SlampPy

Release 1.0

Giordano Laminetti, Filippo Aleotti

Apr 19, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

This project is a python wrapper to SLAM algorithms. At the moment, we provide support for:

- ORB_SLAM2
- ORB_SLAM3

The installation rules can be found *here* .

1.1 Example of usage

in the project can be found two jupyter notebook:

- **example_usage** shows how to use ORB_SLAM2 with a sequence of the KITTI dataset
- **trajectory_example** which provides a video sequence that draws the camera trajectory and the point cloud

it is also present an **run.py** script where, by providing a video sequence, it saves the trajectory as a .txt file for the possible argument see the *Run.py references*

To change the algorithm and its parameter settings see *parameters file*

1.2 Credits

Our project has been developed starting from other repositories, in particular:

- Python bindings to ORB Slam are based on the repo
- ORB Slam 2 has been cloned from the ORB_SLAM2 original repository
- ORB Slam 3 has been cloned from the ORB_SLAM3 original repository

CHAPTER

TWO

INSTALLATION

this project require the installiation of at least one of the supported algorithm, each installation is describet in it's section. We also provide a Docker container on DockerHub, but you can also build your own container all the instruction see *Docker container*.

2.1 Install ORB_SLAM2

for ORB_SLAM2 you need to install a modify version of the original that can be found at ORB_SLAM2 ,

after you have correctly installed the C++ version you need to install the python wrapper form here ORB_SLAM2-PythonBindings

2.2 Install ORB_SLAM3

similar to ORB_SLAM2 for ORB_SLAM3 we use a modify version of ORB_SLAM3 that can be found at ORB_SLAM3

after you have correctly installed the C++ version you need to install the python wrapper form here ORB_SLAM2-PythonBindings

2.3 Install Slampy

you need to install poetry , the istrucntion can be found at this link

after the installation of dependences you can download the repo

```
git clone https://github.com/GiordanoLaminetti/SlamPy.git
```

enter in the root folder and type

```
poetry shell
```

this create a python venv in which install all the requirements

CHAPTER

THREE

SLAMPY

3.1 kitti_odometry module

class kitti_odometry.**KittiEvalOdom**

Bases: object

Evaluate odometry result Usage example:

vo_eval = KittiEvalOdom() vo_eval.eval(gt_pose_txt_dir, result_pose_txt_dir)

calc_sequence_errors(*poses_gt*, *poses_result*)

calculate sequence error :param poses_gt: {idx: 4x4 array}, ground truth poses :type poses_gt: dict :param poses_result: {idx: 4x4 array}, predicted poses :type poses_result: dict

Returns

[first_frame, rotation error, translation error, length, speed]

- first_frame: frist frame index
- rotation error: rotation error per length
- translation error: translation error per length
- length: evaluation trajectory length
- speed: car speed (#FIXME: 10FPS is assumed)

Return type err (list list)

compute_ATE(*gt*, *pred*)

Compute RMSE of ATE :param gt: ground-truth poses :type gt: 4x4 array dict :param pred: predicted poses :type pred: 4x4 array dict

compute_RPE(*gt*, *pred*)

Compute RPE :param gt: ground-truth poses :type gt: 4x4 array dict :param pred: predicted poses :type pred: 4x4 array dict

Returns rpe_trans rpe_rot

compute_overall_err(*seq_err*)

Compute average translation & rotation errors :param seq_err: [[r_err, t_err],[r_err, t_err],...]

- r_err (float): rotation error
- t_err (float): translation error

Returns average translation error ave_r_err (float): average rotation error

Return type ave_t_err (float)

compute_segment_error(*seq_errs*)
This function calculates average errors for different segment. :param seq_errs: list of errs; [first_frame, rotation error, translation error, length, speed]

- first_frame: frist frame index
- rotation error: rotation error per length
- translation error: translation error per length
- length: evaluation trajectory length
- speed: car speed (#FIXME: 10FPS is assumed)

Returns {100:[avg_t_err, avg_r_err],...}

Return type avg_segment_errs (dict)

eval(*args*)
Evaulate required/available sequences :param gt_dir: ground truth poses txt files path :type gt_dir: str :param result_dir: pose predictions txt files directory :type result_dir: str :param alignment: if not None, optimize poses by

- scale: optimize scale factor for trajectory alignment and evaluation
- scale_7dof: optimize 7dof for alignment and use scale for trajectory evaluation
- 7dof: optimize 7dof for alignment and evaluation
- 6dof: optimize 6dof for alignment and evaluation

last_frame_from_segment_length(*dist*, *first_frame*, *length*)
Find frame (index) that away from the first_frame with the required distance :param dist: distance of each pose w.r.t frame-0 :type dist: float list :param first_frame: start-frame index :type first_frame: int :param length: required distance :type length: float

Returns end-frame index. if not found return -1

Return type i (int) / -1

load_poses_from_txt(*file_name*)
Load poses from txt (KITTI format) Each line in the file should follow one of the following structures

(1) idx pose(3x4 matrix in terms of 12 numbers)

(2) pose(3x4 matrix in terms of 12 numbers)

Parameters **file_name** (*str*) – txt file path

Returns {idx: 4x4 array}

Return type poses (dict)

plot_error(*avg_segment_errs*, *file_name*)
Plot per-length error :param avg_segment_errs: {100:[avg_t_err, avg_r_err],...} :type avg_segment_errs: dict :param file_name: the results file named. :type file_name: str

plot_trajectory(*poses_gt*, *poses_result*, *file_name*)
Plot trajectory for both GT and prediction :param poses_gt: {idx: 4x4 array}; ground truth poses :type poses_gt: dict :param poses_result: {idx: 4x4 array}; predicted poses :type poses_result: dict :param file_name: the results file named. :type file_name: str

rotation_error(*pose_error*)
Compute rotation error :param pose_error: relative pose error :type pose_error: 4x4 array

Returns rotation error

Return type rot_error (float)

save_sequence_errors(*err*, *file_name*)
Save sequence error :param err: error information :type err: list list :param file_name: txt file for writing errors :type file_name: str

scale_optimization(*gt*, *pred*)
Optimize scaling factor :param gt: ground-truth poses :type gt: 4x4 array dict :param pred: predicted poses :type pred: 4x4 array dict

Returns predicted poses after optimization

Return type new_pred (4x4 array dict)

trajectory_distances(*poses*)
Compute distance for each pose w.r.t frame-0 :param poses: {idx: 4x4 array} :type poses: dict

Returns distance of each pose w.r.t frame-0

Return type dist (float list)

translation_error(*pose_error*)
Compute translation error :param pose_error: relative pose error :type pose_error: 4x4 array

Returns translation error

Return type trans_error (float)

write_result(*f*, *seq*, *errs*)
Write result into a txt file :param f: :type f: IOWrapper :param seq: sequence number :type seq: int :param errs: [ave_t_err, ave_r_err, ate, rpe_trans, rpe_rot] :type errs: list

kitti_odometry.**scale_lse_solver**(*X*, *Y*)
Least-sqaure-error solver Compute optimal scaling factor so that s(X)-Y is minimum :param X: current data :type X: KxN array :param Y: reference data :type Y: KxN array

Returns scaling factor

Return type scale (float)

kitti_odometry.**umeyama_alignment**(*x*, *y*, *with_scale=False*)
Computes the least squares solution parameters of an Sim(m) matrix that minimizes the distance between a set of registered points. Umeyama, Shinji: Least-squares estimation of transformation parameters

between two point patterns. IEEE PAMI, 1991

Parameters

- **x** – mxn matrix of points, m = dimension, n = nr. of data points
- **y** – mxn matrix of points, m = dimension, n = nr. of data points
- **with_scale** – set to True to align also the scale (default: 1.0 scale)

Returns r, t, c - rotation matrix, translation vector and scale factor

3.2 run module

Run the SLAM system and save, at each frame, the current depth and pose

```
usage: run [-h] [--dataset DATASET] [--settings SETTINGS] [--dest DEST]
           [--pose_id POSE_ID] [--is_evaluate_depth] [--is_evaluate_pose]
           [--is_bash] [--data_type {TUM,KITTI_VO,KITTI,OTHERS}]
           [--gt_depth GT_DEPTH] [--gt_pose_dir GT_POSE_DIR]
           [--gt_pose_txt GT_POSE_TXT] [--align {scale,scale_7dof,7dof,6dof}]
           [--named NAMED]
```

3.2.1 Named Arguments

--dataset path to dataset

Default: "/media/Datasets/KITTI_VO/dataset/sequences/10"

--settings which configuration?

Default: "./settings_kitty.yaml"

--dest where do we save artefacts?

Default: "./results_kitty_vo_10"

--pose_id between which frames do you want compute the pose? If pose_id==-1, get the pose between 0->T; if pose_id >0, compute the pose between T-pose_id->T For instance, if pose_id=2 then compute the pose between T-2->T

Default: -1

--is_evaluate_depth If set, will evalute the orb depth with the gt files

Default: False

--is_evaluate_pose If set, will evalute the orb pose with the gt files

Default: False

--is_bash If set, means use bash shell to evaluate

Default: False

--data_type Possible choices: TUM, KITTI_VO, KITTI, OTHERS

which dataset type

Default: "KITTI_VO"

--gt_depth the gt depth files of the dataset

Default: "/media/Datasets/KITTI_VO_SGM/10/depth"

--gt_pose_dir each frame's gt pose file, saved as previous to current, and filename as current.npy

Default: "/media/Datasets/KITTI_VO_SGM/10/npy_pose"

--gt_pose_txt this is the gt pose file provided by kitty or tum.

Default: "/media/Datasets/KITTI_VO/dataset/poses/10.txt"

--align Possible choices: scale, scale_7dof, 7dof, 6dof

alignment type

	Default: "7dof"
--named	the names for saving pose
	Default: "kitty_vo_10"

3.3 slampy module

class slampy.**Sensor**(*value*)
Bases: enum.Enum

This class is used to set the type of the sensor

Values: MONOCULAR STEREO MONOCULAR_IMU STEREO_IMU RGBD

MONOCULAR = 1

MONOCULAR_IMU = 3

RGBD = 5

STEREO = 2

STEREO_IMU = 4

class slampy.**State**(*value*)
Bases: enum.Enum

This class is used to get the actual state of the tracking Values:

OK LOST NOT_INITIALIZED

LOST = 2

NOT_INITIALIZED = 3

OK = 1

SYSTEM_NOT_READY = 4

class slampy.**System**(*params_file*, *sensor_type*)
Bases: object

This class is a wrapper for the SLAM method in the slam_method folder,

Build the wrapper

Parameters

- **params_file** (*str*) – the Path to the .yaml file.
- **sensor_type** (*Enum*) – the sensort type of the SLAM

get_abs_cloud()
Get the point cloud at the current frame stored in absolute coordinates.

Returns an array with the 3D coordinate of the point, None if the traking is failed

get_camera_matrix()
Get the camera instrinsics

Returns np.ndarray with shape 3x4 containing the camera parameters.

get_depth()
Get the depth computed in the current image.

Returns an array of the image shape with depth when it is aviable otherwise -1 , None if the traking is failed

get_point_cloud()
Get the point cloud at the current frame form the wiev of the current position .

Returns an array with the 3D coordinate of the point, None if the traking is failed

get_point_cloud_colored()
Get the point cloud at the current frame form the wiev of the current position with the RGB color of the point .

Returns an array with the 3D coordinate of the point and the relative RGB color, None if the traking is failed

get_pose_from_target()
Get the pose from the current frame T to the reference one 0.

get_pose_to_target(*precedent_frame=- 1*)
Get the pose between a previous frame X in the sequence and the current one T.

The param *precedent_frame* allows to distinguish between different situations: * precedent_frame = -1: X is the first frame of the sequence. We get the pose between 0->T * precedent_frame > 0 : X is frame at (T-precedent_frame). For instance, if precedent_frame=1, it means we are computing the pose (T-1) -> T.

Parameters **precedent_frame** – id of the frame to use when computing the pose between frame. Default is -1 (i.e., pose 0->T)

Returns the 4x4 pose matrix corresponding to the transormation between the precedent_frame and the current one. If the state is not State.OK, return None

Examples

```
>>> slam.get_pose_to_target() # return the pose 0 -> T
>>> slam.get_pose_to_target(precedent_frame=1) # return the pose (T-1) -> T
>>> slam.get_pose_to_target(precedent_frame=2) # return the pose (T-2) -> T
```

get_state()
Get the current state of the SLAM system

Returns a State corresponding to the state

process_image_imu_mono(*image*, *tframe*, *imu*)
Process an image mono with the imu data.

Note: it works only if the sensor type is MONOCULAR_IMU

Parameters

- **image** (*ndarray*) – image as HxWx3
- **tframe** (*float*) – the timestamp when the image was capture
- **imu** – the imu data stored in an float array in the form of [AccX ,AccY ,AccZ, GyroX, vGyroY, vGyroZ, Timestamp]

Returns the state of the traking in this frame

Raises **Exception** – if the sensor type is different from MONOCULAR_IMU

process_image_imu_stereo(*image_left*, *image_right*, *tframe*, *imu*)
Process an image stereo with the imu data.

Note: it work only if the sensor type is STEREO_IMU

Parameters

- **image_left** (*ndarray*) – left image as HxWx3
- **image_right** (*ndarray*) – right image as HxWx3
- **tframe** (*float*) – the timestamp when the image was capture
- **imu** – the imu data stored in an float array in the form of [AccX ,AccY ,AccZ, GyroX, vGyroY, vGyroZ, Timestamp]

Returns the state of the traking in this frame

Raises **Exception** – if the sensor type is different from STEREO_IMU

process_image_mono(*image*, *tframe*)
Process an image mono.

Note: it works only if the sensor type is MONOCULAR

Parameters

- **image** – ndarray of the image
- **tframe** (*float*) – the timestamp when the image was capture

Returns the state of the traking in this frame

Raises **Exception** – if the sensor type is different from MONOCULAR

process_image_rgbd(*image*, *tframe*)
Process an rgbd image.

Note: it works only if the sensor type is RGBD

Parameters

- **image** (*ndarray*) – RGBD image as HxWx4
- **tframe** (*float*) – the timestamp when the image was capture

Returns the new state of the SLAM system

Raises **Exception** – if the sensor type is different from RGBD

process_image_stereo(*image_left*, *image_right*, *tframe*)
Process a stereo pair.

Note: it works only if the sensor type is STEREO

Parameters

- **image_left** (*ndarray*) – left image as HxWx3
- **image_right** (*ndarray*) – right image as HxWx3
- **tframe** (*float*) – the timestamp when the image was capture

Returns the state of the traking in this frame

Raises **Exception** – if the sensor type is different from STEREO

reset()
Reset SLAM system

shutdown()
Shutdown the SLAM system

3.4 trajectory_drawer module

class trajectory_drawer.**TrajectoryDrawer**(*params_file*, *width=None*, *height=None*, *drawpointcloud=True*, *useFigureWidget=True*)

Bases: object

This class is used to print a the trajectory result from the slam method in a sequence of images, it use the Plotily library

Build the Trajectory drawer

Parameters

- **params_file** (*str*) – the Path to the .yaml file.
- **width** (*int*) – the width of figure in pixel. Defaults to None
- **height** (*int*) – the height of figure in pixel. Defaults to None
- **drawpointcloud** (*bool*) – if is false the plot show only trajectory and not the point cloud. Defaults to True
- **useFigureWidget** (*bool*) – use the plotily.graph_object.FigureWidget instance if false it used the plotily.graph_object.Figure

get_figure()
Return the figure

plot_trajcetory(*slampy_app*)
Compute the trajectory and add it to the figure :param slampy_app: the slampy instance used to compute the pose in the image :type slampy_app: Slampy

3.5 utils module

utils.**compute_ate**(*gtruth_xyz*, *pred_xyz_o*)

utils.**compute_errors**(*gt*, *pred*)
Computation of error metrics (abs rel,sq rel, rmse, rmse log) between predicted and ground truth depths From https://github.com/mrharicot/monodepth

Parameters

- **gt** – an array with the ground truth values
- **pred** – an array with the predicted values

Returns the error

Return type abs_rel,sq_rel,rmse,rmse_log

utils.**convert_scale**(*points*, *gt_depth*)
convert the scale of the predictions to the gt

Parameters

- **points** – the predictions depth
- **gt_filename** – the gt references filename

Returns: the ratio between the predictions and the gt

utils.**create_dir**(*directory*)
Create a directory if not exists :param directory: directory to create

Returns None, but it creates a new folder if not exists

utils.**dump_xyz**(*source_to_target_transformations*)

utils.**evaluate_pose**(*args*)
Evaluate odometry on the KITTI dataset

utils.**get_error**(*args*, *filename*, *points*, *gt_filename*)
Get the realtive gt from it's filename and convert the scale of the predictions in order to compute the error

Parameters

- **points** – the predictions depth
- **gt_filename** – the gt references filename

Returns the error computed on this examples

utils.**get_error_KITTI**(*points*, *gt_filename*)
Get the realtive gt from it's filename and convert the scale of the predictions in order to compute the error

Parameters

- **points** – the predictions depth
- **gt_filename** – the gt references filename

Returns the error computed on this examples

utils.**get_error_TUM**(*points*, *gt_filename*)
Get the realtive gt from it's filename and convert the scale of the predictions in order to compute the error

Parameters

- **points** – the predictions depth
- **gt_filename** – the gt references filename

Returns the error computed on this examples

utils.**load_IMU_datas_TUM_VI**(*path_to_sequence*)

utils.**load_images_EuRoC**(*path_to_sequence*)
This loader is created for Visual Inertial EuRoC datasets. Format of such datasets is: path_to_sequence/mav0/cam0/+data/xxxx.png

/-times.txt

utils.**load_images_KITTI**(*path_to_sequence*)
Return the sequence of the images found in the path and the corrispondent timestamp

Parameters **path_to_sequence** – the sequence in witch we can found the image sequences

Returns : two array : one contains the sequence of the image filename and the second the timestamp in whitch they are acquired

utils.**load_images_KITTI_VO**(*path_to_sequence*)
Return the sequence of the images found in the path and the corrispondent timestamp

Parameters **path_to_sequence** – the sequence in witch we can found the image sequences

Returns : two array : one contains the sequence of the image filename and the second the timestamp in whitch they are acquired

utils.**load_images_OTHERS**(*path_to_sequence*)
Return the sequence of the images found in the path and the corrispondent timestamp

Parameters **path_to_sequence** – the sequence in witch we can found the image sequences

Returns : two array : one contains the sequence of the image filename and the second the timestamp in whitch they are acquired

Inside of path_to_sequence must be: +data

xxxxxxxx.png xxxxxxxy.png ....

-times.txt

where times.txt simply contains timestamps of every frame

utils.**load_images_TUM**(*path_to_sequence*, *file_name*)
Return the sequence of the images found in the path and the corrispondent timestamp

Parameters **path_to_sequence** – the sequence in witch we can found the image sequences

Returns one contains the sequence of the image filename and the second the timestamp in whitch they are acquired

Return type two array

utils.**load_images_TUM_VI**(*path_to_sequence*)
This loader is created for Visual Inertial TUM datasets. Format of such datasets is: path_to_sequence/mav0/cam0/+data/xxxx.png

/-times.txt

utils.**read_depth_KITTI**(*filename*)
loads depth map D from png file and returns it as a numpy array,

utils.**read_depth_TUM**(*filename*)
loads depth map D from png file and returns it as a numpy array,

utils.**save_depth**(*dest*, *depth*)
Save depth as 16 bit png file

Parameters

- **dest** – path to new 16 bit png image wiht depth, w/o exension
- **depth** – depth to save, as ndarray HxW

Returns None, but a new 16 bit png image will be saved at dest

utils.**save_depth_err_results**(*file_path*, *filename*, *err*)

utils.**save_pose**(*dest*, *pose*)
Save pose as npy file

Parameters

- **dest** – path to new npy file wiht pose, w/o exension
- **pose** – ndarray with 4x4 pose matrix (as R|t in homogeneous notation)

Returns None, but it creates a new npy file with the pose

utils.**save_pose_and_times_txt**(*args*, *name*, *pose*)
Save pose and time in two different txt files.

utils.**save_pose_txt**(*args*, *name*, *pose*)
Save pose as txt file

Parameters

- **dir** – directory of pose.txt
- **name** – frame name or id
- **pose** – ndarray with 4x4 pose matrix (as Rlt in homogeneous notation)

Returns None, but it creates a new npy file with the pose

CHAPTER

FOUR

DOCKER CONTAINER

We provide a container on dockerhub, where all the dipendences and the repository are already installed.

4.1 Download docker container

On docker hub there are 4 containers with a different tagname :

- `base` the base container in which only the dipendences are installed
- `orbslam2` a container with only installed the ORB_SLAM2 library
- `orbslam3` a container with only installed the ORB_SLAM3 library
- `latest` a container with installed all the library

for downloading the container you can type

```
docker pull giordanolaminetti/slampy:tagname
```

with the addiction of `-focal` after `tagname` you can download the builds with ubuntu 20.04 (focal fossa), by default the docker are built over ubuntu 18.04 (Bionic Beaver):

```
docker pull giordanolaminetti/slampy:latest-focal
```

with the addiction of `-ros-melodic` after `tagname` you can download the ubuntu 18.04 (Bionic Beaver) with ros noetic already installed

```
docker pull giordanolaminetti/slampy:latest-ros-melodic
```

with the addiction of `-ros-noetic` after `tagname` you can download the ubuntu 20.04 (focal fossa) with ros noetic already installed

```
docker pull giordanolaminetti/slampy:latest-ros-noetic
```

4.2 Build docker container

you can also build your own container, with the shell in the root path, typing this command

```
docker build . -t slampy [--build-args arg=value]*
```

you can change the image name by replacing `slampy` with the desired image name, the ``build args `` can be:

- `base_container` (String) you can change the base container, by default it is `ubuntu:18.04`, it can also be used to install multiple algorithm
- `build_dependences` (Bool) build and install the dependences packages (pangolin, Eigen, Opencv, Python3.8, . . .) default `true`
- `build_orbslam2` (Bool) build and install ORB_SLAM2 and the Python wrapper default `true`
- `build_orbslam3` (Bool) build and install ORB_SLAM3 and the Python wrapper default `true`

For example if you want to build only the ORB_SLAM2 package from the base you can type

```
docker build . -t slampy:orbslam2 --build-arg build_dependences=false --build-arg␣
↪build_orbslam3=false --build-arg base_container=giordanolaminetti/slampy:base
```

For example if you want to install only ORB_SLAM2 and ORB_SLAM3 you can type

```
docker build . -t slampy:orbslam2+3 --build-arg build_dependences=false --build-arg␣
↪build_orbslam2=false --build-arg base_container=giordanolaminetti/slampy:orbslam2
```

4.3 Run docker image

When the image is ready, you can create a new container running:

```
NAME="orb"
DATAPATH="/PATH/TO/KITTI/DATE/DRIVE_SYNC_FOLDER/"
IMAGE_NAME="giordanolaminetti/slampy"
TAG="latest"
sudo docker run -it \
                --name $NAME \
                --mount type=bind,source="$(pwd)",target=/slampy/slampy \
                -v $DATAPATH:"/slampy/slampy/Dataset":ro \
                -p 8888:8888 --rm\
                $IMAGE_NAME:$TAG /bin/bash
```

Doing so, the created container contains both the code and the Dataset (in read-only mode to prevent wrong behaviours)

You can have a test with the *jupyter* example running:

```
jupyter notebook --ip 0.0.0.0
```

after the close connection it will remove the container, if you want to preserve it then remove the ``-rm `` options

CHAPTER

FIVE

SETTINGS.YAML

it is the configuration file and it is used to change the current algorithm and the other parameters of the algorithm.

5.1 change the current algorithm

To change the algorithm settings you can modify the **setting.yaml** file in the line

```
SLAM.alg:'insert here your .py file that contains the wrapper class'
```

5.2 ORB_SLAM2 and ORB_SLAM3 settings

the other params for the ORB_SLAM2/3 algorithm are :

- **SLAM.vocab_path**: "Path to the ORB_SLAM2/3 vocabular file"
- **SLAM.settings_path**: "Path to the ORB_SLAM2/3 .yaml settings file"

5.3 add your own settings

if you add your method with *this* you can add in **settings.yaml** the params needed for your application execution in the form

```
params_name : 'params_values'
```

5.4 Drawer Params

See **setting.yaml** for examples values

- **Drawer.eye.x ** determines the x view point about the origin of this scene.
- **Drawer.eye.y ** determines the y view point about the origin of this scene.
- **Drawer.eye.z ** determines the z view point about the origin of this scene.
- **Drawer.center.x ** determines the x plane translation about the origin of this scene.
- **Drawer.center.y ** determines the y plane translation about the origin of this scene.
- **Drawer.scale_grade.x ** determines the zoom about x plane.

- **Drawer.scale_grade.y ** determines the zoom about y plane.
- **Drawer.scale_grade.z ** (float): determines the zoom about z plane.
- **Drawer.aspectratio.x** to set the x-axis aspecratio.
- **Drawer.aspectratio.y** to set the y-axis aspecratio.
- **Drawer.aspectratio.z** to set the z-axis aspecratio.
- **Drawer.point_size** the size of the marker point in pixel.

CHAPTER

SIX

ADD YOU OWN ALGORITHM

you can add you own algorithm to Slampy by adding a new `your_method_name.py` file with the specs found below, to the `slam_method` folder

the file must contain a class named `Slam` with the method shown below

```
import sys
sys.path.append("..")
from slampy import Sensor
from slampy import State

class Slam:
    def __init__(self, params, sensor_type):

    def process_image_mono(self, image, tframe):

    def process_image_stereo(self, image_left, image_right, tframe):

    def process_image_imu_mono(self, image, tframe, imu):

    def process_image_imu_stereo(self, image_left, image_right, tframe, imu):

    def process_image_rgbd(self, image, tframe):

    def get_pose_to_target(self):

    def get_abs_cloud(self):

    def get_camera_matrix(self):

    def get_state(self):

    def reset(self):

    def shutdown(self):
```

6.1 Slam class references

In this section we describe all the methods with the parameters that you need to write in order to add your algorithm to Slampy.

You can add your own accessor method, but the method described below must be present with this signature and return the exact value as described

class Slam

__init__(*self*, *params*, *sensor_type*)
Initialize your algorithm with the params from the settings.yaml and the sensor type

Parameters

- **params** (*dict*) – the setting.yaml params conver to Python dictionary
- **sensor_type** (*Pyslam.Sensor*) – the type of the sensor as istance of Sensor Class

there are one process_image method for each of the sensor type in Sensor Class. If your algorithm hasn't one of this methods you can add an exception every time that one calls that sensor processing method

process_image_mono(*self*, *image*, *tframe*)
pass the image to the slam system and compute the tracking

Parameters

- **image** (*numpy.ndarray*) – an RGB image
- **tframe** (*float*) – the timestamp in which the frame is caputred

Raises **Exception** – if the Sensor is different from MONOCULAR

process_image_stereo(*self*, *image_left*, *image_right*, *tframe*)
pass the images to the slam system and compute the tracking

Parameters

- **image_left** (*numpy.ndarray*) – an RGB image
- **image_right** (*numpy.ndarray*) – an RGB image
- **tframe** (*float*) – the timestamp in which the frame is caputred

Raises **Exception** – if the Sensor is different from STEREO

process_image_imu_mono(*self*, *image*, *tframe*, *imu*)
pass the image to the slam system and compute the tracking

Parameters

- **image_left** (*numpy.ndarray*) – an RGB image
- **image_right** (*numpy.ndarray*) – an RGB image
- **tframe** (*float*) – the timestamp in which the frame is caputred
- **imu** (*numpy.ndarray*) – the imu data imu data stored in an float array in the form of [AccX ,AccY ,AccZ, GyroX, vGyroY, vGyroZ, Timestamp]

Raises **Exception** – if the Sensor is different from MONOCULAR_IMU

process_image_imu_stereo(*self*, *image_left*, *image_right*, *tframe*, *imu*)
pass the images to the slam system and compute the tracking

Parameters

- **image_left** (*numpy.ndarray*) – an RGB image
- **image_right** (*numpy.ndarray*) – an RGB image
- **tframe** (*float*) – the timestamp in which the frame is caputred
- **imu** (*numpy.ndarray*) – the imu data imu data stored in an float array in the form of [AccX ,AccY ,AccZ, GyroX, vGyroY, vGyroZ, Timestamp]

Raises **Exception** – if the Sensor is different from STEREO_IMU

process_image_rgbd(*self*, *image*, *tframe*)
pass the image to the slam system and compute the tracking

Parameters

- **image** (*numpy.ndarray*) – an RGBD image
- **tframe** (*float*) – the timestamp in which the frame is caputred

Raises **Exception** – if the Sensor is different from RGBD

get_pose_to_target(*self*)
return the pose between the references frame (usually the first frame) and the current one T.

Returns the pose computed from the references frame to the actual ones, None if the tracking is failed

Return type a 4x4 numpy array

get_abs_cloud(*self*)
return the point cloud at the current frame stored in absolute coordinates (coordinates from the references frame)

return an array with the 3D coordinate of the point, None if the traking is failed

rtype a nx3 numpy array

get_camera_matrix(*self*)
return the intrinsec parameter of cameras

return an array with the intrinsec parameter of cameras

rtype a 4x4 numpy array

get_state(*self*)
return the current state of the tracking as istance of State Class

return the state of the tracking

rtype Slampy.State

reset(*self*)
reset/initialize the slam tracking

shutdown(*self*)
shutdown the slam algorithm

PYTHON MODULE INDEX

INDEX

Symbols

C

D

E

G

K

L

M

N

O

P

R

S

T

U

W