MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Real-time recalibration of pan-tilt-zoom camera

Master's Thesis

DANIEL MÚČKA

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Real-time recalibration of pan-tilt-zoom camera

Master's Thesis

DANIEL MÚČKA

Advisor: doc. RNDr. David Svoboda, Ph.D.

Department of Visual Computing

Brno, Spring 2025

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed in complete reference to the due source.

During the preparation of this thesis, I used the following AI tools:

- Grammarly for grammar checking,
- Github Copilot for faster code writing.

Daniel Múčka

Advisor: doc. RNDr. David Svoboda, Ph.D.

Acknowledgements

First, I would like to thank Barbora Bühnová for accepting this thesis. Her remarks helped me to shape the thesis into its current form. Then, I would like to thank Miroslav Krajíček for his guidance in the development of the method, and for giving me the opportunity to work on this thesis, and David Svoboda for his continuous support, patience and insight. Last, but not least, I thank my family, friends and colleagues for their support during my studies.

Abstract

Sports, such as soccer, attract viewers from all around the world. The current technologies allow broadcasters to offer augmented reality by overlaying projections, advertisements, line markings or player information over the observed video frames to increase engagement with fans, and assist referees to better recognize particular situations. The use of such annotations requires a camera with calibrated position, rotation and focal length. During the match, the camera is aimed and zoomed continuously, causing the calibration to invalidate. Because of this, the camera needs to be recalibrated per every change of the pose. In this work, we applied a filtering technique used in robotics to track changes of the camera pose for every frame in real-time. We implemented the novel method into a library and a client-server application. The performance and accuracy was then assessed and compared with the state-of-the-art.

Keywords

sports, camera, real-time, visual odometry, computer vision, optical flow, particle filter, robotics, augmented reality

Contents

Introduction **1**

1 State of the art **2**

2 Background **4**
2.1 Problem analysis 4
2.2 Camera model 5
2.3 Camera recalibration 8

3 Proposed method **10**
3.1 Robotics methodology 10
3.2 Bayes filters 11
3.2.1 Gaussian filters 14
3.2.2 Kalman filter 14
3.2.3 Extended Kalman filter 16
3.2.4 Nonparametric filters 17
3.2.5 Particle filter 18
3.2.6 Sequential importance resampling 21
3.2.7 Bootstrap filter 22
3.3 Motion model 27
3.3.1 Optical flow 27
3.3.2 Motion analysis 28
3.3.3 Pan-tilt movement 29
3.3.4 Zoom movement 30
3.3.5 Movement detection 31
3.3.6 Application 33
3.4 Measurement model 34
3.4.1 Feature matching 35
3.4.2 Template matching 35
3.4.3 Application 37
3.5 State estimation 39

4 Implementation **40**
4.1 Technology stack 42
4.2 Development 44
4.3 RePTZ library 45

4.4 Demo 47
4.5 Architecture 49
4.5.1 Server / CLI 49
4.5.2 Client 50

5 Evaluation **52**
5.1 Methodology 52
5.2 Results 52

6 Conclusion **55**

Bibliography **56**

A Method evaluation summary **62**

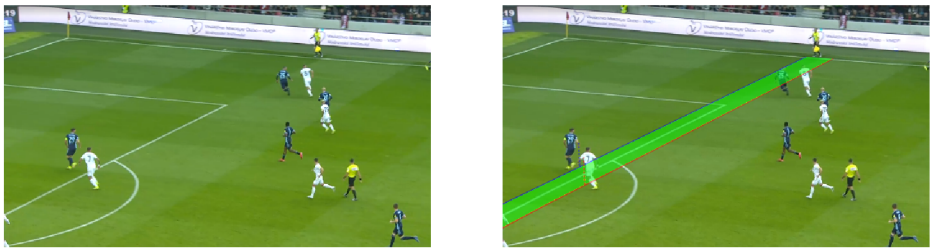
B Detailed statistics **64**

Introduction

Nowadays, technology and sports go hand in hand. Depending on the game rules of the sport in question, the ability of a human referee to properly decide whether a rule has been broken is getting increasingly difficult. Such is the case when examining an offside situation in soccer; an out happens in volleyball or when the bowler missteps over the line in cricket. (*e.g. in soccer, the result of an offside situation can be altered by millimetres difference.*) Sport is a part of the entertainment industry, and its consumers — the viewers and the fans, look up to the outcome of a match. The need to correctly and entertainingly inform the public is paramount to any venue or sports business. As a result of this need, organizations overseeing sports and their leagues (e.g. FIFA) endorse advanced technological solutions, such as VAR (Video Assistant Referee), VOL (Virtual Offside Line), ball tracking, player skeleton tracking, and more. Because the enjoyment of sports is visual by nature, we employ computer vision algorithms and analysis processes to simplify referees' decision-making and improve the clarity of the sport for viewers.

The focus of this thesis was to design and implement a method for soccer matches that annotates video data stream with pose estimations, resulting in continuous recalibration of the camera. The work has been done in cooperation with **GOAL SPORT technology s.r.o.**, with future plans to incorporate it into their VAR and VOL systems.

In the beginning, the current state of the art is analysed, and the usefulness of the existing solutions is evaluated. Then, we define the camera model and the recalibration problem. The proposal for a new method is deeply connected with the field of robotics, which we explain in the robotics methodology section. Afterwards, the algorithms used in the method are presented. In the fourth chapter, we present the implementation — the project structure and how the system should be integrated into other applications. We follow with testing and evaluation of our solution with ground truth and the state of the art. The work concludes with the sixth chapter, summarising the performance and listing possibilities for future development.



(**a**) Offside situation (**b**) VOL evaluation

Figure 1: An example showing an offside situation in soccer. VOL system was used in (b) to show the situation clearly to the referee and/or the audience.

1 State of the art

As stated previously, the calibration of cameras in sports is a common problem, and many sources are available regarding the localization and mapping of a 3D space using a pan-tilt-zoom (PTZ) camera — a manual-operated camera allowing horizontal and vertical movement and zoom. Significant effort has been made by researchers since 2023 in the papers *No Bells, Just Whistles: Sports Field Registration by Leveraging Geometric Properties* [1], *TVCalib* [2], *Pan-tilt-zoom SLAM for Sports Videos* [3] and *BroadTrack* [4]. A different approach is presented in each paper, using segmentation methods, homographies, geometric properties, neural networks, filters and combinations of them. At the time of writing, the only publicly available paper with a working code that provides the solution we seek is *Pan-tilt-zoom SLAM for Sports Videos* by Jikai Lu, Jianhui Chen and James J. Little [3]. We analysed their implementation and verified the viability for use in production systems.

In robotics, SLAM stands for simultaneous localization and mapping — the system tracks the current PTZ parameters (localization) and holds a map with features to relocalize itself when the tracking is lost. The **PTZ-SLAM** method uses ray reprojections instead of 2D to 3D correspondences to track keypoints on the field, complemented by a mapping component based on decision trees. The camera state is predicted using EKF (Extended Kalman Filter). The implementation is

provided in the Python language, along with testing data. The method should work as long as the initial calibration pan-tilt-zoom parameters are provided, regardless of the type of sports field.

The code was tested and compared on real videos of soccer with manually annotated ground truth frames provided by **GOAL SPORT technology s.r.o.** The results in figure 1.1 show a slight difference between PTZ parameters. Additionally, it can be seen that the average computation time for a single frame is almost 300 ms. This method is, therefore, unsuitable for real-time production. According to ch. 4.2 [3], the speed can be improved by a complete rewrite into C++ and by the use of a dedicated graphics card for the calculation of feature descriptors.

Instead of rewriting the state-of-the-art solution, we decided to try an entirely different approach, which has not been tried in any of the papers, using the particle filter used in robotic SLAM systems.

The problem and the method are mathematically defined in the following sections.

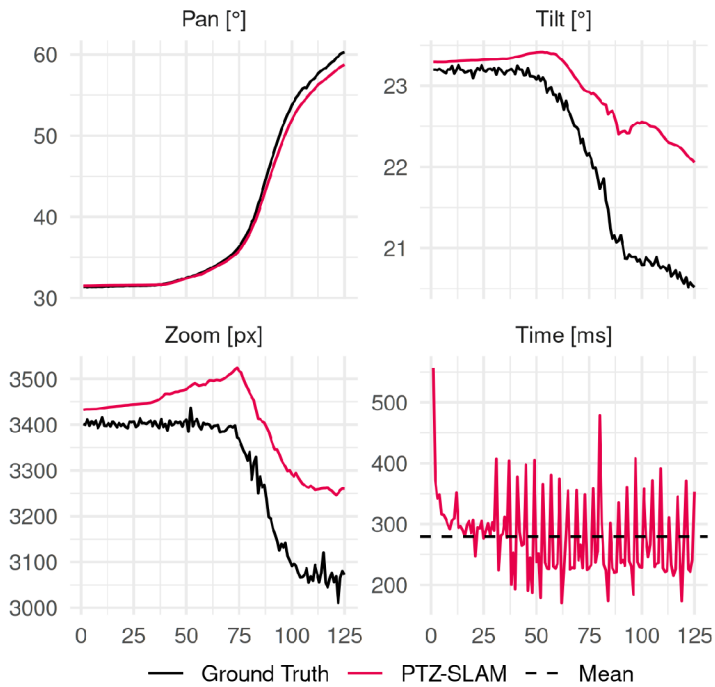


Figure 1.1: Comparison of PTZ-SLAM and the ground truth. Testing has been done on 125 frames of video on a Ryzen 7 8700F CPU.

2 Background

The concern of this chapter is to state the problem parameters and to analyse the options for achieving augmented reality for sports cameras pointed at a soccer pitch.

2.1 Problem analysis

To provide a continuous stream of camera calibration data, the expected outcome of the algorithm is a series $\mathcal{S}_n = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$ of parameter estimations vector $\hat{x}_n = (\hat{\theta}_{pan}, \hat{\theta}_{tilt}, \hat{\theta}_{zoom})$ tracking real camera movement for each input frame F_n. Pan and tilt values are measured in degrees, and zoom is measured in camera-chip calibrated pixel units[1].

As stated in the introduction, several assumptions about the situation are made:

1. There are PTZ (pan-tilt-zoom) cameras[2] directed at a soccer pitch.
2. The soccer pitch is standard-sized.
3. Camera can be manually operated, altering the pan, tilt, or zoom (cannot roll).
4. Camera cannot physically move (i.e. no translation occurs).
5. Camera is initially calibrated (i.e. we have the initial intrinsic and extrinsic parameters).
6. Camera output is a non-interlaced video of reasonable resolution with a standard framerate (e.g. *1080p50, 720p25*).

The environment of a live sports venue is very dynamic, therefore, a robust algorithm must:

- withstand changes in brightness, contrast and object or player movement over time,

1. Pan is equivalent to yaw, tilt to pitch and zoom to focal length.
2. In literature also referred to as sports camera.

- to keep correct tracking even after rapid camera movements,
- in case the tracking is lost, regain control without manual intervention,
- and be fast enough so the consumer application seems real-time.

Ideally, the employed method should be exceptional in all points. An example model of the soccer pitch is presented in fig. 2.1 and the possible camera movements are explained in fig. 2.2.

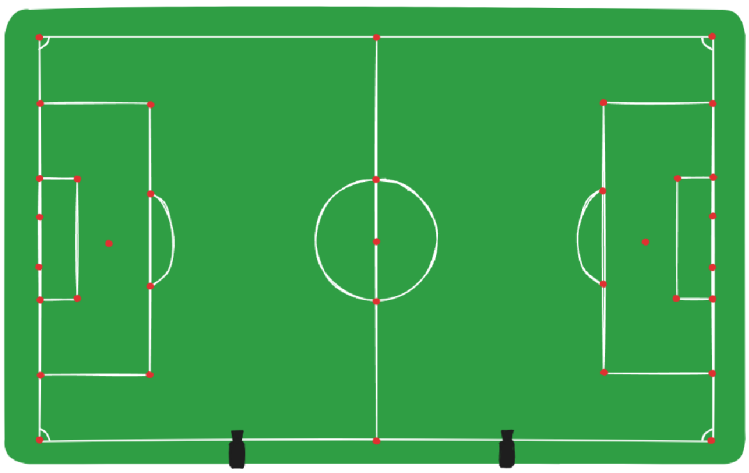


Figure 2.1: FIFA standard soccer pitch. Red circles specify important markings (keypoints). They represent line crossings, circle centres or other standardised points [5]. Black camera models depict example camera positions.

2.2 Camera model

First, to obtain new camera calibration parameters, we have to define a specific camera model that defines how 3D points map to 2D and vice versa. We use the *pinhole camera model* (fig. 2.3) because the initial camera calibration is given by **GOAL SPORT technology s.r.o.** in the form of intrinsic and extrinsic parameters for this model.

The pinhole camera model is defined as a function between Euclidean spaces $\mathbb{R}^3 \mapsto \mathbb{R}^2$, such that:

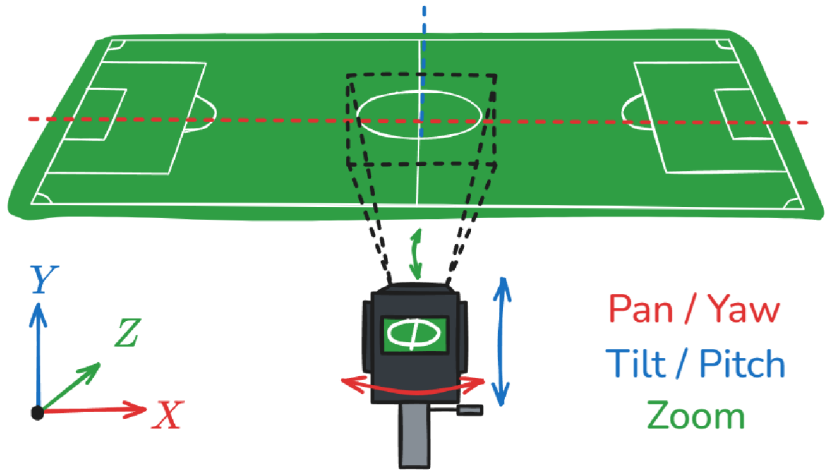


Figure 2.2: Camera coordinate system. Pan or yaw corresponds to left-to-right movement. Tilt or pitch to up-and-down movement [6]. The camera cannot move or roll (X-plane rotation around Z).

$$(X, Y, Z)^T \mapsto (u, v)^T = \left(\frac{fX}{Z} + p_x, \frac{fY}{Z} + p_y\right)^T \tag{2.1}$$

where $f \in \mathbb{R}$ is focal length — the distance between the camera centre $\mathbf{C}$ and the principal point p on the image plane, and $(p_x, p_y) \in \mathbb{R}$ are the coordinates of p. The model projects 3D points $\mathbf{X} = (X, Y, Z)^T$ to 2D points $\mathbf{x} = (u, v)^T$ in Cartesian coordinate systems [7, sec. 6.1].

By adding an extra dimension to the 2D and 3D coordinates, we obtain *homogeneous coordinates* $\dot{\mathbf{X}} = (X, Y, Z, 1)^T$ and $\dot{\mathbf{x}} = (u, v, 1)^T$. Homogeneous coordinates are defined up to a scale k [7, p. 2].

$$\mathbf{p} \in \mathbb{R}^m : \mathbf{p} = \frac{\dot{\mathbf{p}}}{k}$$

For example, $\dot{\mathbf{x}} = (u, v, 1)^T = (2u, 2v, 2)^T = (ku, kv, k)^T = \mathbf{x}$. When $k = 0$, the point is at infinity. This extends the Euclidean space $\mathbb{R}$ to projective space $\mathbb{P}$, allowing us to express the projection P between the 3D point $\dot{\mathbf{X}}$ and 2D point $\dot{\mathbf{x}}$ [7, eq. 6.3]:

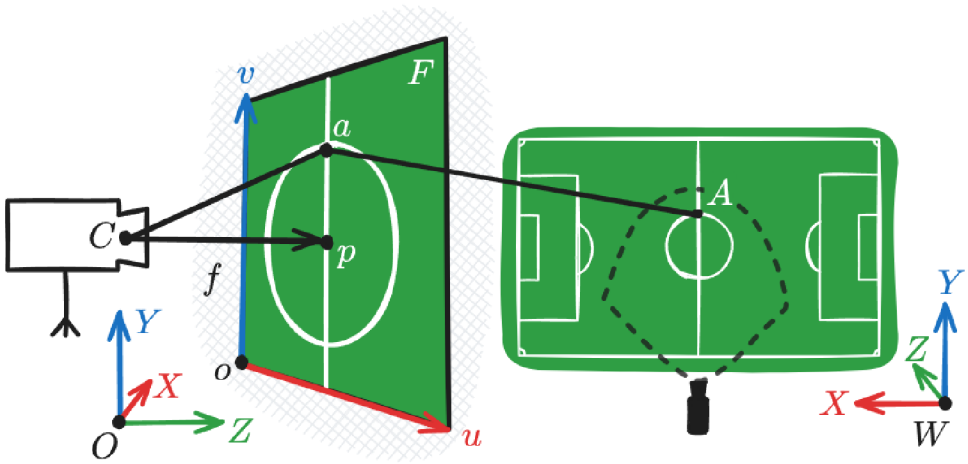


Figure 2.3: Pinhole camera model. The camera 3D coordinate system has an origin **O** under the camera centre **C** with the Z-axis in the same direction as the vector $\vec{f} = (C, p)$. The size of this vector is the focal length. The image plane **F** uses a 2D coordinate system with origin **o** in the lower left corner of the image given by the resolution. Points in 3D space use world 3D coordinates with arbitrarily chosen origin in point **W**. For example, 3D point A is mapped to 2D point a.

$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} Zu \\ Zv \\ Z \end{bmatrix} = \begin{bmatrix} Z(fX/Z + p_x) \\ Z(fY/Z + p_y) \\ Z \end{bmatrix} = \begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\dot{\mathbf{X}} \mapsto \dot{\mathbf{x}} = P\dot{\mathbf{X}}$$

If we set

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{2.2}$$

then:

$$\dot{\mathbf{x}} = K[I \mid 0]\, \dot{\mathbf{X}} \tag{2.3}$$

where $I \in \mathbb{R}^{3x3}$ is an identity matrix, and 0 is a column vector.

Points in 3D space are usually not expressed in terms of a coordinate system with origin **O** under the camera centre **C** and Z-axis in the direction towards the image plane. To generalize the camera model, we introduce another coordinate system with origin **W**. The world and camera coordinate systems are related by some rotation matrix $R \in \mathbb{R}^{3\times3}$, translation vector $t \in \mathbb{R}^3$ and camera centre in world coordinates $\mathbf{C}_{world} \in \mathbb{R}^3$. If $\mathbf{X}_{world}$ is an inhomogeneous 3D point in world coordinates and $\mathbf{X}_{camera}$ represents the same point in camera coordinates, then $\mathbf{X}_{camera} = R(\mathbf{X}_{world} - \mathbf{C}_{world})$. Analogically in homogeneous coordinates [7, eq. 6.6]:

$$\dot{\mathbf{X}}_{camera} = \begin{bmatrix} R & -R\mathbf{C}_{world} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} R & -R\mathbf{C}_{world} \\ 0 & 1 \end{bmatrix} \dot{\mathbf{X}}_{world}$$

We can now express eq. 2.3, in the world coordinate system:

$$\dot{\mathbf{x}} = KR[I \mid -\mathbf{C}_{world}]\, \dot{\mathbf{X}}_{world} \tag{2.4}$$

From eq. 2.4, the translation vector $t = -R\mathbf{C}_{world}$. Now, we can set $\mathbf{X}_{camera} = R\mathbf{X}_{world} + t$ instead, to obtain the projection matrix $P \in \mathbb{R}^{3\times4}$ in the textbook form [7, eq. 6.8]:

$$P = K[R \mid t] \tag{2.5}$$

We call P the *camera matrix*, K the *calibration matrix*, its parameters f, p_x, p_y the *intrinsic parameters* and $[R|t]$ the *extrinsic parameters* of the camera. Having these expressions defined, we may continue to the definition of recalibration.

2.3 Camera recalibration

The calibration process is referred to as the determination of camera matrix P from a set of sufficiently many correspondences using numerical methods [7, ch. 7]. Recalibration is a similar process where it is sufficient to determine only 3 degrees of freedom (DoF). These

parameters are defined in the problem analysis (ch. 2.1) as pan, tilt and zoom.

From the calibration matrix K (eq. 2.2), focal length f corresponds to the estimated zoom parameter $\hat{\theta}_{zoom}$. Because our cameras cannot roll (rotate around the Z axis), we define the rotation matrix R (eq. 2.5) as:

$$R = R_{pan} \cdot R_{tilt} \cdot R_{roll} = R_{pan} \cdot R_{tilt} \cdot I \tag{2.6}$$

Additionally, we will reuse all available calibration information in the recalibration process, such as the initial camera matrix P_0 and calibration keypoints $\mathcal{M}_0$. On each new frame F_n, we can reuse the last recalibrated camera matrix P_{n-1} and newly detected keypoints $\mathcal{M}_n$. The camera matrix P_n can always be reconstructed using the initial calibration P_0 and the last estimated parameter vector $\hat{x}_n$. Therefore, we define the problem as follows:

Recalibrate-camera	
INPUT:	camera matrix $P_0, P_{n-1} = K_{n-1}[R_{n-1}\|t]$ keypoints $\mathcal{M}_0, \mathcal{M}_n$ frames F_{n-1}, F_n
OUTPUT:	vector $\hat{x}_n$, camera matrix P_n

Problem 1

The need to continuously calibrate a camera is a problem known as *continuous self-calibration* [8] and is one of the most important problems in robotics. There are monocular and stereo-imaging techniques, such as Shape from X (SfX), Structure from Motion (SfM), Simultaneous Localization and Mapping (SLAM) and Visual Odometry (VO) - the process of estimating the pose and motion of the camera from an image sequence [9, 10]. A method for Recalibrate-camera based on these techniques is proposed in the following chapter.

3 Proposed method

In this chapter, we discuss our method based on probabilistic robotics and visual odometry. The core of the method is the particle filter framework, which consists of multiple functional blocks, overviewed in figure 3.1. Each block is detailed in the following sections. We begin with an explanation of the theory and respective filters used in robotics to track the camera movement.

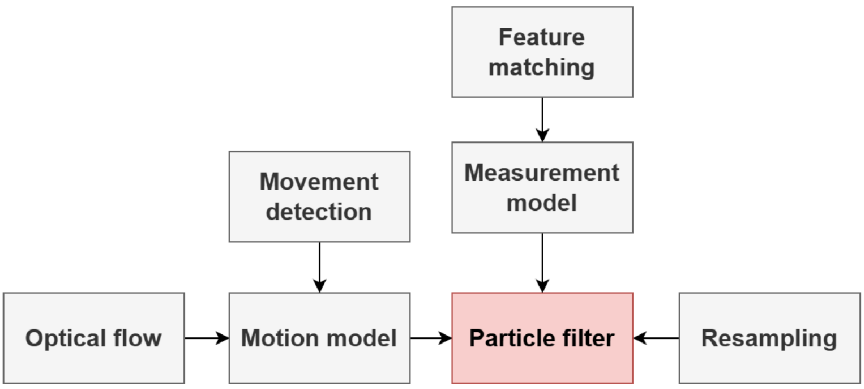


Figure 3.1: Method overview.

3.1 Robotics methodology

First, let us view the Recalibrate-camera problem from a different perspective. We know that the PTZ camera is tracking a soccer field and that we need to find the specific location where it is aimed. This leads us to view the issue as a localization problem, which is closely connected to robotics.

Robotics is an interdisciplinary field focused on the perception of machines (robots) that interact with our world. Robots use software that processes sensor input to navigate in the three-dimensional space. Just as humans have vision and hearing, robots use cameras, radars, LiDARs and other types of visual information [11, 12].

Localization is the problem of estimating a robot's coordinates using sensor data in reference to some map [11, ch. 1.2], in our case, the soccer pitch. Unlike mobile robots, which often use precise systems

such as GPS and odometers to correct their position, the only sensor available to us is monocular vision (specified in ch. 3.3).

There are multiple factors which make the problem difficult to solve. Robots are placed in the real world, which is not steady. This is similar to our camera — during a match, players move on the field; each stadium or soccer field is different; if playing outside, the sun casts shadows, weather changes, and more. Another issue is that sensors are not perfect and have tolerances. These difficulties give rise to uncertainty, which algorithms must account for [11, p. 2].

Because the initial position of cameras is known in advance, the problem is specifically known as *position tracking* [11, p. 159]. By recognizing this problem correctly, the research for possible solutions can be narrowed. The most used algorithms for localization are Bayesian filtering techniques, notably Extended Kalman Filter (EKF) and particle filters, also known as Monte Carlo filters, Importance Sampling techniques or Sequential Importance Sampling (SIS) [11, 13, 14].

The last part of this problem is mapping. Cameras are positioned around the pitch ahead of the match. Therefore, it is possible to prepare a map and update it during online processing. In combination with localization, this is known as the Simultaneous Localization and Mapping (SLAM) problem [11, ch. 9.1], which we already introduced with **PTZ-SLAM** in chapter 1. We decided to utilize only the available camera vision data and features of the soccer pitch. Ignoring the mapping problem, we trade the robustness of our solution for efficiency and straightforward implementation.

3.2 Bayes filters

One of the core concepts of probabilistic robotics is a *belief* function. Beliefs represent the robot's presumption of the real state of the system using conditional probability distributions, assigning a probability value to each possible hypothesis [11, ch. 2.3.4]. State x is a vector of all the variables affecting the robot's worldview. In our case, this would correspond to the pan, tilt and zoom parameters. The state vector is modelled as a random variable. The observation of the state x_n begins at time $n = 0$ and continues with discrete steps.

Theoretically, we differ between control and measurement data. The control data u_n carries information about the change of state, and the measurement data z_n represents the current state of the environment. We summarise these assumptions into the following equation [11, eq. 2.35]:

$$belief(x_n) = p(x_n|z_{0:n}, u_{0:n})$$

In other words, the belief assigns a probability to each possible state, which is dependent on all the previous states that were altered by measurement and control data.

A belief which does not incorporate the last measurement data is called a *prediction* [11, eq. 2.36]:

$$\overline{belief}(x_n) = p(x_n|z_{0:n-1}, u_{0:n})$$

Calculating $belief(x_n)$ from $\overline{belief}(x_n)$ is called *correction* or *measurement update* [11, ch. 2.3.4].

The Bayes filter is the most general algorithm for calculating beliefs [11, ch. 2.4.1]. The predict and update steps are defined recursively as follows:

Bayes-filter	
PREDICT:	$\overline{belief}(x_n) = \int p(x_n\|u_n, x_{n-1})\ belief(x_{n-1})\ dx_{n-1}$
UPDATE:	$belief(x_n) = \eta p(z_n\|x_n)\ \overline{belief}(x_n)$

Filter 1

where η is a normalization factor. To use this filter, we must know the initial $belief(x_0)$. The prediction step contains the *transition* (*motion*) model $p(x_n|u_n, x_{n-1})$, which states the probability that an action u_n changes the previous state to the current. The update step contains the *observation* (*sensor*) model $p(z_n|x_n)$, which states the probability of observing some measurement data given the current state [15].

This form of the filter is limited to simple estimation problems, as it should be possible to perform the integration in the prediction step in closed form [11, p. 24]. The limitations of this method, if used for localization, are presented in figure 3.2.

The Bayes filters work under the *Markov assumption* that past x_{n-1} and future x_{n+1} are independent of x_n. In the real world, this property may not necessarily hold, as the state can be affected by humans and inaccuracies in the probabilistic models [16, 17]. The dependency of random (state) variables is explained with a Bayes network model in figure 3.3.

It is natural to compare the Bayes filter with the Least-Squares method, which tries to optimize (or fit) a function to the data. Bayes filters have the advantage that the computation is being performed online — after every time step, a new measurement improves the estimate [18, ch. 3.2] [19, ch. 3.1.3, 3.4, 13.3.4].

There are two notable families of algorithms based on the Bayes filter — Gaussian and nonparametric filters [11, ch. 3, 4].

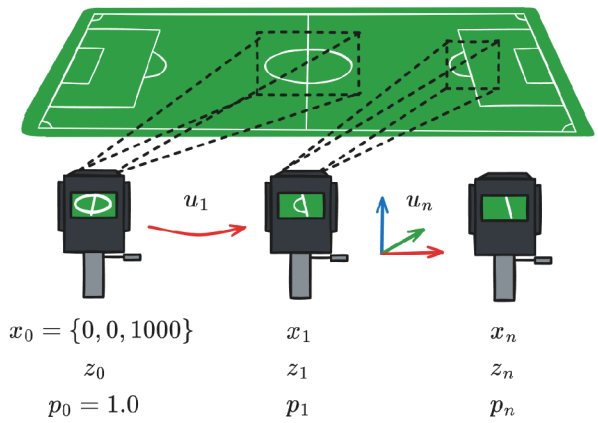


Figure 3.2: Localization using the Bayes filter. The initial belief is determined by the known intrinsic parameters x_0, the measurement z_0 and the trivial probability of the camera being at this state p_0. After the operator moves the camera in time step $n = 1$, an action u_1 happens, changing the state of the camera to x_1. At this point, all of the parameters x_1, z_1, p_1 are unknown. In general, it is impossible to determine the state x_n because the motion of the camera u_n is unpredictable, and the possible state space is practically infinite. As stated previously, the Bayes filter is not helpful in complex situations.

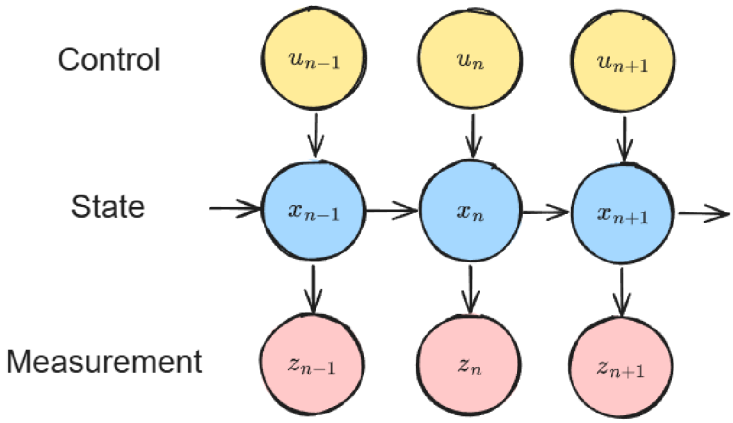


Figure 3.3: Bayes network of the Bayes filter under the Markov assumption. Each state is dependent only on the previous state.

3.2.1 Gaussian filters

The basic Bayes filter is impractical for real-world localization problems, mainly because of the intractability of the probability models. Gaussian filters are specializations of the Bayes filter for continuous spaces using multivariate normal distributions [11, eq. 2.4]:

$$p(x) = \frac{e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}}{\sqrt{det(2\pi\Sigma)}}$$

where μ is the mean and Σ the covariance matrix. The number of elements in Σ is quadratic in relation to the elements of the state vector. The biggest downside of this approach is that Gaussians are unimodal. Therefore, this probabilistic model can accurately predict only problems that have a single hypothesis with some marginal error [11, ch. 3.1]. An example of this model is shown in figure 3.4.

One of the most well-known Gaussian filters is the Kalman filter and its derivations.

3.2.2 Kalman filter

This filter is based on the Bayes filter but with an additional assumption that the next state probability $p(x_n|u_n, x_{n-1})$ must be a linear function

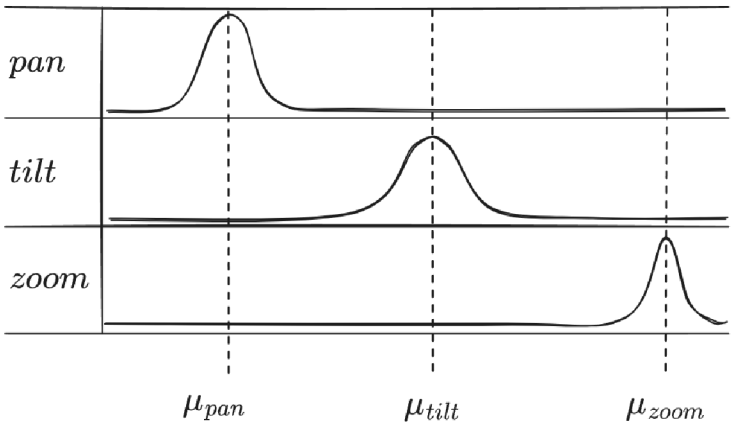


Figure 3.4: Representation of the state model using unimodal Gaussians. The model is missing physical variables, such as velocity or rotation, caused by the lack of reliable sensors. The PTZ variables have no covariance, only noise variances, which we are unable to determine, as the system must work regardless of the camera model and potential noise.

in its arguments and with added Gaussian noise $\varepsilon \sim \mathcal{N}(0, R_n)$:

$$x_n = A_n x_{n-1} + B_n u_n + \varepsilon_n$$

The measurement model $p(z_n|x_n)$ must also be linear in its arguments with added Gaussian noise $\delta \sim \mathcal{N}(0, Q_n)$:

$$z_n = C_n x_n + \delta_n$$

The variables A_n, B_n, C_n are matrices that define the linear relation between state transitions, and R_n, Q_n are covariance matrices.

Kalman-filter	
PREDICT:	$\overline{\mu} = A_n \mu_{n-1} + B_n u_n$
	$\overline{\Sigma}_n = A_n \Sigma_{n-1} A_n^T + R_n$
UPDATE:	$\mu_n = \overline{\mu}_n + K_n(z_n - C_n \overline{\mu}_n)$
	$\Sigma_n = (I - K_n C_n)\overline{\Sigma}_n$

Filter 2

The Kalman gain K_n influences the new state in regard to the last measurement update.

The disadvantage of this filter is that it can only accurately predict linear systems. It is not suitable for dynamic systems and has a problem with *kidnapping*, i.e. when the estimated belief no longer matches the real state of the robot [11, 20, 21].

An altered Kalman filter, which relaxes the linear requirement, is discussed in the next section.

3.2.3 Extended Kalman filter

The Extended Kalman filter (EKF) differs from the Kalman filter by replacing the matrices A_n, B_n, C_n with non-linear functions g, h:

$$
\begin{aligned}
x_n &= g(u_n, x_{n-1}) + \varepsilon_n \\
z_n &= h(x_n) + \delta_n
\end{aligned}
$$

To work correctly as a Bayes filter, EKF employs the first-order Taylor expansion to approximate the belief by a Gaussian [11, ch. 3.3].

Extended-kalman-filter	
PREDICT:	$\overline{\mu} = g(u_n, \mu_{n-1})$
	$\overline{\Sigma}_n = G_n \Sigma_{n-1} G_n^T + R_n$
UPDATE:	$\mu_n = \overline{\mu}_n + K_n(z_n - h(\overline{\mu}_n))$
	$\Sigma_n = (I - K_n H_n)\overline{\Sigma}_n$

Filter 3

Compared to Kalman-filter, the matrices A_n, B_n, C_n are replaced with Jacobians G_n, H_n.

In robotics, EKF is heavily used in localization or SLAM systems. This claim is supported by the fact that the state-of-the-art **PTZ-SLAM** also uses EKF. It gained popularity because of its simplicity and computational efficiency. By vast application, it was also found that EKF can be used in environments where the underlying (Bayes, Gaussian, unimodal, etc.) assumptions are violated. If the system dynamics are

close to being linear, or if the uncertainty is limited, the Gaussian approximations are sufficient to accurately represent the system [11, ch. 3.3.4]. However, real systems are often modelled after multimodal distributions, such as the position of the robot. In such cases, the linearization around the mean is not optimal. There are alternative versions of EKF, such as the Unscented KF (UKF), Robust KF (RKF) or Cubature KF (CKF), which try to improve the performance of EKF [11, 22].

We decided to use an entirely different approach — a nonparametric filter.

3.2.4 Nonparametric filters

Nonparametric filters are based on the idea that instead of relying on a fixed form of the posterior (i.e. after update) density function — such as Gaussian, they approximate the belief state by a finite number of values, each representing a region in a state space (i.e. a state vector).

The particle filter (PF) is a nonparametric filter that uses several techniques from which the filter designer can choose. This allows us to fine-tune the filter to the specifics of a system as accurately as possible. Therefore, it is more convenient to think of PF as a framework [13].

Particle filters started emerging in the second half of the last century, driven by the demand to represent complex real systems [13, 23]. The earliest usage of the particle filter essentials was recorded by scientists trying to model the extension of molecular chains. In the articles *Poor Man's Monte Carlo* [24] and *Monte Carlo Calculation of the Average Extension of Molecular Chains* [25], the researchers explain how they used the seemingly irrational Monte Carlo method — use of repeated random sampling to achieve a sound modelling of a system. They show that the Monte Carlo method is actually the art of reducing the randomness of a system so that it procedurally dissipates from the final result.

The Monte Carlo method is used in conjunction with the *Sequential Importance Sampling* (SIS) technique of recursively drawing samples from an arbitrary distribution to approximate a target distribution using the weighted average of the samples [14]. Often, using only SIS is not enough and more techniques are needed. As with the other

filters we already stated, the particle filter has its own challenges, namely [18, ch. 4.2]:

1. Degeneracy (fig. 3.6)
2. Sample impoverishment (fig. 3.7)
3. Divergence (fig. 3.8)
4. Importance density selection (fig. 3.9)
5. Real-time execution

In the following section, we will provide a more detailed insight into the particle filter, including how we addressed the issues in our method.

3.2.5 Particle filter

Using the terminology of the Bayes filter, the key principle of PF is to represent the posterior $belief(x_n)$ using a set of random state samples (particles) and their weights $\mathcal{X}_n = \{(x_n, w_n)\}_{i=1}^{N}$ from the same distribution $belief(x_{n-1})$. Each particle x_n^i represents a single hypothesis of the belief in the parametric space at time step n. Using terminology of the camera pose estimation problem, a single particle represents a possible pose. An example state space is shown in figure 3.5. The approximation is given by the Strong Law of Large Numbers or the Central Limit Theorem [26, p. 2]:

$$belief(x_n) \approx \sum_{i=1}^{N} w_n^i \delta(x_{0:n} - x_{0:n}^i) \tag{3.1}$$

where $\sum_{i=1}^{N} w_n^i = 1$, δ is the Dirac delta function and N is the amount of particles.

However, drawing samples from $belief(x_n)$ is not possible because it is unknown. We employ the SIS technique to sample particles from an arbitrary *proposal* distribution q and assign weight to them. The proposal distribution should be chosen in a way that minimizes the variance of the importance weight after each recursion (time step) of the filter [27, p. 199][18, ch. 4][14]:

$$w_n^i \propto \frac{p(x_{0:n}^i | z_{0:n}, u_{0:n})}{q(x_{0:n}^i | z_{0:n}, u_{0:n})} \tag{3.2}$$

where w_n^i is proportional and must compensate for the fact that it is not drawn from the posterior density [18, ch. 4.1].

Unfortunately, it can be shown that if we select, for example, a Gaussian as the proposal q, the variance would increase exponentially with N. To improve this algorithm, we introduce an additional *resampling* step [28, ch. 3.4].

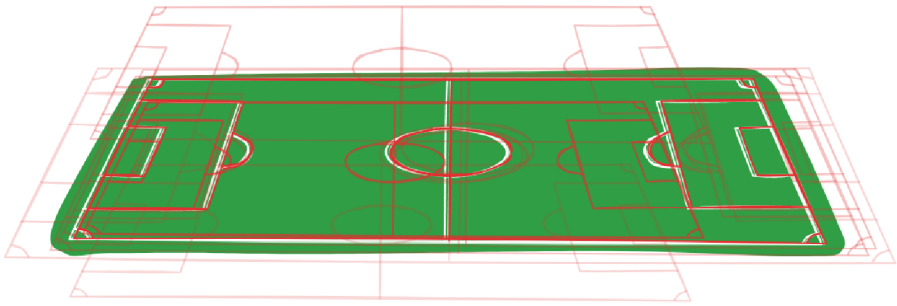


Figure 3.5: Particle state space. White pitch specifies the ground truth state. Each red pitch represents one particle in a 3D state space. Particles with less weight appear more transparent. For the sake of visualisation, the amount of particles is kept low.

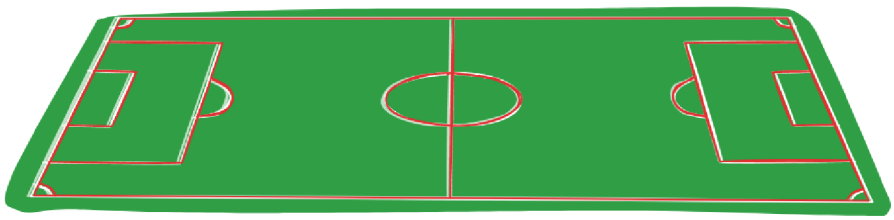


Figure 3.6: Degeneracy problem. After an arbitrary number of steps, the state space "degenerates" to a single particle with maximum weight. This means that the particle filter can never recover to accurately predict the real camera movement.

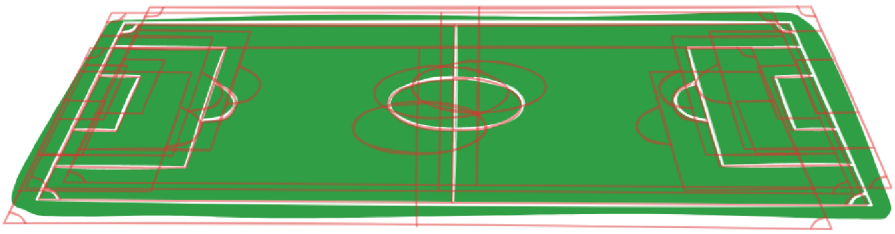


Figure 3.7: Sample impoverishment problem. When the variability of the noise model or the amount of particles is too low, the state space will never contain a particle accurately representing the real position.

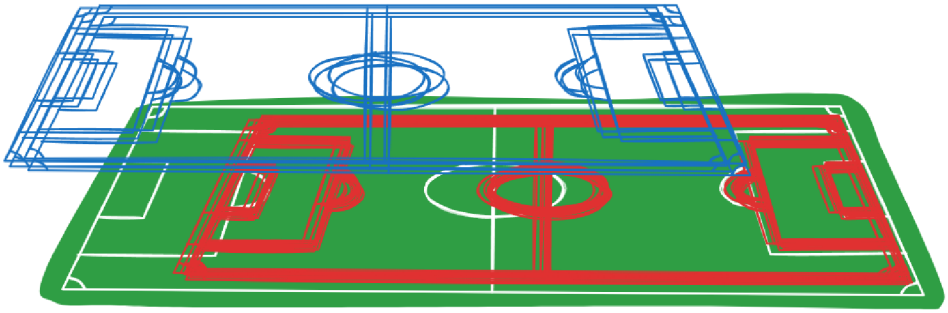


Figure 3.8: Divergence problem. The state space can contain multiple groups of particles close to each other, causing the state to get kidnapped. In such a case, we can determine which group has the highest possibility of being the correct one using metrics (weight, amount, centroid, distance) or specific algorithms (DBSCAN, DKDR) [29].

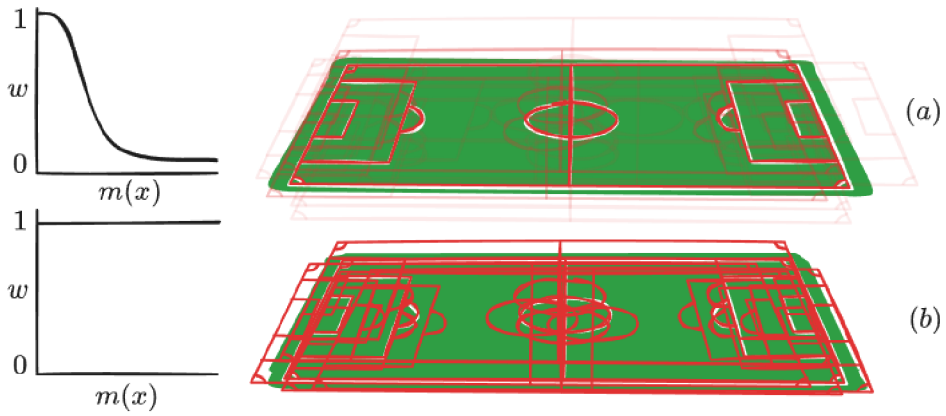


Figure 3.9: Importance density selection. Importance density is defined as a function that assigns weight w to each particle according to some metric $m(x)$. In figure (a), the density is selected in such a way that particles farther from the ground truth are less important. This causes the variance of the system to continually decrease. The system in figure (b) has a degeneracy problem — it will always "float" around some state but never stabilize around the ground truth.

3.2.6 Sequential importance resampling

Because the variance can only increase in SIS, the algorithm has a degeneracy problem. After a finite number of steps, the weights of particles quickly approach zero, which wastes computational resources [30, p. 9] [31].

Sequential Importance Resampling (SIR), or Sequential Monte Carlo (SMC), is a practical version of SIS often employed in the PF framework. Resampling is a process of sampling the set of particles $\mathcal{X}_n = \{(x_n, w_n)\}_{i=1}^{N}$ into a new set of particles $\tilde{\mathcal{X}}_n = \{(\tilde{x}_n, \tilde{w}_n)\}_{j=1}^{N_s}$ according to particle weight biased probability density function (PDF). After this process, we set $\mathcal{X}_n = \tilde{\mathcal{X}}_n$ [32, p. 72]. This means that more particles will be generated in the most probable region of the parametric space. This principle is explained in figure 3.10.

Because the selection of a resampling method is crucial in the convergence of the PF algorithm, a vast amount of methods were researched. However, there are three most popular algorithms: multi-

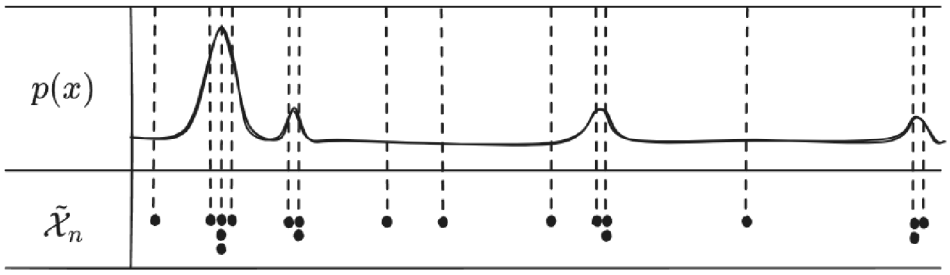


Figure 3.10: Importance sampling (SIR). The figure depicts a 1D particle state to demonstrate the importance sampling principle. The particles from $\mathcal{X}_n$ are resampled according to the latest PDF $p(x)$ and an importance sampling algorithm. Particles are copied, duplicated or omitted according to their representation in the PDF. The weight of the resulting particles is reset to some arbitrary number.

nomial, systematic (or stratified) and residual resampling. In general, the methods differ in the following properties [28, p. 13] [32, p. 72-74]:

- Resampling from $\mathcal{X}_n$ or some other approximate distribution
- The parametric space is resampled homogeneously or using more methods for different parts
- Particles are grouped, or there is only a single distribution present
- The weight of future, past, current or mix of particles is used
- The resampling is performed on every time step or selectively
- Whether the algorithm is deterministic or stochastic

The resampling methods are described in figures 3.11 and 3.12.

Bootstrap filter, which is the most basic particle filter including the SIR technique, is introduced in the next section [28, ch. 3.4].

3.2.7 Bootstrap filter

The bootstrap filter is the base algorithm that our method is built and extended upon. First, the filter is initialized and then the state is estimated in a loop, calling specific routines (predict, update, normalize

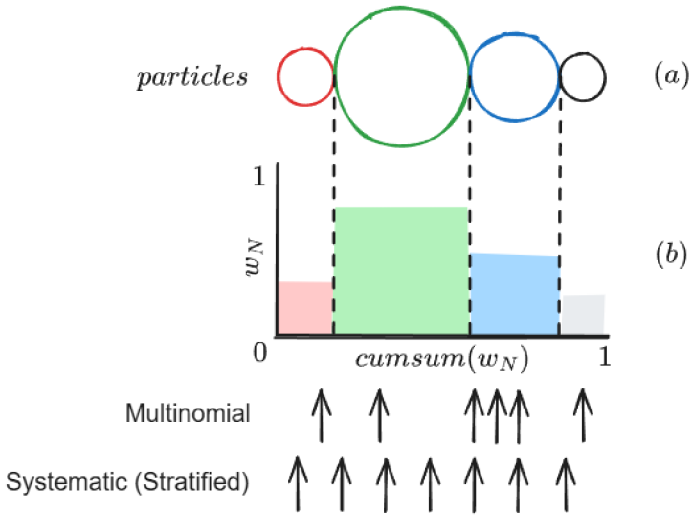


Figure 3.11: Multinomial and systematic resampling. Each particle state (a) is defined by a circle with a radius according to its weight. The weight of all particles is normalised (W_N) so that their cumulative sum (b) is equal to 1. In multinomial resampling, particles are selected or duplicated randomly $\sim \mathcal{U}(0,1]$ from the cumulative sum. Systematic resampling generates a random number $\sim \mathcal{U}(0, \frac{1}{N}]$, and then all other particles are selected according to the distance between selections (strata) equal to $\frac{1}{N}$. Stratified resampling samples randomly in each of the subintervals. [32, p. 75-76]

and resample) in each iteration. The computed state then affects the next estimation. This process is explained in figure 3.13.

For resampling, we have chosen the systematic algorithm, which is arguably favourable to multinomial resampling, as it provides better estimates and is less computationally complex [33, 34]. The degeneracy problem is therefore reduced, causing the state space to converge to a singular value. However, this value could be sampled from a valley of the parametric space that is not reflecting the correct state. This behaviour can be caused by the sample impoverishment issue. The most common way to reduce sample impoverishment, and the one we chose, is to introduce additional random noise to the transition (motion) model during the prediction step or after resampling. This process is called *roughening* [32][p. 80]. Other techniques include

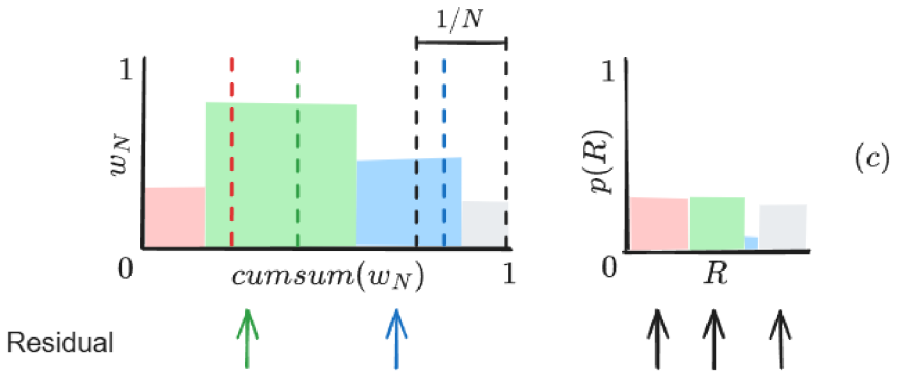


Figure 3.12: Residual resampling. Residual resampling (c) is a more complex, two-step algorithm. First, all particles weighing more than $\frac{1}{N}$ are duplicated, and their weight is subtracted repeatedly so that only the residual weight R remains. Multinomial resampling (or any other) can be used in the second step to pick particles from the residuals. [32, p. 75-76]

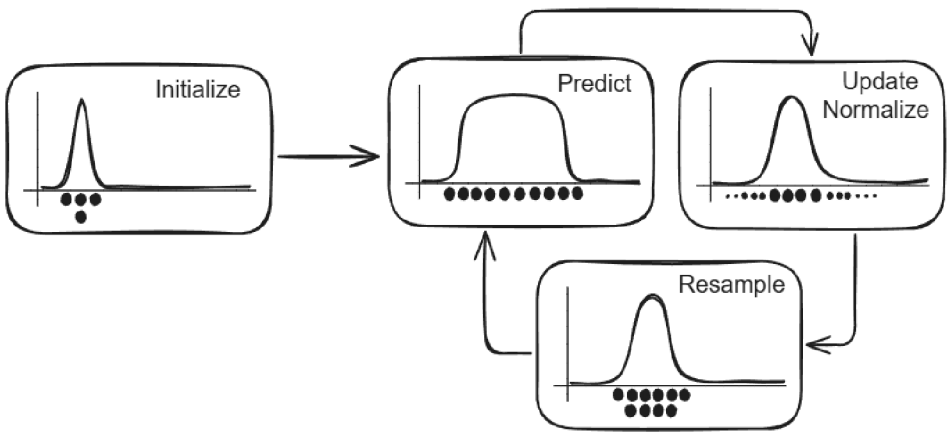


Figure 3.13: Bootstrap filter processing loop. The initial particles with equal weight are spread around the state space with the predict routine. The update and normalization routines cause the particles to converge around a possibly correct state. The resampling step then tries to remove particles with low probability of reflecting the true state, and favours the high probability ones.

Auxiliary PF or introducing the so-called Metropolis-Hastings (HS) step [18, p. 5.2].

In case the PF still concludes a wrong estimate, it is possible that it has diverged. Particle filters usually model a multimodal function, which means there is a non-zero chance that the PF estimates a false reality. Possible causes can include insufficient transition models, measurement noise, or the use of wrong measurements altogether. In our case, the only way to recover is for PF to detect if it has diverged and then perform reinitialization. A common method is to calculate the Kullback-Leibler divergence or some other sort of metric. [18, p. 5.3]. In terms of robotics (sec. 3.1), this is equivalent to the relocalization problem, requiring some sort of map or other ground truth data.

The bootstrap filter is defined with pseudocode on the following page. The predict and update routines are explained in the next sections.

Bootstrap-filter

INPUT:	particles amount N, resampling amount N_s
	initial state $p(x_0)$
OUTPUT:	set of particles $\mathcal{X}_n$

```
n ← 0
X_n ← INITIALIZE(N, p(x_0))
loop
    for i = 1, ..., N do
        PREDICT
        UPDATE
        NORMALIZE
    end for
    RESAMPLE(N_s, X_n)
    n ← n + 1
end loop
```

INITIALIZE

```
for i = 1, ..., N do
    draw x_n^i ~ p(x_0)
    X_n[i] ← (x_n^i, 1)
end for
n ← 1
```

PREDICT: draw $x_n^i \sim q(x_n^i | x_{n-1}^i, z_n, u_n)$

UPDATE: assign weight w_n^i using eq. 3.2

NORMALIZE: $w_n^i = w_n^i \ / \ \sum w_n$

RESAMPLE: draw N_s particles from $\mathcal{X}_n$ to $\tilde{\mathcal{X}}_n$
reset all weights $\tilde{w}_n$ to $1/N_s$
$\mathcal{X}_n \leftarrow \tilde{\mathcal{X}}_n$
$N \leftarrow N_s$

Filter 4

3.3 Motion model

As previously mentioned in section 3.2, the motion model is crucial during the prediction step of the filter. In terms of camera motion, the determination of rotation and translation from sequential images is known as the ego-motion problem. This motion is described by its motion field, which is estimated by optical flow. The optical flow is then numerically analysed to obtain the rotation angles and translation offsets [35]. In robotics, the reconstruction of a 3D scene (e.g. soccer pitch) using vision data is known as Structure from Motion (SfM). The output of this process is also the estimated pose of the camera. Therefore, we based our motion model on the principles of photogrammetry and visual odometry [36].

3.3.1 Optical flow

Optical flow is a notion that started gaining popularity in the 1980s, and it is one of the most crucial concepts in modern computer vision problems, driven by the development of autonomous vehicles or robots. The rapid use of neural networks has created new paths for research — such as deepfake detection, frame generation, image upscaling or object motion detection [37, 38, 39].

In the most basic form, to solve the estimation of optical flow between two consecutive image frames, we assume the intensity of pixels $I_n(x, y)$ between the images remains constant as they move. This is known as the brightness constancy constraint [38, eq. 1]:

$$\frac{\partial I}{\partial n} + u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} = 0$$

where (u, v) is the flow vector of $\Delta I_n(x, y)$. In practical applications, this constraint is not sufficient to accurately describe the motion flow, and more sophisticated methods were developed. We distinguish between sparse methods, which calculate flow for selected pixels only, and dense methods, which use the whole image information. Lucas-Kanade (sparse) and Farneback (dense) are some of the most known methods, respectively [40]. Figure 3.14 shows the use of brightness constraint across multiple frames to compute the optical flow.

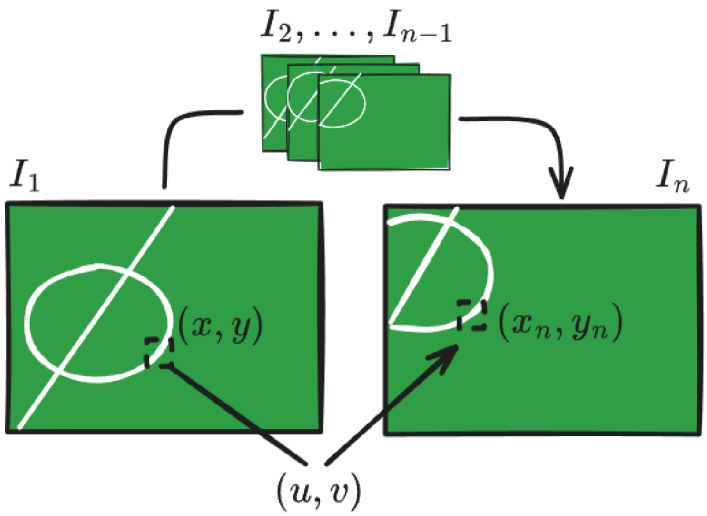


Figure 3.14: Optical flow demonstration. The brightness intensity difference between consecutive frames is usually too small in sports videos that have 24-60 fps. Therefore, we are interested in the flow vector spanning multiple frames. The soccer pitch can be an exceptionally hard environment to track, caused by repeated white marks and vast green fields. In this case, tracking specific landmarks would be more appropriate than analysing the whole frame.

In our case, optical flow is especially useful because of the high image quality and restricted movement. However, the dynamic environment (people and objects) poses a substantial problem, which will be detailed in another section. We track the optical flow of pitch landmarks or other high-quality features on the scene using the Lucas-Kanade method.

Before using flow vectors as the distribution for the predict routine of Bootstrap-filter, we need to continue with the analysis of the motion and obtain exact numerical estimates for pan, tilt, and zoom parameters.

3.3.2 Motion analysis

As is defined in the problem analysis (ch. 2.1), the optical flow cannot be a result of a physical translation of the camera in the world.

Therefore, we assume that the motion is rotational-only. Moreover, the camera is tracking a rectangular soccer pitch field, which, under certain relaxations, is a planar scene. These two assumptions give us enough reason to use homography (a projective transformation) to determine the relative camera movement [7, ch. 8.4.2]. This approach is often used in an augmented reality setting [41]. Another option is to use affine transformation, which can be used if the differences between images are very small [42, p. 683]. The homography calculation is explained in figure 3.15.

In the following sections, we analyse the pan-tilt and zoom movement independently.

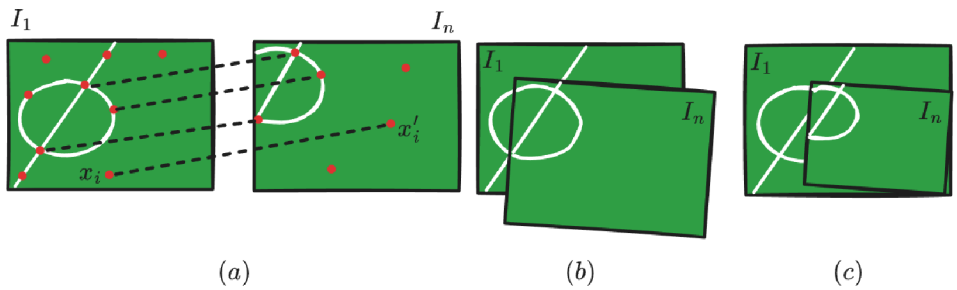


Figure 3.15: Homography between frames. In (a), we select specific pixels (red marks) that we want to use for the homography estimation between frames. If we select good points for the tracking, the homography matrix H will give us the projection resulting in (b). In case the algorithm fails, we obtain a bad estimation (c).

3.3.3 Pan-tilt movement

Two or more camera frames with the same camera centre C are related by a homography matrix $H \in \mathbb{R}^{3x3}$ defined by a set of 2D to 2D correspondences $\{x_i \leftrightarrow x'_i\}$ [7, p. 91]. If there is no change in intrinsic parameters (i.e. focal length), then:

$$H = KRK^{-1}$$

This special case is also known as conjugate rotation [7, p. 204, 628]. We can now obtain the rotation matrix between the frames as:

$$R = K^{-1}HK \tag{3.3}$$

and we convert the rotation matrix R to rotation vector $\vec{r}$ using the Rodrigues formula:

$$R = R_x R_y R_z \xmapsto{Rodr.} \vec{r} = (\Delta\varphi_x, \Delta\varphi_y, \Delta\varphi_z)$$

Assuming that the camera cannot roll (eq. 2.6), we obtain the estimated changes of pan and tilt between two frames:

$$\Delta\varphi_{pan} = \Delta\varphi_x \qquad \Delta\varphi_{tilt} = \Delta\varphi_y$$

This deduction will only hold in the ideal scenario in which all the assumptions hold true. The homography H needs to be calculated using error minimization techniques, such as RANSAC, which tries to pair the correspondences between images with best effort [7, p. 123]. For example, the result could contain a non-zero roll, which is against our established model. In such a case, the resulting homography H would be inaccurate.

A more robust approach is to calculate the rotation and translation parts using a decomposition technique and then discard the translation. There are numerical (Singular Value Decomposition) and analytical methods. However, they always have multiple solutions, ranging from, at most, 8 for SVD techniques and 2 for the analytical method. The algorithm designer would need to always choose the correct solution according to some heuristics – still encountering the possibility of a wrong result [43]. Instead, we use our presented approach of using eq. 3.3 combined with an arbitrarily chosen noise distribution.

3.3.4 Zoom movement

Similarly to pan-tilt, we also use homography to measure the zoom-change ratio. Ignoring the difference in radial distortion during the pure zoom process, the homography H between two calibration matrices K, K' is calculated as: [7, ch. 8.4.1]:

$$H = K'K^{-1} = \begin{bmatrix} k_x & & (1-k_x)p_x \\ & k_y & (1-k_y)p_y \\ & & 1 \end{bmatrix} \tag{3.4}$$

In our camera model, $k = k_x = k_y = f'/f$. Using RANSAC to compute the homography matrix yields the possibility of $k_x \neq k_y$. In such a case, we take the mean of the calculated ratios:

$$\Delta f_{zoom} = \frac{k_x + k_y}{2} \tag{3.5}$$

Same as in the previous section, we sample the values combined with a noise distribution.

3.3.5 Movement detection

Because we separate the camera movements to pan-tilt and zoom, we need to decide which movement should be applied for the particles in the prediction step. We use a simple decision algorithm specified in *Robust Camera Pan and Zoom Change Detection Using Optical Flow* [44], which detects the movement type according to flow vector orientation.

First, the optical flow between frames is calculated, and a motion histogram (fig. 3.16) is constructed from all the motion vector orientations. The algorithm detects pan-tilt and zoom separately (fig. 3.17), using two different equations, which are summarised in the flowchart (fig. 3.18). If both zoom and pan-tilt change is detected at once, we categorize the movement as complex. It is also beneficial to detect if no movement has been detected at all, in which case we can skip the next processing steps.

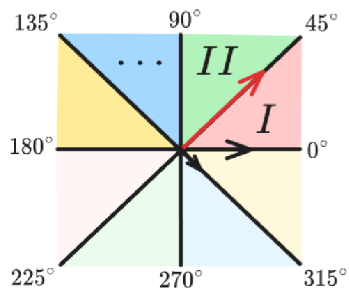


Figure 3.16: Motion histogram. The direction of camera movement is determined by the significant motion vector (in red). The histogram has 8 bins, each corresponding to 45°.

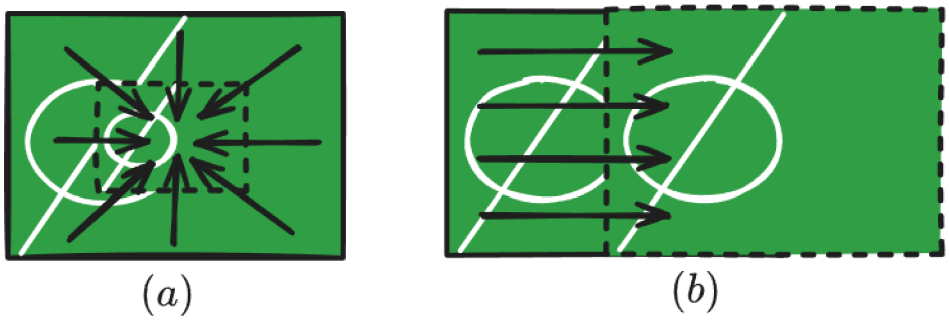


Figure 3.17: Motion vector categories. We recognize two types of motion vectors — converging/diverging into/from the centre (a), or tangential (b), meaning a movement along the X-Y axis.

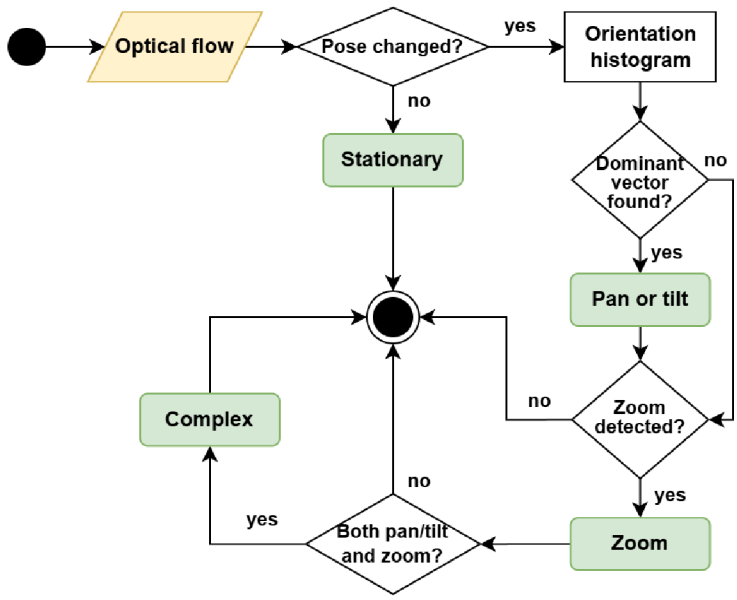


Figure 3.18: Movement detection algorithm. Optical flow is used to compute motion vectors between two frames. Then, for all vectors (x, y) and (x', y') bigger than $\varepsilon \in R$, an orientation histogram is constructed from values $\tan^{-1}\left(\frac{y'-y}{x'-x}\right)$. The bin with the most vectors is marked as the dominant orientation. Next, vectors converging or diverging to/from the image centre point p indicate zoom in/out.

3.3.6 Application

We view the alteration of the pan-tilt-zoom parameters as a change in control data u_n. For each of the parameters $\Delta\varphi_{pan}$, $\Delta\varphi_{tilt}$ and Δf_{zoom}, we define a probability distribution, such that:

$$\begin{aligned}\mathcal{P}_{pan} &= \mathcal{U}(\Delta\varphi_{pan} - \alpha, \Delta\varphi_{pan} + \alpha)\\ \mathcal{P}_{tilt} &= \mathcal{U}(\Delta\varphi_{tilt} - \beta, \Delta\varphi_{tilt} + \beta)\\ \mathcal{P}_{zoom} &= \mathcal{U}(\Delta f_{zoom} - \gamma, \Delta f_{zoom} + \gamma)\end{aligned}$$

where $\mathcal{U}(a, b)$ is a uniform distribution drawing samples between minimum a and maximum b, and $(\alpha, \beta, \gamma) \in \mathbb{R}$ is pan, tilt, and zoom *uncertainty*. The predict step then becomes:

$$\begin{aligned}\Delta x_n^i &= (\theta_{pan} \sim \mathcal{P}_{pan}, \theta_{tilt} \sim \mathcal{P}_{tilt}, \theta_{zoom} \sim \mathcal{P}_{zoom})\\ x_n^i &= x_{n-1}^i + \Delta x_n^i\end{aligned}$$

Therefore, we have roughened the prediction by drawing samples from a distribution with added noise. This gives the particle filter a bigger tolerance to wrong predictions. The tolerances are visualised in figure 3.19.

In cases when there is pure rotation or pure zoom, we can explicitly set $\theta_{pan,tilt} = 0$ or $\theta_{zoom} = 0$ and draw the rest of the parameters from their distributions. The values of α, β, γ depend on the actual system noise and are chosen by experimentation.

If no motion has been detected, it is important to still draw some Δx_n^i so that the particle filter will not get impoverished. In such a case, we sample from the last state x_{n-1}^i. The uncertainties can be different for each possible movement state.

As we stated at the beginning of section 3.3.3, we are using homographies between frames, and thus, we have to define a set of 2D to 2D correspondences. We use points detected by corner detection methods (Harris, Shi-Tomasi), pitch landmarks defined by initial calibration and uniformly scattered random points across the whole image — this combination gives us a big enough dataset for good RANSAC convergence and movement detection results.

Having defined the motion model and the predict routine, we can move on to the sensor (measurement) model.

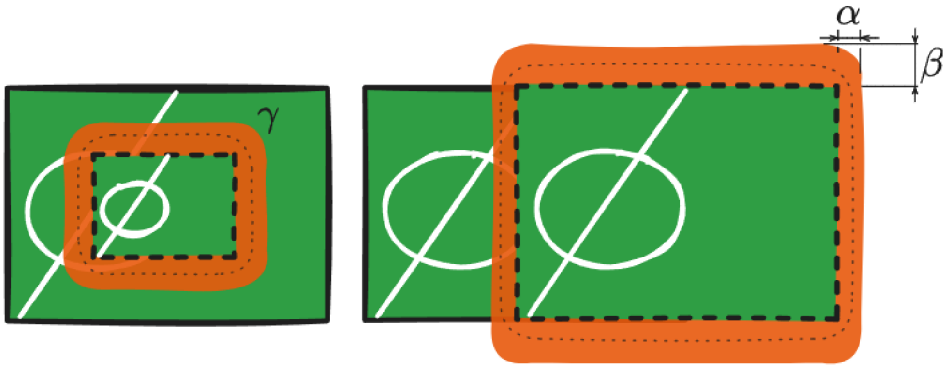


Figure 3.19: Motion uncertainty. The resulting prediction is sampled around values from our motion analysis with added model noise α, β and γ.

3.4 Measurement model

Analogically to the motion model, the measurement model is necessary to assign weight to each particle so that a correction occurs. Particles moving in the proper direction are rewarded with larger weight, while particles moving towards the wrong direction will be eventually discarded.

According to numerous research in augmented reality environments, some of them involving sports [42, 45, 46, 47], tracking of 3D objects (e.g. a soccer pitch) is usually based on the tracking of feature points, landmarks, keypoints, lines, circles or other interests in the 3D scene. There are methods that are online, offline or both. The offline part is used for information gathering — setting up the initial 2D to 3D correspondences. It takes a substantial amount of processing time and often requires manual calibration — checkerboard patterns, QR codes or other additional knowledge. Online processing then tries to find the same feature points, comparing them to the calibration data to obtain information about position change.

Using the offline calibration data provided by **GOAL SPORT technology s.r.o.** (ch. 2.1), we can use the soccer pitch model and initial 2D to 3D correspondences of specific landmarks on the pitch — such as line corner points, to verify the correctness of particles after predict.

3.4.1 Feature matching

In robotics, there is an idea of a point cloud — a summary of a 3D scene captured using some sensor data [48]. However, as we have already stated during the explanation of SLAM (ch. 3.1), robots do not rely on 2D vision only but also use data from high-quality sensing instruments. The soccer pitch is exceptionally hard to describe using common feature descriptors — such as SIFT, ORB, KAZE, AKAZE and others. In our testing, SIFT produced the best results of matching between frames. Unfortunately, the computation of hundreds of SIFT descriptors is slow. Other descriptors, such as ORB, are faster but less accurate. These findings are in line with performance research of descriptors for feature matching [49].

It may be worth preprocessing the soccer pitch image before finding any descriptors, such as changing the contrast and brightness or converting the colour space. However, this would limit the usage of the algorithm to a subset of environments. For example, some methods use vertical and horizontal grass patches to find a grid — these are not standardised, can be destroyed during the match or hidden by snow or shadow.

Instead of using complex descriptors and image processing techniques, we use image patches (template matching) and neural network for keypoints provided by **GOAL SPORT technology s.r.o.**

3.4.2 Template matching

In the simplest form, a matching between two images is given by the function [50, eq. 1.1]:

$$d(a, b) = \frac{1}{N}||a - b||_2^2$$

where a, b are two intensity vectors of the images (single or multi-channel), N is the amount of compared pixels, and $||a - b||_2^2$ is the Euclidean norm. The lower the distance d is, the more similar the images are. This basic concept can be extended with various methods (FFT, Cross Correlation, Pyramids, Log-Polar) to achieve better computational efficiency and robustness [50, ch. 1.2]. The matching algorithm is explained in figure 3.20.

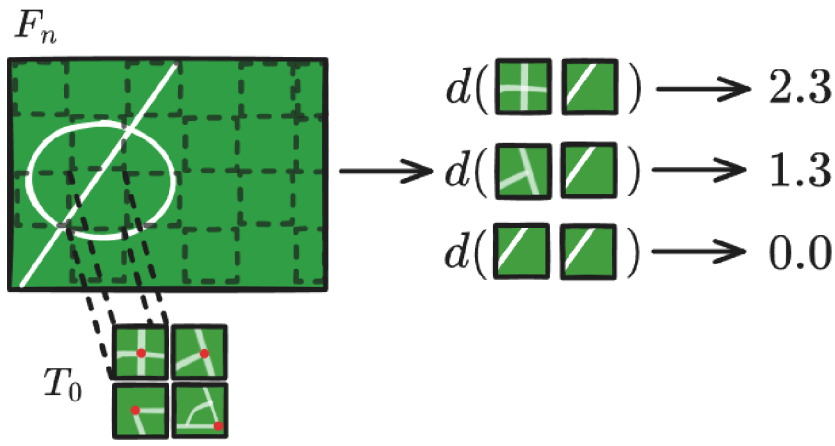


Figure 3.20: Template matching. Using the most naive algorithm, we try to find each template from T_0 corresponding to a 3D to 2D keypoint correspondence in each frame F_n and assign a numeric value to the match. The best match is then the one with the lowest value. However, this method is susceptible to false positives.

We denote the set of 2D to 3D correspondences of specific landmarks taken during offline calibration as:

$$\mathcal{M}_0 = \{(\mathbf{x}, \mathbf{X}) \mid \mathbf{x} \in \mathbb{R}^2, \mathbf{X} \in \mathbb{R}^3\}$$

and template for each i-th correspondence as $T_0^i(x_0^i, y_0^i, a, b)$ where (x_0^i, y_0^i) are 2D coordinates of a landmark in the calibration image and (a, b) is the width and height of the template. The initial calibration procedure is demonstrated in figure 3.21.

During the UPDATE step, we find the 2D coordinates in current frame F_n such that $\min\{d(T_0, F_n)\} \to 0$, giving us a set of found landmarks $\mathcal{M}_n^1$. Additionally, we detect keypoints using the provided machine-learning model to obtain the set $\mathcal{M}_n^2$.

The combination of different landmark detection mechanisms makes our algorithm more error-tolerant, and it is encouraged to find as many sources as possible. This process is also known as *sensor fusion* [46, 51, 52].

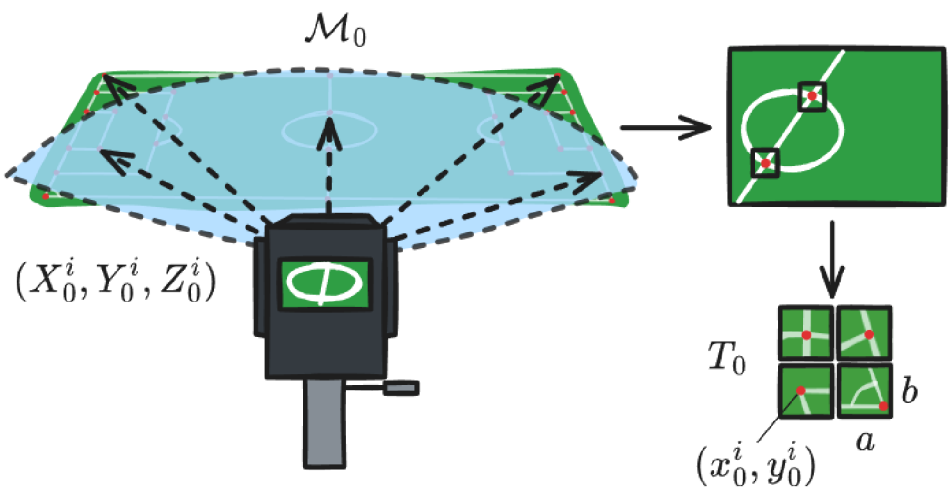


Figure 3.21: Initial calibration procedure. The operator handles the camera, directing it at landmark locations and manually sets the 2D to 3D correspondences $\mathcal{M}_0$. After the calibration is complete, we obtain the initial camera parameters (PTZ, translation) and templates T_0 for each keypoint.

3.4.3 Application

In every predict step of Bootstrap-filter, we gather all the sources of landmarks into the set $\mathcal{M}_n = M_n^1 \cup M_n^2 \cup \ldots$ and for each particle x_n^i, we calculate its *reprojection error*.

Reprojection error is a metric that quantifies the distance between pixels projected from 3D to 2D. First, we project all 3D points $\mathbf{X} \in \mathcal{M}_0$ to 2D as:

$$\mathbf{x} = P_n^i \mathbf{X}$$

where P_n^i is the camera matrix constructed using the current particle x_n^i pan-tilt-zoom parameters. Then, we compare these reprojected pixels with their correspondence $(\hat{\mathbf{x}}_j, \hat{\mathbf{X}}_j) \in \mathcal{M}_n : \hat{\mathbf{X}}_j = \mathbf{X}$ using the Euclidean distance metric:

$$r(\hat{\mathbf{x}}_j, \mathbf{x}) = ||\hat{\mathbf{x}}_j - \mathbf{x}||_2^2$$

The difference between small and large reprojection error in the estimated state is shown in figure 3.22.

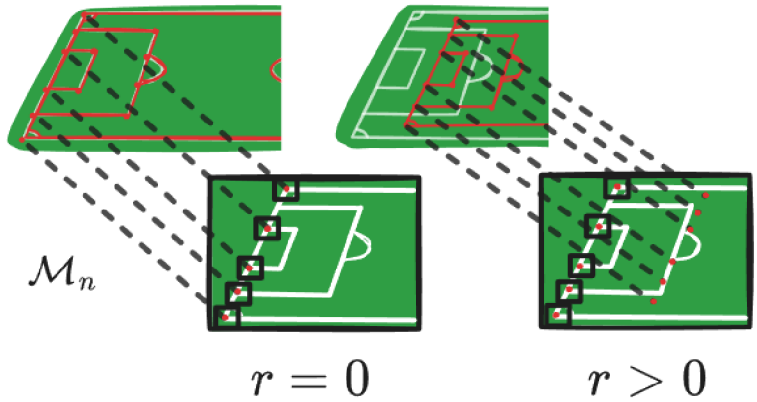


Figure 3.22: Reprojection error. Each landmark of a particle is projected back into 2D and is compared with landmarks in $\mathcal{M}_n$. If we found the landmark locations correctly, the distance between the reprojected pixels and pixels $\mathbf{x} \in \mathcal{M}_n$ is close to zero.

Finally, we assign weight w_n^i to particle x_n^i according to the following likelihood function:

$$w_n^i = \left(\prod_j^{|\mathcal{M}_n|} a \cdot e^{\frac{-r(\hat{\mathbf{x}}_j, \mathbf{x})^2}{2\sigma^2}} + (1-a) \right)$$

This function is a zero-centred Gaussian curve with a minimum at $(1-a)$ and a maximum at 1 (fig. 3.23). The variance σ should be chosen according to observed model convergence. The parameters $\sigma = 40$ and $a = 0.99$ were chosen by experimentation and application of the PF framework best practices.

After this step, we have updated the weights of all particles, allowing us to draw a new estimate of pan-tilt-zoom parameters from the distribution.

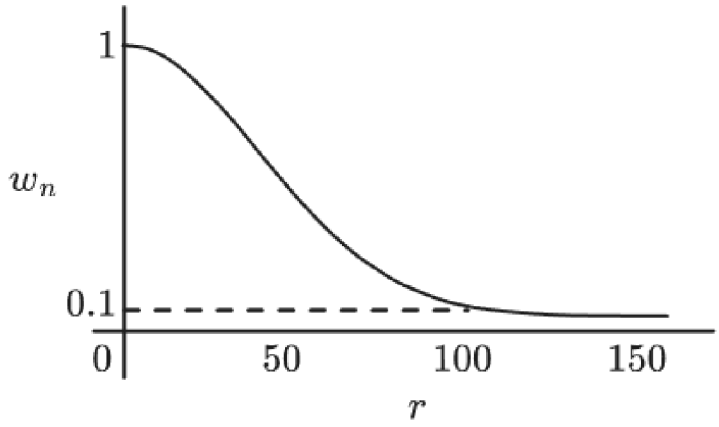


Figure 3.23: Weight likelihood function. The minimum above 0 helps with impoverishment issues, as every particle will have some weight.

3.5 State estimation

The last remaining step after an iteration of the Bootstrap-filter is obtaining an estimation of the parameters $\hat{x}_n = (\hat{\theta}_{pan}, \hat{\theta}_{tilt}, \hat{\theta}_{zoom})$.

Let us return to the beginning of the definition of the particle filter (sec. 3.2.5), where we used the Bayesian theory to define an approximation of the state by the sum of weighted particles. Since the equation 3.1 is the optimal Bayesian approximation of the posterior state [18, sec. 4.1], we define the estimated parameter vector $\hat{x}_n$ as:

$$\hat{x}_n = \sum_{i=1}^{N} x_n^i w_n^i \tag{3.6}$$

The stream of data $\mathcal{S}_n$ is then appended with the current $\hat{x}_n$, allowing seamless processing in the data pipeline by programs.

4 Implementation

The second part of our work involves requirement analysis and subsequent implementation of the method presented in the previous chapter, along with possible system integration procedures.

Because of the scope and nature of this project, we chose an iterative development process suitable for research environments, containing the following iterations:

1. Experimentation and development using existing solutions used in **GOAL SPORT technology s.r.o.**

2. Stand-alone implementation of the method in a separate demo application.

3. Code extraction and refactoring into a library.

4. Development of real use-case applications using the library.

To reduce the complexity of the work required to develop a fully functioning system, we reused specific existing modules from the codebase provided by **GOAL SPORT technology s.r.o.**, namely offline calibration and feature extraction methods (template matching and machine learning models), and rendering. Without these proprietary parts, we would not be able to feed data into the particle filter and then review its performance. The system is summarised in figure 4.1 and a detailed diagram 4.2.

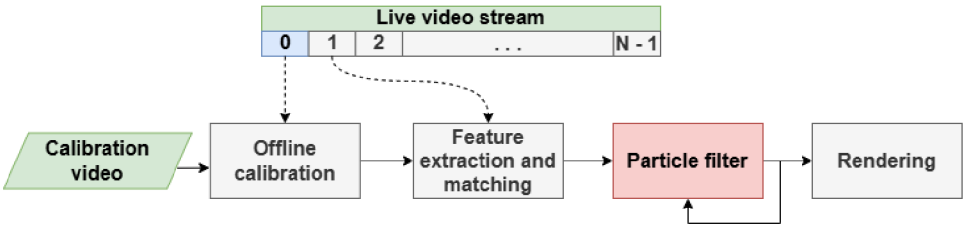


Figure 4.1: Overview system diagram. The system is initially calibrated and then fed a live video stream. The output of the system is used in the rendering module for augmented reality purposes. The particle filter module (in red) has been the main focus of our work.

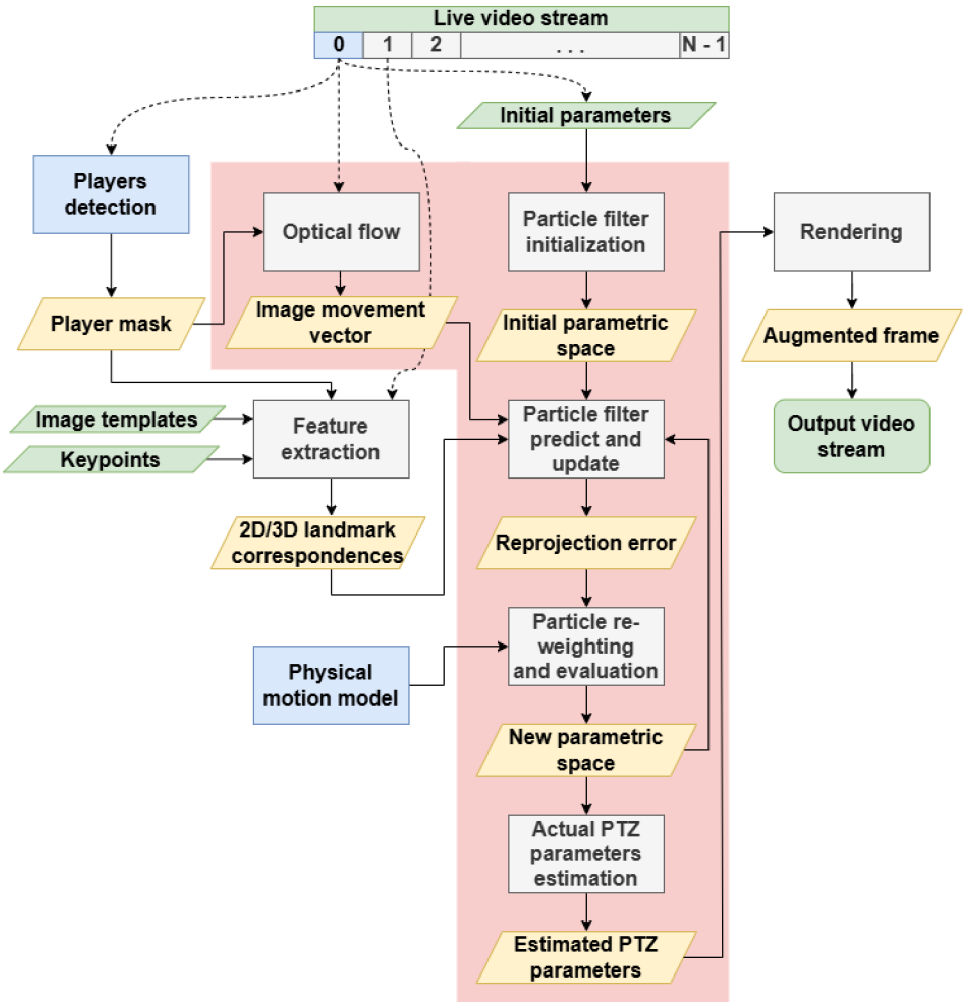


Figure 4.2: Detailed system diagram. The area in red corresponds to the particle filter core. During our research, we recognized modules (in blue) that could help in the development of the method. We left them unimplemented, and they are the subject of further evaluation.

As mentioned in the introduction, the system has been developed in cooperation with **GOAL SPORT technology s.r.o.** and its main use case is to provide continuous camera calibration data to a Video Referee (VAR) software for the rendering of augmented reality overlays. The general requirements of the system have been already specified

in chapter 2.1, however, let us formally define the functional requirements of the system as follows:

- User can specify initial calibration data.
- User can specify algorithm parameters, such as particle count.
- User can specify a video data stream.
- User can view the processed data in real-time.
- The data for every frame has to be stored in a database.
- The data has to be accessible to other applications using an API.

Additionally, the system has the following non-functional requirements:

- Stability - the system will be used in live events, running for several hours.
- Scalability - ability to track data from any number of cameras.
- Speed - real-time processing of 25 FPS video needs max. 40 ms of processing time per frame.
- Data reliability - data has to be sent across the network using reliable protocol.

The specified requirements influenced the selection of technologies and the overall architecture of the system, which is laid out in the next section.

4.1 Technology stack

The selection of a proper programming language has been paramount in fulfilling the specified functional and non-functional requirements. Our research in the field yielded at least three language candidates to choose from — Python, Rust or C++.

Python is used extensively in the development of systems in the field of computer vision, especially because of its simple syntax, ease

of use and a vast amount of helpful libraries with community backing, such as OpenCV, NumPy, SciPy and pandas. This combination of factors makes the language great for quick prototyping. However, Python is often not used in time-critical solutions because of issues with multithreading (Global Interpreter Lock) and a high level of abstraction, often relying on implementation in C or C++ using CPython. In terms of project maintainability, the language often encourages the writing of complex one-liners, which are hard to read. Additionally, Python is dynamically typed and does not have strong OOP encapsulation methods [53, 54, 55]. Because of these issues, we decided to use another language.

Rust is a newer language with a focus on building performance-critical systems. It has gained popularity because of its fresh syntax, combining the best of OOP and functional paradigms. The biggest advantages are built-in concurrency support, no garbage collection and a package manager, resulting in highly reliable and maintainable systems with developing experience similar to Python. On the other hand, the language is not as mature as Python or C++. Thus, the documentation and amount of production-ready libraries are often inadequate [56, 57].

At first, we used Rust during the second iteration of the project while building the demo application, as it seemed to be the most suitable choice for providing a reliant production system. Unfortunately, we encountered several problems along the way. We wanted to use the OpenCV library which is provided only in the form of generated C++ bindings, without any documentation. Moreover, the particle filter is designed in such a way that the computation alters the internal state in a loop. This caused problems with the variable mutability rules of the Rust compiler. Compared to C++, the compile time was also much longer. These issues resulted in a very slow development cycle, and we decided to use C++ instead.

C++ is the language we worked with in the first iteration because it is used in **GOAL SPORT technology s.r.o.** systems. Choosing a different language for the library meant that we would have to make a C-interoperable ABI to communicate with C++ code. We thought that the benefits of Rust would outweigh the necessary additional work. Still, the most critical parts of the C++ code could be rewritten in concurrent, memory-safe Rust.

We combined C++ with the Qt 6 framework, extending the standard C++ library for rapid development, both in GUI and CLI efforts. For the mathematical method itself, we employed the OpenCV and Eigen libraries.

4.2 Development

The project has been developed using the Visual Studio Code IDE, along with the CMake build system and Git version control system. The Git repository uses submodules for proprietary libraries *libgoal* and *vol*, both necessary to provide calibration, feature matching and rendering modules. Our code resides in the **pf** subfolder, containing the source code of different parts of the project:

- **lib** — library of our method, given the name **RePTZ**.
- **demo** — Qt QML GUI application, which loads a video file and initial calibration file from VAR, feeds it into the **RePTZ** library, and then shows debug visualisations.
- **cli** — Qt CLI application, loads frames in the form of PNG images and calibration in the form of a JSON file. It provides various export options, an interactive mode that displays the current rendered state, and optionally acts as a server.
- **client** — an example client application written in TypeScript, sends PNG frames into CLI in server mode, receives the result and saves it into a database, and also provides a REST API endpoint for pose and statistic querying.

There is an additional **utils** subfolder that contains Python, R and Bash scripts used for data analysis and method evaluation.

Development occurred inside of a container provided by **GOAL SPORT technology s.r.o.** using Docker, which saved time setting up Linux system dependencies. Provided applications and libraries were built targeting the Linux platform, however, Windows deployment should also be possible, as we used cross-platform technologies. The first implementation inside of the VAR system is shown in figure 4.3.

Figure 4.3: Demo inside of VAR application. Particle filter is directly embedded in the system because of easy access to calibration and video data. A pitch (in red) is rendered according to the estimated state and overlaid for debugging purposes.

4.3 RePTZ library

The particle filter method defined in chapter 3 has been implemented in the form of C++ classes — **ParticleFilter**, **MotionDetection**, and **HomographyDecomposition**. The motion detection and homography decomposition methods are used in the particle filter predict routine. The public API of the library resides in header files located in the **include** directory, and is documented by class diagrams in figures 4.5 and 4.6.

The methods of the **ParticleFilter** class provide building blocks that can be used to control the operation of the particle filter. There are two main routines that can be built using the API — initialization and frame loop.

The consumer of the API should first initialize the **ParticleFilter** instance by calling *initialize* and *predict* methods with initial frame and PTZ parameters. Then, from each next frame, features should be extracted through methods of choice. The quality of the matching is then assessed, and a relocalization routine should be called if

necessary. The previous state is then updated with the data, and an estimated parametric vector can be used. Afterwards, a state prediction is made and optionally resampled. This approach is documented with a swimlane diagram in figure 4.4.

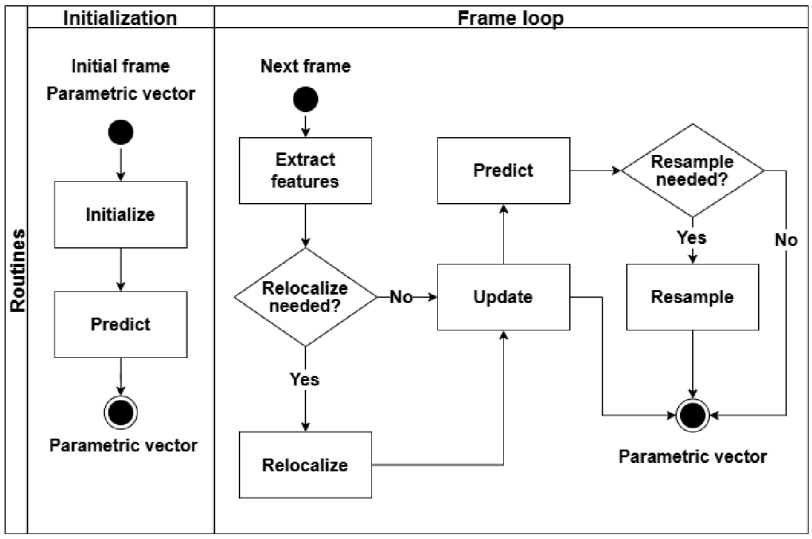


Figure 4.4: RePTZ library API routines.

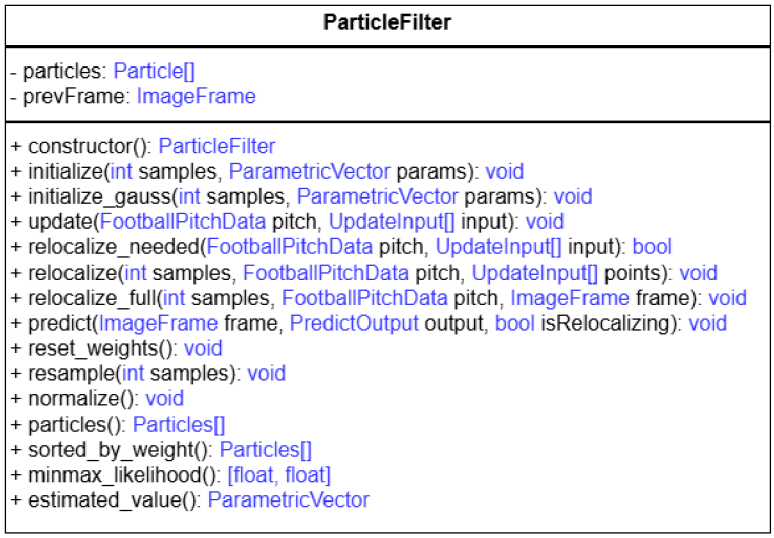


Figure 4.5: Particle filter class diagram.

```
MotionDetection
---
+ getBinOfOrientation(float orientation): Quadrant
+ isPanning(int totalVectors, MotionHistogram hist): MotionState
+ filterSignificantVectors(Points a, Points b): [Points, Points]
+ getMotionFieldHistogram(Points a, Points b): MotionHistogram
+ findDominantMotion(MotionHistogram hist): [Quadrant, int]
+ detectZoomChange(int width, int height,
                   Points a, Points b): MotionState
+ findCameraMotionState(int width, int height,
                        Points a, Points b):
                        [MotionState, MotionHistogram]
+ getMotionDescription(MotionState state): string
```

```
HomographyDecomposition
---
+ getHomographyBetweenPoints
    (Points a, Points b): HomographyMatrix
+ getRotationBetween
    (ImageFrame a, ImageFrame b,
     CameraMatrix cm, Points a, Points b): ParametricVector
+ getRotationMatrixFromHomography
    (HomographyMatrix h, CameraMatrix cm): RotationMatrix
+ getFocalLengthScaleFromHomography
    (HomographyMatrix h): float
```

Figure 4.6: Utility classes diagrams.

4.4 Demo

The principles from the previous section were applied to a Qt QML application, showcasing the full operation of the particle filter.

At the start of the application, a GSS project file is loaded, containing initial calibration data and templates of pitch keypoints. Similarly to the VAR system, a video is played using a proprietary LibAV decoder, and the decoded frames are fed into the initialization and frame loop routines. For the updating process, we use both the template matching and keypoint detection methods. The estimated parametric state is then rendered onto the current image frame, along with important information about the state, such as the camera direction, detected keypoints, matched templates, and the difference between the previous state. Quantitative parameters, such as the actual PTZ values and total computing time are also embedded into the image. A single exported frame is shown in figure 4.7.

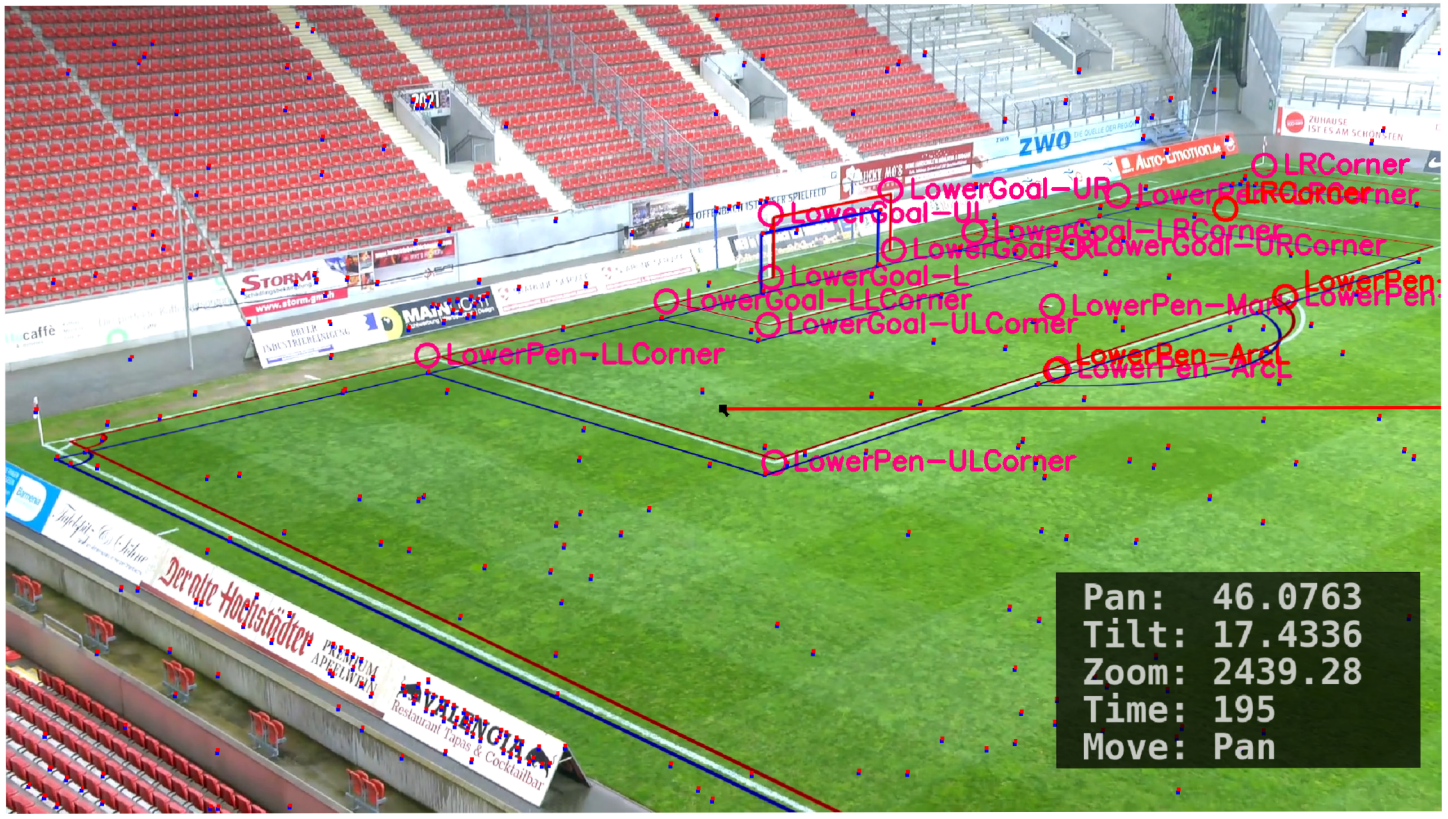


Figure 4.7: RePTZ demo application. It contains the actual updated (red) and the last predicted (blue) state. Pixel blocks signify points selected for homography estimation, and how they moved between frames. There are also found templates (magenta) and keypoints (red). In the middle, the histogram shows the dominant motion with a red line. Lastly, there is a debug window with computed values.

4.5 Architecture

We analysed the possible integration of **RePTZ** into **GOAL SPORT technology s.r.o.** processes during game matches and determined that to accommodate all the requirements (p. 42), a client-server architecture is the best option. A monolithic architecture could be bottlenecked by hardware resources and there are not enough services or use cases for service-oriented or microservices architecture.

Each camera subject to continuous recalibration can be treated as a single client that needs to communicate with the **RePTZ** server. Another client can be used to read the data from a database. Multiple servers can be hosted depending on available resources, either on single or multiple machines. This way, redundancy of servers is also easily achieved.

4.5.1 Server / CLI

The server has been implemented as an optional feature of the command-line interface program utilising **RePTZ**. It uses Qt command-line parsing, HTTP server and TCP web sockets. After a frame is processed, useful information is printed into the terminal output, such as shown in figure 4.8.

The REST API contains a simple heartbeat GET route that can be used to check whether the server is ready to accept the next frame or if an error happened. Then, a new frame can be sent to the server with a POST request containing metadata and an image as a base64 encoded string. If the request is accepted, a response data transfer object (DTO) JSON that contains all computed data and statistics is sent back. An OpenAPI documentation (fig. 4.9) YAML file with all DTO models, routes, and examples, is available in the **doc** subfolder.

The CLI is also used to process decoded frames from video in the form of PNG image sequences. Processed data can be visualised in an OpenCV window or saved to disk as JSON files or images.

We decided to omit the template matching algorithm from this program, as the decomposition from the VAR software project files would be non-trivial.

```
----------------Frame  128----------------
File | "0128.png"
Time | Keypoints Inference: 85.7986 ms
Time | Relocalize: 0.01677 ms
Time | Update: 2.33049 ms
Time | Predict: 14.578 ms
Time | Resample: 0.03192 ms
Time | Total: 102.812 ms
Estimated pose | 0.0423549 0.453609 3078.3
Time | Render: 42.1564 ms
-----------------------------------------
```

Figure 4.8: RePTZ CLI terminal output. Contains the image name, per-routine computation times and estimated pose parameters.

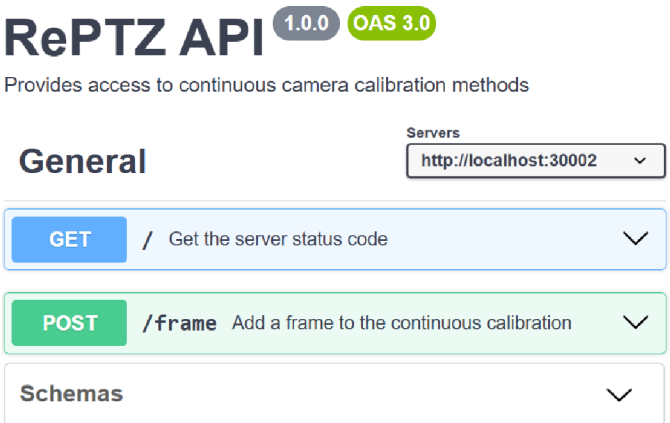


Figure 4.9: REST API documentation. Swagger was used to transform the OpenAPI documentation file into a user-friendly web interface.

4.5.2 Client

The last of our integration efforts has been the implementation of an example client application. We have chosen to build it in TypeScript and Node.js because the application is not time-critical, and the ecosystem offers built-in support for JSON and interaction with the web. We used libraries such as *axios*, *express.js* and *command-line-args*.

The client has several command-line options, such as the path to the directory with PNG images and **RePTZ** server IP and port. In real

production use cases, the image sequence can be replaced with NDI, SDI or a video stream.

The image sequence is sent to the server and per each frame, the received JSON is stored in a MongoDB database as a time-series data collection. This allows for optimal temporal data retrieval. We do not use a relational database, such as PostgreSQL, because we have no referential constraints, only one table and the data access should be as quick as possible. Additionally, MongoDB is a NoSQL database with native support of JSON data structures. The repository pattern is used to send commands to the database, increasing the maintainability of the codebase.

The MongoDB collection data is consumable by visualisation programs over an HTTP endpoint or a direct database connection. We also provide a Docker Compose YAML configuration to set up the MongoDB database and Grafana, which has been used to build a real-time updated data dashboard (fig. 4.10).

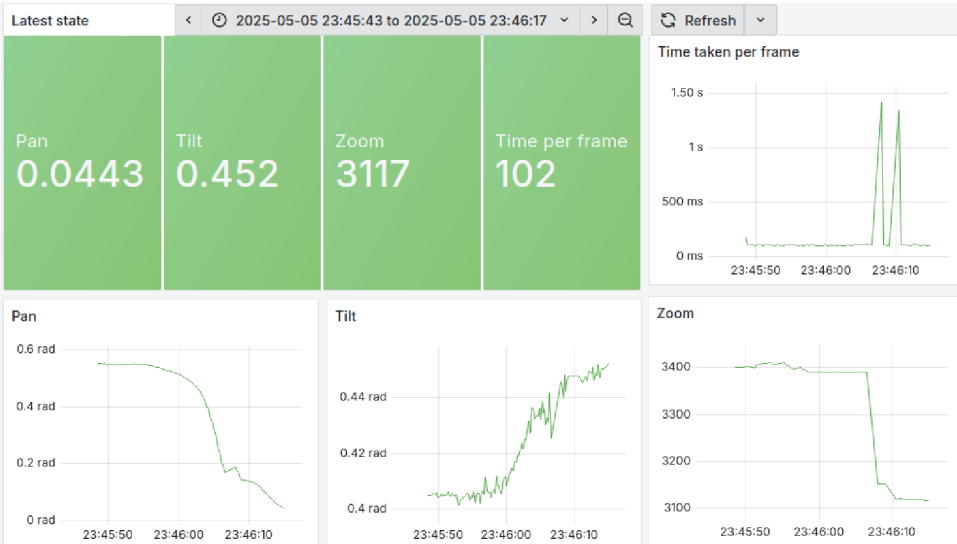


Figure 4.10: Grafana dashboard. Configured with Infinity data source plugin to read the exposed JSON array. Direct MongoDB connection was not used as it is a paid feature. Grafana could be used to aggregate all the client data into one dashboard, potentially over the cloud.

5 Evaluation

To evaluate the performance of our method, we used two videos provided by **GOAL SPORT technology s.r.o.**, comparing the state-of-the-art **PTZ-SLAM** with **RePTZ**.

5.1 Methodology

At first, we tried to evaluate the performance of **PTZ-SLAM** using the default configuration with the mapping component enabled. The first batches were completed successfully; however, as more key-frames were added to the map, the subsequent frames were taking exponentially more time to compute. Our experience with **PTZ-SLAM** is similar to research made by authors in the *BroadTrack* paper [4], where they had to omit the comparison of this method. Therefore, we decided to disable the mapping component during the testing.

For **RePTZ**, we kept the α, β, and γ parameters at default values (defined in the library code) to eliminate bias between datasets, and evaluated the batches with particle quantity of $k = 1000$. We also saved the rendered estimated pitch overlays for demonstration purposes.

As a part of the methodology, we evaluate the pan-tilt-zoom values, computation time and cumulative error. We have chosen the Normalized Root Mean Square Error (NRMSE) metric to compare the relative performance between methods using the PTZ time-sequence data per batch. Used datasets and their evaluation results are presented in the following section.

5.2 Results

The first dataset was created from a 3-minute calibration video taken before a match, resulting in 4320 frames. Therefore, there are no players or other people in the field of play. The camera is moved smoothly from the left of the field to the right, with varying quick or slow zooming patterns.

We split the 3-minute video into 35 batches that contain 125 frames each (except the last batch), to evaluate the accuracy per smaller move-

ment patterns. This allows us to recognize areas where the method is lacking and improve the accuracy in the future.

To evaluate also the accuracy of **PTZ-SLAM**, we generated ground truth data for all the frames using **GOAL SPORT technology s.r.o.** automated offline calibration methods. However, the quality of the generated data is lower in batches without many visible keypoints. Problematic samples can be distinguished by high variability, especially in the zoom parameter. Nonetheless, we kept the data in the comparison, as the majority was manually checked as correct.

A complete list of evaluations per batch, ordered from best accuracy to worst, can be found in appendix A and more detailed results in appendix B.

The second dataset comprises 128 manually annotated frames from a 5-second video of real soccer gameplay with players. The figure 5.1 shows that in this scenario, the **RePTZ** method performed faster and more accurate than state-of-the-art **PTZ-SLAM**.

We summarised our results in tables 5.1 and 5.2, containing averages and standard deviations for differences in PTZ parameters in comparison with the ground truth, including the accuracy error and computation time per method and dataset.

Table 5.1: Average values per dataset.

Dataset	Method	Average				
		Pan Diff.	Tilt Diff.	Zoom Diff.	Time	Error
1	PTZ-SLAM	7.8	18.4	139.3	502.5	1.9
1	RePTZ	13.8	25	134.2	105.5	4.7
2	PTZ-SLAM	106.4	29.3	93.4	376.7	3.9
2	RePTZ	16.3	8.5	61.3	112.4	1.3

Table 5.2: Standard deviation values per dataset.

Dataset	Method	Standard deviation				
		Pan Diff.	Tilt Diff.	Zoom Diff.	Time	Error
1	PTZ-SLAM	12.2	43.1	725.3	293.3	3.3
1	RePTZ	19.6	38.5	644.3	62.7	6.1
2	PTZ-SLAM	113.1	22.9	73.3	99.4	5.1
2	RePTZ	14.3	8.9	89.4	120.2	1.8

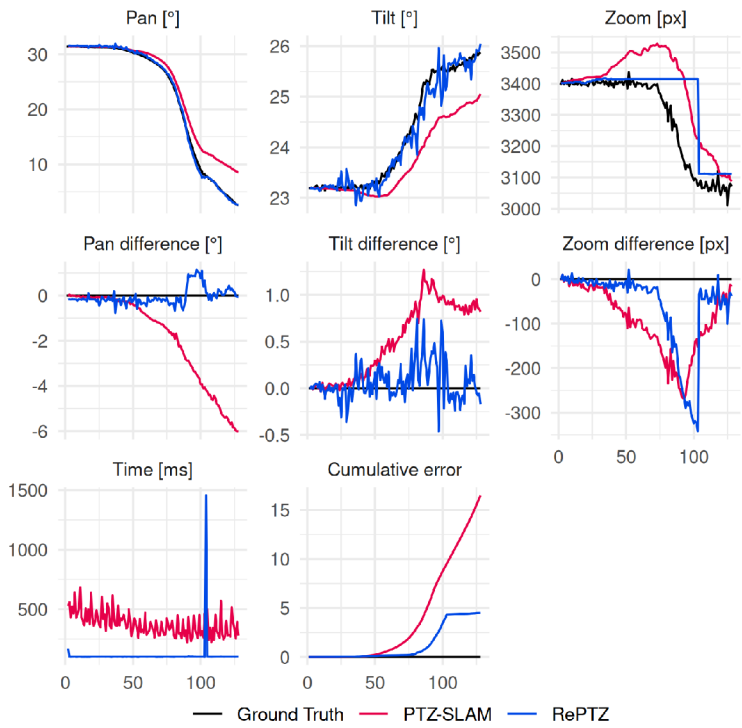


Figure 5.1: Statistics for the dataset 2. In the live match scenario, **RePTZ** has higher accuracy and lower computation time than **PTZ-SLAM**. The spike around the 100th frame was caused by the relocalization procedure.

6 Conclusion

The data presented in the previous chapter shows that our novel method is ~3 to 5 times faster, but ~2.5 times less accurate on average than the state-of-the-art **PTZ-SLAM**. We noticed a large discrepancy between the results of the two chosen datasets. We assume that the live match video is very difficult to process using SIFT descriptors used by **PTZ-SLAM**. The accuracy of our method in the first dataset could be potentially improved by tweaking the α, β, γ parameters. Regarding the performance, we noticed that most of the computing time was taken by the feature detection methods.

We comprehend that our results do not reflect an accurate comparison of the methods, primarily caused by the lack of high-quality, manually annotated ground truth image frames. In the future, a proper assessment could be made using public datasets provided in the EVS camera calibration challenge.

There are many paths left unexplored that could make the particle filter more robust, such as the use of player masks to remove unwanted correspondence points and a better physical motion model that also includes the camera movement speed. Experimentation with various resampling methods or proposal distribution functions could yield valuable insight. The relocalization method should be replaced with a proper mapping component to reduce sudden computation spikes.

From the system integration standpoint, we believe that the proposed architecture is sufficient to accommodate all use cases of **GOAL SPORT technology s.r.o.** The developed command-line tools can be used for further experimentation with real data from cameras. The speed of the application could be improved by better hardware, especially by using a fast dedicated GPU for machine-learning models. The particle filter is also easily parallelizable, as each particle can be computed independently. In the future, the datasets can be used for parametric auto-tuning or automated regression testing.

Overall, we believe that we have shown that the particle filter can be used as a replacement for EKF-based pose estimation algorithms in real-time environments, offering great flexibility.

Bibliography

1. GUTIÉRREZ-PÉREZ, Marc; AGUDO, Antonio. No bells just whistles: Sports field registration by leveraging geometric properties. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 3325–3334.
2. THEINER, Jonas; EWERTH, Ralph. Tvcalib: Camera calibration for sports field registration in soccer. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2023, pp. 1166–1175.
3. LU, Jikai; CHEN, Jianhui; LITTLE, James J. Pan-tilt-zoom SLAM for sports videos. *arXiv preprint arXiv:1907.08816*. 2019.
4. MAGERA, Floriane; HOYOUX, Thomas; BARNICH, Olivier; VAN DROOGENBROECK, Marc. BroadTrack: Broadcast Camera Tracking for Soccer. *arXiv preprint arXiv:2412.01721*. 2024.
5. FIFA. *Pitch dimensions and surrounding areas* [online]. FIFA, 2024 [visited on 2024-12-14]. Available from: `https://publications.fifa.com/de/football-stadiums-guidelines/technical-guideline/stadium-guidelines/pitch-dimensions-and-surrounding-areas/`.
6. LOPES, Rui. *Basics of camera movement* [online]. 2019. [visited on 2024-12-14]. Available from: `https://ruieduardolopeshome.wordpress.com/2019/03/16/basics-of-camera-movement/`.
7. HARTLEY, Richard; ZISSERMAN, Andrew. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
8. MEIDOW, Jochen; KIRCHHOF, Michael. *Continuous self-calibration and ego-motion determination of a moving camera by observing a plane*. Citeseer, 2007.
9. ALAM, Md Shah; OLUOCH, Jared. A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs). *Expert Systems with Applications*. 2021, vol. 179, p. 115091.

10. BAI, Yuhao; ZHANG, Baohua; XU, Naimin; ZHOU, Jun; SHI, Jiayou; DIAO, Zhihua. Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review. *Computers and Electronics in Agriculture*. 2023, vol. 205, p. 107584.

11. THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic Robotics* (*Intelligent Robotics and Autonomous Agents*). The MIT Press, 2005. ISBN 0262201623.

12. URWIN, Matthew. *Robotics: What Are Robots?* [online]. Built In, 2024 [visited on 2024-11-23]. Available from: `https://builtin.com/robotics`.

13. LIU, Jun S; CHEN, Rong; LOGVINENKO, Tanya. A theoretical framework for sequential importance sampling with resampling. In: *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 225–246.

14. TOKDAR, Surya T; KASS, Robert E. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*. 2010, vol. 2, no. 1, pp. 54–60.

15. EWING, Eric; ZHANG, Connie; STRAWN, Kegan; HANSPAL, Pushpreet Singh. *Lecture 6: Bayesian and Kalman Filters* [online]. University of Southern California, 2019 [visited on 2024-11-23]. Available from: `https://stefanosnikolaidis.net/course-files/CS545/Lecture6.pdf`.

16. BAGNELL, Drew; ACHAR, Supreeth. *Lecture 2: Filtering Theory* [online]. Carnegie Mellon University, [n.d.] [visited on 2024-11-23]. Available from: `https://www.cs.cmu.edu/~16831-f14/notes/F09/lec02/16831_lecture02.sachar.pdf`.

17. MAVROGIANNIS, Chris; CHOUDHURY, Sanjiban. *Bayes filtering: A deeper dive* [online]. University of Washington Computer Science and Engineering, [n.d.] [visited on 2024-11-23]. Available from: `https://courses.cs.washington.edu/courses/cse478/20wi/site/resources/lec4_bayesderiv.pdf`.

18. ELFRING, Jos; TORTA, Elena; VAN DE MOLENGRAFT, René. Particle filters: A hands-on tutorial. *Sensors*. 2021, vol. 21, no. 2, p. 438.

19. BISHOP, Christopher M. Pattern recognition and machine learning. *Springer google schola*. 2006, vol. 2, pp. 1122–1128.

20. WELCH, Greg; BISHOP, Gary. *An Introduction to the Kalman Filter* [online]. University of North Carolina at Chapel Hill, 2006 [visited on 2024-11-23]. Available from: `https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf`.

21. NEGENBORN, Rudy. Robot Localization and Kalman Filters On finding your position in a noisy world. *ser. Utrecht University, Utrecht, Netherlands*. 2003.

22. KHODARAHMI, Masoud; MAIHAMI, Vafa. A review on Kalman filter models. *Archives of Computational Methods in Engineering*. 2023, vol. 30, no. 1, pp. 727–747.

23. KÜNSCH, Hans R. Particle filters. *Bernoulli*. 2013, vol. 19, no. 4, pp. 1391–1403. Available from DOI: `10.3150/12-BEJSP07`.

24. HAMMERSLEY, J. M.; MORTON, K. W. Poor Man's Monte Carlo. *Journal of the Royal Statistical Society. Series B (Methodological)* [online]. 1954, vol. 16, no. 1, pp. 23–38 [visited on 2024-11-24]. ISSN 00359246. Available from: `http://www.jstor.org/stable/2984008`.

25. ROSENBLUTH, Marshall N; ROSENBLUTH, Arianna W. Monte Carlo calculation of the average extension of molecular chains. *The Journal of chemical physics*. 1955, vol. 23, no. 2, pp. 356–359.

26. HERBST, Edward; SCHORFHEIDE, Frank. Tempered particle filtering. *Journal of Econometrics*. 2019, vol. 210, no. 1, pp. 26–44.

27. DOUCET, Arnaud; GODSILL, Simon; ANDRIEU, Christophe. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing*. 2000, vol. 10, pp. 197–208.

28. DOUCET, Arnaud; JOHANSEN, Adam M, et al. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*. 2009, vol. 12, no. 656-704, p. 3.

29. TIAN, Yang; MA, Shugen. Kidnapping detection and recognition in previous unknown environment. *Journal of Sensors*. 2017, vol. 2017, no. 1, p. 6468427.

30. CANDY, James V. Bootstrap particle filtering. *IEEE Signal Processing Magazine*. 2007, vol. 24, no. 4, pp. 73–85.

31. KEMP, Freda. *An introduction to sequential Monte Carlo methods*. Oxford University Press, 2003.

32. LI, Tiancheng; BOLIC, Miodrag; DJURIC, Petar M. Resampling methods for particle filtering: classification, implementation, and strategies. *IEEE Signal processing magazine*. 2015, vol. 32, no. 3, pp. 70–86.

33. DOUC, Randal; CAPPÉ, Olivier. Comparison of resampling schemes for particle filtering. In: *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.* Ieee, 2005, pp. 64–69.

34. HOL, Jeroen D; SCHON, Thomas B; GUSTAFSSON, Fredrik. On resampling algorithms for particle filters. In: *2006 IEEE nonlinear statistical signal processing workshop*. IEEE, 2006, pp. 79–82.

35. GLUCKMAN, Joshua; NAYAR, Shree K. Ego-motion and omnidirectional cameras. In: *Sixth International Conference on Computer Vision* (*IEEE Cat. No. 98CH36271*). IEEE, 1998, pp. 999–1005.

36. SINGH, Chahat Deep. *Structure from Motion* [online]. University of Maryland, 2024 [visited on 2024-12-14]. Available from: `https://cmsc426.github.io/sfm/`.

37. BOESCH, Gaudenz. *Optical Flow – Everything You Need to Know in 2025* [online]. viso.ai, 2024-10 [visited on 2024-12-11]. Available from: `https://viso.ai/deep-learning/optical-flow/`.

38. ALFARANO, Andrea; MAIANO, Luca; PAPA, Lorenzo; AMERINI, Irene. Estimating optical flow: A comprehensive review of the state of the art. *Computer Vision and Image Understanding*. 2024, p. 104160.

39. NIEHORSTER, Diederick C. Optic flow: a history. *i-Perception*. 2021, vol. 12, no. 6, p. 20416695211055766.

40. LIN, Chuan-en. *A Comprehensive guide to Motion Estimation with Optical Flow* [online]. Nano Net Technologies Inc., 2019-04 [visited on 2024-12-11]. Available from: `https://nanonets.com/blog/optical-flow/`.

41. PRINCE, Simon JD; XU, Ke; CHEOK, Adrian David. Augmented reality camera tracking with homographies. *IEEE Computer graphics and Applications*. 2002, vol. 22, no. 6, pp. 39–45.

42. XU, Ke; CHIA, Kar Wee; CHEOK, Adrian David. Real-time camera tracking for marker-less and unprepared augmented reality environments. *Image and Vision Computing*. 2008, vol. 26, no. 5, pp. 673–689.

43. MALIS, Ezio; VARGAS VILLANUEVA, Manuel. Deeper understanding of the homography decomposition for vision-based control. 2007.

44. MAKKAPATI, V. Robust camera pan and zoom change detection using optical flow. In: *National conference on computer vision, pattern recognition, image processing and graphics*. 2008, pp. 73–78.

45. THOMAS, Graham. Real-time camera tracking using sports pitch markings. *Journal of Real-Time Image Processing*. 2007, vol. 2, no. 2, pp. 117–132.

46. CHANDARIA, Jigna; THOMAS, Graham A; STRICKER, Didier. The MATRIS project: real-time markerless camera tracking for augmented reality and broadcast applications. *Journal of Real-Time Image Processing*. 2007, vol. 2, pp. 69–79.

47. DEL BIMBO, Alberto; LISANTI, Giuseppe; MASI, Iacopo; PERNICI, Federico. Continuous recovery for real time pan tilt zoom localization and mapping. In: *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2011, pp. 160–165.

48. LI, Yunpeng; SNAVELY, Noah; HUTTENLOCHER, Daniel P. Location recognition using prioritized feature matching. In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part II 11*. Springer, 2010, pp. 791–804.

49. MIKSIK, Ondrej; MIKOLAJCZYK, Krystian. Evaluation of local detectors and descriptors for fast feature matching. In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE, 2012, pp. 2681–2684.

50. BRUNELLI, Roberto. *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009.

51. DORE, Alessio; PINASCO, Matteo; REGAZZONI, Carlo S. *Multi-Modal Data Fusion Techniques and Applications.* 2009.

52. ALTWAIJRY, Hani; VEIT, Andreas; BELONGIE, Serge J; TECH, Cornell. Learning to detect and match keypoints with deep architectures. In: *BMVC*. 2016.

53. ALDRIDGE, David. *Shorter code, fewer bugs: Python's syntax is its saving grace* [online]. TechTarget, 2023 [visited on 2025-05-08]. Available from: `https://www.theserverside.com/tip/Shorter-code-fewer-bugs-Pythons-syntax-is-its-saving-grace`.

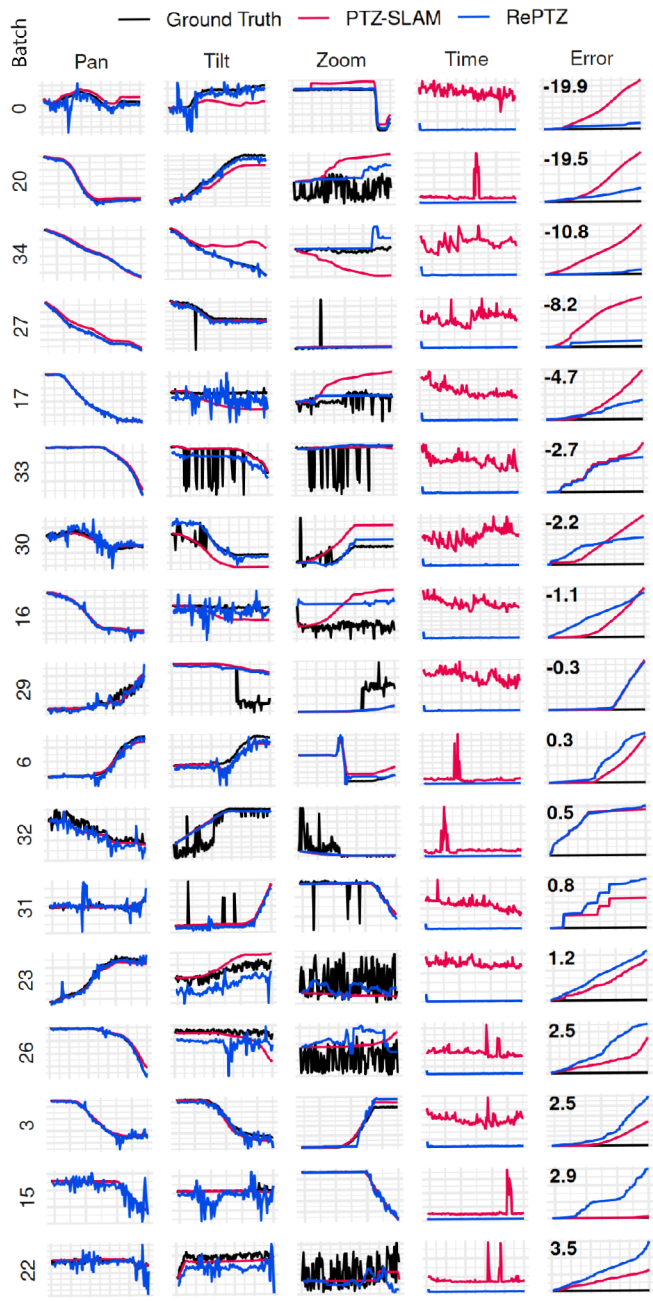
54. CHAWLA, Avi. *What's Missing from Python OOP Encapsulation* [online]. Daily Dose of Data Science, 2024 [visited on 2025-05-08]. Available from: `https://blog.dailydoseofds.com/p/whats-missing-from-python-oop-encapsulation`.

55. SHARMA, Gaurav. *Comparing Python to other popular programming languages* [online]. Medium, 2022 [visited on 2025-05-08]. Available from: `https://medium.com/codex/comparing-python-to-other-popular-programming-languages-15fb757aeaa1`.

56. VERDI, Sara. *Why Rust is the most admired language among developers* [online]. GitHub, Inc., 2023 [visited on 2025-05-08]. Available from: `https://github.blog/developer-skills/programming-languages-and-frameworks/why-rust-is-the-most-admired-language-among-developers/`.

57. PACHECO, Diego. *The dark side of Rust Language* [online]. Medium, 2020 [visited on 2025-05-08]. Available from: `https://medium.com/@ilegra/the-dark-side-of-rust-language-4fe2b9c2faf3`.

A Method evaluation summary



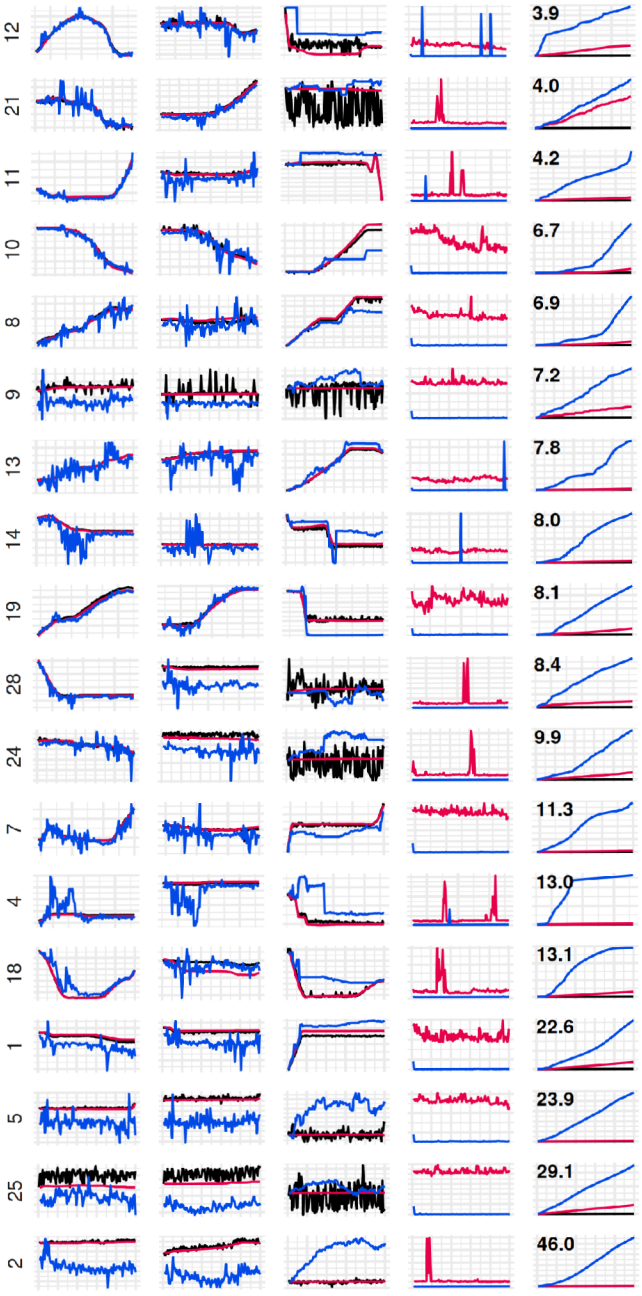


Figure A.1: Evaluation of methods and ground truth per batch. The batches are ordered by the accuracy of **RePTZ** with 1000 particles, from the best to the worst. 9 batches have higher accuracy than **PTZ-SLAM**, 11 batches have slightly lower accuracy (error < 5) and 15 batches have significantly lower accuracy (> 5).

B Detailed statistics

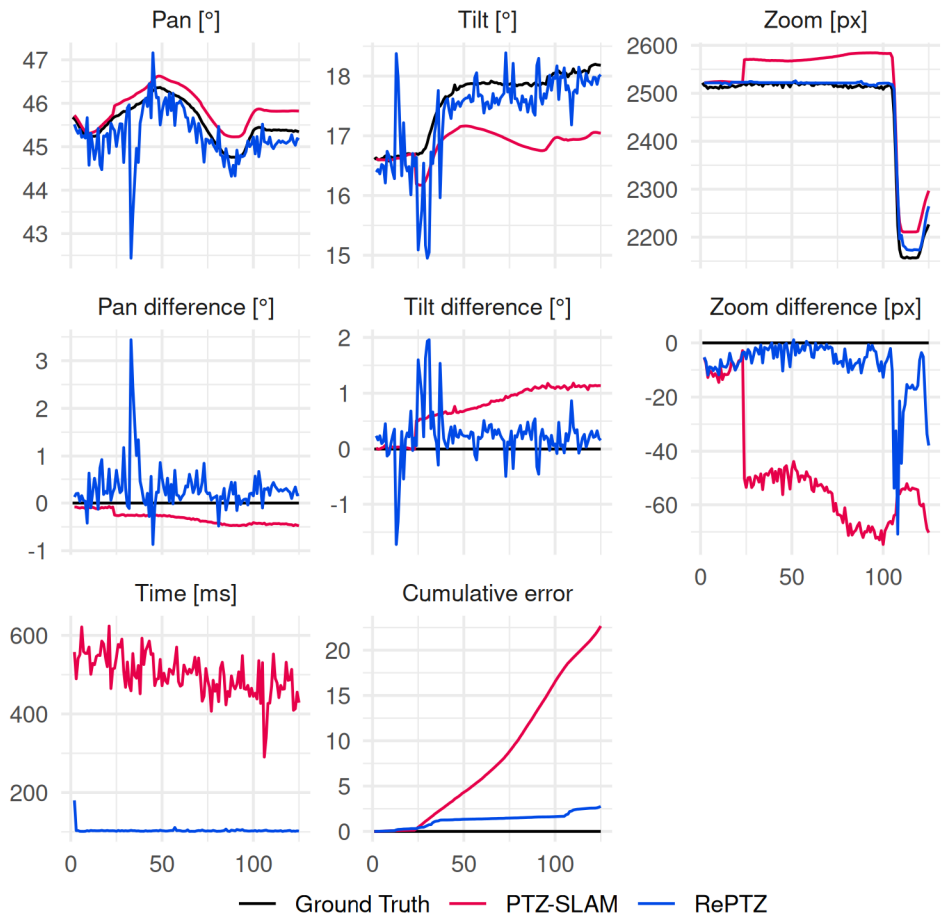


Figure B.1: Statistics for dataset 1 batch 0.

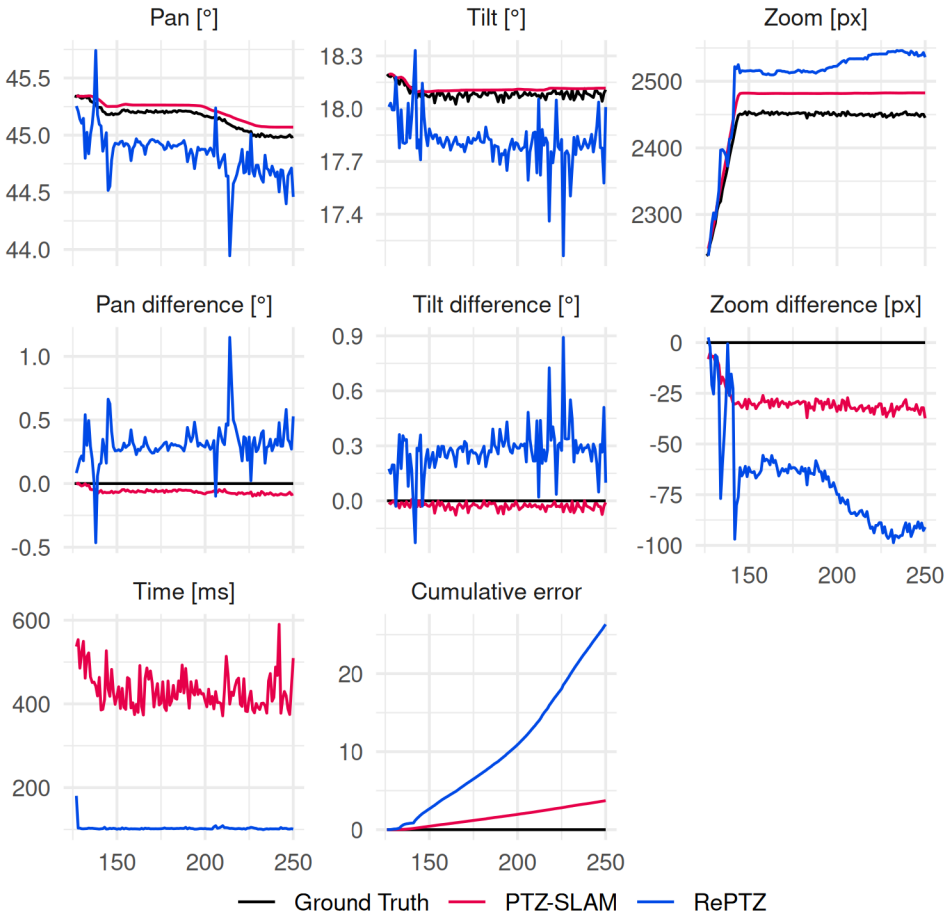


Figure B.2: Statistics for dataset 1 batch 1.

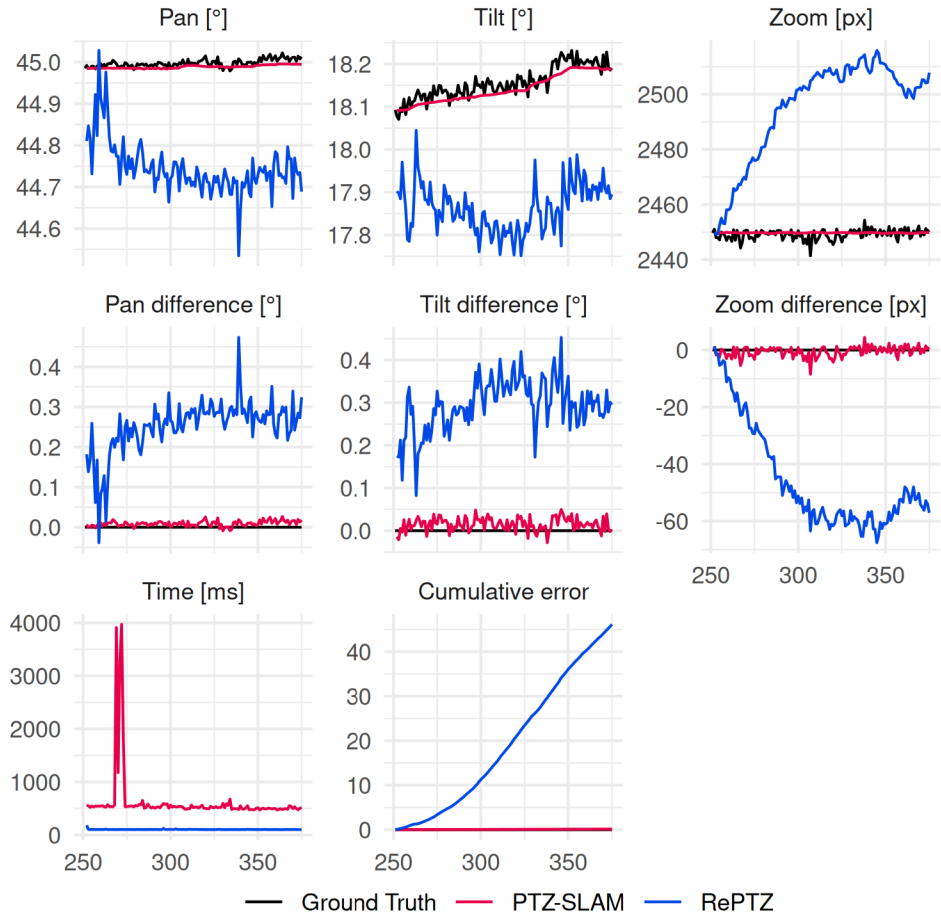


Figure B.3: Statistics for dataset 1 batch 2.

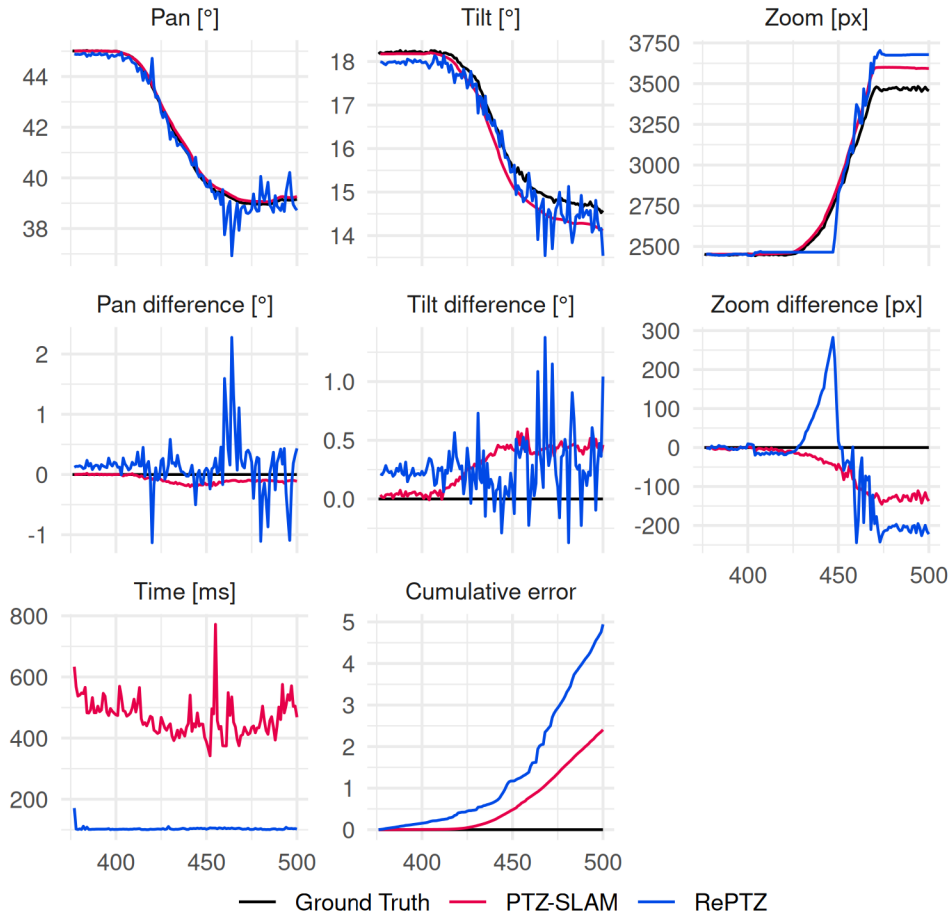


Figure B.4: Statistics for dataset 1 batch 3.

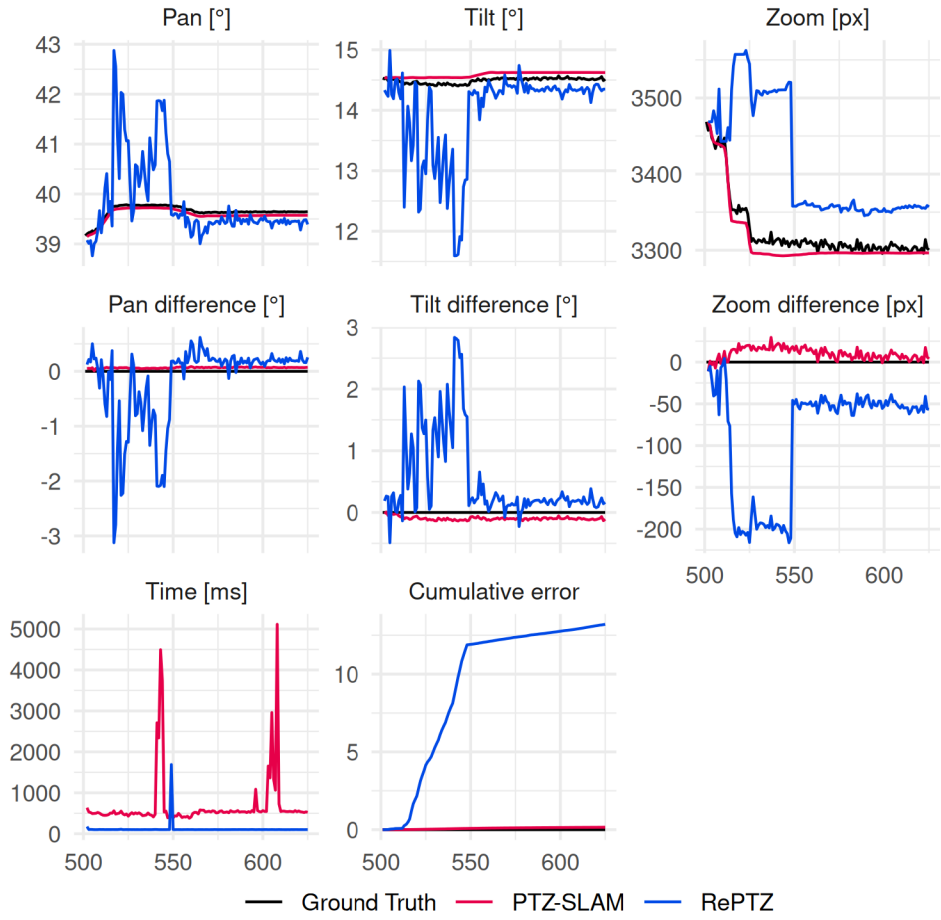


Figure B.5: Statistics for dataset 1 batch 4.

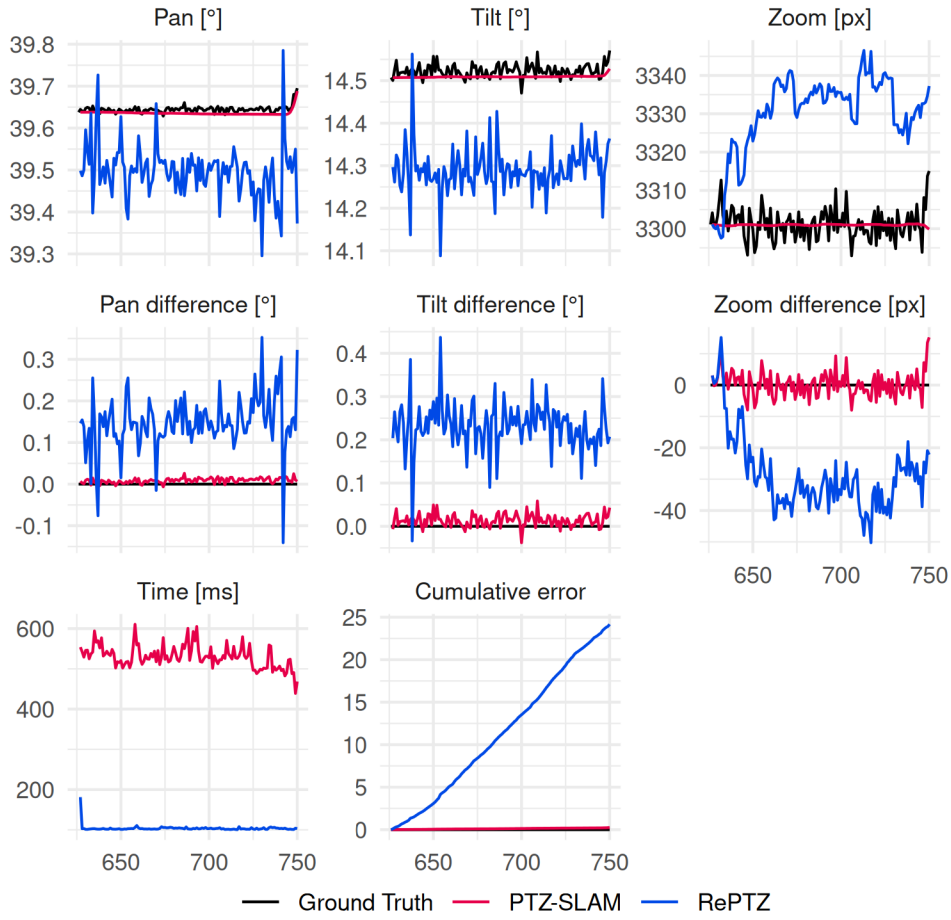


Figure B.6: Statistics for dataset 1 batch 5.

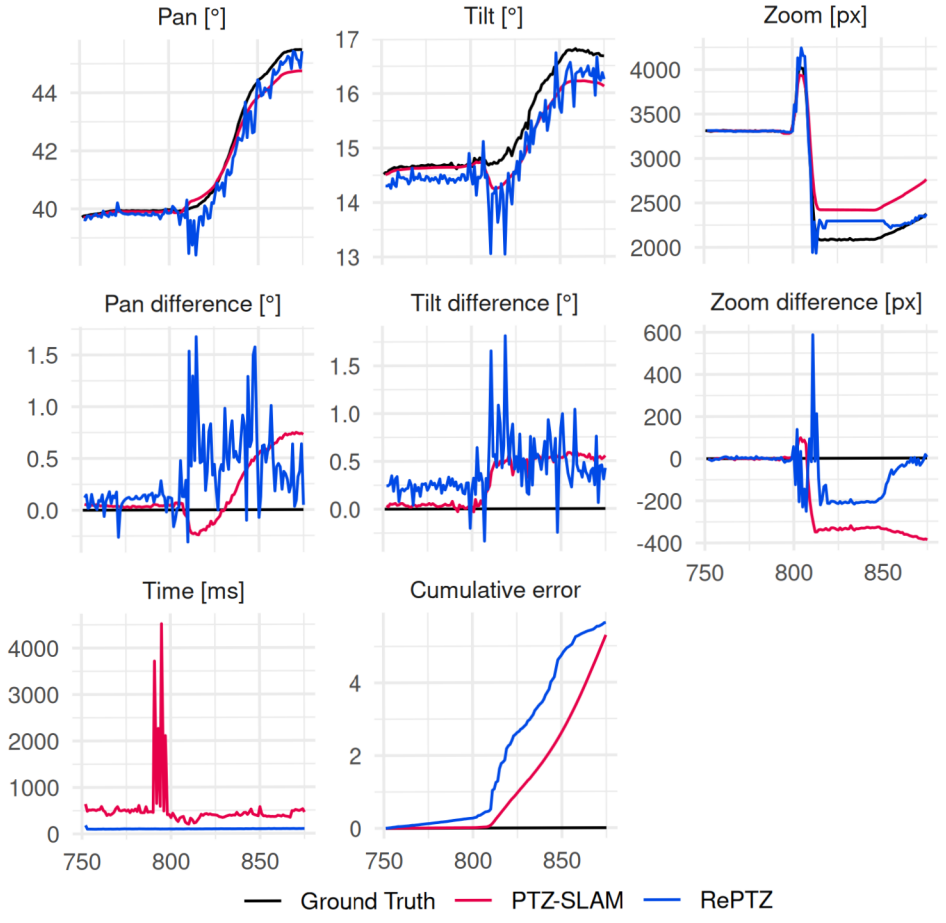


Figure B.7: Statistics for dataset 1 batch 6.

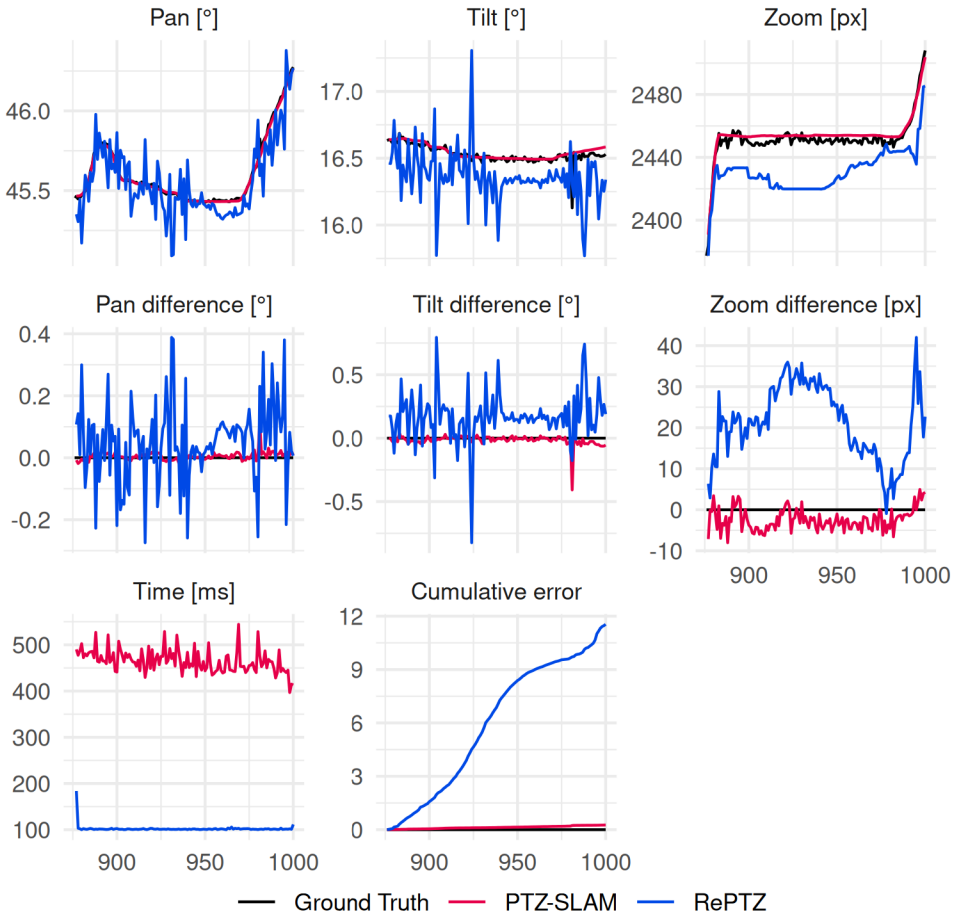


Figure B.8: Statistics for dataset 1 batch 7.

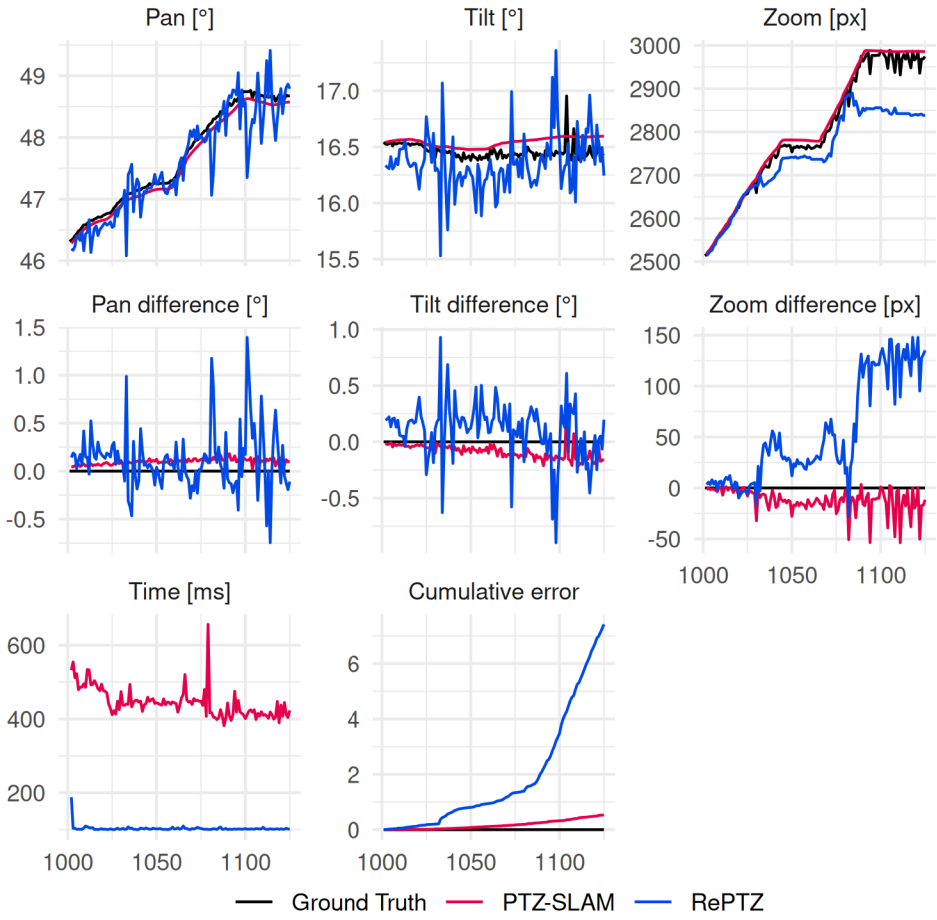


Figure B.9: Statistics for dataset 1 batch 8.

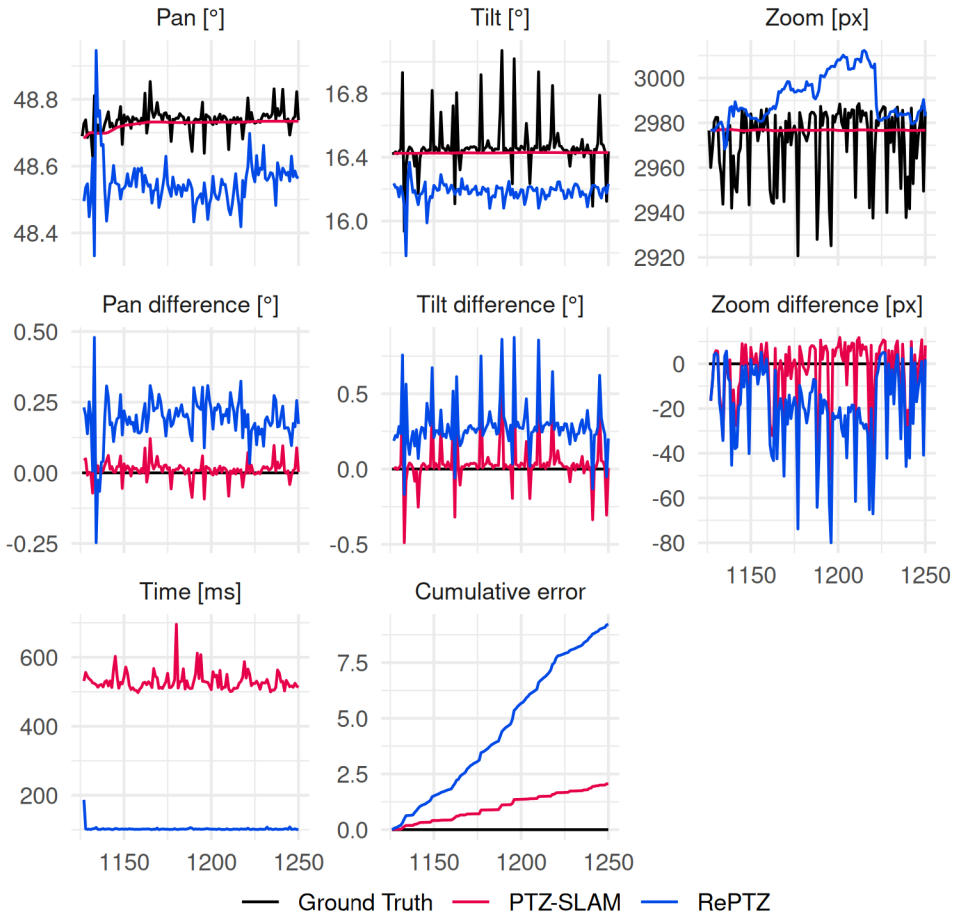


Figure B.10: Statistics for dataset 1 batch 9.

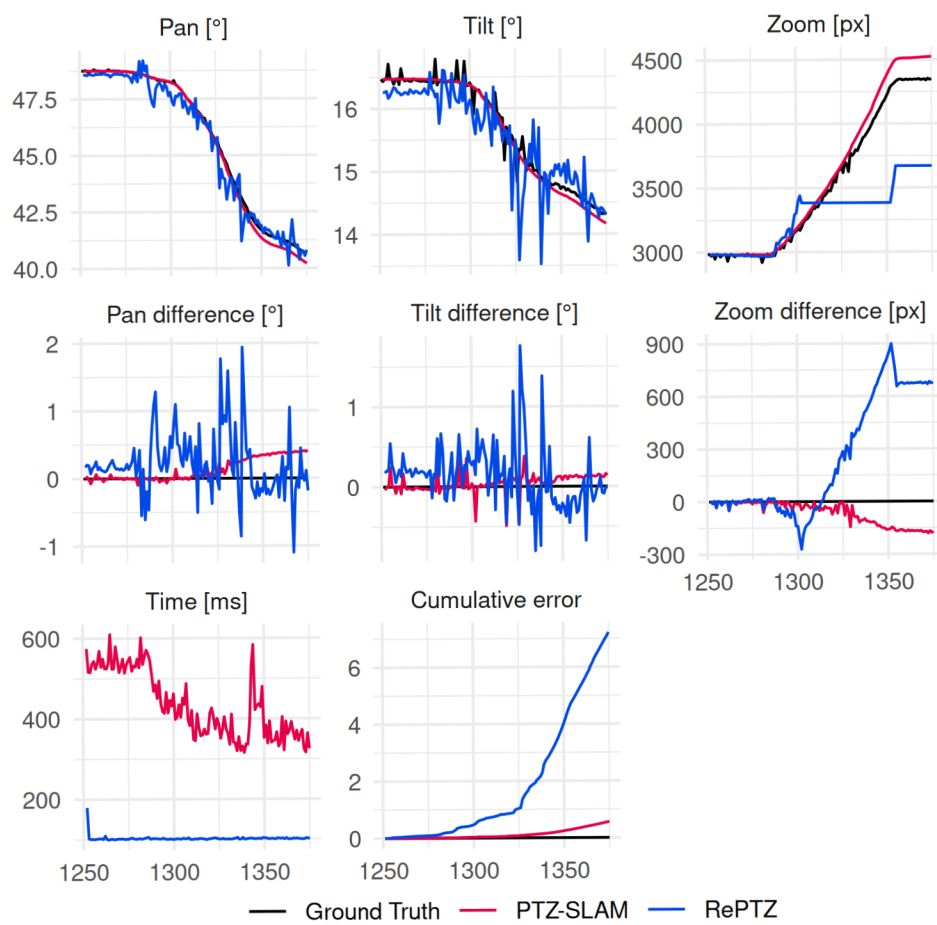


Figure B.11: Statistics for dataset 1 batch 10.

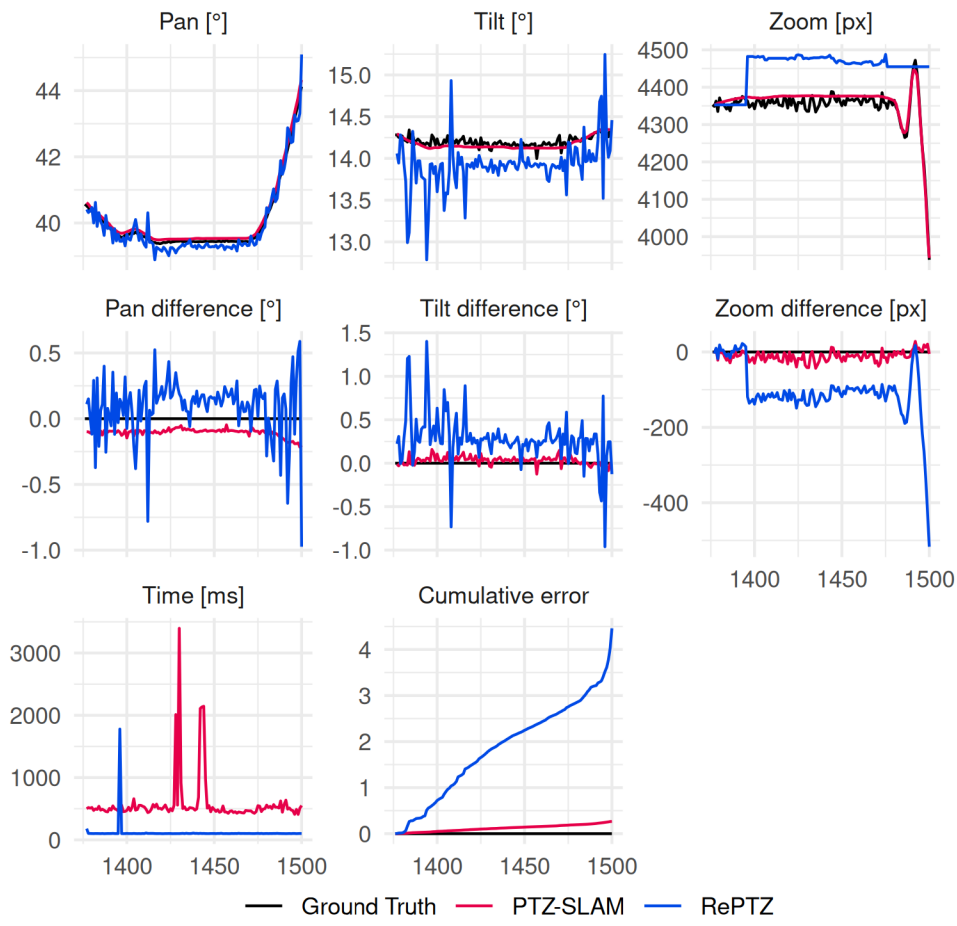


Figure B.12: Statistics for dataset 1 batch 11.

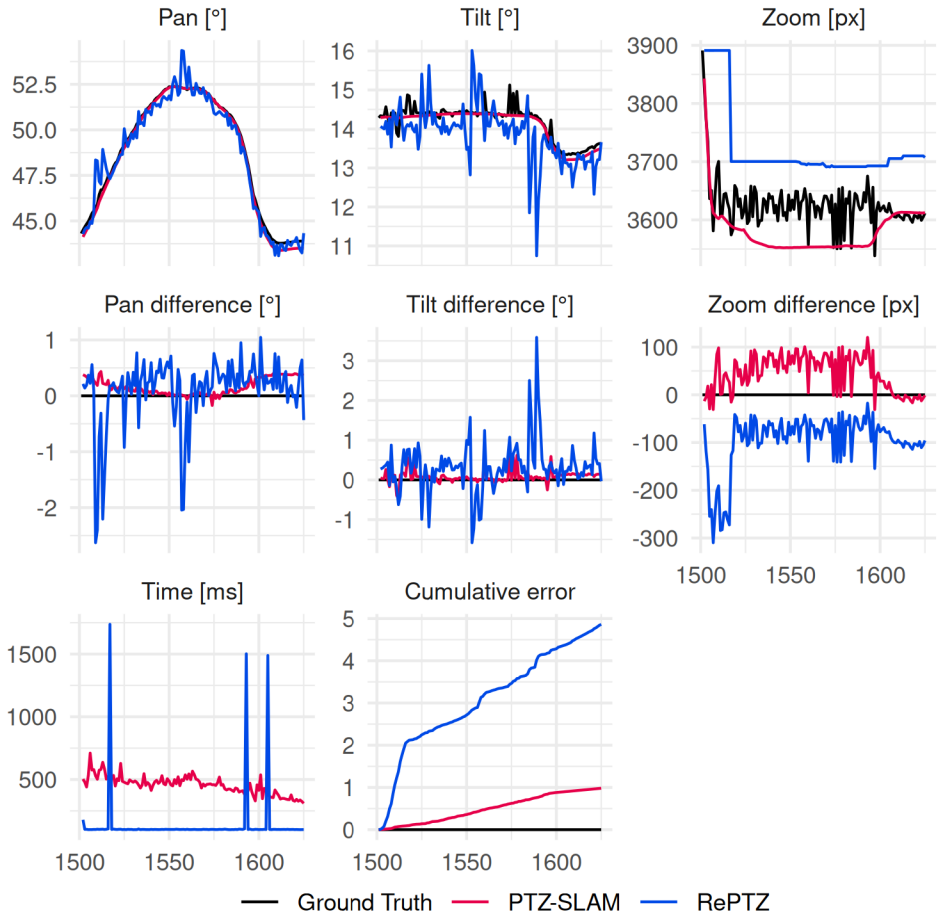


Figure B.13: Statistics for dataset 1 batch 12.

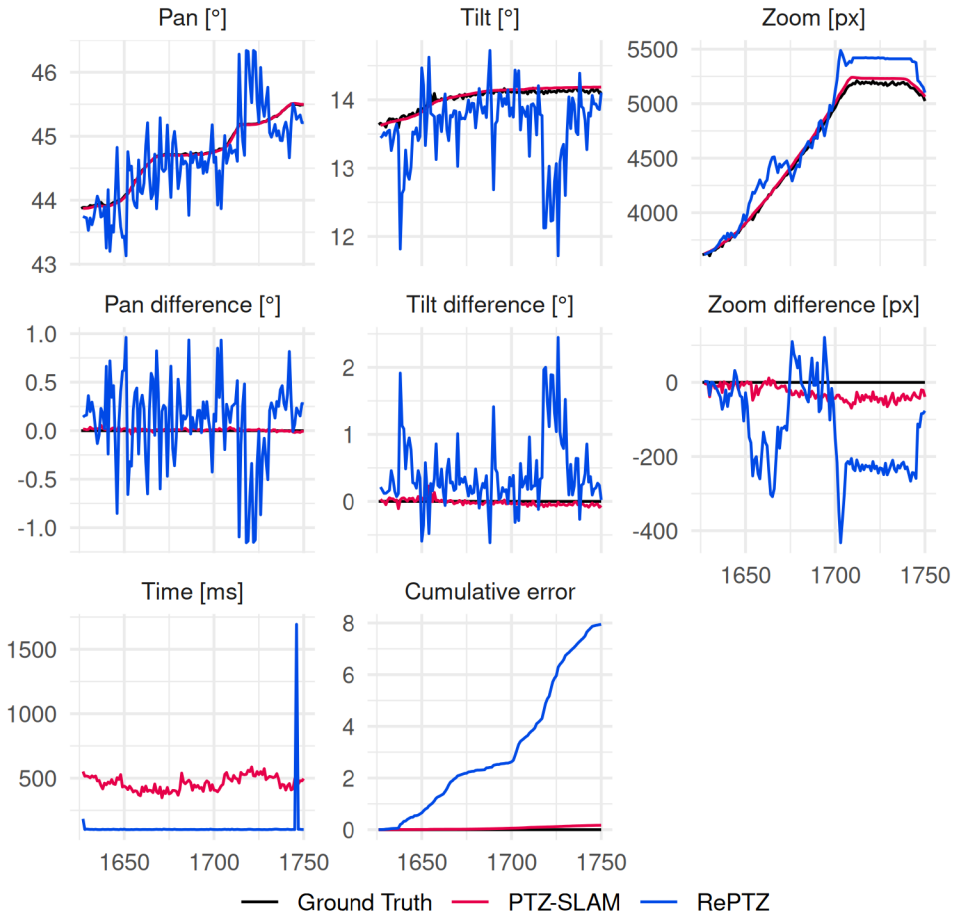


Figure B.14: Statistics for dataset 1 batch 13.

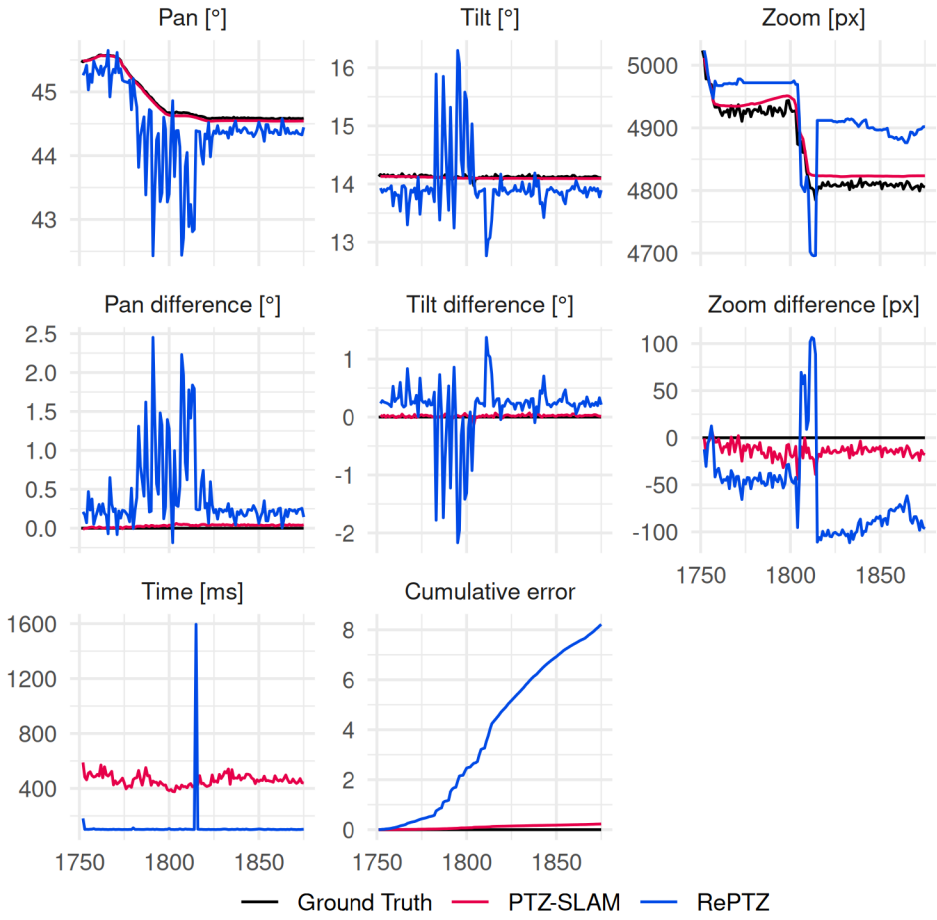


Figure B.15: Statistics for dataset 1 batch 14.

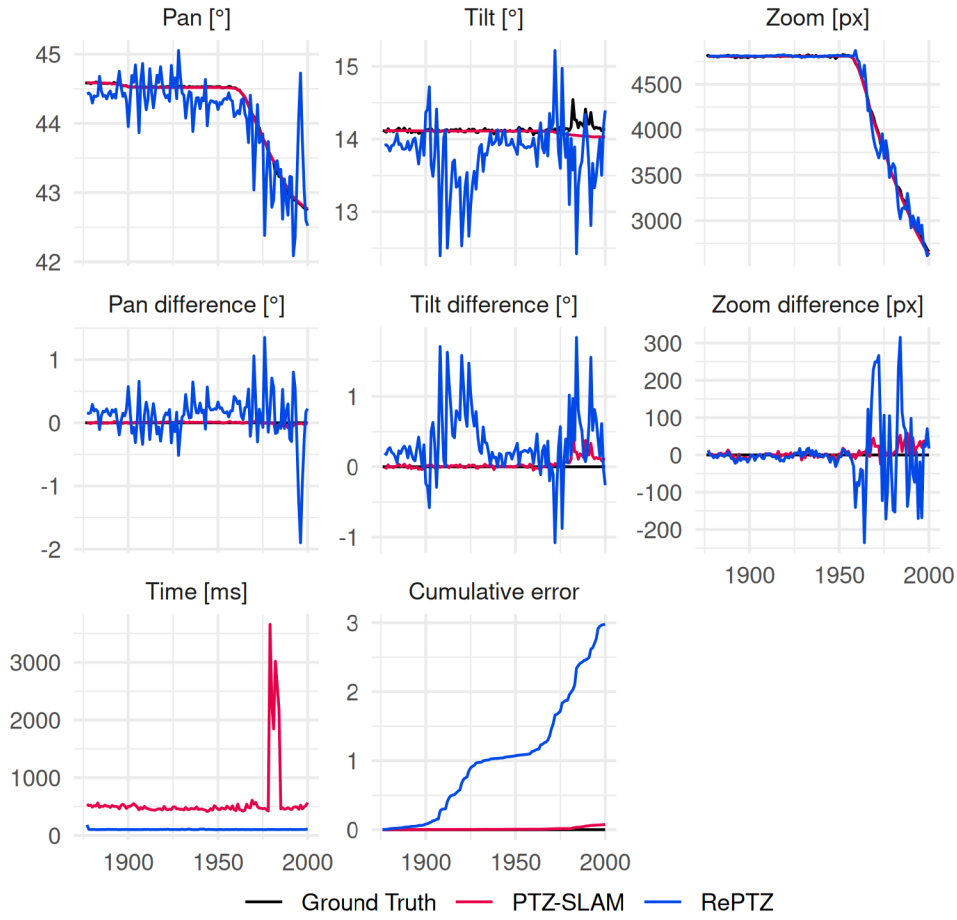


Figure B.16: Statistics for dataset 1 batch 15.

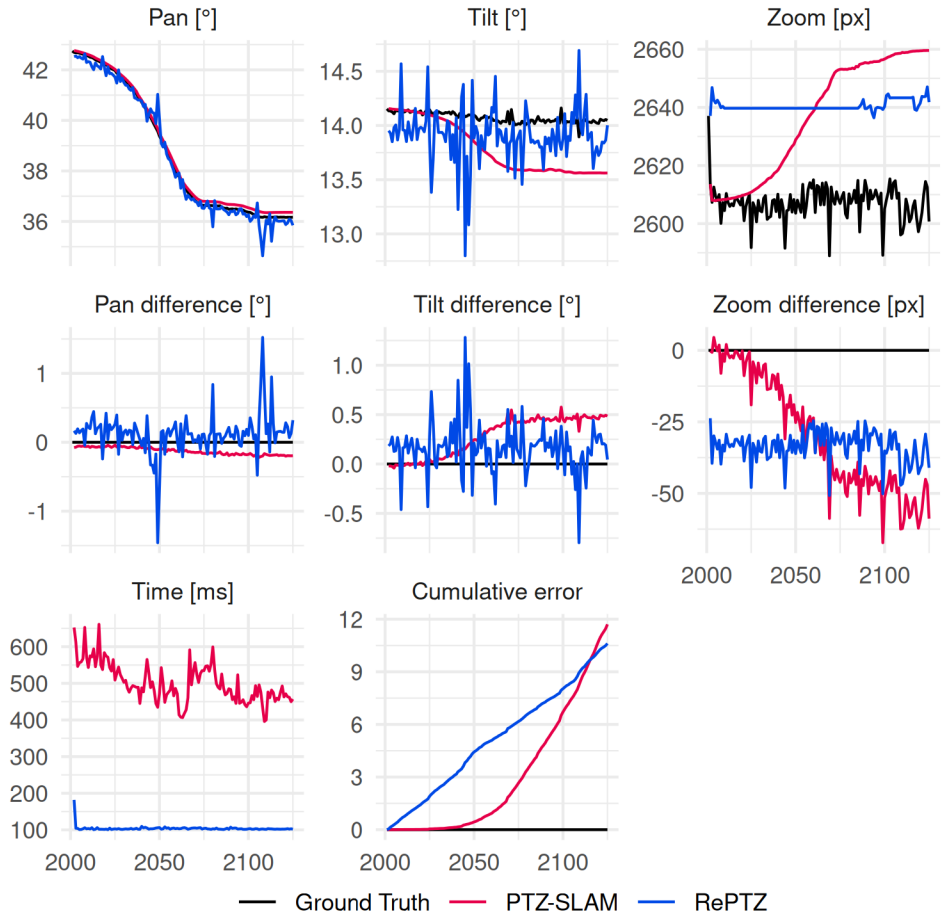


Figure B.17: Statistics for dataset 1 batch 16.

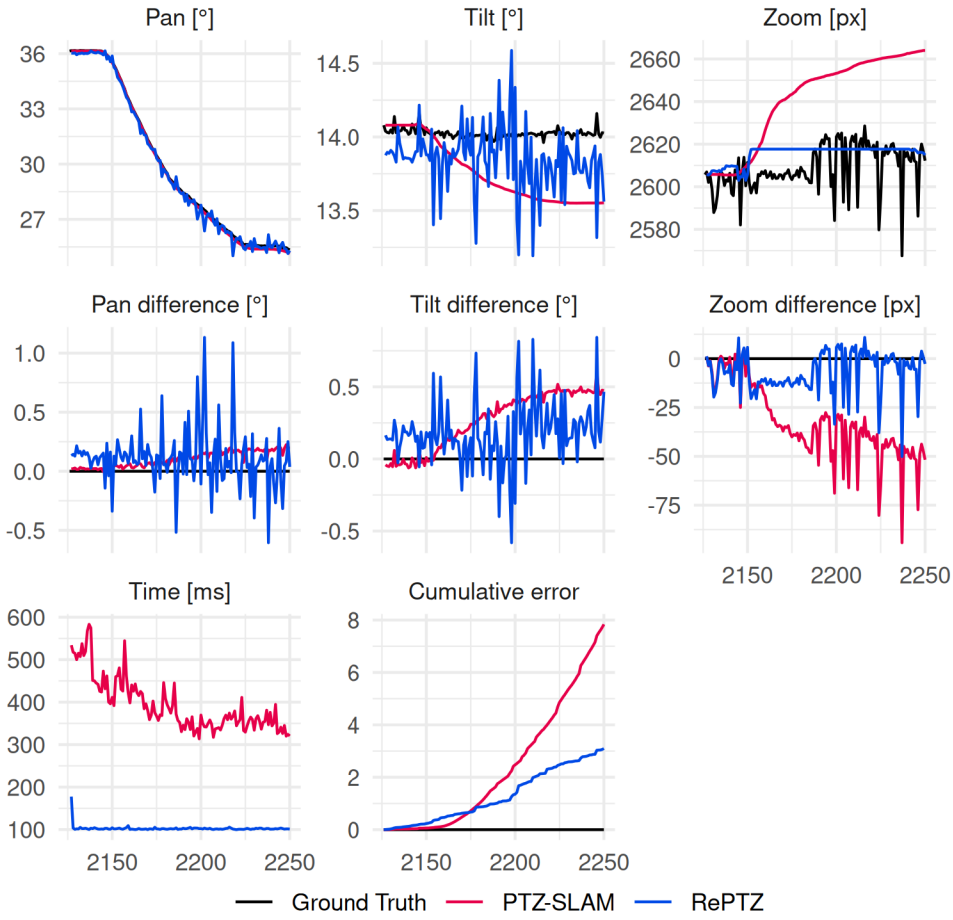


Figure B.18: Statistics for dataset 1 batch 17.

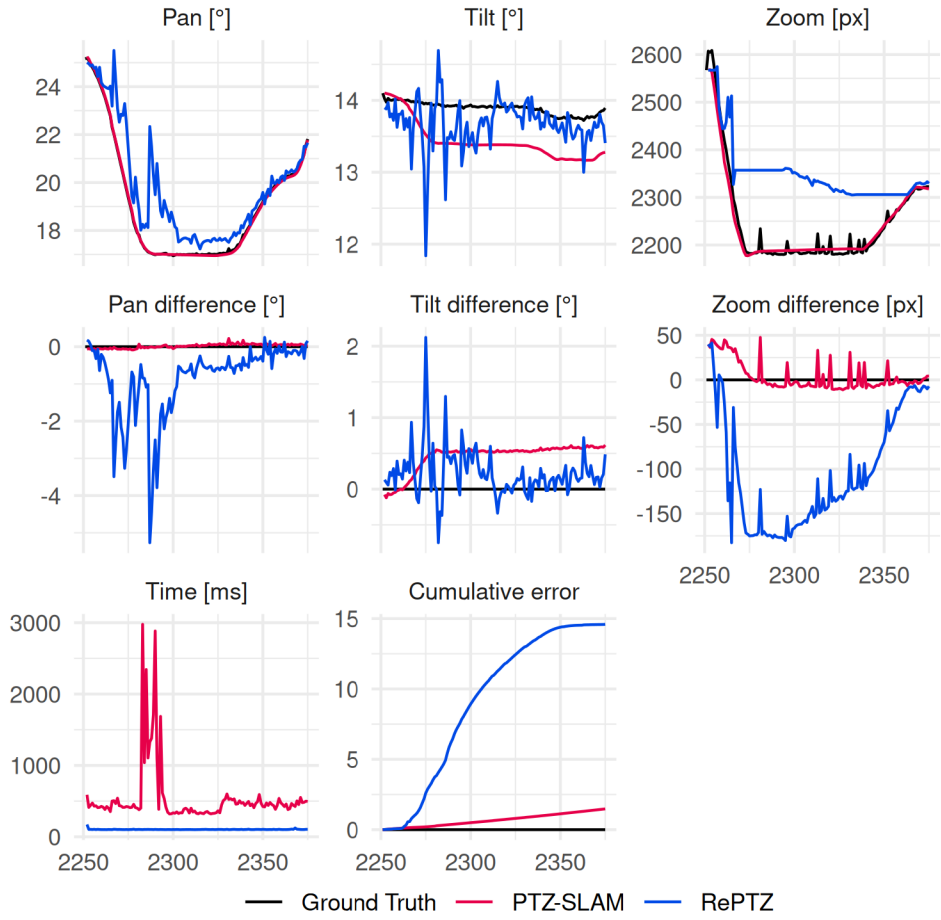


Figure B.19: Statistics for dataset 1 batch 18.

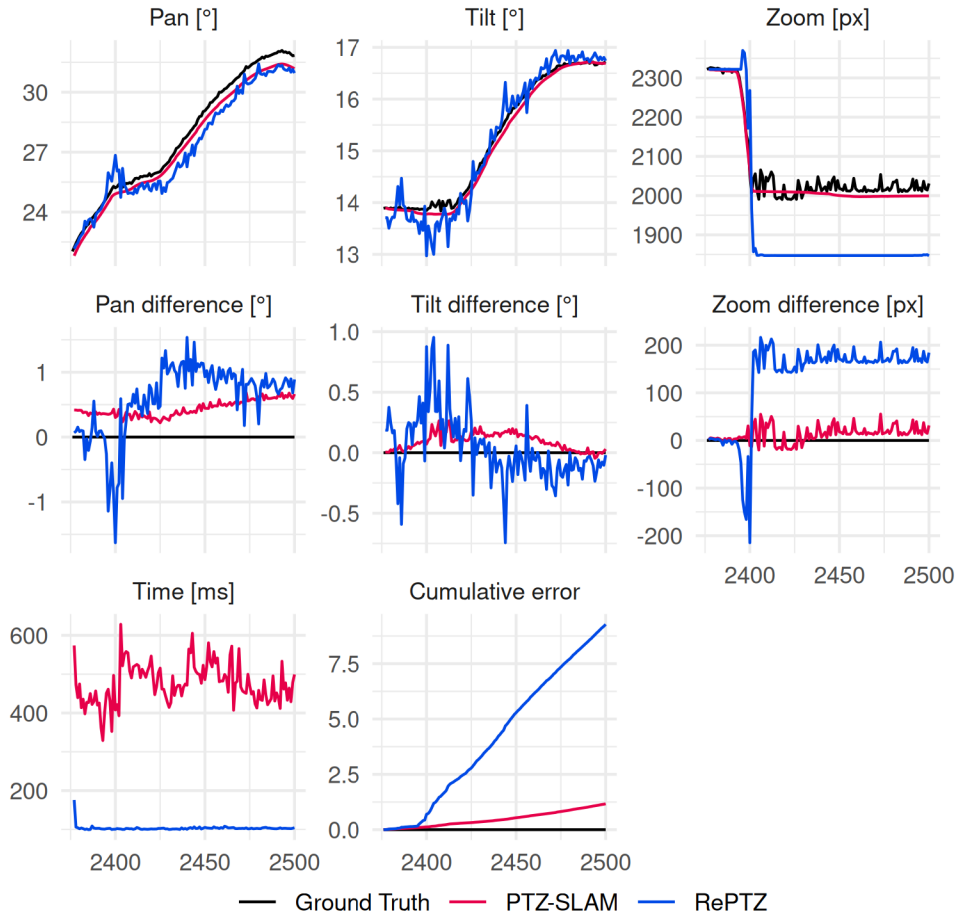


Figure B.20: Statistics for dataset 1 batch 19.

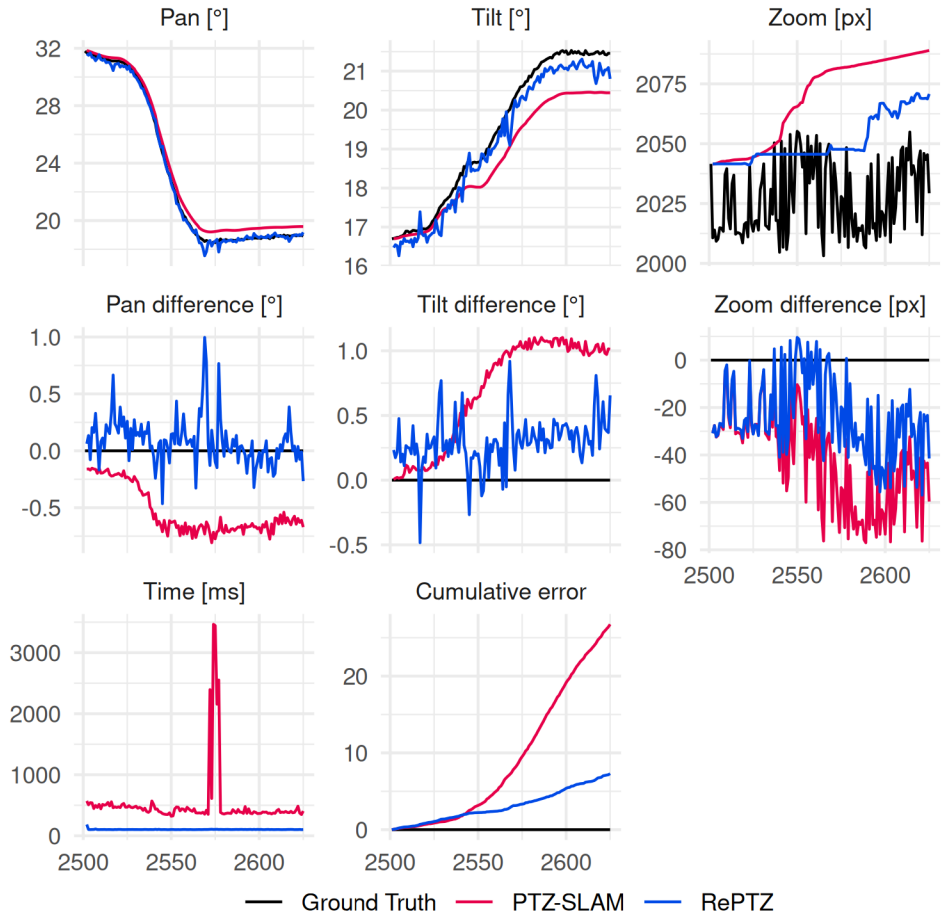


Figure B.21: Statistics for dataset 1 batch 20.

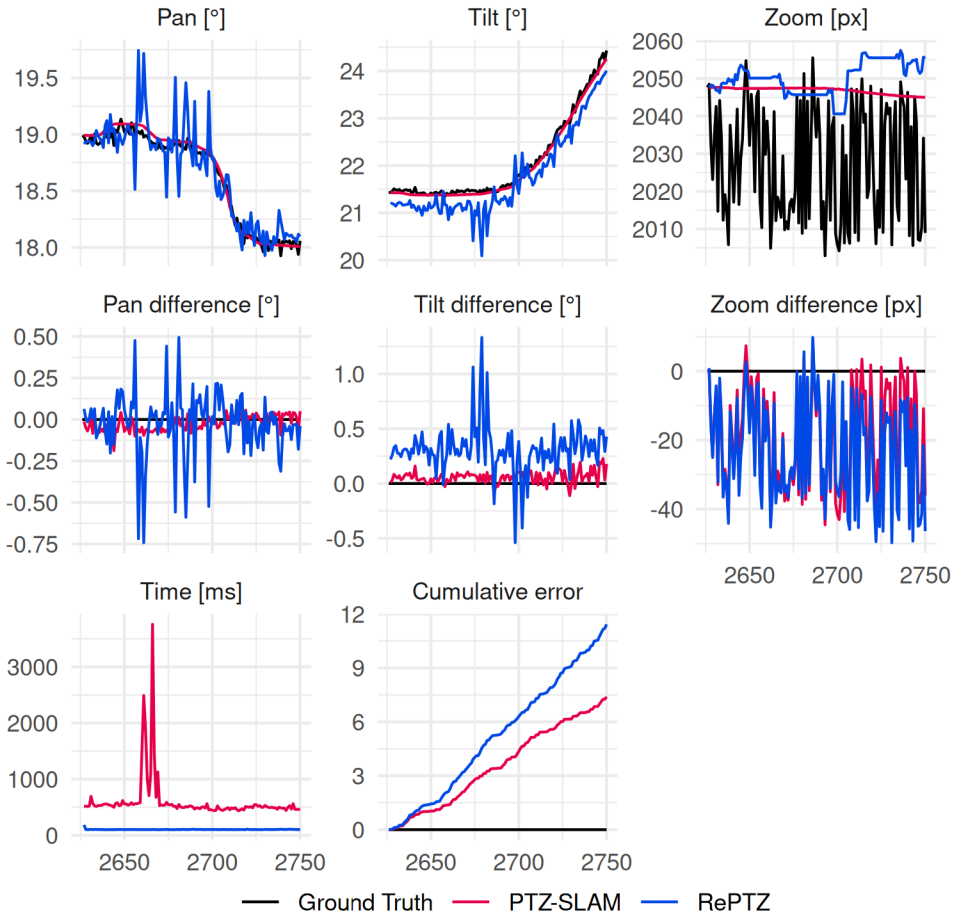


Figure B.22: Statistics for dataset 1 batch 21.

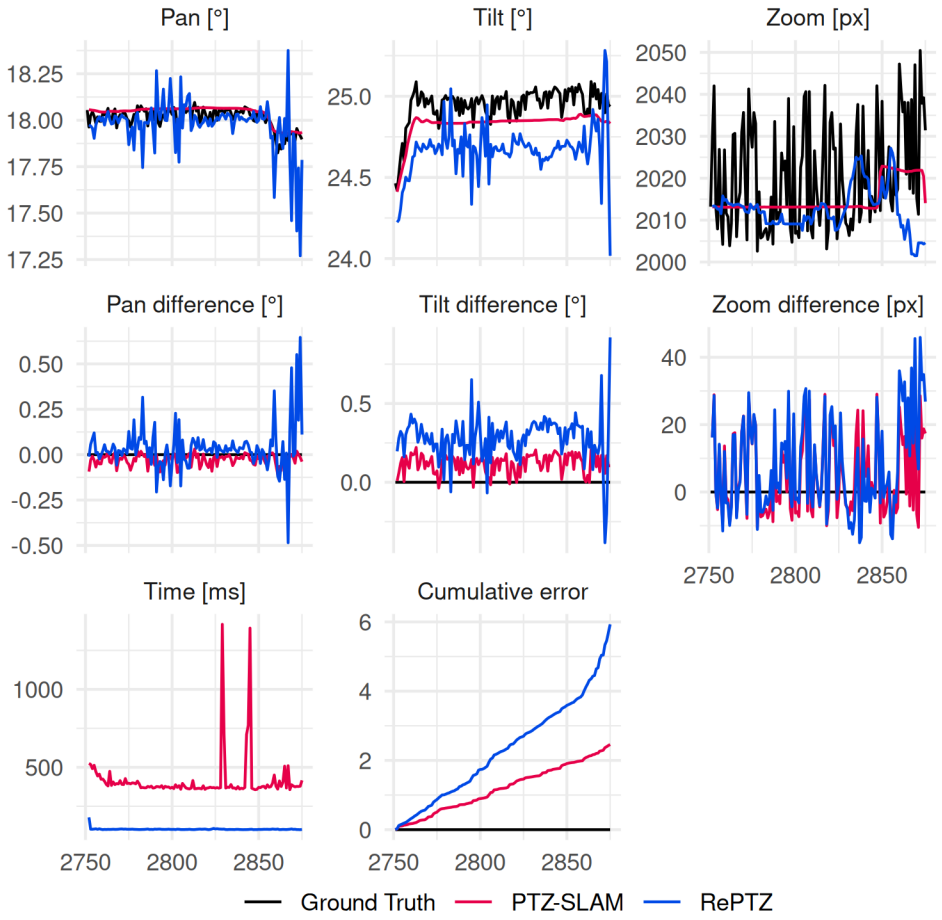


Figure B.23: Statistics for dataset 1 batch 22.

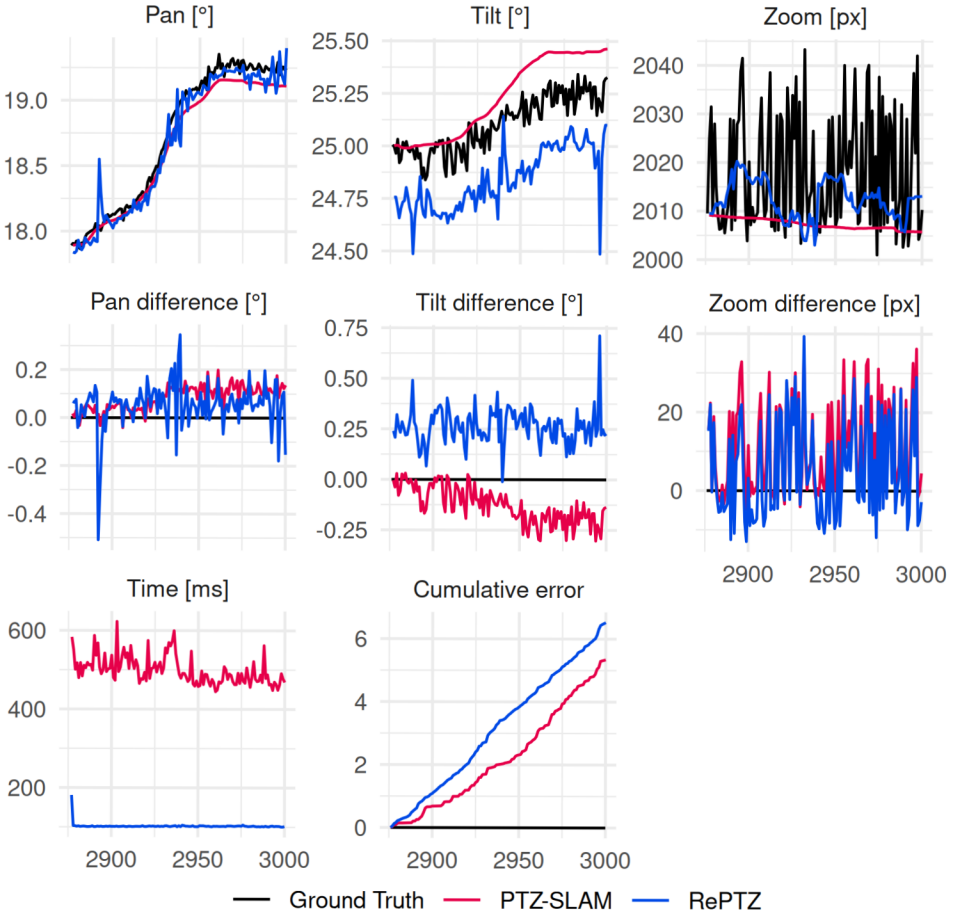


Figure B.24: Statistics for dataset 1 batch 23.

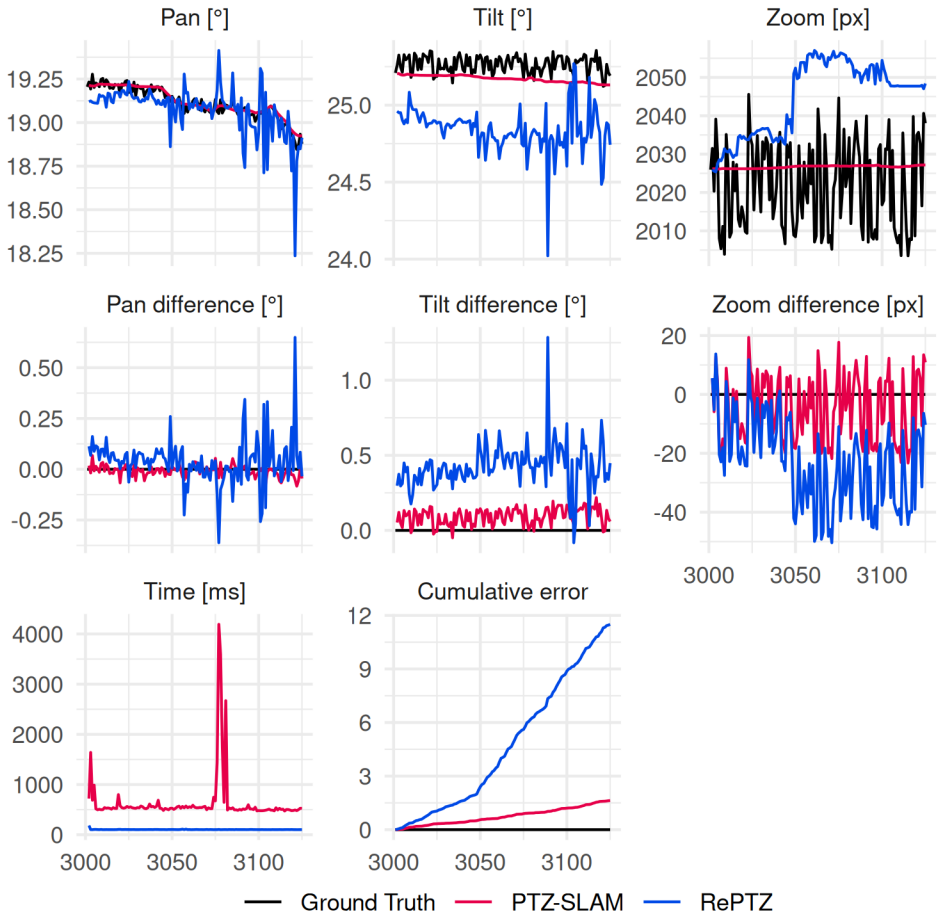


Figure B.25: Statistics for dataset 1 batch 24.

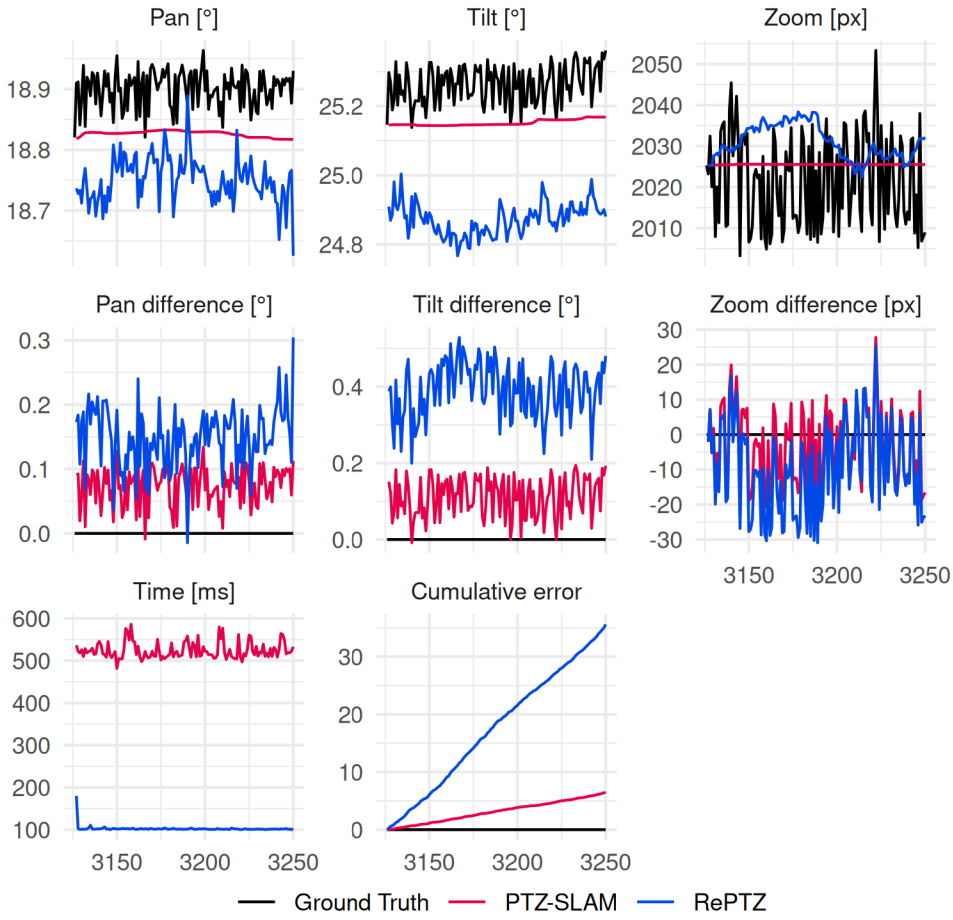


Figure B.26: Statistics for dataset 1 batch 25.

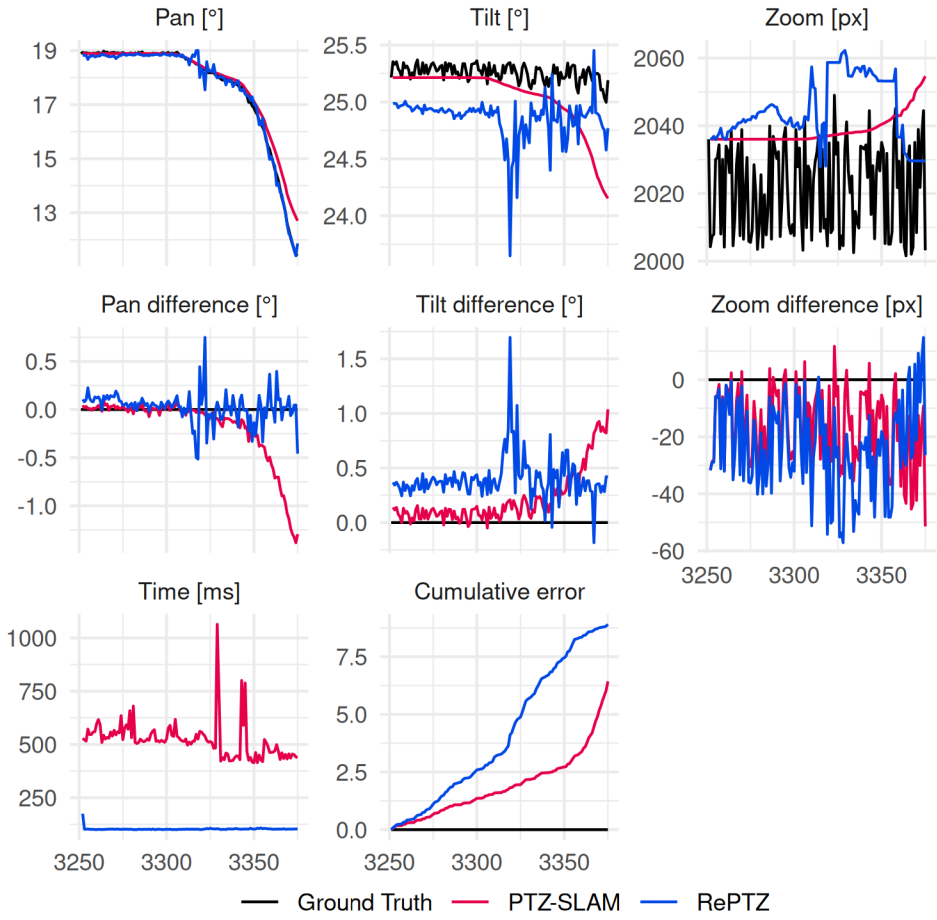


Figure B.27: Statistics for dataset 1 batch 26.

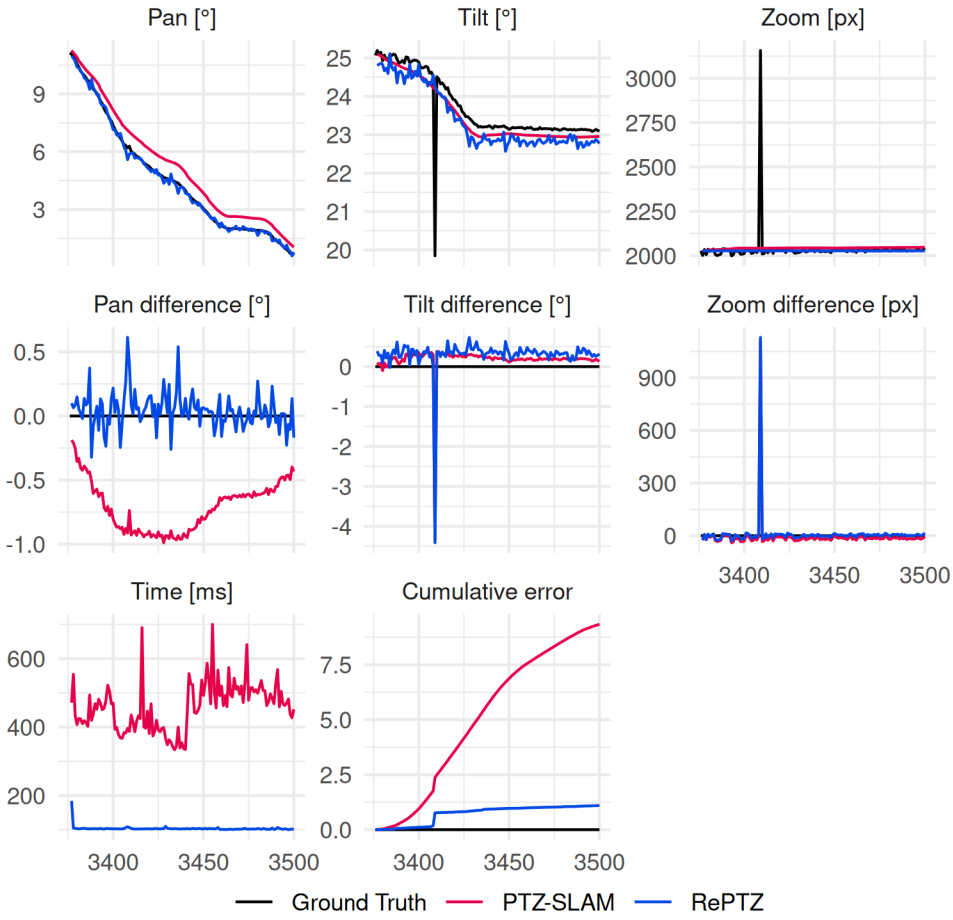


Figure B.28: Statistics for dataset 1 batch 27.

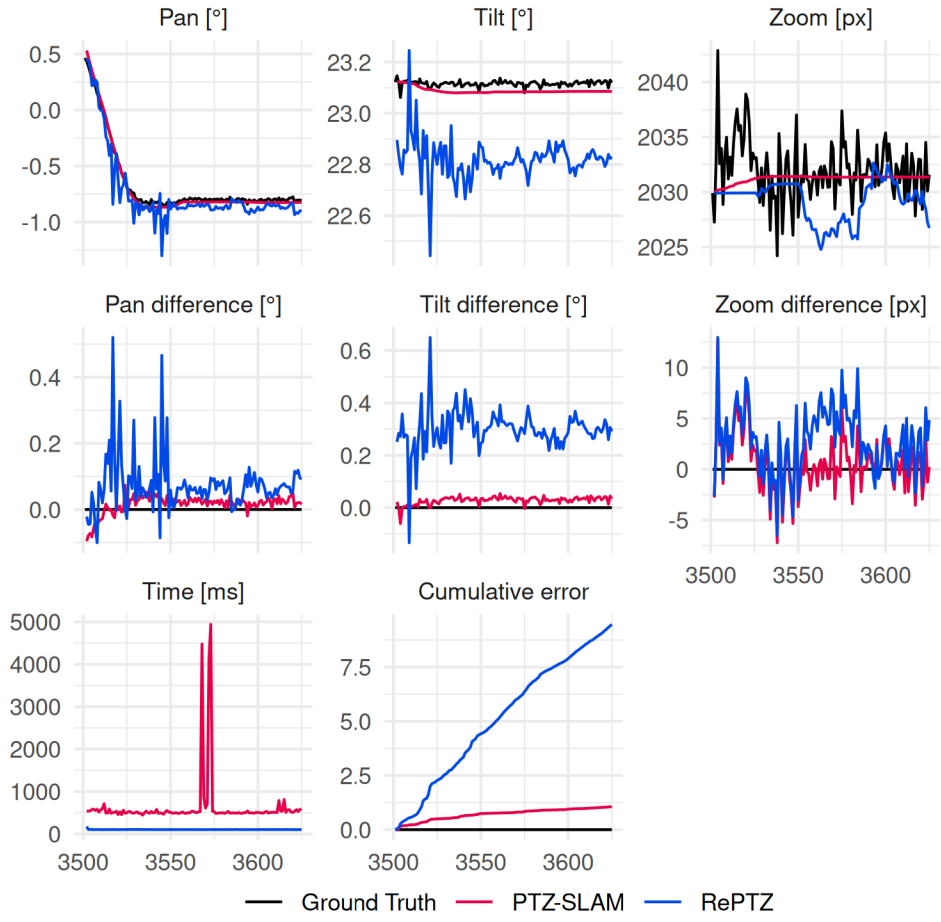


Figure B.29: Statistics for dataset 1 batch 28.

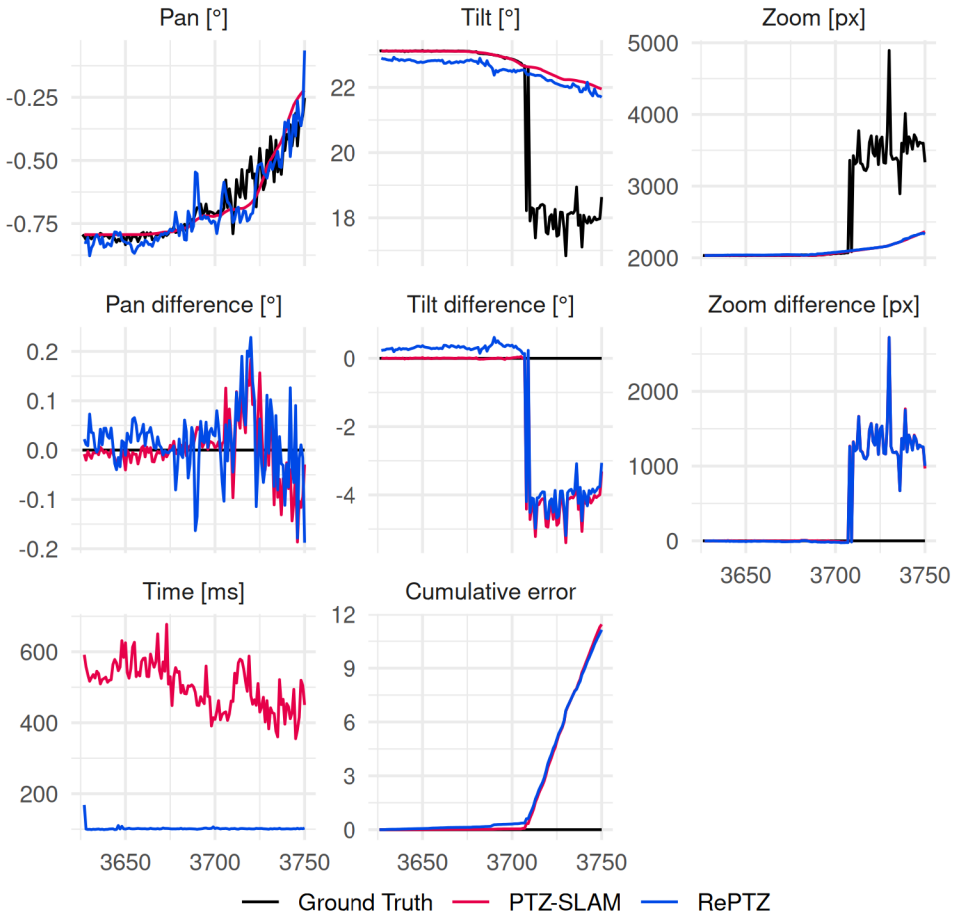


Figure B.30: Statistics for dataset 1 batch 29.

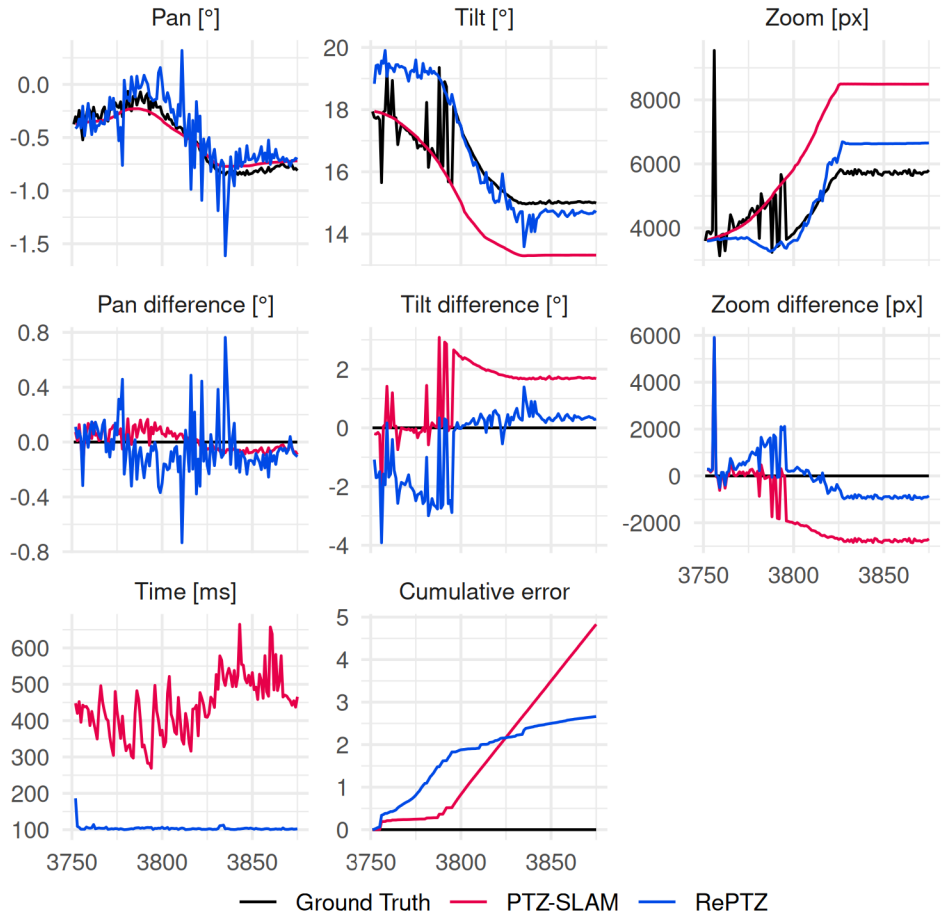


Figure B.31: Statistics for dataset 1 batch 30.

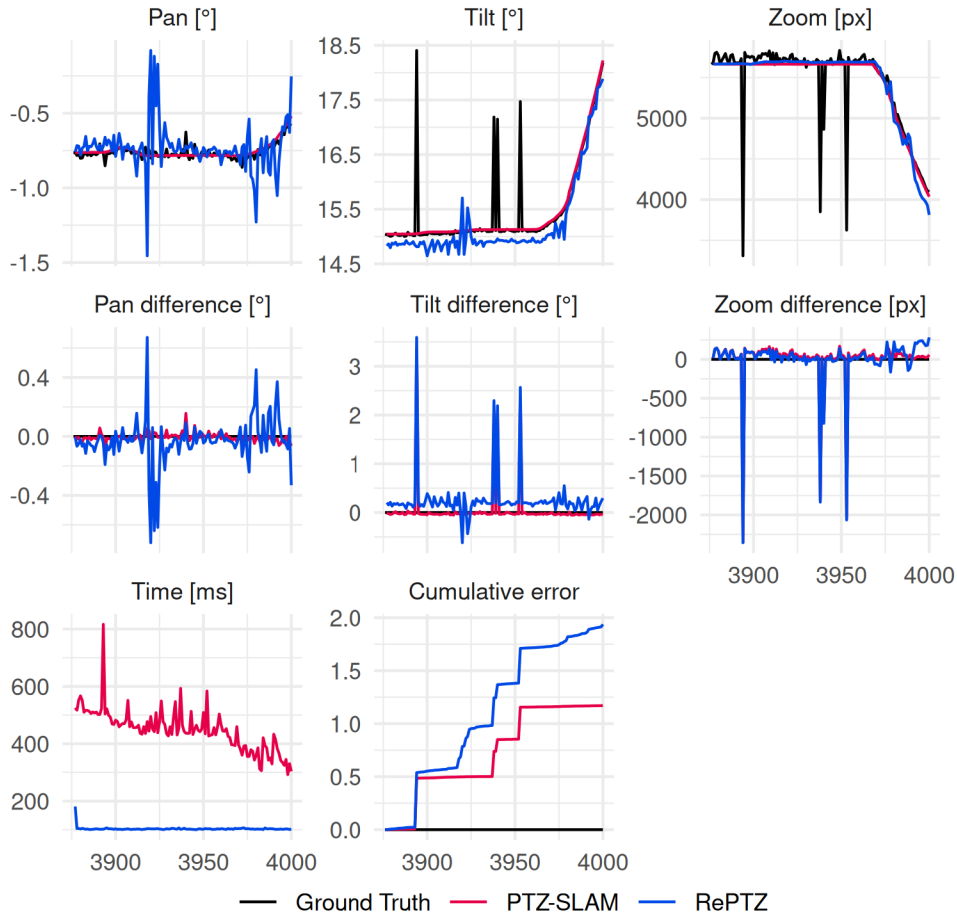


Figure B.32: Statistics for dataset 1 batch 31.

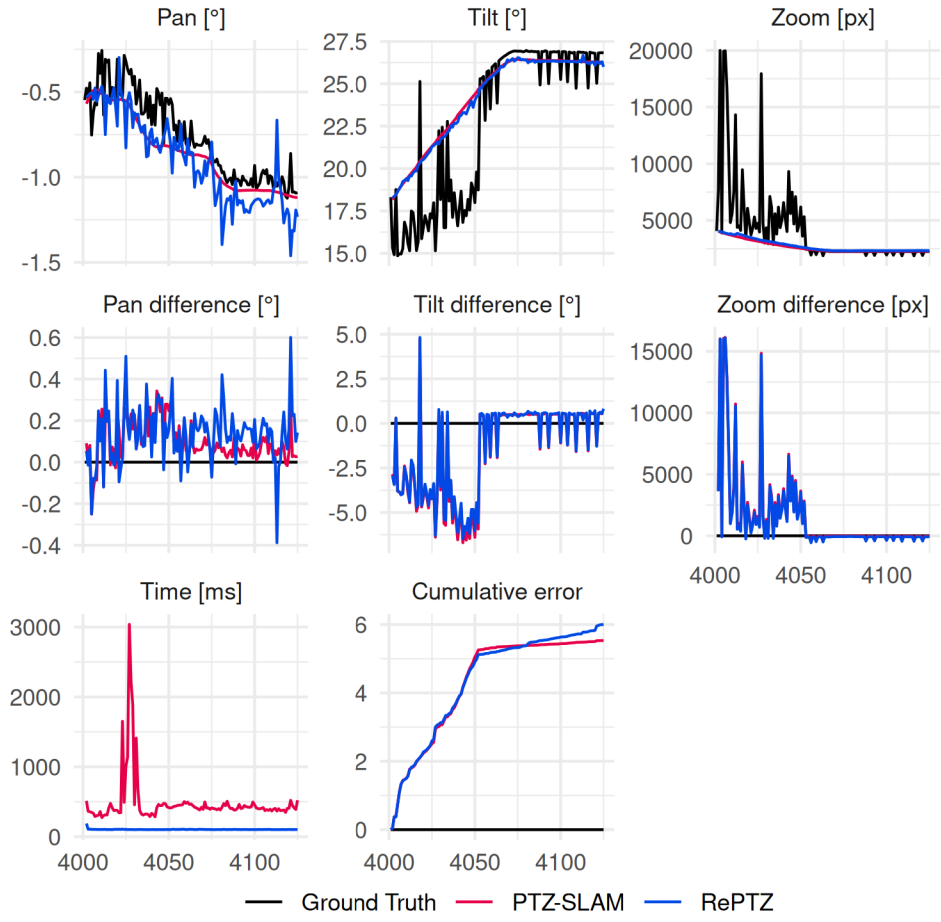


Figure B.33: Statistics for dataset 1 batch 32.

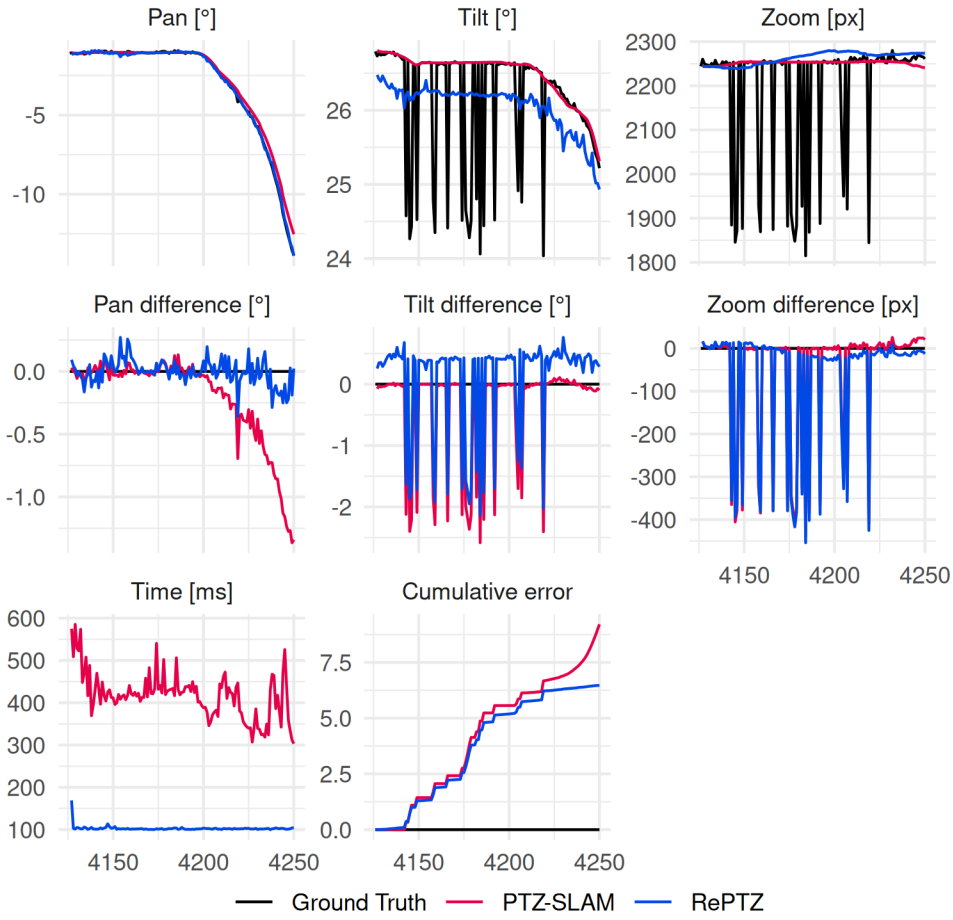


Figure B.34: Statistics for dataset 1 batch 33.

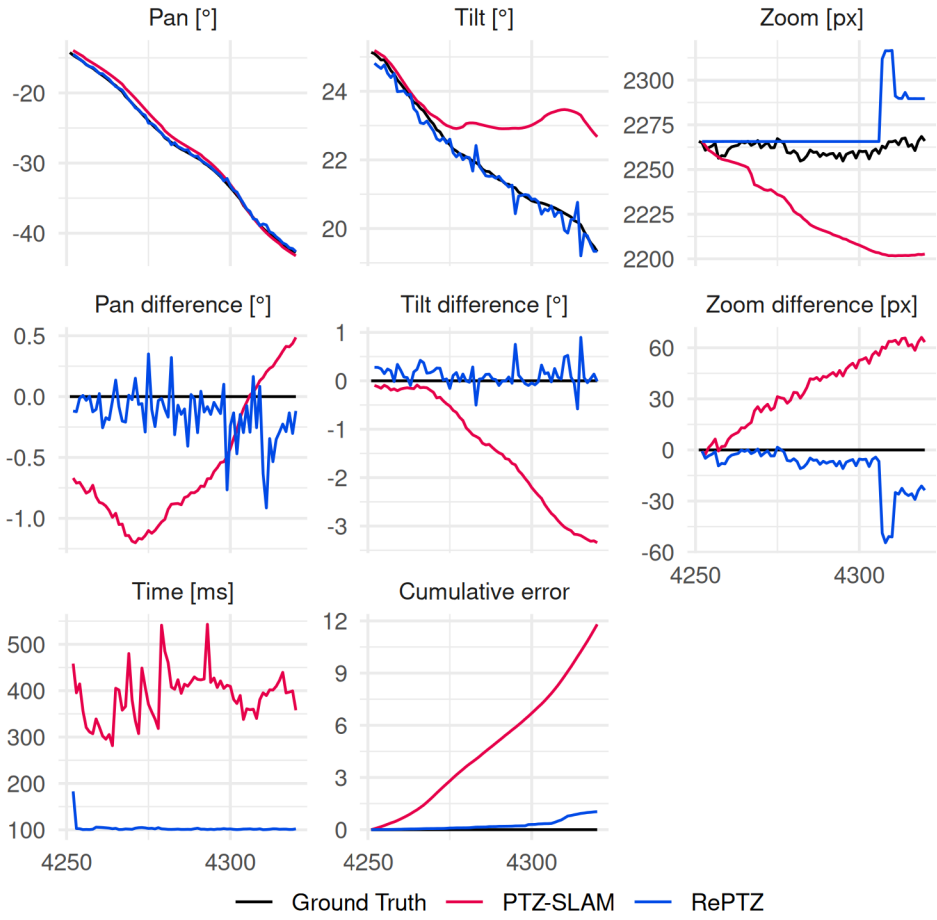


Figure B.35: Statistics for dataset 1 batch 34.