2022-S1

FYW - POC

Find Your Way

Proof Of Concept

CISTERNAS Nicolas
GUILLERMAIN Nicolas
MONNERET Mathieu

SLAM

Publisher : Aubay Innov'

Date : 2022/02/09

State : WIP

Contents

Contents 1

Changes tracking 2

1 Introduction 2

2 Proposed Solution 3

2.1 Principle 3

2.1.1 Installation 4

2.2 Methodology 5

2.2.1 Pathfinding 6

2.3 Dataset (option) 10

3 Tests and results 10

4 Difficulties encountered 11

5 Conclusion 12

6 Acronyms and Abbreviations 13

Proof Of Concept

Object	Décrire l'objet du document et le contexte dans lequel il s'applique. Find Your Way is a project that aims to create a navigation system for Visually Impaired People. It is divided in three Proofs of Concept (POCs): Natural Language Processing (NLP), Computer Vision (CV) and Simultaneous Localization And Mapping (SLAM). This document explains our solution concerning the POC SLAM.

Changes tracking

State	Date	Writers	Verify by
WIP	2022/04/28	Cisternas, Guillermain, Monneret	
WIP	2022/05/05	Cisternas, Guillermain	
WIP	2022/05/16	Cisternas, Guillermain, Monneret	vvv
WIP	2022/05/05	Cisternas, Guillermain	
VAL	YYYY/MM/DD	xxx, yyy, zzz	vvv
…			

1 Introduction

[Rappeler l'objectif du projet et indiquer en quoi ce POC y participe.]

The Find Your Way (FYW) project aims to help visually impaired people (VIP) find their way in indoor environments. The Proof of Concept (POC) Simultaneous Localization and Mapping (SLAM) attempts to create a local map of the environment using a matching of Points of Interests (POI) between the frames seen during the run, to help locate and guide the person using the device. Paired with the POC NLP it will create an interface for the user that can travel in their local environment

[Décrire les différentes étapes de ce POC et son fonctionnement.]

We have spent a major part of the time given for the POC trying to figure out an installation procedure for ORB-SLAM3. We have attempted multiple installations on multiple Operating Systems (OS), such as Windows, Mac, The Windows Subsystem for Linux (WSL2) and a virtual machine (VM).

We were able to write a procedure to install ORB-SLAM3 on WSL2 and on a virtual machine.

We then needed to calibrate the camera to work with ORB-SLAM3, this calibration ensures that the pose estimation is closest in results to the real position of the camera.

Once ORB-SLAM3 was running on our machines, we took the source code in hand to understand how to adapt it to our needs.

Finally with the elements at our disposal, the idea was to implement an algorithm A* allowing us to find the shortest path between the position of the person and the destination.

2 Proposed Solution

Voici les documents d'installation correspondant à notre projet et à son utilisation:

- Installation documentation of ORB-SLAM3
- Calibration procedure with ROS
- Calibration procedure with Kalibr ?

This project is majorly based on the open-source software ORB_SLAM3, Fri, 23 Apr 2021.

[Citer le titre, les auteurs, les années]

3 Principle

[Dans un premier temps décrire brièvement la méthode/le github utilisé, parties par parties]

We worked on the Aubay specific GitLab in which we pushed all our modifications.

Our method is based on the ORB-SLAM3 software. It is an open-source project created in a university in Spain. It has been said to be the state of the art of Simultaneous Localization and Mapping (SLAM) methods.

The idea was to: install the repository, get it working and add to it the functionalities we wished for.

We have attempted and succeeded in the installation of the ORB-SLAM3 software. It is a C++ project built on many various libraries such as Eigen, Pangolin, Sophus, DBoW2, g2o, Kalibr, ROS, …

Before installing ORB-SLAM3, we had to install Eigen and Pangolin. Sophus, DBOW2 and g2o being already included in the repository.

The tricky part of the installation is the fact that the versions required seemed to conflict with each other. We had to investigate which version would suit all the libraries. For OpenCV for instance we had to install the version 4.2.

Having done this, we had to change parts of all the CMakeLists.txt, so as to allow the software to find the needed third parties.

[Entrez en suite dans les détails en décrivant les différentes parties une par une]

4 Installation

5 General

We have attempted the installation of ORB-SLAM3 on 4 different distributions and machines. We have tried on Windows 10, Mac OSX, WSL2 for Windows 10 and eventually on an Ubuntu 20.04 Virtual Machine on VirtualBox.

The installation on Windows was soon declared impossible. For ORB-SLAM3 had been coded for Linux in most part. Some of the Libraries were not available for Windows 10.

The installation on Mac OSX was only temporary, while working from home and not knowing if we had the right to work on WSL2. It seemed promising, being the right middle ground between Linux and Windows. But WSL2 was allowed and we shifted our attention towards it.

We tried to install ORB-SLAM3 on WSL2. It worked great. We even managed to launch the SLAM method on the video example that came with it. However, in the end our product is expected to work live with a stream of data coming from a webcam. The problem is that while WSL2 is able to access USB devices (given that one has the time to create a new kernel with the right drivers and all), and the webcam is visible in the USB drivers, the subsystem is not able to create the video device in /dev/video. Thus, the video stream is not accessible. Hence our product cannot work in live on WSL2.

We eventually turned our faces towards a Linux virtual machine. Ubuntu 20.04 on VirtualBox. This virtual Machine has enabled us to work on a complete Linux distribution without any problems with the installation, but rather problems with the virtual machine itself. It is a very nice technology but it comes with its share of troubles, whether that be the latency, the links with keyboards, the mouse that does not work sometimes, and even a corrupted system emerging out of nowhere. But it is the best solution we have found.

6 Eigen

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. It is required to have an efficient ORB-SLAM3.

To install Eigen, we had to search for the right version of this library. We have tried several versions which did not work for our installation of ORB-SLAM3. We eventually got to the 3.3.1 version. It suited all our needs.

7 Pangolin

Pangolin is a set of lightweight and portable utility libraries for prototyping 3D, numeric or video-based programs and algorithms. It is used to remove platform-specific boilerplate and make it easy to visualize data.

To install Pangolin, we had to look around on their GitHub. As a matter of fact, their latest version didn’t suit the installation of ORB-SLAM3. We have tried several versions which did not work for our installation of ORB-SLAM3. We eventually got to the version 0.6 on their GitHub.

8 ORB-SLAM3

Finally, we have got to the real stuff. The software we want to install, the cake that will someday require its cherry... Our addition to their work.

To install ORB-SLAM3, we have first cloned the repository coming with the v1.0 of the software. It came along with some requirements that did not fit our environment. At least the requirements seemed to conflict. We managed to get around by telling the software to look for our versions of all the softwares.

The install of ORB-SLAM3 starts first with the thirdparties available in their repository. DBoW2, g2o and Sophus. The versions asked for OpenCV could not be satisfied, so we changed the file so as the libraries would be looking for the version 4.2.

9 ROS

ROS stands for Robot Operating System. It is through this package that ORB-SLAM3 is able to access the camera stream and process it on the run.

It was quite difficult to understand fully the installation procedure of this package. It was supposed to be very straight forward. Requiring only a sudo apt install of all different parts of the package. However, it so happened to be difficult for some versions of the package on the Virtual Machine, telling us there were some broken packages and attempting to fix it ended in a loop of broken files. We tried it on WSL2 and it worked like a charm. Eventually we got the installation to work on the Virtual Machine by creating a whole new machine.

10 Kalibr

This thirdparty package is used to calibrate a camera. It will estimate the intrinsic parameters of a camera as well as its projection matrix.

11 Methodology

[Décrire ce que vous avez ensuite ajouté au POC et où cela s’inspire dans les différentes parties]

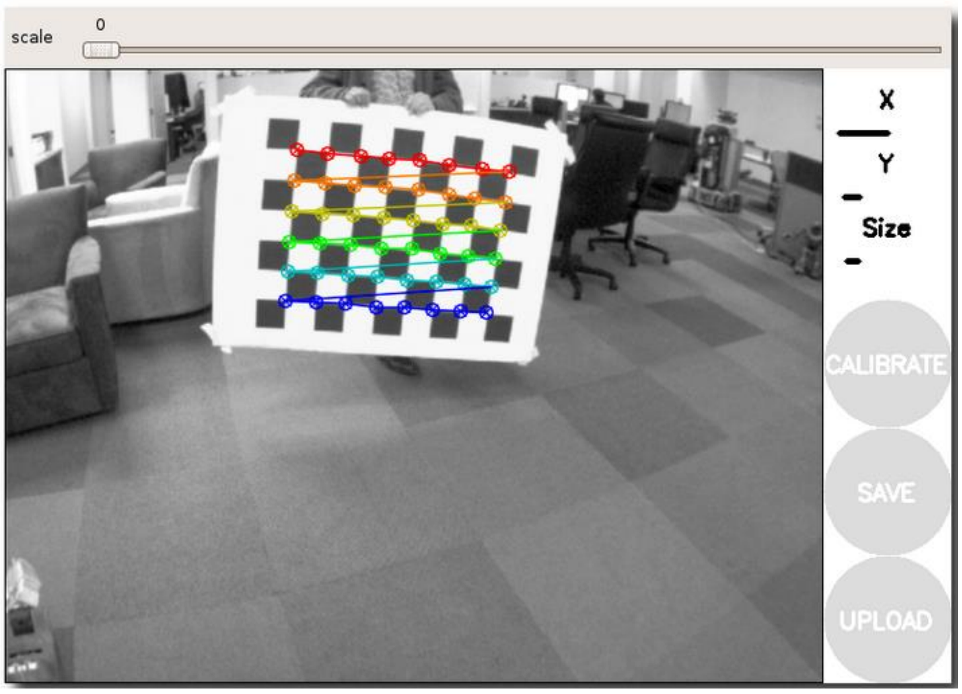
12 Calibration

The first step to a good SLAM method in real-time is to have a good calibration of the camera used. Indeed, the intrinsic and distortion parameters of a camera are the ones that define how an object in the world will be projected onto the image space. This is used with every new frame in the ORB_SLAM3 software to be able to estimate the pose of the camera from the previous keyframes. A good estimation leads to better results, closer to the reality.

These parameters differ between brands but also intra-brand between cameras from the same model. We then have to estimate them accurately.

The calibration can be done using two softwares, ROS or Kalibr. The former is a user-friendly software that gives feedback dynamically to the user to know whether the calibration is over or not. The latter is less user-friendly but is the method advised by the coders of ORB_SLAM3.

The calibration process in either method is done through the use of a known model (checkerboard, apriltag,…) containing black and white squares of known dimensions. The idea is to be able to estimate the deformation induced by the camera on a known template, using various footage of the checkerboard (With skew, with the checkerboard everywhere on the camera’s field of view, changing the distance between the checkerboard and the camera).



We have written a calibration procedure for both methods.

13 Pathfinding

After spending some time on the code to understand how each part works, we were able to start thinking about how to find the way from the initial position to the destination.

The idea is to be able to keep track of the path taken by the user when using the system, to be able to save some Points of Interest (POIs), and to find a path from the current location to one of the POIs saved using the A* algorithm. A lot of these functionalities are made possible thanks to ORB-SLAM3.

14 Grid coordinates

The coordinates given by ORB-SLAM3 are given as floats, they are usually found in the interval [0, 1] interval for short videos. We needed a way to project these coordinates on a grid so as to be able to find a path on a grid. To do so we needed to adapt the ORB-SLAM solution.

This projection is made fairly straight forward, taking the coordinates of the camera in the world (absolute) referential, multiplying them by a certain factor (10 for us), flooring these coordinates so as to have only the integer part of the coordinates and dividing by the factor previously used. This has allowed for a grid that is not too small which would, in real life, make the grid smaller than a step and make guiding the user too present, and not too large to make the error in the pathfindidng negligeable (Being twenty centimeters close is a relatively small error).

15 Structures defined

16 Coordinates

We have created a C++ structure ***Coord*** that has two integer attributes that is equivalent to a 2D vector to save the coordinates of the map.

17 Absolute coordinates

The absolute coordinates are the coordinates of the camera projected on a discrete grid. We have created a C++ structure ***AbsoluteCoord*** struct containing 5 different attributes:

- Int *x_min, x_max, y_min, y_max :* the dimensions of the map.
- Vector<***Coord>*** *coords :* the path taken by the camera.

We have also added setters, getters, savers and printers so as to be able to access these attributes.

18 Path

We have created a C++ structure ***Path*** that has two attributes:

- ***Coord*** *start:* that is the current position on the map when the path is saved that will be fed to the pathfinding algorithm as the start of the search.
- vector<vector<int>> *Map2D:* the matrix representation of the path taken by the user that can be fed to the pathfinding algorithm. It has dimensions (x_max – x_min + 1, y_max – y_min + 1).

This 2D map is created from the absolute coordinates. The idea is to be able to go from one representation to the other being able to get a path from the A* algorithm in the matrix representation and translate it to the ***AbsoluteCoord*** representation.

We have coded a method that is able to create an ***AbsoluteCoord*** instance from the ***Path*** representation. The *coords* attribute of this instance will be the list of ***Coord*** sorted in a chronological way to follow to get to the goal POI.

To add our code to ORB-SLAM3, we had to find where to implement our work. The idea is that a 2D local path should be saved inside the map it is a part of. To do so we have added the needed attributes to the class Map so that it can keep track of the path taken by the user.

19 Instances in Map

20 LocalPath

The attribute ***Path*** *localPath* is an instance of the structure ***Path*** which stores the local path taken by the user in the current map and the start position for the pathfinding algorithm.

21 Absolute2DCoordinates

The attribute ***AbsoluteCoord*** *absolute2DCoordinates* is an instance of the structure ***AbsoluteCoord*** that keeps track of the absolute coordinates followed by the user.

22 PoiList

It is the list of reachable Points of Interest in the map and will be the destination of the A* algorithm.

The attribute map<string, ***Coord***> *poiList* is a dictionary that maps a string, the name of the POI with its Coord.

This list is appended after a user trigger. It toggles a terminal request for a name and adds the corresponding name and Coord to the list.

23 A*

A* is a search algorithm that looks for the shortest path between two positions in a matrix. It is used in various applications, such as maps.

Imagine a square grid that has many obstacles, randomly dispersed. The initial cell (A) and the final cell (B) are provided. The goal is to reach the final cell in the shortest time.

🟧 **A**			
	*	🟦	🟦
🟦		*	
	🟦	*	🟦
			🟩 **B**

The A* algorithm has 3 parameters:

- G: the cost for moving from the initial cell to the current cell. It basically is the sum of all the cells that have been visited since the departure of the first cell.
- H: the heuristic value; this is the estimated cost of moving from the current cell to the final cell. The actual cost can only be calculated once the last cell is reached. Therefore, H is the estimated cost. We must make sure that there never is an overestimation of the cost.
- F = G + H

The way the algorithm makes its decisions is based on the F value. The algorithm selects the cell with the smallest F value and moves to that cell. This process continues until the algorithm reaches its target cell.

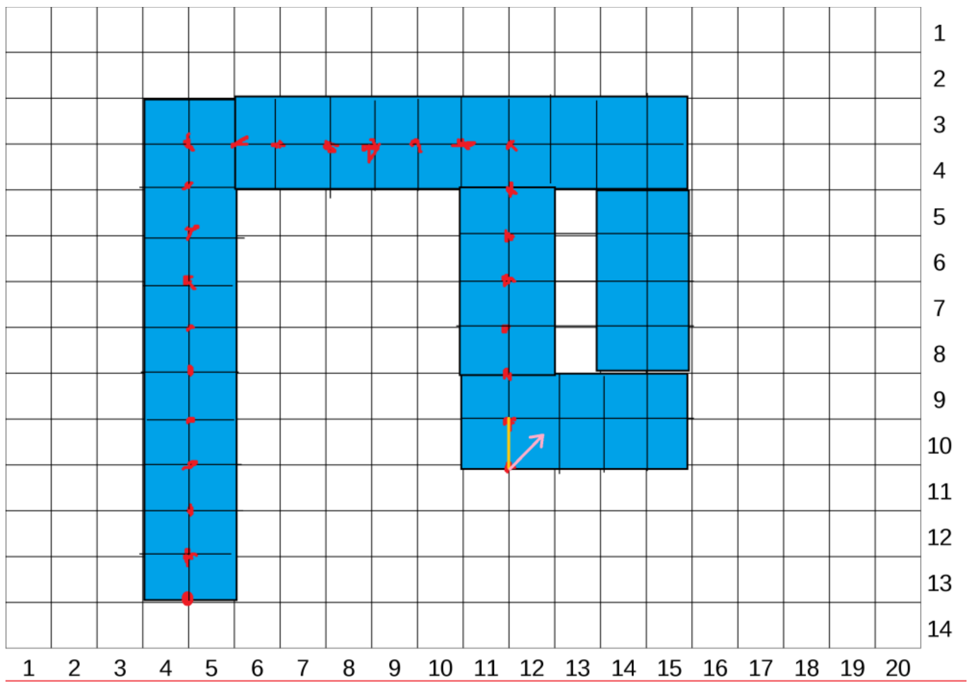
24 Return to POI

In this part we added a button (will be triggered by the POC NLP) on the viewer that prints the POI list and asks the user to choose the one he wants to go back to. Then the chosen coordinate is given to A* to find the walkable path from the current position to the chosen POI. This path is then sent to the function FYWBack.

25 FYWBack

FYWBack uses the output of A* to guide the user to the chosen POI. The output of A* is the sorted list of coordinates of the shortest walkable path between the current position and the chosen POI. From these coordinates we can then recreate the path section by section.

To guide the user, the goal is to use the normal vector to the camera's plane and compute the angle with the closest path section. If the current position is closer to the end of the segment, the current path section is updated to the next one and the program carries on computing the angle with the new segment.



26 Dataset (option)

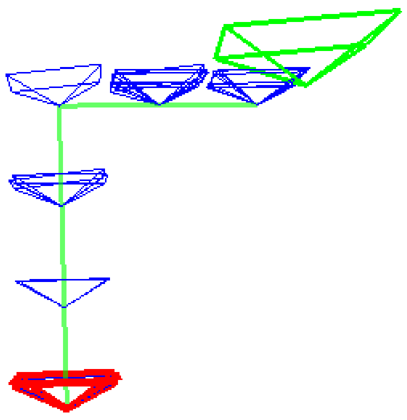
We have used videos taken inside the Aubay Innov Open Space as testing data. It has allowed us to test our work on ORB-SLAM3, check for mistakes and create the best version possible of our work.

We have tried to create an exhaustive dataset of videos. Our videos are created to test ORB-SLAM3 with translations only, rotations only and mixes of the two. Moving across a corridor which does not show much texture or across the open space that allows for much more POIs and more robust POIs.

27 Tests and results

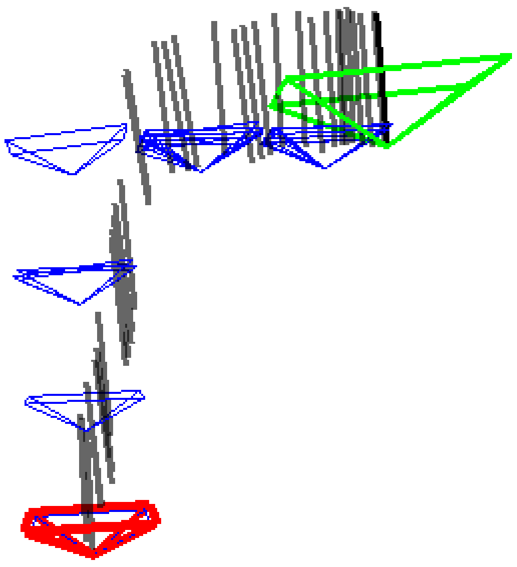
[Indiquer quelles ont été les différentes étapes de test effectuées.]

We tested our work on different videos taken in the OpenSpace.

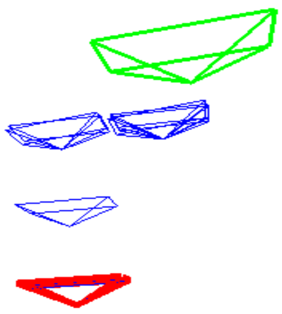


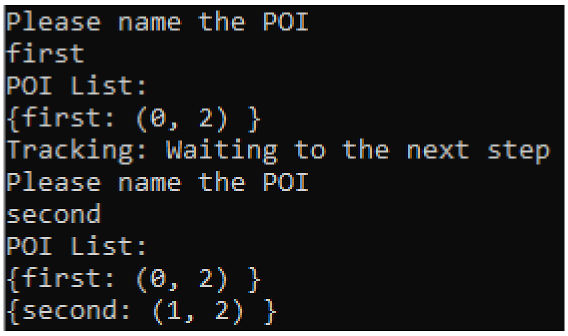
We attempted first to create a 2D grid of the local map

We then managed to create the normals to the keyframes' plane in real-time.



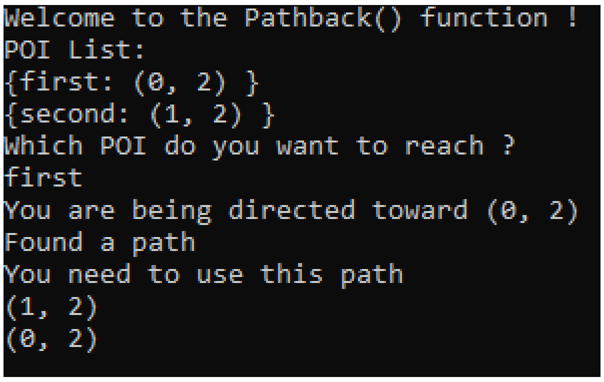
We managed to save and name the current position as a Point Of Interest.





```
Please name the POI
first
POI List:
{first: (0, 2) }
Tracking: Waiting to the next step
Please name the POI
second
POI List:
{first: (0, 2) }
{second: (1, 2) }
```

We created a function to return to one of the POIs saved.



```
Welcome to the Pathback() function !
POI List:
{first: (0, 2) }
{second: (1, 2) }
Which POI do you want to reach ?
first
You are being directed toward (0, 2)
Found a path
You need to use this path
(1, 2)
(0, 2)
```

Comparaison avec l'Etat de l'art

28 Difficulties encountered

[Décrire les problèmes rencontrés et les solutions trouvées]

[Même si une piste a été abandonnée, la détailler brièvement et expliquer la cause de l'abandon]

During this POC, we encountered many difficulties. Starting with the installation procedure which was not fully documented. We had to try different versions of the software to finally get the whole project working. As explained earlier, there have been problems with Windows, with some libraries that were not created to be run under Windows. Problems with Mac OSX which was not entirely figured out because we had started on WSL2 which was promising. But WSL2 had not the capacity of using a webcam. It could not access USB-drivers connected to the computer. We decided to work on a virtual machine.

We worked on virtualbox for a while and the machines started slowing down too regularly. We often had to create new virtual machine because we couldn't start them. We had to be very careful with our installation and the softwares we installed because anything could crash the VM.

One aspect of the POC that could probably have ensured better results is the quality of the camera. Indeed, using a webcam, manufactured to focus on a person using their computer, to capture an indoor environment is not the best choice made. The video seems out of focus and the images and textures are not sharp. This cannot ensure a good extraction of the POIs and the right matching of the frames.

The calibration part of the POC caused some trouble. Indeed, the two methods of calibration seem to output very different data from one to the other. Both seemingly accurate based on the attempts on ORB_SLAM3.

One of the difficulties encountered is the complexity of the code to grasp. ORB-SLAM3 is a big software. Starting to implement took a little while to get a good understanding of what is coded and where.

29 Conclusion

[Donnez votre opinion à propos des résultats obtenus et de comment vous avez réussi à les obtenir.]

[Si vous avez une quelconque idée de comment améliorer les résultats où sur des potentielles pistes pour étoffer le projet mettez les ici.]

In the end we managed to add a functionality to ORB-SLAM3 that enables a user to find their way back to a POI. With a good understanding of the code layout, we added our part and were able to evolve the original ORB-SLAM3 code.

Having a test video with the corresponding ground truth would be precious to be able to appreciate the accuracy of the calibration and its impact on the pose estimation. It would enable us to make a definitive choice between ROS and kalibr.

An improvement for this project could be the use of an embedded hardware with a good amount of memory and find a way to have computing power, probably a miniaturized GPU.

The potential of ORB_SLAM3 might not have been fully exploited. Having more time could have helped us figure out even better the code and could have allowed for better results.

30 Acronyms and Abbreviations

Term	Definition
FYW	Find Your Way
VIP	Visually Impaired Person
OS	Operating System
SLAM	Simultaneous Localization and Mapping
NLP	Natural Language Processing
WSL2	Windows Subsystem for Linux 2
VM	Virtual Machine
POI	Point of Interest
GPU	Graphics Processing Unit
CPU	Central Processing Unit
POC	Proof of Concept
ORB	Oriented FAST and Robust BRIEF, a fast method of feature extraction
FAST	Features From Accelerated Segment Test, a fast method of feature extraction
BRIEF	Binary Robust Independent Elementary Features, a method of feature extraction