Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Parallel Tracking and Mapping algorithms for an Event Based Camera

MEMÒRIA

Autor: Joaquim Ortiz de Haro
Director: Joan Solà Ortega
Ponent: Alberto Sanfeliu Cortés

Convocatòria: 06/2017

Institut de Robòtica i Informàtica Industrial

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

An event camera has independent pixels that sends information, called “events” when they perceive a local change of brightness. The information is transmitted asynchronously exactly when the change occurs, with a microsecond resolution, making this sensor suitable for fast robotics applications.

We present two new tracking and mapping algorithms, designed to work in parallel to estimate the 6 DOF (Degrees Of Freedom) trajectory and the structure of the scene in line based environments.

The tracking thread is based on a Landmark Based map and an asynchronous EKF (Extended Kalman Filter) filter to estimate event per event the state of the camera unlocking the true potential of the camera.

Inside the mapping thread, a line extraction algorithm has been designed to find 3D segments in the Point cloud, computed using event – ray tracing into a discretized world.

Both algorithms have been built from scratch, and at this moment, only tested independently in simulation.

We have obtained very good results on three synthetic self-made datasets.

Some pieces of the complete Parallel Tracking and Mapping system are still missing. The current good work and results encourages to improve and finish the algorithm to achieve the implementation on the real event based camera.

Acknowledgements

I would like to thank my thesis supervisor Dr. Joan Solà, for guiding me in this exciting journey. A journey without a clear map to localize ourselves, without a known trajectory to build map to know where to go.

Oh! That is like a SLAM problem … just a little bit harder, because this time we did not even truly know our sensor. Let's begin the story …

Before we start, I would also like to thank all the members of the Mobile Robotics Group of IRI (Institut de Robòtica i Informàtica Industrial) for useful discussions and good moments, and to Prof. Juan Andrade, IRI director, that together with my supervisor Dr. Joan Solà, have given me the opportunity to work in this challenging problem.

Content

Abstract 1

Acknowledgements 3

1. **Introduction** 7

1.1 Simultaneous localization and mapping (SLAM) 7

1.2 Monocular SLAM 15

1.3 Dynamic Vision Sensor (Event Based Cameras) 19

1.4 Literature Review: Towards Slam with Event Based Cameras 21

2. **Algorithm** 25

2.1 System Overview 25

2.2 Tracking 29

2.3 Mapping 37

3. **Results** 51

3.1 Event Based Simulator in MATLAB 51

3.2 Tracking and Mapping Results 55

3.3 Discussion and future work 69

4. **Budget, impact on society and project planning** 71

Conclusion 73

References 75

1. Introduction

1.1 Simultaneous localization and mapping (SLAM)

Problem statement

Simultaneous localization and mapping (SLAM) is the problem of estimating in real time the structure of the surrounding world (the map) while simultaneously getting localized in it. The big challenge in SLAM is that, while a good map is needed for localization a good pose estimate is also needed for mapping.

The two main elements in SLAM are the robot $\mathcal{R}$ (with a sensor) and the map $\mathcal{M}$ (environment). There are two main basic operations, repeated iteratively at each time step: move and look.

A. The robot **moves**, and due to errors and noise $\boldsymbol{n}$, the uncertainty of the robot localization **increases**. The motion model $f(\cdot)$ is the function that links two consecutive robot states: $\mathcal{R}_k$ and $\mathcal{R}_{k-1}$. It can include the control input vector $\mathbf{u}_k$.

$$\mathcal{R}_k = f(\mathcal{R}_{k-1}, \mathbf{u}_k, \mathbf{n}_k)$$

$$\mathbf{n}_k \sim \mathcal{N}(0, \mathbf{Q})$$

$\mathbf{Q}$ is the covariance matrix of the motion noise $\mathbf{n}$

B. The robot **looks** using its sensor, and sees interesting features in the environment called Landmarks $\mathcal{L}$. The observation of these Landmarks are the measures $\mathbf{y}$

   - If the Landmark is new it is added to the map $\mathcal{M}$. Its location is uncertain because the sensors have noise and error. It is also affected by the robot position uncertainty. The inverse observation model $g(\cdot)$ is the function that determines the new landmark positions $\mathcal{L}$ based on the robot current position $\mathcal{R}$ and the sensor data $\mathbf{y}$.

$$\mathcal{L} = g(\mathcal{R}, \mathbf{y})$$

   - If the landmark is already known (previously mapped), it is used to correct both: the robots position and position of the landmarks. Therefore, localization and landmarks uncertainties decrease. The observation model $h(\cdot)$ is the function that predicts a value of the measurement from the estimated robot and landmarks state. The predicted measure is compared

to the measure (from the sensor) to correct the robot and landmark position estimates.

$$\mathbf{y} = h(\mathcal{R}, \mathcal{L}) + \mathbf{v}$$

$$\mathbf{v} \sim \mathcal{N}(0, \mathbf{R})$$

v and **R** are respectively, the measurement noise and its covariance matrix.

The robot state $\mathcal{R}$ and the landmarks $\mathcal{L}$ are modelled as gaussian variables. The goal of SLAM is to use the measurements received by the robot's sensor to estimate its trajectory and the position of this landmarks. Two different methods to solve this problem are briefly introduced in this section: EKF and Graph Slam

Note: the following introductory material is based on two documents [1], [2], from Joan Solà, director of this thesis. The original document It lecture is highly recommended for unexperienced readers.

EKF (Extended Kalman Filter)

An EKF is an iterative filter with two steps: Prediction and correction, that correspond, respectively, to the two-basic main basic operations in SLAM: move and look. A whole iteration is executed after a new measure is received.

It is based on the linearization of the motion and observation models around the current state estimate to propagate the uncertainty to find the optimal estimation.

The SLAM state vector **x** is a large vector containing the robot state $\mathcal{R}$ and a map $\mathcal{M}$.
This map is a set of Landmarks $\mathcal{M} = \{ \mathcal{L}_1 , \dots , \mathcal{L}_n \}$ where $\mathcal{L}_i$ defines the position of the landmark i.

The state **x** is modelled as a gaussian variable, using the mean $\bar{\mathbf{x}}$ and the covariance matrixes **P**

$$\mathbf{x} = \begin{bmatrix} \mathcal{R} \\ \mathcal{M} \end{bmatrix} = \begin{bmatrix} \mathcal{R} \\ \mathcal{L}_1 \\ \vdots \\ \mathcal{L}_n \end{bmatrix} \qquad \mathbf{x} \sim \mathcal{N}(\bar{\mathbf{x}}, \mathbf{P})$$

$$\bar{\mathbf{x}} = \begin{bmatrix} \bar{\mathcal{R}} \\ \bar{\mathcal{M}} \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{M}} \\ \mathbf{P}_{\mathcal{M}\mathcal{R}} & \mathbf{P}_{\mathcal{M}\mathcal{M}} \end{bmatrix}$$

The goal of EKF is to use the received information of the sensor to update the state estimate $\{ \bar{\mathbf{x}} , \mathbf{P} \}$ each iteration.

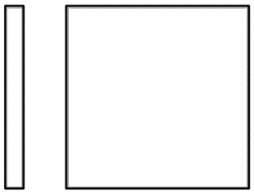


Figure 1.1 The state is represented by its mean $\bar{\mathbf{x}}$ (column vector) and its covariance matrix P square matrix. Source: [1] by Joan Solà. Reproduced with author's permission.

Step 1: Robot Motion - Prediction

The Robot has moved from its last position. The goal of the first EKF step is to predict the new position of the robot. A motion model is used to update the state $\mathbf{x}$ of the system. Only the robot mean state $\bar{\mathcal{R}}$ and all its related covariances are updated.

Motion Model

$$\mathcal{R}_{\mathrm{k}} = \mathrm{f}(\mathcal{R}_{\mathrm{k}-1}, \mathbf{u}_{\mathrm{k}}, \mathbf{n}_{\boldsymbol{k}}) \qquad \mathbf{n}_{\mathbf{k}} \sim \mathcal{N}(0, \mathbf{Q})$$

EKF Predicction

$$\bar{\mathbf{x}} \leftarrow \mathrm{f}(\bar{\mathbf{x}}, \mathbf{u}, 0)$$

$$\mathbf{P} \leftarrow \mathbf{F}_{\mathbf{x}}\mathbf{P}\mathbf{F}_{\mathbf{x}}^{\mathrm{T}} + \mathbf{F}_{\mathbf{n}}\,\mathbf{Q}\mathbf{F}_{\mathbf{x}}^{\mathrm{T}}$$

$$\mathbf{F}_{\mathbf{x}} = \left.\frac{\partial \mathrm{f}}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}},\mathbf{u},0} \qquad \mathbf{F}_{\mathbf{n}} = \left.\frac{\partial \mathrm{f}}{\partial \mathbf{n}}\right|_{\bar{\mathbf{x}},\mathbf{u},0}$$

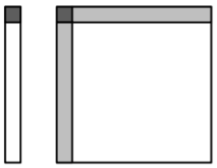


Figure 1.2 Updated parts of the state $\{\bar{\mathbf{x}}, \mathbf{P}\}$ after robot motion. The updated parts, in gray, correspond to the robot's state mean $\bar{\mathcal{R}}$ and covariance $\mathbf{P}_{\mathcal{R}\mathcal{R}}$ (dark gray), and the cross-variances $\mathbf{P}_{\mathcal{R}\mathcal{M}}$ and $\mathbf{P}_{\mathcal{M}\mathcal{R}}$ between the robot and the rest of the map (light gray). Source: [1] by Joan Solà. Reproduced with author's permission.

Step 2: Landmark Observation – Correction

The robot has seen a known Landmark of the map (already in the state vector). The goal of the correction step is to compare the real measure $\mathbf{y}$ with the expected measure $\mathrm{h}(\bar{\mathbf{x}})$, to improve (correct) the state estimate.

The real measure $\mathbf{y}$ is directly the data from the sensor. The observation model $h(\mathcal{R}, \mathcal{L})$ outputs the expected measure given a state. The difference is called innovation $\bar{\mathbf{z}}$

$$\mathbf{y} = h(\mathcal{R}, \mathcal{L}) + \mathbf{v} \qquad \mathbf{v} \sim \mathcal{N}(0, \mathbf{R})$$

$$\bar{\mathbf{z}} = \mathbf{y} - h(\bar{\mathbf{x}})$$

$$\mathbf{Z} = \mathbf{H}_{\mathbf{x}}\mathbf{P}\mathbf{H}_{\mathbf{x}}^{\mathrm{T}} + \mathbf{R}$$

$$\mathbf{H}_{\mathbf{x}} = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}}$$

The innovation $\bar{\mathbf{z}}$ with its associated covariance $\mathbf{Z}$ is an error in the measurement space, and it is used to correct the state estimate.

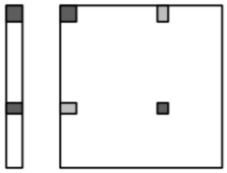


Figure 1.3 The computation of the innovation involves (in dark gray) the robot state $\mathcal{R}$, the concerned landmark state $\mathcal{L}_1$ and their covariances $\mathbf{P}_{\mathcal{R}\mathcal{R}}$ and $\mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}$, and (in light gray) their cross-variances $\mathbf{P}_{\mathcal{R}\mathcal{L}_i}$ and $\mathbf{P}_{\mathcal{L}_i\mathcal{R}}$. Source: [1] by Joan Solà. Reproduced with author's permission.

$$\mathbf{K} = \mathbf{P}\mathbf{H}_{\mathbf{x}}^{\mathbf{T}}\mathbf{Z}^{-\mathbf{1}}$$

$$\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{K}\bar{\mathbf{z}}$$

$$\mathbf{P} \leftarrow \mathbf{P} - \mathbf{K}\mathbf{Z}\mathbf{K}^{\mathbf{T}}$$

K is the Kalman Gain, a matrix that, considering the innovation and current estate covariances, finds the new optimal state estimate $\bar{\mathbf{x}}$**,** which has the minimum uncertainty **P**.

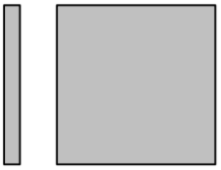


Figure 1.4 In the correction step, all the state $\{\bar{\mathbf{x}}, \mathbf{P}\}$ is updated because the Kalman gain matrix K affects the full state. Source: [1] by Joan Solà. Reproduced with author's permission.

Landmark initialization for full observations

When a robot discovers a new landmark, this new landmark is incorporated to the state vector **x** with the inverse observation model.

$$\mathcal{L}_{n+1} = g(\mathcal{R}, \mathbf{y})$$

The landmarks mean is computed from the expected robot state $\bar{\mathcal{R}}$ and the measure $\mathbf{y}$.

Both the robot and the measurement uncertainties are propagated to the new landmark covariances.

$$\bar{\mathcal{L}}_{n+1} = g(\bar{\mathcal{R}}, \mathbf{y})$$

$$\mathbf{P}_{\mathcal{L}\mathcal{L}} = \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathcal{R}}\mathbf{G}_{\mathcal{R}}^{\mathrm{T}} + \mathbf{G}_{\mathbf{y}}\mathbf{R}\mathbf{G}_{\mathbf{y}}^{\mathrm{T}}$$

$$\mathbf{P}_{\mathcal{L}\mathbf{x}} = \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathbf{x}}$$

$$\mathbf{G}_{\mathcal{R}} = \left.\frac{\partial g}{\partial \mathcal{R}}\right|_{\bar{\mathcal{R}},\mathbf{y}} \quad \mathbf{G}_{\mathbf{y}} = \left.\frac{\partial g}{\partial \mathbf{y}}\right|_{\bar{\mathcal{R}},\mathbf{y}}$$

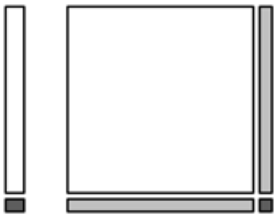


Figure 1.5 New parts added to the state $\{\bar{\mathbf{x}}, \mathbf{P}\}$ after landmark initialization. Landmark's mean and covariance (dark gray), and the cross-variances between the landmark and the rest of the map (light gray). Source: [1] by Joan Solà. Reproduced with author's permission.

Data Association

Another big challenge in Slam is the data association problem, that consists on matching the measures with the landmarks. Usually, the sensor provides an observation of a Landmark without directly identifying from which landmark it is. The Slam system needs an internal method or process to match them before doing the EKF correction step.

GRAPH SLAM

The SLAM problem can be represented by a dynamic Bayes Network, where the probabilities of the measures $\mathbf{y}_t$ depends on the landmark $\mathbf{l}_j$ and the robot position $\mathbf{r}_i$ and each robot poses $\mathbf{r}_i$ depend on the previous pose $\mathbf{r}_{i-1}$ and the control input $\mathbf{u}_i$.

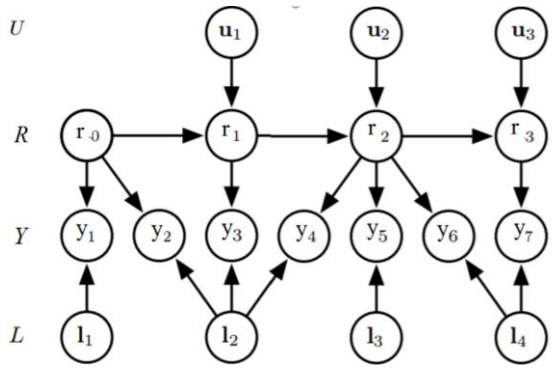


Figure 1.6 An arrow form node A to B means that the probability of B is conditionate by A. The SLAM system consists on robot poses $\mathbf{r}_i$, landmarks $\mathbf{l}_j$, controls inputs $\mathbf{u}_i$ and landmark measurements $\mathbf{y}_t$. Source: [2] by Joan Solà. Reproduced with author's permission.

The robot poses $\mathbf{r}_i$ at different time stamps and the landmarks positions $\mathbf{l}_j$ are concatenated in the state vector **X.**

$$\mathbf{X} = (\mathbf{r}_1 , \dots , \mathbf{r}_i \dots , \mathbf{l}_1, \dots , \mathbf{l}_j, \dots)$$

The state **X** is not directly observable, and must be inferred from the set of received measures **Y** (Hidden Markov model) and the set of control inputs **U.**

The SLAM solution is the state **X*** that maximizes de joint probability Pr(**X**,**Y**,**U**). This joint probability can be written as a product of all the conditionals.

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmax}} \Pr(\mathbf{X}, \mathbf{Y}, \mathbf{U})$$

$$\Pr(\boldsymbol{X}, \mathbf{Y}, \mathbf{U}) \propto \Pr(\mathrm{x}_0) \Pr(\mathbf{X}|\mathbf{X}, \mathbf{U}) \Pr(\mathbf{Y}|\mathbf{X})$$

The graph is dived into two types of variables: states to estimate and observed data. it can be expressed as a factor graph. This graph has only two nodes: states and factors. The factors are either measurement from landmarks observations or input controls that and represent the constraints between the states.

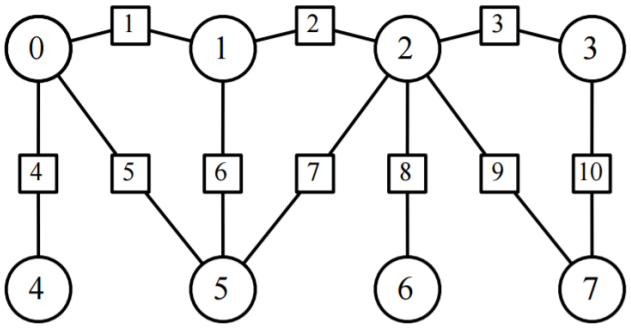


Figure 1.7 Factor graph for the landmark-based SLAM of Figure 1.6. Nodes representing known data have been replaced by factors (squares) that depend on the unknown variables or states (circles). Source: [2] by Joan Solà. Reproduced with author's permission.

The factors ϕ_k are labelled using a unique running index k and are computed from the motion and landmark observation models. Each factor relates two states from **X** and has an associated error $\mathbf{e}_\mathrm{k}$.

Model functions

$$\mathbf{r}_\mathrm{i} = \mathrm{f}(\mathbf{r}_{\mathrm{i}-1}, \mathbf{u}_i) + \mathbf{n}_i \qquad \boldsymbol{n_i} \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, \Omega_i^{-1})$$

$$\mathbf{y}_\mathrm{t} = \mathrm{f}(\mathbf{r}_\mathrm{i}, \mathbf{l}_\mathrm{j}) + \mathbf{v}_t \qquad \mathbf{v}_t \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, \Omega_k^{-1})$$

Ω_i and Ω_k are, respectively, the information matrix (inverse of covariance) of the motion and observations models.

Errors

motion: $\mathbf{e}_k(\mathbf{r}_\mathrm{i}, \mathbf{r}_{\mathrm{i}-1}) = \mathrm{f}(\mathbf{r}_{\mathrm{i}-1}, \mathbf{r}_i) - \mathbf{x}_\mathrm{i}$

observation: $\mathbf{e}_k(\mathbf{r}_\mathrm{i}, \mathbf{l}_\mathrm{j}) = h(\mathbf{r}_i, \mathbf{l}_\mathrm{j}) - \mathbf{y}_t$

General: $\mathrm{e}_\mathrm{k}(\mathbf{X})$

X is the state vector containing all the robots and landmarks variables.

$$\phi_k = \exp\left(-\frac{1}{2}\ \mathrm{e}_\mathrm{k}^\mathrm{T}\Omega_\mathrm{k}\mathrm{e}_\mathrm{k}\right)$$

Note that an error is a normal random variable (with 0 mean). The factor ϕ_k expression comes from its probability distribution function (PDF)

$$\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}^{-1})$$

$$\text{PDF} \quad \frac{1}{(2\pi)^{n/2}\ |\boldsymbol{\Omega}^{-1}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{e}-0)^\mathrm{T}\Omega_\mathrm{k}(\mathbf{e}-0)\right)$$

n is the number of elements in **e** and $|\cdot|$ means the matrix determinant

Finally, the joint probability Pr(**X**,**Y**,**U**) can be written as the product of all its factors ϕ_k [2].

$$\Pr(\mathbf{X}, \mathbf{y}, \mathbf{u}) \propto \prod_{k=1}^{K} \phi_k$$

Maximizing this probability distribution function is equivalent to minimizing its negative log-likelihood.

$$\boldsymbol{X}^* = \underset{\boldsymbol{X}}{\operatorname{argmin}} \sum_{k=1}^{K} \boldsymbol{e}_k(\boldsymbol{X})^T \boldsymbol{\Omega}_{\boldsymbol{k}} \, \boldsymbol{e}_{\boldsymbol{k}}(\boldsymbol{X})$$

This is a nonlinear least square problem and can be solved using the Gauss Newthon method. The sparse structure of the SLAM problem is exploited to efficiently find the optimal solution, even though the big size of the state vector **X.** The two most popular methods to numerically solved this problem are QR and Cholesky factorization. The reader will find good tutorials on Graph Slam in [2] and [3].

1.2 Monocular SLAM

Perspective projection

A monocular Camera is a projective sensor that associates a point **P** in 3D space (object point) with the pixel coordinates (u, v) of the points **p** in the 2D image plane (image Point). It consists of two steps: projection and pixelization.

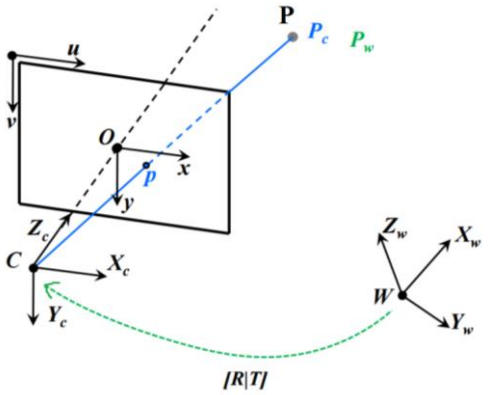


Figure 1.8 Source: Course Vision Algorithms for mobile Robotics, by Prof. Scaramuzza

Projection

The projected point **p** (image point), associated to the object Point **P,** is the intersection between the line **CP** with the image plane. It is obtained applying triangle similarities. f is the focus distance express in metric units.

$$\mathbf{P_c} = \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X_C \\ Y_C \end{bmatrix} \frac{f}{Z_C}$$

The object point **P** must be expressed in camera coordinates, $\mathbf{P_C}$. The coordinates of **P** in camera and world frame, denoted respectively by $\mathbf{P_C}$ and $\mathbf{P_w}$, are related by:

$$\mathbf{P_w} = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

$$\mathbf{P_w} = \mathbf{R}\,\mathbf{P_C} + \mathbf{T}$$

T is the position vector of the camera relative to the world, expressed in world coordinates.

R is the rotation matrix that orientates the camera frame with respect to the world frame. Note that the rotation matrix **R** is named following the standard robotics convention. Let **a** be a vector, then its representations in world $\mathbf{a}_w$ and camera $\mathbf{a}_c$ frames are related by:

$$\mathbf{a}_w = \mathrm{R}\,\mathbf{a}_c$$

Pixelization

In this step, the image point **p** is expressed in pixel units **u** instead of metric units.

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u_0 + s_u x \\ v_0 + s_v y \end{bmatrix}$$

s_u, s_v are the relation between pixels and metric distance in the vertical and horizonal direction [pix/m]. u_0, v_0 are the pixel coordinates of the principal point **O.**

Full model in homogeneous coordinates.

The homogenous representation of the pixel coordinates point $\mathbf{u} \in \mathbb{R}^2$ is denoted by $\underline{\mathbf{u}}$ and it is defined by:

$$\underline{\mathbf{u}} \triangleq \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \in \mathbb{R}^{2+1}$$

$$\underline{\mathbf{u}} = [u_1 \; u_2 \; u_3]^T$$

Working with homogenous coordinates, the pin-hole camera model is expressed as:

$$\underline{\mathbf{u}} = \mathbf{K}\mathbf{P_c} \qquad \underline{\mathbf{u}} = \mathbf{K}\mathbf{R}^{\mathbf{T}}(\mathbf{P_w} - \mathbf{T})$$

$$\mathbf{K} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\alpha_u = f s_u \qquad \alpha_v = f s_v$$

$$\mathbf{u} = [u, v]^{\mathrm{T}} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \frac{1}{u_3}$$

α_u and α_v are the focal length measured in horizontal and vertical pixels.

The matrix **K** is known as the intrinsic matrix, because all the parameters are intrinsic to the camera itself.

Bearing – only Sensor

The observation model of a monocular camera maps a 3D point **P** to its related image point **p.** However, this function is not invertible, and, given an image point, it is impossible to recover the 3D object Point.

It is, therefore, a bearing-only sensor. Only the direction (two angles) of the observed objects is measured, but not the distance or range. The object point $\mathbf{P_w}(\lambda)$ lies on a semi-infinite straight line parametrized by λ

$$\mathbf{P_w}(\lambda) = \mathbf{T} + \lambda\, \mathbf{R}\, \mathbf{v}\,, \quad \lambda > 0$$

$$\mathbf{v} = \mathbf{K}^{-1}\underline{\mathbf{u}}$$

The 3D localization of a point **P** can be determined by considering multiple point of views coming from different camera or from the motion same camera, known as monocular slam.

Monocular Slam

Recovering the standard Slam formulation, the pixel coordinates $\mathbf{u} = (u, v)$ are now the measurements and the 3D points $\mathbf{P_w}$ are the Landmarks.

A monocular camera does partial observations, i.e. a measurement does not observe all the degrees of freedom of a landmark.

This characteristic is challenging in landmark initialization. To invert the observation function $h(\,)$ to find a new landmark Position $\mathbf{P_w}$, a gaussian prior of the unobservable dimensions d must be provided. This new variable is added to the EKF Prediction-update loop, where it will be estimated [4] .

$$d \in [d_{min}, \infty\,]$$

This new variable d is unbounded and the observation function h( ) is non-linear with respect to d. Therefore, a naïve prior will probably break the linearity condition and make EKF fail.

To solve these problems, a new variable ρ is defined, known as inverse depth.

$$\rho \triangleq 1/d$$

$$\rho \in [0, 1/d_{min}\,]$$

The observation function in homogeneous coordinates is linear in ρ, and ρ is bounded. The linearization is now reasonable and valid and the EKF framework will estimate this new variable easily. [5]

1.3 Dynamic Vision Sensor (Event Based Cameras)

An event camera [6] has independent pixels that sends information “events” when they perceive a local change of brightness. The information is transmitted asynchronously exactly when the change occurs, with a microsecond resolution.

Each of these events is a 4-element vector and consists of its space-time coordinates and the polarity of the brightness change.

$$\mathbf{e} = (t, u, v, p)$$

The output of this camera is, therefore, a stream of events, fired independently in the pixels of its sensor plane. This behaviour is completely different from a traditional camera, that sends static images at a given frame rate.

Since events are caused by brightness changes over time, an event camera naturally responds to edges in the scene in presence of relative motion.

Advantages and Applications

The main advantage of the sensor is obviously its time resolution, better than high speed conventional vison sensors running at thousands of frames per second.

The power requirements are very low, and its output is very efficient, as no redundant information is transmitted. Moreover, its independent architecture offers a very high dynamic range (120 dB) making the sensor robust to high contrast scenes.

The potential application of this sensor is huge in fast robotics, where they will enable robots to autonomously and on-board localize itself while performing fast manoeuvres.

For example, indoor navigating is challenging due to close distances and lots of unexpected obstacles. An event camera to perform visual odometry is essential in this environment.

Other sectors such as surveillance, motion analysis and particle tracking will also use the sensor ability to naturally and directly highlight the changes in the scene.

Technology

The architecture of each pixel is independent and it is based on a photoreceptor, a differentiating circuit and two comparators. A detailed explanation on the electronics components and its operation can be found in [6], where the 128x128 DVS is presented.

Mathematical Model

The event based camera responds logarithmically to the pixel intensity I. Events are fired in a pixel when its change in the log intensity is bigger than a threshold C.

$$\bar{\mathrm{I}} \triangleq \log(\mathrm{I})$$

$$\text{Event is fired when: } \left| \Delta \bar{\mathrm{I}} \right| \geq \mathrm{C}$$

The intensity $\bar{\mathrm{I}}(\mathbf{p}, t)$ depends on time t and its pixel position $\mathbf{p}$. If the pixel position is constant, using Taylor first order approximation.

$$\Delta \bar{\mathrm{I}} = \left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathrm{t}} \right|_{\mathbf{p},t} \Delta t$$

The goal is to relate now this the time derivative with two geometric properties: the spatial image gradient $\nabla\bar{\mathrm{I}}$ and the pixel relative velocity $\boldsymbol{u}$ (pixel velocity in the sensor plane).

Under the static scene assumption:

$$\bar{\mathrm{I}}(\mathbf{p} - \boldsymbol{u}\Delta t\,, t + \Delta t) = \bar{\mathrm{I}}(\mathbf{p}, t)$$

Using again first order Taylor Expansion (note now that the pixel position $\mathbf{p}$ is not constant) the relation between time and space derivatives is found.

$$\bar{\mathrm{I}}(\mathbf{p} - \boldsymbol{u}\Delta t\,, t + \Delta t) = \bar{\mathrm{I}}(\mathbf{p}, t) - \left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathbf{p}} \right|_{\mathbf{p},t} \boldsymbol{u}\,\Delta t + \left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathrm{t}} \right|_{\mathbf{p},t} \Delta t$$

$$\left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathbf{p}} \right|_{\mathbf{p},t} \boldsymbol{u}\,\Delta t = \left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathrm{t}} \right|_{\mathbf{p},t} \Delta t$$

$$\text{Event is fired when } \left| \left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathbf{p}} \right|_{\mathbf{p},t} \boldsymbol{u}\,\Delta t \right| \geq C$$

$$\nabla\bar{\mathrm{I}} = \left. \frac{\partial \bar{\mathrm{I}}}{\partial \mathbf{p}} \right|_{\mathbf{p},t}$$

$$\text{Positive polarity event} \qquad < \nabla\bar{\mathrm{I}}\,, \boldsymbol{u} > \Delta t \geq C$$

$$\text{Negative polarity event} \qquad < \nabla\bar{\mathrm{I}}\,, \boldsymbol{u} > \Delta t \leq -C$$

$< \cdot , \cdot >$ denotes the scalar product

Δt is the time difference from the last event fired in the same pixel. The threshold C is not constant and can be expressed as a percentage of the current brightness value.

Note that the event rate ratio, defined as $\frac{1}{\Delta t}$, is proportional to the product of gradient with the parallel component of the motion vector. Therefore, high gradient structures (edges of the scene) will generate events as they move in the sensor plane.

1.4 Literature Review: Towards Slam with Event Based Cameras

In the last years, event based sensors have attracted the interest of the research community as the advantages of this new sensor are very appealing for new algorithms development and real-world applications.

However, the standard vision algorithms based on a fixed frame camera are useless in this new asynchronous and event per event paradigm.

The goal is clear: solving the event-based SLAM problem truly unlocking the camera potential, i.e. adopting an event per event performance.

Since 2012, this problem has been tackled step-by-step in scenarios with increasing complexity: dimensionality (2D or 3D), type of motion and type of scene.

By today, only two research groups have designed and published (in 2016) an algorithm to perform SLAM (or visual Odometry) with 6 DOF. Both methods are shortly introduced in this section.

Note: The reader will find a more comprehensive and literature research in [7], one of the last papers published on the subject (December 2016)

First Method

The Robotics and Perception Group of University of Zurich, led by Prof. Davide Scaramuzza, has work extensively with event based camera since 2013.

In their last work: EVO: A Geometric Approach to Event-Based 6-DOF Parallel Tracking and Mapping in Real-time [7] they present a 6DOF visual odometry algorithm based on parallel tracking and mapping.

The Mapping thread, is presented on their previous work: EMVS: Event-based Multi-View Stereo [8] and it is based on ray projecting the triggered events into a discretized grid. Once all events haven been projected, the 3D points with higher ray density correspond to the edges of the scene. From a reference viewpoint, a depth map is then

created by performing a 1 dimensional maximum search in the ray direction of every pixel of this viewpoint.

The Tracking thread is based on accumulating a big set events on the sensor plane to get an "intensity like" image. The position is optimized by minimizing the photometric error between the projection of the 3D point cloud into the estimated sensor position and the accumulated events.

The algorithm performs visual odometry, as it has no place recognition capabilities. In the publication paper, it is tested in an office-like environment and its accurate results (compared to ground truth) demonstrates the success and potential of their approach.

Second Method

The Robot Vision Research Group of Imperial College London, led by Prof. Andrew Davison, is the other big player working with Event Based Vision.

In their last work: Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera [9] they present a 6 DOF Tracking and 3D Mapping, based on the use of three decoupled filters running in parallel.

The first one estimates the global 6 DOF camera motion using an Extended Kalman filter with a constant position model. The measurement updates rely on a reconstructed intensity image.

The second filter performs Pixel Wise EKF Based Gradient estimation. From this gradients map, the log intensity is reconstructed running on a GPU.

The third filter computes the inverse depth, using another pixel-wise EKF and using also the reconstructed intensity.

Although the tracking results and the intensity reconstruction achieve good qualitative results, no ground truth comparison is available in their publication.

2. Algorithm

2.1 System Overview

Event based camera as an edge detector

When the camera moves through a scene, events are fired in those pixels that detect an intensity change. As the intensity changes correspond to the edges of the scene, the sensor acts as a natural edge detector in presence of relative motion.

Considering that each received event comes from an edge of the scene offers a simple, yet accurate and realistic sensor model. This strategy is an alternative to use the previous mathematical model to relate the pixel intensity gradient, its relative velocity and the events rate.

Line based environments

The proposed system is designed to work in line based environments, where the scene can be accurately described by a set of 3D lines or segments.

Adopting these 3D features leads to a landmark Based Slam in Event Based vision, a completely new and original approach. Until now, only dense or semi-dense methods have been developed in the research community.

The Landmark based Map is a set of 3D segment that describe the structure of the scene, represent by its endpoint.

Although this does not cover all possible scenes, line based environments are very common in human-made environments, such as indoor scenes, cities and rooms. It is particularly in this environment where the use of an event camera has a lot of potential in high-speed robotics.

For example, enabling fast flying robots to autonomously navigate inside a building after a disaster in search and rescue operations.

Event-based parallel tracking and mapping

The proposed Event–based system solves the localization and mapping problem by splitting it into two separated tasks that are processed in parallel: a tracking and a mapping module.

The first thread is responsible for tracking the hand-held camera motion and runs in real-time. Exploiting the asynchronous and accurate time resolution of the camera, the tracking thread is executed at event rate, once every time that a new event is received.

The second thread is the mapping module, and it is executed when the current camera state (position and orientation) has significantly changed with respect to the last key position state. To produce a detailed and accurate map, expensive point cloud building and model fitting

techniques are used. Therefore, this thread is executed at a much slower rate than the tracking module. The trajectory of the handheld camera is divided into key positions, each new position with a significantly different point of view from the last key position.

The rate of operation of the Mapping thread is named as " Key Pose rate ".

This separation principle of the SLAM system was first introduced in [10], and it is known as PTAM (Parallel Tracking and Mapping).

The proposed system is presented visually in a block diagram in Figure 2.1. The tracking module estimates the trajectory of the event camera (6-DOF Pose) using the event stream and assuming that a map of the environment is known. The mapping module builds a new map assuming that the past trajectory is known. Both modules operate in parallel and rely on the output of the other.

Event per event performance

The tracking thread works at event rate, analysing each event individually. This means that that each event provides a little (in fact, very little) piece of information to slightly improve the position estimate and the map.

It is still unclear if an individual event provides enough information to be studied alone or if events must be clustered to get a meaningful data. This open question is one of the big challenges of Event-Based Vision, making these field attractive and appealing for new ideas and scientific research.

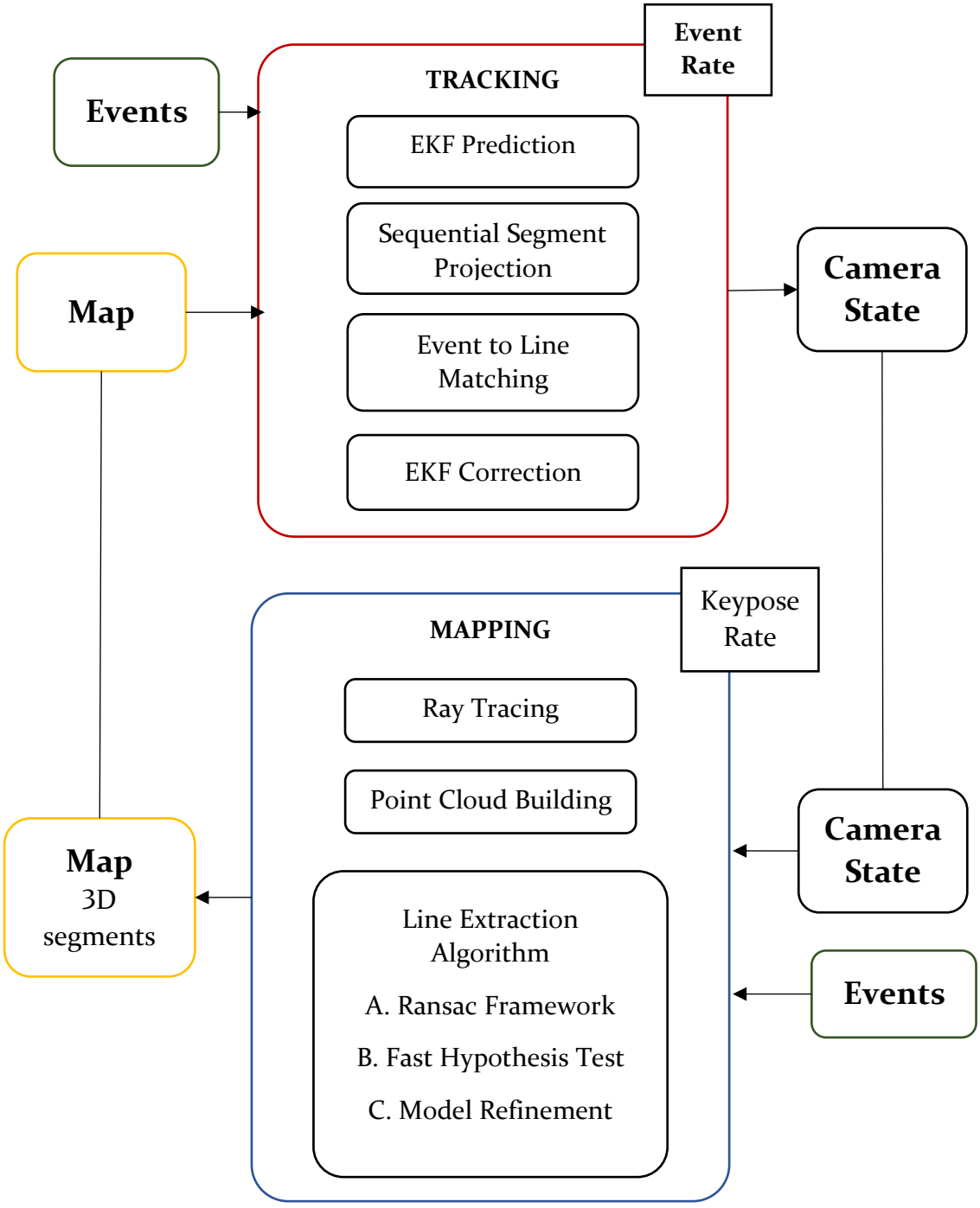


Figure 2.1 Event Based Parallel Tracking and Mapping

2.2 Tracking

Quick overview

The goal of the tracking algorithm is to use the incoming events to autonomously localize the camera in a known map. This known map has been provided by the mapping thread.

The proposed tracking algorithm is presented visually with a block diagram in Figure 2.2. And it is characterized by:

A. Landmark Based Map. The environment is described as a set of 3D segments, each one represented by its endpoints.

B Event per Event Performance: events are studied individually as independent observations. Each one is used to improve the estimated state of the camera.

C. EKF Framework: The EKF prediction and correction steps are performed asynchronous after each individual event is received.

D. Event to Line Matching: each event is associated to a 3D segment of the known map (Landmarks) and the error (point to line distance) is used to correct the last state prediction.

These four element are discussed in the following "*Detailed Description*" section.

Note: If the reader is unfamiliar with Simultaneous Localization and Mapping (SLAM) and the Extended Kalman Filter Framework (EKF) it is highly recommendable to go to the introductory section of this thesis before continuing.

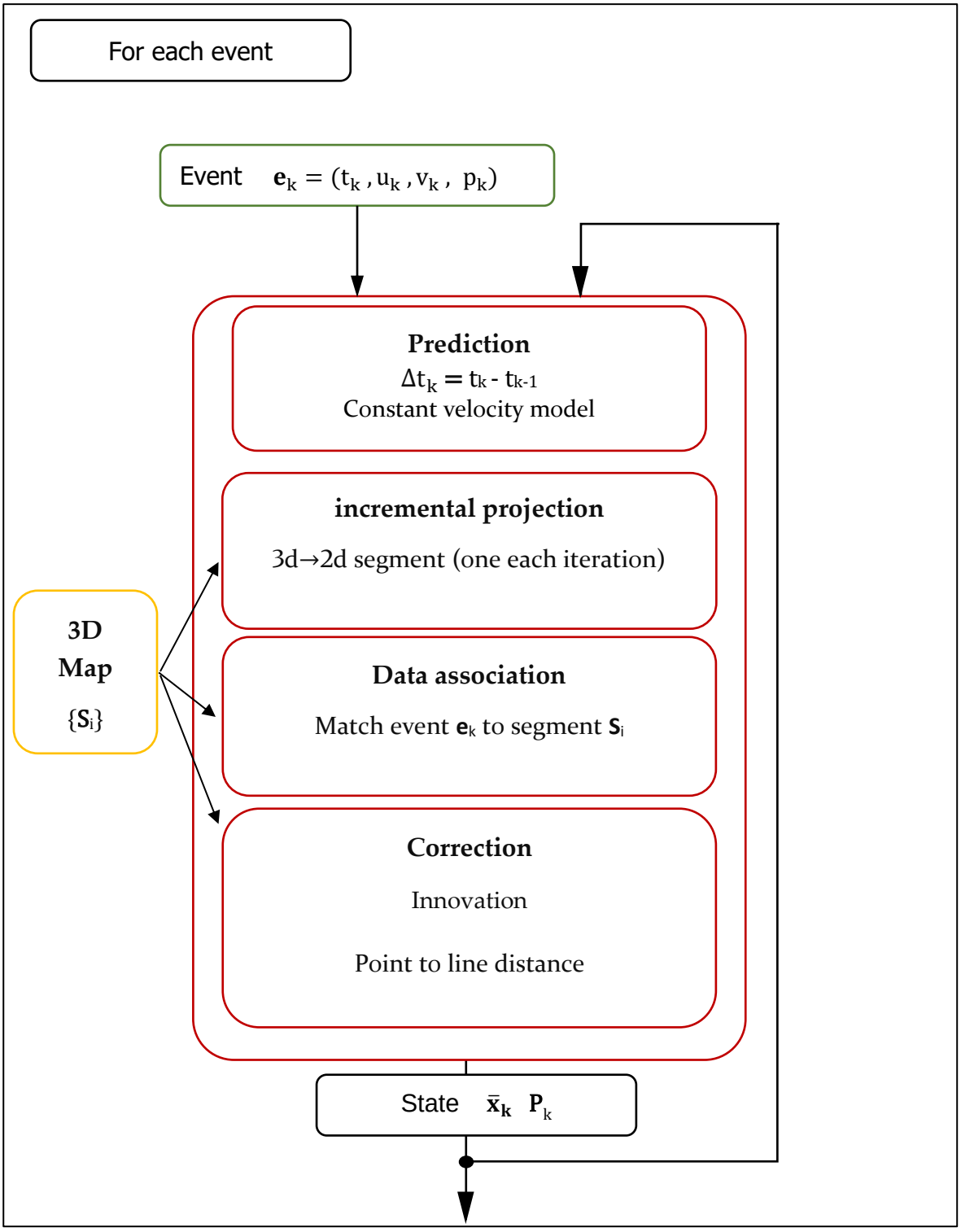


Figure 2.2 Tracking Algorithm

Detailed Description

A. Landmark Based Map

In the proposed algorithm, the map is a set of 3D segments, represented by its two endpoints. It is a mean-only representation of the world without any variance or confidence indicator.

$$\text{Map (3D Segment set)} = \{ \mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_i, \dots \}$$

B. Event per Event Performance

B.1 Event

The event stream is a list of events sorted by their time stamp. Each event is a four-element tuple: time stamp, pixel position (horizontal and vertical components) and polarity of the local intensity change. From these 4 elements, only the time and the pixel position are used in the algorithm.

$$\text{Event Stream} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_k, \dots, \mathbf{e}_n\}$$

$$\mathbf{e}_k = (t_k, u_k, v_k, p_k)$$

In the introductory section of this thesis (see page 19) the reader will find out how an event Camera Works and when the events are generated.

B.2 Incremental 3D – 2D projection

A list of 2D segments, corresponding to the projections of the 3D segments into the state estimate $\bar{\mathbf{x}}_k$ (mean) is used to associate each event $\mathbf{e}_k$ (measure) to its corresponding 3D Segments (Landmark).

Exploiting the asynchronous and event per event behaviour of the algorithm, and incremental 3D-2D line projection is proposed. Instead of projecting all the 3D segments into a given state $\bar{\mathbf{x}}_k$, only one 3D segment is projected each iteration, in a cyclical (ring) way.

Given the extremely fast rate of events, this method keeps the 2D set updated to later perform data association, while being computationally efficient.

C. EKF Framework

EKF is the fastest solution to treat each event individually. This speed will be crucial to achieving a real-time implementation. As events come at an extremely high rate, the linearization of the movement and measurement functions is a valid approximation. The EKF prediction and correction step are performed after each individual event is received.

C.1 State and motion model

The camera state comprises its position (in world coordinates), orientation (with respect to the world) and its linear and angular velocity.

The velocity (linear and angular) has been included in the state to enable the use of a constant velocity function. Given the kinematics and dynamics of a hand-held camera following a random but smooth trajectory, this motion model offers the perfect balance between complexity (medium size state vector and linearity) and accuracy (good local approximation of reality).

The orientation is represented using quaternions $\mathbf{q}$ (represented by a unit 4-vector), a non-minimal representation without any singularity. Quaternions are a mathematical tool with well-defined algebraic operations to work with 3D rotations.

Therefore, the state is a column vector with 13 components.

$$\mathbf{x} = (\,\mathbf{p}, \mathbf{q}\,, \mathbf{v}, \mathbf{w}\,)$$

The state of the camera $\mathbf{x}$ is modelled with a Gaussian variable where $\bar{\mathbf{x}}$ is the mean of $\mathbf{x}$ and $\mathbf{P}$ is its covariance matrix.

$$\mathbf{x} \sim \mathcal{N}(\bar{\mathbf{x}}, \mathbf{P})$$

Constant velocity motion model $\boldsymbol{f}(\cdot)$

$$\mathbf{x}_k = f\,(\mathbf{x}_{k-1}, \Delta t_k\,, \mathbf{n_k})$$

$$\mathbf{n_k} = \begin{bmatrix} \mathbf{v_{nk}} \\ \boldsymbol{\omega}_{\mathbf{nk}} \end{bmatrix} \sim \mathcal{N}\,\{0, \mathbf{Q_k}\}$$

$$\mathbf{Q_k} = \Delta t_k\,\mathbf{Q}$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v}\,\Delta t_k$$

$$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{v_{nk}}$$

$$\mathbf{q} \leftarrow \mathbf{q} \otimes \mathbf{q}\{\boldsymbol{\omega}_{\mathbf{k-1}}\,\Delta t_k\}$$

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{\omega}_{\mathbf{nk}}$$

$\mathbf{x}_k$ is the state of the robots when the event k is received. $\Delta t_k = t_k - t_{k-1}$ is the time difference between two consecutive events. As these events are asynchronous, Δt_k is not constant.

$\mathbf{n_k}$ is the disturbance vector and it depends on Δt_k.

$\mathbf{q}\{\mathbf{w}\,\Delta t\}$ represents the quaternion associated to a rotation of $\theta = \|\mathbf{w}\Delta t\|$ radians around the axis $\mathbf{u} = \mathbf{w}\,/\|\mathbf{w}\|$ .The operator $\otimes$ is the quaternion product.

Note 1. Quaternions and rotations

The reader will find a useful reference about quaternion algebra and its application to rotations in [11]. A very short definition is presented below.

Quaternion definition	$\mathbf{q} = q_w + q_x i + q_y j + q_z k$ $\mathbf{q} = q_w + \mathbf{q}_v$ $\mathbf{q} = (q_w,\ q_x,\ q_y,\ q_z)^{\mathrm{T}}$
Angle - axis representation of a rotation of angle θ around the axis $\mathbf{u}$	$\boldsymbol{\theta} = \mathbf{u}\theta$ $\mathbf{u} = [u_x, u_y, u_z]^{\mathrm{T}}$
Equivalent quaternion representation	$\mathbf{q}\{\boldsymbol{\theta}\} = \begin{bmatrix} \cos(\theta/2) \\ \mathbf{u}\sin(\theta/2) \end{bmatrix}$ $\|\mathbf{q}\| = 1$

C.2 Prediction Step

In the prediction Step, the last state estimate $\mathbf{x}_{k-1}$ and the time increment Δt_k are used to predict the new state $\mathbf{x}_k$ with the chosen constant velocity motion model.

As the state $\mathbf{x}_k$ is modelled as a Gaussian variable, each prediction consists on computing the new values of its mean $\bar{\mathbf{x}}_k$ and covariance $\mathbf{P}_k$

The estimate mean $\bar{\mathbf{x}}_k$ is obtained from the non-linear motion model f evaluated on the expected values (mean) of the last state and disturbance vector: $\bar{\mathbf{x}}_{k-1}$ and $\mathbf{0}$ respectively.

$$\bar{\mathbf{x}}_k = f(\bar{\mathbf{x}}_{k-1}, \Delta t_k, \mathbf{0})$$

To compute $\mathbf{P_k}$, the motion model is linearized and the last state covariance $\mathbf{P_{k-1}}$ is propagated through the Jacobians Matrix.

$$\mathbf{P_k} = \mathbf{F_x P_{k-1} F_x^T} + \mathbf{F_n Q_k F_n^T}$$

$$\mathbf{F}_x = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}},\ \mathbf{n}=0} \qquad \mathbf{F}_n = \left.\frac{\partial f}{\partial \mathbf{n}}\right|_{\bar{\mathbf{x}},\ \mathbf{n}=0}$$

$\mathbf{F}_x$ and $\mathbf{F}_n$ are, respectively, the Jacobian Matrix of the motion function with respect to the state $\mathbf{x}$ and the disturbance $\mathbf{n}$.

C.3 Correction Step

In the correction step of the EKF, the measures received by the sensor are used to improve the estimate of the state of the robot (camera).

Each event $\mathbf{e}_k$ is treated as independent observation and has already been associated to its landmark: a 3D segment. (see event to line – matching).

Following the naming convention commonly used in EKF , the pixel coordinates (u_k , v_k) of the triggered event is the real measure $\mathbf{y}$.

$$\mathbf{y} = (u_k , v_k)$$

However, in point to Line matching, only the orthogonal distance between the point and the associated line is observable.

The innovation $\mathbf{z}$ (error) is, therefore, this 1 dimensional distance and it is computed directly with the point to line distance function.

$$z = d(\mathbf{y}, \mathbf{s_i}, r) = g(\mathbf{y}, \mathbf{x}, v)$$

$$v \sim \mathcal{N}\{0, R\}$$

$$\mathbf{s_i} = \pi(\mathbf{x}, \mathbf{S_i})$$

d is the point to line signed distance in $\mathbb{R}^2$ and π is the pin Hole Projection function.

$\mathbf{S_i}$ and $\mathbf{s_i}$ are, respectively, the associated 3D (Map) and 2D segment (sensor plane)

The innovation $\bar{z}$ is obtained by evaluating the innovation function in the expected values of their input variables.

$$\bar{z} = d(\mathbf{y}, \bar{\mathbf{s}}_\mathbf{i}, 0) = g(\mathbf{y}, \bar{\mathbf{x}}, \mathbf{S_i}, 0)$$

$$\bar{\mathbf{s}}_\mathbf{i} = \pi(\bar{\mathbf{x}}, \mathbf{S_i})$$

$$Z = \mathbf{G}_x \mathbf{P} \mathbf{G}_x^T + R$$

The Jacobian of the innovation function $\mathbf{G}_x$ with respect to the state is computed using the chain rule. Z is the variance of the innovation and R the variance of the innovation.

$$\mathbf{G}_x = \frac{\partial g}{\partial \mathbf{x}}\Big|_{\bar{\mathbf{x}}, \mathbf{S}_i, y} = \frac{\partial d}{\partial \mathbf{s}_i}\Big|_{\bar{\mathbf{s}}_i, y,} \cdot \frac{\partial \pi}{\partial \bar{\mathbf{x}}}\Big|_{\bar{\mathbf{x}}, \mathbf{S_i}}$$

The Correction Step for the mean $\bar{\mathbf{x}}$ and the covariance $\mathbf{P}$ is performed using the optimal Kalman Gain $\mathbf{K}$, that outputs the optimal solution of a linear Bayesian inference problem.

$$\mathbf{K} = -\mathbf{P}\mathbf{G}_{\mathbf{x}}^{\mathrm{T}}\mathbf{Z}^{-1}$$

$$\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{K}\bar{\mathbf{z}}$$

$$\mathbf{P} \leftarrow \mathbf{P} - \mathbf{K}\mathbf{Z}\mathbf{K}^{\mathbf{T}}$$

The sub index k is withdrawn because in the correction step all the variables are belong to the same event step k.

Note 1. Point and lines in sensor plane

As point to line matching is performed, only the position component perpendicular to the line can be observed. The position component that is parallel to the line is not observable, as it is not possible do determine from which exact point of the line does the event "come from".

When a standard fixed frame monocular camera (Pin Hole model) moves parallel to a line, this line is perceived as static in the sensor plane. Although there is relative motion, it seems that the points belonging to the line are static.

D. Event to Line Matching

Using the pixel coordinates u_k , v_k of the event $\mathbf{e}_k$ and the set of 2D segments, each event is associated with the segment $\mathbf{s}_i$ (2D) that minimizes the point to line distance (i.e. the closer line). As the segment $\mathbf{s}_i$ (2D) is the projection of the Map segment $\mathbf{S}_i$ (3D) into the sensor plane, the event $\mathbf{e}_k$ and landmark $\mathbf{S}_i$ (3D segment) are also indirectly associated.

$$\text{Map (3D Segment set)} = \{ \mathbf{S}_1, \mathbf{S}_2 , \dots , \mathbf{S}_i , \dots \}$$

$$\mathbf{s}_i = \pi (\bar{\mathbf{x}} , \mathbf{S}_\mathbf{i})$$

$$\text{2D Segment Set} = \{ \mathbf{s}_1, \mathbf{s}_2 , \dots , \mathbf{s}_i , \dots \}$$

$$\text{Associated Segment:} \qquad i = \operatorname{argmin} \{ d(\mathbf{e}_k , \mathbf{s}_i) \}$$

d is the point to line distance function in $\mathbb{R}^2$ and π is the pin-hole projection function.

Before validating the data association process three additional tests are performed:

- The orthogonal projection of the event position into the line must lie between the start and end points of the segment.
- The minimum distance must be smaller than a threshold. Otherwise, it comes from unknown edges or noise from the sensor.
- If the difference between the two minimum distances must be greater than a threshold. Otherwise, it is not clear which segment is the source of the event and it is risky to perform any data association.

2.3 Mapping

Quick overview

In the tracking algorithm, the camera has already estimated its trajectory using the available map and the triggered events.

In the Mapping thread, this estimated past trajectory, together with the same past triggered events are used now to build a better map. Once this new map is finished, it will replace the old map that the tracking algorithm uses to estimate this trajectory.

The Mapping algorithm is based on three big blocks (Figure 2.3).

Event Ray Tracing: Each event is ray-projected to the 3D space to find the 3D edges, without any explicit data association. The 3D space with a higher ray density will correspond to the edges of the scene.

Point Cloud: The world is discretized into voxels with a score, that counts the number of rays traversing each voxel and behaves as a probability distribution function. High score values indicates high probability of belonging to a 3D edge of the scene.

Line Extraction Algorithm: It analyses the point Cloud to find the 3D segments that will become the next map landmarks.

These three steps are discussed in the following "*Detailed Description*" section. The line Extraction Algorithm is further divided in two sections: a quick Overview with a diagram block and its corresponding detailed description.

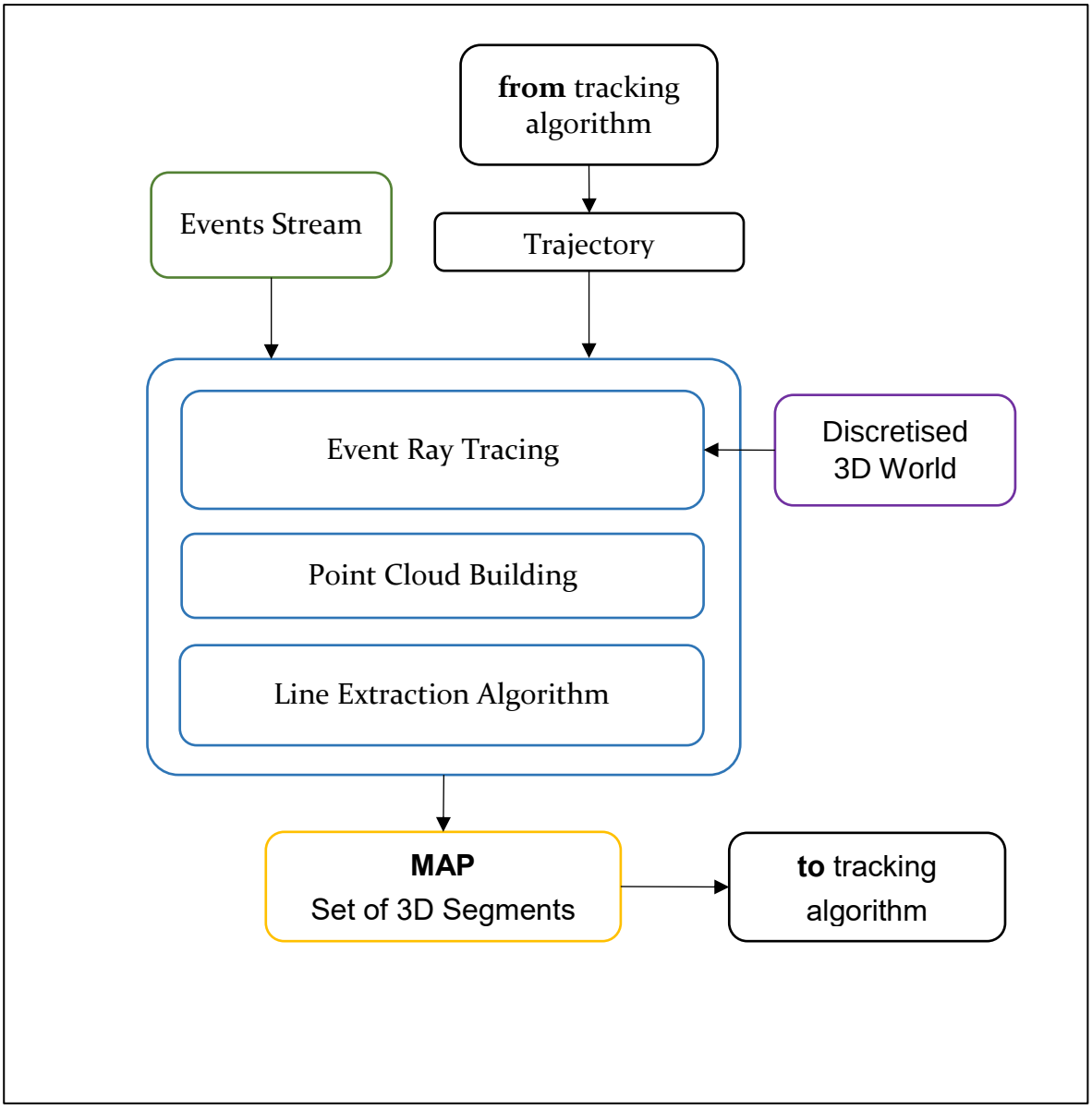


Figure 2.3 Mapping Algorithm

Detailed description

World Discretization

To perform the event-ray tracing and to build the Point Cloud, the 3D space must be discretized. In this project, the world is discretized using a rectilinear grid, based on cartesian coordinates. Each little volume, called Voxel, is rectangular a prism.

The main advantage of the proposed grid is its simplicity. It also offers a uniform space distribution that always work correctly independent of the point of view of the camera.

The grid choice is different from the ray tracing method presented by in [8], where the world is discretised using the pixelized sensor plane of a reference view and a given number of depth planes, discretized with inverse depth. This strategy follows the natural behaviour of a monocular camera, reflecting the uncertainties of the bearing only observation and the pinhole model projection in a more accurate way. If the camera view does not change a lot from the reference view, it will offer better results than a cartesian grid.

Event Ray Tracing

An event $\mathbf{e} = (u, v, t, p)$ triggered in the sensor frame is a bearing-only observation and the pixel position (u, v) does not completely define the correspondent 3D point that has fired the event.

Exploiting the sparse structure generated by an event camera, where events are only fired in the pixels crossing the scene edges, it possible to ray-project each event to the 3D space to find the 3D edges, without any explicit data association [12]. The 3D space with a higher ray density will correspond to the edges of the scene. This approach has been successfully introduced in event based vision in EMVS: Event-based Multi-View Stereo [8].

Each event projects a ray into the discretized 3D Grid and increments the score of all the touched voxels. The ray is defined by the position $\mathbf{p}$ of focus of the camera (position of the camera) and the direction $\mathbf{r}$ based on the pixel coordinates of the events.

$$\text{ray} : \mathbf{p} + \lambda\,\mathbf{r} \;;\; \lambda \geq 0 \quad \text{(Parametric equation)}$$

$$\mathbf{r} = \mathbf{R}\{\mathbf{q}\} \begin{bmatrix} u_k - u_0 \\ v_k - v_0 \\ f \end{bmatrix}$$

f is the focus distance in pixels and, u_0 and v_0 the coordinates of the central point of the camera. $\mathbf{R}\{\mathbf{q}\}$ is the rotation matrix associated to the quaternion $\mathbf{q}$. Each event $\mathbf{e}_k$ is projected using the position $\mathbf{p}$ and quaternion $\mathbf{q}$ of its corrected state estimate $\bar{\mathbf{x}}_k$

In this project, the ray tracing for the Event Based Mapping is performed using the algorithm presented in [13]. It delivers a very fast ray tracing method with a simple and intuitive implementation.

Note: for those readers interested and curious about the performance of the ray tracing algorithm, a detailed description is here presented. The algorithm follows the direction of the ray to, starting from one hit voxel, find the next hit voxel.

ray traversal algorithm

Starting from the first ray point inside the grid, the following basic module is executed iteratively until the ray reaches one of grid limits. The output of one iteration is directly the input of the next iteration and the score of each visited voxel is incremented by one.

If the camera focus lies outside the voxels grid, the intersection of the ray with the external limits of the grid (a prism with six faces) is computed beforehand.

Basic module of the ray traversal algorithm

Input: Voxel Centre, Ray Entering Point $\mathbf{p_a}$

Output: Ray Exit Point $\mathbf{p_b}$, next Voxel Centre,

Pipeline: Each iteration works with the following ray equation, where $\mathbf{p_a}$ is the ray entering point in the current voxel and $\mathbf{r}$ is the ray direction.

$$\mathbf{p_a} + \lambda\,\mathbf{r}\ ;\ \lambda \geq 0$$

The 6 planes that limit the current voxel are easily calculated with the voxel centre and the grid parameters. All these planes are parallel to one of the 3 projective planes.

The values $\lambda_i \quad i = 1,2,\dots,6$ at which the ray crosses each of the 6 limit planes are computed. Each λ_i defines how much is it possible to move along the ray before hitting the plane.

Only the positives values $\lambda_i > 0$ are considered as the ray has a defined signed direction. ($\lambda_i = 0$ corresponds to the entering point in the voxel).

λ^* is the minimum of $\lambda_i > 0$ and it defines how much is it possible to "travel" while reaming inside the voxel.

$$\lambda^* = \min_{\lambda_i > 0} \lambda_i$$

$$i^* = \operatorname*{argmin}_{\lambda_i > 0} \lambda_i$$

Therefore, the exit point $\mathbf{p_b}$ of the ray is:

$$\mathbf{p_b} = \mathbf{p_a} + \ \mathbf{r}\ \lambda^*$$

The next voxel centre is computed using the current voxel centre, the exit plane index i^* and the grid step parameters.

This operation can be done efficiently if the voxel data structure is implemented as an ordered list, where there is a known and direct algebraic relation between the centre coordinates of the voxel and its address index in the data structure.

Point Cloud building

The point cloud builder receives as an input the discretize world as a set of voxels, where each voxel has already a score. Its goal is to refine these scored Point Cloud so that is optimal for the future line Extraction Algorithm.

Intuitively, the points with higher score are the edges of the scene as these points have been triggering events in the sensor plane as the camera has moved through the scene.

Most of the voxels have low score, a clear indicator that they do not belong to the edges of the scene. Therefore, if the score of a voxel is lower than a certain global threshold (computed easily as a fraction of the maximum score) its score is set to zero.

The original high scores of the Point cloud are kept and they are not binarized, as this integer variable acts as a probability distribution function and will be analysed in the line extraction algorithm.

Global thresholding, while being easy to tune and implement, has also offered better results than simple version of adaptive thresholding to the voxel neighbourhood.

The resulting point cloud is stored in two different ways: as a sorted list containing all the voxels (0 and non-zero score) and as unsorted list of only the non-zero score voxels (after the thresholding process). Both data structures will be useful in the Line Extraction Algorithm.

Line Extraction Algorithm

Quick Overview

The goal of line extraction algorithm is to find 3D segments from the scored point Cloud (where each 3D Point has a score). This 3D segments will become the landmarks of the map.

Finding a lot of candidates' lines from a given point cloud, without any prior information about which points belong to the same line is a challenging task. Conceptually, the line extraction problem includes two different tasks, which are performed simultaneously by the designed algorithm.

> **Segmentation:** It is the task of deciding which 3D points belong to each different segment. All the 3D points should be clustered in different groups, each group representing a different segment. This is a crucial and challenging step as there are a lot of different 3D segments in the scene.
>
> **Model Fitting:** Given the subset of points that belong to one segment, model fitting is the task of estimating the model parameters that better describe (explain) the data points.

The proposed line extraction algorithm is presented visually with a block diagram in Figure 2.4. The shown basic unit is executed iteratively for given number of times or until a predefined number of candidates' lines are found. The algorithm is characterized by:

A. **Ransac Framework for 3D Line Extraction:** randomly generated 3D line hypothesis are tested to check if there are consistent with the Point Cloud.
B. **Fast Preliminary hypothesis test:** before testing a hypothesis in the Ransac Framework, a faster and necessary geometric condition is checked first. This enables the algorithm to quickly discard most of the wrong hypothesis.
C. **Iterative model refinement:** starting from a "good enough" random line hypothesis, (strong evidence that there is a line), the line model is improved to find the locally-best line model. It is based on is based on iterative model fitting and inlier/outlier voting.
D. **Selection of winner Lines:** once the Ransac iterations are finished, the winner lines are selected from the found list. Two methods have been proposed: Non-Maximum Suppression and Iterative Line Extraction with Winner Point erasing.

These four elements are discussed in the following "Detailed Description" section.

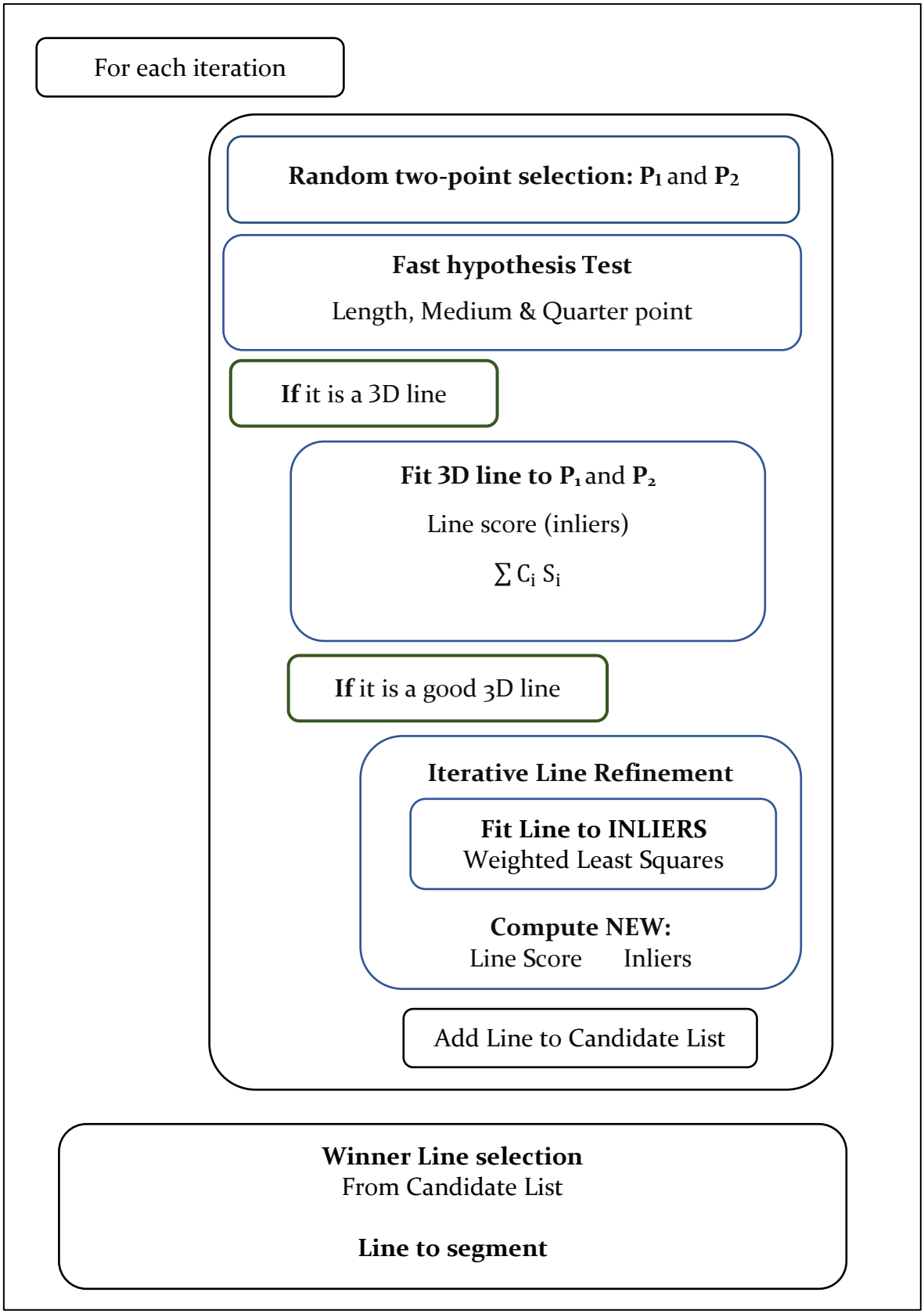


Figure 2.4 Line extraction algorithm

Note: If the reader is unfamiliar with the standard version of Ransac Algorithm, it is highly recommendable to read carefully the following one-page introduction. Otherwise, the elements of the Line Extraction algorithm are presented in the next pages.

Ransac Algorithm – a short introduction

The Ransac Algorithm (Random Sample and Consensus) is an iterative model to estimate the parameters of a certain model from a set of data that contains many outliers.

It was introduced in 1981 by Fischler and Bolles [14] and it outperforms other model fitting algorithm (for example least squares) when the number of outliers the contaminate the data is high. An outlier is a data point that does not fit the true real model, and therefore, it should be ideally not considered in the model estimating process.

Despite many modifications, and improvements, the RANSAC algorithm is essentially composed of two steps that are repeated in an iterative fashion:

- **Hypothesis**

At the beginning of each iteration, a minimum and sufficient number of data points are randomly selected from all the input dataset and a model is computed using only these elements.

- **Test**

In the second step, RANSAC checks which elements of the entire dataset are consistent with the already computed model. All the data points are classified between inliers (if they agree with model) or outlier (disagree). A point is an outlier if their "distance" to the model is over a certain threshold. Optionally, the model can be refined using now all the inliers point. Finally, the model is then ranked by its number of inliers or by an error function that depends exclusively on the inlier data.

This two-step process is repeated until a "good-enough" model is found or for a fixed number of iterations, picking at the end the best-found model.

The whole RANSAC method also includes other features that are not of interest in this algorithm.

Detailed description (Line extraction algorithm)

A. Ransac Framework for 3D Line Extraction

The whole algorithm is based on a Ransac Scheme [14] (Random Sample and Consensus) and it exploits its ability to fit a model to data in the presence of many outliers. This ability is now taken to its extreme. Not only one, but several different line models are fitted to the point cloud, each one related to a different subset of 3D points.

This is possible because RANSAC provides a natural way to find a subset of 3D points that form a line and then find a 3D line model to describe them, without considering all the other 3D points of the scene that do not belong to the found line.

A.1 Random Selection Hypothesis

Following the standard Ransac Framework, two 3D points are selected randomly from the Point Cloud (only considering non-zero score points). The 3D line is fitted to these two 3D points and it is represented by its the parametric equation.

$$\mathbf{P}_1 = [\mathrm{X}_1, \mathrm{Y}_1, \mathrm{Z}_1]^{\mathrm{T}} \qquad \mathbf{P}_2 = [\mathrm{X}_2, \mathrm{Y}_2, \mathrm{Z}_2]^{\mathrm{T}}$$

$$\mathbf{u} = \mathbf{P}_1 \qquad \mathbf{v} = \mathbf{P}_2 - \mathbf{P}_1$$

$$\text{Line Hypothesis} \qquad [x, y, z]^{\mathrm{T}} = \mathbf{u} + t\,\mathbf{v}$$

A.2 Inlier – Outlier classification

Only If the Fast-Preliminary check is passed (see next page), the standard Ransac hypothesis testing step is performed. In this step, all the 3D point from the Point Cloud "vote" if they agree or disagree with the line hypothesis.

The inlier (agree) / outlier (disagree) segmentation is made based on the distance between the model to each 3D point. If this distance (Euclidian point to line distance) is smaller than a threshold C, the 3D point will be an inlier.

$$\text{Line Hypothesis} \qquad (x, y, z) = \mathbf{u} + t\,\mathbf{v}$$

$$\text{Point Cloud} \quad \mathrm{PC} = \{\mathrm{P}_1, \mathrm{P}_2, \ldots., \mathrm{P}_\mathrm{i}, \ldots, \mathrm{P}_\mathrm{N}\} \qquad \mathbf{P}_\mathrm{i} = (\mathrm{X}_\mathrm{i}, \mathrm{Y}_\mathrm{i}, \mathrm{Z}_\mathrm{i})$$

$$\mathrm{d}_i = \frac{|(\mathbf{u} - \mathbf{P}_\mathrm{i}) \times \mathbf{v}|}{|\mathbf{v}|}$$

P_i is inlier if $\mathrm{d}_\mathrm{i} < \mathrm{C}$

A.3 Line Score

Each detected line will have a score that indicates how good or confident the line is, i.e. how does the data support this line model. It is a scalar magnitude and it is based on the number of inliers, its distance to the line and the individual score of each inliner.

Specifically, the line score LS is the sum of the score S_i of its inliers, weighted by its distance to the line.

$$\textbf{Linear mapping} \quad d_i \rightarrow c_i \text{ (distance} \rightarrow \text{weight)}$$
$$0 \rightarrow 1$$
$$C \rightarrow 0$$

$$LS = \sum_{i \in \text{inlier}} c_i \; S_i$$

C is the inlier distance threshold, S_i is the 3D Point Score and LS is the line Model Score

B Fast-Preliminary Check

The Fast Preliminary is a tool to quickly discard Line hypothesis. The proposed strategy is inspired by [15], where the authors efficiently extract lines in a 2D image using a corner detector and an edge map by quickly creating and refusing hypothesis.

The main idea of the Fast-Preliminary Check is to test first a necessary (but no sufficient) condition before performing the regular Ransac hypothesis test (outlier/inlier). If the easy necessary condition is not fulfilled, the model is instantly discarded.

The following conditions are tested:

- **Distance** between the two randomly selected points. To avoid finding very short lines, the distance between the end points of the segment must be greater than a threshold d_ε.

$$\|\mathbf{v}\| = \|\mathbf{P}_2 - \mathbf{P}_1\| \geq d_\varepsilon$$

- **Medium and quarter points**. If the randomly created model is truly a line, the medium and quarter 3D points of the segment should also have associated non-zero score in the Point Cloud. Note that all the original low scores from the ray tracing have been set to zero in the Point Cloud Building Step.

Using the Point Cloud represented as an ordered list (including zero-score elements), the maximum score of the voxels in the neighbourhood of these "key" 3D points is computed. If this maximum is zero, the point does not belong to any segment, and therefore, the hypothesis is not valid.

The neighbourhood is as a box of a predefined radius around the discretized value of the interest 3D point (voxel centre closer to the 3D point).

C Iterative model refinement

If the score of a line is higher than a score threshold, the line hypothesis is classified as a good line and enters the model refinement process. In this process, the line model will be improved so that it estimates the real line equation. It is a robust tool to locally find the best possible line, even though the two randomly picked points are not exactly the end points of the segment. Intuitively, the process uses to scored Point cloud to steer the line, so that it achieves the local maximum possible score.

It is an iterative process, executed for a given number of times or until the line score value has converged (local maximum found). Each iteration has three steps:

- **Model Line update:** a model line is fitted to all the inliers 3D points. This model fitting technique, explained in the following pages, is completely different to the initial random model fitting, where only two points are used to define a line equation.
- **Inlier update:** All the 3D points are tested against this new line hypothesis and they are classified as inliers or outliers.
- **Line Score update:** using the new inliers and its distance to the new line equation, the line score is computed.

Once the iterative refinement is finished, the resulting line model (as a parametric equation), its score and its inliers list are stored in the candidates list. The inliers data of the found line model is not erased of the general point cloud.

Note that the refinement of two different but close random line hypothesis lead to the same best-local line.

To prevent this inefficient behaviour, once a new model line is found, it is checked whether it is already in the line candidate list. In that case, the refinement process only goes on if the current model score better than the already found similar line.

C.1 3D Line Fitting to inliers

The goal of this module is to find the line that best describe a set of 3D points (the inliers of the old-line model) tacking into account the weight of each point, defined here as the square root of its score.

It is performed using a least squares minimization technique where the errors are measured orthogonally to the proposed line and weighted by its score, i.e. orthogonal regression.

The 3D Line model fitting implementation is based on [16], an educational article by David Eberly.

Considering the individual 3D Point score in the model fitting is one of the keys of the whole line extraction algorithm, empowering a quick and accurate line improvement in the general Ransac Framework.

C.2 Similarity Check

The similarity check is a quick test to decide if two line equations are very similar and correspond, therefore, to the same real 3D line. It is an important tool to avoid computing the same line a lot of times in the line refinement process and in the selection of winner lines

The similarity check between two lines represented by a parametric equation is based on two simple comparisons: line to line distance and the angle of their director vectors. If both values are lower than a defined threshold, it will mean that both equations refer to the same real 3D line.

D Selection of winner lines

The proposed line extraction algorithm produces a list of candidate lines, each one represented by a parametric equation and with associated score and list of inliers 3D points.

These candidates must be analysed to extract the winner lines, that will be the landmarks of the new map.

The big challenges of this step can be summarized as:

- the score of the candidates is not uniform, mainly because there are longer and shorter lines.
- similar line candidates of the list could represent the same real line, (the similarity check only prevents worse similar lines entering the list but does not delete the worse old lines)
- Big lines with a lot of associated 3D points influence the line models of smaller lines.
- Lines that are defined by few points, even though these points have a large score, are difficult to find just by randomly selecting two endpoints.

Currently, two different methods have been developed. It is not already clear which one offers better and more efficient results.

Method A: Non-Maximum Suppression

Inspired by the non-maximum suppression commonly used in feature and object detections algorithms, this iterative process finds local maximum in the candidate list. Each iteration comprises two steps that are executed until there is no more elements in the candidate list or a predefined number of lines has been found.

1 – The line with maximum score is selected as a winner line.

2 – The winner line and all the similar lines are deleted from the candidate list. Similar lines are detected using the proposed similarity check, that considers line to line distance and parallelism.

An extra feature, is that, after the winner line are selected, it is checked if the inlier number of the winner line is too high in comparison with the other lines. In that case, the 3D points close to the winner line are erased from the point cloud and the whole line extraction is run again on the smaller point cloud.

Method B: Iterative Line Extraction with Winner Point erasing

The whole line extraction algorithm is run iteratively. After each iteration, a set of candidates' lines is found. The candidate line with a higher score is selected as a winner line, and all the 3D points close to this line are erased form the Point Cloud. This renewed, smaller point cloud is then used in the next execution of the line extraction algorithm.

This technique is much more robust than the naïve approach of just selecting winner lines inside the Line Extraction Algorithm without waiting to compute the whole candidate list. Note that, sometimes, a model line with a good score (even though it is refined) is not the best possible line. Classifying it as a winner and deleting directly the closer 3D points would be a big mistake.

On the other hand, in the iterative version of the line extraction algorithm, waiting for enough Ransac iterations guarantees that the best possible line has been found, and its related 3D points can be safely deleted.

Running the line extraction algorithm iteratively is expensive, however, in each iteration the point cloud becomes smaller, and the algorithm runs faster. Moreover, the basic Ransac iterations limits is now lower than in the standard non-iterative Line Extraction, as the goal is to find only the best line each time.

D.1 from line to segment

The output of the line Extraction Algorithm is a set of winner lines. Each line consists on a parametric equation, a score and a list of the 3D Points that support its model.

The map used by the tracking algorithm to localize the camera is, however, a set of 3D segments (with end points), not infinite lines. Therefore, the last step of the Mapping module is to convert the extracted lines into 3D segments.

For each line, all its associated 3D points are now projected into the line to obtain a 1D sorted list of projected points.

The endpoints of the segment will correspond, respectively, to a very low and high predefined percentile position

Note that, if the estimated lines are correct, it is not crucial that the endpoints match exactly the real 3D points. A small error will only mean slightly shorter or larger lines, with no effect in the tracking algorithm.

This version of the line extraction algorithm is still not able to distinguish between two different finite segments that belong to the same infinite line. This problem will be addressed in future version of the algorithm.

3. Results

3.1 Event Based Simulator in MATLAB

Building a new Event based simulator is a necessary step to develop and test the proposed tracking and mapping algorithms.

Modelling the asynchronous response, the high temporal resolution and the event generation model based on the local intensity change, is a challenging and time demanding task. Small simplifications in the time resolution or asynchronous clock could lead to a useless simulator that do not mimic the performance of the real sensor.

A good balance between accurate behaviour and easy implementation is achieved by considering the camera as an edge detector. Events are triggered by relative motion of the scene edges in the sensor plane. This approach has been already adopted with successful results in [7].

The simulator will receive as an input the camera trajectory (time, position and orientation) and a world representation as a set of 3D segments. The output will be an ordered list (by time) of events, represented as a tuple of three numbers: time stamp, and pixel horizontal and vertical coordinates.

Trajectory Generation

The trajectory is generated by a cubic spline interpolation of a set of defined key positions and orientations, represented with Euler Angles to perform the interpolation.

A scene visualization program has been developed. It allows the user to use a Graphical Interface Unit (GUI) to move the camera around the 3D virtual world, seeing in real time the projection of the world (lines) in the sensor plane and the horizontal and vertical projection (2D) views of the camera position, orientation and world lines.

This user-friendly approach offers an easy way to define the set of control points. The components of the state of the camera (position and orientation) are individually interpolated at time query points, spaced a small-time interval. This time interval will be the temporal resolution of the sensor, as the events will be generated for every interpolated state.

World representation

The world (scene) is represented as a set of 3D segments (lines with a start and end). This representation describes is able to describe scenes where lines are de dominant features.

Event generation

Given a world representation and a trajectory, the following routine will generate the events output. Each event will be a three-element tuple: time stamp, horizontal and vertical coordinates, as the polarity of the intensity change is not used by the algorithm.

For each interpolated state, all the 3D segments are projected into the sensor frame to obtain a set of 2D segments. For each of these 2D segment, several random 2D points (that belong to the segment) are selected and will become the generated events.

This method accurately reflects the real sensor behaviour: as one line moves in the sensor frame, it will generate new events in all the pixels that "cross the line".

Following the event generation model of an event camera, the number of events generated by each line (random points) is not constant, it depends on:

- The length of the 2D segment in the senor plane (longer lines will produce more events as they are crossed by more pixels)
- The relative velocity of the line with respect to the camera, projected into the sensor plane. All the segment is assumed to have the same relative velocity (taking into account camera translation and rotation) as its medium point.
- The angle between the 2D line and the projection of the previously defined relative velocity. If they are parallel, no events will be generated, as there will be no apparent changes in the sensor plane. Therefore, a parallel attenuation factor is introduced.

The event stream generated by this method is consistent and similar with available event dataset taken with a real event based camera [17].

Limitations

The overall performance of the simulator is considered satisfactory and it fulfils its requirements: a tool for the first stages of SLAM algorithm development.

However, it presents some limitations and there are minor aspects where it may differ from the behaviour of the real camera. In further version, they will be corrected or improved.

- Pixel resolution is not considered. This means that event pixel can be any real number inside a range, no necessarily integer values. This means that the simulated camera has infinite resolution.
- Time resolution is limited by the time interval chosen in the interpolation.
- It cannot handle occlusions of lines, as no solid planes have been already defined.

The Simulator clearly have better pixel resolution and worse time resolution than real cameras. There is a duality between these two resolutions. In the real camera, the event will only be fired in integer position value, but exactly in the correct time when the pixel crosses the edge. In the simulator, the event is fired at a given time, but exactly in the good point (with subpixel resolution).

3.2 Tracking and Mapping Results

In this section, the designed mapping and tracking are tested in three Synthetic Datasets: Office, Planar Scene and Geometric Bodies, inspired by real 3D situations where the application of an Event based Camera has a lot of potential to autonomous localize robots

Both algorithms are tested independently on the same 6 DOF trajectories. Given a known scene, the mapping algorithm will estimate the trajectory. On the other hand, and independently, the mapping algorithm will try to build a map of the environment, (using the ground truth trajectory).

The results are presented visually with self-explanatory plots and figures where the comparison between estimated and real values becomes clear. Note that no numeric results or tables are provided, as, in this case, the graphic material offers a better way to evaluate and understand the performance of the algorithms.

At the end of this section, a short and general discussion is presented. Both the strengths and weakness of the algorithm are highlighted and the future work is outlined.

Office Dataset

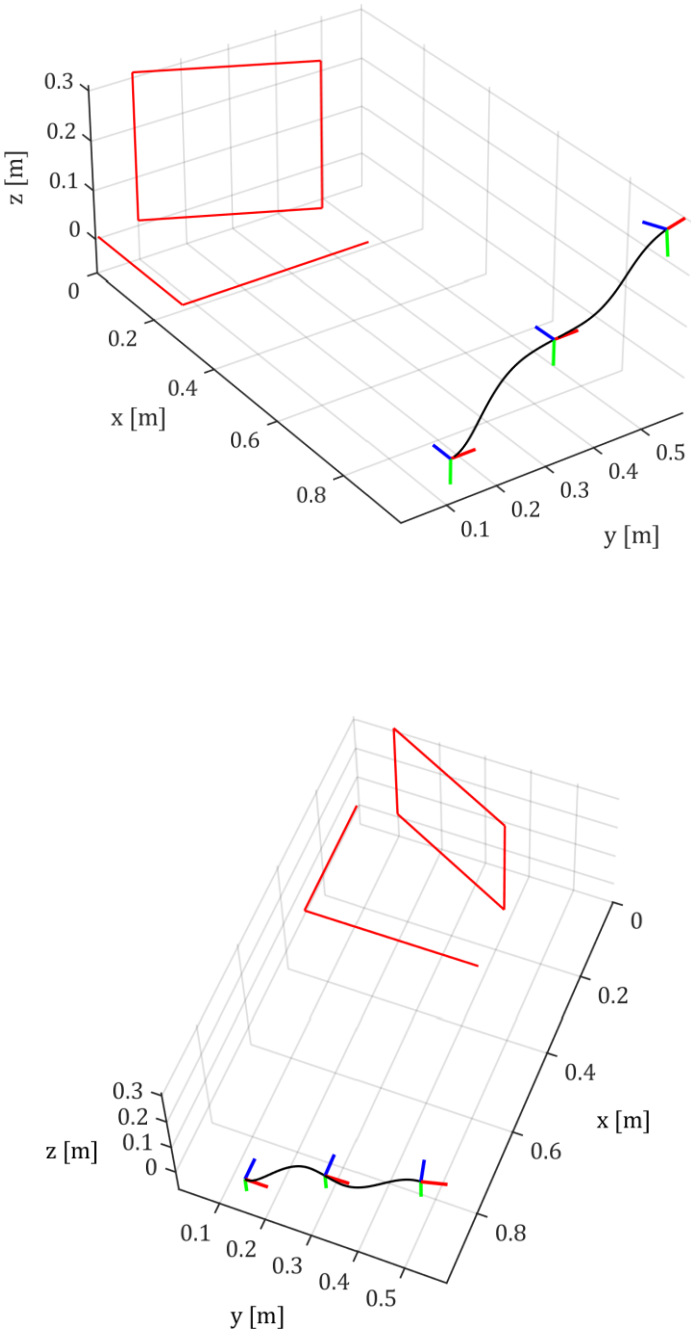


Figure 3.1 Office Dataset

Mapping Results - Office

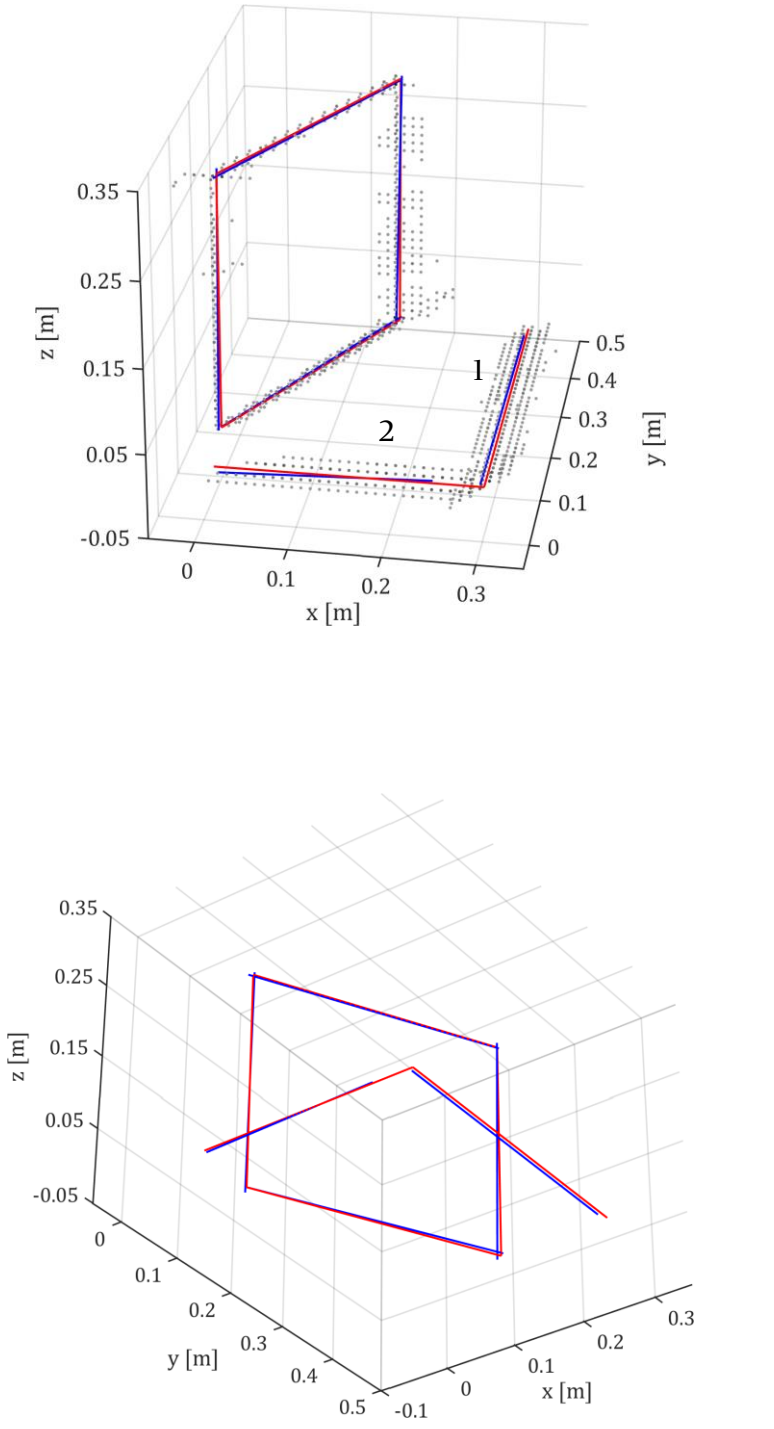


Figure 3.2 Mapping results - Office

In the office dataset, the 3D Scene represent the table corner and a computer screen, with lines in all directions of the 3D space.

The chosen diagonal trajectory (up, right and slightly forward) (Figure 3.1) enables the observation of all lines. The main challenge is that there all lines in all possible directions. The segment parallel to the x axis is always small in the sensor plane, and therefore, more difficult to estimate (number 2, Figure 3.2)

Although the Point Cloud still shows some uncertainty in the less observable directions, the line extraction algorithm is able to accurately estimate all the 3D segments of the scenes.

The algorithm works perfectly because all lines are much longer than their uncertainty and there is no overlap.

Comment on Winner Line Selection

The line number 1 (Figure 3.2) is much longer and, due to its uncertainty, has a lot 3D points associated. It could have a strong influence on other lines models. Once it is found, the algorithm automatically deletes its associated 3D points and the Line Extraction algorithm is run again on the smaller point Cloud.

This strategy avoids contaminating the models of smaller and close lines (Figure 3.3)

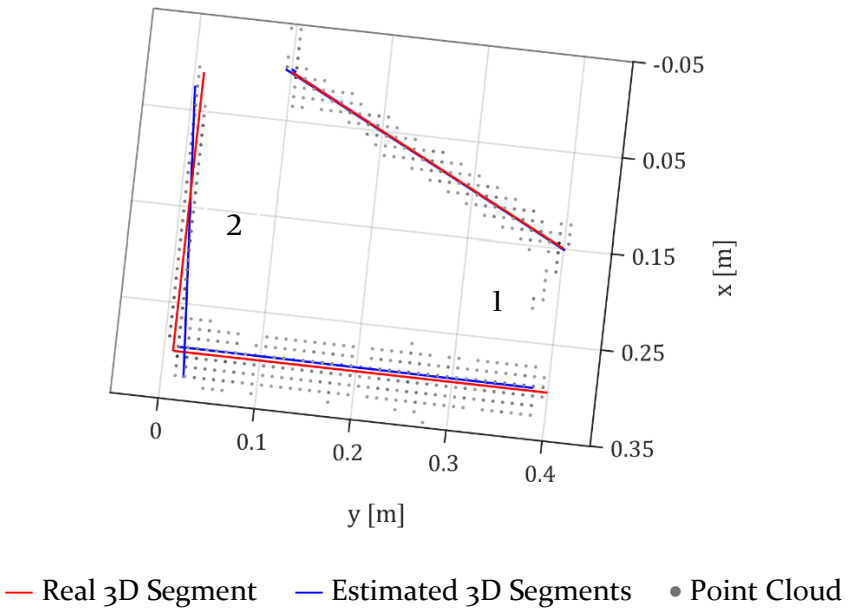


Figure 3.3 . Estimated segments if the winner line is not removed (automatically) from the point Cloud.

Tracking Results – Office

Note: The tracking algorithm is evaluated in the same trajectory.

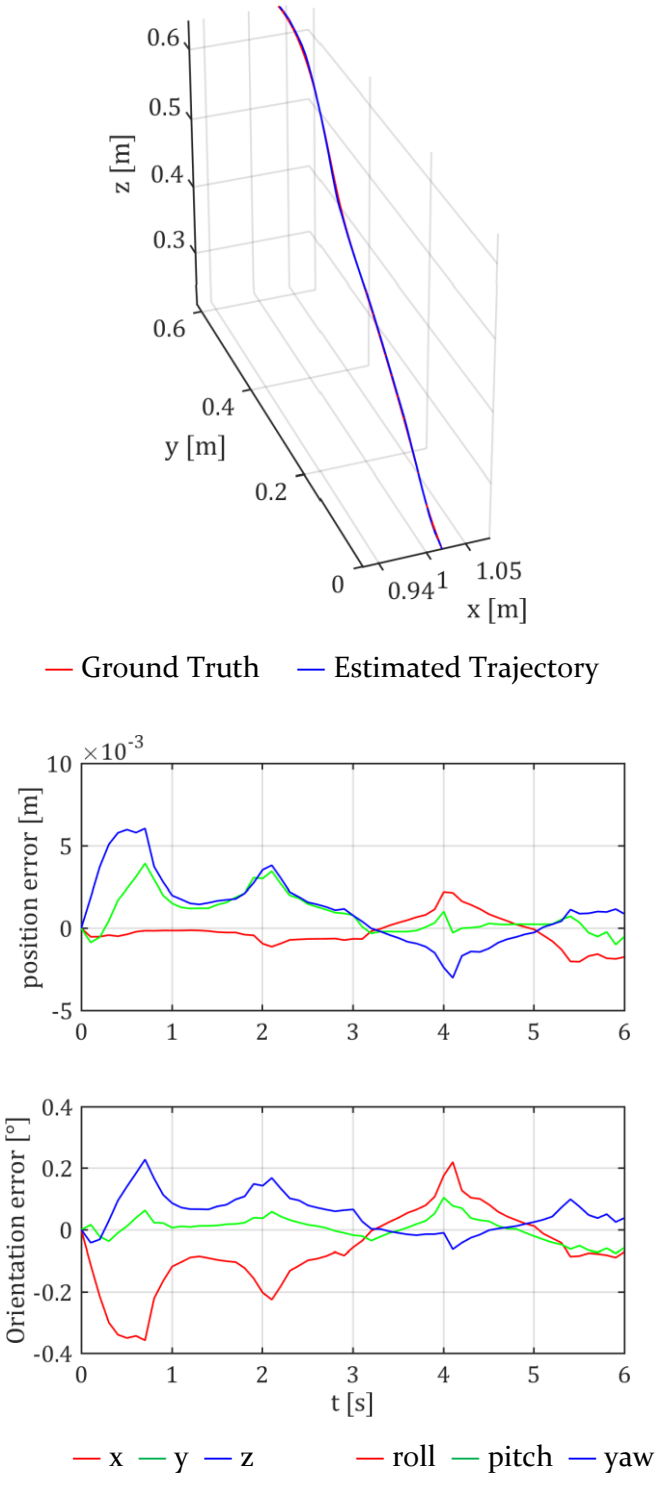


Figure 3.4 Tracking Results Office

Planar Scene Dataset

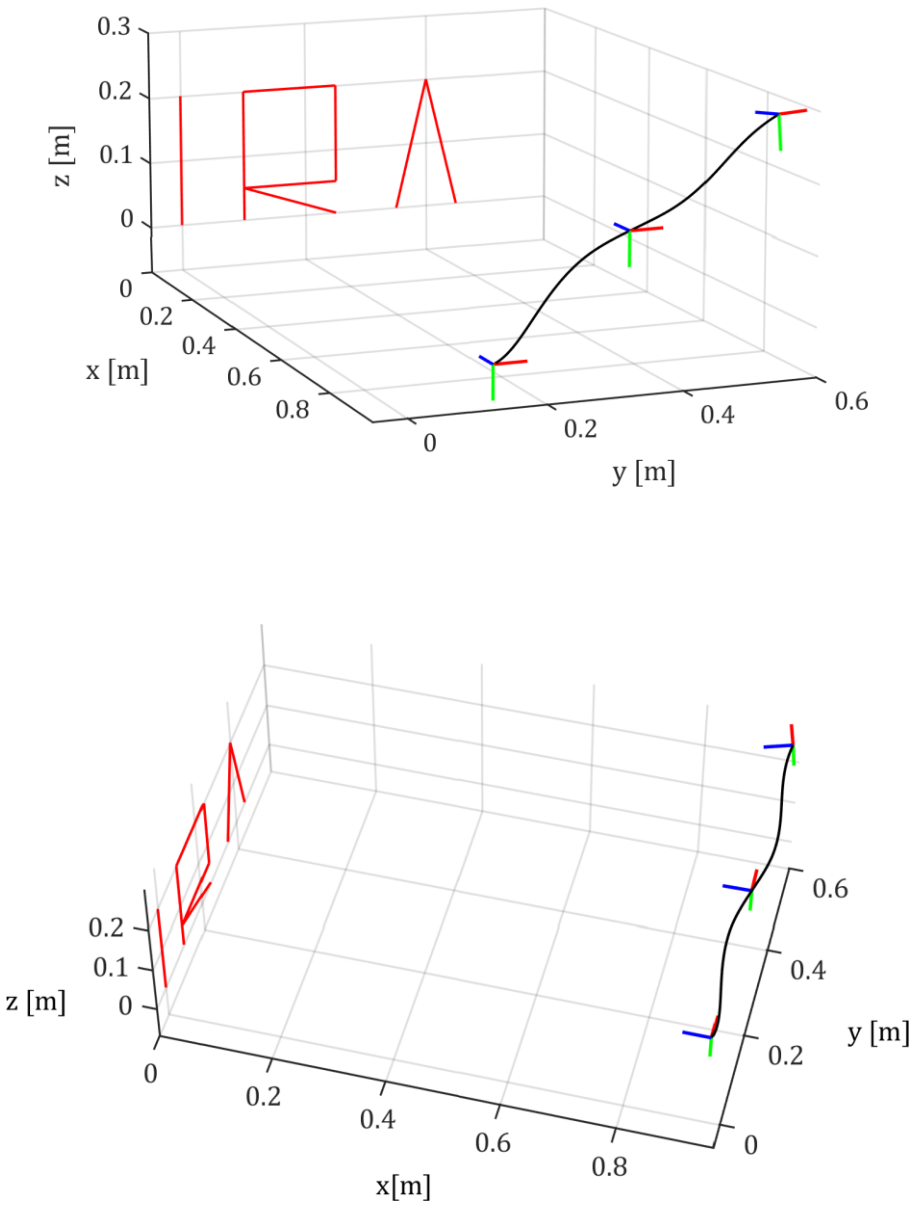


Figure 3.5 Planar Scene Dataset

Mapping Results – Planar Scene

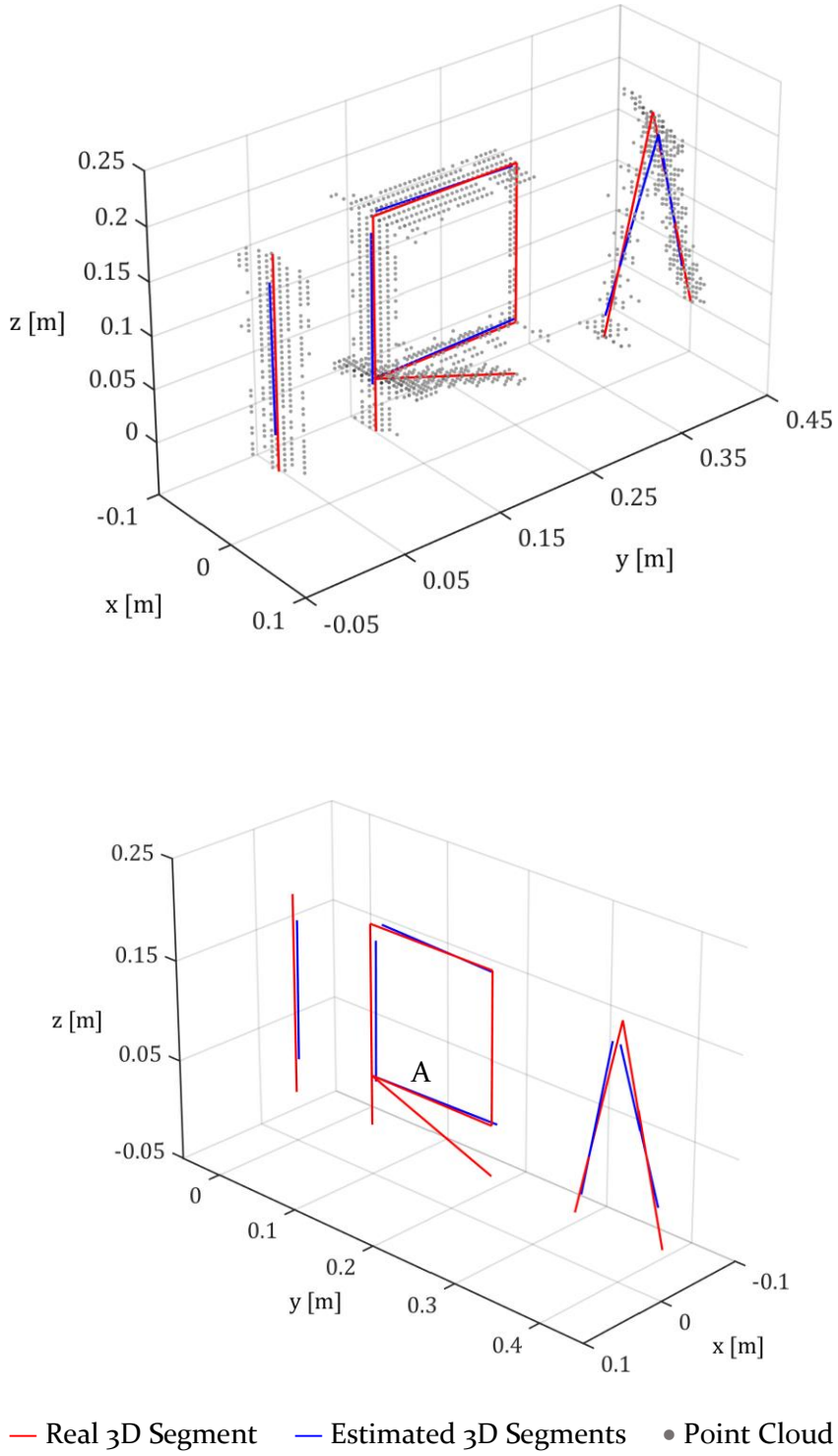


Figure 3.6 Mapping Results – Planar Scene

In the planar scene dataset, the 3D Scene represent a drawing on a poster. The algorithm does not know a priori that it is a planar scene, nor the distance to the plane that contains all the lines.

The chosen trajectory is again diagonal to partially observe all the lines. The main challenge is the letter R, where one 3D Point (Point A, Figure 3.6) is the intersection of 4 different segments.

In this case, the point cloud shows a lot of uncertainty. This uncertainty is, in some cases equal to a quarter of the segment length. However, the proposed line extraction algorithm is still robust enough to estimate correctly most of the 3D segments of the scenes.

Although any wrong line is found, the position errors are bigger than in the office dataset. The intersection of various lines in point A is not interpreted correctly.

Comment on: From line to segment.

The algorithm searches infinite line in 3D space. When the prolongation of a 3D segment intersects with other segments, all the 3D points close to a (infinite) model line are considered inliers.

Although this problem remains unsolved, in the line to segment conversion, inlier points are ordered and conservatives percentiles positions are selected as the end points of the segment. This strategy partially avoids worse results (Figure 3.7.)

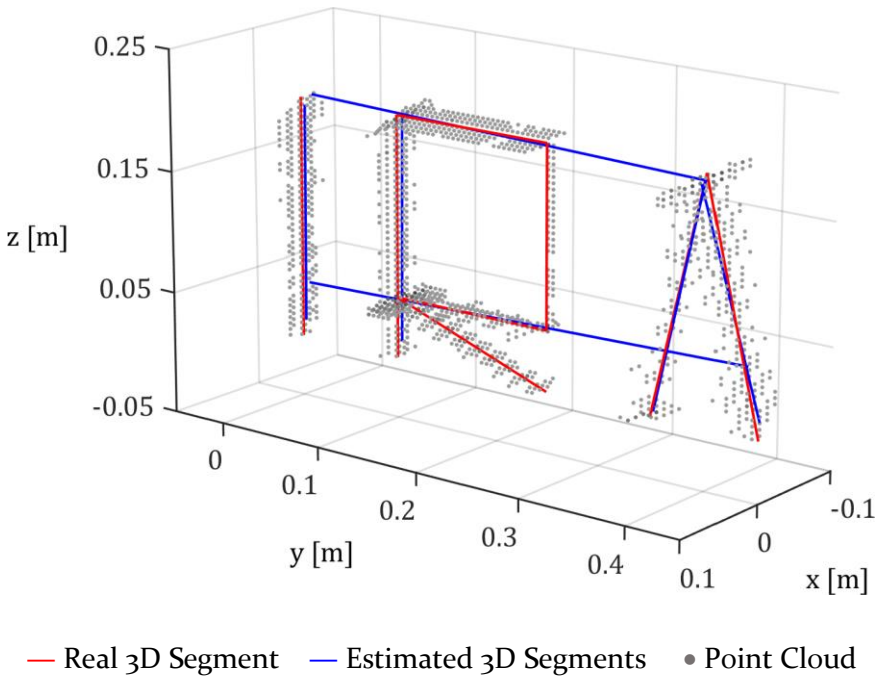


Figure 3.7 Estimated segments if the no precaution in the line to segment conversion is taken.

Geometric Bodies Dataset

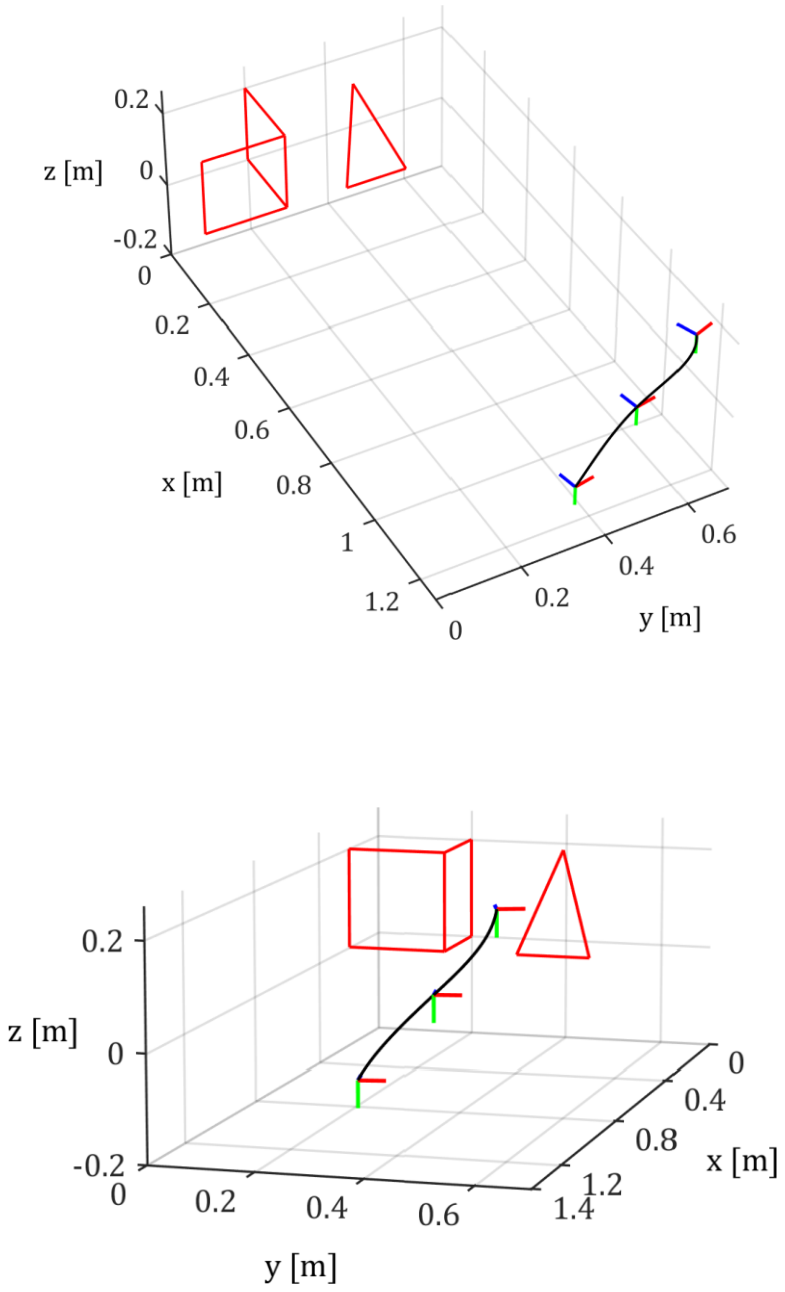


Figure 3.8 Geometric Bodies Dataset

Mapping Results – Geometric Bodies

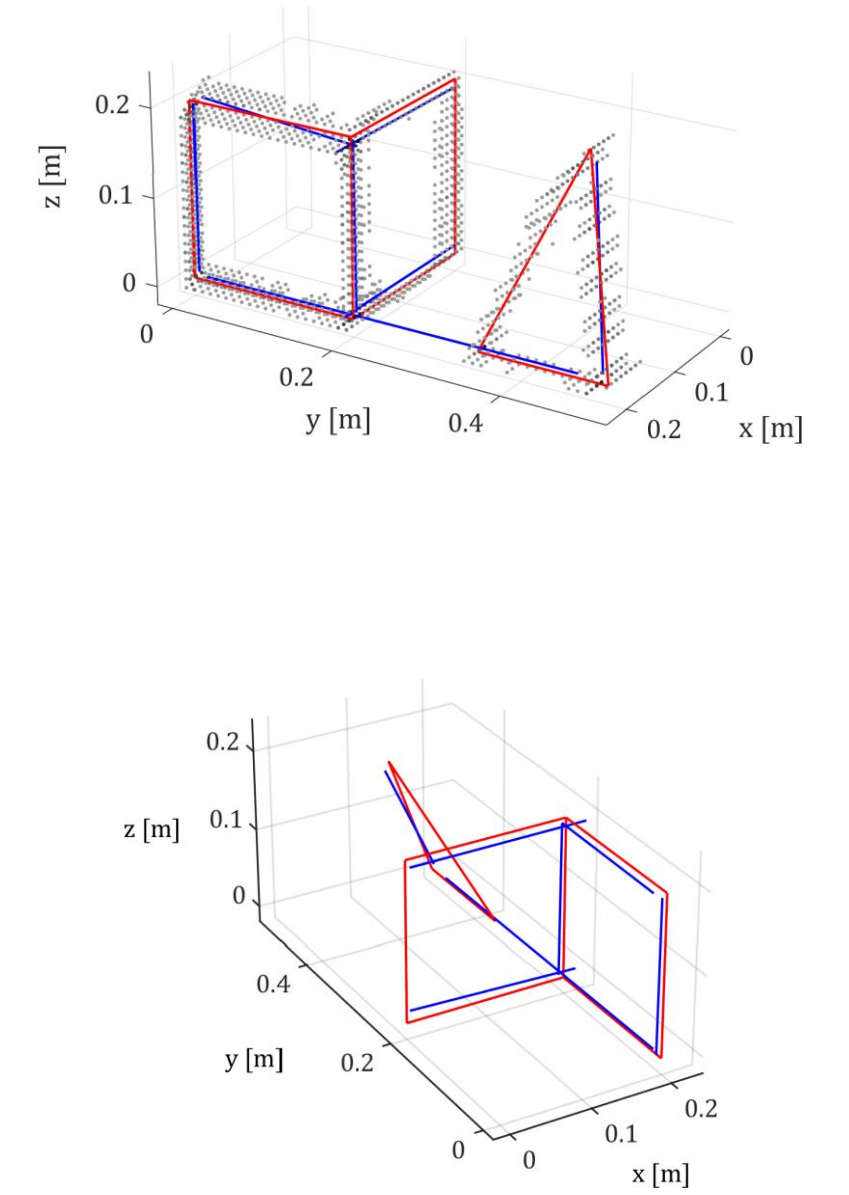


Figure 3.9 Mapping Results – Geometric Bodies

The chosen trajectory is again diagonal to partially observe all the lines. This is the harder scenario, with lines in all the directions, 3 lines intersections and small "open" spaces.

The uncertainty is again big related to the segment dimensions. The algorithm finds most of the lines and makes no wrong models. Although position errors are not negligible, the structure of the scene is successfully estimated.

In the intersections, each line model is deviated by the other lines and the base of the cub and the triangle are estimated with the same segment. Note that this behaviour has already been discussed in the previous datasets.

Comment on: Uncertainty robustness

In all the three datasets, the line extraction algorithm has been able to extract 3D segments from points cloud with a lot of uncertainty in the less observable dimension. However, this robustness has a limit and it request the principal line direction to be longer than "it uncertainty". If some Ransac parameters (threshold, minimum distances ...) are changed from the used in Figure 3.9, the line extraction algorithm tends to make a mistake (Figure 3.10).

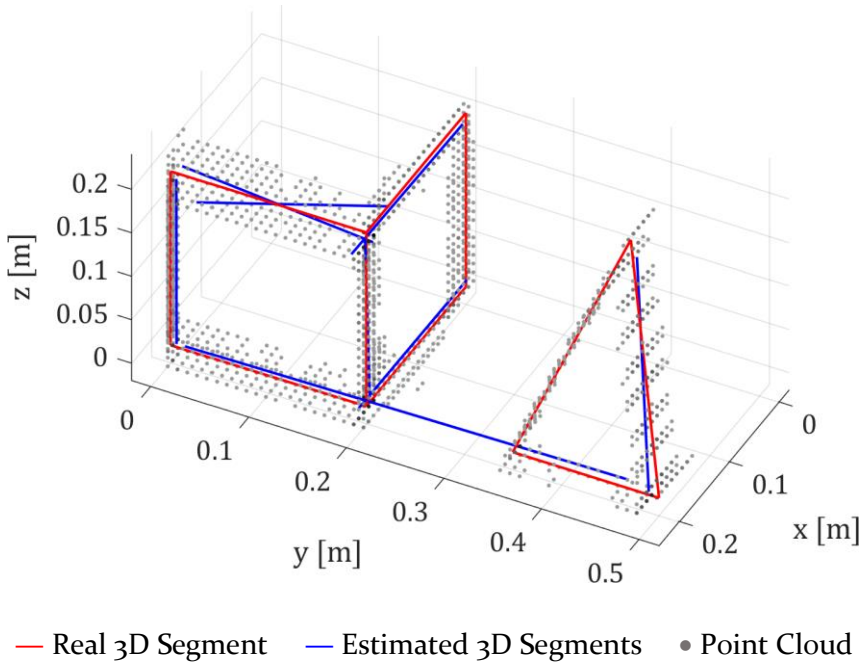
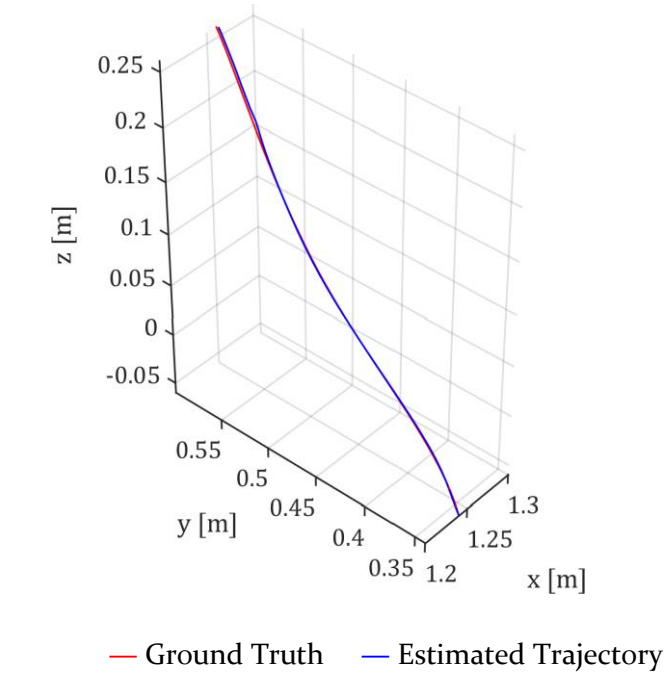


Figure 3.10 In this dataset, the Line Extraction Algorithm partially fails if not tuned correctly.

Tracking results – Geometric Bodies



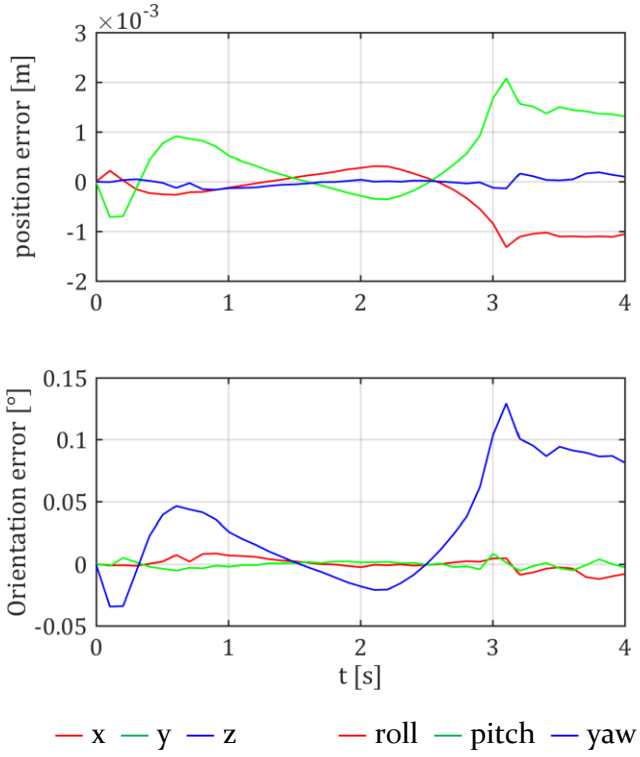


Figure 3.11 Tracking Results – Geometric Bodies

The tracking algorithm estimates the ground truth trajectory with a high degree of accuracy. This estimated trajectory could be used in the Mapping algorithm, instead of the Ground Truth, and the same results would be obtained.

A more challenging, closed loop, trajectory is now tested in this environment. The results are also very accurate. The error increases in the part of the trajectory with higher curvature because the motion model alone its predictions.

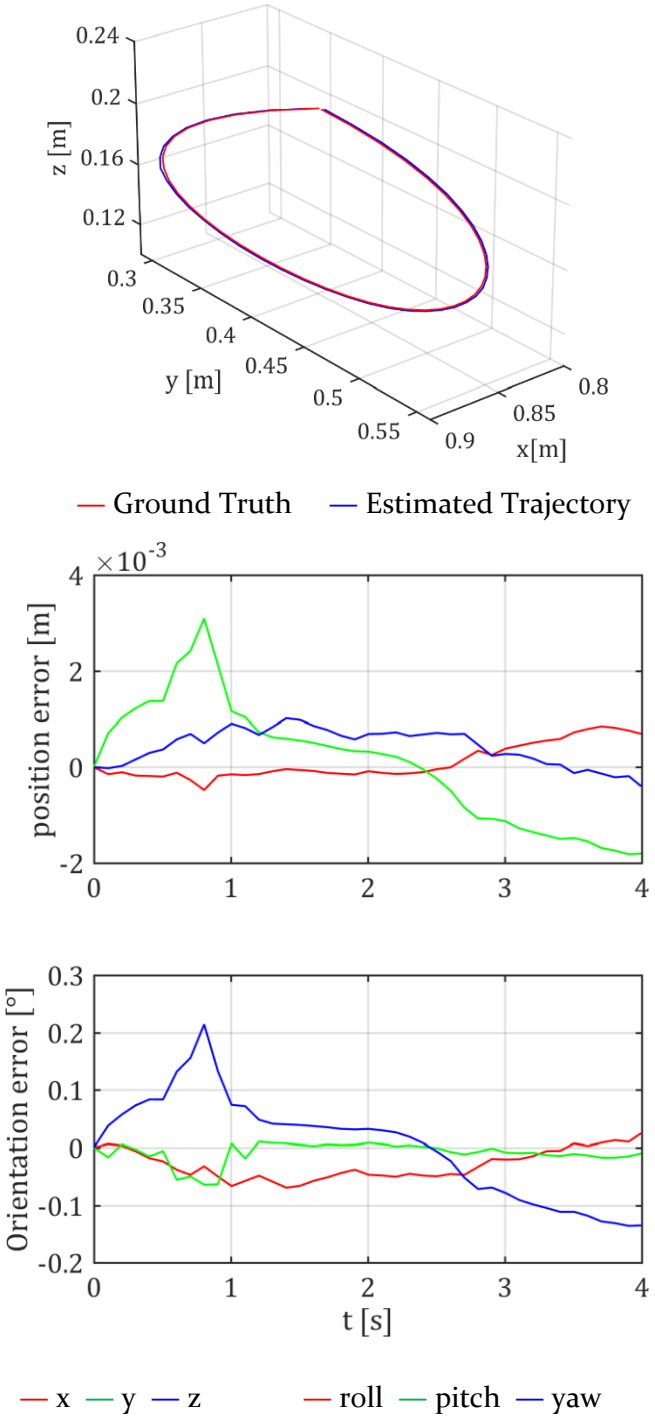


Figure 3.12 Estimated and Ground Truth Trajectory

3.3 Discussion and future work

The tracking algorithm has shown an exceptional performance in the synthetic datasets. The new map could have been built using the estimated trajectories, instead of the ground truth, as they are almost identical. Therefore, the first step of Parallel Tracking and Mapping is already solved (estimate a trajectory given a map).

However, this performance is aided by the no-noise behaviour of the simulator. To prepare the algorithm for real operation, new strategies to deal with the sensor noise and the incoming events from unknown edges should be designed and added to the basic event-per-event EKF algorithm.

Implementing the event per event tracking in real time will also be a challenge, requiring the implementation of little tricks to boost speed. If it were impossible to use a constant motion model, a simple constant position model will be studied.

The results of Mapping algorithm are also encouraging, but a lot of more work is essential to bring it to life in a real sensor. In the first place, the grid choice should be revised, as the current cartesian grid does not fit the natural behaviour of a monocular camera.

The line extraction algorithm is one of the big contributions of this thesis and has proved to be efficient and robust to a reasonable amount of uncertainty. Its ability to fit different line model to an unstructured point cloud will be key to perform Landmark based Slam with and event based camera.

Unfortunately, the proposed algorithm requires a big parallax so that lines are, at least, minimally defined, and cannot deal with complete undetermined lines.

One important a priori known piece of information is, yet, not been considered: the unobservable dimension, defined by the known trajectory of the camera. Adding this information in the point cloud building process could be the solution to the current difficulties.
Once the current Mapping Algorithm can build a map even with low parallax (small change of point of view) the core of the parallel tracking and mapping will be ready.
However, two important elements are still missing:

- Bootstrap strategy: Just at the beginning, there is no map for the tracking and no trajectory for the mapping thread. Different strategies such as the observation of a known feature or start looking to a planar surface will be studied.
- Map Optimization: If each map build by the mapping thread replaces the old map, the system will drift and finally fail. Ideally, the news maps should improve, instead of replacing the global map, through an optimization process. A new and more complex map representation modelled as a gaussian variable will also be studied.

All these new ideas will be tested during the next month (July 2017)

4. Budget, impact on society and project planning

Budget

This project has been supported by an undergraduate research Internship. The project budget can be calculated directly with the Internship salary: 525 €/ month for 4 months.

In these first stages of the project, the real camera has still not been use. Once the algorithms work perfectly in a simulated environment, the real sensor will be necessary to implement and test the proposed method.

Event cameras are still expensive sensors. The newest version of the company IniLabs, called DAVIS240C is currently used by most of the researched groups and costs around 6.000 €. The DVS128, is a cheaper and older sensor with low resolution and costs around 3.000 €

Impact on Society

The research in Event Based Camera, and SLAM in general, will enable robots to autonomously localize itself and navigate through the environment. A well-known example are autonomous cars, that will reach the market in the following years completely changing our transport system.

Event Based Camera have a lot of potential in fast robotics, for example autonomous flying robots. These robots will be used, for example, to operate in search and rescue environments, where direct human intervention is dangerous or inefficient. On the other hand, event based cameras will be also used, inevitably, in military applications.

Project Organization

This project has been developed in the framework of a Bachelor Thesis for 4 months. It has been supported with an undergraduate Research Internship with 20 work hours per week. The project planning is represented with a Gant Chart.

	MARCH	APRIL	MAY	JUNE
Introduction to SLAM				
Literature Research				
Graph Slam for event vision				
Tracking Algorithm				
Mapping Algorithm				
Results				
Thesis writing				

Conclusion

Event based cameras have opened a new paradigm for real time vision, as now, instead of fixed static frames, the sensor sends an asynchronous sequence of events, triggered in the pixels that sense a local intensity change with microsecond resolution.

Standard computer vision techniques fail in this new dynamics and new algorithms are needed to solve the SLAM Problem. We have presented two new tracking and parallel algorithms, designed to work in parallel in line based environment.

The tracking algorithm works perfectly in simulation, where some of the big challenges of the real world, such as noise measurement, have not still ben modelled. However, the results prove that the algorithm has a lot of potential, and we are enthusiastic about testing them in the real sensor.

On the other hand, the mapping thread is more complex and still needs some refinement to success in the real world. The main problem is the uncertainty in the unobservable dimension of the camera. This problem has been solved in simulation by seeing all the lines form different points of view. Unfortunately, this cannot be assured in a real, random motion. Therefore, the algorithm is still not robust enough against this problem. We already have some ideas to improve the Mapping thread, that will be tested soon.

The proposed algorithm will work in line based environments, losing the generality of some already proposed algorithms. However, this landmark based (3D lines) approach is new in the research community and remains an appealing, yet unknown field. Moreover, the assumption of line based environments holds for most of the indoor human scenes, where fast robots will need to localize itself while doing aggressive manoeuvres.

This landmark based map enables and easy place recognition, that will avoid the drift of another visual odometry system.

To truly build the Parallel Tracking and Mapping, two additional modules must be still designed and implemented: a bootstrap engine, capable of starting the when there is no map, nor trajectory estimated a way to improve a global map form the individual maps extracted by the iterations of the mapping thread.

One of the key ingredients of the mapping thread is a completely new line extraction algorithm, that works directly in unstructured 3D point clouds, by fitting a set of 3D segment into this Point Cloud. This algorithm has also a lot of potential on its own, and its philosophy can be applied to solve other model extraction problems or to work with efficiently point clouds.

References

[1] J. Solà, "Simultaneous localization and mapping with the extended Kalman filter," 2014.

[2] J. Solà, "Course on SLAM," 2017.

[3] G. Grisetti, R. Kummerle, C. Stachniss and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine,* 2010.

[4] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE transactions on pattern analysis and machine intelligence,* 2007.

[5] J. Civera, A. J. Davison and J. M. Montiel, "Inverse depth parametrization for monocular SLAM," *IEEE transactions on robotics.*

[6] P. Lichtsteiner, C. Posch and T. Delbruck, "A 128 128 120 dB 15 us Latency Asynchronous Temporal Contrast Vision Sensor," 2008.

[7] H. Rebecq, T. Horstschaefer, G. Gallego and D. Scaramuzza, "EVO: A Geometric Approach to Event-Based 6-DOF Parallel Tracking and Mapping in Real Time," 2017.

[8] H. Rebecq, G. Gallego and D. Scaramuzza, "EMVS: Event-based Multi-View Stereo," *Bristish Machine Vision Conference,* 2016.

[9] H. Kim, S. Leutenegger and A. J. Davison, "Real-time 3D reconstruction and 6-DoF tracking with an event camera," 2016.

[10] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," 2007.

[11] J. Solà, "Quaternion kinematics for the error-state Kalman filter," 2017.

[12] R. T. Collins, "A space-sweep approach to true multi-image matching," p. 1996.

[13] J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing".

[14] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model fitting," 1981.

[15] P. Smith, I. D. Reid and A. J. Davison, "Real-Time Monocular SLAM with Straight Lines," 2006.

[16] D. Eberly, “Least Squares Fitting of Data,” 1999.

[17] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck and D. Scaramuzza, “The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM,” 2016.