detect_meta detect_meta*)

Variables

lib = `CDLL("../darknet/libdarknet.so", RTLD_GLOBAL)`

argtypes

restype

predict = `lib.network_predict_p`

make_boxes = `lib.make_boxes`

free_ptrs = `lib.free_ptrs`

num_boxes = `lib.num_boxes`

make_probs = `lib.make_probs`

detect = `lib.network_predict_p`

reset_rnn = `lib.reset_rnn`

load_net = `lib.load_network_p`

free_image = `lib.free_image`

letterbox_image = `lib.letterbox_image`

load_meta = `lib.get_metadata`

load_image = `lib.load_image_color`

predict_image = `lib.network_predict_image`

network_detect = `lib.network_detect`

int FaceDetect.DONE = 0

namespace FindObjectSrv

Functions

```
service_callback ()
```

Variables

```
int FindObjectSrv.i = 1
request = awaitwebsocket.recv()
type = json.loads(request)["type"]
FoundObjects = ObjectRects.getQueue();
bool FindObjectSrv.found = False
dictionary FindObjectSrv.coordinates = {}
dictionary FindObjectSrv.answer = {'found': found, 'coordinates': coordinates}
level
```

namespace FreezeModel

Functions

```
freeze_graph (model_folder model_folder)
```

Variables

```
dir = os.path.dirname(os.path.realpath(__file__))
parser = argparse.ArgumentParser()
type
str
help
args = parser.parse_args()
```

namespace make_labels

Functions

```
make_labels (s s)
```

Variables

```
string make_labels.font = 'futura-normal'
```

namespace multiprocessing

Functions

TestProcess (*number number*)

detectFaces (*FrameQueue FrameQueue, RectQueue RectQueue*)

tracking (*RectQueue RectQueue, TrackQueue TrackQueue*)

Variables

list multiprocessing.procs = []

FrameQueue = Queue();

RectQueue = Queue();

TrackQueue = Queue();

detectFaceProc = Process(target=detectFaces,args=(FrameQueue,RectQueue,))

trackProc = Process(target=tracking,args=(RectQueue,TrackQueue,))

namespace MultiTracking

1. Coordinates of the detected **object** **is** send though Rects Queue.
2. Currently, MIL tracking algorithm has been employed.
3. A rectangle **is** drawn over the tracked **object** and is sent to the main process,
 ↳ through TrackQueue

Functions

StartTracking (*RectsQueue RectsQueue, TrackQueue TrackQueue*)
 ↳ Drawn through the Rects Queue.
 ↳ MIL MultiTracking **is** used through OpenCV Contrib
 Drawn rectangle **is** passed over the TrackQueue

namespace NewFacialFeaturesMsg

Functions

service_callback ()

namespace ObjectRecognition

1. This is a python wrapper for the YOLO implementation in C.
2. uses Ctypes as a way to access C module.
3. libdarknet.so is a pre compiled library which works only on Linux!

Functions

sample (*probs probs*)

c_array (*ctype ctype, values values*)

detect (*net net, meta meta, image image, thresh thresh = .5, hier_thresh hier_thresh = .5, nms nms = .45*)

draw_results (*res res, img img*)

```
Initialize()
detectObjects (frame frame)
```

Variables

```
int ObjectRecognition.detect_net = 0
int ObjectRecognition.detect_meta = 0
lib = CDLL("../darknet/libdarknet.so", RTLD_GLOBAL)
argtypes
restype
predict = lib.network_predict_p
make_boxes = lib.make_boxes
free_ptrs = lib.free_ptrs
num_boxes = lib.num_boxes
make_probs = lib.make_probs
detect = lib.network_predict_p
reset_rnn = lib.reset_rnn
load_net = lib.load_network_p
free_image = lib.free_image
letterbox_image = lib.letterbox_image
load_meta = lib.get_metadata
load_image = lib.load_image_color
predict_image = lib.network_predict_image
network_detect = lib.network_detect
int ObjectRecognition.DONE = 0
```

nameSpace: RecogniseFace

1. Module responsible **for** Face Recognition
2. This uses a model already trained on LWF, to extract facial features
3. Another model of members **from Roboy** are trained
4. We use a SVM classifier to classify the detected face to match **with** the model_ of trained faces

Functions

```
recogniseFace (RectsQueue RectsQueue)
align_face_dlib (image image, face_box face_box, landmarkIndices landmarkIndices =
AlignDlib.OUTER_EYES_AND_NOSE)
load_model (model_dir model_dir, model_meta model_meta, model_content model_content)
    Function to load a tensorflow model.
    TODO: To be moved into other module
```

Return Returns a tensorflow session

Parameters

- `model_dir`: model directory
- `model_meta`: meta file
- `model_content`: checkpoint file

get_model_filenames (*model_dir model_dir*)

Helper Function to load a tensorflow model.

TODO: To be moved into other module The function finds the meta_file and checkpoint within a given path

Return Returns meta_file and checkpoint

Parameters

- `model_dir`: Path where the model is stored

get_image_paths_and_labels (*dataset dataset*)

get_image_paths (*facedir facedir*)

get_dataset (*paths paths, has_class_directories has_class_directories = True*)

load_data (*image_paths image_paths, do_random_crop do_random_crop, do_random_flip do_random_flip, image_size image_size, do_prewhiten do_prewhiten = True*)

processImage (*img img, do_random_crop do_random_crop, do_random_flip do_random_flip, image_size image_size, do_prewhiten do_prewhiten = True*)

to_rgb (*img img*)

prewhiten (*x x*)

crop (*image image, random_crop random_crop, image_size image_size*)

flip (*image image, random_flip random_flip*)

train (*session session*)

Variables

int RecogniseFace.IMAGE_SIZE = 160

int RecogniseFace.EXPECT_SIZE = 160

namespace RoboyVision

1. This **is** the main module.
2. Each other components are created **as** seperate processes **and** spawned.
3. This also creates Message queues **and** passes them onto different processes

Functions

detectFaces (*CameraQueue CameraQueue, FrameQueue FrameQueue, RectQueue RectQueue, FacePointQueue FacePointQueue, SpeakerQueue SpeakerQueue*)

tracking (*RectQueue RectQueue, TrackQueue TrackQueue*)

speakerDetect (*FacePointQueue FacePointQueue, SpeakerQueue SpeakerQueue, FrameQueue FrameQueue, VisualQueue VisualQueue*)

recogniseFace (*RectsQueue RectsQueue*)

visualizer (*CameraQueue CameraQueue, RectQueue RectQueue, FacePointQueue FacePointQueue, SpeakerQueue SpeakerQueue, FrameQueue FrameQueue, VisualQueue VisualQueue*)

ObjectRecognise (*CameraQueue CameraQueue, ObjectsQueue ObjectsQueue*)

Variables

list RoboyVision.procs = []

CameraQueue = Queue()

FrameQueue = Queue()

RectQueue = Queue()

TrackQueue = Queue()

VisualQueue = Queue()

SpeakerQueue = Queue()

FacePointQueue = Queue()

ObjectsQueue = Queue()

detectFaceProc = \

target

detectFaces

args

SpeakerProc = \

speakerDetect

namespace RosMsgUtil

1. This module has Util functions to send out ROS messages.
2. The send functions here are called **from different** modules.

Functions

AdvertiseNewFacialFeatures ()

AdvertiseFaceCoordinates ()

AdvertiseFindObject ()

AdvertiseDescribeScene ()

AdvertiseLookAtSpeaker ()

AdvertiseContinously ()

SendNewFacialFeatures (*ff ff, speaking speaking, i i*)

SendFaceCoordinates (*id id, speaking speaking, position position, i i*)

FindObject (*type type, i i*)

DescribeScene (*i i*)

```
LookAtSpeaker (i i)
```

```
ReceiveServiceRequests ()
```

Variables

```
ws = websocket.WebSocket();
```

namespace setup

Variables

```
module = Extension('pyyolo', library_dirs=['.'], libraries=['yolo'], include_dirs=[numpy.get_include(), './darknet/include'], sou
```

```
name
```

```
version
```

```
description
```

```
ext_modules
```

namespace setup_gpu

Variables

```
module = Extension('pyyolo', library_dirs=['.', '/usr/local/cuda/lib64', '/usr/local/'], #libraries=['yolo', 'cuda', 'cudart', 'cublas
```

```
name
```

```
version
```

```
description
```

```
ext_modules
```

namespace sl

ENUM to string

```
static std::string resolution2str (RESOLUTION res)
```

```
static std::string statusCode2str (SELF_CALIBRATION_STATE stat)
```

```
static DEPTH_MODE str2mode (std::string mode)
```

```
static std::string depthMode2str (DEPTH_MODE mode)
```

```
static std::string sensingMode2str (SENSING_MODE mode)
```

```
static std::string unit2str (UNIT unit)
```

```
static UNIT str2unit (std::string unit)
```

```
static std::string trackingState2str (TRACKING_STATE state)
```

```
static std::string spatialMappingState2str (SPATIAL_MAPPING_STATE state)
```

Typedefs

```
typedef float float1
typedef Vector2<float> float2
typedef Vector3<float> float3
typedef Vector4<float> float4
typedef unsigned char uchar1
typedef Vector2<unsigned char> uchar2
typedef Vector3<unsigned char> uchar3
typedef Vector4<unsigned char> uchar4
typedef double double1
typedef Vector2<double> double2
typedef Vector3<double> double3
typedef Vector4<double> double4
typedef unsigned int uint1
typedef Vector2<unsigned int> uint2
typedef Vector3<unsigned int> uint3
typedef Vector4<unsigned int> uint4
```

Enums

enum MEM

List available memory type.

Values:

```
MEM_CPU = 1
    CPU Memory (Processor side).
MEM_GPU = 2
    GPU Memory (Graphic card side).
```

enum COPY_TYPE

List available copy operation on *Mat*.

Values:

```
COPY_TYPE_CPU_CPU
    copy data from CPU to CPU.
COPY_TYPE_CPU_GPU
    copy data from CPU to GPU.
COPY_TYPE_GPU_GPU
    copy data from GPU to GPU.
COPY_TYPE_GPU_CPU
    copy data from GPU to CPU.
```

enum MAT_TYPE

List available *Mat* formats.

Values:

MAT_TYPE_32F_C1

float 1 channel.

MAT_TYPE_32F_C2

float 2 channels.

MAT_TYPE_32F_C3

float 3 channels.

MAT_TYPE_32F_C4

float 4 channels.

MAT_TYPE_8U_C1

unsigned char 1 channel.

MAT_TYPE_8U_C2

unsigned char 2 channels.

MAT_TYPE_8U_C3

unsigned char 3 channels.

MAT_TYPE_8U_C4

unsigned char 4 channels.

enum RESOLUTION

List available video resolutions.

Warning Since v1.0, VGA mode has been updated to WVGA (from 640*480 to 672*376) and requires a firmware update to function (≥ 1142). Firmware can be updated in the ZED Explorer.

Warning NVIDIA Jetson boards do not support all ZED video resolutions and framerates. For more information, please read the on-line API documentation.

Values:

RESOLUTION_HD2K

2208*1242, available framerates: 15 fps.

RESOLUTION_HD1080

1920*1080, available framerates: 15, 30 fps.

RESOLUTION_HD720

1280*720, available framerates: 15, 30, 60 fps.

RESOLUTION_VGA

672*376, available framerates: 15, 30, 60, 100 fps.

RESOLUTION_LAST

enum CAMERA_SETTINGS

List available camera settings for the ZED camera (contrast, hue, saturation, gain...).

Each enum defines one of those settings.

Values:

CAMERA_SETTINGS_BRIGHTNESS

Defines the brightness control. Affected value should be between 0 and 8.

CAMERA_SETTINGS_CONTRAST

Defines the contrast control. Affected value should be between 0 and 8.

CAMERA_SETTINGS_HUE

Defines the hue control. Affected value should be between 0 and 11.

CAMERA_SETTINGS_SATURATION

Defines the saturation control. Affected value should be between 0 and 8.

CAMERA_SETTINGS_GAIN

Defines the gain control. Affected value should be between 0 and 100 for manual control. If ZED_EXPOSURE is set to -1, the gain is in auto mode too.

CAMERA_SETTINGS_EXPOSURE

Defines the exposure control. A -1 value enable the AutoExposure/AutoGain control, as the boolean parameter (default) does. Affected value should be between 0 and 100 for manual control.

CAMERA_SETTINGS_WHITEBALANCE

Defines the color temperature control. Affected value should be between 2800 and 6500 with a step of 100. A value of -1 set the AWB (auto white balance), as the boolean parameter (default) does.

CAMERA_SETTINGS_AUTO_WHITEBALANCE

Defines the status of white balance (automatic or manual). A value of 0 disable the AWB, while 1 activate it.

CAMERA_SETTINGS_LAST**enum SELF_CALIBRATION_STATE**

Status for self calibration.

Since v0.9.3, self-calibration is done in background and start in the *sl::Camera::open* or Reset function. You can follow the current status for the self-calibration any time once ZED object has been construct.

Values:

SELF_CALIBRATION_STATE_NOT_STARTED

Self calibration has not run yet (no *sl::Camera::open* or *sl::Camera::resetSelfCalibration* called).

SELF_CALIBRATION_STATE_RUNNING

Self calibration is currently running.

SELF_CALIBRATION_STATE_FAILED

Self calibration has finished running but did not manage to get accurate values. Old parameters are taken instead.

SELF_CALIBRATION_STATE_SUCCESS

Self calibration has finished running and did manage to get accurate values. New parameters are set.

SELF_CALIBRATION_STATE_LAST**enum DEPTH_MODE**

List available depth computation modes.

Values:

DEPTH_MODE_NONE

This mode does not compute any depth map. Only rectified stereo images will be available.

DEPTH_MODE_PERFORMANCE

Fastest mode for depth computation.

DEPTH_MODE_MEDIUM

Balanced quality mode. Depth map is robust in any environment and requires medium resources for computation.

DEPTH_MODE_QUALITY

Best quality mode. Requires more compute power.

DEPTH_MODE_LAST

enum SENSING_MODE

List available depth sensing modes.

Values:

SENSING_MODE_STANDARD

This mode outputs ZED standard depth map that preserves edges and depth accuracy. Applications example: Obstacle detection, Automated navigation, People detection, 3D reconstruction.

SENSING_MODE_FILL

This mode outputs a smooth and fully dense depth map. Applications example: AR/VR, Mixed-reality capture, Image post-processing.

SENSING_MODE_LAST

enum UNIT

Enumerate for available metric unit of the depth.

Values:

UNIT_MILLIMETER

UNIT_CENTIMETER

UNIT_METER

UNIT_INCH

UNIT_FOOT

UNIT_LAST

enum COORDINATE_SYSTEM

List available coordinates systems for positional tracking and points cloud representation.

Positional tracking is provided in the given coordinates system.

Values:

COORDINATE_SYSTEM_IMAGE

Standard coordinates system in computer vision. Used in OpenCV : see here : http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

COORDINATE_SYSTEM_LEFT_HANDED_Y_UP

Left-Handed with Y up and Z forward. Used in Unity with DirectX.

COORDINATE_SYSTEM_RIGHT_HANDED_Y_UP

Right-Handed with Y pointing up and Z backward. Used in OpenGL.

COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP

Right-Handed with Z pointing up and Y forward. Used in 3DSMax.

COORDINATE_SYSTEM_LEFT_HANDED_Z_UP

Left-Handed with Z axis pointing up and X forward. Used in Unreal Engine.

COORDINATE_SYSTEM_LAST

enum MEASURE

List retrievable measures.

Values:

MEASURE_DISPARITY

Disparity map, 1 channel, FLOAT.

MEASURE_DEPTH

Depth map, 1 channel, FLOAT.

MEASURE_CONFIDENCE

Certainty/confidence of the disparity map, 1 channel, FLOAT.

MEASURE_XYZ

Point cloud, 4 channels, FLOAT, channel 4 is empty.

MEASURE_XYZRGBA

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in R-G-B-A order.

MEASURE_XYZBGRA

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in B-G-R-A order.

MEASURE_XYZARGB

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in A-R-G-B order.

MEASURE_XYZABGR

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in A-B-G-R order.

MEASURE_LAST**enum VIEW**

List available views.

Values:

VIEW_LEFT

Rectified left image.

VIEW_RIGHT

Rectified right image.

VIEW_LEFT_GRAY

Rectified left grayscale image.

VIEW_RIGHT_GRAY

Rectified right grayscale image.

VIEW_LEFT_UNRECTIFIED

Raw left image.

VIEW_RIGHT_UNRECTIFIED

Raw right image.

VIEW_LEFT_UNRECTIFIED_GRAY

Raw left grayscale image.

VIEW_RIGHT_UNRECTIFIED_GRAY

Raw right grayscale image.

VIEW_SIDE_BY_SIDE

Left and right image (the image width is therefore doubled).

VIEW_DEPTH

Normalized depth image.

VIEW_CONFIDENCE

Normalized confidence image.

VIEW_LAST

enum DEPTH_FORMAT

List available file formats for saving depth maps.

Values:

DEPTH_FORMAT_PNG

PNG image format in 16bits. 32bits depth is mapped to 16bits color image to preserve the consistency of the data range.

DEPTH_FORMAT_PFM

stream of bytes, graphic image file format.

DEPTH_FORMAT_PGM

gray-scale image format.

DEPTH_FORMAT_LAST

enum POINT_CLOUD_FORMAT

List available file formats for saving point clouds.

Stores the spatial coordinates (x,y,z) of each pixel and optionally its RGB color.

Values:

POINT_CLOUD_FORMAT_XYZ_ASCII

Generic point cloud file format, without color information.

POINT_CLOUD_FORMAT_PCD_ASCII

Point Cloud Data file, with color information.

POINT_CLOUD_FORMAT_PLY_ASCII

PoLYgon file format, with color information.

POINT_CLOUD_FORMAT_VTK_ASCII

Visualization ToolKit file, without color information.

POINT_CLOUD_FORMAT_LAST

enum TRACKING_STATE

List the different states of positional tracking.

Values:

TRACKING_STATE_SEARCHING

The camera is searching for a previously known position to locate itself.

TRACKING_STATE_OK

Positional tracking is working normally.

TRACKING_STATE_OFF

Positional tracking is not enabled.

TRACKING_STATE_FPS_TOO_LOW

Effective FPS is too low to give proper results for motion tracking. Consider using PERFORMANCES parameters (DEPTH_MODE_PERFORMANCE, low camera resolution (VGA,HD720))

TRACKING_STATE_LAST

enum AREA_EXPORT_STATE

List the different states of spatial memory area export.

Values:

AREA_EXPORT_STATE_SUCCESS

The spatial memory file has been successfully created.

AREA_EXPORT_STATE_RUNNING

The spatial memory is currently written.

AREA_EXPORT_STATE_NOT_STARTED

The spatial memory file exportation has not been called.

AREA_EXPORT_STATE_FILE_EMPTY

The spatial memory contains no data, the file is empty.

AREA_EXPORT_STATE_FILE_ERROR

The spatial memory file has not been written because of a wrong file name.

AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED

The spatial memory learning is disable, no file can be created.

AREA_EXPORT_STATE_LAST**enum REFERENCE_FRAME**

Define which type of position matrix is used to store camera path and pose.

Values:

REFERENCE_FRAME_WORLD

The transform of *sl::Pose* will contains the motion with reference to the world frame (previously called PATH).

REFERENCE_FRAME_CAMERA

The transform of *sl::Pose* will contains the motion with reference to the previous camera frame (previously called POSE).

REFERENCE_FRAME_LAST**enum SPATIAL_MAPPING_STATE**

Gives the spatial mapping state.

Values:

SPATIAL_MAPPING_STATE_INITIALIZING

The spatial mapping is initializing.

SPATIAL_MAPPING_STATE_OK

The depth and tracking data were correctly integrated in the fusion algorithm.

SPATIAL_MAPPING_STATE_NOT_ENOUGH_MEMORY

The maximum memory dedicated to the scanning has been reach, the mesh will no longer be updated.

SPATIAL_MAPPING_STATE_NOT_ENABLED

Camera::enableSpatialMapping() wasn't called (or the scanning was stopped and not relaunched).

SPATIAL_MAPPING_STATE_FPS_TOO_LOW

Effective FPS is too low to give proper results for spatial mapping. Consider using PERFORMANCES parameters (DEPTH_MODE_PERFORMANCE, low camera resolution (VGA,HD720), spatial mapping low resolution)

SPATIAL_MAPPING_STATE_LAST**enum SVO_COMPRESSION_MODE**

List available compression modes for SVO recording.

sl::SVO_COMPRESSION_MODE_LOSSLESS is an improvement of previous lossless compression (used in ZED Explorer), even if size may be bigger, compression time is much faster.

Values:

SVO_COMPRESSION_MODE_RAW

RAW images, no compression.

SVO_COMPRESSION_MODE_LOSSLESS

new Lossless, with PNG/ZSTD based compression : avg size = 42% of RAW).

SVO_COMPRESSION_MODE_LOSSY

new Lossy, with JPEG based compression : avg size = 22% of RAW).

SVO_COMPRESSION_MODE_LAST

enum MESH_FILE_FORMAT

Values:

MESH_FILE_PLY

Contains only vertices and faces.

MESH_FILE_PLY_BIN

Contains only vertices and faces, encoded in binary.

MESH_FILE_OBJ

Contains vertices, normals, faces and textures informations if possible.

MESH_FILE_LAST

enum MESH_TEXTURE_FORMAT

Values:

MESH_TEXTURE_RGB

MESH_TEXTURE_RGBA

MESH_TEXTURE_LAST

enum ERROR_CODE

List error codes in the ZED SDK.

Values:

SUCCESS

Standard code for successful behavior.

ERROR_CODE_FAILURE

Standard code for unsuccessful behavior.

ERROR_CODE_NO_GPU_COMPATIBLE

No GPU found or CUDA capability of the device is not supported.

ERROR_CODE_NOT_ENOUGH_GPUMEM

Not enough GPU memory for this depth mode please try a faster mode (such as PERFORMANCE mode).

ERROR_CODE_CAMERA_NOT_DETECTED

The ZED camera is not plugged or detected.

ERROR_CODE_INVALID_RESOLUTION

For Jetson only, resolution not yet supported (USB3.0 bandwidth).

ERROR_CODE_LOW_USB_BANDWIDTH

This issue can occurs when you use multiple ZED or a USB 2.0 port (bandwidth issue).

ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE

ZED calibration file is not found on the host machine. Use ZED Explorer or ZED Calibration to get one.

ERROR_CODE_INVALID_SVO_FILE

The provided SVO file is not valid.

ERROR_CODE_SVO_RECORDING_ERROR

An recorder related error occurred (not enough free storage, invalid file).

ERROR_CODE_INVALID_COORDINATE_SYSTEM

The requested coordinate system is not available.

ERROR_CODE_INVALID_FIRMWARE

The firmware of the ZED is out of date. Update to the latest version.

ERROR_CODE_NOT_A_NEW_FRAME

in grab() only, the current call return the same frame as last call. Not a new frame.

ERROR_CODE_CUDA_ERROR

in grab() only, a CUDA error has been detected in the process. Activate verbose in *sl::Camera::open* for more info.

ERROR_CODE_CAMERA_NOT_INITIALIZED

in grab() only, ZED SDK is not initialized. Probably a missing call to *sl::Camera::open*.

ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE

your NVIDIA driver is too old and not compatible with your current CUDA version.

ERROR_CODE_INVALID_FUNCTION_CALL

the call of the function is not valid in the current context. Could be a missing call of *sl::Camera::open*.

ERROR_CODE_CORRUPTED_SDK_INSTALLATION

The SDK wasn't able to load its dependencies, the installer should be launched.

ERROR_CODE_LAST

Functions

SLSTEREO_EXPORT_DLL bool sl::saveDepthAs(sl::Camera & zed, sl::DEPTH_FORMAT format, s

Writes the current depth map into a file.

Return False if something wrong happen, else return true.

Parameters

- **zed:** : the current camera object.
- **format:** : the depth file format you desired.
- **name:** : the name (path) in which the depth will be saved.
- **factor:** : only for PNG and PGM, apply a gain to the depth value (default 1.). The maximum value is 65536, so you can set the Camera::setDepthClampValue to 20000 and give a factor to 3, Do not forget to scale (by 1./factor) the pixel value to get the real depth. The occlusions are represented by 0.

SLSTEREO_EXPORT_DLL bool sl::savePointCloudAs(sl::Camera & zed, sl::POINT_CLOUD_FORMA

Writes the current point cloud into a file.

Return False if something wrong happen, else return true.

Note The color is not saved for XYZ and VTK files.

Parameters

- `zed`: : the current camera object.
- `format`: : the point cloud file format you desired.
- `name`: : the name (path) in which the point cloud will be saved.
- `with_color`: : indicates if the color must be saved (default false). Not available for XYZ and VTK.
- `keep_occluded_point`: : indicates if the non available data should be saved and set to 0 (default false), if set to true this give a Point Cloud with a size = height * width.

MEM operator | (*MEM* *a*, *MEM* *b*)

`static cudaError __cudaSafeCall (cudaError err, const char *func, const char *file, const int line)`

`static std::string errorCode2str (ERROR_CODE err)`

`void sleep_ms (int time)`

Tells the program to wait for x ms.

Parameters

- `time`: : the number of ms to wait.

`template <typename T>`

`std::ostream &operator<< (std::ostream &os, const Vector2<T> &v2)`

`template <typename T>`

`std::ostream &operator<< (std::ostream &os, const Vector3<T> &v3)`

`template <typename T>`

`std::ostream &operator<< (std::ostream &os, const Vector4<T> &v4)`

Variables

`class SL_CORE_EXPORT_DLL Rotation`

`class SL_CORE_EXPORT_DLL Translation`

`class SL_CORE_EXPORT_DLL Orientation`

`class SL_CORE_EXPORT_DLL Transform`

`std::vector<std::pair<int, int>> cameraResolution = [] { std::vector<std::pair<int, int>> v; v.emplace_back(2208, 1242); v.emplace_back(1242, 2208); }`

Available video modes for the ZED camera

`namespace SpeakerDetect`

1. Face **is** received **from the** Facedetect module
2. The speaking algorithm tries to identify speaking
3. returns a **dict** of people who speak **and** their ID

Functions

`DetectSpeaker (FacepointQueue FacepointQueue, SpeakerQueue SpeakerQueue, FrameQueue FrameQueue, VisualQueue VisualQueue)`

`namespace std`

`namespace TransformCoordinates`

Functions

coordinate_transform (*x_camera x_camera, y_camera y_camera, z z*)

```
namespace Visualizer
    void Visualizer::displayFace
    This is currently not used. We need to figure out way to synchronize image access_
    ↪ across different processes.
    1. One place used to visualization.
```

Functions

StartVisualization (*CameraQueue CameraQueue, RectQueue RectQueue, FacePointQueue Face-
PointQueue, SpeakerQueue SpeakerQueue, FrameQueue FrameQueue, Visu-
alQueue VisualQueue*)

namespace voc_label

Functions

convert (*size size, box box*)

convert_annotation (*year year, image_id image_id*)

Variables

list voc_label.sets = [('2012', 'train'), ('2012', 'val'), ('2007', 'train'), ('2007', 'val'), ('2007', 'test')]

list voc_label.classes = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningta

wd = getcwd()

image_ids = open('VOCdevkit/VOC%s/ImageSets/Main/%s.txt'%(year, image_set)).read().strip().split()

list_file = open('%s_%s.txt'%(year, image_set), 'w')

namespace zedRoboy

Functions

load_meta (*ff*)

load_net (*cfg cfg, weights weights*)

load_img (*ff*)

letterbox_img (*im im, w w, h h*)

predict (*net net, im im*)

classify (*net net, meta meta, im im*)

detect (*net net, meta meta, im im*)

Variables

```
lib = CDLL("/usr/local/.so", RTLD_GLOBAL)
net = load_net("cfg/densenet.cfg", "/home/pjreddie/trained/densenet201.weights")
im = load_img("data/wolf.jpg")
meta = load_meta("cfg/imagenet1k.data")
r = classify(net, meta, im)
```

file **goal.txt**

file **make_labels.py**

file **DescribeSceneSrv.py**

file **FaceDetect.py**

file **FindObjectSrv.py**

file **FreezeModel.py**

file **multiprocess.py**

file **Multitracking.py**

file **NewFacialFeaturesMsg.py**

file **ObjectRecognition.py**

file **art.c**

```
#include "darknet.h"#include <sys/time.h>
```

Functions

```
void demo_art (char *cfgfile, char *weightfile, int cam_index)
```

```
void run_art (int argc, char **argv)
```

file **captcha.c**

```
#include "darknet.h"
```

Functions

```
void fix_data_captcha (data d, int mask)
```

```
void train_captcha (char *cfgfile, char *weightfile)
```

```
void test_captcha (char *cfgfile, char *weightfile, char *filename)
```

```
void valid_captcha (char *cfgfile, char *weightfile, char *filename)
```

```
void run_captcha (int argc, char **argv)
```

file **cifar.c**

```
#include "darknet.h"
```

Functions

```
void train_cifar (char *cfgfile, char *weightfile)
void train_cifar_distill (char *cfgfile, char *weightfile)
void test_cifar_multi (char *filename, char *weightfile)
void test_cifar (char *filename, char *weightfile)
void extract_cifar ()
void test_cifar_csv (char *filename, char *weightfile)
void test_cifar_csvtrain (char *filename, char *weightfile)
void eval_cifar_csv ()
void run_cifar (int argc, char **argv)
```

file **classifier.c**

```
#include "darknet.h" #include <sys/time.h> #include <assert.h>
```

Functions

```
float *get_regression_values (char **labels, int n)
void train_classifier (char *datacfg, char *cfgfile, char *weightfile, int *gpus, int ngpus, int clear)
void validate_classifier_crop (char *datacfg, char *filename, char *weightfile)
void validate_classifier_10 (char *datacfg, char *filename, char *weightfile)
void validate_classifier_full (char *datacfg, char *filename, char *weightfile)
void validate_classifier_single (char *datacfg, char *filename, char *weightfile)
void validate_classifier_multi (char *datacfg, char *filename, char *weightfile)
void try_classifier (char *datacfg, char *cfgfile, char *weightfile, char *filename, int layer_num)
void predict_classifier (char *datacfg, char *cfgfile, char *weightfile, char *filename, int top)
void label_classifier (char *datacfg, char *filename, char *weightfile)
void test_classifier (char *datacfg, char *cfgfile, char *weightfile, int target_layer)
void threat_classifier (char *datacfg, char *cfgfile, char *weightfile, int cam_index, const char
                        *filename)
void gun_classifier (char *datacfg, char *cfgfile, char *weightfile, int cam_index, const char *file-
                    name)
void demo_classifier (char *datacfg, char *cfgfile, char *weightfile, int cam_index, const char *file-
                    name)
void run_classifier (int argc, char **argv)
```

file **coco.c**

```
#include "darknet.h" #include <stdio.h>
```

Functions

```
void train_coco (char *cfgfile, char *weightfile)
void print_cocos (FILE *fp, int image_id, box *boxes, float **probs, int num_boxes, int classes, int
                  w, int h)
int get_coco_image_id (char *filename)
void validate_coco (char *cfgfile, char *weightfile)
void validate_coco_recall (char *cfgfile, char *weightfile)
void test_coco (char *cfgfile, char *weightfile, char *filename, float thresh)
void run_coco (int argc, char **argv)
```

Variables

```
char *coco_classes[] = {"person","bicycle","car","motorcycle","airplane","bus","train","truck","boat","traffic light","fire hydrant",
int coco_ids[] = {1,2,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,27,28,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99}
```

```
file darknet.c
#include "darknet.h"#include <time.h>#include <stdlib.h>#include <stdio.h>
```

Functions

```
void predict_classifier (char *datacfg, char *cfgfile, char *weightfile, char *filename, int top)
void test_detector (char *datacfg, char *cfgfile, char *weightfile, char *filename, float thresh, float
                    hier_thresh, char *outfile, int fullscreen)
void run_voxel (int argc, char **argv)
void run_yolo (int argc, char **argv)
void run_detector (int argc, char **argv)
void run_coco (int argc, char **argv)
void run_writing (int argc, char **argv)
void run_captcha (int argc, char **argv)
void run_nightmare (int argc, char **argv)
void run_dice (int argc, char **argv)
void run_compare (int argc, char **argv)
void run_classifier (int argc, char **argv)
void run_regressor (int argc, char **argv)
void run_segmenter (int argc, char **argv)
void run_char_rnn (int argc, char **argv)
void run_vid_rnn (int argc, char **argv)
void run_tag (int argc, char **argv)
void run_cifar (int argc, char **argv)
```



```

void run_go (int argc, char **argv)
void run_art (int argc, char **argv)
void run_super (int argc, char **argv)
void run_lsd (int argc, char **argv)
void average (int argc, char *argv[])
void speed (char *cfgfile, int tics)
void operations (char *cfgfile)
void oneoff (char *cfgfile, char *weightfile, char *outfile)
void oneoff2 (char *cfgfile, char *weightfile, char *outfile, int l)
void partial (char *cfgfile, char *weightfile, char *outfile, int max)
void rescale_net (char *cfgfile, char *weightfile, char *outfile)
void rgbgr_net (char *cfgfile, char *weightfile, char *outfile)
void reset_normalize_net (char *cfgfile, char *weightfile, char *outfile)
layer normalize_layer (layer l, int n)
void normalize_net (char *cfgfile, char *weightfile, char *outfile)
void statistics_net (char *cfgfile, char *weightfile)
void denormalize_net (char *cfgfile, char *weightfile, char *outfile)
void mkimg (char *cfgfile, char *weightfile, int h, int w, int num, char *prefix)
void visualize (char *cfgfile, char *weightfile)
int main (int argc, char **argv)

```

```

file detector.c
#include "darknet.h"

```

Functions

```

void train_detector (char *datacfg, char *cfgfile, char *weightfile, int *gpus, int ngpus, int clear)
static int get_coco_image_id (char *filename)
static void print_cocos (FILE *fp, char *image_path, box *boxes, float **probs, int num_boxes, int classes, int w, int h)
void print_detector_detections (FILE **fps, char *id, box *boxes, float **probs, int total, int classes, int w, int h)
void print_imagenet_detections (FILE *fp, int id, box *boxes, float **probs, int total, int classes, int w, int h)
void validate_detector_flip (char *datacfg, char *cfgfile, char *weightfile, char *outfile)
void validate_detector (char *datacfg, char *cfgfile, char *weightfile, char *outfile)
void validate_detector_recall (char *cfgfile, char *weightfile)
void test_detector (char *datacfg, char *cfgfile, char *weightfile, char *filename, float thresh, float hier_thresh, char *outfile, int fullscreen)
void run_detector (int argc, char **argv)

```

Variables

```
int coco_ids[] = { 1,2,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,27,28,31,32,33,34,35,36,37,38,39,40,41,42,4
```

```
file dice.c  
#include "darknet.h"
```

Functions

```
void train_dice (char *cfgfile, char *weightfile)  
void validate_dice (char *filename, char *weightfile)  
void test_dice (char *cfgfile, char *weightfile, char *filename)  
void run_dice (int argc, char **argv)
```

Variables

```
char *dice_labels[] = {"face1","face2","face3","face4","face5","face6"}
```

```
file go.c  
#include "darknet.h"#include <unistd.h>
```

Functions

```
char *fgetgo (FILE *fp)  
moves load_go_moves (char *filename)  
void string_to_board (char *s, float *board)  
void board_to_string (char *s, float *board)  
data random_go_moves (moves m, int n)  
void train_go (char *cfgfile, char *weightfile, char *filename, int *gpus, int ngpus, int clear)  
void propagate_liberty (float *board, int *lib, int *visited, int row, int col, int side)  
int *calculate_liberties (float *board)  
void print_board (FILE *stream, float *board, int swap, int *indexes)  
void flip_board (float *board)  
void predict_move (network net, float *board, float *move, int multi)  
void remove_connected (float *b, int *lib, int p, int r, int c)  
void move_go (float *b, int p, int r, int c)  
int makes_safe_go (float *b, int *lib, int p, int r, int c)  
int suicide_go (float *b, int p, int r, int c)  
int legal_go (float *b, char *ko, int p, int r, int c)  
int generate_move (network net, int player, float *board, int multi, float thresh, float temp, char *ko,  
int print)  
void valid_go (char *cfgfile, char *weightfile, int multi, char *filename)
```

```

int print_game (float *board, FILE *fp)
void engine_go (char *filename, char *weightfile, int multi)
void test_go (char *cfg, char *weights, int multi)
float score_game (float *board)
void self_go (char *filename, char *weightfile, char *f2, char *w2, int multi)
void run_go (int argc, char **argv)

```

Variables

```

int inverted = 1
int noi = 1
const int nind = 2

```

file **lsd.c**
#include "darknet.h"

Functions

```

void test_dcgan (char *cfgfile, char *weightfile)
void dcgan_batch (network gnet, network anet)
void train_dcgan (char *cfg, char *weight, char *acfg, char *aweight, int clear, int display, char
                 *train_images)
void train_colorizer (char *cfg, char *weight, char *acfg, char *aweight, int clear, int display)
void test_lsd (char *cfgfile, char *weightfile, char *filename, int gray)
void run_lsd (int argc, char **argv)

```

file **nightmare.c**
#include "darknet.h" #include <math.h>

Functions

```

float abs_mean (float *x, int n)
void calculate_loss (float *output, float *delta, int n, float thresh)
void optimize_picture (network *net, image orig, int max_layer, float scale, float rate, float thresh,
                      int norm)
void smooth (image recon, image update, float lambda, int num)
void reconstruct_picture (network net, float *features, image recon, image update, float rate, float
                        momentum, float lambda, int smooth_size, int iters)
void run_nightmare (int argc, char **argv)

```

file **regressor.c**
#include "darknet.h" #include <sys/time.h> #include <assert.h>

Functions

```
void train_regressor (char *datacfg, char *cfgfile, char *weightfile, int *gpus, int ngpus, int clear)
void predict_regressor (char *cfgfile, char *weightfile, char *filename)
void demo_regressor (char *datacfg, char *cfgfile, char *weightfile, int cam_index, const char *filename)
void run_regressor (int argc, char **argv)
```

file **rnn.c**

```
#include "darknet.h"#include <math.h>
```

Functions

```
int *read_tokenized_data (char *filename, size_t *read)
char **read_tokens (char *filename, size_t *read)
float_pair get_rnn_token_data (int *tokens, size_t *offsets, int characters, size_t len, int batch, int steps)
float_pair get_rnn_data (unsigned char *text, size_t *offsets, int characters, size_t len, int batch, int steps)
void reset_rnn_state (network net, int b)
void train_char_rnn (char *cfgfile, char *weightfile, char *filename, int clear, int tokenized)
void print_symbol (int n, char **tokens)
void test_char_rnn (char *cfgfile, char *weightfile, int num, char *seed, float temp, int rseed, char *token_file)
void test_tactic_rnn_multi (char *cfgfile, char *weightfile, int num, float temp, int rseed, char *token_file)
void test_tactic_rnn (char *cfgfile, char *weightfile, int num, float temp, int rseed, char *token_file)
void valid_tactic_rnn (char *cfgfile, char *weightfile, char *seed)
void valid_char_rnn (char *cfgfile, char *weightfile, char *seed)
void vec_char_rnn (char *cfgfile, char *weightfile, char *seed)
void run_char_rnn (int argc, char **argv)
```

file **rnn_vid.c**

```
#include "darknet.h"
```

Functions

```
void run_vid_rnn (int argc, char **argv)
```

file **segmenter.c**

```
#include "darknet.h"#include <sys/time.h>#include <assert.h>
```

Functions

void **train_segmenter** (char *datacfg, char *cfgfile, char *weightfile, int *gpus, int ngpus, int clear, int display)

void **predict_segmenter** (char *datafile, char *cfgfile, char *weightfile, char *filename)

void **demo_segmenter** (char *datacfg, char *cfgfile, char *weightfile, int cam_index, const char *filename)

void **run_segmenter** (int argc, char **argv)

file **super.c**
#include "darknet.h"

Functions

void **train_super** (char *cfgfile, char *weightfile, int clear)

void **test_super** (char *cfgfile, char *weightfile, char *filename)

void **run_super** (int argc, char **argv)

file **swag.c**
#include "darknet.h" #include <sys/time.h>

Functions

void **train_swag** (char *cfgfile, char *weightfile)

void **run_swag** (int argc, char **argv)

file **tag.c**
#include "darknet.h"

Functions

void **train_tag** (char *cfgfile, char *weightfile, int clear)

void **test_tag** (char *cfgfile, char *weightfile, char *filename)

void **run_tag** (int argc, char **argv)

file **voxel.c**
#include "darknet.h"

Functions

void **extract_voxel** (char *lfile, char *rfile, char *prefix)

void **train_voxel** (char *cfgfile, char *weightfile)

void **test_voxel** (char *cfgfile, char *weightfile, char *filename)

void **run_voxel** (int argc, char **argv)

file **writing.c**
#include "darknet.h"

Functions

```
void train_writing (char *cfgfile, char *weightfile)
void test_writing (char *cfgfile, char *weightfile, char *filename)
void run_writing (int argc, char **argv)
```

```
file yolo.c
#include "darknet.h"
```

Functions

```
void train_yolo (char *cfgfile, char *weightfile)
void print_yolo_detections (FILE **fps, char *id, box *boxes, float **probs, int total, int classes,
                           int w, int h)
void validate_yolo (char *cfgfile, char *weightfile)
void validate_yolo_recall (char *cfgfile, char *weightfile)
void test_yolo (char *cfgfile, char *weightfile, char *filename, float thresh)
void run_yolo (int argc, char **argv)
```

Variables

```
char *voc_names[] = {"aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "sheep", "sofa", "train", "truck", "tvmonitor", "zebra", "zebra", "zebra"}
```

```
file darknet.h
#include <stdlib.h>#include <stdio.h>#include <string.h>#include <pthread.h>
```

Defines

```
SECRET_NUM
```

Typedefs

```
typedef struct network network
typedef struct layer layer
typedef struct matrix matrix
typedef struct load_args load_args
typedef struct node node
typedef struct list list
```

Enums

enum ACTIVATION

Values:

LOGISTIC

RELU

RELIE

LINEAR

RAMP

TANH

PLSE

LEAKY

ELU

LOGGY

STAIR

HARDTAN

LHTAN

enum LAYER_TYPE

Values:

CONVOLUTIONAL

DECONVOLUTIONAL

CONNECTED

MAXPOOL

SOFTMAX

DETECTION

DROPOUT

CROP

ROUTE

COST

NORMALIZATION

AVGPOOL

LOCAL

SHORTCUT

ACTIVE

RNN

GRU

LSTM

```
CRNN
BATCHNORM
NETWORK
XNOR
REGION
REORG
BLANK
enum COST_TYPE
    Values:
        SSE
        MASKED
        L1
        SEG
        SMOOTH
enum learning_rate_policy
    Values:
        CONSTANT
        STEP
        EXP
        POLY
        STEPS
        SIG
        RANDOM
enum data_type
    Values:
        CLASSIFICATION_DATA
        DETECTION_DATA
        CAPTCHA_DATA
        REGION_DATA
        IMAGE_DATA
        COMPARE_DATA
        WRITING_DATA
        SWAG_DATA
        TAG_DATA
        OLD_CLASSIFICATION_DATA
        STUDY_DATA
        DET_DATA
```


SUPER_DATA
LETTERBOX_DATA
REGRESSION_DATA
SEGMENTATION_DATA
INSTANCE_DATA

Functions

metadata **get_metadata** (char *file)
 void **free_layer** (*layer*)
network **load_network** (char *cfg, char *weights, int clear)
network ***load_network_p** (char *cfg, char *weights, int clear)
load_args **get_base_args** (*network* net)
 void **free_data** (*data* d)
 pthread_t **load_data** (*load_args* args)
list ***read_data_cfg** (char *filename)
list ***read_cfg** (char *filename)
 void **forward_network** (*network* net)
 void **backward_network** (*network* net)
 void **update_network** (*network* net)
 void **axpy_cpu** (int N, float ALPHA, float *X, int INCX, float *Y, int INCY)
 void **copy_cpu** (int N, float *X, int INCX, float *Y, int INCY)
 void **scal_cpu** (int N, float ALPHA, float *X, int INCX)
 void **normalize_cpu** (float *x, float *mean, float *variance, int batch, int filters, int spatial)
 int **best_3d_shift_r** (*image* a, *image* b, int min, int max)
 void **save_image_png** (*image* im, const char *name)
 void **get_next_batch** (*data* d, int n, int offset, float *X, float *y)
 void **grayscale_image_3c** (*image* im)
 void **normalize_image** (*image* p)
 void **matrix_to_csv** (*matrix* m)
 float **train_network_sgd** (*network* net, *data* d, int n)
 void **rgbgr_image** (*image* im)
data **copy_data** (*data* d)
data **concat_data** (*data* d1, *data* d2)
data **load_cifar10_data** (char *filename)
 float **matrix_topk_accuracy** (*matrix* truth, *matrix* guess, int k)
 void **matrix_add_matrix** (*matrix* from, *matrix* to)

```
void scale_matrix (matrix m, float scale)
matrix csv_to_matrix (char *filename)
float *network_accuracies (network net, data d, int n)
float train_network_datum (network net)
image make_random_image (int w, int h, int c)
void denormalize_connected_layer (layer l)
void denormalize_convolutional_layer (layer l)
void statistics_connected_layer (layer l)
void rescale_weights (layer l, float scale, float trans)
void rgbgr_weights (layer l)
image *get_weights (layer l)
void demo (char *cfgfile, char *weightfile, float thresh, int cam_index, const char *filename, char
          **names, int classes, int frame_skip, char *prefix, int avg, float hier_thresh, int w, int h,
          int fps, int fullscreen)
void get_detection_boxes (layer l, int w, int h, float thresh, float **probs, box *boxes, int
                          only_objectness)
char *option_find_str (list *l, char *key, char *def)
int option_find_int (list *l, char *key, int def)
network parse_network_cfg (char *filename)
void save_weights (network net, char *filename)
void load_weights (network *net, char *filename)
void save_weights_upto (network net, char *filename, int cutoff)
void load_weights_upto (network *net, char *filename, int start, int cutoff)
void zero_objectness (layer l)
void get_region_boxes (layer l, int w, int h, int netw, int neth, float thresh, float **probs, box *boxes,
                        float **masks, int only_objectness, int *map, float tree_thresh, int relative)
void free_network (network net)
void set_batch_network (network *net, int b)
image load_image (char *filename, int w, int h, int c)
image load_image_color (char *filename, int w, int h)
image make_image (int w, int h, int c)
image resize_image (image im, int w, int h)
image letterbox_image (image im, int w, int h)
image crop_image (image im, int dx, int dy, int w, int h)
image resize_min (image im, int min)
image threshold_image (image im, float thresh)
image mask_to_rgb (image mask)
int resize_network (network *net, int w, int h)
```

```

void free_matrix (matrix m)
void test_resize (char *filename)
void save_image (image p, const char *name)
void show_image (image p, const char *name)
image copy_image (image p)
void draw_box_width (image a, int x1, int y1, int x2, int y2, int w, float r, float g, float b)
float get_current_rate (network net)
void composite_3d (char *f1, char *f2, char *out, int delta)
data load_data_old (char **paths, int n, int m, char **labels, int k, int w, int h)
size_t get_current_batch (network net)
void constrain_image (image im)
image get_network_image_layer (network net, int i)
layer get_network_output_layer (network net)
void top_predictions (network net, int n, int *index)
void flip_image (image a)
image float_to_image (int w, int h, int c, float *data)
void ghost_image (image source, image dest, int dx, int dy)
float network_accuracy (network net, data d)
void random_distort_image (image im, float hue, float saturation, float exposure)
void fill_image (image m, float s)
image grayscale_image (image im)
void rotate_image_cw (image im, int times)
double what_time_is_it_now ()
image rotate_image (image m, float rad)
void visualize_network (network net)
float box_iou (box a, box b)
void do_nms (box *boxes, float **probs, int total, int classes, float thresh)
data load_all_cifar10 ()
box_label *read_boxes (char *filename, int *n)
box float_to_box (float *f, int stride)
void draw_detections (image im, int num, float thresh, box *boxes, float **probs, float **masks, char
    **names, image **alphabet, int classes)
matrix network_predict_data (network net, data test)
image **load_alphabet ()
image get_network_image (network net)
float *network_predict (network net, float *input)
float *network_predict_p (network *net, float *input)

```

```
int network_width (network *net)
int network_height (network *net)
float network_predict_image (network *net, image im)
char **get_labels (char *filename)
void do_nms_sort (box *boxes, float **probs, int total, int classes, float thresh)
void do_nms_obj (box *boxes, float **probs, int total, int classes, float thresh)
matrix make_matrix (int rows, int cols)
void free_image (image m)
float train_network (network net, data d)
pthread_t load_data_in_thread (load_args args)
void load_data_blocking (load_args args)
list *get_paths (char *filename)
void hierarchy_predictions (float *predictions, int n, tree *hier, int only_leaves, int stride)
void change_leaves (tree *t, char *leaf_list)
int find_int_arg (int argc, char **argv, char *arg, int def)
float find_float_arg (int argc, char **argv, char *arg, float def)
int find_arg (int argc, char *argv[], char *arg)
char *find_char_arg (int argc, char **argv, char *arg, char *def)
char *basecfg (char *cfgfile)
void find_replace (char *str, char *orig, char *rep, char *output)
void free_ptrs (void **ptrs, int n)
char *fgetl (FILE *fp)
void strip (char *s)
float sec (clock_t clocks)
void **list_to_array (list *l)
void top_k (float *a, int n, int k, int *index)
int *read_map (char *filename)
void error (const char *s)
int max_index (float *a, int n)
int sample_array (float *a, int n)
void free_list (list *l)
float mse_array (float *a, int n)
float variance_array (float *a, int n)
float mag_array (float *a, int n)
float mean_array (float *a, int n)
void normalize_array (float *a, int n)
```

```
int *read_intlist (char *s, int *n, int d)
size_t rand_size_t ()
float rand_normal ()
```

Variables

```
int gpu_index
file darknet.py
file README.md
file README.md
file voc_label.py
file activation_layer.c
#include "activation_layer.h"#include "utils.h"#include "cuda.h"#include "blas.h"#include
"gemm.h"#include <math.h>#include <stdio.h>#include <stdlib.h>#include <string.h>
```

Functions

```
layer make_activation_layer (int batch, int inputs, ACTIVATION activation)
void forward_activation_layer (layer l, network net)
void backward_activation_layer (layer l, network net)
file activation_layer.h
#include "activations.h"#include "layer.h"#include "network.h"
```

Functions

```
layer make_activation_layer (int batch, int inputs, ACTIVATION activation)
void forward_activation_layer (layer l, network net)
void backward_activation_layer (layer l, network net)
file activations.c
#include "activations.h"#include <math.h>#include <stdio.h>#include <stdlib.h>#include <string.h>
```

Functions

```
char *get_activation_string (ACTIVATION a)
ACTIVATION get_activation (char *s)
float activate (float x, ACTIVATION a)
void activate_array (float *x, const int n, const ACTIVATION a)
float gradient (float x, ACTIVATION a)
void gradient_array (const float *x, const int n, const ACTIVATION a, float *delta)
file activations.h
#include "darknet.h"#include "cuda.h"#include "math.h"
```

Functions

ACTIVATION **get_activation** (char *s)
char ***get_activation_string** (*ACTIVATION* a)
float **activate** (float x, *ACTIVATION* a)
float **gradient** (float x, *ACTIVATION* a)
void **gradient_array** (const float *x, const int n, const *ACTIVATION* a, float *delta)
void **activate_array** (float *x, const int n, const *ACTIVATION* a)
static float **stair_activate** (float x)
static float **hardtan_activate** (float x)
static float **linear_activate** (float x)
static float **logistic_activate** (float x)
static float **loggy_activate** (float x)
static float **relu_activate** (float x)
static float **elu_activate** (float x)
static float **relie_activate** (float x)
static float **ramp_activate** (float x)
static float **leaky_activate** (float x)
static float **tanh_activate** (float x)
static float **plse_activate** (float x)
static float **lhtan_activate** (float x)
static float **lhtan_gradient** (float x)
static float **hardtan_gradient** (float x)
static float **linear_gradient** (float x)
static float **logistic_gradient** (float x)
static float **loggy_gradient** (float x)
static float **stair_gradient** (float x)
static float **relu_gradient** (float x)
static float **elu_gradient** (float x)
static float **relie_gradient** (float x)
static float **ramp_gradient** (float x)
static float **leaky_gradient** (float x)
static float **tanh_gradient** (float x)
static float **plse_gradient** (float x)

file **avgpool_layer.c**
#include "avgpool_layer.h"#include "cuda.h"#include <stdio.h>

Functions

```

avgpool_layer make_avgpool_layer (int batch, int w, int h, int c)
void resize_avgpool_layer (avgpool_layer *l, int w, int h)
void forward_avgpool_layer (const avgpool_layer l, network net)
void backward_avgpool_layer (const avgpool_layer l, network net)

```

```

file avgpool_layer.h
#include "image.h" #include "cuda.h" #include "layer.h" #include "network.h"

```

Typedefs

```
typedef layer avgpool_layer
```

Functions

```

image get_avgpool_image (avgpool_layer l)
avgpool_layer make_avgpool_layer (int batch, int w, int h, int c)
void resize_avgpool_layer (avgpool_layer *l, int w, int h)
void forward_avgpool_layer (const avgpool_layer l, network net)
void backward_avgpool_layer (const avgpool_layer l, network net)

```

```

file batchnorm_layer.c
#include "convolutional_layer.h" #include "batchnorm_layer.h" #include "blas.h" #include <stdio.h>

```

Functions

```

layer make_batchnorm_layer (int batch, int w, int h, int c)
void backward_scale_cpu (float *x_norm, float *delta, int batch, int n, int size, float
                        *scale_updates)
void mean_delta_cpu (float *delta, float *variance, int batch, int filters, int spatial, float
                    *mean_delta)
void variance_delta_cpu (float *x, float *delta, float *mean, float *variance, int batch, int filters,
                        int spatial, float *variance_delta)
void normalize_delta_cpu (float *x, float *mean, float *variance, float *mean_delta, float *variance_delta,
                          int batch, int filters, int spatial, float *delta)
void resize_batchnorm_layer (layer *layer, int w, int h)
void forward_batchnorm_layer (layer l, network net)
void backward_batchnorm_layer (layer l, network net)

```

```

file batchnorm_layer.h
#include "image.h" #include "layer.h" #include "network.h"

```

Functions

layer **make_batchnorm_layer** (int *batch*, int *w*, int *h*, int *c*)

void **forward_batchnorm_layer** (*layer l*, *network net*)

void **backward_batchnorm_layer** (*layer l*, *network net*)

file **blas.c**

```
#include "blas.h"#include <math.h>#include <assert.h>#include <float.h>#include <stdio.h>#include <stdlib.h>#include <string.h>
```

Functions

void **reorg_cpu** (float **x*, int *w*, int *h*, int *c*, int *batch*, int *stride*, int *forward*, float **out*)

void **flatten** (float **x*, int *size*, int *layers*, int *batch*, int *forward*)

void **weighted_sum_cpu** (float **a*, float **b*, float **s*, int *n*, float **c*)

void **weighted_delta_cpu** (float **a*, float **b*, float **s*, float **da*, float **db*, float **ds*, int *n*, float **dc*)

void **shortcut_cpu** (int *batch*, int *w1*, int *h1*, int *c1*, float **add*, int *w2*, int *h2*, int *c2*, float **out*)

void **mean_cpu** (float **x*, int *batch*, int *filters*, int *spatial*, float **mean*)

void **variance_cpu** (float **x*, float **mean*, int *batch*, int *filters*, int *spatial*, float **variance*)

void **normalize_cpu** (float **x*, float **mean*, float **variance*, int *batch*, int *filters*, int *spatial*)

void **const_cpu** (int *N*, float *ALPHA*, float **X*, int *INCX*)

void **mul_cpu** (int *N*, float **X*, int *INCX*, float **Y*, int *INCY*)

void **pow_cpu** (int *N*, float *ALPHA*, float **X*, int *INCX*, float **Y*, int *INCY*)

void **axpy_cpu** (int *N*, float *ALPHA*, float **X*, int *INCX*, float **Y*, int *INCY*)

void **scal_cpu** (int *N*, float *ALPHA*, float **X*, int *INCX*)

void **fill_cpu** (int *N*, float *ALPHA*, float **X*, int *INCX*)

void **deinter_cpu** (int *NX*, float **X*, int *NY*, float **Y*, int *B*, float **OUT*)

void **inter_cpu** (int *NX*, float **X*, int *NY*, float **Y*, int *B*, float **OUT*)

void **copy_cpu** (int *N*, float **X*, int *INCX*, float **Y*, int *INCY*)

void **mult_add_into_cpu** (int *N*, float **X*, float **Y*, float **Z*)

void **smooth_l1_cpu** (int *n*, float **pred*, float **truth*, float **delta*, float **error*)

void **l1_cpu** (int *n*, float **pred*, float **truth*, float **delta*, float **error*)

void **l2_cpu** (int *n*, float **pred*, float **truth*, float **delta*, float **error*)

float **dot_cpu** (int *N*, float **X*, int *INCX*, float **Y*, int *INCY*)

void **softmax** (float **input*, int *n*, float *temp*, int *stride*, float **output*)

void **softmax_cpu** (float **input*, int *n*, int *batch*, int *batch_offset*, int *groups*, int *group_offset*, int *stride*, float *temp*, float **output*)

file **blas.h**

```
#include "darknet.h"
```


Functions

```

void flatten (float *x, int size, int layers, int batch, int forward)
void pm (int M, int N, float *A)
float *random_matrix (int rows, int cols)
void time_random_matrix (int TA, int TB, int m, int k, int n)
void reorg_cpu (float *x, int w, int h, int c, int batch, int stride, int forward, float *out)
void test_blas ()
void inter_cpu (int NX, float *X, int NY, float *Y, int B, float *OUT)
void deinter_cpu (int NX, float *X, int NY, float *Y, int B, float *OUT)
void mult_add_into_cpu (int N, float *X, float *Y, float *Z)
void const_cpu (int N, float ALPHA, float *X, int INCX)
void constrain_gpu (int N, float ALPHA, float *X, int INCX)
void pow_cpu (int N, float ALPHA, float *X, int INCX, float *Y, int INCY)
void mul_cpu (int N, float *X, int INCX, float *Y, int INCY)
void fill_cpu (int N, float ALPHA, float *X, int INCX)
float dot_cpu (int N, float *X, int INCX, float *Y, int INCY)
int test_gpu_blas ()
void shortcut_cpu (int batch, int w1, int h1, int c1, float *add, int w2, int h2, int c2, float *out)
void mean_cpu (float *x, int batch, int filters, int spatial, float *mean)
void variance_cpu (float *x, float *mean, int batch, int filters, int spatial, float *variance)
void scale_bias (float *output, float *scales, int batch, int n, int size)
void backward_scale_cpu (float *x_norm, float *delta, int batch, int n, int size, float
    *scale_updates)
void mean_delta_cpu (float *delta, float *variance, int batch, int filters, int spatial, float
    *mean_delta)
void variance_delta_cpu (float *x, float *delta, float *mean, float *variance, int batch, int filters,
    int spatial, float *variance_delta)
void normalize_delta_cpu (float *x, float *mean, float *variance, float *mean_delta, float *vari-
    ance_delta, int batch, int filters, int spatial, float *delta)
void smooth_l1_cpu (int n, float *pred, float *truth, float *delta, float *error)
void l2_cpu (int n, float *pred, float *truth, float *delta, float *error)
void l1_cpu (int n, float *pred, float *truth, float *delta, float *error)
void weighted_sum_cpu (float *a, float *b, float *s, int num, float *c)
void weighted_delta_cpu (float *a, float *b, float *s, float *da, float *db, float *ds, int n, float *dc)
void softmax (float *input, int n, float temp, int stride, float *output)
void softmax_cpu (float *input, int n, int batch, int batch_offset, int groups, int group_offset, int stride,
    float temp, float *output)

```

file **box.c**

```
#include "box.h"#include <stdio.h>#include <math.h>#include <stdlib.h>
```

Functions

box **float_to_box** (float *f, int stride)

dbox **derivative** (*box* a, *box* b)

float **overlap** (float x1, float w1, float x2, float w2)

float **box_intersection** (*box* a, *box* b)

float **box_union** (*box* a, *box* b)

float **box_iou** (*box* a, *box* b)

float **box_rmse** (*box* a, *box* b)

dbox **dintersect** (*box* a, *box* b)

dbox **dunion** (*box* a, *box* b)

void **test_dunion** ()

void **test_dintersect** ()

void **test_box** ()

dbox **diou** (*box* a, *box* b)

int **nms_comparator** (const void *pa, const void *pb)

void **do_nms_obj** (*box* *boxes, float **probs, int total, int classes, float thresh)

void **do_nms_sort** (*box* *boxes, float **probs, int total, int classes, float thresh)

void **do_nms** (*box* *boxes, float **probs, int total, int classes, float thresh)

box **encode_box** (*box* b, *box* anchor)

box **decode_box** (*box* b, *box* anchor)

file **box.h**

```
#include "darknet.h"
```

Functions

float **box_rmse** (*box* a, *box* b)

dbox **diou** (*box* a, *box* b)

box **decode_box** (*box* b, *box* anchor)

box **encode_box** (*box* b, *box* anchor)

file **classifier.h**

file **col2im.c**

```
#include <stdio.h>#include <math.h>
```

Functions

```
void col2im_add_pixel (float *im, int height, int width, int channels, int row, int col, int channel, int
    pad, float val)
```

```
void col2im_cpu (float *data_col, int channels, int height, int width, int ksize, int stride, int pad, float
    *data_im)
```

file **col2im.h**

Functions

```
void col2im_cpu (float *data_col, int channels, int height, int width, int ksize, int stride, int pad, float
    *data_im)
```

file **compare.c**

```
#include <stdio.h>#include "network.h"#include "detection_layer.h"#include "cost_layer.h"#include
"utils.h"#include "parser.h"#include "box.h"
```

Functions

```
void train_compare (char *cfgfile, char *weightfile)
```

```
void validate_compare (char *filename, char *weightfile)
```

```
int elo_comparator (const void *a, const void *b)
```

```
int bbox_comparator (const void *a, const void *b)
```

```
void bbox_update (sortable_bbox *a, sortable_bbox *b, int class, int result)
```

```
void bbox_fight (network net, sortable_bbox *a, sortable_bbox *b, int classes, int class)
```

```
void SortMaster3000 (char *filename, char *weightfile)
```

```
void BattleRoyaleWithCheese (char *filename, char *weightfile)
```

```
void run_compare (int argc, char **argv)
```

Variables

```
int total_compares = 0
```

```
int current_class = 0
```

file **connected_layer.c**

```
#include "connected_layer.h"#include "convolutional_layer.h"#include "batchnorm_layer.h"#include
"utils.h"#include "cuda.h"#include "blas.h"#include "gemm.h"#include <math.h>#include
<stdio.h>#include <stdlib.h>#include <string.h>
```

Functions

```
layer make_connected_layer (int batch, int inputs, int outputs, ACTIVATION activation, int
    batch_normalize, int adam)
```

```
void update_connected_layer (layer l, update_args a)
```

```
void forward_connected_layer (layer l, network net)
```

```
void backward_connected_layer (layer l, network net)
void denormalize_connected_layer (layer l)
void statistics_connected_layer (layer l)
```

```
file connected_layer.h
#include "activations.h" #include "layer.h" #include "network.h"
```

Functions

```
layer make_connected_layer (int batch, int inputs, int outputs, ACTIVATION activation, int
                             batch_normalize, int adam)
void forward_connected_layer (layer l, network net)
void backward_connected_layer (layer l, network net)
void update_connected_layer (layer l, update_args a)
```

```
file convolutional_layer.c
#include "convolutional_layer.h" #include "utils.h" #include "batchnorm_layer.h" #include
"im2col.h" #include "col2im.h" #include "blas.h" #include "gemm.h" #include <stdio.h> #include <time.h>
```

Functions

```
void swap_binary (convolutional_layer *l)
void binarize_weights (float *weights, int n, int size, float *binary)
void binarize_cpu (float *input, int n, float *binary)
void binarize_input (float *input, int n, int size, float *binary)
int convolutional_out_height (convolutional_layer l)
int convolutional_out_width (convolutional_layer l)
image get_convolutional_image (convolutional_layer l)
image get_convolutional_delta (convolutional_layer l)
static size_t get_workspace_size (layer l)
convolutional_layer make_convolutional_layer (int batch, int h, int w, int c, int n, int size,
                                                int stride, int padding, ACTIVATION activation,
                                                int batch_normalize, int binary, int xnor,
                                                int adam)
void denormalize_convolutional_layer (convolutional_layer l)
void resize_convolutional_layer (convolutional_layer *l, int w, int h)
void add_bias (float *output, float *biases, int batch, int n, int size)
void scale_bias (float *output, float *scales, int batch, int n, int size)
void backward_bias (float *bias_updates, float *delta, int batch, int n, int size)
void forward_convolutional_layer (convolutional_layer l, network net)
void backward_convolutional_layer (convolutional_layer l, network net)
void update_convolutional_layer (convolutional_layer l, update_args a)
```

```

image get_convolutional_weight (convolutional_layer l, int i)
void rgbgr_weights (convolutional_layer l)
void rescale_weights (convolutional_layer l, float scale, float trans)
image *get_weights (convolutional_layer l)
image *visualize_convolutional_layer (convolutional_layer l, char *window, image
                                     *prev_weights)

```

file **convolutional_layer.h**

```
#include "cuda.h"#include "image.h"#include "activations.h"#include "layer.h"#include "network.h"
```

Typedefs

```
typedef layer convolutional_layer
```

Functions

```

convolutional_layer make_convolutional_layer (int batch, int h, int w, int c, int n, int size,
                                              int stride, int padding, ACTIVATION activation,
                                              int batch_normalize, int binary, int xnor,
                                              int adam)

void resize_convolutional_layer (convolutional_layer *layer, int w, int h)
void forward_convolutional_layer (const convolutional_layer layer, network net)
void update_convolutional_layer (convolutional_layer layer, update_args a)
image *visualize_convolutional_layer (convolutional_layer layer, char *window, image
                                     *prev_weights)

void binarize_weights (float *weights, int n, int size, float *binary)
void swap_binary (convolutional_layer *l)
void binarize_weights2 (float *weights, int n, int size, char *binary, float *scales)
void backward_convolutional_layer (convolutional_layer layer, network net)
void add_bias (float *output, float *biases, int batch, int n, int size)
void backward_bias (float *bias_updates, float *delta, int batch, int n, int size)
image get_convolutional_image (convolutional_layer layer)
image get_convolutional_delta (convolutional_layer layer)
image get_convolutional_weight (convolutional_layer layer, int i)
int convolutional_out_height (convolutional_layer layer)
int convolutional_out_width (convolutional_layer layer)

```

file **cost_layer.c**

```
#include "cost_layer.h"#include "utils.h"#include "cuda.h"#include "blas.h"#include <math.h>#include
<string.h>#include <stdlib.h>#include <stdio.h>
```

Functions

```
COST_TYPE get_cost_type (char *s)
char *get_cost_string (COST_TYPE a)
cost_layer make_cost_layer (int batch, int inputs, COST_TYPE cost_type, float scale)
void resize_cost_layer (cost_layer *l, int inputs)
void forward_cost_layer (cost_layer l, network net)
void backward_cost_layer (const cost_layer l, network net)
```

```
file cost_layer.h
#include "layer.h" #include "network.h"
```

Typedefs

```
typedef layer cost_layer
```

Functions

```
COST_TYPE get_cost_type (char *s)
char *get_cost_string (COST_TYPE a)
cost_layer make_cost_layer (int batch, int inputs, COST_TYPE type, float scale)
void forward_cost_layer (const cost_layer l, network net)
void backward_cost_layer (const cost_layer l, network net)
void resize_cost_layer (cost_layer *l, int inputs)
```

```
file crnn_layer.c
#include "crnn_layer.h" #include "convolutional_layer.h" #include "utils.h" #include "cuda.h" #include
"blas.h" #include "gemm.h" #include <math.h> #include <stdio.h> #include <stdlib.h> #include <string.h>
```

Functions

```
static void increment_layer (layer *l, int steps)
layer make_crnn_layer (int batch, int h, int w, int c, int hidden_filters, int output_filters, int steps,
                        ACTIVATION activation, int batch_normalize)
void update_crnn_layer (layer l, update_args a)
void forward_crnn_layer (layer l, network net)
void backward_crnn_layer (layer l, network net)
```

```
file crnn_layer.h
#include "activations.h" #include "layer.h" #include "network.h"
```

Functions

layer **make_crnn_layer** (int *batch*, int *h*, int *w*, int *c*, int *hidden_filters*, int *output_filters*, int *steps*,
ACTIVATION *activation*, int *batch_normalize*)

void **forward_crnn_layer** (*layer l*, *network net*)

void **backward_crnn_layer** (*layer l*, *network net*)

void **update_crnn_layer** (*layer l*, *update_args a*)

file **crop_layer.c**

#include "crop_layer.h" #include "cuda.h" #include <stdio.h>

Functions

image **get_crop_image** (*crop_layer l*)

void **backward_crop_layer** (const *crop_layer l*, *network net*)

void **backward_crop_layer_gpu** (const *crop_layer l*, *network net*)

crop_layer **make_crop_layer** (int *batch*, int *h*, int *w*, int *c*, int *crop_height*, int *crop_width*, int *flip*,
float *angle*, float *saturation*, float *exposure*)

void **resize_crop_layer** (*layer *l*, int *w*, int *h*)

void **forward_crop_layer** (const *crop_layer l*, *network net*)

file **crop_layer.h**

#include "image.h" #include "layer.h" #include "network.h"

Typedefs

typedef *layer* **crop_layer**

Functions

image **get_crop_image** (*crop_layer l*)

crop_layer **make_crop_layer** (int *batch*, int *h*, int *w*, int *c*, int *crop_height*, int *crop_width*, int *flip*,
float *angle*, float *saturation*, float *exposure*)

void **forward_crop_layer** (const *crop_layer l*, *network net*)

void **resize_crop_layer** (*layer *l*, int *w*, int *h*)

file **cuda.c**

Variables

int **gpu_index** = 0

file **cuda.h**

#include "darknet.h"

file **data.c**

```
#include "data.h"#include "utils.h"#include "image.h"#include "cuda.h"#include <stdio.h>#include <stdlib.h>#include <string.h>
```

Defines

NUMCHARS

Functions

list ***get_paths** (char *filename)

char ****get_random_paths** (char **paths, int n, int m)

char ****find_replace_paths** (char **paths, int n, char *find, char *replace)

matrix **load_image_paths_gray** (char **paths, int n, int w, int h)

matrix **load_image_paths** (char **paths, int n, int w, int h)

matrix **load_image_augment_paths** (char **paths, int n, int min, int max, int size, float angle, float aspect, float hue, float saturation, float exposure, int center)

box_label ***read_boxes** (char *filename, int *n)

void **randomize_boxes** (*box_label* *b, int n)

void **correct_boxes** (*box_label* *boxes, int n, float dx, float dy, float sx, float sy, int flip)

void **fill_truth_swag** (char *path, float *truth, int classes, int flip, float dx, float dy, float sx, float sy)

void **fill_truth_region** (char *path, float *truth, int classes, int num_boxes, int flip, float dx, float dy, float sx, float sy)

void **load_rle** (*image* im, int *rle, int n)

void **or_image** (*image* src, *image* dest, int c)

void **exclusive_image** (*image* src)

box **bound_image** (*image* im)

void **fill_truth_iseg** (char *path, int num_boxes, float *truth, int classes, int w, int h, *augment_args* aug, int flip, int mw, int mh)

void **fill_truth_detection** (char *path, int num_boxes, float *truth, int classes, int flip, float dx, float dy, float sx, float sy)

void **print_letters** (float *pred, int n)

void **fill_truth_captcha** (char *path, int n, float *truth)

data **load_data_captcha** (char **paths, int n, int m, int k, int w, int h)

data **load_data_captcha_encode** (char **paths, int n, int m, int w, int h)

void **fill_truth** (char *path, char **labels, int k, float *truth)

void **fill_hierarchy** (float *truth, int k, *tree* *hierarchy)

matrix **load_regression_labels_paths** (char **paths, int n)

matrix **load_labels_paths** (char **paths, int n, char **labels, int k, *tree* *hierarchy)


```

matrix load_tags_paths (char **paths, int n, int k)
char **get_labels (char *filename)
void free_data (data d)
image get_segmentation_image (char *path, int w, int h, int classes)
image get_segmentation_image2 (char *path, int w, int h, int classes)
data load_data_seg (int n, char **paths, int m, int w, int h, int classes, int min, int max, float angle,
                    float aspect, float hue, float saturation, float exposure, int div)
data load_data_iseg (int n, char **paths, int m, int w, int h, int classes, int boxes, int coords, int min,
                    int max, float angle, float aspect, float hue, float saturation, float exposure)
data load_data_region (int n, char **paths, int m, int w, int h, int size, int classes, float jitter, float
                      hue, float saturation, float exposure)
data load_data_compare (int n, char **paths, int m, int classes, int w, int h)
data load_data_swag (char **paths, int n, int classes, float jitter)
data load_data_detection (int n, char **paths, int m, int w, int h, int boxes, int classes, float jitter,
                        float hue, float saturation, float exposure)
void *load_thread (void *ptr)
pthread_t load_data_in_thread (load_args args)
void *load_threads (void *ptr)
void load_data_blocking (load_args args)
pthread_t load_data (load_args args)
data load_data_writing (char **paths, int n, int m, int w, int h, int out_w, int out_h)
data load_data_old (char **paths, int n, int m, char **labels, int k, int w, int h)
data load_data_super (char **paths, int n, int m, int w, int h, int scale)
data load_data_regression (char **paths, int n, int m, int min, int max, int size, float angle, float
                          aspect, float hue, float saturation, float exposure)
data load_data_augment (char **paths, int n, int m, char **labels, int k, tree *hierarchy, int min,
                      int max, int size, float angle, float aspect, float hue, float saturation, float
                      exposure, int center)
data load_data_tag (char **paths, int n, int m, int k, int min, int max, int size, float angle, float aspect,
                  float hue, float saturation, float exposure)
matrix concat_matrix (matrix m1, matrix m2)
data concat_data (data d1, data d2)
data concat_datas (data *d, int n)
data load_categorical_data_csv (char *filename, int target, int k)
data load_cifar10_data (char *filename)
void get_random_batch (data d, int n, float *X, float *y)
void get_next_batch (data d, int n, int offset, float *X, float *y)
void smooth_data (data d)
data load_all_cifar10 ()
data load_go (char *filename)

```

```
void randomize_data (data d)
void scale_data_rows (data d, float s)
void translate_data_rows (data d, float s)
data copy_data (data d)
void normalize_data_rows (data d)
data get_data_part (data d, int part, int total)
data get_random_data (data d, int num)
data *split_data (data d, int part, int total)
```

Variables

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER
```

file **data.h**

```
#include <pthread.h>#include "darknet.h"#include "matrix.h"#include "list.h"#include "image.h"#include "tree.h"
```

Functions

```
static float distance_from_edge (int x, int max)
void load_data_blocking (load_args args)
void print_letters (float *pred, int n)
data load_data_captcha (char **paths, int n, int m, int k, int w, int h)
data load_data_captcha_encode (char **paths, int n, int m, int w, int h)
data load_data_detection (int n, char **paths, int m, int w, int h, int boxes, int classes, float jitter,
                          float hue, float saturation, float exposure)
data load_data_tag (char **paths, int n, int m, int k, int min, int max, int size, float angle, float aspect,
                   float hue, float saturation, float exposure)
matrix load_image_augment_paths (char **paths, int n, int min, int max, int size, float angle, float
                                aspect, float hue, float saturation, float exposure, int center)
data load_data_super (char **paths, int n, int m, int w, int h, int scale)
data load_data_augment (char **paths, int n, int m, char **labels, int k, tree *hierarchy, int min,
                       int max, int size, float angle, float aspect, float hue, float saturation, float
                       exposure, int center)
data load_data_regression (char **paths, int n, int m, int min, int max, int size, float angle, float
                           aspect, float hue, float saturation, float exposure)
data load_go (char *filename)
data load_data_writing (char **paths, int n, int m, int w, int h, int out_w, int out_h)
void get_random_batch (data d, int n, float *X, float *y)
data get_data_part (data d, int part, int total)
data get_random_data (data d, int num)
data load_categorical_data_csv (char *filename, int target, int k)
```

```

void normalize_data_rows (data d)
void scale_data_rows (data d, float s)
void translate_data_rows (data d, float s)
void randomize_data (data d)
data *split_data (data d, int part, int total)
data concat_datas (data *d, int n)
void fill_truth (char *path, char **labels, int k, float *truth)

```

file **deconvolutional_layer.c**

```

#include "deconvolutional_layer.h" #include "convolutional_layer.h" #include "batchnorm_layer.h" #include
"utils.h" #include "im2col.h" #include "col2im.h" #include "blas.h" #include "gemm.h" #include
<stdio.h> #include <time.h>

```

Functions

```

static size_t get_workspace_size (layer l)

layer make_deconvolutional_layer (int batch, int h, int w, int c, int n, int size, int stride, int
padding, ACTIVATION activation, int batch_normalize, int
adam)

void denormalize_deconvolutional_layer (layer l)

void resize_deconvolutional_layer (layer *l, int h, int w)

void forward_deconvolutional_layer (const layer l, network net)

void backward_deconvolutional_layer (layer l, network net)

void update_deconvolutional_layer (layer l, update_args a)

```

file **deconvolutional_layer.h**

```

#include "cuda.h" #include "image.h" #include "activations.h" #include "layer.h" #include "network.h"

```

Functions

```

layer make_deconvolutional_layer (int batch, int h, int w, int c, int n, int size, int stride, int
padding, ACTIVATION activation, int batch_normalize, int
adam)

void resize_deconvolutional_layer (layer *l, int h, int w)

void forward_deconvolutional_layer (const layer l, network net)

void update_deconvolutional_layer (layer l, update_args a)

void backward_deconvolutional_layer (layer l, network net)

```

file **demo.c**

```

#include "network.h" #include "detection_layer.h" #include "region_layer.h" #include "cost_layer.h" #include
"utils.h" #include "parser.h" #include "box.h" #include "image.h" #include "demo.h" #include <sys/time.h>

```

Defines

DEMO

Functions

```
void demo (char *cfgfile, char *weightfile, float thresh, int cam_index, const char *filename, char
          **names, int classes, int delay, char *prefix, int avg, float hier, int w, int h, int frames, int
          fullscreen)
```

```
file demo.h
```

```
#include "image.h"
```

```
file detection_layer.c
```

```
#include "detection_layer.h"#include "activations.h"#include "softmax_layer.h"#include "blas.h"#include
"box.h"#include "cuda.h"#include "utils.h"#include <stdio.h>#include <assert.h>#include
<string.h>#include <stdlib.h>
```

Functions

```
detection_layer make_detection_layer (int batch, int inputs, int n, int side, int classes, int coords,
                                       int rescore)
```

```
void forward_detection_layer (const detection_layer l, network net)
```

```
void backward_detection_layer (const detection_layer l, network net)
```

```
void get_detection_boxes (layer l, int w, int h, float thresh, float **probs, box *boxes, int
                          only_objectness)
```

```
file detection_layer.h
```

```
#include "layer.h"#include "network.h"
```

Typedefs

```
typedef layer detection_layer
```

Functions

```
detection_layer make_detection_layer (int batch, int inputs, int n, int size, int classes, int coords,
                                       int rescore)
```

```
void forward_detection_layer (const detection_layer l, network net)
```

```
void backward_detection_layer (const detection_layer l, network net)
```

```
file dropout_layer.c
```

```
#include "dropout_layer.h"#include "utils.h"#include "cuda.h"#include <stdlib.h>#include <stdio.h>
```

Functions

```
dropout_layer make_dropout_layer (int batch, int inputs, float probability)
```

```
void resize_dropout_layer (dropout_layer *l, int inputs)
```

```
void forward_dropout_layer (dropout_layer l, network net)
```

```
void backward_dropout_layer (dropout_layer l, network net)
```

```
file dropout_layer.h
```

```
#include "layer.h"#include "network.h"
```

Typedefs

```
typedef layer dropout_layer
```

Functions

```
dropout_layer make_dropout_layer (int batch, int inputs, float probability)
```

```
void forward_dropout_layer (dropout_layer l, network net)
```

```
void backward_dropout_layer (dropout_layer l, network net)
```

```
void resize_dropout_layer (dropout_layer *l, int inputs)
```

file **gemm.c**

```
#include "gemm.h"#include "utils.h"#include "cuda.h"#include <stdlib.h>#include <stdio.h>#include <math.h>
```

Functions

```
void gemm_bin (int M, int N, int K, float ALPHA, char *A, int lda, float *B, int ldb, float *C, int ldc)
```

```
float *random_matrix (int rows, int cols)
```

```
void time_random_matrix (int TA, int TB, int m, int k, int n)
```

```
void gemm (int TA, int TB, int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float BETA, float *C, int ldc)
```

```
void gemm_nn (int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float *C, int ldc)
```

```
void gemm_nt (int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float *C, int ldc)
```

```
void gemm_tn (int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float *C, int ldc)
```

```
void gemm_tt (int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float *C, int ldc)
```

```
void gemm_cpu (int TA, int TB, int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float BETA, float *C, int ldc)
```

file **gemm.h**

Functions

```
void gemm_bin (int M, int N, int K, float ALPHA, char *A, int lda, float *B, int ldb, float *C, int ldc)
```

```
void gemm (int TA, int TB, int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float BETA, float *C, int ldc)
```

```
void gemm_cpu (int TA, int TB, int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float BETA, float *C, int ldc)
```

file **gru_layer.c**

```
#include "gru_layer.h"#include "connected_layer.h"#include "utils.h"#include "cuda.h"#include "blas.h"#include "gemm.h"#include <math.h>#include <stdio.h>#include <stdlib.h>#include <string.h>
```

Functions

```
static void increment_layer (layer *l, int steps)
layer make_gru_layer (int batch, int inputs, int outputs, int steps, int batch_normalize, int adam)
void update_gru_layer (layer l, update_args a)
void forward_gru_layer (layer l, network net)
void backward_gru_layer (layer l, network net)
```

```
file gru_layer.h
#include "activations.h"#include "layer.h"#include "network.h"
```

Functions

```
layer make_gru_layer (int batch, int inputs, int outputs, int steps, int batch_normalize, int adam)
void forward_gru_layer (layer l, network state)
void backward_gru_layer (layer l, network state)
void update_gru_layer (layer l, update_args a)
```

```
file im2col.c
#include "im2col.h"#include <stdio.h>
```

Functions

```
float im2col_get_pixel (float *im, int height, int width, int channels, int row, int col, int channel, int pad)
void im2col_cpu (float *data_im, int channels, int height, int width, int ksize, int stride, int pad, float *data_col)
```

```
file im2col.h
```

Functions

```
void im2col_cpu (float *data_im, int channels, int height, int width, int ksize, int stride, int pad, float *data_col)
```

```
file image.c
#include "image.h"#include "utils.h"#include "blas.h"#include "cuda.h"#include <stdio.h>#include <math.h>#include "stb_image.h"#include "stb_image_write.h"
```

Defines

```
STB_IMAGE_IMPLEMENTATION
STB_IMAGE_WRITE_IMPLEMENTATION
```

Functions

```

float get_color (int c, int x, int max)
image mask_to_rgb (image mask)
void composite_image (image source, image dest, int dx, int dy)
image border_image (image a, int border)
image tile_images (image a, image b, int dx)
image get_label (image **characters, char *string, int size)
void draw_label (image a, int r, int c, image label, const float *rgb)
void draw_box (image a, int x1, int y1, int x2, int y2, float r, float g, float b)
void draw_box_width (image a, int x1, int y1, int x2, int y2, int w, float r, float g, float b)
void draw_bbox (image a, box bbox, int w, float r, float g, float b)
image **load_alphabet ()
void draw_detections (image im, int num, float thresh, box *boxes, float **probs, float **masks, char
    **names, image **alphabet, int classes)
void transpose_image (image im)
void rotate_image_cw (image im, int times)
void flip_image (image a)
image image_distance (image a, image b)
void ghost_image (image source, image dest, int dx, int dy)
void embed_image (image source, image dest, int dx, int dy)
image collapse_image_layers (image source, int border)
void constrain_image (image im)
void normalize_image (image p)
void normalize_image2 (image p)
void copy_image_into (image src, image dest)
image copy_image (image p)
void rgbgr_image (image im)
void show_image (image p, const char *name)
void save_image_png (image im, const char *name)
void save_image (image im, const char *name)
void show_image_layers (image p, char *name)
void show_image_collapsed (image p, char *name)
image make_empty_image (int w, int h, int c)
image make_image (int w, int h, int c)
image make_random_image (int w, int h, int c)
image float_to_image (int w, int h, int c, float *data)

```

```
void place_image (image im, int w, int h, int dx, int dy, image canvas)
image center_crop_image (image im, int w, int h)
image rotate_crop_image (image im, float rad, float s, int w, int h, float dx, float dy, float aspect)
image rotate_image (image im, float rad)
void fill_image (image m, float s)
void translate_image (image m, float s)
void scale_image (image m, float s)
image crop_image (image im, int dx, int dy, int w, int h)
int best_3d_shift_r (image a, image b, int min, int max)
int best_3d_shift (image a, image b, int min, int max)
void composite_3d (char *f1, char *f2, char *out, int delta)
void letterbox_image_into (image im, int w, int h, image boxed)
image letterbox_image (image im, int w, int h)
image resize_max (image im, int max)
image resize_min (image im, int min)
image random_crop_image (image im, int w, int h)
augment_args random_augment_args (image im, float angle, float aspect, int low, int high, int w, int h)
image random_augment_image (image im, float angle, float aspect, int low, int high, int w, int h)
float three_way_max (float a, float b, float c)
float three_way_min (float a, float b, float c)
void yuv_to_rgb (image im)
void rgb_to_yuv (image im)
void rgb_to_hsv (image im)
void hsv_to_rgb (image im)
void grayscale_image_3c (image im)
image grayscale_image (image im)
image threshold_image (image im, float thresh)
image blend_image (image fore, image back, float alpha)
void scale_image_channel (image im, int c, float v)
void translate_image_channel (image im, int c, float v)
image binarize_image (image im)
void saturate_image (image im, float sat)
void hue_image (image im, float hue)
void exposure_image (image im, float sat)
void distort_image (image im, float hue, float sat, float val)
void random_distort_image (image im, float hue, float saturation, float exposure)
```



```

void saturate_exposure_image (image im, float sat, float exposure)
float bilinear_interpolate (image im, float x, float y, int c)
image resize_image (image im, int w, int h)
void test_resize (char *filename)
image load_image_stb (char *filename, int channels)
image load_image (char *filename, int w, int h, int c)
image load_image_color (char *filename, int w, int h)
image get_image_layer (image m, int l)
float get_pixel (image m, int x, int y, int c)
float get_pixel_extend (image m, int x, int y, int c)
void set_pixel (image m, int x, int y, int c, float val)
void add_pixel (image m, int x, int y, int c, float val)
void print_image (image m)
image collapse_images_vert (image *ims, int n)
image collapse_images_horz (image *ims, int n)
void show_image_normalized (image im, const char *name)
void show_images (image *ims, int n, char *window)
void free_image (image m)

```

Variables

```

int windows = 0
float colors[6][3] = { { 1,0,1 }, { 0,0,1 }, { 0,1,1 }, { 0,1,0 }, { 1,1,0 }, { 1,0,0 } }

```

file **image.h**

```

#include <stdlib.h>#include <stdio.h>#include <float.h>#include <string.h>#include <math.h>#include
"box.h"#include "darknet.h"

```

Functions

```

float get_color (int c, int x, int max)
void draw_box (image a, int x1, int y1, int x2, int y2, float r, float g, float b)
void draw_bbox (image a, box bbox, int w, float r, float g, float b)
void draw_label (image a, int r, int c, image label, const float *rgb)
void write_label (image a, int r, int c, image *characters, char *string, float *rgb)
image image_distance (image a, image b)
void scale_image (image m, float s)
image rotate_crop_image (image im, float rad, float s, int w, int h, float dx, float dy, float aspect)
image center_crop_image (image im, int w, int h)

```

```
image random_crop_image (image im, int w, int h)
image random_augment_image (image im, float angle, float aspect, int low, int high, int w, int h)
augment_args random_augment_args (image im, float angle, float aspect, int low, int high, int w, int
                                   h)
void letterbox_image_into (image im, int w, int h, image boxed)
image resize_max (image im, int max)
void translate_image (image m, float s)
void embed_image (image source, image dest, int dx, int dy)
void place_image (image im, int w, int h, int dx, int dy, image canvas)
void saturate_image (image im, float sat)
void exposure_image (image im, float sat)
void distort_image (image im, float hue, float sat, float val)
void saturate_exposure_image (image im, float sat, float exposure)
void rgb_to_hsv (image im)
void hsv_to_rgb (image im)
void yuv_to_rgb (image im)
void rgb_to_yuv (image im)
image collapse_image_layers (image source, int border)
image collapse_images_horz (image *ims, int n)
image collapse_images_vert (image *ims, int n)
void show_image_normalized (image im, const char *name)
void show_images (image *ims, int n, char *window)
void show_image_layers (image p, char *name)
void show_image_collapsed (image p, char *name)
void print_image (image m)
image make_empty_image (int w, int h, int c)
void copy_image_into (image src, image dest)
float get_pixel (image m, int x, int y, int c)
float get_pixel_extend (image m, int x, int y, int c)
void set_pixel (image m, int x, int y, int c, float val)
void add_pixel (image m, int x, int y, int c, float val)
float bilinear_interpolate (image im, float x, float y, int c)
image get_image_layer (image m, int l)
```

```
file layer.c
#include "layer.h" #include "cuda.h" #include <stdlib.h>
```

Functions

void **free_layer** (*layer l*)

```
file layer.h
#include "darknet.h"
```

```
file list.c
#include <stdlib.h>#include <string.h>#include "list.h"
```

Functions

list ***make_list** ()

void ***list_pop** (*list *l*)

void **list_insert** (*list *l*, void **val*)

void **free_node** (*node *n*)

void **free_list** (*list *l*)

void **free_list_contents** (*list *l*)

void ****list_to_array** (*list *l*)

```
file list.h
#include "darknet.h"
```

Functions

list ***make_list** ()

int **list_find** (*list *l*, void **val*)

void **list_insert** (*list **, void *)

void **free_list_contents** (*list *l*)

```
file local_layer.c
#include "local_layer.h"#include "utils.h"#include "im2col.h"#include "col2im.h"#include "blas.h"#include
"gemm.h"#include <stdio.h>#include <time.h>
```

Functions

int **local_out_height** (*local_layer l*)

int **local_out_width** (*local_layer l*)

local_layer **make_local_layer** (int *batch*, int *h*, int *w*, int *c*, int *n*, int *size*, int *stride*, int *pad*, *ACTI-*
VATION *activation*)

void **forward_local_layer** (const *local_layer l*, *network net*)

void **backward_local_layer** (*local_layer l*, *network net*)

void **update_local_layer** (*local_layer l*, *update_args a*)

```
file local_layer.h
#include "cuda.h"#include "image.h"#include "activations.h"#include "layer.h"#include "network.h"
```

Typedefs

typedef *layer* local_layer

Functions

local_layer make_local_layer (int batch, int h, int w, int c, int n, int size, int stride, int pad, *ACTIVATION* activation)

void forward_local_layer (const *local_layer* layer, *network* net)

void backward_local_layer (*local_layer* layer, *network* net)

void update_local_layer (*local_layer* layer, *update_args* a)

void bias_output (float *output, float *biases, int batch, int n, int size)

void backward_bias (float *bias_updates, float *delta, int batch, int n, int size)

file lstm_layer.c

```
#include "lstm_layer.h" #include "connected_layer.h" #include "utils.h" #include "cuda.h" #include "blas.h" #include "gemm.h" #include <math.h> #include <stdio.h> #include <stdlib.h> #include <string.h>
```

Functions

static void increment_layer (*layer* *l, int steps)

layer make_lstm_layer (int batch, int inputs, int outputs, int steps, int batch_normalize, int adam)

void update_lstm_layer (*layer* l, *update_args* a)

void forward_lstm_layer (*layer* l, *network* state)

void backward_lstm_layer (*layer* l, *network* state)

file lstm_layer.h

```
#include "activations.h" #include "layer.h" #include "network.h"
```

Defines

USET

Functions

layer make_lstm_layer (int batch, int inputs, int outputs, int steps, int batch_normalize, int adam)

void forward_lstm_layer (*layer* l, *network* net)

void update_lstm_layer (*layer* l, *update_args* a)

file matrix.c

```
#include "matrix.h" #include "utils.h" #include "blas.h" #include <stdio.h> #include <stdlib.h> #include <string.h> #include <assert.h> #include <math.h>
```

Functions

```

void free_matrix (matrix m)

float matrix_topk_accuracy (matrix truth, matrix guess, int k)

void scale_matrix (matrix m, float scale)

matrix resize_matrix (matrix m, int size)

void matrix_add_matrix (matrix from, matrix to)

matrix copy_matrix (matrix m)

matrix make_matrix (int rows, int cols)

matrix hold_out_matrix (matrix *m, int n)

float *pop_column (matrix *m, int c)

matrix csv_to_matrix (char *filename)

void matrix_to_csv (matrix m)

void print_matrix (matrix m)

```

```

file matrix.h
#include "darknet.h"

```

Functions

```

matrix copy_matrix (matrix m)

void print_matrix (matrix m)

matrix hold_out_matrix (matrix *m, int n)

matrix resize_matrix (matrix m, int size)

float *pop_column (matrix *m, int c)

```

```

file maxpool_layer.c
#include "maxpool_layer.h" #include "cuda.h" #include <stdio.h>

```

Functions

```

image get_maxpool_image (maxpool_layer l)

image get_maxpool_delta (maxpool_layer l)

maxpool_layer make_maxpool_layer (int batch, int h, int w, int c, int size, int stride, int padding)

void resize_maxpool_layer (maxpool_layer *l, int w, int h)

void forward_maxpool_layer (const maxpool_layer l, network net)

void backward_maxpool_layer (const maxpool_layer l, network net)

```

```

file maxpool_layer.h
#include "image.h" #include "cuda.h" #include "layer.h" #include "network.h"

```

Typedefs

typedef *layer* **maxpool_layer**

Functions

image **get_maxpool_image** (*maxpool_layer* l)

maxpool_layer **make_maxpool_layer** (int batch, int h, int w, int c, int size, int stride, int padding)

void **resize_maxpool_layer** (*maxpool_layer* *l, int w, int h)

void **forward_maxpool_layer** (const *maxpool_layer* l, *network* net)

void **backward_maxpool_layer** (const *maxpool_layer* l, *network* net)

file **network.c**

```
#include <stdio.h>#include <time.h>#include <assert.h>#include "network.h"#include "image.h"#include  
"data.h"#include "utils.h"#include "blas.h"#include "crop_layer.h"#include "connected_layer.h"#include  
"gru_layer.h"#include "rnn_layer.h"#include "crnn_layer.h"#include "local_layer.h"#include "convolu-  
tional_layer.h"#include "activation_layer.h"#include "detection_layer.h"#include "region_layer.h"#include  
"normalization_layer.h"#include "batchnorm_layer.h"#include "maxpool_layer.h"#include "re-  
org_layer.h"#include "avgpool_layer.h"#include "cost_layer.h"#include "softmax_layer.h"#include  
"dropout_layer.h"#include "route_layer.h"#include "shortcut_layer.h"#include "parser.h"
```

Functions

load_args **get_base_args** (*network* net)

network **load_network** (char *cfg, char *weights, int clear)

network ***load_network_p** (char *cfg, char *weights, int clear)

size_t **get_current_batch** (*network* net)

void **reset_momentum** (*network* net)

float **get_current_rate** (*network* net)

char ***get_layer_string** (*LAYER_TYPE* a)

network **make_network** (int n)

void **forward_network** (*network* net)

void **update_network** (*network* net)

void **calc_network_cost** (*network* net)

int **get_predicted_class_network** (*network* net)

void **backward_network** (*network* net)

float **train_network_datum** (*network* net)

float **train_network_sgd** (*network* net, *data* d, int n)

float **train_network** (*network* net, *data* d)

void **set_batch_network** (*network* *net, int b)

int **resize_network** (*network* *net, int w, int h)

```

detection_layer get_network_detection_layer (network net)
image get_network_image_layer (network net, int i)
image get_network_image (network net)
void visualize_network (network net)
void top_predictions (network net, int k, int *index)
float *network_predict (network net, float *input)
float *network_predict_p (network *net, float *input)
float *network_predict_image (network *net, image im)
int network_width (network *net)
int network_height (network *net)
matrix network_predict_data_multi (network net, data test, int n)
matrix network_predict_data (network net, data test)
void print_network (network net)
void compare_networks (network n1, network n2, data test)
float network_accuracy (network net, data d)
float *network_accuracies (network net, data d, int n)
layer get_network_output_layer (network net)
float network_accuracy_multi (network net, data d, int n)
void free_network (network net)
layer network_output_layer (network net)
int network_inputs (network net)
int network_outputs (network net)
float *network_output (network net)

```

file **network.h**

```
#include "darknet.h" #include "image.h" #include "layer.h" #include "data.h" #include "tree.h"
```

Functions

```

void compare_networks (network n1, network n2, data d)
char *get_layer_string (LAYER_TYPE a)
network make_network (int n)
float network_accuracy_multi (network net, data d, int n)
int get_predicted_class_network (network net)
void print_network (network net)
int resize_network (network *net, int w, int h)
void calc_network_cost (network net)

```

```
file normalization_layer.c
#include "normalization_layer.h" #include "blas.h" #include <stdio.h>
```

Functions

```
layer make_normalization_layer (int batch, int w, int h, int c, int size, float alpha, float beta, float
                                kappa)
void resize_normalization_layer (layer *layer, int w, int h)
void forward_normalization_layer (const layer layer, network net)
void backward_normalization_layer (const layer layer, network net)
```

```
file normalization_layer.h
#include "image.h" #include "layer.h" #include "network.h"
```

Functions

```
layer make_normalization_layer (int batch, int w, int h, int c, int size, float alpha, float beta, float
                                kappa)
void resize_normalization_layer (layer *layer, int h, int w)
void forward_normalization_layer (const layer layer, network net)
void backward_normalization_layer (const layer layer, network net)
void visualize_normalization_layer (layer layer, char *window)
```

```
file option_list.c
#include <stdlib.h> #include <stdio.h> #include <string.h> #include "option_list.h" #include "utils.h"
```

Functions

```
list *read_data_cfg (char *filename)
metadata get_metadata (char *file)
int read_option (char *s, list *options)
void option_insert (list *l, char *key, char *val)
void option_unused (list *l)
char *option_find (list *l, char *key)
char *option_find_str (list *l, char *key, char *def)
int option_find_int (list *l, char *key, int def)
int option_find_int_quiet (list *l, char *key, int def)
float option_find_float_quiet (list *l, char *key, float def)
float option_find_float (list *l, char *key, float def)
```

```
file option_list.h
#include "list.h"
```


Functions

```
int read_option (char *s, list *options)
void option_insert (list *l, char *key, char *val)
char *option_find (list *l, char *key)
int option_find_int_quiet (list *l, char *key, int def)
float option_find_float (list *l, char *key, float def)
float option_find_float_quiet (list *l, char *key, float def)
void option_unused (list *l)
```

file **parser.c**

```
#include <stdio.h>#include <string.h>#include <stdlib.h>#include <assert.h>#include "activation_layer.h"#include "activations.h"#include "avgpool_layer.h"#include "batchnorm_layer.h"#include "blas.h"#include "connected_layer.h"#include "deconvolutional_layer.h"#include "convolutional_layer.h"#include "cost_layer.h"#include "crnn_layer.h"#include "crop_layer.h"#include "detection_layer.h"#include "dropout_layer.h"#include "gru_layer.h"#include "list.h"#include "local_layer.h"#include "maxpool_layer.h"#include "normalization_layer.h"#include "option_list.h"#include "parser.h"#include "region_layer.h"#include "reorg_layer.h"#include "rnn_layer.h"#include "route_layer.h"#include "shortcut_layer.h"#include "softmax_layer.h"#include "lstm_layer.h"#include "utils.h"
```

Typedefs

```
typedef struct size_params size_params
```

Functions

```
list *read_cfg (char *filename)
LAYER_TYPE string_to_layer_type (char *type)
void free_section (section *s)
void parse_data (char *data, float *a, int n)
local_layer parse_local (list *options, size_params params)
layer parse_deconvolutional (list *options, size_params params)
convolutional_layer parse_convolutional (list *options, size_params params)
layer parse_crnn (list *options, size_params params)
layer parse_rnn (list *options, size_params params)
layer parse_gru (list *options, size_params params)
layer parse_lstm (list *options, size_params params)
layer parse_connected (list *options, size_params params)
softmax_layer parse_softmax (list *options, size_params params)
layer parse_region (list *options, size_params params)
detection_layer parse_detection (list *options, size_params params)
```

```
cost_layer parse_cost (list *options, size_params params)
crop_layer parse_crop (list *options, size_params params)
layer parse_reorg (list *options, size_params params)
maxpool_layer parse_maxpool (list *options, size_params params)
avgpool_layer parse_avgpool (list *options, size_params params)
dropout_layer parse_dropout (list *options, size_params params)
layer parse_normalization (list *options, size_params params)
layer parse_batchnorm (list *options, size_params params)
layer parse_shortcut (list *options, size_params params, network net)
layer parse_activation (list *options, size_params params)
route_layer parse_route (list *options, size_params params, network net)
learning_rate_policy get_policy (char *s)
void parse_net_options (list *options, network *net)
int is_network (section *s)
network parse_network_cfg (char *filename)
void save_convolutional_weights_binary (layer l, FILE *fp)
void save_convolutional_weights (layer l, FILE *fp)
void save_batchnorm_weights (layer l, FILE *fp)
void save_connected_weights (layer l, FILE *fp)
void save_weights_upto (network net, char *filename, int cutoff)
void save_weights (network net, char *filename)
void transpose_matrix (float *a, int rows, int cols)
void load_connected_weights (layer l, FILE *fp, int transpose)
void load_batchnorm_weights (layer l, FILE *fp)
void load_convolutional_weights_binary (layer l, FILE *fp)
void load_convolutional_weights (layer l, FILE *fp)
void load_weights_upto (network *net, char *filename, int start, int cutoff)
void load_weights (network *net, char *filename)
```

```
file parser.h
#include "darknet.h" #include "network.h"
```

Functions

```
void save_network (network net, char *filename)
void save_weights_double (network net, char *filename)
```

```
file region_layer.c
#include "region_layer.h" #include "activations.h" #include "blas.h" #include "box.h" #include
"cuda.h" #include "utils.h" #include <stdio.h> #include <assert.h> #include <string.h> #include <stdlib.h>
```

Functions

```

layer make_region_layer (int batch, int w, int h, int n, int classes, int coords)
void resize_region_layer (layer *l, int w, int h)
box get_region_box (float *x, float *biases, int n, int index, int i, int j, int w, int h, int stride)
float delta_region_box (box truth, float *x, float *biases, int n, int index, int i, int j, int w, int h, float
                        *delta, float scale, int stride)
void delta_region_mask (float *truth, float *x, int n, int index, float *delta, int stride, int scale)
void delta_region_class (float *output, float *delta, int index, int class, int classes, tree *hier, float
                        scale, int stride, float *avg_cat)

float logit (float x)
float tisnan (float x)
int entry_index (layer l, int batch, int location, int entry)
void forward_region_layer (const layer l, network net)
void backward_region_layer (const layer l, network net)
void correct_region_boxes (box *boxes, int n, int w, int h, int netw, int neth, int relative)
void get_region_boxes (layer l, int w, int h, int netw, int neth, float thresh, float **probs, box *boxes,
                      float **masks, int only_objectness, int *map, float tree_thresh, int relative)
void zero_objectness (layer l)

```

```

file region_layer.h
#include "darknet.h" #include "layer.h" #include "network.h"

```

Functions

```

layer make_reorg_layer (int batch, int h, int w, int n, int classes, int coords)
void forward_region_layer (const layer l, network net)
void backward_region_layer (const layer l, network net)
void resize_region_layer (layer *l, int w, int h)

```

```

file reorg_layer.c
#include "reorg_layer.h" #include "cuda.h" #include "blas.h" #include <stdio.h>

```

Functions

```

layer make_reorg_layer (int batch, int w, int h, int c, int stride, int reverse, int flatten, int extra)
void resize_reorg_layer (layer *l, int w, int h)
void forward_reorg_layer (const layer l, network net)
void backward_reorg_layer (const layer l, network net)

```

```

file reorg_layer.h
#include "image.h" #include "cuda.h" #include "layer.h" #include "network.h"

```

Functions

layer **make_reorg_layer** (int *batch*, int *w*, int *h*, int *c*, int *stride*, int *reverse*, int *flatten*, int *extra*)

void **resize_reorg_layer** (*layer* **l*, int *w*, int *h*)

void **forward_reorg_layer** (const *layer* *l*, *network* *net*)

void **backward_reorg_layer** (const *layer* *l*, *network* *net*)

file **rnn_layer.c**

```
#include "rnn_layer.h" #include "connected_layer.h" #include "utils.h" #include "cuda.h" #include  
"blas.h" #include "gemm.h" #include <math.h> #include <stdio.h> #include <stdlib.h> #include <string.h>
```

Functions

static void **increment_layer** (*layer* **l*, int *steps*)

layer **make_rnn_layer** (int *batch*, int *inputs*, int *outputs*, int *steps*, *ACTIVATION* *activation*, int
batch_normalize, int *adam*)

void **update_rnn_layer** (*layer* *l*, *update_args* *a*)

void **forward_rnn_layer** (*layer* *l*, *network* *net*)

void **backward_rnn_layer** (*layer* *l*, *network* *net*)

file **rnn_layer.h**

```
#include "activations.h" #include "layer.h" #include "network.h"
```

Defines

USET

Functions

layer **make_rnn_layer** (int *batch*, int *inputs*, int *outputs*, int *steps*, *ACTIVATION* *activation*, int
batch_normalize, int *adam*)

void **forward_rnn_layer** (*layer* *l*, *network* *net*)

void **backward_rnn_layer** (*layer* *l*, *network* *net*)

void **update_rnn_layer** (*layer* *l*, *update_args* *a*)

file **route_layer.c**

```
#include "route_layer.h" #include "cuda.h" #include "blas.h" #include <stdio.h>
```

Functions

route_layer **make_route_layer** (int *batch*, int *n*, int **input_layers*, int **input_sizes*)

void **resize_route_layer** (*route_layer* **l*, *network* **net*)

void **forward_route_layer** (const *route_layer* *l*, *network* *net*)

void **backward_route_layer** (const *route_layer* *l*, *network* *net*)

```
file route_layer.h
#include "network.h"#include "layer.h"
```

Typedefs

```
typedef layer route_layer
```

Functions

```
route_layer make_route_layer (int batch, int n, int *input_layers, int *input_size)
void forward_route_layer (const route_layer l, network net)
void backward_route_layer (const route_layer l, network net)
void resize_route_layer (route_layer *l, network *net)
```

```
file shortcut_layer.c
#include "shortcut_layer.h"#include "cuda.h"#include "blas.h"#include "activations.h"#include
<stdio.h>#include <assert.h>
```

Functions

```
layer make_shortcut_layer (int batch, int index, int w, int h, int c, int w2, int h2, int c2)
void forward_shortcut_layer (const layer l, network net)
void backward_shortcut_layer (const layer l, network net)
```

```
file shortcut_layer.h
#include "layer.h"#include "network.h"
```

Functions

```
layer make_shortcut_layer (int batch, int index, int w, int h, int c, int w2, int h2, int c2)
void forward_shortcut_layer (const layer l, network net)
void backward_shortcut_layer (const layer l, network net)
```

```
file softmax_layer.c
#include "softmax_layer.h"#include "blas.h"#include "cuda.h"#include <float.h>#include <math.h>#include
<stdlib.h>#include <stdio.h>#include <assert.h>
```

Functions

```
softmax_layer make_softmax_layer (int batch, int inputs, int groups)
void forward_softmax_layer (const softmax_layer l, network net)
void backward_softmax_layer (const softmax_layer l, network net)
```

```
file softmax_layer.h
#include "layer.h"#include "network.h"
```

Typedefs

```
typedef layer softmax_layer
```

Functions

```
void softmax_array(float *input, int n, float temp, float *output)
softmax_layer make_softmax_layer(int batch, int inputs, int groups)
void forward_softmax_layer(const softmax_layer l, network net)
void backward_softmax_layer(const softmax_layer l, network net)
```

file **stb_image.h**

```
#include <stdio.h>#include <stdarg.h>#include <stddef.h>#include <stdlib.h>#include <string.h>#include
<math.h>#include <assert.h>#include <stdint.h>
```

Defines

```
STBI_VERSION
```

```
STBIDEF
```

Typedefs

```
typedef unsigned char stbi_uc
```

Enums

```
enum [anonymous]
```

Values:

```
STBI_default = 0
```

```
STBI_grey = 1
```

```
STBI_grey_alpha = 2
```

```
STBI_rgb = 3
```

```
STBI_rgb_alpha = 4
```

Functions

```
STBIDEF stbi_uc* stbi_load(char const * filename, int * x, int * y, int * comp, int req_comp)
STBIDEF stbi_uc* stbi_load_from_memory(stbi_uc const * buffer, int len, int * x, int * y, int * comp, int req_comp)
STBIDEF stbi_uc* stbi_load_from_callbacks(stbi_io_callbacks const * clbk, void * user_data)
STBIDEF stbi_uc* stbi_load_from_file(FILE * f, int * x, int * y, int * comp, int req_comp)
STBIDEF float* stbi_loadf(char const * filename, int * x, int * y, int * comp, int req_comp)
STBIDEF float* stbi_loadf_from_memory(stbi_uc const * buffer, int len, int * x, int * y, int * comp, int req_comp)
```

```

STBIDEF float* stbi_loadf_from_callbacks(stbi_io_callbacks const * clbk, void * user,
STBIDEF float* stbi_loadf_from_file(FILE * f, int * x, int * y, int * comp, int req_comp)
STBIDEF void stbi_hdr_to_ldr_gamma(float gamma)
STBIDEF void stbi_hdr_to_ldr_scale(float scale)
STBIDEF void stbi_ldr_to_hdr_gamma(float gamma)
STBIDEF void stbi_ldr_to_hdr_scale(float scale)
STBIDEF int stbi_is_hdr_from_callbacks(stbi_io_callbacks const * clbk, void * user)
STBIDEF int stbi_is_hdr_from_memory(stbi_uc const * buffer, int len)
STBIDEF int stbi_is_hdr(char const * filename)
STBIDEF int stbi_is_hdr_from_file(FILE * f)
STBIDEF const char* stbi_failure_reason(void)
STBIDEF void stbi_image_free(void * retval_from_stbi_load)
STBIDEF int stbi_info_from_memory(stbi_uc const * buffer, int len, int * x, int * y, int * comp)
STBIDEF int stbi_info_from_callbacks(stbi_io_callbacks const * clbk, void * user, int * x, int * y, int * comp)
STBIDEF int stbi_info(char const * filename, int * x, int * y, int * comp)
STBIDEF int stbi_info_from_file(FILE * f, int * x, int * y, int * comp)
STBIDEF void stbi_set_unpremultiply_on_load(int flag_true_if_should_unpremultiply)
STBIDEF void stbi_convert_iphone_png_to_rgb(int flag_true_if_should_convert)
STBIDEF void stbi_set_flip_vertically_on_load(int flag_true_if_should_flip)
STBIDEF char* stbi_zlib_decode_malloc_guesssize(const char * buffer, int len, int initial_size, int * outlen)
STBIDEF char* stbi_zlib_decode_malloc_guesssize_headerflag(const char * buffer, int len, int initial_size, int * outlen, int header_flag)
STBIDEF char* stbi_zlib_decode_malloc(const char * buffer, int len, int * outlen)
STBIDEF int stbi_zlib_decode_buffer(char * obuffer, int olen, const char * ibuffer, int ilen)
STBIDEF char* stbi_zlib_decode_noheader_malloc(const char * buffer, int len, int * outlen)
STBIDEF int stbi_zlib_decode_noheader_buffer(char * obuffer, int olen, const char * ibuffer, int ilen)

```

file **stb_image_write.h**

```

#include <stdarg.h>#include <stdlib.h>#include <stdio.h>#include <string.h>#include <math.h>#include <assert.h>

```

Functions

```

int stbi_write_png(char const *filename, int w, int h, int comp, const void *data, int stride_in_bytes)
int stbi_write_bmp(char const *filename, int w, int h, int comp, const void *data)
int stbi_write_tga(char const *filename, int w, int h, int comp, const void *data)
int stbi_write_hdr(char const *filename, int w, int h, int comp, const float *data)

```

file **tree.c**

```

#include <stdio.h>#include <stdlib.h>#include "tree.h"#include "utils.h"#include "data.h"

```

Functions

```
void change_leaves (tree *t, char *leaf_list)  
float get_hierarchy_probability (float *x, tree *hier, int c, int stride)  
void hierarchy_predictions (float *predictions, int n, tree *hier, int only_leaves, int stride)  
int hierarchy_top_prediction (float *predictions, tree *hier, float thresh, int stride)  
tree *read_tree (char *filename)
```

```
file tree.h  
#include "darknet.h"
```

Functions

```
tree *read_tree (char *filename)  
int hierarchy_top_prediction (float *predictions, tree *hier, float thresh, int stride)  
float get_hierarchy_probability (float *x, tree *hier, int c, int stride)
```

```
file utils.c  
#include <stdio.h>#include <stdlib.h>#include <string.h>#include <math.h>#include <assert.h>#include  
<unistd.h>#include <float.h>#include <limits.h>#include <time.h>#include "utils.h"
```

Functions

```
double what_time_is_it_now ()  
int *read_intlist (char *gpu_list, int *ngpus, int d)  
int *read_map (char *filename)  
void sorta_shuffle (void *arr, size_t n, size_t size, size_t sections)  
void shuffle (void *arr, size_t n, size_t size)  
void del_arg (int argc, char **argv, int index)  
int find_arg (int argc, char *argv[], char *arg)  
int find_int_arg (int argc, char **argv, char *arg, int def)  
float find_float_arg (int argc, char **argv, char *arg, float def)  
char *find_char_arg (int argc, char **argv, char *arg, char *def)  
char *basecfg (char *cfgfile)  
int alphanum_to_int (char c)  
char int_to_alphanum (int i)  
void pm (int M, int N, float *A)  
void find_replace (char *str, char *orig, char *rep, char *output)  
float sec (clock_t clocks)  
void top_k (float *a, int n, int k, int *index)  
void error (const char *s)
```



```

void malloc_error ()
void file_error (char *s)
list *split_str (char *s, char delim)
void strip (char *s)
void strip_char (char *s, char bad)
void free_ptrs (void **ptrs, int n)
char *fgetl (FILE *fp)
int read_int (int fd)
void write_int (int fd, int n)
int read_all_fail (int fd, char *buffer, size_t bytes)
int write_all_fail (int fd, char *buffer, size_t bytes)
void read_all (int fd, char *buffer, size_t bytes)
void write_all (int fd, char *buffer, size_t bytes)
char *copy_string (char *s)
list *parse_csv_line (char *line)
int count_fields (char *line)
float *parse_fields (char *line, int n)
float sum_array (float *a, int n)
float mean_array (float *a, int n)
void mean_arrays (float **a, int n, int els, float *avg)
void print_statistics (float *a, int n)
float variance_array (float *a, int n)
int constrain_int (int a, int min, int max)
float constrain (float min, float max, float a)
float dist_array (float *a, float *b, int n, int sub)
float mse_array (float *a, int n)
void normalize_array (float *a, int n)
void translate_array (float *a, int n, float s)
float mag_array (float *a, int n)
void scale_array (float *a, int n, float s)
int sample_array (float *a, int n)
int max_index (float *a, int n)
int rand_int (int min, int max)
float rand_normal ()
size_t rand_size_t ()
float rand_uniform (float min, float max)

```

float **rand_scale** (float *s*)

float ****one_hot_encode** (float **a*, int *n*, int *k*)

file **utils.h**

#include <stdio.h>#include <time.h>#include "darknet.h"#include "list.h"

Defines

TWO_PI

Functions

double **what_time_is_it_now** ()

void **shuffle** (void **arr*, size_t *n*, size_t *size*)

void **sorta_shuffle** (void **arr*, size_t *n*, size_t *size*, size_t *sections*)

void **free_ptrs** (void ***ptrs*, int *n*)

int **alphanum_to_int** (char *c*)

char **int_to_alphanum** (int *i*)

int **read_int** (int *fd*)

void **write_int** (int *fd*, int *n*)

void **read_all** (int *fd*, char **buffer*, size_t *bytes*)

void **write_all** (int *fd*, char **buffer*, size_t *bytes*)

int **read_all_fail** (int *fd*, char **buffer*, size_t *bytes*)

int **write_all_fail** (int *fd*, char **buffer*, size_t *bytes*)

void **find_replace** (char **str*, char **orig*, char **rep*, char **output*)

void **malloc_error** ()

void **file_error** (char **s*)

void **strip** (char **s*)

void **strip_char** (char **s*, char *bad*)

list ***split_str** (char **s*, char *delim*)

char ***fgetl** (FILE **fp*)

list ***parse_csv_line** (char **line*)

char ***copy_string** (char **s*)

int **count_fields** (char **line*)

float ***parse_fields** (char **line*, int *n*)

void **scale_array** (float **a*, int *n*, float *s*)

void **translate_array** (float **a*, int *n*, float *s*)

float **constrain** (float *min*, float *max*, float *a*)

int **constrain_int** (int *a*, int *min*, int *max*)

```

float rand_uniform (float min, float max)
float rand_scale (float s)
int rand_int (int min, int max)
float sum_array (float *a, int n)
void mean_arrays (float **a, int n, int els, float *avg)
float dist_array (float *a, float *b, int n, int sub)
float **one_hot_encode (float *a, int n, int k)
float sec (clock_t clocks)
void print_statistics (float *a, int n)

```

file **example.py**

file **libyolo.c**

```

#include <stdio.h>#include <string.h>#include <stdlib.h>#include <unistd.h>#include "option_list.h"#include "network.h"#include "parser.h"#include "region_layer.h"#include "utils.h"#include "libyolo.h"

```

Functions

```

void get_detection_info (image im, int num, float thresh, box *boxes, float **probs, int classes, char **names, list *output)
yolo_handle yolo_init (char *darknet_path, char *datacfg, char *cfgfile, char *weightfile)
void yolo_cleanup (yolo_handle handle)
detection_info **yolo_detect (yolo_handle handle, image im, float thresh, float hier_thresh, int *num)
detection_info **yolo_test (yolo_handle handle, char *filename, float thresh, float hier_thresh, int *num)

```

file **libyolo.h**

```

#include "../darknet/src/image.h"

```

Typedefs

```

typedef void *yolo_handle

```

Functions

```

yolo_handle yolo_init (char *darknet_path, char *datacfg, char *cfgfile, char *weightfile)
void yolo_cleanup (yolo_handle handle)
detection_info **yolo_detect (yolo_handle handle, image im, float thresh, float hier_thresh, int *num)
detection_info **yolo_test (yolo_handle handle, char *filename, float thresh, float hier_thresh, int *num)

```

file **module.c**

```

#include <Python.h>#include <numpy/arrayobject.h>#include <stdio.h>#include "libyolo.h"#include "../darknet/src/image.h"

```

Defines

`NPY_NO_DEPRECATED_API`

Functions

```
static PyObject *pyyolo_init (PyObject *self, PyObject *args)
static PyObject *pyyolo_cleanup (PyObject *self, PyObject *args)
static PyObject *pyyolo_detect (PyObject *self, PyObject *args)
static PyObject *pyyolo_test (PyObject *self, PyObject *args)
PyMODINIT_FUNC initypyyolo (void)
```

Variables

PyObject *PyYoloError

yolo_handle `g_handle` = NULL

PyMethodDef `pyyolo_methods`[] = { {“init”, pyyolo_init, METH_VARARGS, “Initialize YOLO.”}, {“cleanup”, pyyolo_cleanup, METH_NOARGS, “Cleanup YOLO.”}, {0, 0}}

file `setup.py`

file `setup_gpu.py`

file `RecogniseFace.py`

file `RoboyVision.py`

file `RosMsgUtil.py`

file `SpeakerDetect.py`

file `TransformCoordinates.py`

file `CMakeLists.txt`

Functions

`cmake_minimum_required(VERSION 2.8.3)`

file `face_detection.py`

file `Visualizer.py`

file `Camera.hpp`

```
#include "sl/Core.hpp" #include "sl/Mesh.hpp" #include "sl/defines.hpp" #include <cuda.h> #include <opencv2/opencv.hpp>
```

file `Core.hpp`

```
#include <sl/types.hpp>
```

file `defines.hpp`

```
#include <stdint> #include <cstring> #include <iostream> #include <vector> #include <limits> #include <unistd.h>
```

Defines

```

TOO_FAR
TOO_CLOSE
OCCLUSION_VALUE
isValidMeasure (v)

```

Variables

```

const int ZED_SDK_MAJOR_VERSION = 2
const int ZED_SDK_MINOR_VERSION = 0
const int ZED_SDK_PATCH_VERSION = 1

```

file **Mesh.hpp**

```
#include <vector>#include <string>#include <fstream>#include <sl/Core.hpp>
```

Defines

```
SL_EXPORT_DLL
```

file **types.hpp**

```

#include <algorithm>#include <chrono>#include <cmath>#include <cstdio>#include <cstdlib>#include
<cstring>#include <ctime>#include <ctype.h>#include <fstream>#include <iomanip>#include
<iostream>#include <memory.h>#include <mutex>#include <sstream>#include <thread>#include
<vector>#include <cuda.h>#include <cuda_runtime.h>#include <cuda_runtime_api.h>#include <de-
vice_launch_parameters.h>#include <unistd.h>

```

Defines

```

__CUSTOM__PRETTY__FUNC__
__FILENAME__
_FCT_CPU_GPU_
IS_FINITE (x)
ZEDcudaSafeCall (err)
TIMING
INIT_TIMER
START_TIMER
DEF_START_TIMER
STOP_TIMER (name)

```

file **utils.hpp**

```

#include <cstdio>#include <cstring>#include <signal.h>#include <cstdlib>#include <chrono>#include
<thread>#include <mutex>#include <sys/stat.h>#include <opencv2/opencv.hpp>#include <sl/Camera.hpp>

```

Functions

```
bool testFileExist (std::string &filename)
void parse_args (int argc, char **argv, InfoOption &info)
void recordVideo (sl::Mat &image)
void recordImages (sl::Mat &image)
void initActions (sl::Camera *zed, InfoOption &modes)
void manageActions (sl::Camera *zed, char &key, InfoOption &modes)
void exitActions ()
void generateImageToRecord (sl::Camera *zed, InfoOption &modes, sl::Mat &out)
```

file **zed.cpp**

```
#include <iostream>#include <opencv2/core/core.hpp>#include <opencv2/highgui/highgui.hpp>#include
<opencv2/imgproc/imgproc.hpp>#include <zed/Camera.hpp>
```

Functions

```
int main (int argc, char **argv)
```

Variables

```
char keyboard = ‘ ‘
```

file **zedRoboy.cpp**

```
#include <sl/Camera.hpp>#include “utils.hpp”
```

Functions

```
sl::Camera InitCamera ()
int main (int argc, char **argv)
```

file **zedRoboy.py**

group **Enumerations**

Enums

```
enum RESOLUTION
```

List the spatial mapping resolution presets.

Values:

```
RESOLUTION_HIGH
```

Create a detail geometry, requires lots of memory.

```
RESOLUTION_MEDIUM
```

Smalls variations in the geometry will disappear, useful for big object

```
RESOLUTION_LOW
```

Keeps only huge variations of the geometry , useful outdoor.

enum RANGE

List the spatial mapping depth range presets.

Values:

RANGE_NEAR

Only depth close to the camera will be used by the spatial mapping.

RANGE_MEDIUM

Medium depth range.

RANGE_FAR

Takes into account objects that are far, useful outdoor.

enum MEM

List available memory type.

Values:

MEM_CPU = 1

CPU Memory (Processor side).

MEM_GPU = 2

GPU Memory (Graphic card side).

enum COPY_TYPE

List available copy operation on *Mat*.

Values:

COPY_TYPE_CPU_CPU

copy data from CPU to CPU.

COPY_TYPE_CPU_GPU

copy data from CPU to GPU.

COPY_TYPE_GPU_GPU

copy data from GPU to GPU.

COPY_TYPE_GPU_CPU

copy data from GPU to CPU.

enum MAT_TYPE

List available *Mat* formats.

Values:

MAT_TYPE_32F_C1

float 1 channel.

MAT_TYPE_32F_C2

float 2 channels.

MAT_TYPE_32F_C3

float 3 channels.

MAT_TYPE_32F_C4

float 4 channels.

MAT_TYPE_8U_C1

unsigned char 1 channel.

MAT_TYPE_8U_C2

unsigned char 2 channels.

MAT_TYPE_8U_C3

unsigned char 3 channels.

MAT_TYPE_8U_C4

unsigned char 4 channels.

enum RESOLUTION

List available video resolutions.

Warning Since v1.0, VGA mode has been updated to WVGA (from 640*480 to 672*376) and requires a firmware update to function (≥ 1142). Firmware can be updated in the ZED Explorer.

Warning NVIDIA Jetson boards do not support all ZED video resolutions and framerates. For more information, please read the on-line API documentation.

Values:

RESOLUTION_HD2K

2208*1242, available framerates: 15 fps.

RESOLUTION_HD1080

1920*1080, available framerates: 15, 30 fps.

RESOLUTION_HD720

1280*720, available framerates: 15, 30, 60 fps.

RESOLUTION_VGA

672*376, available framerates: 15, 30, 60, 100 fps.

RESOLUTION_LAST

enum CAMERA_SETTINGS

List available camera settings for the ZED camera (contrast, hue, saturation, gain...).

Each enum defines one of those settings.

Values:

CAMERA_SETTINGS_BRIGHTNESS

Defines the brightness control. Affected value should be between 0 and 8.

CAMERA_SETTINGS_CONTRAST

Defines the contrast control. Affected value should be between 0 and 8.

CAMERA_SETTINGS_HUE

Defines the hue control. Affected value should be between 0 and 11.

CAMERA_SETTINGS_SATURATION

Defines the saturation control. Affected value should be between 0 and 8.

CAMERA_SETTINGS_GAIN

Defines the gain control. Affected value should be between 0 and 100 for manual control. If ZED_EXPOSURE is set to -1, the gain is in auto mode too.

CAMERA_SETTINGS_EXPOSURE

Defines the exposure control. A -1 value enable the AutoExposure/AutoGain control, as the boolean parameter (default) does. Affected value should be between 0 and 100 for manual control.

CAMERA_SETTINGS_WHITEBALANCE

Defines the color temperature control. Affected value should be between 2800 and 6500 with a step of 100. A value of -1 set the AWB (auto white balance), as the boolean parameter (default) does.

CAMERA_SETTINGS_AUTO_WHITEBALANCE

Defines the status of white balance (automatic or manual). A value of 0 disable the AWB, while 1 activate it.

CAMERA_SETTINGS_LAST**enum SELF_CALIBRATION_STATE**

Status for self calibration.

Since v0.9.3, self-calibration is done in background and start in the *sl::Camera::open* or Reset function. You can follow the current status for the self-calibration any time once ZED object has been construct.

Values:

SELF_CALIBRATION_STATE_NOT_STARTED

Self calibration has not run yet (no *sl::Camera::open* or *sl::Camera::resetSelfCalibration* called).

SELF_CALIBRATION_STATE_RUNNING

Self calibration is currently running.

SELF_CALIBRATION_STATE_FAILED

Self calibration has finished running but did not manage to get accurate values. Old parameters are taken instead.

SELF_CALIBRATION_STATE_SUCCESS

Self calibration has finished running and did manage to get accurate values. New parameters are set.

SELF_CALIBRATION_STATE_LAST**enum DEPTH_MODE**

List available depth computation modes.

Values:

DEPTH_MODE_NONE

This mode does not compute any depth map. Only rectified stereo images will be available.

DEPTH_MODE_PERFORMANCE

Fastest mode for depth computation.

DEPTH_MODE_MEDIUM

Balanced quality mode. Depth map is robust in any environment and requires medium resources for computation.

DEPTH_MODE_QUALITY

Best quality mode. Requires more compute power.

DEPTH_MODE_LAST**enum SENSING_MODE**

List available depth sensing modes.

Values:

SENSING_MODE_STANDARD

This mode outputs ZED standard depth map that preserves edges and depth accuracy. Applications example: Obstacle detection, Automated navigation, People detection, 3D reconstruction.

SENSING_MODE_FILL

This mode outputs a smooth and fully dense depth map. Applications example: AR/VR, Mixed-reality capture, Image post-processing.

SENSING_MODE_LAST

enum UNIT

Enumerate for available metric unit of the depth.

Values:

UNIT_MILLIMETER

UNIT_CENTIMETER

UNIT_METER

UNIT_INCH

UNIT_FOOT

UNIT_LAST

enum COORDINATE_SYSTEM

List available coordinates systems for positional tracking and points cloud representation.

Positional tracking is provided in the given coordinates system.

Values:

COORDINATE_SYSTEM_IMAGE

Standard coordinates system in computer vision. Used in OpenCV : see here : http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

COORDINATE_SYSTEM_LEFT_HANDED_Y_UP

Left-Handed with Y up and Z forward. Used in Unity with DirectX.

COORDINATE_SYSTEM_RIGHT_HANDED_Y_UP

Right-Handed with Y pointing up and Z backward. Used in OpenGL.

COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP

Right-Handed with Z pointing up and Y forward. Used in 3DSMax.

COORDINATE_SYSTEM_LEFT_HANDED_Z_UP

Left-Handed with Z axis pointing up and X forward. Used in Unreal Engine.

COORDINATE_SYSTEM_LAST

enum MEASURE

List retrievable measures.

Values:

MEASURE_DISPARITY

Disparity map, 1 channel, FLOAT.

MEASURE_DEPTH

Depth map, 1 channel, FLOAT.

MEASURE_CONFIDENCE

Certainty/confidence of the disparity map, 1 channel, FLOAT.

MEASURE_XYZ

Point cloud, 4 channels, FLOAT, channel 4 is empty.

MEASURE_XYZRGBA

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in R-G-B-A order.

MEASURE_XYZBGRA

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in B-G-R-A order.

MEASURE_XYZARGB

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in A-R-G-B order.

MEASURE_XYZABGR

Colored point cloud, 4 channels, FLOAT, channel 4 contains color in A-B-G-R order.

MEASURE_LAST**enum VIEW**

List available views.

Values:

VIEW_LEFT

Rectified left image.

VIEW_RIGHT

Rectified right image.

VIEW_LEFT_GRAY

Rectified left grayscale image.

VIEW_RIGHT_GRAY

Rectified right grayscale image.

VIEW_LEFT_UNRECTIFIED

Raw left image.

VIEW_RIGHT_UNRECTIFIED

Raw right image.

VIEW_LEFT_UNRECTIFIED_GRAY

Raw left grayscale image.

VIEW_RIGHT_UNRECTIFIED_GRAY

Raw right grayscale image.

VIEW_SIDE_BY_SIDE

Left and right image (the image width is therefore doubled).

VIEW_DEPTH

Normalized depth image.

VIEW_CONFIDENCE

Normalized confidence image.

VIEW_LAST**enum DEPTH_FORMAT**

List available file formats for saving depth maps.

Values:

DEPTH_FORMAT_PNG

PNG image format in 16bits. 32bits depth is mapped to 16bits color image to preserve the consistency of the data range.

DEPTH_FORMAT_PFM

stream of bytes, graphic image file format.

DEPTH_FORMAT_PGM

gray-scale image format.

DEPTH_FORMAT_LAST

enum POINT_CLOUD_FORMAT

List available file formats for saving point clouds.

Stores the spatial coordinates (x,y,z) of each pixel and optionally its RGB color.

Values:

POINT_CLOUD_FORMAT_XYZ_ASCII

Generic point cloud file format, without color information.

POINT_CLOUD_FORMAT_PCD_ASCII

Point Cloud Data file, with color information.

POINT_CLOUD_FORMAT_PLY_ASCII

PoLYgon file format, with color information.

POINT_CLOUD_FORMAT_VTK_ASCII

Visualization ToolKit file, without color information.

POINT_CLOUD_FORMAT_LAST

enum TRACKING_STATE

List the different states of positional tracking.

Values:

TRACKING_STATE_SEARCHING

The camera is searching for a previously known position to locate itself.

TRACKING_STATE_OK

Positional tracking is working normally.

TRACKING_STATE_OFF

Positional tracking is not enabled.

TRACKING_STATE_FPS_TOO_LOW

Effective FPS is too low to give proper results for motion tracking. Consider using PERFORMANCES parameters (DEPTH_MODE_PERFORMANCE, low camera resolution (VGA,HD720))

TRACKING_STATE_LAST

enum AREA_EXPORT_STATE

List the different states of spatial memory area export.

Values:

AREA_EXPORT_STATE_SUCCESS

The spatial memory file has been successfully created.

AREA_EXPORT_STATE_RUNNING

The spatial memory is currently written.

AREA_EXPORT_STATE_NOT_STARTED

The spatial memory file exportation has not been called.

AREA_EXPORT_STATE_FILE_EMPTY

The spatial memory contains no data, the file is empty.

AREA_EXPORT_STATE_FILE_ERROR

The spatial memory file has not been written because of a wrong file name.

AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED

The spatial memory learning is disable, no file can be created.

AREA_EXPORT_STATE_LAST

enum REFERENCE_FRAME

Define which type of position matrix is used to store camera path and pose.

Values:

REFERENCE_FRAME_WORLD

The transform of *sl::Pose* will contains the motion with reference to the world frame (previously called PATH).

REFERENCE_FRAME_CAMERA

The transform of *sl::Pose* will contains the motion with reference to the previous camera frame (previously called POSE).

REFERENCE_FRAME_LAST**enum SPATIAL_MAPPING_STATE**

Gives the spatial mapping state.

Values:

SPATIAL_MAPPING_STATE_INITIALIZING

The spatial mapping is initializing.

SPATIAL_MAPPING_STATE_OK

The depth and tracking data were correctly integrated in the fusion algorithm.

SPATIAL_MAPPING_STATE_NOT_ENOUGH_MEMORY

The maximum memory dedicated to the scanning has been reach, the mesh will no longer be updated.

SPATIAL_MAPPING_STATE_NOT_ENABLED

Camera::enableSpatialMapping() wasn't called (or the scanning was stopped and not relaunched).

SPATIAL_MAPPING_STATE_FPS_TOO_LOW

Effective FPS is too low to give proper results for spatial mapping. Consider using PERFORMANCES parameters (DEPTH_MODE_PERFORMANCE, low camera resolution (VGA,HD720), spatial mapping low resolution)

SPATIAL_MAPPING_STATE_LAST**enum SVO_COMPRESSION_MODE**

List available compression modes for SVO recording.

sl::SVO_COMPRESSION_MODE_LOSSLESS is an improvement of previous lossless compression (used in ZED Explorer), even if size may be bigger, compression time is much faster.

Values:

SVO_COMPRESSION_MODE_RAW

RAW images, no compression.

SVO_COMPRESSION_MODE_LOSSLESS

new Lossless, with PNG/ZSTD based compression : avg size = 42% of RAW).

SVO_COMPRESSION_MODE_LOSSY

new Lossy, with JPEG based compression : avg size = 22% of RAW).

SVO_COMPRESSION_MODE_LAST**enum ERROR_CODE**

List error codes in the ZED SDK.

Values:

SUCCESS

Standard code for successful behavior.

ERROR_CODE_FAILURE

Standard code for unsuccessful behavior.

ERROR_CODE_NO_GPU_COMPATIBLE

No GPU found or CUDA capability of the device is not supported.

ERROR_CODE_NOT_ENOUGH_GPUMEM

Not enough GPU memory for this depth mode please try a faster mode (such as PERFORMANCE mode).

ERROR_CODE_CAMERA_NOT_DETECTED

The ZED camera is not plugged or detected.

ERROR_CODE_INVALID_RESOLUTION

For Jetson only, resolution not yet supported (USB3.0 bandwidth).

ERROR_CODE_LOW_USB_BANDWIDTH

This issue can occurs when you use multiple ZED or a USB 2.0 port (bandwidth issue).

ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE

ZED calibration file is not found on the host machine. Use ZED Explorer or ZED Calibration to get one.

ERROR_CODE_INVALID_SVO_FILE

The provided SVO file is not valid.

ERROR_CODE_SVO_RECORDING_ERROR

An recorder related error occurred (not enough free storage, invalid file).

ERROR_CODE_INVALID_COORDINATE_SYSTEM

The requested coordinate system is not available.

ERROR_CODE_INVALID_FIRMWARE

The firmware of the ZED is out of date. Update to the latest version.

ERROR_CODE_NOT_A_NEW_FRAME

in grab() only, the current call return the same frame as last call. Not a new frame.

ERROR_CODE_CUDA_ERROR

in grab() only, a CUDA error has been detected in the process. Activate verbose in *sl::Camera::open* for more info.

ERROR_CODE_CAMERA_NOT_INITIALIZED

in grab() only, ZED SDK is not initialized. Probably a missing call to *sl::Camera::open*.

ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE

your NVIDIA driver is too old and not compatible with your current CUDA version.

ERROR_CODE_INVALID_FUNCTION_CALL

the call of the function is not valid in the current context. Could be a missing call of *sl::Camera::open*.

ERROR_CODE_CORRUPTED_SDK_INSTALLATION

The SDK wasn't able to load its dependencies, the installer should be launched.

ERROR_CODE_LAST

page md_home_docs_checkouts_readthedocs.org_user_builds_robeyvision_checkouts_devel_src_pyy

doxyxml/darknet-black-small.png

#Darknet# Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.

For more information see the [Darknet project website](#).

For questions or issues please use the [Google Group](#).

page `md__home_docs_checkouts_readthedocs.org_user_builds_roboyvision_checkouts_devel_src_pyyolo`
 pyyolo is a simple wrapper for YOLO.

Building

- 1.git clone recursive <https://github.com/thomaspark-pkj/pyyolo.git>
- 2.(optional) Set GPU=1 and CUDNN=1 in Makefile to use GPU.
- 3.make
- 4.rm -rf build
- 5.python setup.py build (use setup_gpu.py for GPU)
- 6.sudo python setup.py install (use setup_gpu.py for GPU)

Test

Edit parameters in example.py

```
python example.py
```

Result

```
{'right': 194, 'bottom': 353, 'top': 264, 'class': 'dog', 'prob': 0.
↪8198755383491516, 'left': 71}
{'right': 594, 'bottom': 338, 'top': 109, 'class': 'horse', 'prob': 0.
↪6106302738189697, 'left': 411}
{'right': 274, 'bottom': 381, 'top': 101, 'class': 'person', 'prob': 0.
↪702547550201416, 'left': 184}
{'right': 583, 'bottom': 347, 'top': 137, 'class': 'sheep', 'prob': 0.
↪7186083197593689, 'left': 387}
```

```
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo/
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/data
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo/
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo/la
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo/
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/vision_
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo/
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/zed/sl
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/pyyolo/
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboyvision/checkouts/devel/src/vision_
```

`dir /home/docs/checkouts/readthedocs.org/user_builds/robeyvision/checkouts/devel/src/zed`

About arc42

This information should stay in every repository as per their license: <http://www.arc42.de/template/licence.html>

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 6.5 EN (based on asciidoc), Juni 2014

© We acknowledge that this document uses material from the arc 42 architecture template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke. For additional contributors see <http://arc42.de/sonstiges/contributors.html>

Note

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6. Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

Examples

- [HTML Sanity Checker](#)
- [DocChess](#) (german)
- [Gradle](#) (german)
- [MaMa CRM](#) (german)
- [Financial Data Migration](#) (german)

Acknowledgements and collaborations

arc42 originally envisioned by [Dr. Peter Hruschka](#) and [Dr. Gernot Starke](#).

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in [GitHub](#) under the [aim42-Organisation](#).

Issues We maintain a list of [open topics and bugs](#).

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer

Symbols

`_FCT_CPU_GPU_` (C macro), 145
`__CUSTOM_PRETTY_FUNC__` (C macro), 145
`__FILENAME__` (C macro), 145
`__anonymous0` (C++ type), 138

A

`abs_mean` (C++ function), 95
`activate` (C++ function), 105, 106
`activate_array` (C++ function), 105, 106
`ACTIVATION` (C++ type), 99
`ACTIVE` (C++ class), 99
`add_bias` (C++ function), 112, 113
`add_pixel` (C++ function), 125, 126
`alphanum_to_int` (C++ function), 140, 142
`args` (FreezeModel attribute), 73
`args` (RoboVision attribute), 77
`argtypes` (darknet attribute), 67
`argtypes` (FaceDetect attribute), 72
`argtypes` (ObjectRecognition attribute), 75
`augment_args` (C++ class), 14
`augment_args::aspect` (C++ member), 14
`augment_args::dx` (C++ member), 14
`augment_args::dy` (C++ member), 14
`augment_args::h` (C++ member), 14
`augment_args::rad` (C++ member), 14
`augment_args::scale` (C++ member), 14
`augment_args::w` (C++ member), 14
`average` (C++ function), 93
`AVGPOOL` (C++ class), 99
`avgpool_layer` (C++ type), 107
`axpy_cpu` (C++ function), 101, 108

B

`backward_activation_layer` (C++ function), 105
`backward_avgpool_layer` (C++ function), 107
`backward_batchnorm_layer` (C++ function), 107, 108
`backward_bias` (C++ function), 112, 113, 128
`backward_connected_layer` (C++ function), 111, 112

`backward_convolutional_layer` (C++ function), 112, 113
`backward_cost_layer` (C++ function), 114
`backward_crnn_layer` (C++ function), 114, 115
`backward_crop_layer` (C++ function), 115
`backward_crop_layer_gpu` (C++ function), 115
`backward_deconvolutional_layer` (C++ function), 119
`backward_detection_layer` (C++ function), 120
`backward_dropout_layer` (C++ function), 120, 121
`backward_gru_layer` (C++ function), 122
`backward_local_layer` (C++ function), 127, 128
`backward_lstm_layer` (C++ function), 128
`backward_maxpool_layer` (C++ function), 129, 130
`backward_network` (C++ function), 101, 130
`backward_normalization_layer` (C++ function), 132
`backward_region_layer` (C++ function), 135
`backward_reorg_layer` (C++ function), 135, 136
`backward_rnn_layer` (C++ function), 136
`backward_route_layer` (C++ function), 136, 137
`backward_scale_cpu` (C++ function), 107, 109
`backward_shortcut_layer` (C++ function), 137
`backward_softmax_layer` (C++ function), 137, 138
`basecfg` (C++ function), 104, 140
`BATCHNORM` (C++ class), 100
`BattleRoyaleWithCheese` (C++ function), 111
`bbox_comparator` (C++ function), 111
`bbox_fight` (C++ function), 111
`bbox_update` (C++ function), 111
`best_3d_shift` (C++ function), 124
`best_3d_shift_r` (C++ function), 101, 124
`bias_output` (C++ function), 128
`bilinear_interpolate` (C++ function), 125, 126
`binarize_cpu` (C++ function), 112
`binarize_image` (C++ function), 124
`binarize_input` (C++ function), 112
`binarize_weights` (C++ function), 112, 113
`binarize_weights2` (C++ function), 113
`BLANK` (C++ class), 100
`blend_image` (C++ function), 124
`board_to_string` (C++ function), 94
`border_image` (C++ function), 123

`bound_image` (C++ function), 116
`box` (C++ class), 14
`box::h` (C++ member), 14
`box::w` (C++ member), 14
`box::x` (C++ member), 14
`box::y` (C++ member), 14
`box_intersection` (C++ function), 110
`box_iou` (C++ function), 103, 110
`box_label` (C++ class), 14
`box_label::bottom` (C++ member), 15
`box_label::h` (C++ member), 15
`box_label::id` (C++ member), 15
`box_label::left` (C++ member), 15
`box_label::right` (C++ member), 15
`box_label::top` (C++ member), 15
`box_label::w` (C++ member), 15
`box_label::x` (C++ member), 15
`box_label::y` (C++ member), 15
`box_rmse` (C++ function), 110
`box_union` (C++ function), 110

C

`c` (example attribute), 68
`c` (face_detection attribute), 71
`calc_network_cost` (C++ function), 130, 131
`calculate_liberties` (C++ function), 94
`calculate_loss` (C++ function), 95
`cam` (example attribute), 68
`CameraQueue` (RoboyVision attribute), 77
`CAPTCHA_DATA` (C++ class), 100
`center_crop_image` (C++ function), 124, 125
`change_leaves` (C++ function), 104, 140
`ckpt_file` (face_detection attribute), 71
`CLASSIFICATION_DATA` (C++ class), 100
`clf` (face_detection attribute), 71
`coco_classes` (C++ member), 92
`coco_ids` (C++ member), 92, 94
`col2im_add_pixel` (C++ function), 111
`col2im_cpu` (C++ function), 111
`collapse_image_layers` (C++ function), 123, 126
`collapse_images_horz` (C++ function), 125, 126
`collapse_images_vert` (C++ function), 125, 126
`colors` (C++ member), 125
`COMPARE_DATA` (C++ class), 100
`compare_networks` (C++ function), 131
`composite_3d` (C++ function), 103, 124
`composite_image` (C++ function), 123
`concat_data` (C++ function), 101, 117
`concat_datas` (C++ function), 117, 119
`concat_matrix` (C++ function), 117
`CONNECTED` (C++ class), 99
`const_cpu` (C++ function), 108, 109
`CONSTANT` (C++ class), 100
`constrain` (C++ function), 141, 142

`constrain_gpu` (C++ function), 109
`constrain_image` (C++ function), 103, 123
`constrain_int` (C++ function), 141, 142
`CONVOLUTIONAL` (C++ class), 99
`convolutional_layer` (C++ type), 113
`convolutional_out_height` (C++ function), 112, 113
`convolutional_out_width` (C++ function), 112, 113
`copy_cpu` (C++ function), 101, 108
`copy_data` (C++ function), 101, 118
`copy_image` (C++ function), 103, 123
`copy_image_into` (C++ function), 123, 126
`copy_matrix` (C++ function), 129
`copy_string` (C++ function), 141, 142
`correct_boxes` (C++ function), 116
`correct_region_boxes` (C++ function), 135
`COST` (C++ class), 99
`cost_layer` (C++ type), 114
`COST_TYPE` (C++ type), 100
`count_fields` (C++ function), 141, 142
`CRNN` (C++ class), 99
`CROP` (C++ class), 99
`crop_image` (C++ function), 102, 124
`crop_layer` (C++ type), 115
`csv_to_matrix` (C++ function), 102, 129
`ctypes` (built-in class), 67
`current_class` (C++ member), 111

D

`d_img` (face_detection attribute), 71
`darknet` (built-in class), 67
`darknet.classify()` (built-in function), 67
`darknet.detect()` (built-in function), 67
`darknet.IMAGE` (built-in class), 25
`darknet.letterbox_img()` (built-in function), 67
`darknet.load_img()` (built-in function), 67
`darknet.load_meta()` (built-in function), 67
`darknet.load_net()` (built-in function), 67
`darknet.METADATA` (built-in class), 51
`darknet.predict()` (built-in function), 67
`data` (C++ class), 24
`data` (example attribute), 68
`data::boxes` (C++ member), 25
`data::h` (C++ member), 24
`data::num_boxes` (C++ member), 25
`data::shallow` (C++ member), 24
`data::w` (C++ member), 24
`data::X` (C++ member), 24
`data::y` (C++ member), 24
`data_type` (C++ type), 100
`dbbox` (C++ class), 25
`dbbox::dh` (C++ member), 25
`dbbox::dw` (C++ member), 25
`dbbox::dx` (C++ member), 25
`dbbox::dy` (C++ member), 25

dcnan_batch (C++ function), 95
 decode_box (C++ function), 110
 DECONVOLUTIONAL (C++ class), 99
 DEF_START_TIMER (C macro), 145
 deinter_cpu (C++ function), 108, 109
 del_arg (C++ function), 140
 delta_region_box (C++ function), 135
 delta_region_class (C++ function), 135
 delta_region_mask (C++ function), 135
 DEMO (C macro), 119
 demo (C++ function), 102, 120
 demo_art (C++ function), 90
 demo_classifier (C++ function), 91
 demo_regressor (C++ function), 96
 demo_segmenter (C++ function), 97
 denormalize_connected_layer (C++ function), 102, 112
 denormalize_convolutional_layer (C++ function), 102, 112
 denormalize_deconvolutional_layer (C++ function), 119
 denormalize_net (C++ function), 93
 derivative (C++ function), 110
 DescribeSceneSrv (built-in class), 67
 DescribeSceneSrv.service_callback() (built-in function), 67
 description (setup attribute), 78
 description (setup_gpu attribute), 78
 DET_DATA (C++ class), 100
 detect (FaceDetect attribute), 72
 detect (ObjectRecognition attribute), 75
 detectFaceProc (multiprocess attribute), 74
 detectFaceProc (RoboyVision attribute), 77
 detectFaces (RoboyVision attribute), 77
 DETECTION (C++ class), 99
 DETECTION_DATA (C++ class), 100
 detection_info (C++ class), 25
 detection_info::bottom (C++ member), 25
 detection_info::left (C++ member), 25
 detection_info::name (C++ member), 25
 detection_info::prob (C++ member), 25
 detection_info::right (C++ member), 25
 detection_info::top (C++ member), 25
 detection_layer (C++ type), 120
 dev (face_detection attribute), 71
 dice_labels (C++ member), 94
 dintersect (C++ function), 110
 diou (C++ function), 110
 dir (FreezeModel attribute), 73
 dist_array (C++ function), 141, 143
 distance_from_edge (C++ function), 118
 distort_image (C++ function), 124, 126
 do_nms (C++ function), 103, 110
 do_nms_obj (C++ function), 104, 110
 do_nms_sort (C++ function), 104, 110
 dot_cpu (C++ function), 108, 109

draw (face_detection attribute), 71
 draw_bbox (C++ function), 123, 125
 draw_box (C++ function), 123, 125
 draw_box_width (C++ function), 103, 123
 draw_detections (C++ function), 103, 123
 draw_label (C++ function), 123, 125
 DROPOUT (C++ class), 99
 dropout_layer (C++ type), 121
 dunion (C++ function), 110

E

elo_comparator (C++ function), 111
 ELU (C++ class), 99
 elu_activate (C++ function), 106
 elu_gradient (C++ function), 106
 embed_image (C++ function), 123, 126
 embeddings (face_detection attribute), 71
 encode_box (C++ function), 110
 engine_go (C++ function), 95
 entry_index (C++ function), 135
 Enumerations::AREA_EXPORT_STATE (C++ type), 152
 Enumerations::AREA_EXPORT_STATE_FILE_EMPTY (C++ class), 152
 Enumerations::AREA_EXPORT_STATE_FILE_ERROR (C++ class), 152
 Enumerations::AREA_EXPORT_STATE_LAST (C++ class), 152
 Enumerations::AREA_EXPORT_STATE_NOT_STARTED (C++ class), 152
 Enumerations::AREA_EXPORT_STATE_RUNNING (C++ class), 152
 Enumerations::AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED (C++ class), 152
 Enumerations::AREA_EXPORT_STATE_SUCCESS (C++ class), 152
 Enumerations::CAMERA_SETTINGS (C++ type), 148
 Enumerations::CAMERA_SETTINGS_AUTO_WHITEBALANCE (C++ class), 148
 Enumerations::CAMERA_SETTINGS_BRIGHTNESS (C++ class), 148
 Enumerations::CAMERA_SETTINGS_CONTRAST (C++ class), 148
 Enumerations::CAMERA_SETTINGS_EXPOSURE (C++ class), 148
 Enumerations::CAMERA_SETTINGS_GAIN (C++ class), 148
 Enumerations::CAMERA_SETTINGS_HUE (C++ class), 148
 Enumerations::CAMERA_SETTINGS_LAST (C++ class), 149
 Enumerations::CAMERA_SETTINGS_SATURATION (C++ class), 148

Enumerations::CAMERA_SETTINGS_WHITEBALANCE (C++ class), 148
 Enumerations::COORDINATE_SYSTEM (C++ type), 150
 Enumerations::COORDINATE_SYSTEM_IMAGE (C++ class), 150
 Enumerations::COORDINATE_SYSTEM_LAST (C++ class), 150
 Enumerations::COORDINATE_SYSTEM_LEFT_HANDED (C++ class), 150
 Enumerations::COORDINATE_SYSTEM_LEFT_HANDED_Z_UP (C++ class), 150
 Enumerations::COORDINATE_SYSTEM_RIGHT_HANDED (C++ class), 150
 Enumerations::COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP (C++ class), 150
 Enumerations::COPY_TYPE (C++ type), 147
 Enumerations::COPY_TYPE_CPU_CPU (C++ class), 147
 Enumerations::COPY_TYPE_CPU_GPU (C++ class), 147
 Enumerations::COPY_TYPE_GPU_CPU (C++ class), 147
 Enumerations::COPY_TYPE_GPU_GPU (C++ class), 147
 Enumerations::DEPTH_FORMAT (C++ type), 151
 Enumerations::DEPTH_FORMAT_LAST (C++ class), 151
 Enumerations::DEPTH_FORMAT_PFM (C++ class), 151
 Enumerations::DEPTH_FORMAT_PGM (C++ class), 151
 Enumerations::DEPTH_FORMAT_PNG (C++ class), 151
 Enumerations::DEPTH_MODE (C++ type), 149
 Enumerations::DEPTH_MODE_LAST (C++ class), 149
 Enumerations::DEPTH_MODE_MEDIUM (C++ class), 149
 Enumerations::DEPTH_MODE_NONE (C++ class), 149
 Enumerations::DEPTH_MODE_PERFORMANCE (C++ class), 149
 Enumerations::DEPTH_MODE_QUALITY (C++ class), 149
 Enumerations::ERROR_CODE (C++ type), 153
 Enumerations::ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE (C++ class), 154
 Enumerations::ERROR_CODE_CAMERA_NOT_DETECTED (C++ class), 154
 Enumerations::ERROR_CODE_CAMERA_NOT_INITIALIZED (C++ class), 154
 Enumerations::ERROR_CODE_CORRUPTED_SDK_INSTALLATION (C++ class), 154
 Enumerations::ERROR_CODE_CUDA_ERROR (C++ class), 154
 Enumerations::ERROR_CODE_FAILURE (C++ class), 153
 Enumerations::ERROR_CODE_INVALID_COORDINATE_SYSTEM (C++ class), 154
 Enumerations::ERROR_CODE_INVALID_FIRMWARE (C++ class), 154
 Enumerations::ERROR_CODE_INVALID_FUNCTION_CALL (C++ class), 154
 Enumerations::ERROR_CODE_INVALID_RESOLUTION (C++ class), 154
 Enumerations::ERROR_CODE_INVALID_SVO_FILE (C++ class), 154
 Enumerations::ERROR_CODE_LAST (C++ class), 154
 Enumerations::ERROR_CODE_LOW_USB_BANDWIDTH (C++ class), 154
 Enumerations::ERROR_CODE_NO_GPU_COMPATIBLE (C++ class), 154
 Enumerations::ERROR_CODE_NOT_A_NEW_FRAME (C++ class), 154
 Enumerations::ERROR_CODE_NOT_ENOUGH_GPUMEM (C++ class), 154
 Enumerations::ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE (C++ class), 154
 Enumerations::ERROR_CODE_SVO_RECORDING_ERROR (C++ class), 154
 Enumerations::MAT_TYPE (C++ type), 147
 Enumerations::MAT_TYPE_32F_C1 (C++ class), 147
 Enumerations::MAT_TYPE_32F_C2 (C++ class), 147
 Enumerations::MAT_TYPE_32F_C3 (C++ class), 147
 Enumerations::MAT_TYPE_32F_C4 (C++ class), 147
 Enumerations::MAT_TYPE_8U_C1 (C++ class), 147
 Enumerations::MAT_TYPE_8U_C2 (C++ class), 147
 Enumerations::MAT_TYPE_8U_C3 (C++ class), 147
 Enumerations::MAT_TYPE_8U_C4 (C++ class), 148
 Enumerations::MEASURE (C++ type), 150
 Enumerations::MEASURE_CONFIDENCE (C++ class), 150
 Enumerations::MEASURE_DEPTH (C++ class), 150
 Enumerations::MEASURE_DISPARITY (C++ class), 150
 Enumerations::MEASURE_LAST (C++ class), 151
 Enumerations::MEASURE_XYZ (C++ class), 150
 Enumerations::MEASURE_XYZABGR (C++ class), 151
 Enumerations::MEASURE_XYZARGB (C++ class), 150
 Enumerations::MEASURE_XYZBGRA (C++ class), 150
 Enumerations::MEASURE_XYZRGBA (C++ class), 150
 Enumerations::MEM (C++ type), 147
 Enumerations::MEM_CPU (C++ class), 147
 Enumerations::MEM_GPU (C++ class), 147
 Enumerations::POINT_CLOUD_FORMAT (C++ type), 151
 Enumerations::POINT_CLOUD_FORMAT_LAST (C++ class), 152
 Enumerations::POINT_CLOUD_FORMAT_PCD_ASCII

- (C++ class), 152
- Enumerations::POINT_CLOUD_FORMAT_PLY_ASCII (C++ class), 152
- Enumerations::POINT_CLOUD_FORMAT_VTK_ASCII (C++ class), 152
- Enumerations::POINT_CLOUD_FORMAT_XYZ_ASCII (C++ class), 152
- Enumerations::RANGE (C++ type), 147
- Enumerations::RANGE_FAR (C++ class), 147
- Enumerations::RANGE_MEDIUM (C++ class), 147
- Enumerations::RANGE_NEAR (C++ class), 147
- Enumerations::REFERENCE_FRAME (C++ type), 152
- Enumerations::REFERENCE_FRAME_CAMERA (C++ class), 153
- Enumerations::REFERENCE_FRAME_LAST (C++ class), 153
- Enumerations::REFERENCE_FRAME_WORLD (C++ class), 153
- Enumerations::RESOLUTION (C++ type), 146, 148
- Enumerations::RESOLUTION_HD1080 (C++ class), 148
- Enumerations::RESOLUTION_HD2K (C++ class), 148
- Enumerations::RESOLUTION_HD720 (C++ class), 148
- Enumerations::RESOLUTION_HIGH (C++ class), 146
- Enumerations::RESOLUTION_LAST (C++ class), 148
- Enumerations::RESOLUTION_LOW (C++ class), 146
- Enumerations::RESOLUTION_MEDIUM (C++ class), 146
- Enumerations::RESOLUTION_VGA (C++ class), 148
- Enumerations::SELF_CALIBRATION_STATE (C++ type), 149
- Enumerations::SELF_CALIBRATION_STATE_FAILED (C++ class), 149
- Enumerations::SELF_CALIBRATION_STATE_LAST (C++ class), 149
- Enumerations::SELF_CALIBRATION_STATE_NOT_STARTED (C++ class), 149
- Enumerations::SELF_CALIBRATION_STATE_RUNNING (C++ class), 149
- Enumerations::SELF_CALIBRATION_STATE_SUCCESS (C++ class), 149
- Enumerations::SENSING_MODE (C++ type), 149
- Enumerations::SENSING_MODE_FILL (C++ class), 149
- Enumerations::SENSING_MODE_LAST (C++ class), 149
- Enumerations::SENSING_MODE_STANDARD (C++ class), 149
- Enumerations::SPATIAL_MAPPING_STATE (C++ type), 153
- Enumerations::SPATIAL_MAPPING_STATE_FPS_TOO_LOW (C++ class), 153
- Enumerations::SPATIAL_MAPPING_STATE_INITIALIZING (C++ class), 153
- Enumerations::SPATIAL_MAPPING_STATE_LAST (C++ class), 153
- Enumerations::SPATIAL_MAPPING_STATE_NOT_ENABLED (C++ class), 153
- Enumerations::SPATIAL_MAPPING_STATE_NOT_ENOUGH_MEMORY (C++ class), 153
- Enumerations::SPATIAL_MAPPING_STATE_OK (C++ class), 153
- Enumerations::SUCCESS (C++ class), 153
- Enumerations::SVO_COMPRESSION_MODE (C++ type), 153
- Enumerations::SVO_COMPRESSION_MODE_LAST (C++ class), 153
- Enumerations::SVO_COMPRESSION_MODE_LOSSLESS (C++ class), 153
- Enumerations::SVO_COMPRESSION_MODE_LOSSY (C++ class), 153
- Enumerations::SVO_COMPRESSION_MODE_RAW (C++ class), 153
- Enumerations::TRACKING_STATE (C++ type), 152
- Enumerations::TRACKING_STATE_FPS_TOO_LOW (C++ class), 152
- Enumerations::TRACKING_STATE_LAST (C++ class), 152
- Enumerations::TRACKING_STATE_OFF (C++ class), 152
- Enumerations::TRACKING_STATE_OK (C++ class), 152
- Enumerations::TRACKING_STATE_SEARCHING (C++ class), 152
- Enumerations::UNIT (C++ type), 149
- Enumerations::UNIT_CENTIMETER (C++ class), 150
- Enumerations::UNIT_FOOT (C++ class), 150
- Enumerations::UNIT_INCH (C++ class), 150
- Enumerations::UNIT_LAST (C++ class), 150
- Enumerations::UNIT_METER (C++ class), 150
- Enumerations::UNIT_MILLIMETER (C++ class), 150
- Enumerations::VIEW (C++ type), 151
- Enumerations::VIEW_CONFIDENCE (C++ class), 151
- Enumerations::VIEW_DEPTH (C++ class), 151
- Enumerations::VIEW_LAST (C++ class), 151
- Enumerations::VIEW_LEFT (C++ class), 151
- Enumerations::VIEW_LEFT_GRAY (C++ class), 151
- Enumerations::VIEW_LEFT_UNRECTIFIED (C++ class), 151
- Enumerations::VIEW_LEFT_UNRECTIFIED_GRAY (C++ class), 151
- Enumerations::VIEW_RIGHT (C++ class), 151
- Enumerations::VIEW_RIGHT_GRAY (C++ class), 151
- Enumerations::VIEW_RIGHT_UNRECTIFIED (C++ class), 151
- Enumerations::VIEW_RIGHT_UNRECTIFIED_GRAY (C++ class), 151
- Enumerations::VIEW_SIDE_BY_SIDE (C++ class), 151

error (C++ function), 104, 140
eval_cifar_csv (C++ function), 91
example (built-in class), 67
exclusive_image (C++ function), 116
exitActions (C++ function), 146
EXP (C++ class), 100
exposure_image (C++ function), 124, 126
ext_modules (setup attribute), 78
ext_modules (setup_gpu attribute), 78
extract_cifar (C++ function), 91
extract_voxel (C++ function), 97

F

face_detection (built-in class), 68
face_detection.align_face_mtcnn() (built-in function), 69
face_detection.detect_face_and_landmarks_mtcnn()
(built-in function), 69
face_detection.draw_landmarks() (built-in function), 69
face_detection.draw_rects() (built-in function), 69
face_detection.face_detected() (built-in function), 70
face_detection.get_closest_face() (built-in function), 69
face_detection.get_model_filenames() (built-in function),
70
face_detection.load_model() (built-in function), 70
face_detection.recognize_face() (built-in function), 70
FaceDetect (built-in class), 71
FaceDetect.BOX (built-in class), 14
FaceDetect.c_array() (built-in function), 72
FaceDetect.detect() (built-in function), 72
FaceDetect.detectObjects() (built-in function), 72
FaceDetect.draw_results() (built-in function), 72
FaceDetect.IMAGE (built-in class), 25
FaceDetect.Initialize() (built-in function), 72
FaceDetect.METADATA (built-in class), 50
FaceDetect.sample() (built-in function), 72
FaceDetect.StartDetection() (built-in function), 72
FacePointQueue (RoboyVision attribute), 77
fgetgo (C++ function), 94
fgetl (C++ function), 104, 141, 142
file_error (C++ function), 141, 142
fill_cpu (C++ function), 108, 109
fill_hierarchy (C++ function), 116
fill_image (C++ function), 103, 124
fill_truth (C++ function), 116, 119
fill_truth_captcha (C++ function), 116
fill_truth_detection (C++ function), 116
fill_truth_iseg (C++ function), 116
fill_truth_region (C++ function), 116
fill_truth_swag (C++ function), 116
find_arg (C++ function), 104, 140
find_char_arg (C++ function), 104, 140
find_float_arg (C++ function), 104, 140
find_int_arg (C++ function), 104, 140
find_replace (C++ function), 104, 140, 142

find_replace_paths (C++ function), 116
FindObjectSrv (built-in class), 72
FindObjectSrv.service_callback() (built-in function), 73
fix_data_captcha (C++ function), 90
flatten (C++ function), 108, 109
flip_board (C++ function), 94
flip_image (C++ function), 103, 123
float_pair (C++ class), 25
float_pair::x (C++ member), 25
float_pair::y (C++ member), 25
float_to_box (C++ function), 103, 110
float_to_image (C++ function), 103, 123
forward_activation_layer (C++ function), 105
forward_avgpool_layer (C++ function), 107
forward_batchnorm_layer (C++ function), 107, 108
forward_connected_layer (C++ function), 111, 112
forward_convolutional_layer (C++ function), 112, 113
forward_cost_layer (C++ function), 114
forward_crnn_layer (C++ function), 114, 115
forward_crop_layer (C++ function), 115
forward_deconvolutional_layer (C++ function), 119
forward_detection_layer (C++ function), 120
forward_dropout_layer (C++ function), 120, 121
forward_gru_layer (C++ function), 122
forward_local_layer (C++ function), 127, 128
forward_lstm_layer (C++ function), 128
forward_maxpool_layer (C++ function), 129, 130
forward_network (C++ function), 101, 130
forward_normalization_layer (C++ function), 132
forward_region_layer (C++ function), 135
forward_reorg_layer (C++ function), 135, 136
forward_rnn_layer (C++ function), 136
forward_route_layer (C++ function), 136, 137
forward_shortcut_layer (C++ function), 137
forward_softmax_layer (C++ function), 137, 138
FoundObjects (FindObjectSrv attribute), 73
fourcc (example attribute), 68
frame (example attribute), 68
FrameQueue (multiprocess attribute), 74
FrameQueue (RoboyVision attribute), 77
free_data (C++ function), 101, 117
free_image (C++ function), 104, 125
free_image (FaceDetect attribute), 72
free_image (ObjectRecognition attribute), 75
free_layer (C++ function), 101, 127
free_list (C++ function), 104, 127
free_list_contents (C++ function), 127
free_matrix (C++ function), 102, 129
free_network (C++ function), 102, 131
free_node (C++ function), 127
free_ptrs (C++ function), 104, 141, 142
free_ptrs (FaceDetect attribute), 72
free_ptrs (ObjectRecognition attribute), 75
free_section (C++ function), 133

FreezeModel (built-in class), 73
 FreezeModel.freeze_graph() (built-in function), 73

G

g_handle (C++ member), 144
 gemm (C++ function), 121
 gemm_bin (C++ function), 121
 gemm_cpu (C++ function), 121
 gemm_nn (C++ function), 121
 gemm_nt (C++ function), 121
 gemm_tn (C++ function), 121
 gemm_tt (C++ function), 121
 generate_move (C++ function), 94
 generateImageToRecord (C++ function), 146
 get_activation (C++ function), 105, 106
 get_activation_string (C++ function), 105, 106
 get_avgpool_image (C++ function), 107
 get_base_args (C++ function), 101, 130
 get_coco_image_id (C++ function), 92, 93
 get_color (C++ function), 123, 125
 get_convolutional_delta (C++ function), 112, 113
 get_convolutional_image (C++ function), 112, 113
 get_convolutional_weight (C++ function), 112, 113
 get_cost_string (C++ function), 114
 get_cost_type (C++ function), 114
 get_crop_image (C++ function), 115
 get_current_batch (C++ function), 103, 130
 get_current_rate (C++ function), 103, 130
 get_data_part (C++ function), 118
 get_detection_boxes (C++ function), 102, 120
 get_detection_info (C++ function), 143
 get_hierarchy_probability (C++ function), 140
 get_image_layer (C++ function), 125, 126
 get_label (C++ function), 123
 get_labels (C++ function), 104, 117
 get_layer_string (C++ function), 130, 131
 get_maxpool_delta (C++ function), 129
 get_maxpool_image (C++ function), 129, 130
 get_metadata (C++ function), 101, 132
 get_network_detection_layer (C++ function), 130
 get_network_image (C++ function), 103, 131
 get_network_image_layer (C++ function), 103, 131
 get_network_output_layer (C++ function), 103, 131
 get_next_batch (C++ function), 101, 117
 get_paths (C++ function), 104, 116
 get_pixel (C++ function), 125, 126
 get_pixel_extend (C++ function), 125, 126
 get_policy (C++ function), 134
 get_predicted_class_network (C++ function), 130, 131
 get_random_batch (C++ function), 117, 118
 get_random_data (C++ function), 118
 get_random_paths (C++ function), 116
 get_region_box (C++ function), 135
 get_region_boxes (C++ function), 102, 135

get_regression_values (C++ function), 91
 get_rnn_data (C++ function), 96
 get_rnn_token_data (C++ function), 96
 get_segmentation_image (C++ function), 117
 get_segmentation_image2 (C++ function), 117
 get_weights (C++ function), 102, 113
 get_workspace_size (C++ function), 112, 119
 ghost_image (C++ function), 103, 123
 gpu_index (C++ member), 105, 115
 gradient (C++ function), 105, 106
 gradient_array (C++ function), 105, 106
 graph (face_detection attribute), 71
 grayscale_image (C++ function), 103, 124
 grayscale_image_3c (C++ function), 101, 124
 GRU (C++ class), 99
 gun_classifier (C++ function), 91

H

h (example attribute), 68
 HARDTAN (C++ class), 99
 hardtan_activate (C++ function), 106
 hardtan_gradient (C++ function), 106
 help (FreezeModel attribute), 73
 hierarchy_predictions (C++ function), 104, 140
 hierarchy_top_prediction (C++ function), 140
 hold_out_matrix (C++ function), 129
 hsv_to_rgb (C++ function), 124, 126
 hue_image (C++ function), 124

I

im (darknet attribute), 67
 im (zedRoboy attribute), 90
 im2col_cpu (C++ function), 122
 im2col_get_pixel (C++ function), 122
 image (C++ class), 26
 image::c (C++ member), 26
 image::data (C++ member), 26
 image::h (C++ member), 26
 image::w (C++ member), 26
 image_batch (face_detection attribute), 71
 IMAGE_DATA (C++ class), 100
 image_distance (C++ function), 123, 125
 image_ids (voc_label attribute), 89
 image_paths (RecogniseFace.ImageClass attribute), 26
 img (example attribute), 68
 img (face_detection attribute), 71
 increment_layer (C++ function), 114, 122, 128, 136
 InfoOption (C++ class), 26
 InfoOption::computeDisparity (C++ member), 26
 InfoOption::output_path (C++ member), 26
 InfoOption::recordingMode (C++ member), 26
 InfoOption::svo_path (C++ member), 26
 InfoOption::videoMode (C++ member), 26
 INIT_TIMER (C macro), 145

initActions (C++ function), 146
InitCamera (C++ function), 146
initpyyolo (C++ function), 144
INSTANCE_DATA (C++ class), 101
int_to_alphanum (C++ function), 140, 142
inter_cpu (C++ function), 108, 109
inverted (C++ member), 95
IS_FINITE (C macro), 145
is_network (C++ function), 134
isValidMeasure (C macro), 145

K

keyboard (C++ member), 146
kvp (C++ class), 28
kvp::key (C++ member), 29
kvp::used (C++ member), 29
kvp::val (C++ member), 29

L

L1 (C++ class), 100
l1_cpu (C++ function), 108, 109
l2_cpu (C++ function), 108, 109
label_classifier (C++ function), 91
layer (C++ class), 29
layer (C++ type), 98
layer::absolute (C++ member), 31
layer::activation (C++ member), 29
layer::alpha (C++ member), 31
layer::angle (C++ member), 30
layer::background (C++ member), 30
layer::backward (C++ member), 29
layer::backward_gpu (C++ member), 29
layer::batch (C++ member), 29
layer::batch_normalize (C++ member), 29
layer::beta (C++ member), 31
layer::bias_m (C++ member), 32
layer::bias_match (C++ member), 31
layer::bias_updates (C++ member), 32
layer::bias_v (C++ member), 32
layer::biases (C++ member), 32
layer::binary (C++ member), 30
layer::binary_input (C++ member), 33
layer::binary_weights (C++ member), 32
layer::c (C++ member), 29
layer::c_cpu (C++ member), 33
layer::cell_cpu (C++ member), 33
layer::class_scale (C++ member), 31
layer::classes (C++ member), 30
layer::classfix (C++ member), 31
layer::combine_cpu (C++ member), 31
layer::combine_delta_cpu (C++ member), 31
layer::concat (C++ member), 31
layer::concat_delta (C++ member), 31
layer::coord_scale (C++ member), 31

layer::coords (C++ member), 30
layer::cost (C++ member), 31
layer::cost_type (C++ member), 29
layer::cweights (C++ member), 31
layer::dc_cpu (C++ member), 33
layer::delta (C++ member), 32
layer::dh_cpu (C++ member), 32
layer::does_cost (C++ member), 30
layer::dontload (C++ member), 31
layer::dontloadscales (C++ member), 31
layer::dot (C++ member), 30
layer::exposure (C++ member), 30
layer::extra (C++ member), 29
layer::f_cpu (C++ member), 33
layer::flatten (C++ member), 30
layer::flip (C++ member), 30
layer::flipped (C++ member), 29
layer::forced (C++ member), 29
layer::forgot_delta (C++ member), 31
layer::forgot_state (C++ member), 31
layer::forward (C++ member), 29
layer::forward_gpu (C++ member), 29
layer::g_cpu (C++ member), 33
layer::groups (C++ member), 29
layer::h (C++ member), 29
layer::h_cpu (C++ member), 32
layer::hh_cpu (C++ member), 32
layer::hidden (C++ member), 30
layer::i_cpu (C++ member), 33
layer::index (C++ member), 30
layer::indexes (C++ member), 31
layer::input_gate_layer (C++ member), 33
layer::input_h_layer (C++ member), 33
layer::input_layer (C++ member), 33
layer::input_layers (C++ member), 31
layer::input_r_layer (C++ member), 33
layer::input_save_layer (C++ member), 33
layer::input_sizes (C++ member), 31
layer::input_state_layer (C++ member), 33
layer::input_z_layer (C++ member), 33
layer::inputs (C++ member), 29
layer::jitter (C++ member), 30
layer::joint (C++ member), 30
layer::kappa (C++ member), 31
layer::learning_rate_scale (C++ member), 30
layer::log (C++ member), 30
layer::m (C++ member), 32
layer::map (C++ member), 31
layer::mask_scale (C++ member), 31
layer::max_boxes (C++ member), 29
layer::mean (C++ member), 32
layer::mean_delta (C++ member), 32
layer::n (C++ member), 29
layer::nbases (C++ member), 29

layer::noadjust (C++ member), 30
 layer::noobject_scale (C++ member), 31
 layer::norms (C++ member), 32
 layer::nweights (C++ member), 29
 layer::o_cpu (C++ member), 33
 layer::object_scale (C++ member), 31
 layer::objectness (C++ member), 30
 layer::onlyforward (C++ member), 31
 layer::out_c (C++ member), 29
 layer::out_h (C++ member), 29
 layer::out_w (C++ member), 29
 layer::output (C++ member), 32
 layer::output_layer (C++ member), 33
 layer::outputs (C++ member), 29
 layer::pad (C++ member), 30
 layer::prev_cell_cpu (C++ member), 32
 layer::prev_state (C++ member), 31
 layer::prev_state_cpu (C++ member), 32
 layer::probability (C++ member), 31
 layer::r_cpu (C++ member), 32
 layer::rand (C++ member), 31
 layer::random (C++ member), 31
 layer::ratio (C++ member), 30
 layer::reorg (C++ member), 30
 layer::rescore (C++ member), 30
 layer::reset_layer (C++ member), 33
 layer::reverse (C++ member), 30
 layer::rolling_mean (C++ member), 32
 layer::rolling_variance (C++ member), 32
 layer::saturation (C++ member), 30
 layer::scale (C++ member), 31
 layer::scale_m (C++ member), 32
 layer::scale_updates (C++ member), 32
 layer::scale_v (C++ member), 32
 layer::scales (C++ member), 32
 layer::self_layer (C++ member), 33
 layer::shift (C++ member), 30
 layer::shortcut (C++ member), 29
 layer::side (C++ member), 30
 layer::size (C++ member), 30
 layer::smooth (C++ member), 30
 layer::softmax (C++ member), 30
 layer::softmax_tree (C++ member), 34
 layer::spatial (C++ member), 30
 layer::spatial_mean (C++ member), 32
 layer::sqrt (C++ member), 30
 layer::squared (C++ member), 32
 layer::state (C++ member), 31
 layer::state_delta (C++ member), 31
 layer::state_gate_layer (C++ member), 33
 layer::state_h_layer (C++ member), 33
 layer::state_layer (C++ member), 33
 layer::state_r_layer (C++ member), 33
 layer::state_save_layer (C++ member), 33
 layer::state_state_layer (C++ member), 33
 layer::state_z_layer (C++ member), 33
 layer::steps (C++ member), 30
 layer::stopbackward (C++ member), 31
 layer::stride (C++ member), 30
 layer::tanh (C++ member), 30
 layer::temp2_cpu (C++ member), 32
 layer::temp3_cpu (C++ member), 32
 layer::temp_cpu (C++ member), 32
 layer::temperature (C++ member), 31
 layer::thresh (C++ member), 31
 layer::truth (C++ member), 30
 layer::truths (C++ member), 29
 layer::type (C++ member), 29
 layer::uf (C++ member), 33
 layer::ug (C++ member), 34
 layer::uh (C++ member), 33
 layer::ui (C++ member), 34
 layer::uo (C++ member), 33
 layer::update (C++ member), 29
 layer::update_gpu (C++ member), 29
 layer::update_layer (C++ member), 33
 layer::ur (C++ member), 33
 layer::uz (C++ member), 33
 layer::v (C++ member), 32
 layer::variance (C++ member), 32
 layer::variance_delta (C++ member), 32
 layer::w (C++ member), 29
 layer::weight_updates (C++ member), 32
 layer::weights (C++ member), 32
 layer::wf (C++ member), 33
 layer::wg (C++ member), 34
 layer::wh (C++ member), 33
 layer::wi (C++ member), 34
 layer::wo (C++ member), 33
 layer::workspace_size (C++ member), 34
 layer::wr (C++ member), 33
 layer::wz (C++ member), 33
 layer::x (C++ member), 32
 layer::x_norm (C++ member), 32
 layer::xnor (C++ member), 30
 layer::z_cpu (C++ member), 32
 LAYER_TYPE (C++ type), 99
 LEAKY (C++ class), 99
 leaky_activate (C++ function), 106
 leaky_gradient (C++ function), 106
 learning_rate_policy (C++ type), 100
 legal_go (C++ function), 94
 LETTERBOX_DATA (C++ class), 101
 letterbox_image (C++ function), 102, 124
 letterbox_image (FaceDetect attribute), 72
 letterbox_image (ObjectRecognition attribute), 75
 letterbox_image_into (C++ function), 124, 126
 level (FindObjectSrv attribute), 73

LHTAN (C++ class), 99
lhtan_activate (C++ function), 106
lhtan_gradient (C++ function), 106
lib (darknet attribute), 67
lib (FaceDetect attribute), 72
lib (ObjectRecognition attribute), 75
lib (zedRoboy attribute), 90
LINEAR (C++ class), 99
linear_activate (C++ function), 106
linear_gradient (C++ function), 106
list (C++ class), 34
list (C++ type), 98
list::back (C++ member), 34
list::front (C++ member), 34
list::size (C++ member), 34
list_file (voc_label attribute), 89
list_find (C++ function), 127
list_insert (C++ function), 127
list_pop (C++ function), 127
list_to_array (C++ function), 104, 127
load_all_cifar10 (C++ function), 103, 117
load_alphabet (C++ function), 103, 123
load_args (C++ class), 34
load_args (C++ type), 98
load_args::angle (C++ member), 35
load_args::aspect (C++ member), 35
load_args::background (C++ member), 34
load_args::center (C++ member), 34
load_args::classes (C++ member), 34
load_args::coords (C++ member), 35
load_args::d (C++ member), 35
load_args::exposure (C++ member), 35
load_args::h (C++ member), 34
load_args::hierarchy (C++ member), 35
load_args::hue (C++ member), 35
load_args::im (C++ member), 35
load_args::jitter (C++ member), 35
load_args::labels (C++ member), 34
load_args::m (C++ member), 34
load_args::max (C++ member), 34
load_args::min (C++ member), 34
load_args::n (C++ member), 34
load_args::nh (C++ member), 34
load_args::num_boxes (C++ member), 34
load_args::nw (C++ member), 34
load_args::out_h (C++ member), 34
load_args::out_w (C++ member), 34
load_args::path (C++ member), 34
load_args::paths (C++ member), 34
load_args::resized (C++ member), 35
load_args::saturation (C++ member), 35
load_args::scale (C++ member), 34
load_args::size (C++ member), 34
load_args::threads (C++ member), 34
load_args::type (C++ member), 35
load_args::w (C++ member), 34
load_batchnorm_weights (C++ function), 134
load_categorical_data_csv (C++ function), 117, 118
load_cifar10_data (C++ function), 101, 117
load_connected_weights (C++ function), 134
load_convolutional_weights (C++ function), 134
load_convolutional_weights_binary (C++ function), 134
load_data (C++ function), 101, 117
load_data_augment (C++ function), 117, 118
load_data_blocking (C++ function), 104, 117, 118
load_data_captcha (C++ function), 116, 118
load_data_captcha_encode (C++ function), 116, 118
load_data_compare (C++ function), 117
load_data_detection (C++ function), 117, 118
load_data_in_thread (C++ function), 104, 117
load_data_iseg (C++ function), 117
load_data_old (C++ function), 103, 117
load_data_region (C++ function), 117
load_data_regression (C++ function), 117, 118
load_data_seg (C++ function), 117
load_data_super (C++ function), 117, 118
load_data_swag (C++ function), 117
load_data_tag (C++ function), 117, 118
load_data_writing (C++ function), 117, 118
load_go (C++ function), 117, 118
load_go_moves (C++ function), 94
load_image (C++ function), 102, 125
load_image (FaceDetect attribute), 72
load_image (ObjectRecognition attribute), 75
load_image_augment_paths (C++ function), 116, 118
load_image_color (C++ function), 102, 125
load_image_paths (C++ function), 116
load_image_paths_gray (C++ function), 116
load_image_stb (C++ function), 125
load_labels_paths (C++ function), 116
load_meta (FaceDetect attribute), 72
load_meta (ObjectRecognition attribute), 75
load_net (FaceDetect attribute), 72
load_net (ObjectRecognition attribute), 75
load_network (C++ function), 101, 130
load_network_p (C++ function), 101, 130
load_regression_labels_paths (C++ function), 116
load_rle (C++ function), 116
load_tags_paths (C++ function), 116
load_thread (C++ function), 117
load_threads (C++ function), 117
load_weights (C++ function), 102, 134
load_weights_upto (C++ function), 102, 134
LOCAL (C++ class), 99
local_layer (C++ type), 128
local_out_height (C++ function), 127
local_out_width (C++ function), 127
LOGGY (C++ class), 99

[loggy_activate \(C++ function\), 106](#)
[loggy_gradient \(C++ function\), 106](#)
[LOGISTIC \(C++ class\), 99](#)
[logistic_activate \(C++ function\), 106](#)
[logistic_gradient \(C++ function\), 106](#)
[logit \(C++ function\), 135](#)
[LSTM \(C++ class\), 99](#)

M

[mag_array \(C++ function\), 104, 141](#)
[main \(C++ function\), 93, 146](#)
[make_activation_layer \(C++ function\), 105](#)
[make_avgpool_layer \(C++ function\), 107](#)
[make_batchnorm_layer \(C++ function\), 107, 108](#)
[make_boxes \(FaceDetect attribute\), 72](#)
[make_boxes \(ObjectRecognition attribute\), 75](#)
[make_connected_layer \(C++ function\), 111, 112](#)
[make_convolutional_layer \(C++ function\), 112, 113](#)
[make_cost_layer \(C++ function\), 114](#)
[make_crnn_layer \(C++ function\), 114, 115](#)
[make_crop_layer \(C++ function\), 115](#)
[make_deconvolutional_layer \(C++ function\), 119](#)
[make_detection_layer \(C++ function\), 120](#)
[make_dropout_layer \(C++ function\), 120, 121](#)
[make_empty_image \(C++ function\), 123, 126](#)
[make_gru_layer \(C++ function\), 122](#)
[make_image \(C++ function\), 102, 123](#)
[make_labels \(built-in class\), 73](#)
[make_labels.make_labels\(\) \(built-in function\), 73](#)
[make_list \(C++ function\), 127](#)
[make_local_layer \(C++ function\), 127, 128](#)
[make_lstm_layer \(C++ function\), 128](#)
[make_matrix \(C++ function\), 104, 129](#)
[make_maxpool_layer \(C++ function\), 129, 130](#)
[make_network \(C++ function\), 130, 131](#)
[make_normalization_layer \(C++ function\), 132](#)
[make_probs \(FaceDetect attribute\), 72](#)
[make_probs \(ObjectRecognition attribute\), 75](#)
[make_random_image \(C++ function\), 102, 123](#)
[make_region_layer \(C++ function\), 135](#)
[make_reorg_layer \(C++ function\), 135, 136](#)
[make_rnn_layer \(C++ function\), 136](#)
[make_route_layer \(C++ function\), 136, 137](#)
[make_shortcut_layer \(C++ function\), 137](#)
[make_softmax_layer \(C++ function\), 137, 138](#)
[makes_safe_go \(C++ function\), 94](#)
[malloc_error \(C++ function\), 140, 142](#)
[manageActions \(C++ function\), 146](#)
[mask_to_rgb \(C++ function\), 102, 123](#)
[MASKED \(C++ class\), 100](#)
[matrix \(C++ class\), 42](#)
[matrix \(C++ type\), 98](#)
[matrix::cols \(C++ member\), 42](#)
[matrix::rows \(C++ member\), 42](#)

[matrix::vals \(C++ member\), 42](#)
[matrix_add_matrix \(C++ function\), 101, 129](#)
[matrix_to_csv \(C++ function\), 101, 129](#)
[matrix_topk_accuracy \(C++ function\), 101, 129](#)
[max_index \(C++ function\), 104, 141](#)
[MAXPOOL \(C++ class\), 99](#)
[maxpool_layer \(C++ type\), 130](#)
[mean_array \(C++ function\), 104, 141](#)
[mean_arrays \(C++ function\), 141, 143](#)
[mean_cpu \(C++ function\), 108, 109](#)
[mean_delta_cpu \(C++ function\), 107, 109](#)
[meta \(darknet attribute\), 67](#)
[meta \(zedRoboy attribute\), 90](#)
[meta_file \(face_detection attribute\), 71](#)
[metadata \(C++ class\), 50](#)
[metadata::classes \(C++ member\), 50](#)
[metadata::names \(C++ member\), 50](#)
[mking \(C++ function\), 93](#)
[module \(setup attribute\), 78](#)
[module \(setup_gpu attribute\), 78](#)
[move_go \(C++ function\), 94](#)
[moves \(C++ class\), 51](#)
[moves::data \(C++ member\), 51](#)
[moves::n \(C++ member\), 51](#)
[mse_array \(C++ function\), 104, 141](#)
[mul_cpu \(C++ function\), 108, 109](#)
[mult_add_into_cpu \(C++ function\), 108, 109](#)
[multiprocess \(built-in class\), 73](#)
[multiprocess.detectFaces\(\) \(built-in function\), 74](#)
[multiprocess.TestProcess\(\) \(built-in function\), 74](#)
[multiprocess.tracking\(\) \(built-in function\), 74](#)
[Multitracking \(built-in class\), 74](#)
[Multitracking.StartTracking\(\) \(built-in function\), 74](#)
[mutex \(C++ member\), 118](#)

N

[name \(RecogniseFace.ImageClass attribute\), 26](#)
[name \(setup attribute\), 78](#)
[name \(setup_gpu attribute\), 78](#)
[net \(darknet attribute\), 67](#)
[net \(zedRoboy attribute\), 90](#)
[NETWORK \(C++ class\), 100](#)
[network \(C++ class\), 51](#)
[network \(C++ type\), 98](#)
[network::adam \(C++ member\), 52](#)
[network::angle \(C++ member\), 52](#)
[network::aspect \(C++ member\), 52](#)
[network::B1 \(C++ member\), 52](#)
[network::B2 \(C++ member\), 52](#)
[network::batch \(C++ member\), 51](#)
[network::burn_in \(C++ member\), 52](#)
[network::c \(C++ member\), 52](#)
[network::center \(C++ member\), 52](#)
[network::cost \(C++ member\), 52](#)

- network::decay (C++ member), 51
- network::delta (C++ member), 52
- network::epoch (C++ member), 51
- network::eps (C++ member), 52
- network::exposure (C++ member), 52
- network::gamma (C++ member), 51
- network::gpu_index (C++ member), 52
- network::h (C++ member), 52
- network::hierarchy (C++ member), 52
- network::hue (C++ member), 52
- network::index (C++ member), 52
- network::input (C++ member), 52
- network::inputs (C++ member), 52
- network::layers (C++ member), 51
- network::learning_rate (C++ member), 51
- network::max_batches (C++ member), 51
- network::max_crop (C++ member), 52
- network::min_crop (C++ member), 52
- network::momentum (C++ member), 51
- network::n (C++ member), 51
- network::notruth (C++ member), 52
- network::num_steps (C++ member), 51
- network::output (C++ member), 51
- network::outputs (C++ member), 52
- network::policy (C++ member), 51
- network::power (C++ member), 51
- network::saturation (C++ member), 52
- network::scale (C++ member), 51
- network::scales (C++ member), 51
- network::seen (C++ member), 51
- network::step (C++ member), 51
- network::steps (C++ member), 51
- network::subdivisions (C++ member), 51
- network::t (C++ member), 51
- network::time_steps (C++ member), 51
- network::train (C++ member), 52
- network::truth (C++ member), 52
- network::truths (C++ member), 52
- network::w (C++ member), 52
- network::workspace (C++ member), 52
- network_accuracies (C++ function), 102, 131
- network_accuracy (C++ function), 103, 131
- network_accuracy_multi (C++ function), 131
- network_detect (FaceDetect attribute), 72
- network_detect (ObjectRecognition attribute), 75
- network_height (C++ function), 104, 131
- network_inputs (C++ function), 131
- network_output (C++ function), 131
- network_output_layer (C++ function), 131
- network_outputs (C++ function), 131
- network_predict (C++ function), 103, 131
- network_predict_data (C++ function), 103, 131
- network_predict_data_multi (C++ function), 131
- network_predict_image (C++ function), 104, 131

- network_predict_p (C++ function), 103, 131
- network_width (C++ function), 104, 131
- NewFacialFeaturesMsg (built-in class), 74
- NewFacialFeaturesMsg.service_callback() (built-in function), 74
- nind (C++ member), 95
- nms_comparator (C++ function), 110
- node (C++ class), 52
- node (C++ type), 98
- node::next (C++ member), 52
- node::prev (C++ member), 52
- node::val (C++ member), 52
- noi (C++ member), 95
- NORMALIZATION (C++ class), 99
- normalize_array (C++ function), 104, 141
- normalize_cpu (C++ function), 101, 108
- normalize_data_rows (C++ function), 118
- normalize_delta_cpu (C++ function), 107, 109
- normalize_image (C++ function), 101, 123
- normalize_image2 (C++ function), 123
- normalize_layer (C++ function), 93
- normalize_net (C++ function), 93
- NPY_NO_DEPRECATED_API (C macro), 144
- num_boxes (FaceDetect attribute), 72
- num_boxes (ObjectRecognition attribute), 75
- NUMCHARS (C macro), 116

O

- ObjectRecognition (built-in class), 74
- ObjectRecognition.BOX (built-in class), 14
- ObjectRecognition.c_array() (built-in function), 74
- ObjectRecognition.detect() (built-in function), 74
- ObjectRecognition.detectObjects() (built-in function), 75
- ObjectRecognition.draw_results() (built-in function), 74
- ObjectRecognition.IMAGE (built-in class), 25
- ObjectRecognition.Initialize() (built-in function), 74
- ObjectRecognition.METADATA (built-in class), 50
- ObjectRecognition.sample() (built-in function), 74
- ObjectsQueue (RoboyVision attribute), 77
- OCCLUSION_VALUE (C macro), 145
- ok (example attribute), 68
- OLD_CLASSIFICATION_DATA (C++ class), 100
- one_hot_encode (C++ function), 142, 143
- oneoff (C++ function), 93
- oneoff2 (C++ function), 93
- onet (face_detection attribute), 71
- operations (C++ function), 93
- optimize_picture (C++ function), 95
- option_find (C++ function), 132, 133
- option_find_float (C++ function), 132, 133
- option_find_float_quiet (C++ function), 132, 133
- option_find_int (C++ function), 102, 132
- option_find_int_quiet (C++ function), 132, 133
- option_find_str (C++ function), 102, 132

option_insert (C++ function), 132, 133
 option_unused (C++ function), 132, 133
 or_image (C++ function), 116
 outputs (example attribute), 68
 outVideo (example attribute), 68
 overlap (C++ function), 110

P

parse_activation (C++ function), 134
 parse_args (C++ function), 146
 parse_avgpool (C++ function), 134
 parse_batchnorm (C++ function), 134
 parse_connected (C++ function), 133
 parse_convolutional (C++ function), 133
 parse_cost (C++ function), 133
 parse_crnn (C++ function), 133
 parse_crop (C++ function), 134
 parse_csv_line (C++ function), 141, 142
 parse_data (C++ function), 133
 parse_deconvolutional (C++ function), 133
 parse_detection (C++ function), 133
 parse_dropout (C++ function), 134
 parse_fields (C++ function), 141, 142
 parse_gru (C++ function), 133
 parse_local (C++ function), 133
 parse_lstm (C++ function), 133
 parse_maxpool (C++ function), 134
 parse_net_options (C++ function), 134
 parse_network_cfg (C++ function), 102, 134
 parse_normalization (C++ function), 134
 parse_region (C++ function), 133
 parse_reorg (C++ function), 134
 parse_rnn (C++ function), 133
 parse_route (C++ function), 134
 parse_shortcut (C++ function), 134
 parse_softmax (C++ function), 133
 parser (FreezeModel attribute), 73
 partial (C++ function), 93
 phase_train_placeholder (face_detection attribute), 71
 place_image (C++ function), 123, 126
 PLSE (C++ class), 99
 plse_activate (C++ function), 106
 plse_gradient (C++ function), 106
 pm (C++ function), 109, 140
 pnet (face_detection attribute), 71
 points (face_detection attribute), 71
 POLY (C++ class), 100
 pop_column (C++ function), 129
 pow_cpu (C++ function), 108, 109
 predict (FaceDetect attribute), 72
 predict (ObjectRecognition attribute), 75
 predict_classifier (C++ function), 91, 92
 predict_image (FaceDetect attribute), 72
 predict_image (ObjectRecognition attribute), 75

predict_move (C++ function), 94
 predict_regressor (C++ function), 96
 predict_segmenter (C++ function), 97
 print_board (C++ function), 94
 print_cocos (C++ function), 92, 93
 print_detector_detections (C++ function), 93
 print_game (C++ function), 94
 print_image (C++ function), 125, 126
 print_imagenet_detections (C++ function), 93
 print_letters (C++ function), 116, 118
 print_matrix (C++ function), 129
 print_network (C++ function), 131
 print_statistics (C++ function), 141, 143
 print_symbol (C++ function), 96
 print_yolo_detections (C++ function), 98
 propagate_liberty (C++ function), 94
 pyyolo_cleanup (C++ function), 144
 pyyolo_detect (C++ function), 144
 pyyolo_init (C++ function), 144
 pyyolo_methods (C++ member), 144
 pyyolo_test (C++ function), 144
 PyyoloError (C++ member), 144

R

r (darknet attribute), 67
 r (zedRoboy attribute), 90
 RAMP (C++ class), 99
 ramp_activate (C++ function), 106
 ramp_gradient (C++ function), 106
 rand_int (C++ function), 141, 143
 rand_normal (C++ function), 105, 141
 rand_scale (C++ function), 141, 143
 rand_size_t (C++ function), 105, 141
 rand_uniform (C++ function), 141, 142
 RANDOM (C++ class), 100
 random_augment_args (C++ function), 124, 126
 random_augment_image (C++ function), 124, 126
 random_crop_image (C++ function), 124, 125
 random_distort_image (C++ function), 103, 124
 random_go_moves (C++ function), 94
 random_matrix (C++ function), 109, 121
 randomize_boxes (C++ function), 116
 randomize_data (C++ function), 118, 119
 read_all (C++ function), 141, 142
 read_all_fail (C++ function), 141, 142
 read_boxes (C++ function), 103, 116
 read_cfg (C++ function), 101, 133
 read_data_cfg (C++ function), 101, 132
 read_int (C++ function), 141, 142
 read_intlist (C++ function), 104, 140
 read_map (C++ function), 104, 140
 read_option (C++ function), 132, 133
 read_tokenized_data (C++ function), 96
 read_tokens (C++ function), 96

`read_tree` (C++ function), 140
`RecogniseFace` (built-in class), 75
`RecogniseFace.align_face_dlib()` (built-in function), 75
`RecogniseFace.crop()` (built-in function), 76
`RecogniseFace.flip()` (built-in function), 76
`RecogniseFace.get_dataset()` (built-in function), 76
`RecogniseFace.get_image_paths()` (built-in function), 76
`RecogniseFace.get_image_paths_and_labels()` (built-in function), 76
`RecogniseFace.get_model_filenames()` (built-in function), 76
`RecogniseFace.ImageClass` (built-in class), 26
`RecogniseFace.ImageClass.__init__()` (built-in function), 26
`RecogniseFace.ImageClass.__len__()` (built-in function), 26
`RecogniseFace.ImageClass.__str__()` (built-in function), 26
`RecogniseFace.load_data()` (built-in function), 76
`RecogniseFace.load_model()` (built-in function), 75
`RecogniseFace.prewhiten()` (built-in function), 76
`RecogniseFace.processImage()` (built-in function), 76
`RecogniseFace.recogniseFace()` (built-in function), 75
`RecogniseFace.to_rgb()` (built-in function), 76
`RecogniseFace.train()` (built-in function), 76
`reconstruct_picture` (C++ function), 95
`recordImages` (C++ function), 146
`recordVideo` (C++ function), 146
`RectQueue` (multiprocess attribute), 74
`RectQueue` (RoboyVision attribute), 77
`REGION` (C++ class), 100
`REGION_DATA` (C++ class), 100
`REGRESSION_DATA` (C++ class), 101
`RELIE` (C++ class), 99
`relie_activate` (C++ function), 106
`relie_gradient` (C++ function), 106
`RELU` (C++ class), 99
`relu_activate` (C++ function), 106
`relu_gradient` (C++ function), 106
`remove_connected` (C++ function), 94
`REORG` (C++ class), 100
`reorg_cpu` (C++ function), 108, 109
`request` (FindObjectSrv attribute), 73
`rescale_net` (C++ function), 93
`rescale_weights` (C++ function), 102, 113
`reset_momentum` (C++ function), 130
`reset_normalize_net` (C++ function), 93
`reset_rnn` (FaceDetect attribute), 72
`reset_rnn` (ObjectRecognition attribute), 75
`reset_rnn_state` (C++ function), 96
`resize_avgpool_layer` (C++ function), 107
`resize_batchnorm_layer` (C++ function), 107
`resize_convolutional_layer` (C++ function), 112, 113
`resize_cost_layer` (C++ function), 114
`resize_crop_layer` (C++ function), 115
`resize_deconvolutional_layer` (C++ function), 119
`resize_dropout_layer` (C++ function), 120, 121
`resize_image` (C++ function), 102, 125
`resize_matrix` (C++ function), 129
`resize_max` (C++ function), 124, 126
`resize_maxpool_layer` (C++ function), 129, 130
`resize_min` (C++ function), 102, 124
`resize_network` (C++ function), 102, 130, 131
`resize_normalization_layer` (C++ function), 132
`resize_region_layer` (C++ function), 135
`resize_reorg_layer` (C++ function), 135, 136
`resize_route_layer` (C++ function), 136, 137
`restype` (darknet attribute), 67
`restype` (FaceDetect attribute), 72
`restype` (ObjectRecognition attribute), 75
`ret_val` (example attribute), 68
`rgb_to_hsv` (C++ function), 124, 126
`rgb_to_yuv` (C++ function), 124, 126
`rgbgr_image` (C++ function), 101, 123
`rgbgr_net` (C++ function), 93
`rgbgr_weights` (C++ function), 102, 113
`rnet` (face_detection attribute), 71
`RNN` (C++ class), 99
`RoboyVision` (built-in class), 76
`RoboyVision.detectFaces()` (built-in function), 76
`RoboyVision.ObjectRecognise()` (built-in function), 77
`RoboyVision.recogniseFace()` (built-in function), 77
`RoboyVision.speakerDetect()` (built-in function), 76
`RoboyVision.tracking()` (built-in function), 76
`RoboyVision.visualizer()` (built-in function), 77
`RosMsgUtil` (built-in class), 77
`RosMsgUtil.AdvertiseContinuously()` (built-in function), 77
`RosMsgUtil.AdvertiseDescribeScene()` (built-in function), 77
`RosMsgUtil.AdvertiseFaceCoordinates()` (built-in function), 77
`RosMsgUtil.AdvertiseFindObject()` (built-in function), 77
`RosMsgUtil.AdvertiseLookAtSpeaker()` (built-in function), 77
`RosMsgUtil.AdvertiseNewFacialFeatures()` (built-in function), 77
`RosMsgUtil.DescribeScene()` (built-in function), 77
`RosMsgUtil.FindObject()` (built-in function), 77
`RosMsgUtil.LookAtSpeaker()` (built-in function), 77
`RosMsgUtil.ReceiveServiceRequests()` (built-in function), 78
`RosMsgUtil.SendFaceCoordinates()` (built-in function), 77
`RosMsgUtil.SendNewFacialFeatures()` (built-in function), 77
`rotate_crop_image` (C++ function), 124, 125
`rotate_image` (C++ function), 103, 124

[rotate_image_cw \(C++ function\), 103, 123](#)
[ROUTE \(C++ class\), 99](#)
[route_layer \(C++ type\), 137](#)
[run_art \(C++ function\), 90, 93](#)
[run_captcha \(C++ function\), 90, 92](#)
[run_char_rnn \(C++ function\), 92, 96](#)
[run_cifar \(C++ function\), 91, 92](#)
[run_classifier \(C++ function\), 91, 92](#)
[run_coco \(C++ function\), 92](#)
[run_compare \(C++ function\), 92, 111](#)
[run_detector \(C++ function\), 92, 93](#)
[run_dice \(C++ function\), 92, 94](#)
[run_go \(C++ function\), 92, 95](#)
[run_lsd \(C++ function\), 93, 95](#)
[run_nightmare \(C++ function\), 92, 95](#)
[run_regressor \(C++ function\), 92, 96](#)
[run_segmenter \(C++ function\), 92, 97](#)
[run_super \(C++ function\), 93, 97](#)
[run_swag \(C++ function\), 97](#)
[run_tag \(C++ function\), 92, 97](#)
[run_vid_rnn \(C++ function\), 92, 96](#)
[run_voxel \(C++ function\), 92, 97](#)
[run_writing \(C++ function\), 92, 98](#)
[run_yolo \(C++ function\), 92, 98](#)

S

[sample_array \(C++ function\), 104, 141](#)
[saturate_exposure_image \(C++ function\), 125, 126](#)
[saturate_image \(C++ function\), 124, 126](#)
[save_batchnorm_weights \(C++ function\), 134](#)
[save_connected_weights \(C++ function\), 134](#)
[save_convolutional_weights \(C++ function\), 134](#)
[save_convolutional_weights_binary \(C++ function\), 134](#)
[save_image \(C++ function\), 103, 123](#)
[save_image_png \(C++ function\), 101, 123](#)
[save_network \(C++ function\), 134](#)
[save_weights \(C++ function\), 102, 134](#)
[save_weights_double \(C++ function\), 134](#)
[save_weights_upto \(C++ function\), 102, 134](#)
[scal_cpu \(C++ function\), 101, 108](#)
[scale_array \(C++ function\), 141, 142](#)
[scale_bias \(C++ function\), 109, 112](#)
[scale_data_rows \(C++ function\), 118, 119](#)
[scale_image \(C++ function\), 124, 125](#)
[scale_image_channel \(C++ function\), 124](#)
[scale_matrix \(C++ function\), 101, 129](#)
[score_game \(C++ function\), 95](#)
[sec \(C++ function\), 104, 140, 143](#)
[SECRET_NUM \(C macro\), 98](#)
[section \(C++ class\), 58](#)
[section::options \(C++ member\), 58](#)
[section::type \(C++ member\), 58](#)
[SEG \(C++ class\), 100](#)
[SEGMENTATION_DATA \(C++ class\), 101](#)
[self_go \(C++ function\), 95](#)
[sess \(face_detection attribute\), 71](#)
[session \(face_detection attribute\), 71](#)
[set_batch_network \(C++ function\), 102, 130](#)
[set_pixel \(C++ function\), 125, 126](#)
[setup \(built-in class\), 78](#)
[setup_gpu \(built-in class\), 78](#)
[SHORTCUT \(C++ class\), 99](#)
[shortcut_cpu \(C++ function\), 108, 109](#)
[show_image \(C++ function\), 103, 123](#)
[show_image_collapsed \(C++ function\), 123, 126](#)
[show_image_layers \(C++ function\), 123, 126](#)
[show_image_normalized \(C++ function\), 125, 126](#)
[show_images \(C++ function\), 125, 126](#)
[shuffle \(C++ function\), 140, 142](#)
[SIG \(C++ class\), 100](#)
[size_params \(C++ class\), 58](#)
[size_params \(C++ type\), 133](#)
[size_params::batch \(C++ member\), 59](#)
[size_params::c \(C++ member\), 59](#)
[size_params::h \(C++ member\), 59](#)
[size_params::index \(C++ member\), 59](#)
[size_params::inputs \(C++ member\), 59](#)
[size_params::net \(C++ member\), 59](#)
[size_params::time_steps \(C++ member\), 59](#)
[size_params::w \(C++ member\), 59](#)
[sl \(C++ type\), 78](#)
[sl::__cudaSafeCall \(C++ function\), 88](#)
[sl::AREA_EXPORT_STATE \(C++ type\), 84](#)
[sl::AREA_EXPORT_STATE_FILE_EMPTY \(C++ class\), 85](#)
[sl::AREA_EXPORT_STATE_FILE_ERROR \(C++ class\), 85](#)
[sl::AREA_EXPORT_STATE_LAST \(C++ class\), 85](#)
[sl::AREA_EXPORT_STATE_NOT_STARTED \(C++ class\), 85](#)
[sl::AREA_EXPORT_STATE_RUNNING \(C++ class\), 84](#)
[sl::AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED \(C++ class\), 85](#)
[sl::AREA_EXPORT_STATE_SUCCESS \(C++ class\), 84](#)
[sl::CalibrationParameters \(C++ class\), 15](#)
[sl::CalibrationParameters::left_cam \(C++ member\), 15](#)
[sl::CalibrationParameters::R \(C++ member\), 15](#)
[sl::CalibrationParameters::right_cam \(C++ member\), 15](#)
[sl::CalibrationParameters::T \(C++ member\), 15](#)
[sl::Camera \(C++ class\), 15](#)
[sl::Camera::~~Camera \(C++ function\), 20](#)
[sl::Camera::Camera \(C++ function\), 20](#)
[sl::Camera::close \(C++ function\), 20](#)
[sl::Camera::disableRecording \(C++ function\), 20](#)
[sl::Camera::disableSpatialMapping \(C++ function\), 20](#)
[sl::Camera::disableTracking \(C++ function\), 18](#)
[sl::Camera::enableRecording \(C++ function\), 20](#)

- sl::Camera::enableSpatialMapping (C++ function), 18
- sl::Camera::enableTracking (C++ function), 17
- sl::Camera::extractWholeMesh (C++ function), 19
- sl::Camera::getAreaExportState (C++ function), 18
- sl::Camera::getCameraFPS (C++ function), 16
- sl::Camera::getCameraInformation (C++ function), 17
- sl::Camera::getCameraSettings (C++ function), 16
- sl::Camera::getCameraTimestamp (C++ function), 17
- sl::Camera::getConfidenceThreshold (C++ function), 22
- sl::Camera::getCUDAContext (C++ function), 22
- sl::Camera::getCurrentFPS (C++ function), 16
- sl::Camera::getCurrentTimestamp (C++ function), 17
- sl::Camera::getDepthMaxRangeValue (C++ function), 22
- sl::Camera::getDepthMinRangeValue (C++ function), 22
- sl::Camera::getFrameDroppedCount (C++ function), 17
- sl::Camera::getMeshRequestStatusAsync (C++ function), 19
- sl::Camera::getPosition (C++ function), 18
- sl::Camera::getResolution (C++ function), 22
- sl::Camera::getSDKVersion (C++ function), 22
- sl::Camera::getSelfCalibrationState (C++ function), 17
- sl::Camera::getSpatialMappingState (C++ function), 19
- sl::Camera::getSVONumberOfFrames (C++ function), 16
- sl::Camera::getSVOPosition (C++ function), 16
- sl::Camera::grab (C++ function), 21
- sl::Camera::h (C++ member), 23
- sl::Camera::initMemory (C++ function), 23
- sl::Camera::initRectifier (C++ function), 23
- sl::Camera::isOpened (C++ function), 21
- sl::Camera::isZEDconnected (C++ function), 22
- sl::Camera::nextImage (C++ function), 23
- sl::Camera::open (C++ function), 20
- sl::Camera::openCamera (C++ function), 23
- sl::Camera::opened (C++ member), 23
- sl::Camera::pauseSpatialMapping (C++ function), 19
- sl::Camera::record (C++ function), 20
- sl::Camera::requestMeshAsync (C++ function), 19
- sl::Camera::resetSelfCalibration (C++ function), 17
- sl::Camera::resetTracking (C++ function), 18
- sl::Camera::retrieveImage (C++ function), 21
- sl::Camera::retrieveMeasure (C++ function), 21
- sl::Camera::retrieveMeshAsync (C++ function), 19
- sl::Camera::setCameraFPS (C++ function), 16
- sl::Camera::setCameraSettings (C++ function), 16
- sl::Camera::setConfidenceThreshold (C++ function), 21
- sl::Camera::setDepthMaxRangeValue (C++ function), 22
- sl::Camera::setSVOPosition (C++ function), 15
- sl::Camera::sticktoCPUCore (C++ function), 23
- sl::CAMERA_SETTINGS (C++ type), 80
- sl::CAMERA_SETTINGS_AUTO_WHITEBALANCE (C++ class), 81
- sl::CAMERA_SETTINGS_BRIGHTNESS (C++ class), 80
- sl::CAMERA_SETTINGS_CONTRAST (C++ class), 80
- sl::CAMERA_SETTINGS_EXPOSURE (C++ class), 81
- sl::CAMERA_SETTINGS_GAIN (C++ class), 81
- sl::CAMERA_SETTINGS_HUE (C++ class), 81
- sl::CAMERA_SETTINGS_LAST (C++ class), 81
- sl::CAMERA_SETTINGS_SATURATION (C++ class), 81
- sl::CAMERA_SETTINGS_WHITEBALANCE (C++ class), 81
- sl::CameraInformation (C++ class), 23
- sl::CameraInformation::calibration_parameters (C++ member), 23
- sl::CameraInformation::calibration_parameters_raw (C++ member), 23
- sl::CameraInformation::firmware_version (C++ member), 23
- sl::CameraInformation::serial_number (C++ member), 23
- sl::CameraParameters (C++ class), 23
- sl::CameraParameters::cx (C++ member), 24
- sl::CameraParameters::cy (C++ member), 24
- sl::CameraParameters::d_fov (C++ member), 24
- sl::CameraParameters::disto (C++ member), 24
- sl::CameraParameters::fx (C++ member), 24
- sl::CameraParameters::fy (C++ member), 24
- sl::CameraParameters::h_fov (C++ member), 24
- sl::CameraParameters::image_size (C++ member), 24
- sl::CameraParameters::SetUp (C++ function), 24
- sl::CameraParameters::v_fov (C++ member), 24
- sl::cameraResolution (C++ member), 88
- sl::COORDINATE_SYSTEM (C++ type), 82
- sl::COORDINATE_SYSTEM_IMAGE (C++ class), 82
- sl::COORDINATE_SYSTEM_LAST (C++ class), 82
- sl::COORDINATE_SYSTEM_LEFT_HANDED_Y_UP (C++ class), 82
- sl::COORDINATE_SYSTEM_LEFT_HANDED_Z_UP (C++ class), 82
- sl::COORDINATE_SYSTEM_RIGHT_HANDED_Y_UP (C++ class), 82
- sl::COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP (C++ class), 82
- sl::COPY_TYPE (C++ type), 79
- sl::COPY_TYPE_CPU_CPU (C++ class), 79
- sl::COPY_TYPE_CPU_GPU (C++ class), 79
- sl::COPY_TYPE_GPU_CPU (C++ class), 79
- sl::COPY_TYPE_GPU_GPU (C++ class), 79
- sl::DEPTH_FORMAT (C++ type), 83
- sl::DEPTH_FORMAT_LAST (C++ class), 84
- sl::DEPTH_FORMAT_PFM (C++ class), 84
- sl::DEPTH_FORMAT_PGM (C++ class), 84
- sl::DEPTH_FORMAT_PNG (C++ class), 84
- sl::DEPTH_MODE (C++ type), 81
- sl::DEPTH_MODE_LAST (C++ class), 82
- sl::DEPTH_MODE_MEDIUM (C++ class), 81
- sl::DEPTH_MODE_NONE (C++ class), 81
- sl::DEPTH_MODE_PERFORMANCE (C++ class), 81

sl::DEPTH_MODE_QUALITY (C++ class), 81
 sl::depthMode2str (C++ function), 78
 sl::double1 (C++ type), 79
 sl::double2 (C++ type), 79
 sl::double3 (C++ type), 79
 sl::double4 (C++ type), 79
 sl::ERROR_CODE (C++ type), 86
 sl::ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE (C++ class), 86
 sl::ERROR_CODE_CAMERA_NOT_DETECTED (C++ class), 86
 sl::ERROR_CODE_CAMERA_NOT_INITIALIZED (C++ class), 87
 sl::ERROR_CODE_CORRUPTED_SDK_INSTALLATION (C++ class), 87
 sl::ERROR_CODE_CUDA_ERROR (C++ class), 87
 sl::ERROR_CODE_FAILURE (C++ class), 86
 sl::ERROR_CODE_INVALID_COORDINATE_SYSTEM (C++ class), 87
 sl::ERROR_CODE_INVALID_FIRMWARE (C++ class), 87
 sl::ERROR_CODE_INVALID_FUNCTION_CALL (C++ class), 87
 sl::ERROR_CODE_INVALID_RESOLUTION (C++ class), 86
 sl::ERROR_CODE_INVALID_SVO_FILE (C++ class), 86
 sl::ERROR_CODE_LAST (C++ class), 87
 sl::ERROR_CODE_LOW_USB_BANDWIDTH (C++ class), 86
 sl::ERROR_CODE_NO_GPU_COMPATIBLE (C++ class), 86
 sl::ERROR_CODE_NOT_A_NEW_FRAME (C++ class), 87
 sl::ERROR_CODE_NOT_ENOUGH_GPUMEM (C++ class), 86
 sl::ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE (C++ class), 87
 sl::ERROR_CODE_SVO_RECORDING_ERROR (C++ class), 87
 sl::errorCode2str (C++ function), 88
 sl::float1 (C++ type), 79
 sl::float2 (C++ type), 79
 sl::float3 (C++ type), 79
 sl::float4 (C++ type), 79
 sl::InitParameters (C++ class), 27
 sl::InitParameters::camera_buffer_count_linux (C++ member), 28
 sl::InitParameters::camera_disable_self_calib (C++ member), 28
 sl::InitParameters::camera_fps (C++ member), 27
 sl::InitParameters::camera_image_flip (C++ member), 28
 sl::InitParameters::camera_linux_id (C++ member), 27
 sl::InitParameters::camera_resolution (C++ member), 27
 sl::InitParameters::coordinate_system (C++ member), 28
 sl::InitParameters::coordinate_units (C++ member), 28
 sl::InitParameters::depth_minimum_distance (C++ member), 28
 sl::InitParameters::depth_mode (C++ member), 28
 sl::InitParameters::InitParameters (C++ function), 27
 sl::InitParameters::load (C++ function), 27
 sl::InitParameters::save (C++ function), 27
 sl::InitParameters::sdk_gpu_id (C++ member), 28
 sl::InitParameters::sdk_verbose (C++ member), 28
 sl::InitParameters::svo_input_filename (C++ member), 27
 sl::InitParameters::svo_real_time_mode (C++ member), 27
 sl::Mat (C++ class), 35
 sl::Mat::~Mat (C++ function), 37
 sl::Mat::alloc (C++ function), 37
 sl::Mat::channels (C++ member), 42
 sl::Mat::clone (C++ function), 41
 sl::Mat::copyTo (C++ function), 38
 sl::Mat::data_type (C++ member), 42
 sl::Mat::free (C++ function), 37
 sl::Mat::getChannels (C++ function), 40
 sl::Mat::getDataType (C++ function), 40
 sl::Mat::getHeight (C++ function), 40
 sl::Mat::getInfos (C++ function), 41
 sl::Mat::getMemoryType (C++ function), 40
 sl::Mat::getPixelBytes (C++ function), 41
 sl::Mat::getPtr (C++ function), 40
 sl::Mat::getResolution (C++ function), 40
 sl::Mat::getStep (C++ function), 41
 sl::Mat::getStepBytes (C++ function), 40
 sl::Mat::getValue (C++ function), 39
 sl::Mat::getWidth (C++ function), 40
 sl::Mat::getWidthBytes (C++ function), 41
 sl::Mat::init (C++ member), 42
 sl::Mat::isInit (C++ function), 41
 sl::Mat::isMemoryOwner (C++ function), 41
 sl::Mat::Mat (C++ function), 35–37
 sl::Mat::mem_type (C++ member), 42
 sl::Mat::memory_owner (C++ member), 42
 sl::Mat::name (C++ member), 41
 sl::Mat::operator= (C++ function), 38
 sl::Mat::pixel_bytes (C++ member), 42
 sl::Mat::ptr_cpu (C++ member), 42
 sl::Mat::ptr_gpu (C++ member), 42
 sl::Mat::ptr_internal (C++ member), 42
 sl::Mat::read (C++ function), 38
 sl::Mat::setFrom (C++ function), 38
 sl::Mat::setTo (C++ function), 39
 sl::Mat::setValue (C++ function), 39
 sl::Mat::size (C++ member), 42
 sl::Mat::step_cpu (C++ member), 42
 sl::Mat::step_gpu (C++ member), 42

sl::Mat::updateCPUfromGPU (C++ function), 38
 sl::Mat::updateGPUfromCPU (C++ function), 38
 sl::Mat::verbose (C++ member), 41
 sl::Mat::write (C++ function), 39
 sl::MAT_TYPE (C++ type), 80
 sl::MAT_TYPE_32F_C1 (C++ class), 80
 sl::MAT_TYPE_32F_C2 (C++ class), 80
 sl::MAT_TYPE_32F_C3 (C++ class), 80
 sl::MAT_TYPE_32F_C4 (C++ class), 80
 sl::MAT_TYPE_8U_C1 (C++ class), 80
 sl::MAT_TYPE_8U_C2 (C++ class), 80
 sl::MAT_TYPE_8U_C3 (C++ class), 80
 sl::MAT_TYPE_8U_C4 (C++ class), 80
 sl::Matrix3f (C++ class), 42
 sl::Matrix3f::getInfos (C++ function), 43
 sl::Matrix3f::identity (C++ function), 44
 sl::Matrix3f::inverse (C++ function), 43, 44
 sl::Matrix3f::Matrix3f (C++ function), 42
 sl::Matrix3f::matrix_name (C++ member), 43
 sl::Matrix3f::nbElem (C++ member), 44
 sl::Matrix3f::operator
 = (C++ function), 43
 sl::Matrix3f::operator() (C++ function), 43
 sl::Matrix3f::operator* (C++ function), 42, 43
 sl::Matrix3f::operator== (C++ function), 43
 sl::Matrix3f::r (C++ member), 43
 sl::Matrix3f::r00 (C++ member), 43
 sl::Matrix3f::r01 (C++ member), 43
 sl::Matrix3f::r02 (C++ member), 43
 sl::Matrix3f::r10 (C++ member), 43
 sl::Matrix3f::r11 (C++ member), 43
 sl::Matrix3f::r12 (C++ member), 43
 sl::Matrix3f::r20 (C++ member), 43
 sl::Matrix3f::r21 (C++ member), 43
 sl::Matrix3f::r22 (C++ member), 43
 sl::Matrix3f::setIdentity (C++ function), 43
 sl::Matrix3f::setZeros (C++ function), 43
 sl::Matrix3f::transpose (C++ function), 43, 44
 sl::Matrix3f::zeros (C++ function), 44
 sl::Matrix4f (C++ class), 44
 sl::Matrix4f::getInfos (C++ function), 46
 sl::Matrix4f::identity (C++ function), 47
 sl::Matrix4f::inverse (C++ function), 45, 47
 sl::Matrix4f::m (C++ member), 47
 sl::Matrix4f::m30 (C++ member), 46
 sl::Matrix4f::m31 (C++ member), 46
 sl::Matrix4f::m32 (C++ member), 46
 sl::Matrix4f::m33 (C++ member), 47
 sl::Matrix4f::Matrix4f (C++ function), 44, 45
 sl::Matrix4f::matrix_name (C++ member), 47
 sl::Matrix4f::nbElem (C++ member), 47
 sl::Matrix4f::operator
 = (C++ function), 45
 sl::Matrix4f::operator() (C++ function), 45
 sl::Matrix4f::operator* (C++ function), 45
 sl::Matrix4f::operator== (C++ function), 45
 sl::Matrix4f::r00 (C++ member), 46
 sl::Matrix4f::r01 (C++ member), 46
 sl::Matrix4f::r02 (C++ member), 46
 sl::Matrix4f::r10 (C++ member), 46
 sl::Matrix4f::r11 (C++ member), 46
 sl::Matrix4f::r12 (C++ member), 46
 sl::Matrix4f::r20 (C++ member), 46
 sl::Matrix4f::r21 (C++ member), 46
 sl::Matrix4f::r22 (C++ member), 46
 sl::Matrix4f::setIdentity (C++ function), 45
 sl::Matrix4f::setSubMatrix3f (C++ function), 45
 sl::Matrix4f::setSubVector3f (C++ function), 45
 sl::Matrix4f::setSubVector4f (C++ function), 46
 sl::Matrix4f::setZeros (C++ function), 45
 sl::Matrix4f::transpose (C++ function), 45, 47
 sl::Matrix4f::tx (C++ member), 46
 sl::Matrix4f::ty (C++ member), 46
 sl::Matrix4f::tz (C++ member), 46
 sl::Matrix4f::zeros (C++ function), 47
 sl::MEASURE (C++ type), 82
 sl::MEASURE_CONFIDENCE (C++ class), 83
 sl::MEASURE_DEPTH (C++ class), 83
 sl::MEASURE_DISPARITY (C++ class), 82
 sl::MEASURE_LAST (C++ class), 83
 sl::MEASURE_XYZ (C++ class), 83
 sl::MEASURE_XYZABGR (C++ class), 83
 sl::MEASURE_XYZARGB (C++ class), 83
 sl::MEASURE_XYZBGRA (C++ class), 83
 sl::MEASURE_XYZRGBA (C++ class), 83
 sl::MEM (C++ type), 79
 sl::MEM_CPU (C++ class), 79
 sl::MEM_GPU (C++ class), 79
 sl::Mesh (C++ class), 47
 sl::Mesh::~~Mesh (C++ function), 47
 sl::Mesh::applyTexture (C++ function), 48
 sl::Mesh::cam_param (C++ member), 49
 sl::Mesh::clear (C++ function), 48
 sl::Mesh::filter (C++ function), 48
 sl::Mesh::im_pool (C++ member), 49
 sl::Mesh::Indice (C++ class), 26
 sl::Mesh::Indice::uv_ind (C++ member), 26
 sl::Mesh::Indice::v_vn_ind (C++ member), 26
 sl::Mesh::load (C++ function), 48
 sl::Mesh::material_indices (C++ member), 49
 sl::Mesh::max_d (C++ member), 49
 sl::Mesh::memory (C++ member), 49
 sl::Mesh::Mesh (C++ function), 47
 sl::Mesh::min_d (C++ member), 49
 sl::Mesh::normals (C++ member), 49
 sl::Mesh::save (C++ function), 48
 sl::Mesh::Texture (C++ class), 62
 sl::Mesh::Texture::indice_gl (C++ member), 62

sl::Mesh::Texture::name (C++ member), 62
 sl::Mesh::Texture::texture (C++ member), 62
 sl::Mesh::textures (C++ member), 49
 sl::Mesh::triangles (C++ member), 48
 sl::Mesh::uv (C++ member), 49
 sl::Mesh::vertices (C++ member), 48
 sl::Mesh::welded (C++ member), 49
 sl::MESH_FILE_FORMAT (C++ type), 86
 sl::MESH_FILE_LAST (C++ class), 86
 sl::MESH_FILE_OBJ (C++ class), 86
 sl::MESH_FILE_PLY (C++ class), 86
 sl::MESH_FILE_PLY_BIN (C++ class), 86
 sl::MESH_TEXTURE_FORMAT (C++ type), 86
 sl::MESH_TEXTURE_LAST (C++ class), 86
 sl::MESH_TEXTURE_RGB (C++ class), 86
 sl::MESH_TEXTURE_RGBA (C++ class), 86
 sl::MeshFilterParameters (C++ class), 49
 sl::MeshFilterParameters::FILTER (C++ type), 49
 sl::MeshFilterParameters::FILTER_HIGH (C++ class), 49
 sl::MeshFilterParameters::FILTER_LOW (C++ class), 49
 sl::MeshFilterParameters::FILTER_MEDIUM (C++ class), 49
 sl::MeshFilterParameters::filtering (C++ member), 50
 sl::MeshFilterParameters::load (C++ function), 50
 sl::MeshFilterParameters::MeshFilterParameters (C++ function), 50
 sl::MeshFilterParameters::save (C++ function), 50
 sl::MeshFilterParameters::set (C++ function), 50
 sl::operator| (C++ function), 88
 sl::operator<< (C++ function), 88
 sl::Orientation (C++ class), 52
 sl::Orientation (C++ member), 88
 sl::Orientation::getRotation (C++ function), 54
 sl::Orientation::identity (C++ function), 54
 sl::Orientation::normalise (C++ function), 54
 sl::Orientation::operator() (C++ function), 53
 sl::Orientation::operator* (C++ function), 53
 sl::Orientation::Orientation (C++ function), 53
 sl::Orientation::setIdentity (C++ function), 54
 sl::Orientation::setRotation (C++ function), 53
 sl::Orientation::setZeros (C++ function), 54
 sl::Orientation::zeros (C++ function), 54
 sl::POINT_CLOUD_FORMAT (C++ type), 84
 sl::POINT_CLOUD_FORMAT_LAST (C++ class), 84
 sl::POINT_CLOUD_FORMAT_PCD_ASCII (C++ class), 84
 sl::POINT_CLOUD_FORMAT_PLY_ASCII (C++ class), 84
 sl::POINT_CLOUD_FORMAT_VTK_ASCII (C++ class), 84
 sl::POINT_CLOUD_FORMAT_XYZ_ASCII (C++ class), 84
 sl::Pose (C++ class), 54
 sl::Pose::~~Pose (C++ function), 55
 sl::Pose::getOrientation (C++ function), 55
 sl::Pose::getRotation (C++ function), 55
 sl::Pose::getRotationVector (C++ function), 55
 sl::Pose::getTranslation (C++ function), 55
 sl::Pose::Pose (C++ function), 54
 sl::Pose::pose_confidence (C++ member), 55
 sl::Pose::pose_data (C++ member), 55
 sl::Pose::timestamp (C++ member), 55
 sl::Pose::valid (C++ member), 55
 sl::RecordingState (C++ class), 55
 sl::RecordingState::average_compression_ratio (C++ member), 56
 sl::RecordingState::average_compression_time (C++ member), 55
 sl::RecordingState::current_compression_ratio (C++ member), 55
 sl::RecordingState::current_compression_time (C++ member), 55
 sl::RecordingState::status (C++ member), 55
 sl::REFERENCE_FRAME (C++ type), 85
 sl::REFERENCE_FRAME_CAMERA (C++ class), 85
 sl::REFERENCE_FRAME_LAST (C++ class), 85
 sl::REFERENCE_FRAME_WORLD (C++ class), 85
 sl::Resolution (C++ class), 56
 sl::RESOLUTION (C++ type), 80
 sl::resolution2str (C++ function), 78
 sl::Resolution::area (C++ function), 56
 sl::Resolution::height (C++ member), 56
 sl::Resolution::operator = (C++ function), 56
 sl::Resolution::operator== (C++ function), 56
 sl::Resolution::Resolution (C++ function), 56
 sl::Resolution::width (C++ member), 56
 sl::RESOLUTION_HD1080 (C++ class), 80
 sl::RESOLUTION_HD2K (C++ class), 80
 sl::RESOLUTION_HD720 (C++ class), 80
 sl::RESOLUTION_LAST (C++ class), 80
 sl::RESOLUTION_VGA (C++ class), 80
 sl::Rotation (C++ class), 56
 sl::Rotation (C++ member), 88
 sl::Rotation::getOrientation (C++ function), 57
 sl::Rotation::getRotationVector (C++ function), 57
 sl::Rotation::Rotation (C++ function), 56, 57
 sl::Rotation::setOrientation (C++ function), 57
 sl::Rotation::setRotationVector (C++ function), 57
 sl::RuntimeParameters (C++ class), 57
 sl::RuntimeParameters::enable_depth (C++ member), 58
 sl::RuntimeParameters::enable_point_cloud (C++ member), 58
 sl::RuntimeParameters::load (C++ function), 58
 sl::RuntimeParameters::move_point_cloud_to_world_frame (C++ member), 58

sl::RuntimeParameters::RuntimeParameters (C++ function), 58

sl::RuntimeParameters::save (C++ function), 58

sl::RuntimeParameters::sensing_mode (C++ member), 58

sl::SELF_CALIBRATION_STATE (C++ type), 81

sl::SELF_CALIBRATION_STATE_FAILED (C++ class), 81

sl::SELF_CALIBRATION_STATE_LAST (C++ class), 81

sl::SELF_CALIBRATION_STATE_NOT_STARTED (C++ class), 81

sl::SELF_CALIBRATION_STATE_RUNNING (C++ class), 81

sl::SELF_CALIBRATION_STATE_SUCCESS (C++ class), 81

sl::SENSING_MODE (C++ type), 82

sl::SENSING_MODE_FILL (C++ class), 82

sl::SENSING_MODE_LAST (C++ class), 82

sl::SENSING_MODE_STANDARD (C++ class), 82

sl::sensingMode2str (C++ function), 78

sl::sleep_ms (C++ function), 88

sl::SPATIAL_MAPPING_STATE (C++ type), 85

sl::SPATIAL_MAPPING_STATE_FPS_TOO_LOW (C++ class), 85

sl::SPATIAL_MAPPING_STATE_INITIALIZING (C++ class), 85

sl::SPATIAL_MAPPING_STATE_LAST (C++ class), 85

sl::SPATIAL_MAPPING_STATE_NOT_ENABLED (C++ class), 85

sl::SPATIAL_MAPPING_STATE_NOT_ENOUGH_MEMORY (C++ class), 85

sl::SPATIAL_MAPPING_STATE_OK (C++ class), 85

sl::SpatialMappingParameters (C++ class), 59

sl::SpatialMappingParameters::allowed_max (C++ member), 61

sl::SpatialMappingParameters::allowed_min (C++ member), 61

sl::SpatialMappingParameters::allowed_resolution (C++ member), 61

sl::SpatialMappingParameters::get (C++ function), 61

sl::SpatialMappingParameters::interval (C++ type), 60

sl::SpatialMappingParameters::load (C++ function), 60

sl::SpatialMappingParameters::max_memory_usage (C++ member), 61

sl::SpatialMappingParameters::RANGE (C++ type), 60

sl::SpatialMappingParameters::RANGE_FAR (C++ class), 60

sl::SpatialMappingParameters::RANGE_MEDIUM (C++ class), 60

sl::SpatialMappingParameters::range_meter (C++ member), 61

sl::SpatialMappingParameters::RANGE_NEAR (C++ class), 60

sl::SpatialMappingParameters::RESOLUTION (C++ type), 59

sl::SpatialMappingParameters::RESOLUTION_HIGH (C++ class), 59

sl::SpatialMappingParameters::RESOLUTION_LOW (C++ class), 59

sl::SpatialMappingParameters::RESOLUTION_MEDIUM (C++ class), 59

sl::SpatialMappingParameters::resolution_meter (C++ member), 61

sl::SpatialMappingParameters::save (C++ function), 60

sl::SpatialMappingParameters::save_texture (C++ member), 61

sl::SpatialMappingParameters::set (C++ function), 60

sl::SpatialMappingParameters::SpatialMappingParameters (C++ function), 60

sl::spatialMappingState2str (C++ function), 78

sl::statusCode2str (C++ function), 78

sl::str2mode (C++ function), 78

sl::str2unit (C++ function), 78

sl::SUCCESS (C++ class), 86

sl::SVO_COMPRESSION_MODE (C++ type), 85

sl::SVO_COMPRESSION_MODE_LAST (C++ class), 86

sl::SVO_COMPRESSION_MODE_LOSSLESS (C++ class), 86

sl::SVO_COMPRESSION_MODE_LOSSY (C++ class), 86

sl::SVO_COMPRESSION_MODE_RAW (C++ class), 85

sl::TRACKING_STATE (C++ type), 84

sl::TRACKING_STATE_FPS_TOO_LOW (C++ class), 84

sl::TRACKING_STATE_LAST (C++ class), 84

sl::TRACKING_STATE_OFF (C++ class), 84

sl::TRACKING_STATE_OK (C++ class), 84

sl::TRACKING_STATE_SEARCHING (C++ class), 84

sl::TrackingParameters (C++ class), 62

sl::TrackingParameters::area_file_path (C++ member), 63

sl::TrackingParameters::enable_spatial_memory (C++ member), 62

sl::TrackingParameters::initial_world_transform (C++ member), 62

sl::TrackingParameters::load (C++ function), 62

sl::TrackingParameters::save (C++ function), 62

sl::TrackingParameters::TrackingParameters (C++ function), 62

sl::trackingState2str (C++ function), 78

sl::Transform (C++ class), 63

sl::Transform (C++ member), 88

sl::Transform::getOrientation (C++ function), 64

sl::Transform::getRotation (C++ function), 64

sl::Transform::getRotationVector (C++ function), 64

sl::Transform::getTranslation (C++ function), 64

- sl::Transform::setOrientation (C++ function), 64
- sl::Transform::setRotation (C++ function), 64
- sl::Transform::setRotationVector (C++ function), 64
- sl::Transform::setTranslation (C++ function), 64
- sl::Transform::Transform (C++ function), 63
- sl::Translation (C++ class), 65
- sl::Translation (C++ member), 88
- sl::Translation::normalize (C++ function), 65, 66
- sl::Translation::operator() (C++ function), 65
- sl::Translation::operator* (C++ function), 65
- sl::Translation::Translation (C++ function), 65
- sl::uchar1 (C++ type), 79
- sl::uchar2 (C++ type), 79
- sl::uchar3 (C++ type), 79
- sl::uchar4 (C++ type), 79
- sl::uint1 (C++ type), 79
- sl::uint2 (C++ type), 79
- sl::uint3 (C++ type), 79
- sl::uint4 (C++ type), 79
- sl::UNIT (C++ type), 82
- sl::unit2str (C++ function), 78
- sl::UNIT_CENTIMETER (C++ class), 82
- sl::UNIT_FOOT (C++ class), 82
- sl::UNIT_INCH (C++ class), 82
- sl::UNIT_LAST (C++ class), 82
- sl::UNIT_METER (C++ class), 82
- sl::UNIT_MILLIMETER (C++ class), 82
- sl::VIEW (C++ type), 83
- sl::VIEW_CONFIDENCE (C++ class), 83
- sl::VIEW_DEPTH (C++ class), 83
- sl::VIEW_LAST (C++ class), 83
- sl::VIEW_LEFT (C++ class), 83
- sl::VIEW_LEFT_GRAY (C++ class), 83
- sl::VIEW_LEFT_UNRECTIFIED (C++ class), 83
- sl::VIEW_LEFT_UNRECTIFIED_GRAY (C++ class), 83
- sl::VIEW_RIGHT (C++ class), 83
- sl::VIEW_RIGHT_GRAY (C++ class), 83
- sl::VIEW_RIGHT_UNRECTIFIED (C++ class), 83
- sl::VIEW_RIGHT_UNRECTIFIED_GRAY (C++ class), 83
- sl::VIEW_SIDE_BY_SIDE (C++ class), 83
- SL_EXPORT_DLL (C macro), 145
- SMOOTH (C++ class), 100
- smooth (C++ function), 95
- smooth_data (C++ function), 117
- smooth_l1_cpu (C++ function), 108, 109
- SOFTMAX (C++ class), 99
- softmax (C++ function), 108, 109
- softmax_array (C++ function), 138
- softmax_cpu (C++ function), 108, 109
- softmax_layer (C++ type), 138
- sorta_shuffle (C++ function), 140, 142
- sortable_bbox (C++ class), 59
- sortable_bbox::classes (C++ member), 59
- sortable_bbox::elo (C++ member), 59
- sortable_bbox::elos (C++ member), 59
- sortable_bbox::filename (C++ member), 59
- sortable_bbox::index (C++ member), 59
- sortable_bbox::net (C++ member), 59
- sortable_bbox::probs (C++ member), 59
- SortMaster3000 (C++ function), 111
- SpeakerDetect (built-in class), 88
- speakerDetect (RoboyVision attribute), 77
- SpeakerDetect.DetectSpeaker() (built-in function), 88
- SpeakerProc (RoboyVision attribute), 77
- SpeakerQueue (RoboyVision attribute), 77
- speed (C++ function), 93
- split_data (C++ function), 118, 119
- split_str (C++ function), 141, 142
- SSE (C++ class), 100
- STAIR (C++ class), 99
- stair_activate (C++ function), 106
- stair_gradient (C++ function), 106
- start_recognize_face (face_detection attribute), 71
- START_TIMER (C macro), 145
- statistics_connected_layer (C++ function), 102, 112
- statistics_net (C++ function), 93
- STB_IMAGE_IMPLEMENTATION (C macro), 122
- STB_IMAGE_WRITE_IMPLEMENTATION (C macro), 122
- STBI_default (C++ class), 138
- STBI_grey (C++ class), 138
- STBI_grey_alpha (C++ class), 138
- stbi_io_callbacks (C++ class), 61
- stbi_io_callbacks::eof (C++ member), 61
- stbi_io_callbacks::read (C++ member), 61
- stbi_io_callbacks::skip (C++ member), 61
- STBI_rgb (C++ class), 138
- STBI_rgb_alpha (C++ class), 138
- stbi_uc (C++ type), 138
- STBI_VERSION (C macro), 138
- stbi_write_bmp (C++ function), 139
- stbi_write_hdr (C++ function), 139
- stbi_write_png (C++ function), 139
- stbi_write_tga (C++ function), 139
- STBIDEF (C macro), 138
- std (C++ type), 88
- STEP (C++ class), 100
- STEPS (C++ class), 100
- STOP_TIMER (C macro), 145
- str (FreezeModel attribute), 73
- string_to_board (C++ function), 94
- string_to_layer_type (C++ function), 133
- strip (C++ function), 104, 141, 142
- strip_char (C++ function), 141, 142
- STUDY_DATA (C++ class), 100
- suicide_go (C++ function), 94

sum_array (C++ function), 141, 143
SUPER_DATA (C++ class), 100
SWAG_DATA (C++ class), 100
swap_binary (C++ function), 112, 113

T

TAG_DATA (C++ class), 100
TANH (C++ class), 99
tanh_activate (C++ function), 106
tanh_gradient (C++ function), 106
target (RoboyVision attribute), 77
test_blas (C++ function), 109
test_box (C++ function), 110
test_captcha (C++ function), 90
test_char_rnn (C++ function), 96
test_cifar (C++ function), 91
test_cifar_csv (C++ function), 91
test_cifar_csvtrain (C++ function), 91
test_cifar_multi (C++ function), 91
test_classifier (C++ function), 91
test_coco (C++ function), 92
test_dcgan (C++ function), 95
test_detector (C++ function), 92, 93
test_dice (C++ function), 94
test_dintersect (C++ function), 110
test_dunion (C++ function), 110
test_go (C++ function), 95
test_gpu_blas (C++ function), 109
test_lsd (C++ function), 95
test_resize (C++ function), 103, 125
test_super (C++ function), 97
test_tactic_rnn (C++ function), 96
test_tactic_rnn_multi (C++ function), 96
test_tag (C++ function), 97
test_voxel (C++ function), 97
test_writing (C++ function), 98
test_yolo (C++ function), 98
testFileExist (C++ function), 146
threat_classifier (C++ function), 91
three_way_max (C++ function), 124
three_way_min (C++ function), 124
threshold_image (C++ function), 102, 124
tile_images (C++ function), 123
time_random_matrix (C++ function), 109, 121
TIMING (C macro), 145
tisnan (C++ function), 135
TOO_CLOSE (C macro), 145
TOO_FAR (C macro), 145
top_k (C++ function), 104, 140
top_predictions (C++ function), 103, 131
total_boxes (face_detection attribute), 71
total_compares (C++ member), 111
trackProc (multiprocess attribute), 74
TrackQueue (multiprocess attribute), 74

TrackQueue (RoboyVision attribute), 77
train_captcha (C++ function), 90
train_char_rnn (C++ function), 96
train_cifar (C++ function), 91
train_cifar_distill (C++ function), 91
train_classifier (C++ function), 91
train_coco (C++ function), 92
train_colorizer (C++ function), 95
train_compare (C++ function), 111
train_dcgan (C++ function), 95
train_detector (C++ function), 93
train_dice (C++ function), 94
train_go (C++ function), 94
train_network (C++ function), 104, 130
train_network_datum (C++ function), 102, 130
train_network_sgd (C++ function), 101, 130
train_regressor (C++ function), 96
train_segmenter (C++ function), 97
train_super (C++ function), 97
train_swag (C++ function), 97
train_tag (C++ function), 97
train_voxel (C++ function), 97
train_writing (C++ function), 98
train_yolo (C++ function), 98
TransformCoordinates (built-in class), 88
TransformCoordinates.coordinate_transform() (built-in function), 89
translate_array (C++ function), 141, 142
translate_data_rows (C++ function), 118, 119
translate_image (C++ function), 124, 126
translate_image_channel (C++ function), 124
transpose_image (C++ function), 123
transpose_matrix (C++ function), 134
tree (C++ class), 66
tree::child (C++ member), 66
tree::group (C++ member), 66
tree::group_offset (C++ member), 66
tree::group_size (C++ member), 66
tree::groups (C++ member), 66
tree::leaf (C++ member), 66
tree::n (C++ member), 66
tree::name (C++ member), 66
tree::parent (C++ member), 66
try_classifier (C++ function), 91
TWO_PI (C macro), 142
type (FindObjectSrv attribute), 73
type (FreezeModel attribute), 73

U

update_args (C++ class), 66
update_args::adam (C++ member), 66
update_args::B1 (C++ member), 66
update_args::B2 (C++ member), 66
update_args::batch (C++ member), 66

update_args::decay (C++ member), 66
 update_args::eps (C++ member), 66
 update_args::learning_rate (C++ member), 66
 update_args::momentum (C++ member), 66
 update_args::t (C++ member), 66
 update_connected_layer (C++ function), 111, 112
 update_convolutional_layer (C++ function), 112, 113
 update_crnn_layer (C++ function), 114, 115
 update_deconvolutional_layer (C++ function), 119
 update_gru_layer (C++ function), 122
 update_local_layer (C++ function), 127, 128
 update_lstm_layer (C++ function), 128
 update_network (C++ function), 101, 130
 update_rnn_layer (C++ function), 136
 USET (C macro), 128, 136

V

valid_captcha (C++ function), 90
 valid_char_rnn (C++ function), 96
 valid_go (C++ function), 94
 valid_tactic_rnn (C++ function), 96
 validate_classifier_10 (C++ function), 91
 validate_classifier_crop (C++ function), 91
 validate_classifier_full (C++ function), 91
 validate_classifier_multi (C++ function), 91
 validate_classifier_single (C++ function), 91
 validate_coco (C++ function), 92
 validate_coco_recall (C++ function), 92
 validate_compare (C++ function), 111
 validate_detector (C++ function), 93
 validate_detector_flip (C++ function), 93
 validate_detector_recall (C++ function), 93
 validate_dice (C++ function), 94
 validate_yolo (C++ function), 98
 validate_yolo_recall (C++ function), 98
 variance_array (C++ function), 104, 141
 variance_cpu (C++ function), 108, 109
 variance_delta_cpu (C++ function), 107, 109
 vec_char_rnn (C++ function), 96
 version (setup attribute), 78
 version (setup_gpu attribute), 78
 visualize (C++ function), 93
 visualize_convolutional_layer (C++ function), 113
 visualize_network (C++ function), 103, 131
 visualize_normalization_layer (C++ function), 132
 Visualizer (built-in class), 89
 Visualizer.StartVisualization() (built-in function), 89
 VisualQueue (RoboyVision attribute), 77
 voc_label (built-in class), 89
 voc_label.convert() (built-in function), 89
 voc_label.convert_annotation() (built-in function), 89
 voc_names (C++ member), 98

W

w (example attribute), 68
 wd (voc_label attribute), 89
 weighted_delta_cpu (C++ function), 108, 109
 weighted_sum_cpu (C++ function), 108, 109
 what_time_is_it_now (C++ function), 103, 140, 142
 windows (C++ member), 125
 write_all (C++ function), 141, 142
 write_all_fail (C++ function), 141, 142
 write_int (C++ function), 141, 142
 write_label (C++ function), 125
 WRITING_DATA (C++ class), 100
 ws (RosMsgUtil attribute), 78

X

XNOR (C++ class), 100

Y

yolo_cleanup (C++ function), 143
 yolo_detect (C++ function), 143
 yolo_handle (C++ type), 143
 yolo_init (C++ function), 143
 yolo_obj (C++ class), 66
 yolo_obj::boxes (C++ member), 67
 yolo_obj::darknet_path (C++ member), 67
 yolo_obj::names (C++ member), 67
 yolo_obj::net (C++ member), 67
 yolo_obj::nms (C++ member), 67
 yolo_obj::probs (C++ member), 67
 yolo_test (C++ function), 143
 yuv_to_rgb (C++ function), 124, 126

Z

ZED_SDK_MAJOR_VERSION (C++ member), 145
 ZED_SDK_MINOR_VERSION (C++ member), 145
 ZED_SDK_PATCH_VERSION (C++ member), 145
 ZEDcudaSafeCall (C macro), 145
 zedRoboy (built-in class), 89
 zedRoboy.classify() (built-in function), 89
 zedRoboy.detect() (built-in function), 89
 zedRoboy.letterbox_img() (built-in function), 89
 zedRoboy.load_img() (built-in function), 89
 zedRoboy.load_meta() (built-in function), 89
 zedRoboy.load_net() (built-in function), 89
 zedRoboy.predict() (built-in function), 89
 zero_objectness (C++ function), 102, 135