Online System Identification and Calibration of Dynamic Models for Autonomous Ground Vehicles

Sina Aghli and Christoffer Heckman*

***Abstract*— This paper is concerned with system identification and the calibration of parameters of dynamic models used in different robotic platforms. A constant time algorithm has been developed in order to automatically calibrate the parameters of a high-fidelity dynamical model for a robotic platform. The presented method is capable of choosing informative motion segments in order to calibrate model parameters in real time while also calculating a confidence level on each estimated parameter. Simulations and experiments with a $\frac{1}{8}^{\text{th}}$ scale four wheel drive vehicle are performed which demonstrate the accuracy and efficiency of the approach.**

I. INTRODUCTION

Dynamical models are used in planning, control and state estimation of robotic platforms; however, having a model which well describes a real physical platform. Despite the amount of effort one might put into designing a high-fidelity dynamical model, there exists yet another challenge that arises: such models typically have a bogglingly large number of parameters which must be tuned very precisely. For instance, when operating a ground vehicle around a turn at high speed, a small inaccuracy in the coefficient of friction could send the vehicle into slip through a discontinuous change in dynamics. Such behavior would be deleterious and potentially hazardous unless it were predictable, in which case it might be used to an advantage in e.g. guiding the vehicle through a tight corner. Even if an accurate set of parameters are chosen via expert tuning, some of these parameters would inevitably change in time due to mechanical degradation or environmental disturbances.

Even more complicating to the scenario is that only some of these parameters are perceptible in driving unless an obscure set of conditions are met. For instance, in a ground vehicle the static coefficient of friction between tires and the ground is only measureable at the point of slip; the dynamic coefficient is only measureable *when slipping,* which is potentially even more challening. Also, it is not immediately obvious which types of motions a vehicle platform might undergo that allow a human operator to determine the suspension stiffness. No one motion will allow all of these parameters to be observed, thereby creating a quandary of how to operate a vehicle such that all relevant physical parameters might be known to some degree of certainty.

And yet despite these difficulties the goal of grounding a system's representation in concrete parameters that represent physical relationships remains highly desirable. Were the estimation of these physical parameters to be robust and reflexive for different operating scenarios, it is possible that model-predictive control could be applied in contexts that are currently challenging, such as when dynamic parameters are changing in time or are unknown before plant operation. Furthermore, these parameters might be estimated through measurements through external sensors, e.g. cameras or infrared, providing another mechanism to directly influence controllers built on these models. Finally, reasoning about such systems can be transparent and tractable from a nearly intuitive standpoint.

To address this, the present work develops a probabilistic calibration method which may be used to estimate parameters of a dynamical model of a ground vehicle.

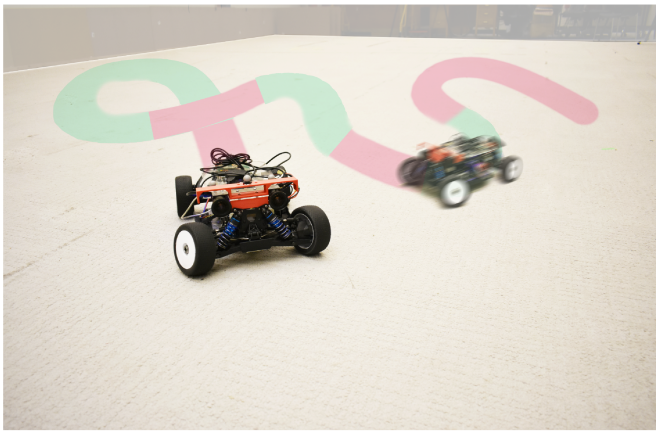

Fig. 1. Calibrating wheel friction coefficient and wheel base of model four wheel drive car.

II. RELATED WORK

The parameter estimation problem has been well-studied on robotic platforms, traditionally aimed at particular parameters on certain experimental platforms; these efforts have largely grown out of a need to estimate parameters that cannot be directly measured, such as the coefficient of friction for wheels or the extrinsic transformation between an IMU sensor and image sensor of a camera. Another possibility is that such parameters are not truly static and change during operation. For example in the case of ground vehicles, GPS measurements have been used in order to estimate vehicle mass [1], roll stiffness and damping ratio [2] or tire/road friction coefficient [3]. In the case of quadrotor UAVs, IMU measurements have been used to estimate the body moment of inertia [4]. For other platforms, the 6-DOF

This work was supported by DARPA award no. N65236–16–1–1000.

Both authors are with the University of Colorado, Boulder Autonomous Robotics and Perception Group.

* Corresponding author; e-mail: christoffer.heckman at colorado.edu

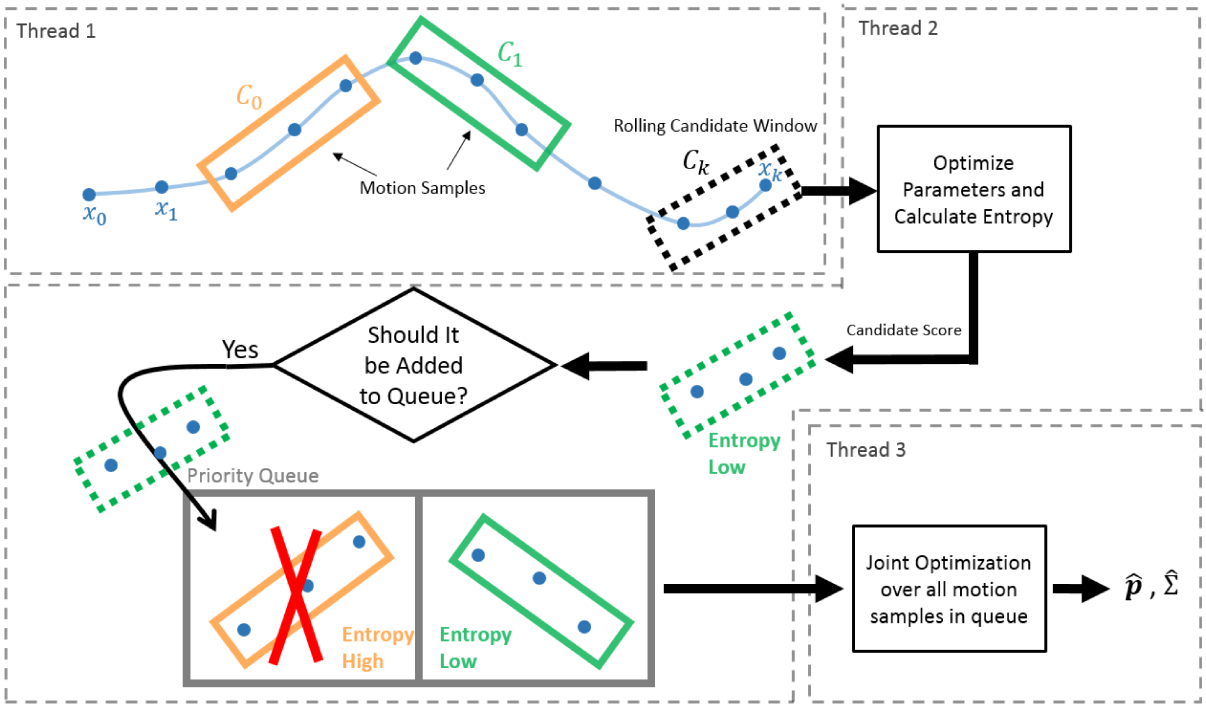

Fig. 2. A flow chart overview of the method. First, a sequence of state information known as a "motion sample" is captured within a candidate window, which is used to estimate calibration parameters. Next, the motion sample is compared to a priority queue of motion samples. The comparison operator in this priority queue determines if the sample in the candidate window reduces the entropy over the whole calibration parameter set while maintaining a balance of informative motion segments across all calibration parameters. If the motion sample is comparatively better than any within the priority queue, the candidate window is augmented to the priority queue and the lowest-scoring window in the queue is purged.

pose measurements have been leveraged to estimate inertial parameters of a grasped object [5] or thrust factor of actuators [6]. Geometric/kinematic parameters have also been estimated for different platforms, like linkage length in a robotic arm [7], [8], as well as distances and angles between a car chassis coordinate frame and its tire frames [9]. More recently in the robotics community, probabilistic approaches have been developed in order to estimate camera/IMU/laser scanner extrinsic parameters [10], [11], [12].

The calibration task, frequently known as "system identification" for physical platforms, generally involves estimating some parameters of the model from some indirect sensory measurements, however the choice of sensors and their attachment location on an autonomous platform is very important for achieving accurate calibration estimates, since only some specific motions of robot would render parameters observable. Regressing parameters on data which does not render parameters of interest observable generally leads to wildly inaccurate results or a lack of convergence. One approach previously taken is to attach a sensor in a specific location which properly excites its output signal; however, finding the ideal location might be very hard if not impossible [13], [14]. Two ways to avoid calibration a platform using a dataset which does not make parameters observable is to restrict motions of robot to some pre-planned trajectories which have good excitation properties [9], [15] or to algorithmically choose the most informative motion segments [12], [10], [11] from a platforms trajectory. In these approaches, one must be careful in choosing informative segments of motion since an uneven amount of information could bias the results toward selecting segments only for parameters with more informative segments without balancing this with accuracy over the entire parameter set. To avoid this problem, techniques like normalization of the posterior estimate has been proposed [16], but this has been exclusively applied to the SLAM problem.

III. Methodology

In this work we develop a self-calibration approach to the estimation of parameters for a dynamic physical model of a ground platform using as input the state estimates to the vehicle. The approach develops an algorithm for choosing informative motion segments for all calibration parameters, and is extensible to intrinsic and extrinsic parameters (e.g. tire friction coefficients and wheel base) for the vehicle.

In general, dynamic models of robotic platforms are developed under the assumption that the platform operate only in the same regimes as those in which they are calibrated. This work represents a step toward loosening this highly constraining assumption, allowing for such parameters to potentially change in a dynamic environment and for the system to estimate these new parameters in real-time. Furthermore, the procedure developed allows for the estimation of a high-fidelity (and high-dimensional) model which, from a principled standpoint, has proven to be quite elusive. Many approaches for example assume a four-wheeled vehicle is

reducible to a bicycle model [17], however the assumptions implicit in such a model are very constraining: a vehicle must drive on a flat surface, with tires always in contact with the ground and without any roll. Many vehicles in normal operation, and especially in challenging off-road conditions, do not exhibit such behavior.

We now will provide a high-level explanation of the algorithm, accompanied by a flow chart in Figure 2. At the core of the method, and the key innovation in this work, is the evaluation of discrete-segment motion samples by assigning a "score" that quantifies the sample's informativeness toward estimating the parameters. The collection of these motion samples in a priority queue are used to jointly estimate calibration parameters online. Since both the candidate window and priority queue have fixed sizes, the calibration parameter estimates are produced in constant time. Our approach is also highly parallelizable, with the consumption, candidate window and priority queue optimization operations each running in separate threads. Note that the parallelization of this algorithm is shown using dotted windows for each part of algorithm in Figure 2.

A. Dynamics Model

First we suppose that we have a model of the dynamics of the platform given as:

$$\boldsymbol{x}'(t) = \boldsymbol{\Phi}(\boldsymbol{x}(t), F(t), \boldsymbol{u}(t), \boldsymbol{p}), \tag{1}$$

in which $\boldsymbol{\Phi}(\cdot)$ represents the dynamical model, $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state of the system, $\boldsymbol{x}'(t)$ is the derivative of the state under the governing dynamics, f consolidate external forces applied to the system, $\boldsymbol{u}(t) \in \mathbb{R}^m$ is a control input (for vehicles, generally the steering and acceleration), and $\boldsymbol{p} \in \mathbb{R}^w$ is model parameter vector to be calibrated. Distinctions between the system's "real dynamics" and modeled ones are ignored with the restriction that modeling assumptions are not violated when operating the platform. For example, if a physical model is utilized which restricts the vehicle to not be airborne, it is assumed the vehicle will never go airborne. This model is discretized in time, such that

$$\boldsymbol{x}_i = \left[x_i^1, \ldots, x_i^n\right], \tag{2}$$

is the state of the platform at time step k.

For a simple car model, a calibration parameter vector might include vehicle properties like tire and chassis geometry; more complicated models might consider inertial properties, linear and rolling friction coefficients of all bodies, suspension anchor points, suspension damping and stiffness coefficients, motor speed-torque parameters for steering and acceleration motors and a chassis's center of mass location.

Strictly speaking, the generality of the method we described may be extended to non-ground vehicle platforms; however, we will describe this method as we have developed it for such systems. The state $\boldsymbol{x}_i$ includes the $\mathbb{SE}(3)$ pose of the wheels and chassis, as well as the current steering wheel angle, wheel speeds. Many of these quantities may be directly estimated through, e.g. modern SLAM algorithms or properly placed sensors like linear or angular encoders.

The control input $\boldsymbol{u}(t)$ is discretized by sampling the continuous input signal over time intervals $\Delta t_k = t_k - t_{k-1}$ which need not be uniform. Through this, it is assumed that the control input is constant within the interval $[t_k, t_k + \Delta t_k]$. Current control inputs, system state and the timestamp of the system are sampled and stored as a triplet $\boldsymbol{s}_k = [\boldsymbol{x}_k, \boldsymbol{u}_k, t_k]$ which compose the motion samples at timestep k.

B. Rolling Candidate Window

In order to score motion segments based on how much information they contribute to the calibration parameter estimation problem, a rolling window of size n_c is used. The rolling candidate window $\boldsymbol{C}_k = [\boldsymbol{s}_{k-n_c}, \ldots, \boldsymbol{s}_k]$ is composed of the n_c latest motion samples. Within $\boldsymbol{C}_k$ a score of informativeness is calculated. The score is calculated by estimating parameters $\boldsymbol{p}$ and calculating a score based on the entropy of the posterior. Since the entropy is a way to quantify uncertainty, lower scores represent more desirable motion samples contained within the window.

For a given candidate window $\boldsymbol{C}_k$, a cost function is constructed as follows:

1) Initialize Eq. (1) at a state/input pair given at the beginning of the window as the canonical estimate of that state; i.e.,

$$\hat{\boldsymbol{s}}_{k-n_c} = \boldsymbol{s}_{k-n_c}. \tag{3}$$

2) Integrate Eq. (1) forward with the measured control inputs over the time intervals stored in $\boldsymbol{C}_k$ to get $\hat{\boldsymbol{C}}_k$:

$$\hat{\boldsymbol{C}}_k = \left[\hat{\boldsymbol{s}}_{k-n_c}, \ldots, \hat{\boldsymbol{s}}_k\right]. \tag{4}$$

3) Construct a cost function:

$$\boldsymbol{\Psi}_{\boldsymbol{C}_k} = \boldsymbol{C}_k\left[\boldsymbol{x}\right] \boxminus \hat{\boldsymbol{C}}_k\left[\hat{\boldsymbol{x}}\right], \tag{5}$$

where $\boldsymbol{C}\left[\boldsymbol{x}\right]$ represents only the vector of states in the corresponding window, and operator $\boxminus$ calculates a weighted error between two state vectors.

The cost function $\boldsymbol{\Psi}_{\boldsymbol{C}_k}$ is later minimized in order to calculate the covariance of the posterior $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{C}_k}$ and a score for current window. The candidate window $\boldsymbol{C}_k$ is a rolling window, meaning that when a new measurement arrives, $\boldsymbol{s}_{k-n_c}$ is removed and the new measurement is pushed back to the window; a new score is then calculated.

C. Optimization

Calibration parameters are estimated in a nonlinear maximum likelihood estimation framework. The joint probability distribution of parameter $\boldsymbol{p}$ given state measurements $\boldsymbol{s}_k$ in window $\boldsymbol{C}_k$ and dynamics provided by Eq. (1) is:

$$P(\boldsymbol{p}, \boldsymbol{Q}_k[\boldsymbol{x}]) = P(\boldsymbol{Q}_k[\boldsymbol{x}]|\boldsymbol{p})P(\boldsymbol{p}) = \prod_{i=1}^{n_q} P(\boldsymbol{C}^i[\boldsymbol{x}]|\boldsymbol{p}), \tag{6}$$

where $\boldsymbol{Q}_k = [\boldsymbol{C}^0, \ldots, \boldsymbol{C}^i]$, $i \in \{0, \ldots, n_Q\}$ represents all candidate windows in the queue at time step k. In

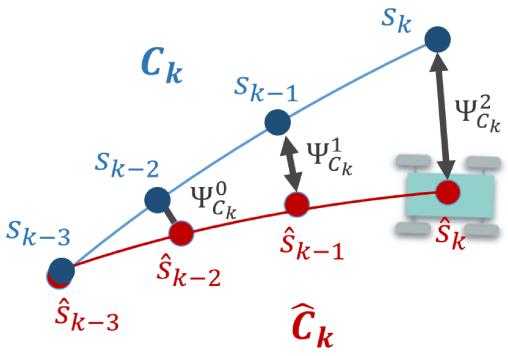

Fig. 3. Elements of cost function before candidate window optimization. For a given motion segment $\boldsymbol{C}_k$ (blue dots) in candidate window for a window of size $n_c = 4$ and an integrated path $\hat{\boldsymbol{C}}_k$ with initial parameter values (red dots), a corresponding cost function $\boldsymbol{\Psi}_{\boldsymbol{C}_k}$ is constructed.

Eq. (6), the likelihood term $P(\boldsymbol{Q}_k[\boldsymbol{x}]|\boldsymbol{p})$ is factored since different candidate windows are selected such that they are independent of each other (see Section III-D). Note that the prior term $P(\boldsymbol{p})$ can be dropped since the priority queue will carry the prior of the estimate.

Noting that the prior term in Eq. (6) vanishes since the first state variable is always fixed to the first measurement (see Eq. (3)), hence we have:

$$P(\boldsymbol{p}, \boldsymbol{C}_k[\boldsymbol{x}]) = P(\boldsymbol{C}_k[\boldsymbol{x}]|\boldsymbol{p}). \tag{7}$$

An optimal estimate $\hat{\boldsymbol{p}}$ for the parameter vector $\boldsymbol{p}$ can be calculated by minimizing the joint probability:

$$\hat{\boldsymbol{p}} = \underset{p}{\operatorname{argmax}}\, P(\boldsymbol{p}, \boldsymbol{C}(\cdot)[\boldsymbol{x}]). \tag{8}$$

According to Eq. (6), the solution to this estimation problem can also be achieved by maximizing the likelihood term. Assuming that parameter noise is Gaussian distributed, we can write probability density function of each likelihood term in Eq. (6) as:

$$\begin{aligned} P(\boldsymbol{x}_{\boldsymbol{C}_i}|\boldsymbol{p}) &\propto \exp(-\frac{1}{2}\|\boldsymbol{C}_i[\boldsymbol{x}] \boxminus \Phi(\boldsymbol{C}_i, \cdot, p)\|^2_\Sigma) \\ &\propto \exp(-\frac{1}{2}\|\boldsymbol{C}_i[\boldsymbol{x}] \boxminus \hat{\boldsymbol{C}}_i[\boldsymbol{x}]\|^2_\Sigma) \\ &\propto \exp(-\frac{1}{2}\|\boldsymbol{\Psi}_{\boldsymbol{C}_i}\|^2_\Sigma), \end{aligned} \tag{9}$$

in which $||\cdot||^2_\Sigma$ signifies squared Mahalanobis distance given measurement uncertainty $\boldsymbol{\Sigma}$ and $\boldsymbol{C}_i[\boldsymbol{x}]$ for the i^{th} candidate window. In the case of a rolling candidate window, the equation above will only have one likelihood term as mentioned at Eq. (7). then Eq. (8) is solved by iteratively updating parameter vector $\boldsymbol{p}$ with a trust region method [18], [19]. Given the Gaussian distribution assumption, the covariance of the posterior is computed over calibration parameters by inverting the Fisher information matrix:

$$\hat{\boldsymbol{\Sigma}}(\hat{\boldsymbol{p}}) = Covariance(\hat{\boldsymbol{p}}) = (\boldsymbol{J}^\intercal(\hat{\boldsymbol{p}})\, \boldsymbol{J}(\hat{\boldsymbol{p}}))^{-1}. \tag{10}$$

Here $\boldsymbol{J}(\hat{\boldsymbol{p}})$ is Jacobian of $\boldsymbol{\Psi}_{\boldsymbol{C}_i}$ at $\hat{\boldsymbol{p}}$. The Jacobian is checked to make sure it is full rank before calculating the covariance; if it is not of full rank, then the current motion sample is ignored due to lacking observability in all calibration parameters.

D. Entropy Score Board

Solving the MLE problem within a candidate window completes results in the covariance of the estimate $\hat{\boldsymbol{\Sigma}}(\cdot)$, which is used to calculate a score vector for the current motion sample. Since the eigenvalues of the covariance matrix represent uncertainty in each parameter direction, an eigen-decomposition of $\hat{\boldsymbol{\Sigma}}(\cdot)$ is calculated and stored as an entropy value for current window. Recall that a lower entropy implies a better candidate. We designate $\boldsymbol{\kappa}_k$ as the vector of eigenvalues of the matrix $\hat{\boldsymbol{\Sigma}}$ for the candidate window $\boldsymbol{C}_k$.

One of the primary considerations is that we would contain the number of motion samples we are tracking at any given time; this was part of our comparison operator design, and was found to aid robustness in collecting salient and informative segments for all parameters. To do this, we introduce the notion of a score board which is part of our comparison as follows:

1) The maximal entropy for each parameter from all motion samples in the priority queue is stored in a vector $\boldsymbol{\lambda} \in \mathbb{R}^w$ (since each element of λ corresponds to an element of $\boldsymbol{p}$).
2) A $w \times n_Q$ table $\mathbf{T}$ is constructed to keeps track of which candidate windows are informative for which parameters, where $n_Q = n_s \times w$ is size of priority queue and n_s is the number of samples per parameter desired. $\mathbf{T} \in \mathbb{B}^{w \times n_Q}$ contains boolean values in its entries, where an entry in index $\mathbf{T}[i, j]$ shows that candidate segment in index j of priority queue is informative for parameter i.
3) A vector $\boldsymbol{\tau} \in \mathbb{R}^w$ keeps track of the total number of informative segments for each parameter which currently exist in priority queue by summing over the rows of $\mathbf{T}$:

$$\boldsymbol{\tau}_i = \sum_{j=0}^{n_Q} \mathbf{T}[i, j]. \tag{11}$$

For each newly calculated $\boldsymbol{\kappa}_k$, we compare corresponding indexes in this vector to current worst entropy $\boldsymbol{\lambda}$ according to Algorithm 1.

Note in Algorithm 1, n_M is a tuning parameter representing the minimum acceptable score, and $\alpha = 0.95$ is a safety margin to ensure at least a 5% reduction on maximum entropy occurs when adding a motion sample to the priority queue. A candidate window might overlap with previous candidates in queue which would result in double counting the information in the queue and over-fitting the results; to avoid this, the current candidate window is checked for such

Data: $\boldsymbol{C}_k, \boldsymbol{\kappa}_k, n_{\boldsymbol{p}}, \boldsymbol{\lambda}, n_M, \alpha$
Output: $update\_condition, current\_score$
$current\_score \leftarrow 0$;
for $i \in \{0, \ldots, n_{\boldsymbol{p}} - 1\}$ **do**
  if $\boldsymbol{\kappa}_k[i] < \alpha \times \boldsymbol{\lambda}[i]$ **then**
    $current\_score \leftarrow current\_score + 1$;
  end
end
if $current\_score \geq n_M$ **then**
  $update\_condition \leftarrow true$;
else
  $update\_condition \leftarrow false$;
end

Algorithm 1: Priority queue update condition

a condition. If there is overlap, then the current candidate is compared with the sample in the priority queue with an overlap as in Alg. 1 and replaces the entry in the queue if the resulting score is higher.

In the case that the update condition in Alg. 1 is satisfied and there are not any common segments between the current candidate window and any candidates in the priority queue, then a final condition is checked to make sure the amount of data entered is not biased toward a group of parameters. In this check, the new candidate is added to priority queue and corresponding entries of $\mathbf{T}$ which reduce entropy of parameter i are marked with 1s and corresponding $\boldsymbol{\tau_i}$ and $\boldsymbol{\lambda_i}$ values are updated. If adding a new 1 to the table falsifies the queue limit inequality $\boldsymbol{\tau}_i \leq n_s$ on any parameters, then the table entry which previously had the highest entropy in vector $\boldsymbol{\lambda}$ is changed to a 0. Figure 4 shows a flow chart for this procedure.

To initialize this process, the very first candidate is accepted without any checks and later candidates are checked for having an overlapping window or if they have a lower entropy score but no replacements (with higher entropy candidates) takes place until the queue is full. Once a decision is made a pruning step takes effect by removing columns of table T which have all zeros and also removing corresponding candidate window from the queue.

E. Priority Queue

The priority queue $\boldsymbol{Q}_k = [\boldsymbol{C}^0, \ldots, \boldsymbol{C}^i]$, $i \in \{0, \ldots, n_Q\}$ is responsible for storing the set of candidate windows which have been selected using the criteria described in Section III-D. At every update to this queue we construct a cost function and jointly optimize over all candidate windows in the queue. from Eq. (5) we can write

$$\boldsymbol{\Psi}_{Q_k} = \sum_{i=0}^{n_{Q_k}} (\boldsymbol{\Psi}_{C^i}), \tag{12}$$

Then cost function $\boldsymbol{\Psi}_{Q_k}$ is passed to the optimization pipeline in section III-C to find a new estimate $\hat{\boldsymbol{p}}_{\boldsymbol{P}}$ as well as its corresponding covariance $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{P}}$. The estimated covariance is then used to asses the confidence level on each estimated parameter.

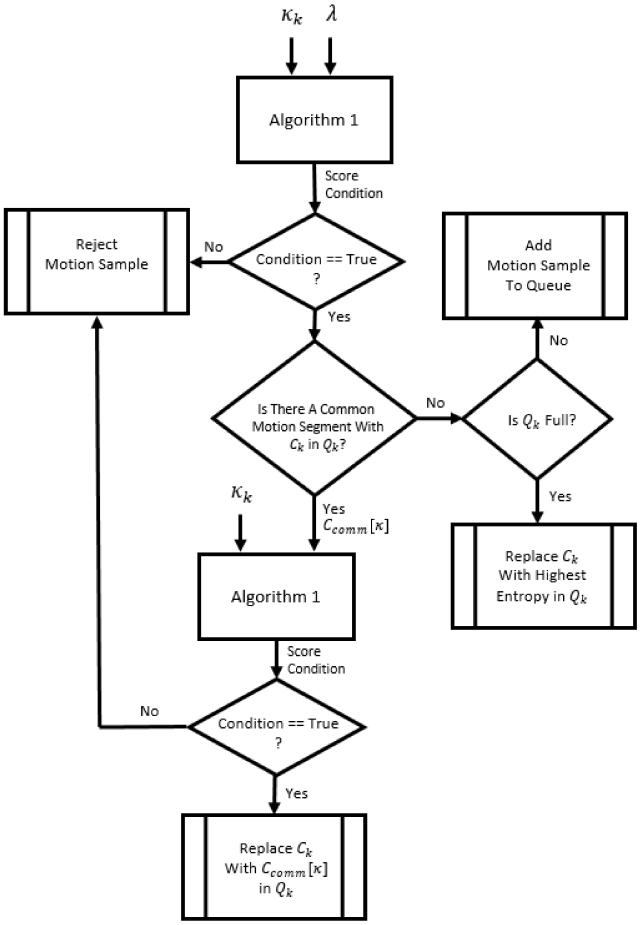

Fig. 4. Flowchart for deciding if a motion sample should be added to the priority queue or not. In this chart $\boldsymbol{C}_{comm}[\boldsymbol{\kappa}]$ is the motion sample in queue which has overlapping segment with candidate in current rolling window.

IV. Experiments

In order to validate this algorithm, we have performed simulations on a model four wheel drive vehicle as well as experiments on a modified $\frac{1}{8}^{\text{th}}$-scale vehicle platform. The method as described has been implemented in the C++ language and tested in both simulation and experiment, and has reliably performed in real-time on an Intel Core i7. As shown in Figure 2 with dashed lines, the pipeline consists of three main threads. The first thread is responsible for capturing motion samples in discrete time and constructing a rolling candidate windows as described in Section III-B. When this task is complete, this thread signals to a second thread that the motion sample is ready for analysis. This second thread then minimizes the cost $\boldsymbol{\Psi}_{\boldsymbol{c}_k}$ using the optimization technique explained at Section III-C. In addition to assigning a score to a given motion sample, this thread also handles the decision making explained in Section III-D and updating $\boldsymbol{Q}_k$. This thread then provides a synchronization token to a third thread which calculates and updates the cost $\boldsymbol{\Psi}_{Q_k}$. The optimization in the third thread results in a new parameter and covariance estimate.

As a model for experiments, a physics based dynamics

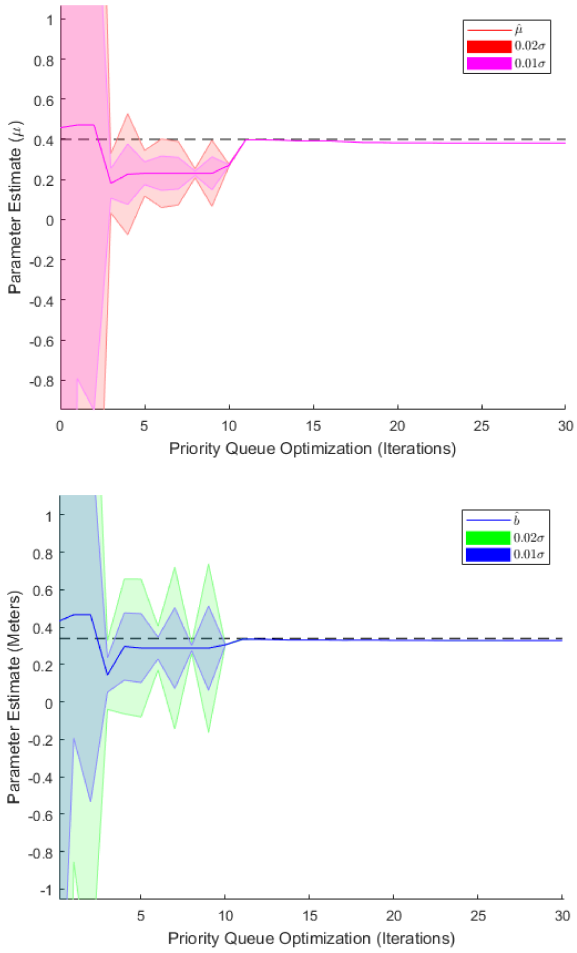

Fig. 5. Estimation results for friction coefficient (top), and wheel base (bottom) of simulated car. Solid colored line represent mean value, filled areas are scaled variances and black dashed line is ground truth value.

model using the Bullet Physics Engine [20] is constructed. To accomplish this, a high fidelity four-wheel drive vehicle model was implemented; the model includes parameters such as mass, inertia and geometry for each wheel and the chassis, dampers and springs as a suspension for each wheel and their anchor points, steering and acceleration motor speed/torque properties and speed limits and also rigid body linear/rolling friction coefficients. This model is also capable interacting with a map of the environment and modeling contact forces [21].

We will now describe the two sets of experiments we conducted in procedural detail, and present their results.

A. Simulation

For this experiment a simulated world with a flat floor and constant friction coefficient at every point was constructed. The simulated vehicle was developed to accept steering and throttle commands from a user who would manually drive the vehicle in the simulated environment using a gamepad. The state of the vehicle used in this case is:

$$\boldsymbol{x}_i = [\boldsymbol{x}_{ps}, \dot{\boldsymbol{x}}_{ps}, \boldsymbol{\omega}], \tag{13}$$

where $\boldsymbol{x}_{ps} \in \mathbb{SE}(3)$ is the pose vector of the chassis with rotations in axis-angle format, $\dot{\boldsymbol{x}}_{ps}$ is vector of linear and rotational velocities and $\boldsymbol{\omega}$ is vector of wheel velocities. In these experiments, the model for the simulated vehicle are exactly equal to those in Eq. (1) used to regress the calibration parameters, however the parameters are significantly perturbed. The target of this experiment is to identify the perturbation and regress the correct physical parameters.

In this experiment, the user first drives the simulated car in a back and forth motion in the virtual environment until a sufficient number of priority queue updates have occurred (in our case four), suggesting a reasonable estimate may have been found. The user then drives on circular trajectories in order to consider alternate motions that may influence the parameter calibration. This sequence of driving demonstrated what is intuitively expected: in the first type of motion, both the friction and wheel base estimates are very uncertain since a back and forth motion does not render these parameters observable. In the second type of motion however, and especially when introducing some obvious sliding in the driving mode, more informative segments for these calibration parameters are added to the priority queue. Through the combination of these motions, both parameters eventually converge to very close to their ground truth values with very low uncertainty.

A set of two calibration parameters are considered in this case: the car's wheel base and the ground/tire friction coefficient are simultaneously estimated. Since these parameters are known for the simulated vehicle, we may compare the resulted estimates to the ground truth values directly. Figure 5 shows the results for applying this calibration method with $n_C = 20$ and $n_Q = 10$ for about one minute with discretization steps of 100ms. In both charts, the solid line shows the mean of the estimate and the two shaded areas with different colors show variance bounds on the estimate while a black dashed line shows the ground truth value.

B. Robotic Platform Tests

We have designed and built a four-wheel drive, eighth-scale car (see Figure 1) which accepts throttle and steering commands and is capable of measuring wheel speeds, current steering angle position and suspension lengths in real time. In order to get pose and velocity measurements of chassis, experiments were performed in a large open room equipped with a motion capture system. In these trials, the vehicle has been modeled with same physics based model explained in previous section and the state vector is the same as that considered in Eq. (13). Whenever required, derivatives of state are achieved by numerically differentiating the model around current state at a given time instance. Figure 6 shows the priority queue estimates of friction coefficient and wheel base on our model vehicle. The wheel base estimate of 0.32 meters is fairly close to the measured value of 0.34, this error could be due to the differences of the physics based dynamic

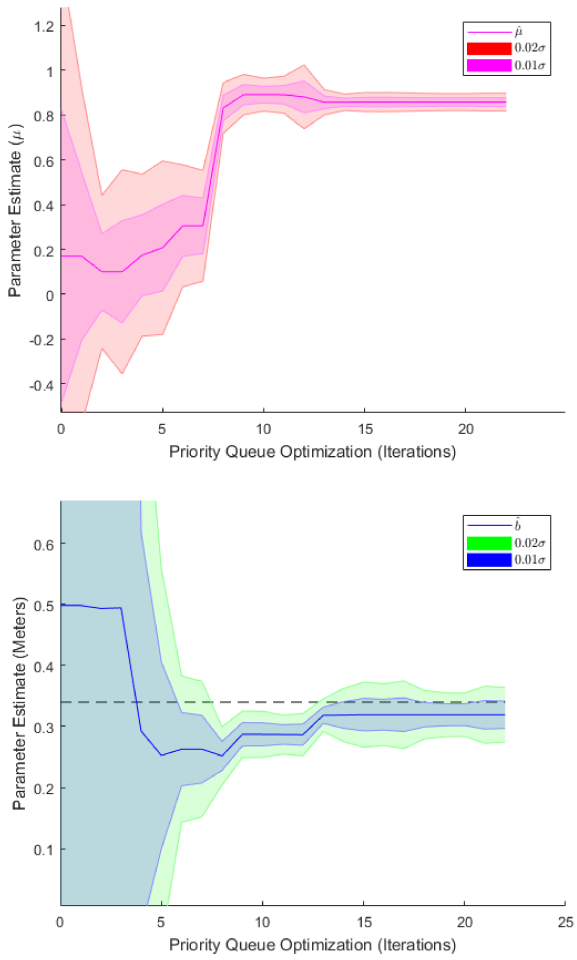

Fig. 6. results from the car

model and the actual vehicle. As one might realize, the parameter confidence after convergence on real data is low compared to simulated data, which is a result of using a less accurate model. Since comparison of the friction coefficient estimate to a ground truth value is impractical we evaluate it qualitatively. Friction coefficient is a number in the range $[0 \ldots 1]$ where zero means no friction. Since the experiment happened on a carpeted floor with treaded tires, a high value of $\hat{\mu}$ was expected and the priority queue estimate shows a similar value (0.85).

V. Conclusions

As described before parameter estimation is a crucial task for self driving vehicles in the sense that wrong parameters might result in unstable systems, which use a dynamical models as their core to predict vehicle's behavior. In this paper a new method in estimating parameters has been shown which chooses most informative motion samples of the trajectory of vehicle and estimates the parameters while avoiding any biasing toward a group of parameters with higher chance to be excited. Method has been implemented and tested both in simulation and physical experiments. Both, results show usefulness of this method while giving a confidence level on the current estimate. We believe this method is not limited only to AGV's and in future more experiments with other robotic platforms will be done.

References

[1] H. S. Bae, J. Ryu, and J. C. Gerdes, "Road grade and vehicle parameter estimation for longitudinal control using gps," in *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, 2001, pp. 25–29.

[2] J. Ryu, E. J. Rossetter, and J. C. Gerdes, "Vehicle sideslip and roll parameter estimation using gps," in *Proceedings of the AVEC International Symposium on Advanced Vehicle Control*, 2002.

[3] J. Wang, L. Alexander, and R. Rajamani, "Friction estimation on highway vehicles using longitudinal measurements," *Journal of dynamic systems, measurement, and control*, vol. 126, no. 2, pp. 265–275, 2004.

[4] N. Abas, A. Legowo, and R. Akmeliawati, "Parameter identification of an autonomous quadrotor," in *Mechatronics (ICOM), 2011 4th International Conference On*. IEEE, 2011, pp. 1–8.

[5] D. Mellinger, Q. Lindsey, M. Shomin, and V. Kumar, "Design, modeling, estimation and control for aerial grasping and manipulation," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 2668–2673.

[6] K. U. Lee, Y. H. Yun, W. Chang, J. B. Park, and Y. H. Choi, "Modeling and altitude control of quad-rotor uav," in *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*. IEEE, 2011, pp. 1897–1902.

[7] R. He, Y. Zhao, S. Yang, and S. Yang, "Kinematic-parameter identification for serial-robot calibration based on poe formula," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 411–423, 2010.

[8] J.-M. Renders, E. Rossignol, M. Becquet, and R. Hanus, "Kinematic calibration and geometrical parameter identification for robots," *IEEE Transactions on robotics and automation*, vol. 7, no. 6, pp. 721–732, 1991.

[9] G. Venture, P.-J. Ripert, W. Khalil, M. Gautier, and P. Bodson, "Modeling and identification of passenger car dynamics using robotics formalism," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 3, pp. 349–359, 2006.

[10] N. Keivan and G. Sibley, "Asynchronous adaptive conditioning for visual–inertial slam," *The International Journal of Robotics Research*, vol. 34, no. 13, pp. 1573–1589, 2015.

[11] F. Nobre, M. Kasper, and C. Heckman, "Drift-correcting self-calibration for visual-inertial slam," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 6525–6532.

[12] T. Schneider, M. Li, M. Burri, J. Nieto, R. Siegwart, and I. Gilitschenski, "Visual-inertial self-calibration on informative motion segments," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 6487–6494.

[13] A. Khan, D. Ceglarek, and J. Ni, "Sensor location optimization for fault diagnosis in multi-fixture assembly systems," *Journal of manufacturing science and engineering*, vol. 120, no. 4, pp. 781–792, 1998.

[14] S. L. Padula and R. K. Kincaid, "Optimization strategies for sensor and actuator placement," 1999.

[15] K. Hausman, J. Preiss, G. S. Sukhatme, and S. Weiss, "Observability-aware trajectory optimization for self-calibration with application to uavs," *IEEE Robotics and Automation Letters*, 2017.

[16] N. Keivan and G. Sibley, "Constant-time monocular self-calibration," in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1590–1595.

[17] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE, 2015, pp. 1094–1099.

[18] S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

[19] S. J. Wright, "Numerical optimization."

[20] E. Coumans, "Bullet physics engine," *Open Source Software: http://bulletphysics. org*, vol. 1, 2010.

[21] E. Catto, "Iterative dynamics with temporal coherence," in *Game developer conference*, vol. 2, no. 4, 2005, p. 5.