Evaluating Set-based Occlusion-Aware Planners in Traffic Scenarios with Perception Uncertainties

Christiaan Theunisse
Department of Cognitive Robotics
Delft University of Technology
Delft, The Netherlands
C.Theunisse@student.tudelft.nl

***Abstract*—To ensure safe operation of autonomous vehicles (AVs), trajectory planners should account for occlusions. These are areas invisible to the AV that might contain vehicles. Set-based methods can guarantee safety by calculating the reachable set, which is the set of possible states, for each potentially hidden vehicle. A recently published method proved in simulation experiments to reduce the cautiousness by reasoning about these occluded areas over time, assuming perfect input data [1]. We present a novel algorithm that uses this reasoning and is applicable on a real AV with its accompanying uncertainties and imperfect sensor data. The uncertainties include sensor errors and noise, computation and communication delays and control errors in the trajectory following. This is achieved by modelling the error distributions and accounting for them in the calculations, where the confidence interval for each error is exposed as a setting. Experiments indicate that our algorithm can reduce the traversal time through an intersection by 2.2 seconds with reasoning. An ablation study of the different error measures shows that the errors in the construction of the field of view (FOV) limit the performance the most. Reducing the errors in the FOV construction is therefore the most important recommendation, besides making the method interaction-aware.**

I. Introduction

Autonomous driving and advanced driver assistance systems (ADAS) are supposed to have numerous benefits, given the huge and ever-increasing investments made [2]. One of these benefits is an increase in traffic safety, given that over 90% of the accidents are human mistakes [3].

However, before the benefits can be unlocked, the failure rate of autonomous vehicles (AVs) needs to be reduced. A common problem are occlusions (Fig. 1), which are unobservable areas within the environment of the AV that possibly contain other traffic participants that are relevant to the trajectory planning of the AV.

There are multiple approaches proposed in the literature to planning trajectories that account for possible vehicles in these occlusions. However, only set-based approaches can guarantee to be legally safe, meaning that the AV will not be involved in a collision it is responsible for [4], [5].

A recently published set-based method keeps track of the parts of the environment that are occluded [1]. Over time, it can guarantee that certain occluded parts of the environment are free of vehicles, reducing the area that possibly contains vehicles and thus the cautiousness. However, this approach has only been used and tested in simulation, where it provides safe trajectories by using perfect input data. In the real world, input data is always imperfect because of errors and noise in the measurements from sensors like the Lidar. Besides, the real world adds delays due to computations and communication and a trajectory cannot be perfectly followed, rendering the algorithm in [1] unsafe on a real mobile robot.

Our goal is to map the uncertainties inherent to the real world and generate models describing the noise and error in the sensors. This will be used to develop an algorithm that provides safe trajectories for real AVs, while using this reasoning over time about occlusions from [1].

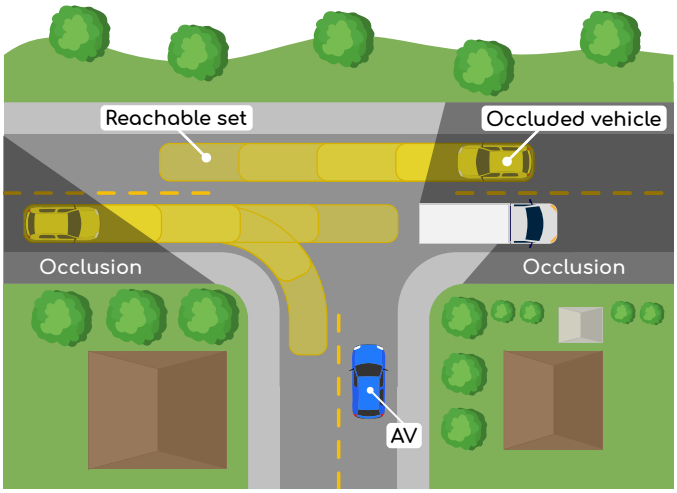


Fig. 1: A scenario with occlusions, occluded vehicles and the reachable set calculated for these vehicles.

A. Related work

When it comes to trajectory planners that account for occlusions, there are several approaches, one of which is learning based. Reference [6] trains a controller with reinforcement learning that decides to accelerate or decelerate in every time step. However, the tests in simulation indicated that the AV was still involved in many collisions. This approach was later extended with a risk-aware reward function [7], which

Thanks to ALTEN for giving me the opportunity to do my graduation project at their company and providing me with the robots and space to perform the real-world experiments.

significantly reduced the number of collisions. Another option is to use expert human driver data to make a controller because it contains plenty of examples of correct driving behavior. One approach learned a cost function for an MPC with inverse reinforcement learning [8], but the result was only demonstrated in a few handcrafted scenarios. The method in [9] determines the relation between a set of driving behaviors and a map. However, this method is only applicable to the map seen during training and simulation experiments reveal that it struggles with trajectories that are not in the training set. Besides, none of these methods have yet been implemented on a real mobile robot or AV.

Another approach is to model the planning problem, including occlusions, as a partial observable Markov decision process (POMDP) and use a solver to find trajectories. The probabilistic nature of a POMDP allows assigning a probability to each occlusion for the presence of an occluded vehicle. This results in comfortable and human-like driving behavior [10]. Besides, the generic approach of the solver allows changing the model and cost function to make it, for instance, incentivize a visibility increase [11]. The computational load of the solver is a significant disadvantage which hinders real-world implementations, but improvements are made [12]. Nevertheless, there are real-world implementations that work on simple scenarios [13]. But all the same, a trajectory planner based on a POMDP can never guarantee safety due to the inherently probabilistic approach.

Set-based methods, on the other hand, calculate the reachable set. The reachable set of a vehicle includes all the states it can reach at a certain time in the future, when it adheres to certain legal and physical constraints (Fig. 1). By calculating the reachable set for any (possibly occluded) traffic participant and planning a trajectory that avoids these sets, the AV can guarantee to be safe [4]. This approach was extended to occluded pedestrians and cyclists and tested on a real AV [5].

Later, a method was developed that reasons about the set of possible positions and velocities for vehicles in occlusions [14]. This is achieved by considering only the position of vehicles along the centerline of the road. As a result, the full width of the road is considered to be visible when in fact only the centerline is. This assumption might cause problems for vehicles not on the centerline of the road or smaller ones, like motorcycles, and negates the safety guarantee.

Another option is to only reason about the set of possible positions, but to do this for any 2D position on the road [1]. Although this results in more cautious behavior because the range of possible velocities is not reduced, it prevents from making any assumption about the size of other vehicles. Since our goal is to provide safe trajectories, the reasoning from [1] will be used for our algorithm.

B. Contributions

Our contribution consists of making a set-based algorithm using the reasoning from [1] that is applicable on a real AV and accounts for the errors and uncertainties introduced by the real-world. More specifically, our contributions are:

1) Modelling the expected error and uncertainty of the Lidar, state estimation and trajectory following.
2) Using these models to make the algorithm account for errors and uncertainties in the measurements and delays in the computation and communication.
3) Performing experiments with a mobile robot to assess the performance of the developed algorithm and an ablation study to distinguish the contribution to the cautiousness of the different errors.

Developing an algorithm that properly accounts for the real-world uncertainties and assessing its performance, contributes to the goal of delivering a swift trajectory planner that provides a legal safety guarantee and is deployable on a real AV.

II. Problem Statement

The algorithm presented in [1], further referred to as `foresee`, assumes perfect input data. The goal of this paper is to develop an algorithm that allows for imperfect input data and provides the same reasoning as `foresee`. To ensure that a trajectory is safe, and the reachable set really accounts for every possible state, set-based methods overapproximate the reachable sets in case there is ambiguity. This overapproximating characteristic should be maintained in the real world despite uncertainties and delays. To achieve this, the different sources of errors and delays are mapped out in section IV. Section V discusses the modelling of these errors and section VI describes the new algorithm that relies on these models to account for these errors. The newly developed algorithm is called `foresee++`.

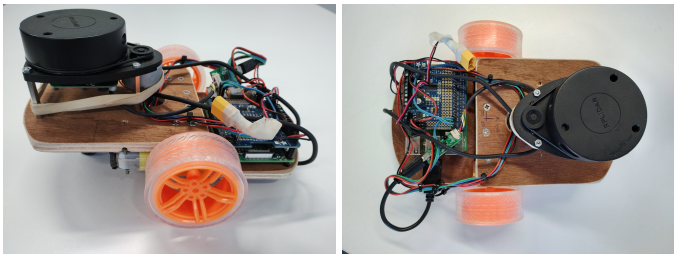
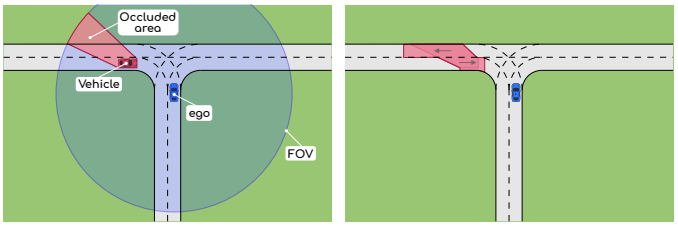


Fig. 2: Mobile robot used for the experiments

Section VII covers the experiments performed with a differential-driven mobile robot (see Fig. 2) running the developed algorithm `foresee++`. This robot is the perfect test bed for the new algorithm, since its sensors provide input data that is at least as imperfect as seen in real AVs. It sensor suite consists of a 2D Lidar, IMU and wheel encoders.

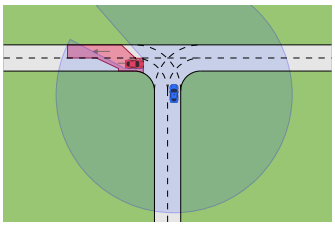
III. How to make Foresee work with sensor data

Before diving into the specifics, a general description is given of `foresee`. The field of view (FOV) is a polygon describing the area visible to the ego vehicle, further referred to as ego. In the initial update step, the occluded part of the road is the area projection of the set of states for possible hidden obstacles $\mathcal{P}^0$ (see Fig. 3a). In the next update step, the reachable set $\mathcal{R}(\mathcal{P}^0)$ of the set $\mathcal{P}^0$ is calculated for the time passed between the two update steps (Fig. 3b) and a new

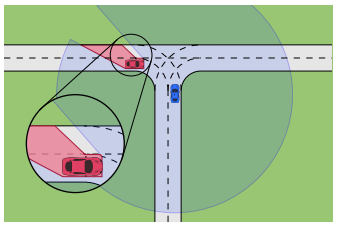


(a) Initial update step: the occluded areas are the area projection of the set of states of possible hidden obstacles $\mathcal{P}^0$.

(b) Next time step: the reachable set of the set of states of possible hidden obstacles $\mathcal{R}(\mathcal{P}^0)$.



(c) The intersection between the projection of the reachable set $\mathcal{R}(\mathcal{P}^0)$ and the FOV is removed from $\mathcal{R}(\mathcal{P}^0)$.



(d) Updated set of states of possible hidden obstacles $\mathcal{P}^1$.

Fig. 3: Visualization of the reasoning performed in `foresee`

FOV is obtained. The part of the projection of the reachable set $\mathcal{R}(\mathcal{P}^0)$ that intersects with the new FOV is removed (Fig. 3c), resulting in the updated set of states of possible hidden obstacles $\mathcal{P}^1$ (Fig. 3d). This last figure shows that an occluded part of the environment is guaranteed to not contain vehicles, which is the advantage of reasoning.

The reasoning is also given in Eq. 1, where $\mathcal{X}$ is the set of all possible states and $\text{proj}_{\text{xy}}(x)$ is the area projection of a state x. The last line shows how this reasoning is continued in subsequent steps.

$$\begin{aligned} \mathcal{P}^0 &= \{x \in \mathcal{X} \,|\, \text{proj}_{\text{xy}}(x) \notin \text{FOV}^0\} \\ \mathcal{P}^1 &= \{x \in \mathcal{R}(\mathcal{P}^0) \,|\, \text{proj}_{\text{xy}}(x) \notin \text{FOV}^1\} \\ \mathcal{P}^i &= \{x \in \mathcal{R}(\mathcal{P}^{i-1}) \,|\, \text{proj}_{\text{xy}}(x) \notin \text{FOV}^i\} \end{aligned} \tag{1}$$

`Foresee` does not rely on simulated sensor data, but just receives the FOV and ego state from the simulation. Besides, a trajectory is followed by simply assigning the desired state to the ego. Therefore, the first step is to determine how the FOV and state of the ego can be calculated from the sensor measurements and how the motors can be controlled to follow a trajectory (Fig. 4).

A. FOV

The FOV is constructed from the point cloud that results from the Lidar measurements. This approach is easy to implement because it does not rely on obstacle detection, requires few computations and fits the sensor suite of the robot.

B. State

The odometry is obtained from respectively the wheel encoders and IMU which are combined in an extended Kalman

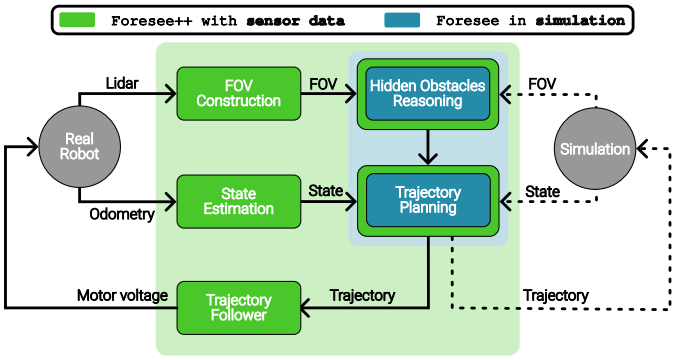


Fig. 4: Flow diagram that describes `foresee` with the simulation input and `foresee++` with sensor data input

filter (EKF), the *velocity EKF*. The odometry is necessary to smoothly follow a trajectory. Lidar-based SLAM is used for the localization and is merged with the odometry in the *position EKF* to obtain the state. The global localization keeps the uncertainty on the pose (position and orientation) estimate bounded, and the EKFs provide a real-time pose estimate based on all sensors with the accompanying uncertainty.

C. Trajectory

To follow a trajectory, the steering angle control law from [15] is used together with two PID controllers for the linear and for the angular velocity. Both the control law and PIDs are simple controllers that give an upper bound on the performance, since AVs normally rely on more advanced controllers.

IV. Sources of Error When Using Imperfect Sensor Data

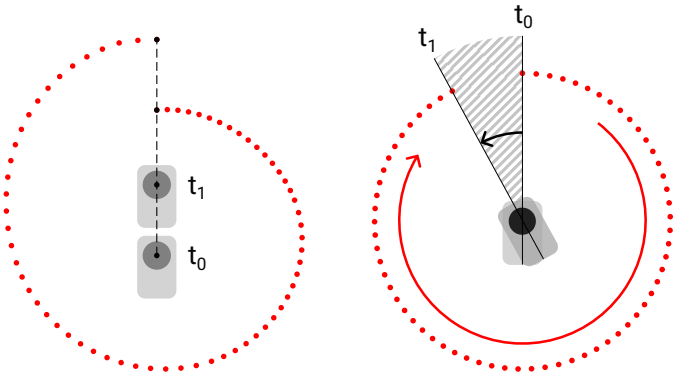
When imperfect sensor data is used to calculate the FOV and ego state, errors are introduced in the calculations. Besides, the actual state will unavoidably deviate from the trajectory. This section discusses the sensor and control errors and the delays that should be accounted for in the trajectory planning.

A. FOV

The FOV is constructed from Lidar measurements, which consist of a list of angles and the ranges measured at these angles. This can be converted to a point cloud.

The different sources of error are:

1) the error in the angles and ranges from the Lidar measurement.
2) the uncertainty in the pose of the ego relative to the road, and thus the pose of the point cloud.
3) the change of the ego pose during a scan (Fig. 5a).
4) a rotation of the ego opposite to the Lidar, resulting in a scan with less than 360° of the environment (Fig. 5b).
5) the significant delay between the scan and actual use of the FOV, affecting the validity of the FOV.



(a) Movement of the ego during the Lidar scan from t_0 until t_1 influences the resulting point cloud.

(b) Movement of the ego might cause that a part of the environment is unobserved by the scan.

Fig. 5: Two error sources in the Lidar explained

B. State

The state is estimated from the odometry and global localization. Errors are introduced by:

1) uncertainty in the IMU velocity measurements.
2) uncertainty in the wheel encoder velocity measurements.
3) uncertainty in SLAM pose estimate.

C. Trajectory

The occupancy set of a trajectory is calculated by placing the ego shape at the planned pose for each time step. This is the area the ego is expected to occupy and is similar to the area projection of a reachable set. Errors between the actual and planned pose are introduced because:

1) the ego cannot follow perfectly the planned trajectory.
2) the estimated state, which is used as a feedback signal, is imprecise itself.
3) a trajectory planned from the state at the beginning of an update step will already have a significant tracking error when it is returned at the end of the update step.

V. Modelling and Mapping the Errors

To account for the errors identified in the previous section, the error distributions should be determined. The error distribution in the Lidar measurement and on the trajectory following are modelled as a function of parameters for which a Gaussian process is used as described in the first section. The next subsections describe the errors in the Lidar, trajectory following, odometry and pose estimation.[1]

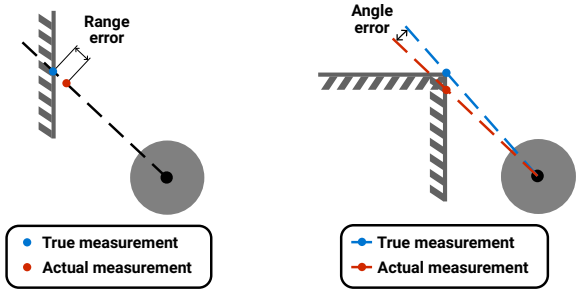
A. Gaussian Processes

Gaussian processes (GPs) are used to model the scalar error y as a distribution $y \sim \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x})^2)$, which is a function of a parameter vector $\mathbf{x}$. Both the $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are modelled by a separate GP to be able to have a heteroskedastic (non-constant) variance. GPs are nonparametric, their hyperparameters can be optimized using the log-likelihood and they model the epistemic uncertainty. We will use GPs with either 1 or 2 parameters, further referred to as 1D and 2D GPs. Error data that is strictly positive can be modelled with a lognormal distribution by converting the original error data $y_{log} = \log(y)$.

Since inference on a GP is computational costly and time-consuming, a lookup table is generated for $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$. These tables are linearly interpolated for online inference. The results in Tab. V in appendix A-A show that the error introduced using the lookup tables is negligible.

B. Lidar error

In general, the Lidar measurement accuracy depends on the distance to, inclination angle to and reflectivity of the reflecting object. External factors like sunlight and rain are also influential. For our implementation, we modelled both the error in the measured ranges and angles with a 2D GP as a function of the parameters range and inclination angle. The parameters are relatively easy to obtain and are expected to have a considerable influence.



(a) Lidar range error

(b) Lidar angle error

Fig. 6: Lidar errors explained

1) Range error: The range error is the distance between the measured and true range (Fig. 6a). Data about the range error was obtained by scanning a straight wall and manually measuring the true distance to it (Fig. 19a in appendix A-B). The true orientation of the wall is obtained by fitting a line through the measurements. Using the true position and orientation, the range errors can be calculated (Fig. 7).

The range errors rapidly increase beyond an inclination angle of 70 degrees. The maximum inclination angle is therefore set at 70 degrees, and any data with an inclination angle bigger than this is not included in the model (GP plot in Fig. 20 in appendix A-B).

The Lidar used in this project does not obtain the inclination angle, so the dependency on the inclination angle was removed by assuming the worst-case value at each range.

2) Angle error: The range error is the discrepancy between the reported angle for a range measurement and the true angle (Fig. 6b). There are at least two different causes for this error: 1) the error in the orientation of the rotating Lidar and 2) the divergence of the Lidar beam.

[1]The code and data for the error models can be found at: https://github.com/christiaantheunisse/Foresee-Error-Models.git

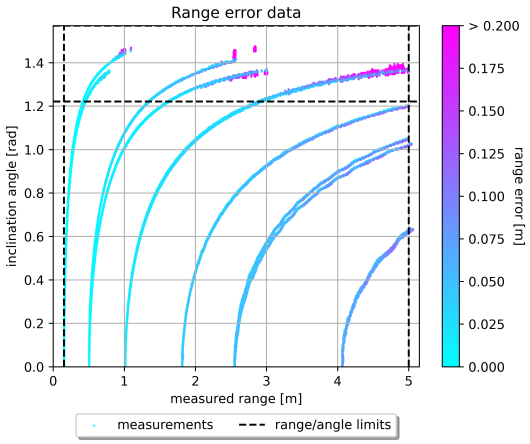


Fig. 7: Range error as function of range and inclination angle.

Angle error data was obtained by scanning an object placed in different positions on a grid that was aligned with the Lidar (Fig. 19b in appendix A-B). The biggest angle with a beam reflecting on the object was compared to the true angle to the edge of the object. Fig. 8 shows the resulting error measurements (GP plot in Fig. 21 in appendix A-B).

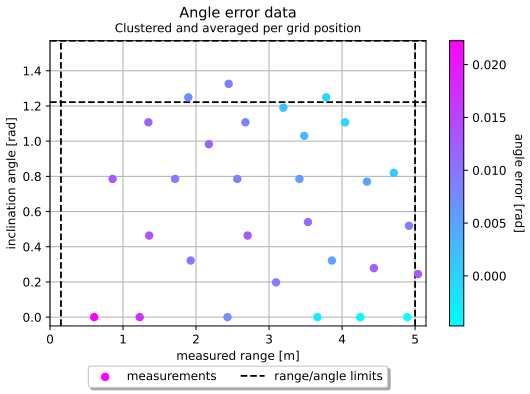


Fig. 8: Angle error as function of range and inclination angle

C. Trajectory following error

The trajectory following error is the error between the estimated pose and the planned pose and is distinguished into three errors (Fig. 9). A trajectory is a list of poses and velocities for each step on the planning horizon.

In general, the trajectory following error in a certain state is expected to depend both on the current and previous states. This is especially true for the poses, since these form the shape of the path, which has a significant influence on the error. The dependency on the previous states asks for a complicated model and a considerable number of test trajectories.

Therefore, our approach focussed on finding the dependencies between the different errors and the parameters used to describe a trajectory: velocity, acceleration and curvature. The acceleration is added because it captures the relation with the velocity in the previous state, and the curvature describes the shape of the path.

Two sets of test trajectories are run to collect error data (more details in appendix section A-C): 1) straight trajectories with varying accelerations and 2) corner trajectories with varying curvatures but constant accelerations. This produces two error datasets, which are used to evaluate the dependencies and fit the corresponding error model for each error:

- `VelAcc`: contains data from the straight trajectories for varying velocities and accelerations and curvature 0.
- `VelCurv`: contains data for varying velocities and curvatures and acceleration 0, obtained from corner trajectories and straight trajectories parts with acceleration 0.

The error models for the trajectory following errors are 2D GPs fitted on one of these datasets. Thus, the parameters of the error models are either velocity and acceleration when using `VelAcc` or velocity and curvature when using `VelCurv`.

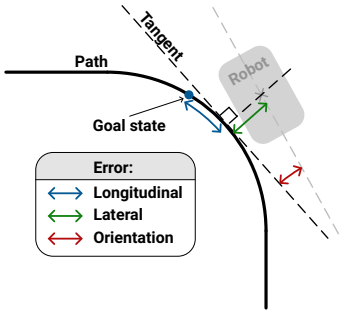


Fig. 9: Trajectory following errors

1) Longitudinal error: The velocity error or the longitudinal error rate is modelled because the trajectory controller tracks the velocity instead of the position. The longitudinal error rate mean and variance are both dependent on the acceleration (Fig. 10a) and the variance also depends on the curvature and velocity (Fig. 10b and c). Therefore, the longitudinal error rate model should include the acceleration, so it is fitted on the `VelAcc` dataset.

2) Lateral error: The absolute value of the lateral error is used, which assumes that the error is equally distributed around zero. As a consequence, the error is strictly positive and a lognormal distribution fits the data better. The lateral error measurements are subsampled because they are not i.i.d., which violate the GP assumptions (appendix A-C).

Fig. 10e and f show that respectively the lateral error mean and variance relate with the velocity and the variance also weakly relates with the curvature. The acceleration data (fig. 10d) is not reliable, since positive accelerations always occur at the start of a trajectory, while the initial lateral error is 0. The error model is fitted on the `VelCurv` dataset because the relation with the acceleration in `VelAcc` is unreliable.

3) Orientation error: The orientation error also uses a lognormal distribution and has been subsampled for similar reasons (appendix A-C). The mean and variance of the error strongly correlate with the curvature, while much weaker with the acceleration (Fig. 10g and h). The variance also has a

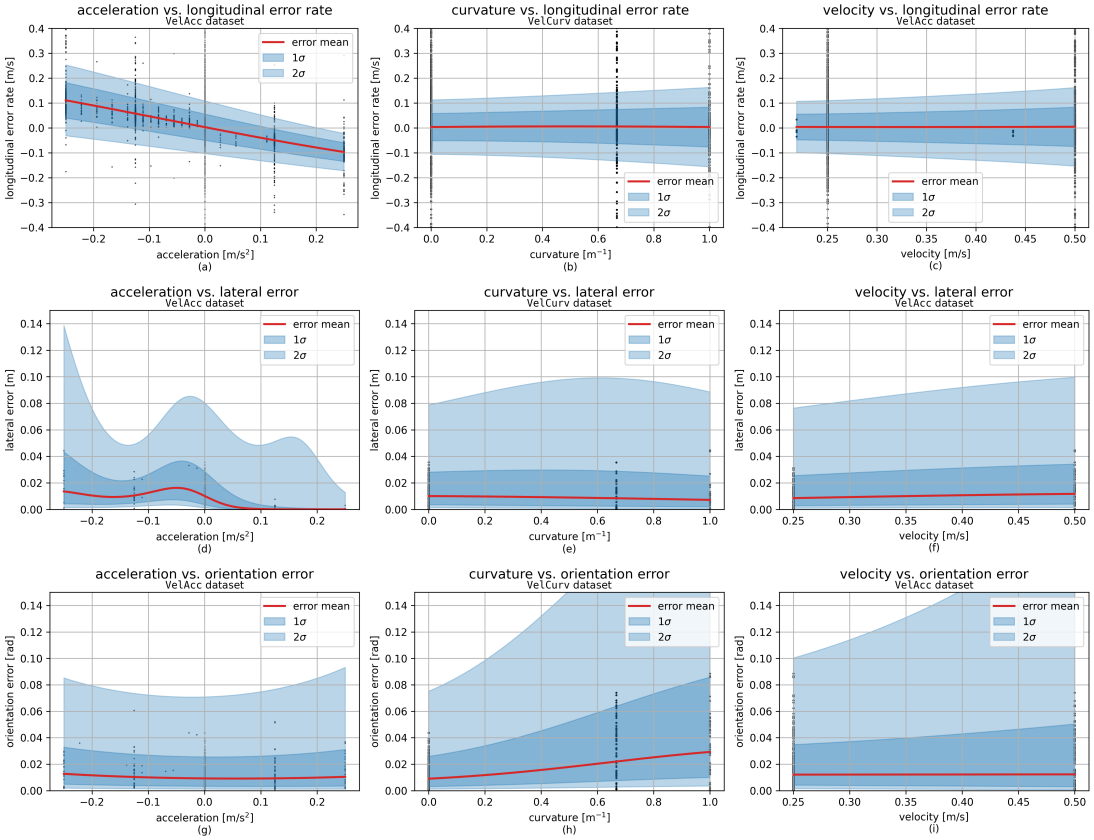


Fig. 10: Trajectory parameters and following errors fitted with 1D GPs to show the correlation.

strong relationship with the velocity (Fig. 10i). Consequently, the error model is fitted on the `VelCurv` dataset.

D. Odometry error

The error distribution of the odometry sensors, the wheel encoders and IMU, will likely depend on the velocity and acceleration. A proper error analysis method will compare the true velocity with the measured velocity for a wide range of velocities and accelerations, and fit a GP. Our approach just models the error with a constant calibration factor and constant variance because it is difficult to obtain true velocity values, especially for the IMU.

For the wheel encoders, the sensor readings were compared to tachometer measurements for different velocities to obtain error data. The model is shown in Fig. 26 in appendix A-D.

For the IMU, the orientation over 30 seconds of a constantly rotating robot reported by the SLAM was used to calculate the average velocity and compared to the average of the velocities from the IMU. This gives at least a rough estimate of the calibration factor. The variance was estimated with the Mean Square Successive Difference (MSSD) [16] because the variance cannot be calculated assuming that the mean is the same for every measurement (Fig. 27 in appendix A-D).

E. Pose estimation error

The pose with its accompanying variance is estimated by the *velocity* and *position EKF*. Both have a tunable process noise covariance matrix $\mathbf{Q}$ that influences the output uncertainty. The state consists of the position (x, y) and orientation ψ and the derivatives, resulting in the state vector: $[x, y, \psi, \dot{x}, \dot{y}, \dot{\psi}, \ddot{x}, \ddot{y}]$.

The matrix $\mathbf{Q}$ of the *velocity EKF* is tuned in such a way that it heavily relies on the input data to leave the predictions to the *position EKF*. This was done by comparing the in- and output velocity plots.

The matrix $\mathbf{Q}$ of the *position EKF* should be tuned such that the estimated uncertainty is at least as conservative as the actual uncertainty. In general, this can be achieved by comparing the measured error distribution to the predicted covariance. More specifically, for a conservatively tuned EKF it should be true that at least $x\%$ of the error values are in the $x\%$-confidence interval for every value in the range [0, 100].

For our approach, the SLAM poses were used as an approximation of the true poses to calculate the error. The chosen $\mathbf{Q}$ matrices (Tab. VI in appendix A-E) result in the distributions in Fig. 11 indicating a conservative uncertainty estimation.

Appendix A-E explains that using the uncertain SLAM pose results in an overapproximation of the actual position error. This effect is mitigated when the EKF pose uncertainty is relatively big compared to the SLAM, which is also reflected by the results for the longer SLAM intervals (Fig. 11).

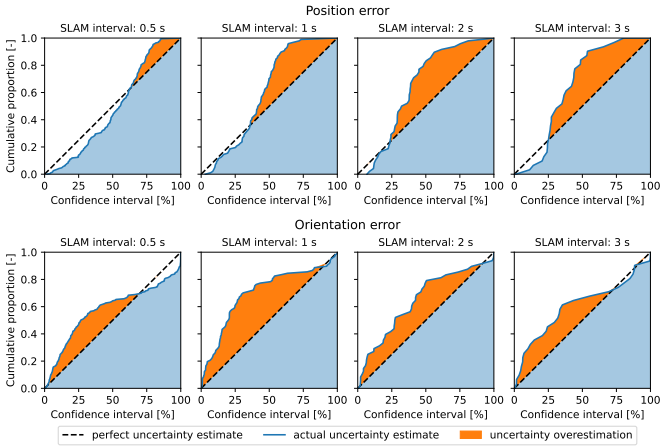


Fig. 11: The cumulative proportion of the position and orientation errors plotted against the EKF confidence interval estimation for different SLAM update intervals.

VI. Inner Workings of the foresee++ Algorithm

This section describes `foresee++`, which accounts for the errors described in section IV using the models from V. The confidence interval to use for each error is exposed as a setting, since we believe that these values are not engineering decisions, but should be a result of public debate.

The size of the error ϵ to account for is calculated from the modelled mean μ, variance σ^2 and the desired confidence interval, the z-score z (Eq. 2). In case a lognormal distribution is used, the predicted distribution has to be converted (Eq. 3).

$$\epsilon = \mu + z \cdot \sigma \tag{2}$$

$$\epsilon = e^{\mu_{log} + z \cdot \sigma_{log}} \tag{3}$$

A. FOV

The FOV is constructed from Lidar measurements (pseudocode in Alg. 1). The first step (line 2) is to ensure that the scan contains exactly a 360° view of the environment. If necessary, a part of the previous scan is added or a part of the current scan is removed. This introduces an inherent, but negligible error as described in appendix B-A.

Next, only 1 in every 5 measurements is used to reduce the computational effort of the following steps (line 8). The error for each range and angle is obtained from the Lidar error models (line 11). In addition, the uncertainty of the ego pose relative to the road is obtained from the EKF.

To construct the FOV (line 16), the position and orientation errors are first added to the range and angle errors, respectively. The angle and range measurements are adjusted to compensate for the errors (Fig. 12a). Subsequently, the ranges are changed to account for the maximum inclination angle (Fig. 12b). The resulting scan is step interpolated in such a way that the FOV is never overestimated (Fig. 12c), which is particularly relevant when the resolution of the Lidar is low. Finally, the scan is converted to 2D points that form the vertices of a polygon.

This polygon is transformed to the road frame using a different transform for each point to account for the time passed during the scan (line 22).

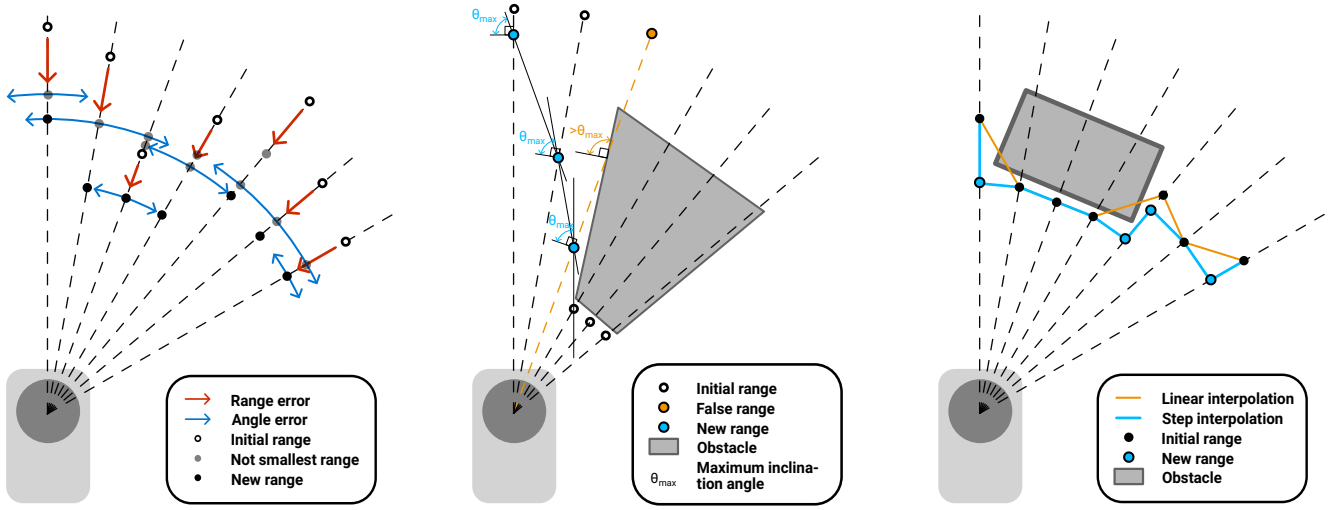
Algorithm 1 FOV construction in `foresee++`

1: **define** $scan$ = Lidar measurement
2: **procedure** COMPLETE SCAN($scan_t, scan_{t-1}$)
3: **if** counterclockwise rotation **then**
4: $scan_{complete} \leftarrow scan_t$ + part of $scan_{t-1}$
5: **else**
6: $scan_{complete} \leftarrow$ part of $scan_t$
7: **return** $scan_{complete}$
8: **procedure** SUBSAMPLE SCAN($scan$)
9: $scan_{sub} \leftarrow$ Keep 1 in 5 (angle, range)-pairs in $scan$
10: **return** $scan_{sub}$
11: **procedure** CALCULATE ERRORS($scan$)
12: Use the Lidar error models to calculate the range
13: ϵ_{ranges} and angle errors ϵ_{angles}.
14: Get the pose uncertainty ϵ_{pose}
15: **return** $\epsilon_{ranges}, \epsilon_{angles}, \epsilon_{pose}$
16: **procedure** BUILD FOV($scan, \epsilon_{ranges}, \epsilon_{angles}, \epsilon_{pose}$)
17: Adjust the $scan$ according to the errors and the
18: maximum inclination angle (see Fig. 12a and b)
19: Step interpolate the $scan$ (see Fig. 12c)
20: Convert the $scan$ to $FOV\ polygon\ vertices$
21: **return** $polygon\ vertices$ (in the Lidar frame)
22: **procedure** DESKEW FOV($scan, transforms$)
23: Transform the $polygon\ vertices$ to the world frame
24: with the appropriate transforms.
25: **return** $polygon\ vertices$ (in the map frame)

B. Reasoning about occlusions

The reasoning and its implementation in `foresee++` (Alg. 2) is almost the same as in `foresee`, described in Eq. 1. On the implementation level, the set of state of possible hidden obstacles $\mathcal{P}$ is considered separately for each different occluded part of the road, referred to as *shadows*. First, the reachable set for each shadow is calculated (line 4). Next, `foresee++` accounts for the fact that there is a significant delay Δt_{FOV} between a Lidar scan and the use of the FOV. This is achieved by applying a padding on the FOV similar to the distance an occluded vehicle can drive in an Δt_{FOV} amount of time (line 8). Finally, the intersection with the FOV is removed from the shadows (line 12).

The calculation of the reachable set is taken from [1], but it is not interaction-aware. This means that it is possible that the reachable set of (occluded) vehicles following the ego extend beyond the position of the ego. This makes it impossible to find collision-free trajectories and therefore freezes the ego. As a workaround for the experiments, it was assumed that no vehicles exist in the starting lane of the mobile robot.



(a) First, the range errors are subtracted from the ranges. Subsequently, each range is projected on several adjacent angles, with the exact number depending on the angle error.

(b) The difference between adjacent ranges should respect the maximum inclination angle to account for objects with a too large inclination angle w.r.t. the Lidar.

(c) Extra (angle, range)-pairs are added to the scan if necessary to ensure that the FOV is step interpolated and not linearly, to avoid intersecting obstacles.

Fig. 12

Algorithm 2 Reasoning about occlusions in `foresee++`

1: **define** $shadow$ = area that possibly contains occl. vehicles
2: **define** Δt_{step} = time since last update step
3: **define** Δt_{FOV} = delay between scan and FOV use
4: **procedure** CALC REACHABLE SET($shadows$, Δt_{step})
5: **for** $shadow$ **in** $shadows$ **do**
6: Calculate the reachable set for Δt_{step}
7: **return** $\mathcal{R}(shadows)$
8: **procedure** FOV ACCOUNT DELAYS(FOV, Δt_{FOV})
9: $d_{padding} \leftarrow v_{max} \cdot \Delta t_{FOV}$
10: $FOV_{pad} \leftarrow$ Apply a padding of $d_{padding}$ on the FOV
11: **return** FOV_{pad}
12: **procedure** UPDATE SHADOWS($shadows_{reach}$, FOV_{pad})
13: **for** $shadow$ **in** $\mathcal{R}(shadows)$ **do**
14: $shadow \leftarrow shadow \backslash FOV_{pad}$ ▷ Set difference
15: **return** $shadows_{new}$

C. Trajectory planning

A safe trajectory avoids the reachable sets of the shadows and ends with a safe state. A safe state is defined as a standstill outside the no-stop-zone, which is, in our case, the intersection. The trajectory planner (Alg. 3) therefore, starts with generating velocity profiles that end with zero (line 4).

Since the total planning time is limited, the velocity profiles are considered one at a time to ensure that there is a safe trajectory found as soon as possible (line 9). This starts with mapping the profile along the center of the lane, resulting in the poses and velocities that form a trajectory.

Subsequently, the errors are obtained for each state using the velocity and acceleration in each state. The highest curvature in the trajectory is used for every state to ensure a conservative error estimation, which has two reasons: 1) the curvature in one state is expected to influence the errors in subsequent states and 2) the errors increase with the curvature. The longitudinal error rate is integrated to get the absolute error, while the variance is corrected (see appendix B-B). For the pose uncertainty, the worst-case value encountered during normal operation is used.

Next, the set of occupancies for each time step is constructed by placing the ego shape at each planned pose and expanding it with the errors (Fig. 13). The orientation errors are applied by calculating the additional length and width.

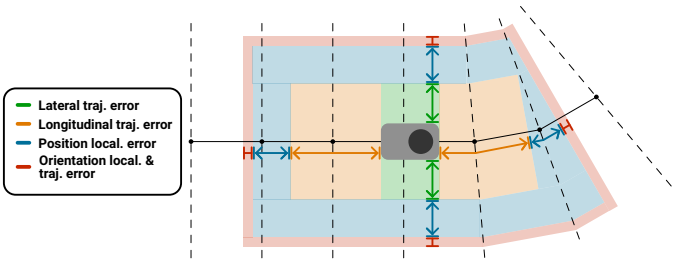


Fig. 13: Occupancy set construction: the shape of the ego vehicle is extended with the errors.

If the occupancy set does not collide with the area projection of the reachable sets of the shadows, it is stored as the safe trajectory and returned when the planning time is over. The velocity profiles are selected with a branch-and-bound approach

that selects only profiles faster than the safe trajectory.

Algorithm 3 Trajectory planning in `foresee++`

define *v-profile* = list of velocities v
define $v_{current}$ = current velocity
define v_{next} = planned velocity at end of planning step
procedure GENERATE V-PROFILES($v_{current}$, v_{next})
    Make trapezoidal *v-profiles* starting with
        $v_{current}$ and v_{next} for the first two velocities
    return Set of *v-profiles*
$trajectory_{safe} \leftarrow$ empty variable for a trajectory
while time left for planning **do**
    B&B approach to select a *v-profile* from *v-profiles*
    procedure GENERATE TRAJECTORY(*v-profile*, $lane$)
        $trajectory \leftarrow$ Map the *v-profile* along the $lane$
        return $trajectory$
    procedure GET THE ERRORS($trajectory$)
        Get the trajectory following errors $\epsilon_{traject}$
        Get the localization errors ϵ_{local}
        return $\epsilon_{traject}, \epsilon_{local}$
    procedure MAKE OCCUPANCY SET($trajectory$, $\epsilon_{traject}$, ϵ_{local})
        Construct the *occupancy set* from the $trajectory$
            and the errors $\epsilon_{traject}$ and ϵ_{local} (Fig. 13)
        return *occupancy set*
    procedure COLLISION CHECK(*occupancy set*)
        Check for collisions with (occluded) vehicles
        if collision-free **then**
            $trajectory_{safe} \leftarrow trajectory$
return $trajectory_{safe}$

VII. EXPERIMENTS

We evaluate `foresee++` in two parts: 1) experiments that focus on the cautiousness and 2) an ablation study of the different error measures. The cautiousness is quantified by 1) the *traversal time* until the intersection is crossed and 2) the *gap size* between two vehicles necessary for the ego to cross.

Two other mobile robots are used as obstacle cars. Additional vehicles can be simulated to be able to do experiments with more than 2 obstacle cars and use a four-way intersection despite the limited space. The four-way intersection used consists of 0.5 meters wide lanes which is necessary because of the errors, 1 meter corners to align with the error models and a median strip of 0.06 meters (see Fig. 16 for an impression). The robot hardware is discussed in appendix section C.[2]

A. Performance experiments

The first series of experiments explore the effect of error models and reasoning on the performance, resulting in the following algorithms:

1) `baseline`: algorithm without reasoning, so the set $\mathcal{P}^i$ is calculated as $\mathcal{P}^0$ in Eq. 1 in every update step.

[2]The code for the algorithms can be found at: https://github.com/christiaantheunisse/Foresee-the-Unseen-ROS.git

TABLE I: Parameters for the experiments

Parameter	Performance experiments	Ablation study	Unit
Planning frequency	3	3	Hz
Planning Horizon	7.33	6.67	s
FOV range	5	5	m
Ego goal velocity	0.4	0.4	m/s
Obstacle maximum velocity	0.48	0.48	m/s
Maximum acceleration	0.2	0.2	m/s^2
Maximum deceleration	0.25	0.25	m/s^2
z-values* FOV			
Range error	2	2	-
Angle error	2	2	-
Localization position error	2	2	-
Localization orientation error	2	2	-
z-values* occupancy set			
Longitudinal traj. follow error	2	2	-
Lateral traj. follow error	1	1	-
Orientation traj. follow error	1	1	-
Localization position error	1	1	-
Localization orientation error	1	1	-

*z-values are the values for z in Eq. 2

2) `foresee`: algorithm described in section III
3) `baseline++`: `baseline` with error measures.
4) `foresee++`: `foresee` with error measures.

Each algorithm runs three times on each scenario, unless otherwise specified, and the parameters are given in Table I.

1) Normal traffic scenarios: The first experiment considers six normal traffic situations from the 300 scenarios in [1]. Two with no traversal time improvement, two with a considerable improvement and two where `foresee` even finds a gap earlier than `baseline`.

Reference [1] uses SUMO [17] to simulate the obstacle cars. Our intersection layout was converted to SUMO and the start positions were changed to maintain the same travel time to the intersection. Subsequently, simulations were run to obtain trajectories for the simulated and obstacle vehicles.

The traversal times are shown in Fig. 14. Appendix C-A discusses the statistical significance of each of the time improvements using the T-test. In essence, it can be concluded that for all scenarios, the improvement between `baseline` and `foresee` is significant. For the algorithms with error measures, scenarios 2, 3 and 6 show a significant improvement.

The biggest advantage for algorithms that reason is achieved when the ego is waiting for a vehicle that leaves in the east direction because this vehicle blocks the view of the adjacent lane for a long period of time. By reasoning, it can be concluded that the adjacent lane cannot contain vehicles, giving these algorithms the advantage of not having to wait until the lane is observable again. This effect results in the significant difference for scenarios 2, 3 and 6 and is visualized for the algorithms with error models in Fig. 15a.

In scenario 1 there are two vehicles leaving in the east direction. `Foresee` can still gather enough information about the adjacent lane to have an advantage over `baseline`. `Foresee++`, on the other hand, cannot because the two

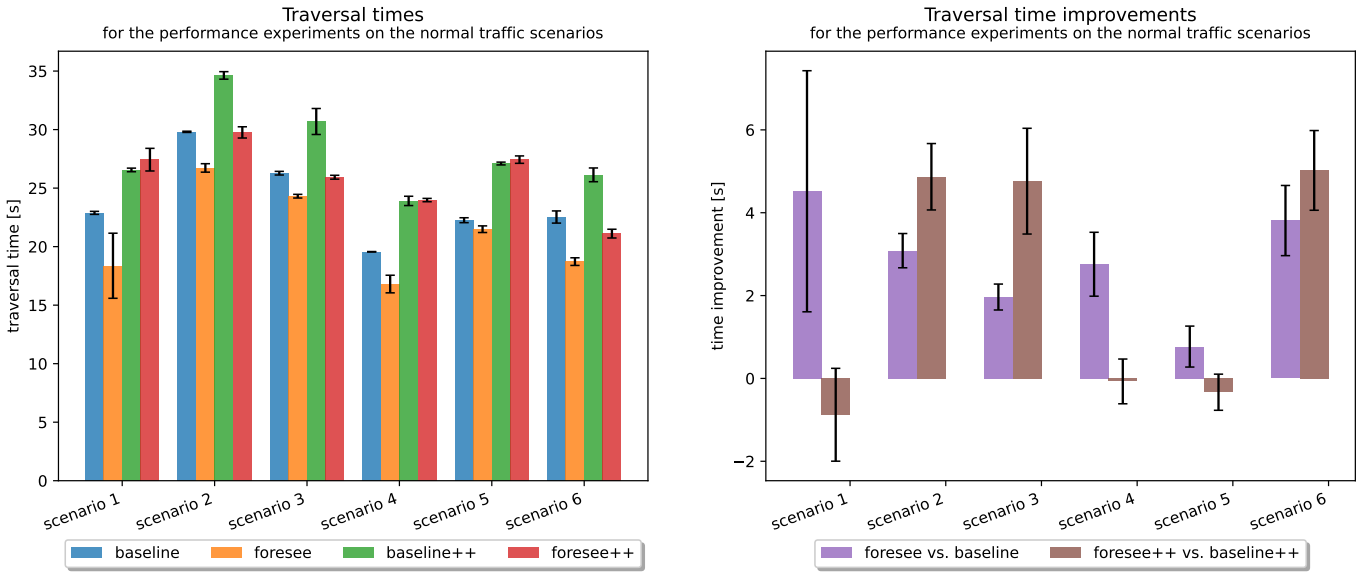


Fig. 14: Traversal times recorded on the normal traffic scenarios during the performance experiments.

vehicles are too close, and the error measures on the FOV block its sight (Fig. 15b).

In scenario 4, two vehicles are leaving the intersection at approximately the same time in east and west direction. `Foresee` can merge behind the vehicle in the west direction and has the advantage of reasoning for the vehicle driving east. However, `foresee++` needs more space to merge and while it is waiting for that, it looses the required information about the east lane to have an advantage (Fig. 15c).

In scenario 5, the ego has to give priority to a vehicle driving from north to south. Due to reasoning, `foresee` does not have to wait for the vehicle to clear its view on the west lane, while `baseline` does. However, due to the maximum inclination angle and the close distance to the passing vehicle, the FOV for `foresee++` is greatly reduced. Consequently, it cannot gather enough information to outperform `baseline++` (Fig. 15d).

Besides, a test was performed to find out if the error models make the ego behave more cautious. Since this requires to compare more than two distributions, i.e., multiple algorithms over six scenarios, a two-way ANOVA test was used which indicates that algorithms with error models are significantly more conservative than algorithms without (Appendix C-A).

2) Gap size: The following experiment measures the minimum gap size for each algorithm. The gap size was considered big enough if the algorithm did not fail to cross in the first three attempts. The results (Tab. II) show an approximately similar gap size reduction by reasoning for both the algorithms with and without error models.

3) Parked vehicle: The simulation experiments in [1] demonstrated that `foresee` can solve the "freezing robot problem" (FRP). The goal is to find out if that ability is retained for `foresee++`. Therefore, a parked vehicle is placed to the right of the ego lane and the distance from the intersection is varied (Fig. 16a).

TABLE II: Minimum gap size for performance experiments

Algorithm	Gap size (m)	Reduction
`baseline`	3.9	-31%
`foresee`	2.7	
`baseline++`	6.7	-30%
`foresee++`	4.7	

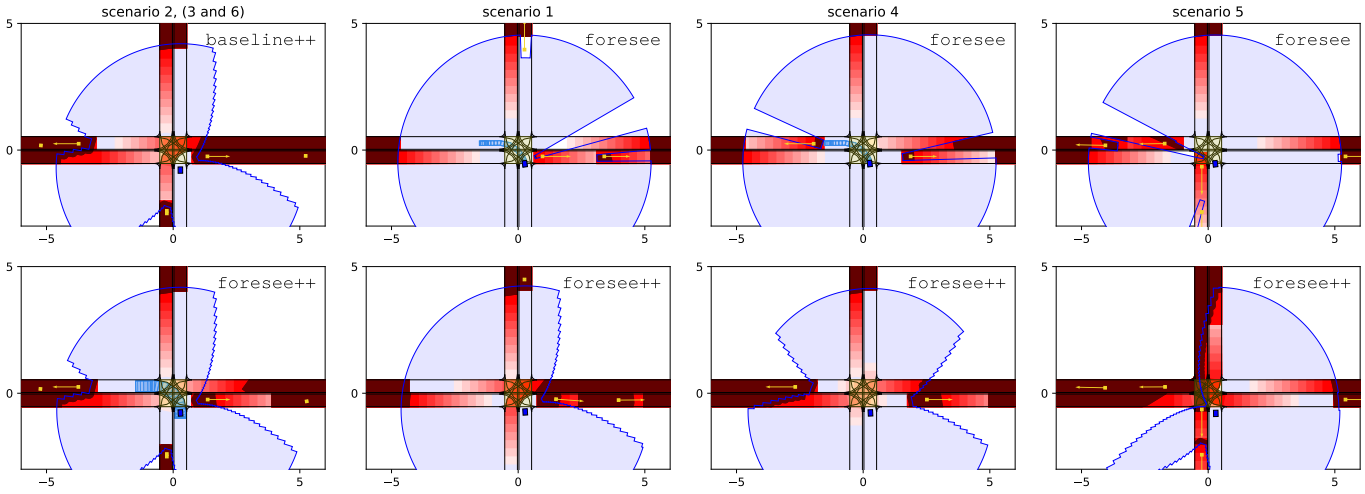
The different parked vehicle positions are given in Table IV together with the number of successful crossings out of 7 tries. The no-stop-zone size is increased in the first setup to create an even harder scenario, allowing `foresee` to demonstrate its capabilities. The no-stop-zone is increased with an additional 0.17 m for the algorithms without the error measures, since this is the mean distance with which they violated the no-stop-zone in the normal traffic experiments.

The results (Tab. IV) show that `foresee` can indeed solve the FRP, but `foresee++` lost the ability. Fig. 16b shows the reason for this, namely that the error measures for the FOV make it impossible for the ego to look around the parked car into the relevant lane before passing the parked car.

B. Ablation study

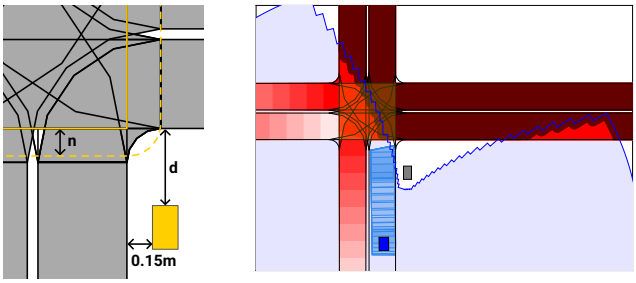
The ablation study aims to find the contribution of each error measure, and thus each error, to the increased cautiousness of `foresee++`. The ablated versions of `foresee++` are:

- `foresee++ abl. FOV`: does not consider the Lidar errors, maximum inclination angle, pose uncertainty and delays in the FOV
- `foresee++ abl. Lidar`: does not consider the Lidar errors, maximum inclination angle and pose uncertainty in the FOV



(a) Scenario 2, (3 and 6) (b) Scenario 1 (c) Scenario 4 (d) Scenario 5

Fig. 15: The position of the obstacle cars (yellow boxes with velocity vectors) is updated in real-time, but the shadows (dark red areas with predictions), trajectory (cyan areas) and FOV (blue area) are updated after the planner computations.



(a) The setup used in the experiments with the parked vehicle. d = distance to the intersection and n = no-stop-zone size increase.

(b) `Foresee++` cannot view the lane behind the parked vehicle because of the error measures for the FOV.

Fig. 16: Parked vehicle scenario

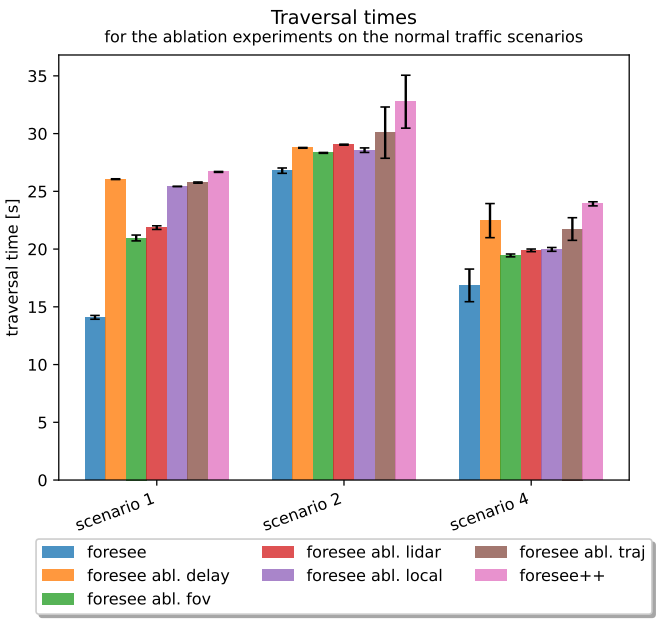


Fig. 17: Traversal times average over three runs for the ablation study on the normal traffic scenarios.

- `foresee++ abl. local`: does not consider the pose uncertainty in the FOV and occupancy set
- `foresee++ abl. traject`: does not consider the trajectory following errors in the occupancy set
- `foresee++ abl. delay`: does not consider the delays in the FOV

1) Converted experiments: The normal traffic scenarios 1, 2 and 4 are chosen from the performance experiments because they provide three unique obstacle vehicles configurations. The resulting traversal times are shown in Fig. 17.

To determine if there are statistically significant differences among the ablated algorithms, Tukey's HSD test was used to do a multiple comparison test. This test is often used and recommended in the literature as a robust method for post hoc testing, that is neither extremely conservative nor liberal [18], [19]. The details of the test results are described in appendix section C-B, and indicate that the reduction in cautiousness is significantly more for `foresee++ abl. Lidar` and `FOV` than for `foresee++ abl. delay` and `traj`. The performance of `foresee++ abl. local` is somewhere in the middle of the two aforementioned groups. This means that the errors in the perception limit the performance the most.

2) Gap size: The experiments to find the minimal gap were also done for the ablated algorithms (Table III). The results show a similar or better performance for the ablations concerning the occupancy set calculation, namely `foresee++ abl. local` and `traject` compared to the perception

ablations `foresee++ abl. FOV` and `Lidar`.

TABLE III: Minimum gap size for ablation study

Algorithm	Gap size (m)
`foresee`	2.6
`foresee++ abl. FOV`	3.9
`foresee++ abl. Lidar`	4.5
`foresee++ abl. local`	3.7
`foresee++ abl. traject`	3.9
`foresee++ abl. delay`	4.4
`foresee++`	4.6

3) Parked vehicle: The ablated algorithms are compared against a version of `baseline++` with similar ablations to ensure a fair comparison. The results are shown in Table IV. It can be concluded that `foresee++ abl. FOV` can solve the FRP in some scenarios. The data on `foresee++ abl. Lidar` indicates that a specific scenario might be found, where it can convincingly solve the FRP.

TABLE IV: Parked vehicle scenario experiments results

Scenario setup					
Distance to intersection (m)*	0.0	0.0	0.1	0.3	0.5
No-stop-zone increase (m)*	0.15	0.0	0.0	0.0	0.0
Algorithm	**Number of successful passes out of seven runs**				
`foresee`	7	7	7	7	7
`baseline`	0	7	7	7	7
`foresee++ abl. FOV`	7	7	7	7	7
`baseline++ abl. FOV`	-	0	7	-	-
`foresee++ abl. Lidar`	0	3	7	7	7
`baseline++ abl. Lidar`	-	0	7	-	-
`foresee++ abl. local`	0	0	0	7	7
`baseline++ abl. local`	-	-	-	7	-
`foresee++ abl. traj`	0	0	0	7	7
`baseline++ abl. traj`	-	-	-	7	-
`foresee++ abl. delay`	0	0	0	7	7
`baseline++ abl. delay`	-	-	-	7	7
`foresee++`	0	0	0	0	7
`baseline++`	0	0	0	0	7

Scenario where reasoning solves the FRP

*These are defined in Fig. 16a

VIII. Discussion

A. Does reasoning reduce the cautiousness?

The performance experiments on the normal traffic scenarios show that `foresee++` provides an advantage by reasoning in some situations despite the error measures. However, in other situations this advantage is gone. The *gap size* scenarios similarly showed that `foresee++` can provide the same proportional improvement as `foresee`. This leads to the conclusion that reasoning still has an advantage despite the error measures, but the threshold for the available information is higher and the advantage is mostly all or nothing.

B. Can `foresee++` solve the FRP?

Furthermore, it is shown that `foresee++` lost the ability to solve the FRP. However, `foresee++ abl. FOV` and `foresee++ abl. Lidar` (partly) regain the ability to solve the FRP.

C. Which errors increase the cautiousness the most?

The FRP results lead to the conclusion that the errors in the perception part of the algorithm are the greatest limitation. This is supported by the performance of the ablations on the normal traffic scenarios, since `foresee++ abl. FOV` and `Lidar` both perform significantly better than `foresee++ abl. delay` and `traject`. An important footnote is that `foresee++ abl. FOV` ablates many error measures and is expected to perform well in any case. Nevertheless, `foresee++ abl. Lidar` shows a similar improvement, indicating that the perception is the main bottleneck.

This makes the results from the *gap size* experiments for the ablated algorithms particularly interesting. Here, the ablations considering the occupancy set, namely `foresee++ abl. local` and `traject`, are among the best performing ones. This gives rise to the conclusion that when perception is not the bottleneck, there is a lot to win by reducing the necessary margins on the occupancy set calculation.

D. What is the influence of not being interaction-aware?

Besides, it was noticed that the fact that the planner is not interaction-aware reduces the performance considerably. When the ego wants to cross an intersection, the reachable set of nearby vehicles coming from other directions and moving into the same lane extend from behind over the occupancy set of the ego. This results in a collision for every possible trajectory when the other vehicles are too close. This is amplified because the other vehicles are assumed to drive faster to account for minor speed violations, meaning that a longer planning horizon will perform worse. This is supported by the results in Fig. 18, where a longer horizon results in a longer traversal time. Besides, for a planning horizon of 30 steps, the ego cannot even pass the intersection because it collides with potentially occluded vehicles at the boundary of the FOV.

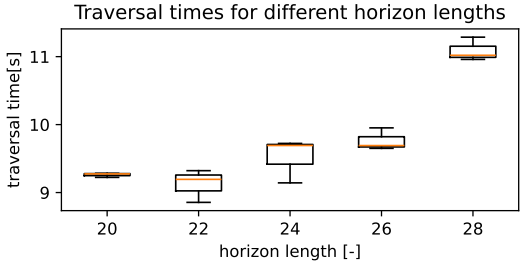


Fig. 18: Traversal times for an empty intersection for three runs for different horizons with planning time step 0.33s.

E. What are occlusion tracking problems to solve?

Another problem which is not accounted for, is that the Lidar very rarely does not have a single detection on an obstacle car, meaning that the vehicle is completely missed. Subsequently, a related problem will occur, namely, that due to

the reasoning, once the algorithm thinks that the area occluded by this missed vehicle is clear, it will lose track of this area and thus of this vehicle. Despite detecting the vehicle again in the next scan, this area is not re-added and collisions are likely to happen. This could potentially also happen for a vehicle that is substantially exceeding the speed limit.

IX. Conclusions

A set-based planner that properly handles imperfect input data and reasons about the set of hidden obstacles has been implemented. The experiments indicate that reasoning still provides an advantage and is thus worth further development.

Since the perception of the environment which results in the FOV contains the most limiting source of error, it is probably the most relevant issue to work on. This can be achieved by adding other sensors, like a camera, or improving the current sensors, e.g., implementing a Lidar algorithm that obtains the inclination angle for each measurement [20].

Another important issue to solve is to adjust the reasoning process to deal with vehicles the algorithm looses track of. This could be achieved by adding Lidar detections within the road boundaries to the set of possible hidden obstacles.

A possibility that will significantly reduce the cautiousness around other vehicles is to make the algorithm interaction-aware, like the method in [21]. In this approach, the reachable set of the (occluded) vehicles is constraint to never extend beyond the vehicle in front.

Lastly, it has been argued that the confidence intervals for each error measure should be the result of public debate. Regardless of the chosen values, the errors will exceed these intervals occasionally. To be able to make well-informed decisions, it is necessary to provide information about the possible consequences and accompanying probabilities for each error exceedance. For instance, not every underestimation of the localization error will result in an accident. Besides, this will also differ for each environment, e.g., highway or urban, and thus require different settings for each environment.

The suggestions are just a few pointers resulting from this research to accomplish a reliable and swift set-based algorithm for trajectory planning in AVs.

References

[1] J. M. G. Sanchez, T. Nyberg, C. Pek, J. Tumova, and M. Torngren, "Foresee the unseen: Sequential reasoning about hidden obstacles for safe driving," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, Aachen, Germany: IEEE, Jun. 5, 2022, pp. 255–264, ISBN: 978-1-66548-821-1. DOI: 10.1109/IV51971.2022.9827171. [Online]. Available: https://ieeexplore.ieee.org/document/9827171/ (visited on 11/14/2023).

[2] J. Deichmann, E. Ebel, K. Heineke, R. Heuss, M. Kellner, and F. Steiner, "The future of autonomous driving: Convenient and connected," Tech. Rep., Jan. 2023. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected.

[3] NHTSA, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," DOT HS 812 506, Mar. 2018. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506 (visited on 01/10/2024).

[4] P. F. Orzechowski, A. Meyer, and M. Lauer, "Tackling occlusions & limited sensor range with set-based safety verification," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI: IEEE, Nov. 2018, pp. 1729–1736, ISBN: 978-1-72810-323-5. DOI: 10.1109/ITSC.2018.8569332. [Online]. Available: https://ieeexplore.ieee.org/document/8569332/ (visited on 11/14/2023).

[5] M. Koschi and M. Althoff, "Set-based prediction of traffic participants considering occlusions and traffic rules," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 249–265, Jun. 2021, ISSN: 2379-8904, 2379-8858. DOI: 10.1109/TIV.2020.3017385. [Online]. Available: https://ieeexplore.ieee.org/document/9170860/ (visited on 11/14/2023).

[6] Z. Qiao, K. Muelling, J. Dolan, P. Palanisamy, and P. Mudalige, "POMDP and hierarchical options MDP with continuous actions for autonomous driving at intersections," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, ISSN: 2153-0017, Nov. 2018, pp. 2377–2382. DOI: 10.1109/ITSC.2018.8569400. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8569400 (visited on 12/04/2023).

[7] D. Kamran, C. F. Lopez, M. Lauer, and C. Stiller, "Risk-aware high-level decisions for automated driving at occluded intersections with reinforcement learning," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, ISSN: 2642-7214, Oct. 2020, pp. 1205–1212. DOI: 10.1109/IV47402.2020.9304606. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9304606 (visited on 12/04/2023).

[8] L. Sun, W. Zhan, C.-Y. Chan, and M. Tomizuka, "Behavior planning of autonomous cars with social perception," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, ISSN: 2642-7214, Jun. 2019, pp. 207–213. DOI: 10.1109/IVS.2019.8814223. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8814223 (visited on 12/14/2023).

[9] K. Sama, Y. Morales, H. Liu, *et al.*, "Extracting human-like driving behaviors from expert driver data using deep learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9315–9329, Sep. 2020, Conference Name: IEEE Transactions on Vehicular Technology, ISSN: 1939-9359. DOI: 10.1109/TVT.2020.2980197. [Online]. Available: https://ieeexplore.ieee.org/document/9037102 (visited on 12/04/2023).

[10] C. Hubmann, N. Quetschlich, J. Schulz, J. Bernhard, D. Althoff, and C. Stiller, "A POMDP maneuver planner for occlusions in urban scenarios," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, Paris, France: IEEE,

Jun. 2019, pp. 2172–2179, ISBN: 978-1-72810-560-4. DOI: 10.1109/IVS.2019.8814179. [Online]. Available: https://ieeexplore.ieee.org/document/8814179/ (visited on 11/22/2023).

[11] C. Zhang, F. Steinhauser, G. Hinz, and A. Knoll, "Improved occlusion scenario coverage with a POMDP-based behavior planner for autonomous urban driving," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, IN, USA: IEEE, Sep. 19, 2021, pp. 593–600, ISBN: 978-1-72819-142-3. DOI: 10.1109/ITSC48978.2021.9564424. [Online]. Available: https://ieeexplore.ieee.org/document/9564424/ (visited on 12/21/2023).

[12] C. Zhang, S. Ma, M. Wang, G. Hinz, and A. Knoll, "Efficient POMDP behavior planning for autonomous driving in dense urban environments using multi-step occupancy grid maps," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, Macau, China: IEEE, Oct. 8, 2022, pp. 2722–2729, ISBN: 978-1-66546-880-0. DOI: 10.1109/ITSC55140.2022.9922353. [Online]. Available: https://ieeexplore.ieee.org/document/9922353/ (visited on 12/21/2023).

[13] K. H. Wray, B. Lange, A. Jamgochian, *et al.*, "POMDPs for safe visibility reasoning in autonomous vehicles," in *2021 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, Tokoname, Japan: IEEE, Mar. 4, 2021, pp. 191–195, ISBN: 978-1-66543-862-9. DOI: 10.1109/ISR50024.2021.9419519. [Online]. Available: https://ieeexplore.ieee.org/document/9419519/ (visited on 11/29/2023).

[14] L. Wang, C. Burger, and C. Stiller, "Reasoning about potential hidden traffic participants by tracking occluded areas," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, IN, USA: IEEE, Sep. 19, 2021, pp. 157–163, ISBN: 978-1-72819-142-3. DOI: 10.1109/ITSC48978.2021.9564584. [Online]. Available: https://ieeexplore.ieee.org/document/9564584/ (visited on 11/14/2023).

[15] S. Thrun, M. Montemerlo, H. Dahlkamp, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[16] J. von Neumann, R. H. Kent, H. R. Bellinson, and B. I. Hart, "The mean square successive difference," *The Annals of Mathematical Statistics*, vol. 12, no. 2, pp. 153–162, 1941, ISSN: 00034851. [Online]. Available: http://www.jstor.org/stable/2235765 (visited on 03/13/2024).

[17] P. Alvarez Lopez, M. Behrisch, L. Bieker-Walz, *et al.*, "Microscopic traffic simulation using sumo," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, Nov. 2018, pp. 2575–2582. [Online]. Available: https://elib.dlr.de/127994/.

[18] G. Ingersoll, "Multiple comparisons of means: A practical guide," *International Journal for Research in Education*, vol. 28, pp. 15–39, 2010.

[19] S. Midway, M. Robertson, S. Flinn, and M. Kaller, "Comparing multiple comparisons: Practical guidance for choosing the best multiple comparisons test," *PeerJ*, vol. 8, e10387, 2020.

[20] T. Yang, J. Lai, C. Wang, *et al.*, "Influence of a target's inclination on lidar waveform and its application," *IET Optoelectronics*, vol. 16, no. 1, pp. 27–33, 2022.

[21] M. Koschi and M. Althoff, "Interaction-aware occupancy prediction of road vehicles," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2017, pp. 1–8.

[22] G. M. Ljung and G. E. Box, "On a measure of lack of fit in time series models," *Biometrika*, vol. 65, no. 2, pp. 297–303, 1978.

APPENDIX A
MODELLING AND MAPPING THE ERRORS

A. Lookup table errors

Using a lookup table introduces an error. Tab. V gives the maximum error size found for each of the error models, indicating that the errors are negligible.

TABLE V: Lookup table errors

Error model	Maximum error		Average value	
	μ	σ	$\mu_{average}$	$\sigma_{average}$
Lidar range [m]	3.24e-7	1.16e-6	0.036	0.0078
Lidar angle [rad]	1.22e-7	3.88e-8	0.0078	0.0057
Trajectory longitudinal rate [m/s]	1.18e-7	3.01e-8	0.0044	0.054
Trajectory lateral [log(m)]*	4.97e-7	7.59e-7	-4.86	1.27
Trajectory orientation [log(rad)]*	1.59e-7	2.00e-7	-4.13	1.17

*these are lognormal distributions

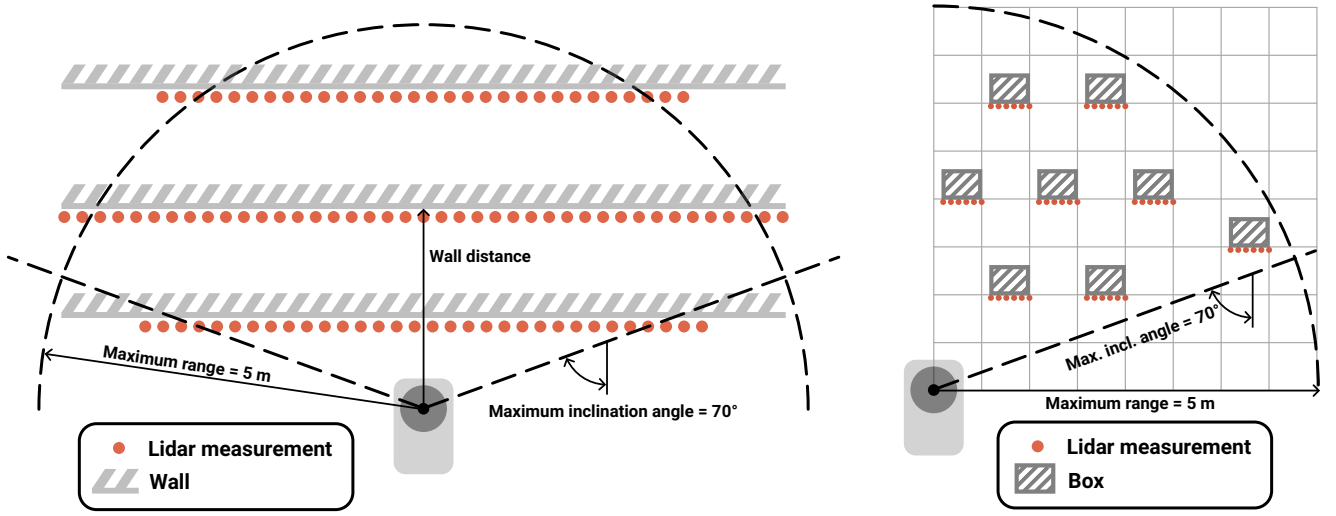
B. Lidar error

The procedure to collect data for the range error of the Lidar is visualized in Fig. 19a. Similarly, the procedure for the angle error is visualized in Fig. 19b.

The Gaussian process fits for the range and angle data are visualized in Fig. 20 and 21, respectively.

C. Trajectory following error

1) Test trajectories configurations: The maximum velocity of the robot is around 0.6 m/s with a full battery. With a horizon length of approximately 7 seconds, this results in trajectories at most 4 meters long. Besides, the acceleration should be limited to achieve acceleration and deceleration times similar to a real vehicle. This results in the following parameters for the test trajectories with varying velocity v, acceleration a, curvature $kappa$ and length l. The straight trajectories ($\kappa = 0$ m^{-1}) have a trapezoidal velocity profile with varying velocities ($v \in \{0.25, 0.5\}$ m/s), accelerations



(a) Method to collect Lidar range error data

(b) Method to collect Lidar angle error data

Fig. 19: The methods to collect Lidar error data

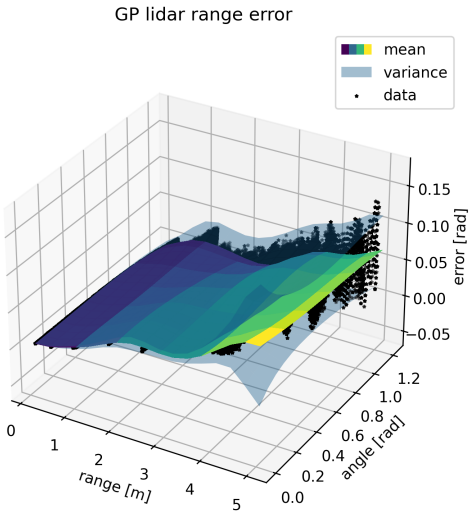


Fig. 20: GP that models the Lidar range error distribution as a function of the range and inclination angle.

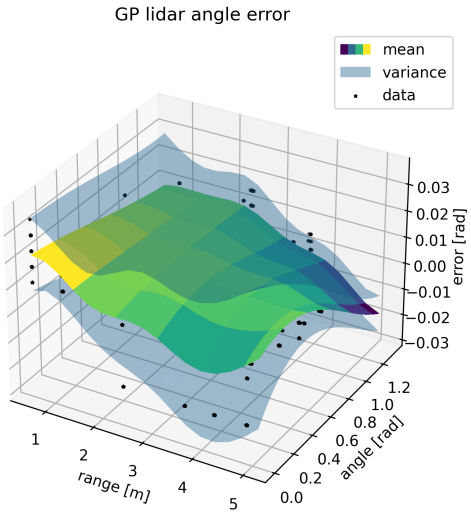


Fig. 21: GP that models the Lidar angle error distribution as a function of the range and inclination angle.

($a \in \{0.125, 0.25\}$ m/s^2) and lengths ($l \in \{0.4, 0.8, ..., 4\}$ m). The constant velocity trajectories ($a = 0$ m/s) with a corner have varying curvature ($\kappa \in [0.67, 1]$ m^{-1}) and velocity ($v \in [0.25, 0.5]$ m/s).

2) Subsampling of the lateral and orientation errors: The error data consists of 30 measurements per second. However, the subsequent lateral and orientation error measurements are each not independent over time because the robot cannot immediately change its position. However, one of the assumptions of the input data of a Gaussian Process (GP) is that it is i.i.d, so subsampling is needed

The Ljung-Box test [22] relies on the autocorrelation and provides a formal approach to quantify the probability that the data is i.i.d. More specifically, it calculates the probability of observing similar or more extreme data given that the hypothesis that the data is i.i.d. is true. A probability smaller than 0.05 is considered to be too small for the hypothesis to hold and therefore means that the data is not i.i.d.

Since every test trajectory is a different series, the data measurements should be subsampled with such a rate that the data of 95% of all trajectories has a probability bigger than 0.05. By using only 1 in 30 samples for the orientation error and 1 in 60 for the lateral error, i.e., one per second and one per two seconds, respectively, the error data of 52 out of 55 test trajectories has a score bigger than 0.05 for both errors.

The error measurements of the different trajectories are

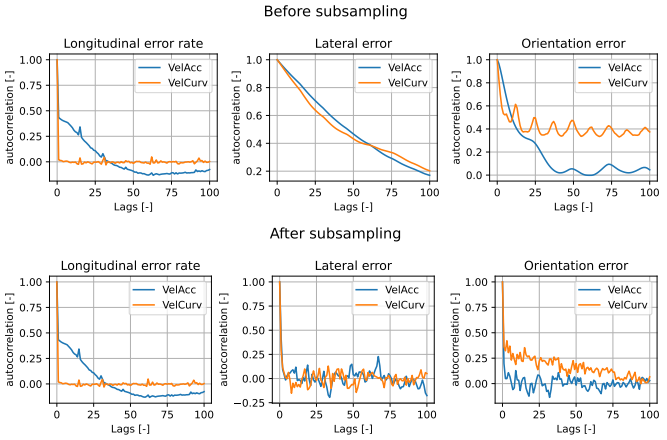


Fig. 22: Autocorrelation values for each error and dataset before and after the subsampling.

concatenated for each dataset, and the autocorrelations before and after the subsampling are shown in Fig. 22. Although concatenating the trajectories is not strictly proper when calculating the autocorrelations, it gives an indication of the independence.

When it comes to the longitudinal error rate, the error rate depends on the velocity, which changes much more rapidly. The autocorrelation does not work for the straight start-stop trajectories (`VelAcc` dataset), since the acceleration varies over these trajectories. As a consequence, the autocorrelation assesses the general trend, which is expected to show a strong correlation. This is indeed observed in Fig. 22 for the blue line belonging to the `VelAcc` dataset.

The Ljung-Box test is therefore only applied for the data from the corner trajectories, where the velocity is constant. As expected, the autocorrelation is entirely different for these trajectories, as shown by the line for the `VelCurv` dataset in Fig. 22. This is also supported by the results from the Ljung-Box test, where all corner trajectories have a probability bigger than 0.05, rendering subsampling not necessary.

3) The Gaussian process fits of the errors: The longitudinal, lateral and orientation error (rate) distribution fits are given in Fig. 23, 24 and 25, respectively.

D. Odometry errors

The velocity errors are described with a constant calibration factor and a variance estimate. The results for the wheel encoders and the IMU are visualized in Figures 26 and 27, respectively. For the IMU, the highest MSSD value is used as the variance estimate, which also belongs to the measurements with the highest average velocity.

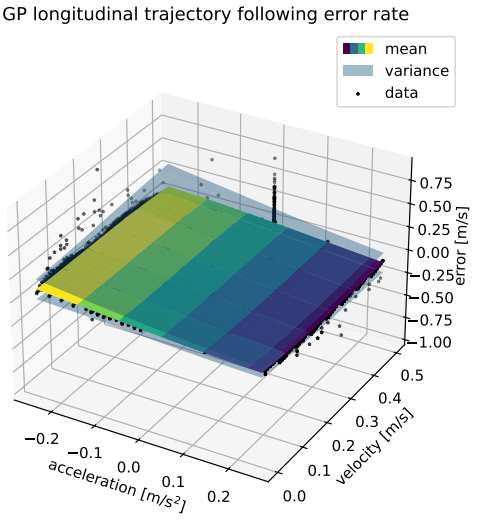


Fig. 23: GP that models the longitudinal error rate distribution as a function of the velocity and acceleration.

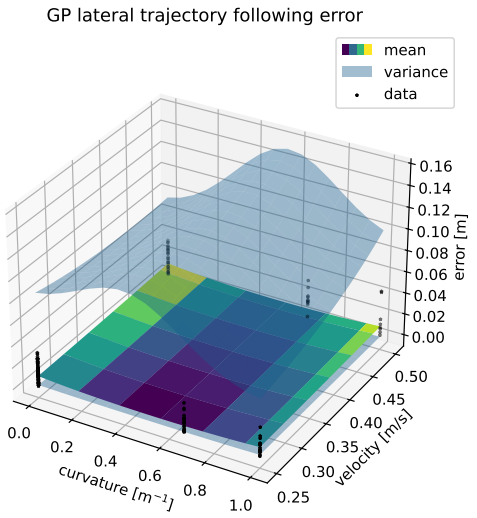


Fig. 24: GP that models the lateral error distribution as a function of the velocity and curvature.

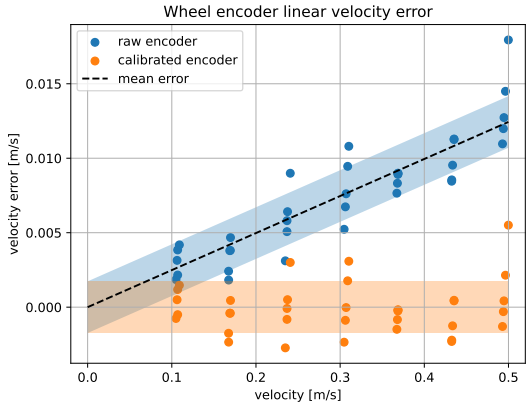


Fig. 26: Linear velocity errors and the fitted error distribution

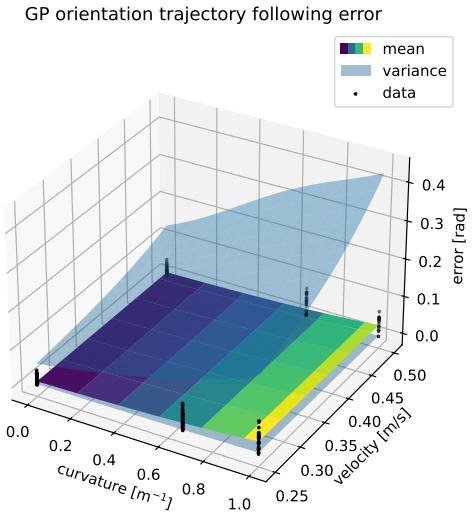


Fig. 25: GP that models the orientation error distribution as a function of the velocity and curvature.

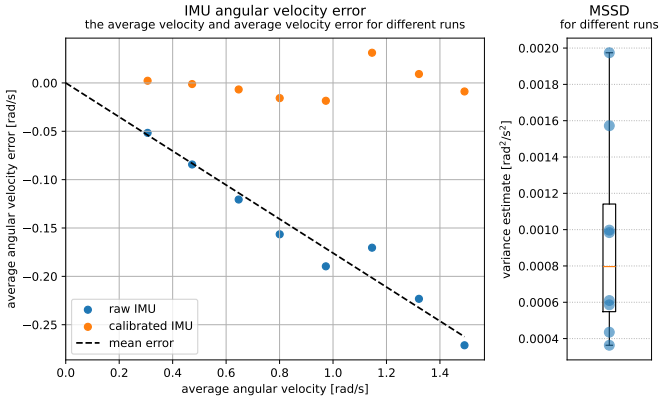


Fig. 27: Angular velocity errors and fitted calibration factor. The MSSD estimate for each run is also visualized.

E. State estimation

1) EKF noise covariance matrices: The values for the noise variances are given in Tab. VI for both EKFs. These are the values on the diagonal of the noise covariance matrices $\mathbf{Q}$.

TABLE VI: Process noise variances for both EKFs.

EKF	State variable							
	x	y	ψ	$\dot{x}$	$\dot{y}$	$\dot{\psi}$	$\ddot{x}$	$\ddot{y}$
Velocity EKF	n/a	n/a	n/a	0.01	0	0.1	0.1	0.1
Position EKF	0.01	0.01	0.01	0.01	0	0.05	0.01	0.01

2) Error introduced by uncertain SLAM pose: Fig. 28 visualizes the effect of using the uncertain SLAM pose to calculate the error. If both the EKF and SLAM have the true position at the boundary of the 68% confidence interval, the confidence interval of the EKF is often overestimated, i.e., higher than 68% for almost all positions. Only if the EKF and SLAM position deviate in approximately the same direction, the error will be underestimated. This effect is stronger when the SLAM position uncertainty is relatively big compared to the uncertainty of the EKF position.

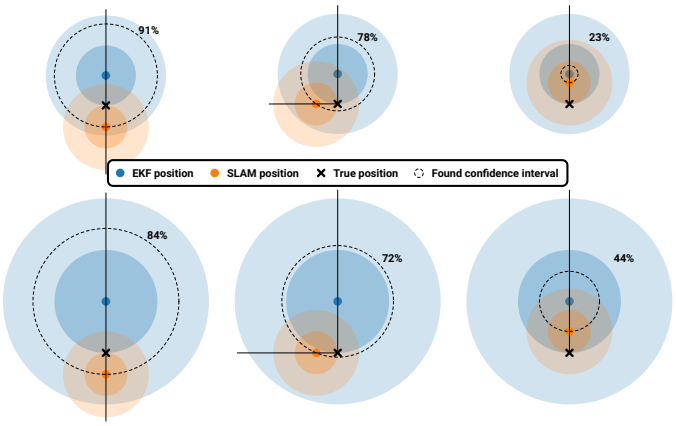


Fig. 28: The effect of the SLAM position uncertainty on the estimation of the EKF position accuracy. If both have the true position at the boundary of the 68% confidence interval, the confidence interval of the EKF is often overestimated, i.e., higher than 68%.

APPENDIX B
INNER WORKINGS OF THE FORESEE++ ALGORITHM

A. FOV construction

Adding a part of the previous scan to create a 360° view of the environment still leaves a small part of the environment unobserved. Fig. 29 shows that the combination of moving forward and steering results in a lateral movement. The size of the lateral movement is the width of the region not observed, referred to as ϵ_{merge}.

A single scan takes 125 ms to complete. The maximum linear velocity is 0.5 m/s and the minimum corner radius is 0.25 meters, which is a setting of the trajectory follower. So during one scan, the robot can travel 14 degrees and 0.0625 meters along this minimum corner radius. This means that the maximum size of the error ϵ_{merge} is 7.4 mm. For comparison, the resolution of the Lidar at 5 meters is 2.9 cm.

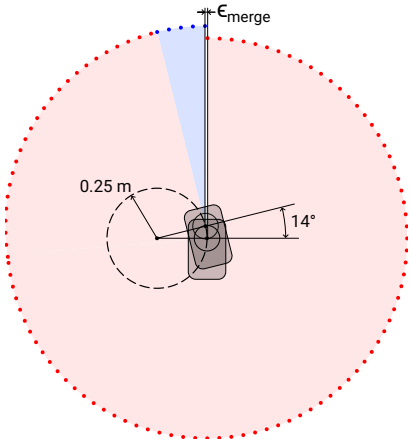


Fig. 29: ϵ_{merge} is the width of the area that is not included when merging two scans.

B. Trajectory planning

The longitudinal trajectory error rate needs to be integrated over time to obtain the longitudinal error, which is done numerically as shown in equation 4. Here is $\dot{\epsilon}_{long,i}$ the longitudinal error rate distribution and $\epsilon_{long,i}$ the longitudinal error distribution. The first state on the trajectory is $i = 0$, meaning that the rate in the first state is not considered in the integration, since the backward Euler method is used. Δt is the planning time step of the trajectory.

$$\begin{aligned} \dot{\epsilon}_{long,i} &= N(\mu_{rate,i},\ \sigma^2_{rate,i}) \\ \epsilon_{long,i} &= N(\sum_{j=1}^{i} \mu_{rate,j} \cdot \Delta t,\ \sum_{j=1}^{i} \sigma^2_{rate,j} \cdot \Delta t^2) \end{aligned} \tag{4}$$

However, it is also important to consider the difference between Δt and the time interval between the longitudinal error rate measurements used for the variance calculation, since a longer time interval results in a lower variance. The formula to account for this difference and to correct the variance estimate accordingly, is given in Eq. 5.

The formula is derived from the fact that different time steps and therefore a different number of integration steps should result in the same variance for the integrated error. Herein are Δt_{rate} and Δt_{traj} respectively the time step at which the data is recorded and the time step of the trajectory planner. σ^2_{rate} is the variance predicted by the model and σ^2_{traj} is the variance to be calculated that is valid for the integration with Δt_{traj}. n_{rate} and n_{traj} are the number of integration steps and since the total time is fixed, the following equality holds: $n_{rate} \cdot \Delta t_{rate} = n_{traj} \cdot \Delta t_{traj}$.

$$\begin{aligned} \sigma^2_{integral} = \sum_{i=1}^{n_{traj}} \sigma^2_{traj,i} \cdot \Delta t^2_{traj} &= \sum_{i=1}^{n_{rate}} \sigma^2_{rate,i} \cdot \Delta t^2_{rate} \\ n_{traj} \cdot \sigma^2_{traj} \cdot \Delta t^2_{traj} &= n_{rate} \cdot \sigma^2_{rate} \cdot \Delta t^2_{rate} \\ \sigma^2_{traj} \cdot \Delta t_{traj} &= \sigma^2_{rate} \cdot \Delta t_{rate} \\ \sigma^2_{traj} &= \frac{\Delta t_{rate}}{\Delta t_{traj}} \cdot \sigma^2_{rate} \end{aligned} \tag{5}$$

Provided with this information, equation 4 should be rewritten to account the time step difference (Eq. 6).

$$\epsilon_{long,i} = N(\sum_{j=1}^{i} \mu_{rate,j} \cdot \Delta t_{traj},\ \sum_{j=1}^{i} \frac{\Delta t_{rate}}{\Delta t_{traj}} \cdot \sigma^2_{rate,j} \cdot \Delta t^2_{traj}) \tag{6}$$

APPENDIX C
EXPERIMENTS

The mobile robot used in the experiments (Fig. 2) is differential-driven with a Raspberry Pi 4 with an Adafruit motor HAT to control the two 7.4V powered motors. The sensor suite consists of wheel encoders, a LSM9DS1 IMU (Inertial Measurement Unit) and a 2D Slamtec RPLIDAR A1. A laptop is used to do the computations for `foresee++` and SLAM, and the communication happens over the local Wi-Fi network. All the software is implemented in ROS 2. The ROS package `robot_localization` is used for the EKFs, and the package `slam_toolbox` is used for the SLAM.

A. Performance experiments

1) Significance test of traversal time improvements: A T-test is performed to determine if the traversal time improvements per scenario (Fig. 14) between `foresee` and `baseline` and between `foresee++` and `baseline++` are statistical significant. A T-test simply compares two distributions and indicates if the difference between the means is significant, i.e., a significant improvement. The threshold for rejecting the null hypothesis is 0.05.

The traversal time difference between `baseline` and `foresee` is significant for all scenarios except scenario 1 according to the T-test results because it has such a high variance. This is because `foresee` can find a gap in one of the runs, leading to a big difference in the traversal time. The distribution of the traversal times is therefore not Gaussian, which violates the assumption of the T-test. Replacing this value with the worst traversal time for `foresee` on this scenario, does result in a significant difference according to the T-test. It is therefore safe to say that the improvement for the algorithms without error measures is statistically significant for all scenarios.

When it comes to the algorithms with error measures, scenarios 2, 3 and 6 show a significant improvement in the traversal time. The results of the T-tests are shown in Tab. VII.

2) Significance test of traversal time difference for the error measures: To assess if adding error measures results in a significant increase in the traversal time, the algorithms with reasoning, i.e., `foresee` and `foresee++` and the algorithms without, i.e., `baseline` and `baseline++` are compared. The result of the two-way ANOVA for `foresee` and `foresee++` is $F(1) = 64.21$, $p < 0.05$ and for `baseline` and `baseline++` $F(1) = 537.7$, $p < 0.05$. This clearly indicates a significant increase in the traversal time for the error measures.

TABLE VII: T-test results for the normal traffic scenarios in the performance experiments

scenario	t-value	degrees of freedom	p-value
baseline vs. foresee			
1	2.295	0.148	2.009
1 adjusted	12.248	0.001	2.972
2	11.941	0.006	2.083
3	12.558	0.000	4.000
4	5.181	0.035	2.003
5	3.078	0.041	3.684
6	8.773	0.002	3.367
baseline++ vs. foresee++			
1	-1.267	0.328	2.101
2	11.917	0.001	3.492
3	5.999	0.024	2.090
4	-0.241	0.828	2.504
5	-1.392	0.272	2.561
6	10.191	0.001	3.392

B. Ablation study

As described in section VII-B, Tukey's HSD was chosen as the multi comparison test (MCT). Where an ANOVA can only tell that a certain quantity (e.g., algorithm type) exhibits a significant correlation with the results, an MCT also makes pairwise comparisons among all groups within this quantity (i.e., between each pair of two different algorithms).

The traversal time improvements are compared to the worst-case performance of `foresee++` for each scenario. These are used to normalize the results for the different scenarios and allow comparing the results for multiple scenarios. The Tukey's HSD test results indicate a statistical significant difference between `foresee abl. Lidar` and `FOV` on the one hand and `foresee abl. delay` and `traj` on the other hand, where the former are less cautious. `Foresee abl. local` did not have a significant difference with any of the other algorithms, which aligns with the visually observed result that its performance is somewhere in between the two aforementioned groups of algorithms (Fig. 30). The results of the test are given in Table VIII.

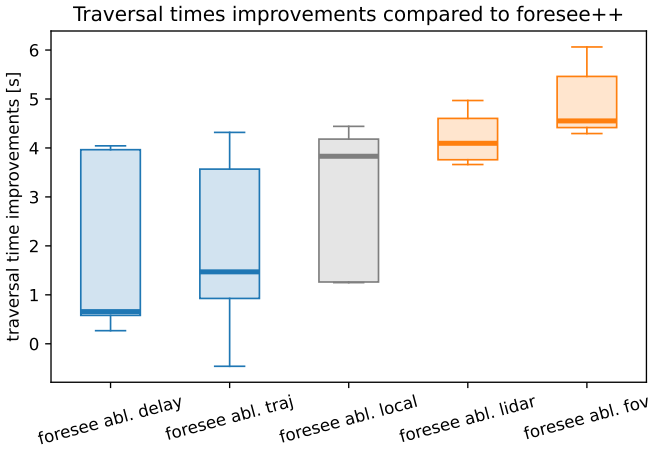


Fig. 30: Distribution of the traversal time improvements for the different versions of the ablated algorithms. The difference between the algorithms in the blue and orange groups is statistical significant, according to Tukey's HSD test.

TABLE VIII: Tukey's HSD test results for the ablated algorithms on the normal traffic scenarios.

Compared `foresee++` versions		mean difference	adjusted p-value	is significant
`abl. delay`	`abl. fov`	2.848	0.0004	X
`abl. delay`	`abl. Lidar`	2.165	0.0099	X
`abl. delay`	`abl. local`	1.109	0.3955	
`abl. delay`	`abl. traj`	-0.098	0.9999	
`abl. fov`	`abl. Lidar`	-0.683	0.8051	
`abl. fov`	`abl. local`	-1.740	0.0565	
`abl. fov`	`abl. traj`	-2.946	0.0002	X
`abl. Lidar`	`abl. local`	-1.057	0.4436	
`abl. Lidar`	`abl. traj`	-2.263	0.0064	X
`abl. local`	`abl. traj`	-1.206	0.3119	