Chapter 2

Simultaneous Localization And Mapping

2.1 Pose Estimation

In this chapter, we focus on the use of an embedded visual sensor for pose estimation. These sensors[1], mounted on-board moving robots, are used to gather information to map the environment and to estimate the robot's trajectory. To this end, VO and SLAM methods are dominant.

2.1.1 Visual Odometry

Overview

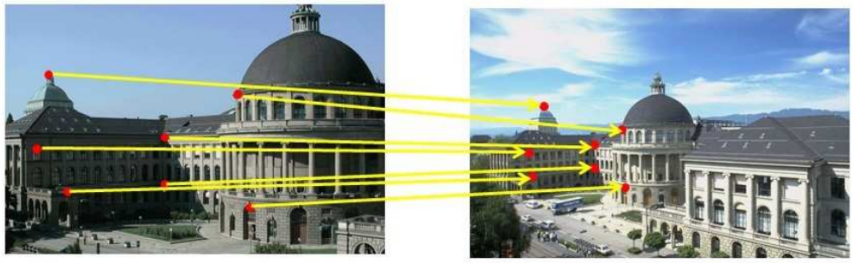
The visual odometry process consists in incrementally estimating the pose of a vehicle by examining the changes that motion induces on the images taken from its on-board rigidly attached camera [Scaramuzza and Fraundorfer, 2011]. VO is a 3D motion estimation (translation + rotation) computed from sequential optical sensors data such as images (See Figure 2.1).

Assuming static scenes, two consecutive images at time k and *k-1* are related by a rigid body transformation $\mathbf{T}_k$ in Eq.2.1

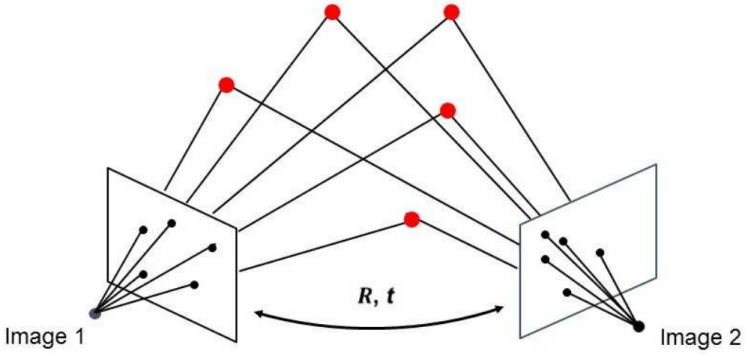
$$\mathbf{T}_k = \begin{bmatrix} \mathbf{R}_{k,k\text{-}1} & \mathbf{t}_{k,k\text{-}1} \\ \mathbf{O}_{1\text{x}3} & 1 \end{bmatrix} \tag{2.1}$$

Where $\mathbf{T}_k \in \mathbb{R}_{4\text{x}4}$ in the relative transformation, $\mathbf{R}_{k,k-1} \in \mathbb{R}_{3\text{x}3}$ is the rotation made on each axis from the previous pose to the next one, and

[1]The used visual sensors are supposed to be calibrated.



(a) Consecutive images correspondences.



(b) Consecutive images motion estimation.

Figure 2.1: The VO working principle. Images from [Schöps et al., 2014].

$\mathbf{t}_{k,k-1} \in \mathbb{R}_{3x1}$ is the translation on the three axes. Figure 2.2 shows an illustration of VO problem formulation where the following sets an be introduced:

- $\mathcal{T}_{0:n} = \{\mathbf{T_1}, \mathbf{T_2}, ..., \mathbf{T}_n\}$: The set of camera motions.
- $\mathcal{C}_{0:n} = \{\mathbf{C_0}, \mathbf{C_1}, ..., \mathbf{C}_n\}$: The set of camera poses w.r.t the initial coordinate frame.

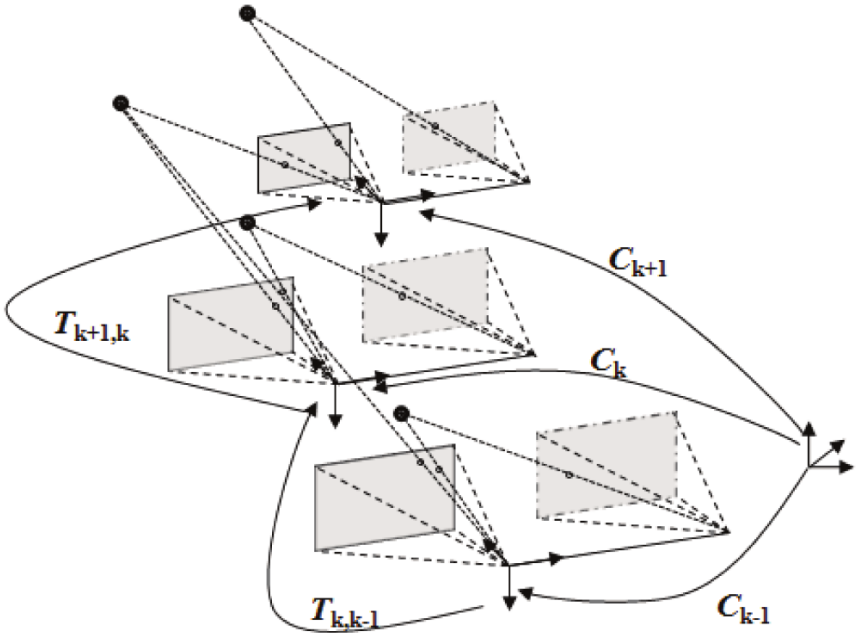


Figure 2.2: Illustration of the visual odometry problem. Case of a stereo camera (At time k, two boxes representing two image frames are available). Image from Scaramuzza and Fraundorfer, 2011.

The VO consists in estimating the transformation $\mathcal{T}_{0:n}$. Then, to recover the camera trajectory $\mathcal{C}_{0:n}$. the computed $\mathcal{T}_{0:n}$ is concatenated using Eq.2.2 where, the initial camera pose $\mathbf{C_0}$ is arbitrary set.

$$\mathbf{C}_n = \mathbf{C}_{n-1}\mathbf{T}_n \tag{2.2}$$

VO estimation requires assumptions such as:

- The environment has to be sufficiently illuminated and structured.

- The scene has to be mostly static.
- The consecutive frames need sufficient overlap among them.

Related work

VO can be computed by two approaches classified into: Sparse visual odometry Engel et al., 2016 (where only a part of the image data is used) and dense visual odometry Kerl et al., 2013b (where all the available image data are used). According to Schops et al., 2014, the existing VO methods (also SLAM methods) can be classified into Feature-based method and Direct method (See Figure 2.3).

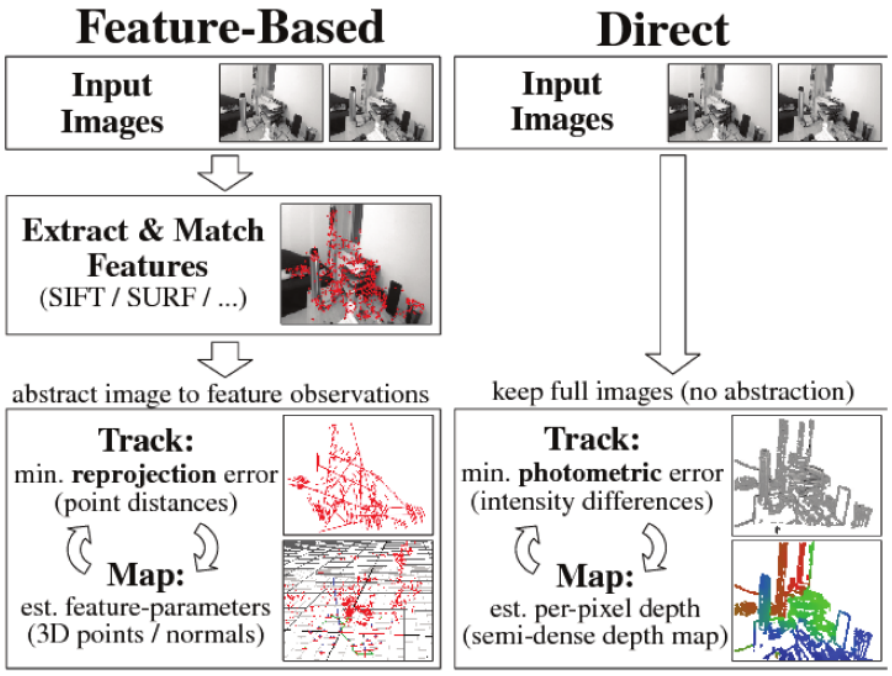


Figure 2.3: Feature-based versus direct methods. Image from [Schops et al., 2014]

The feature-based methods abstract images to features and discard all the order information. The main pipeline of these methods are:

- To acquire the image sequences.

- To detect/extract the features using approaches such as Scale-Invariant Feature Transform (SIFT) or Speed Up Robust Features (SURF).
- To match/track the features using techniques such as RANdom SAmple Consensus (RANSAC)
- To estimate the relative motion between the features (2D/2D, 3D/3D, 3D/2D)
- To optimize the estimated transformation.

These steps may slightly differ depending on the adopted method. For example, the bundle adjustment-based approach is a commonly used feature-based method where the main steps [Sibley et al., 2010, Shade and Newman, 2011] are:

- Image processing: To remove the distortion in lens and to filter the images to allow faster matching of features.
- Image alignment: To make an initial estimate of a 3D rotation using a gradient descent method-based on image intensity.
- Feature match in time: To project the 3D features into the left and the right images and match them using a sum-absolute-difference error metric.
- Initialize new features: To track about 100 to 150 features and to ensure a spatial distribution using a quad-tree.

Results, presented in [Shade and Newman, 2011], have shown that these estimations are robust even under difficult conditions. They are mostly adapted for large scale environments.

The direct methods [Schöps et al., 2014] perform tracking directly on the image intensity (instead of extracting and matching features). This allows to achieve a higher accuracy and robustness, especially in indoor environments where only few features are available. But, this method requires a powerful Graphic Processing Unit (GPU) to run in real time.

According to [Fang and Scherer, 2014, Fang and Zhang, 2015], the VO for RGB-D sensor can be classified within three categories:

- The image-based category [Huang et al., 2011, Kerl et al., 2013b, Endres et al., 2014, Li et al., 2015] that uses RGB-D and depth data. It is mostly adapted when there is a good gray image value or visual features.
- The depth-based category [Wang et al., 2017] that uses point cloud and is commonly used in featureless or dark environment.
- The hybrid category [Zhang et al., 2014] that uses point cloud and RGB-D.

The Fovis presented in [Huang et al., 2011], is a feature-based VO method that provides consistent motion estimation but needs to work at high frequencies for a correct estimation.

The Dense VO method (DVO), introduced in [Steinbrücker et al., 2011], estimates dense VO directly from the RGB-D frame by minimizing the difference between the previous image and the back-projected current RGB-D image. This approach is optimized in [Kerl et al., 2013b] by a probabilistic derivation and the possibility of prior integration of the motion and the sensor noise. It has been extended by adding weight to each pixel and by incorporating a motion prior. Actually, this method is based on the photo-consistency assumption that assumes that if a point is observed by two cameras, it has the same brightness in both images.

Authors in [Zhang et al., 2014] propose the Depth Enhanced Monocular Odometry method (DEMO) to enhance VO from monocular images by the assistance of depth information even if it is sparsely or locally unavailable. According to [Fang and Scherer, 2014, Zhang et al., 2014, Fang and Zhang, 2015], DVO is adapted for environment with relatively dark illumination and DEMO [Zhang et al., 2014] for areas with no sufficient depth information.

A Fast Semi-Direct Monocular Visual Odometry called SVO is proposed in [Forster et al., 2014]. The algorithm operates directly on pixel intensities. The 3D points are estimated using probabilistic mapping method that allows to reduce the outliers (false matching points) and get more reliable points. Results show that the proposed method is robust, and faster than current state-of-the-art methods.

In some works, authors use Structure From Motion (SFM) term as a synonym of VO. Actually, VO is a particular case of SFM [Scaramuzza and

Fraundorfer, 2011]. In fact, SFM is a more general process that treats both 3D problem of camera pose estimation and structure from images set that can even be unordered. They are generally refined with an off-line optimization known as bundle adjustment. The SFM's computation time grows when the image number grows too. Compared to the SFM, VO focuses on 3D sequential pose estimation in real time. In VO, the trajectory estimation optimization is optional.

2.1.2 Simultaneous Localization And Mapping (SLAM)

Overview

The SLAM problem is one of the most important topics in the robotic community. It consists in answering simultaneously two important questions: Where is the robot? And what does the world looks like? Let's consider a robot with a visual sensor mounted on it. It is moving in an a priori unknown environment and is collecting information about relative observations of landmarks. Figure 2.4 shows the evolution of the robot poses and landmarks during a short time of navigation [Durrant-Whyte and Bailey, 2006]. The following sets are then introduced:

- $\mathcal{P}_{0:k} = \{\mathbf{p_0}, \mathbf{p_1}, ..., \mathbf{p}_k\}$ represents the set of poses of robot. Each pose includes the position and the orientation of the robot.

- $\mathcal{U}_{0:k} = \{\mathbf{u_1}, \mathbf{u_2}, ..., \mathbf{u}_k\}$ represents the set of control vectors used to drive the robot from state $l-1$ to state l at time l with $l \in [1..k]$

- $\mathcal{M}_{0:k} = \{\mathbf{m_1}, \mathbf{m_2}, ..., \mathbf{m}_k\}$ represents the set of vectors that defines the states of landmarks. These locations are considered as time invariant.

- $\mathcal{Z}_{0:k,i} = \{\mathbf{z}_{1,i}, \mathbf{u}_{2,i}, ..., \mathbf{u}_{k,i}\}$ represents the set of observations of the landmark i made at times $[0..k]$

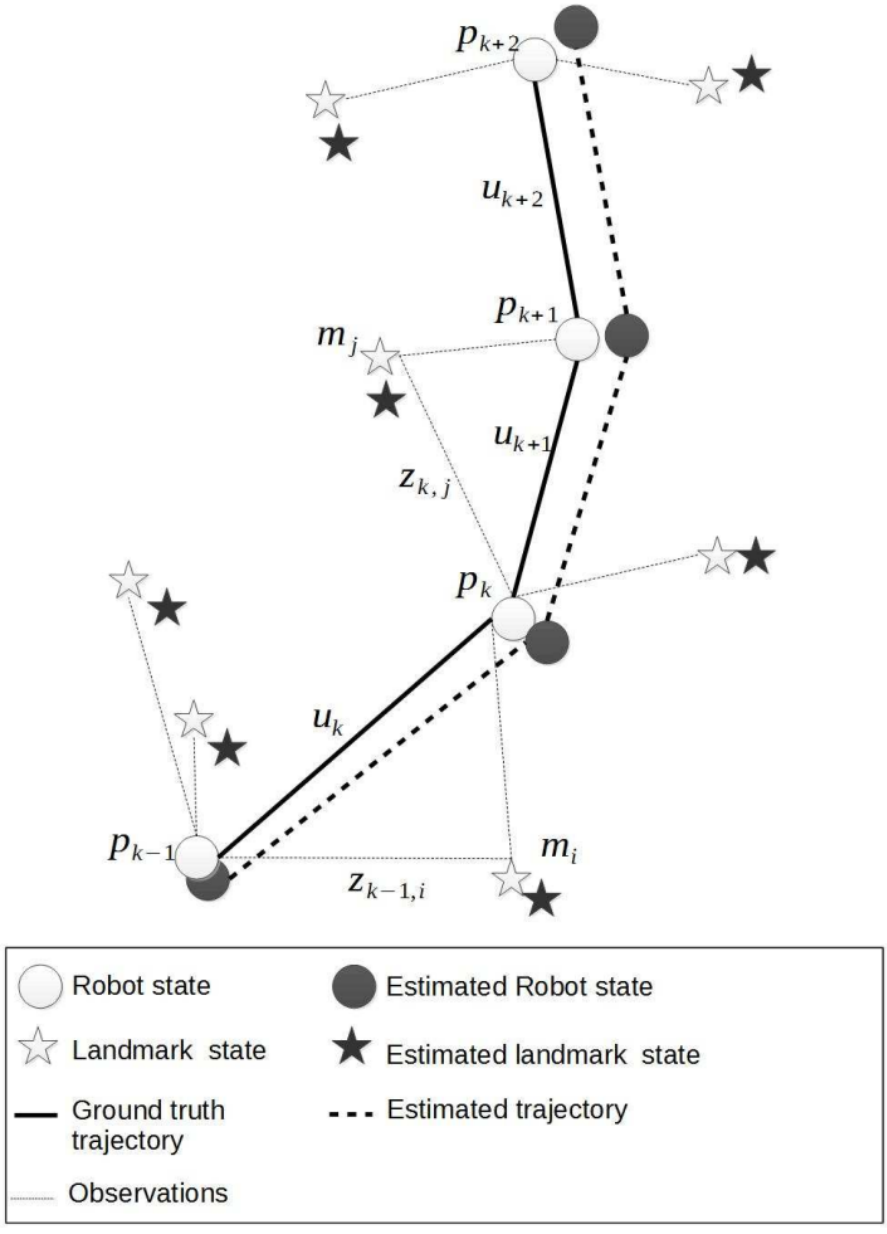


Figure 2.4: SLAM problem formulation.

The formulation of a probabilistic SLAM can be written in a probabilistic form as expressed in Eq.2.3. This probability distribution has to be com-

puted at each time k, knowing the observed landmarks, the control vectors, and the initial robot pose.

$$P(\mathbf{p}_k, \mathbf{m} | \mathcal{Z}_{0:k}, \mathcal{U}_{0:k}, \mathbf{p_0}) \tag{2.3}$$

Generally, the SLAM problem is resolved through a recursive solution that requires a motion model and an observation model. By far, the most commonly used approaches for these model representations are:

- The Extended Kalman Filter (EKF) for an EKF-SLAM solution.
- The Rao-Blackwellised particle filter for a Fast-SLAM solution.

Some other solutions, explained in the next section, are proposed to attempt to solve the SLAM problem [Cadena et al., 2016].

Related work

The VO's main objective is to ensure a local consistency while SLAM aims at a global consistency of the map [Renaudeau et al., 2018] and the trajectory [Mur-Artal and Tardós, 2017a]. Indeed, SLAM is used to obtain a global and consistent estimate of the robot path based on loop closure. It allows the algorithm to apply a global optimization to reduce drift on both the trajectory and the map. Whereas, VO aims at estimating the trajectory pose after pose, and applying optimization after a certain number of poses called windowed optimization. The choice between using VO and Visual SLAM is based on a trade-off between performance and consistency, and simplicity of implementation.

The Parallel Tracking and Mapping (PTAM) proposed in [Klein and Murray, 2007] is a monocular SLAM based on a parallel framework that includes a tracker and a mapper, in order to increase the responsiveness and robustness of the whole system. The tracker enables fast camera localization in real time; whereas keyframe based mapper builds the global map. Authors in [Ta et al., 2013] modified PTAM – originally designed for augmented reality – making it more suitable for robot navigation. Instead of using a motion model, odometry and visual measurements are fused into the framework to deal with the lack of visual features and the lack of motion in the environment. In addition, a loop closer mechanism is performed.

Authors in [Cunningham et al., 2010] use an extending Smoothing And Mapping (SAM) approach consisting on a graphical model approach that

introduces the Constrained Factor Graph (CFG). A Decentralized Data Fusion-SAM (DDF-SAM), that satisfies the DDF requirements while taking into account the benefits of naive approach, is proposed. The framework is composed of three modules:

- The Local Optimizer Module performs the SLAM for one robot in its local environment and produces its local map and condensed local graph.
- The Communication Module shares the previous condensed local graph so that each robot maintains its local graph and a cache of neighboring robots' condensed graphs.
- The Neighborhood Optimizer Module merges the condensed graphs to obtain a neighborhood graph that can be used to build the map.

By applying loop closing process along with DVO method, authors in [Kerl et al., 2013a] propose a SLAM method that applies a global optimization to reduce drift on both the trajectory and the map. Yet, many implementation issues of this SLAM method rise due to versions incompatibility.

Authors in [Forster et al., 2013a] propose a distributed monocular SLAM for a multi- robot system. To determine each robot's individual motion, measurements from an on- board camera and Inertial Measurement Unit (IMU) are combined together. Specific data such as image coordinates, descriptors as features of selected keyframes, and relative pose estimation are streamed to a ground station – called Collaborative Structure From Motion (CSFM) – where a map for each robot is created and merged if there is an overlap among them.

A software architecture is proposed in [Brand et al., 2014] to perform a distributed SLAM. The on-board stereo-vision based mapping system proves its effectiveness in indoor, unstructured outdoor as well as mixed environment.

A novel direct and feature-less Large-Scale Direct monocular SLAM (LSD-SLAM) method is proposed in [Engel et al., 2014]. It performs an accurate pose estimation using direct image alignment along with filtering-based estimation of semi-dense depth maps. A 3D reconstruction of the environment is represented as a pose graph where keyframes are vertices.

The OKVIS SLAM [Leutenegger et al., 2015] proposes a non-linear optimization approach that tightly fuses visual measurements along with readings from an IMU. This allows to significant advantages in quality of performance and computational complexity.

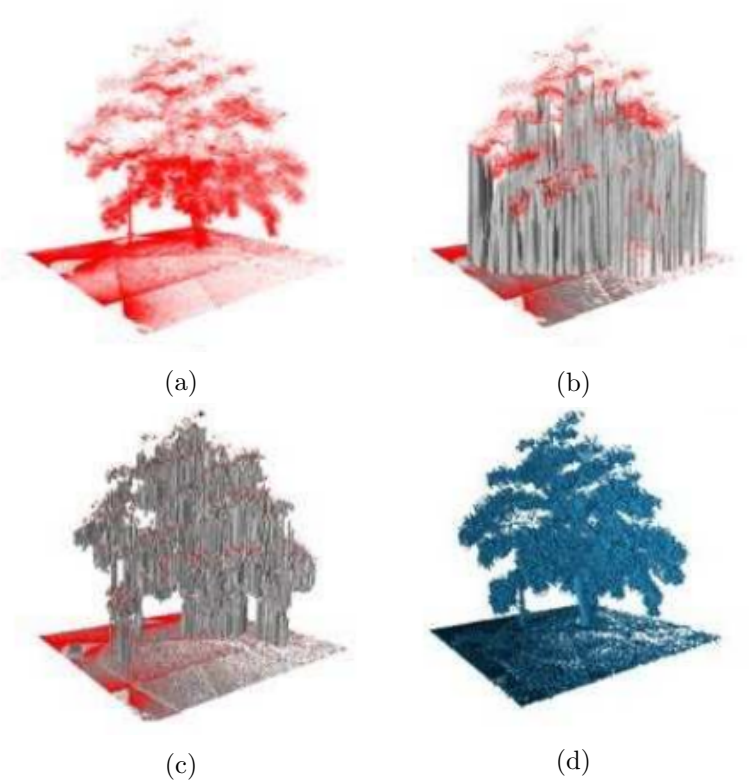
A novel tightly coupled visual-inertial SLAM system is proposed in [Mur-Artal and Tardós, 2017b]. This system is able to be applied to monocular as well as stereo and RGB-D sensors. It performs loop closing to attempt a zero-drift localization in already mapped areas.

Authors in [Mur-Artal and Tardós, 2017a] propose a lightweight RGB-D feature-based SLAM method called ORB-SLAM2. It is adapted for monocular (depth triangulated from different view), stereo and RGB-D sensors. Using the TUM RGB-D data-set [Sturm et al., 2012], in most cases, ORB-SLAM2 performs better than Elastic-Fusion [Whelan et al., 2016], kintinuous [Whelan et al., 2015], DVO SLAM [Kerl et al., 2013a] and RGB- D SLAM [Endres et al., 2014] in terms of Root Mean Square Error (RMSE) translation error.

A new open framework for research in Visual Inertial (VI) mapping and localization – called Maplab – is proposed in [Schneider et al., 2018]. It contains a RObust Visual Inertial Odometry (ROVIO) with Localization Integration (ROVIOLI) and an off-line Maplab-console. ROVIOLI, composed of an on-line Visual-Inertial Odometry (VIO) and a localization front-end [Bloesch et al., 2017], is used for pose estimation and visual-inertial map building. The Maplab-console is used to apply algorithms on map in an off-line batch fashion such as map alignment and merging, VI optimization, loop closure detection, etc. Using EuRoC data-sets for comparison, ROVIOLI outperforms ORB-SLAM2 which itself outperforms in its tern ROVIO in terms of position and orientation RMSE. Nonetheless, ROVIOLI requires a global shutter camera and an IMU to work. It also does not make any use of depth information which makes it not optimal when using a RGB-D camera.

2.2 Metric map representation

The map reflects the environment by representing its model using either topological or metric method. A topological map is a graph data structure composed of vertices that represent the locations in the map, and edges to show the connection/link between them. Whereas, a metric map is a geometric representation of the environment. Figure 2.5 shows an example of different map-structure representations.



(a) (b)

(c) (d)

Figure 2.5: Examples of a map represented in different structures. (a) Point cloud map. (b) Elevation map. (c) Multi-level surface map. (d) Occupancy grid map based on an octree. Image from [Hornung et al., 2013].

Point cloud representation

The point cloud is one of the simplest metric map representation of the environment (see Figure 2.5a). The points gathered from a range sensor are

transformed into a global coordinate frame. But, this representation is not adapted for dynamic environment and does not cope with sensor noise.

Occupancy grid representation

The occupancy grid map is a discretization of the environment in regularly sized 2D squares – called cells – or 3D cubic volumes – called voxels – (See Figure 2.5d). The occupancy grid map is based on a hierarchical data structure – called Octree – which represents a 3D space that is recursively subdivided until attending a minimum voxel size – called resolution –(See Figure 2.6). By increasing the resolution, the map becomes less coarser.

Figure 2.6: Examples of an occupancy grid map with resolutions of $0.08m$, $0.64m$, and $1.28m$, respectively. Image from [Hornung et al., 2013].

The occupancy grid representation introduces several advantages [Hornung et al., 2013] such as:

- Arbitrary environment representation without prior assumptions.
- Fast data access.
- Flexibility in extending and combining different maps with different resolutions.
- Updatability in adding new informations or sensor readings.
- Compact memory storage.
- Obstacle distinction for safe robot navigation.

Using the sensor measurements, the cells of the 2D (or the voxels of the 3D) occupancy grid map are labeled as unknown, free or occupied as shown in Figure 2.7.

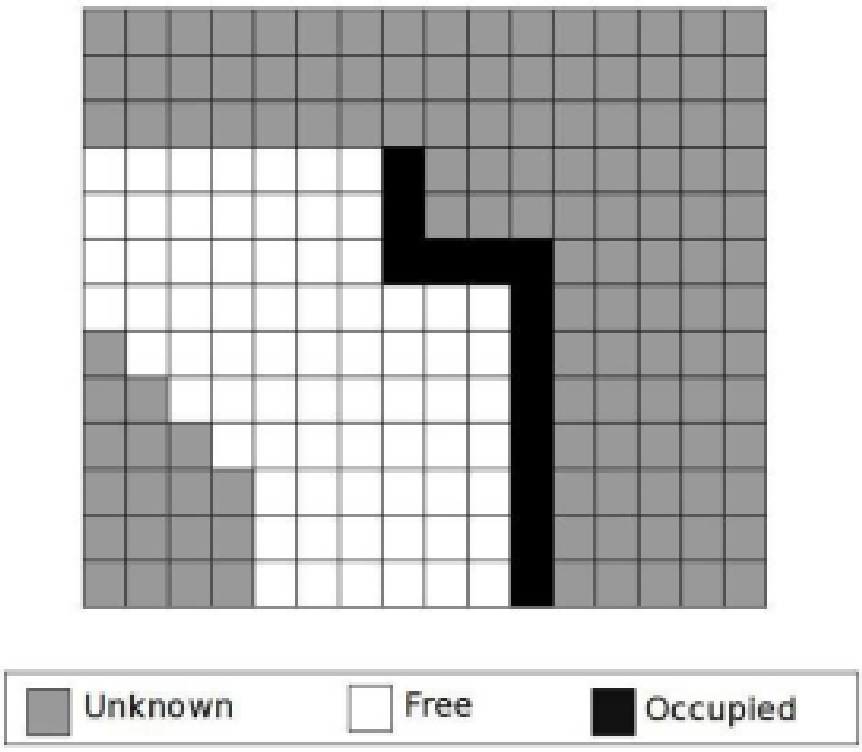


Figure 2.7: 2D occupancy grid example.

Occupancy grid matching We propose and implement a simple map matching process in order to evaluate and illustrate the global map during the mission. Suppose that we have two maps M 1 and M 2 (See Figure 2.8) with the following assumptions:

- Belonging to the same global frame.

- Having the same resolution.

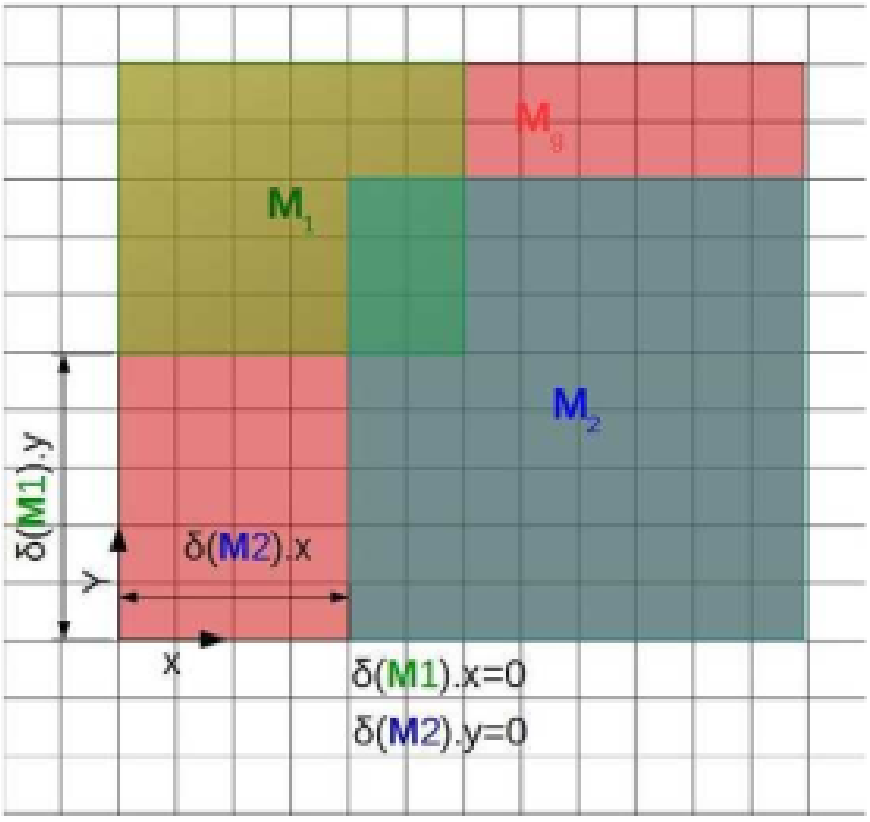


Figure 2.8: Map matching of $\mathbf{M_1}$ and $\mathbf{M_2}$.

Each map is composed of:

- *Header*: It contains the sequence ID that is consecutively increasing, the stamp that defines the seconds and nanoseconds, and the frame this data is associated with.

- *Meta data*: It contains the time load map, which is the time at which the map hasd been loaded, the map resolution ($\mathbf{M}_i.resolution$) that defines the metric size of the cells, the map width ($\mathbf{M}_i.width$) and height ($\mathbf{M}_i.heigth$) in number of cells, and the origin of the map ($\mathbf{M}_i.origin$).

- *Data*: It contains the probability of occupancy of the cells in a row-major order.

Algorithm 2.1 describes the pipeline to match M 1 and M 2 where the steps can be summarized as follows:

- Step 1 to 5: The meta data of the fused map are defined.
- Steps 6 to 7: Each cell in the fused map ($grid(\mathbf{M}_g)$) is initialized to $\mathbf{l}_u$ (cell labeled as $UNKNOWN$).
- Each unknown cell of the grid ($grid(\mathbf{M}_g) = \mathbf{l}_u$) is filled in using the value of either $grid(\mathbf{M}_1)$ or $grid(\mathbf{M}_2)$.
- Return the fused map $\mathbf{M}_g$

Algorithm 1 Map matching.

Input: Maps $\mathbf{M}_i$ with $i \in n_c$
Output: Fused map $\mathbf{M}_g$

1: $\mathbf{M}_g.origin = argmin_{i \in n_c}(\mathbf{M}_i.origin)$;
2: $\delta(\mathbf{M}_i).x = \frac{(\mathbf{M}_g.origin.x - \mathbf{M}_i.origin.x)}{\mathbf{M}_i.resolution}$;
3: $\delta(\mathbf{M}_i).y = \frac{(\mathbf{M}_g.origin.y - \mathbf{M}_i.origin.y)}{\mathbf{M}_i.resolution}$;
4: $\mathbf{M}_g.width = argmax_{i \in n_c}(\mathbf{M}_i.width + \delta(\mathbf{M}_i).x)$;
5: $\mathbf{M}_g.height = argmax_{i \in n_c}(\mathbf{M}_i.height + \delta(\mathbf{M}_i).x)$;
6: **for** all grids in $\mathbf{M}_g$ **do**
7: $grid(\mathbf{M}_g) = \mathbf{l}_u$
8: **for** $i = 0; i < n_c; i++$ **do**
9: **for** $j = 0; j < \mathbf{M}_i.width; j++$ **do**
10: **for** $k = 0; k < \mathbf{M}_i.length; k++$ **do**
11: **if** $grid(\mathbf{M}_i)\,[j + k * \mathbf{M}_i.width]! = \mathbf{l}_u$ **then**
12: $grid(\mathbf{M}_g)\,[(j - \delta(\mathbf{M}_i).x) + (k + \delta(\mathbf{M}_i).y) * \mathbf{M}_g.width] = grid\,[j + k * \mathbf{M}_i.width]$
13: **end if**
14: **end for**
15: **end for**
16: **end for**
17: **end for**
18: **return** $\mathbf{M}_g$.

The proposed algorithm is a basic map matching approach used to merge two maps. It can be adapted to merge more than two maps at a time. This algorithm deals only with simple rectangular shape maps (This can be improved in our future work.).

2.3 Monocular SLAM

2.3.1 Monocular sensor

Monocular sensors are those where the only sensing device is a single camera. This sensor provides a set of images taken at discrete times k = [0..n] (See Eq.2.4).

$$I_{0:n} = \{I_0, \ldots, I_n\} \tag{2.4}$$

Each image I_k is a set of pixels – also called color components – that are stored in a $h \times w \times 3$ matrix where the height h and width w of the image correspond to the matrix's number of rows and columns, respectively (See Figure 2.9). The element of (u, v) index in the matrix represents the pixels' RGB value.

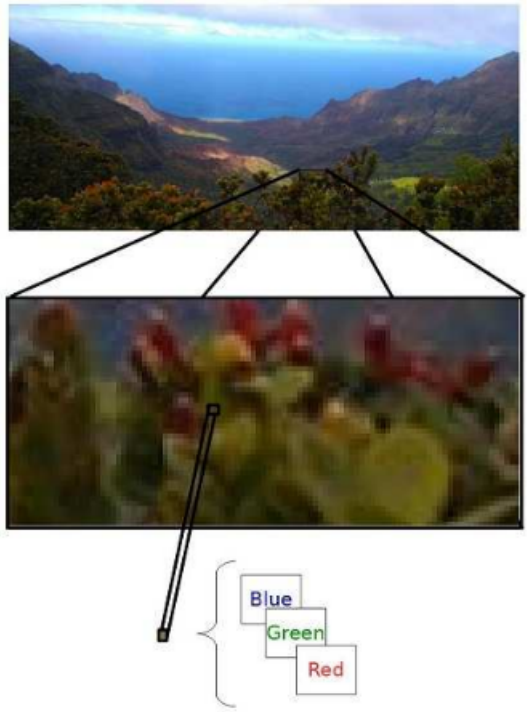


Figure 2.9: Image[2] pixels representation

[2]This image was taken in Kauai Island, Hawaii.

2.3.2 Visual SLAM

Monocular sensors do not provide neither depth measurements (RGB-D camera example) nor two images of the same scene at the same time (stereo camera example) to compute these depth measurements. This leads to the inherent problem of scale ambiguity which consists in computing the scale factor. Despite the great progress in problems related to monocular SLAM, the main challenge is still the metric scale estimation. The scale[3] defines the relationship between sizes of the world and the created map. As a single camera cannot compute the scale factor, another sensor is added to be able to measure metrics from the environment. In the literature [Forster et al., 2015, Concha et al., 2016, Mur-Artal and Tardós, 2017b, Spaenlehauer et al., 2017], this additional sensor is mostly chosen to be an IMU due to its light weight, small size, and easiness to be mounted on an UAV.

An architecture for visual inertial SLAM system is proposed in Figure **??**. It uses a monocular camera and an IMU as sensors. Similarly to the PTAM approach [Klein and Murray, 2007], two threads are used: A tracker for a fast response to changes in the environment and a mapper to build a high quality map of the environment. The only difference is that the proposed approach uses odometry measurements instead of a motion model.

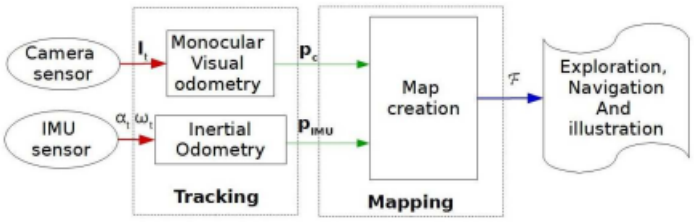


Figure 2.10: Visual inertial SLAM outline. $\mathbf{I}_t$ represent the images input sets from the camera. α_t and ω_t are, respectively, the acceleration and angular measurements from the IMU sensor. $\mathbf{p}_c$ and $\mathbf{p}_IMU$ are the estimated states from the camera and the IMU sensors, respectively. $\mathcal{F}$ is the generated 2D factor graph.

The monocular visual odometry is computed by detecting and tracking features of images coming from an extrinsic camera sensor. Additional inertial odometry is computed using the acceleration (α_t) and the angular

[3]Source:`https://www.kudan.eu/kudan-news/scale-simultaneous-localisation-mapping/`

velocities (ω_t) measurements from an intrinsic IMU sensor. Both are then fused in the mapper thread where a factor graph is created. An optimized pose estimation is then generated from this graph. For the map creation, a sparse nonlinear incremental optimization approach – called iSAM2 – [Kaess et al., 2012] is used. It allows to provide updated pose estimations when new measurements are available.

The proposed architecture allows not only to overcome the scale ambiguity but also to reduce the accumulated drift from the estimated trajectory.

Problem formulation

The proposed system is a graph-based SLAM problem that can be formulated in a factor graph. The graph is a factorization of a function $\mathbf{F}(\theta)$ described in Eq.2.5.

$$\mathbf{F}(\theta) = \prod_i \mathbf{F}(\theta_i), \tag{2.5}$$

with θ_i is a variable node i.

The purpose is to find the variable θ^* that maximizes the function $\mathbf{F}(\theta)$:

$$\theta^* = argmax_\theta \mathbf{F}(\theta) \tag{2.6}$$

The factor graph, presented in Figure 2.11 is composed of:

- $\mathcal{P}_{0:n} = \{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n\}$: The set of unknown poses.
- $\mathcal{M}_{0:m} = \{\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_m\}$: The set of landmarks.
- $\mathcal{Z}_{i=cste,j=0:m} = \{\mathbf{z}_{i0}, \mathbf{z}_{i1}, \ldots, \mathbf{z}_{im}\}$: The set of visual measurements.
- $\mathcal{B}_{0:n-1} = \{\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}\}$: The set of odometry measurements. $\mathbf{b}_i$ is the odometry between the pose $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$.

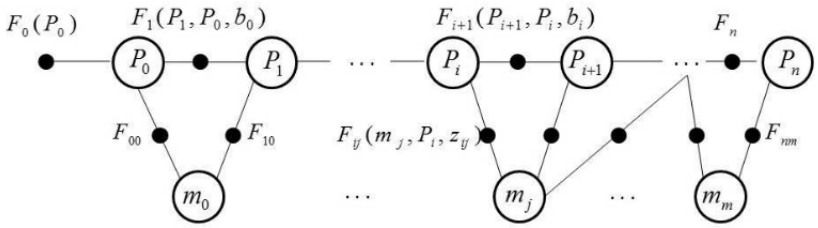


Figure 2.11: Factor graph for monocular SLAM. $\mathbf{F}_0(\mathbf{p}_0)$ and $\mathbf{m}_i$ are the variable nodes. $\mathbf{F}(0)$ is the prior density factor. $\mathbf{F}_{ij}(\mathbf{m}_j, \mathbf{p}_i, \mathbf{z}_{ij})$ is the odometry factor between the pose $\mathbf{p}_i$ and $\mathbf{F}_{ij}(\mathbf{m}_j, \mathbf{p}_i, \mathbf{z}_{ij})$ is the measurement likelihood model between the pose $\mathbf{p}_i$ and its landmark $\mathbf{m}_i$.

This bipartite factor graph is composed of:

- Variable node:
  - Camera pose $\mathbf{p}_i$.
  - Landmarks $\mathbf{m}_j$.
- Factor nodes:
  - Prior densities on the variable nodes $\mathbf{F}_0(\mathbf{p}_0) = p(\mathbf{p}_0)$.
  - The motion models between two camera poses $\mathbf{F}_i(\mathbf{p}_{i+1}, \mathbf{p}_i, \mathbf{b}_i) = p(\mathbf{p}_{i+1}|\mathbf{p}_i, \mathbf{b}_i)$ given the odometry measurement $\mathbf{b}_i$.
  - The measurement likelihood models $\mathbf{F}_{ij}(\mathbf{m}_j, \mathbf{p}_i, \mathbf{z}_{ij}) = p(\mathbf{m}_j|\mathbf{p}_j, \mathbf{z}_{ij})$ between the pose $\mathbf{p}_i$ and the landmark $\mathbf{m}_j$ given the visual measurement $\mathbf{z}_{ij}$.

Incremental Smoothing and Mapping 2 (iSAM2)

To resolve the factor graph and estimate the poses, the iSAM2 approach [Kaess et al., 2012] is used. The estimation problem is based on three graphical models (See Figure 2.12):

- Explicit factor graph.
- Implicit chordal Bayes net.
- Bayes tree.

Therefore, the iSAM2 algorithm can incrementally obtain an estimate of unknown variables (such as UAV poses and landmarks) given a set of non linear factors (such as odometry) to finally construct the UAV's map.

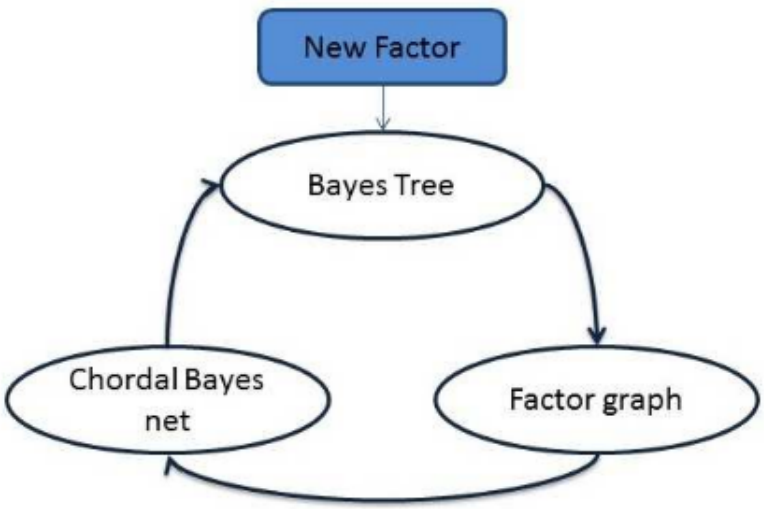


Figure 2.12: iSAM2 graphical models.

When a new factor is taken into account, the affected part of the Bayes Tree is isolated. From it, data are formulated as a factor graph and an associated Jacobian matrix. Then, the factor graph is transformed into a chordal Bayes net and a square root information matrix using a specific variable order elimination. The last one eliminated is called the root. Finally, based on the clique structure in the chordal Bayes net, a Bayes tree is formed with the square root information matrix. Based on these models, the incremental resolution and update of the iSAM2 are resumed in five important steps presented in Figure 2.13.

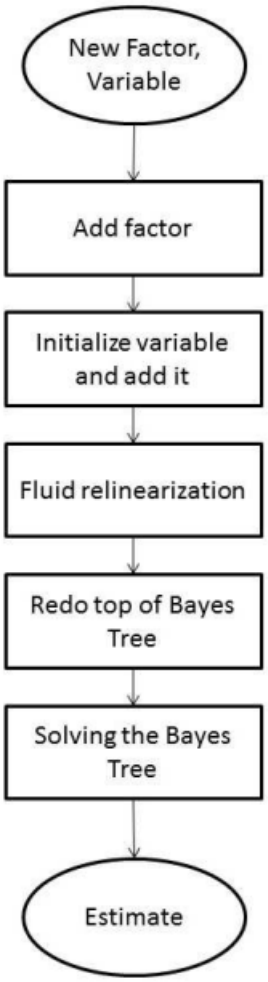


Figure 2.13: The iSAM2 steps.

Taking into account the available new factors, variables are initialized and added to the variable nodes. Thereafter, the process performs a fluid relinearization for marked variables in order to track the validity of the linearization point for each variable when needed. These marked variables are chosen according to a threshold between its current estimate and the linearization point. Then, the Bayes tree is partially updated starting from the cliques involved by the marked variables and the variables affected by new factors, up to the top of the tree. Using the obtained new Bayes tree, an update of the current estimate is performed.

2.3.3 Results and discussions

The SLAM algorithm was implement with the Georgia Tech Smoothing And Mapping toolbox[4] (GTSAM) using the factor graph implementation. The experimental code is written in MATLAB[5] and includes MEX functions[6] of the c++ library. Tests were performed on a 2.50GHz i5 Linux machine. To validate the SLAM algorithm, the measurements were simulated with raw data from the Kitti vision data-set Geiger et al., 2013. These data contain IMU measurements used for dead reckoning, and images used for the visual features. To compute the inter frame visual odometry, the libviso2 library[7] is used. Hence, given the set of non linear factors including IMU and visual odometry measurements, the iSAM2 estimates and optimizes poses and landmarks.

To see the improvement performed with the proposed approach, the estimated trajectory from iSAM2 is compared to those from IMU, libviso2 and GPS, in Figure 2.14.

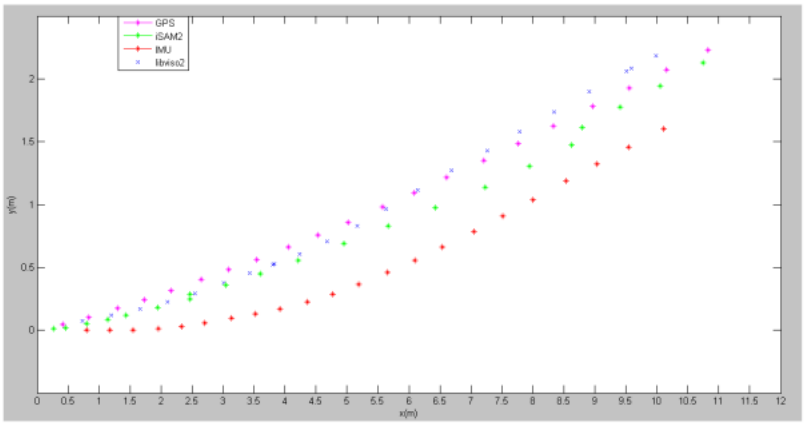


Figure 2.14: Trajectory results (2D projection) using IMU, libviso2, iSAM2 and GPS.

[4]Source: `https://borg.cc.gatech.edu/index.html`
[5]Source: `https://fr.mathworks.com/`
[6]Source: `https://fr.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html`
[7]Source: `http://www.cvlibs.net/software/libviso/`

The GPS trajectory is obtained using Eq.2.7:

$$\begin{aligned} x &= R * \cos(latitude) * \cos(longtitude), \\ y &= R * \cos(latitude) * \sin(longtitude), \\ z &= R * \sin(latitude), \end{aligned} \tag{2.7}$$

with R the radius of the earth.

Results show that combining the IMU and the libviso2 trajectory using the iSAM2 helps to improve the poses estimation, and to reduce the IMU drift compared to the ground truth. Consequently, by obtaining a graph to exchange data instead of images, the data size is considerably reduced. Indeed, theoretically, sending images require about 1.843 Mbps (assuming that an image is encoded in 24 bits); whereas sending poses and landmarks needs about 0.323 Mbps (assuming that an IEEE standard encodes reals under 32 bits). Hence, thanks to the graph-based SLAM, we manage to reduce data size by about 5 times.

2.4 RGB-D SLAM

The RGB-D SLAM uses an RGB-D camera as a visual sensor to sense the environment.This sensor avoids the scale estimation problem thanks to the depth information.

2.4.1 RGB-D sensor

The RGB-D camera is composed of a digital camera for colored images $\mathbf{C} - t$, and another device for the depth image $\mathbf{D}_t$. The colored image $\mathbf{C}_t$ is a 3D matrix encoded in the RGB space where each color represents a dimension with a number of raws and columns corresponding to the image resolution (See section 1.3.1 on monocular sensor). The depth image $\mathbf{D}_t$ is a 2D matrix where $\mathbf{D}_t(u, v)$ is the depth of the pixel of column u and row v. The depth measurements can be computed by different technologies (e.g. camera, infrared) where the RGB-D sensor can be classified as follows:

- *Active*: It is composed of a camera and a projector. The projector emits known patterns that are compared with the camera to get the correspondences. The depth is computed from these correspondences and the known transformation between the camera and the projector.

- *Passive*: It represents stereo sensors composed of two RGB cameras. The depth is computed by knowing the transformation between them.

RGB-D sensor Comparison
Different types of RGB-D camera exist. One of the most famous is the kinect[8] camera (See Figure 2.15a). Other relatively new technologies are available such as DUO MLX[9] (See Figure 2.15b), ZED[10] stereo (See Figure 2.15c), and Intel RealSense ZR300[11] (See Figure 2.15d).

(a) Kinect camera.

(b) DUO MLX camera.

(c) ZED camera

(d) ZR300 camera.

Figure 2.15: Examples of RGB-D sensors.

A brief comparison is summarized in Table 2.1. The ZED camera have the higher resolution and range compared to DUO and Kinect. However,

[8]Source: `https://msdn.microsoft.com/en-us/library/hh438998.aspx`
[9]Source: `https://duo3d.com/product/duo-minilx-lv1`
[10]Source: `https://www.stereolabs.com/zed/specs/`
[11]Source: `https://software.intel.com/en-us/realsense/zr300`

this camera requires a powerful processor to work properly. DUO MLX is a lightweight camera with a good resolution, small dimensions but a small range and a relatively high price. The ZR300 has a small size and a good resolution but it is more expensive than the kinect. Despite its bigger dimension and weight, the kinect is the cheaper and the easiest to use. Taking into account the UAVs' mission requirements and purpose, we make the choice to use the kinect. Yet, the SLAM algorithm is not restricted to a defined type of a RGB-D sensor.

Table 2.1: RGB-D sensors comparison

Characteristic	**kinect xbox 360**	**DUO MLX (integrated IMU)**	**ZED**	**ZR300**
Output Resolution	640 x 480 x 24 bpp 4:3 RGB @30fps 640 x 480 x 16 bpp 4:3 YUV @15fps	752 x 480 @56fps	2x (1920 x 1080) @30fps	2 x VGA @60fps
Depth Range (m)	0.8 to 4.0	0.23 to 2.5	1 to 15	0.4 to 2.8
Baseline (mm)	75	30	120	20
Dimensions (mm)	279 x 50,8 x 25	52 x 24 x 13	175 x 30 x 33	9.5 x 101.56 x 3.8
Weight (grams)	750	12.5	159	-
System requirements	1.9 GHz CPU 4GB RAM	Modern Processor Intel i5/i7, AMD or ARM Minumum 4GB RAM	Dual-core 2.4GHz or faster processor Minumum 4GB RAM Nvidia GPU with compute capability >3.0	Intel Joule compute module with Ubuntu 16.04
price ($)	33	595	449	129