A General Framework for Trajectory Optimization with Respect to Multiple Measures

David D. Diel
Scientific Systems Company
ddiel@alum.mit.edu

Robert E. Smith
Scientific Systems Company
robert.smith@ssci.com

Jimmy Touma
Air Force Research Laboratory
jimmy.touma@eglin.af.mil

Niels da Vitoria Lobo
University of Central Florida
niels@cs.ucf.edu

Omar Oreifej
University of Central Florida
oreifej@eecs.ucf.edu

***Abstract*—In the context of GPS-denied navigation, many algorithms are being developed that fuse information from alternative sensors such as visible and infrared cameras, RADAR, and LIDAR. However, the vast majority of these algorithms are tied to particular combinations of sensors and platforms. This is especially common at high technology readiness levels. Therefore, in order to limit the cost of future development, testing, and integration, it is desirable to break coupled solutions into component parts that can be managed independently.**

In this paper, we describe a modular framework for trajectory optimization. This framework serves three major purposes: i) To facilitate the development of new sensors and algorithms; ii) To assist system integrators in testing a variety of navigation components such that their accuracy can be characterized on a level playing field; and iii) To enable hot-swapping of navigation components that have not necessarily been tested together.

Our approach is to identify the broad structure of the problem while leaving the details to be implemented differently for each application. We derive a probabilistic objective function closely related to a graph-based SLAM formulation and then illustrate the physical interpretation of each mathematical term with specific examples. Novel contributions to visual processing are discussed, and a new evolutionary optimization method is proposed.

To encourage adoption of our methodology, the framework and several example components are being made available online as an Open Source project [5].

I. Introduction

Trajectory optimization is a general approach to navigation, or continuous self-localization relative to a global frame, that optimally fuses information from all available sources while satisfying dynamic constraints. Algorithms that use this approach are often considered too slow for real-time applications because the general problem varies in structure and complexity with increasing time, changes in platform dynamics, and sensor availability. Therefore, a variety of approximations to the general nonlinear problem have been introduced, such as; linearization of motion dynamics, graph- or tree-based pose representation, particle representation of instantaneous states, and sensor model simplification and linearization. Some algorithms estimate motion in fewer than six Degrees of Freedom (6-DoF), providing position only, orientation only, or coordinates constrained to a local plane. Some maintain a history of motion, while others keep a current state estimate, resulting in non-uniform treatment of absolute information (e.g. GPS) versus relative information (e.g. visual feature tracking). These approximation techniques have led to a variety of tractable problems and corresponding optimization algorithms.

Many effective navigation devices are already in widespread use. However, nearly all existing implementations are tied to particular sensors and platforms, resulting in stovepiped architectures. For example, algorithms that map the environment usually require a rangefinder or camera to be available continuously. Although some systems are robust to sensor cutouts, we know of none that support the addition of new sensor types at runtime.

In this paper, we describe the Trajectory Optimization Manager for Multiple Algorithms and Sensors (TOMMAS) [5][7]. TOMMAS is an object-oriented framework that standardizes the all-source, all-platform, maximum likelihood navigation problem with nonlinear state evolution and measurement error distributions. It clarifies the line between abstract trajectory optimization and many specific practical implementations. It is not an algorithm, so it cannot be evaluated in terms of computational complexity. However, it facilitates rapid algorithm development, assists system integrators in testing the accuracy of algorithms and sensors on a level playing field, and enables hot-swapping of navigation components that have not necessarily been tested together in advance.

A. The Framework and Its Component Parts

The TOMMAS framework defines interfaces for the communication of navigation-relevant information, partitioning the navigation problem into three abstract classes of components:

This material is based upon work supported by The Air Force Research Laboratory (AFRL) under Contract No. FA8651-09-C-0184. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Air Force Research Laboratory or the U. S. Government.

1. A `DynamicModel` relates a set of parameters to the rigid body trajectory of a physical system such as a car, airplane, or person. This interface also provides stochastic information about the parameters known prior to data collection.
2. A `Measure` evaluates a given trajectory against navigation-relevant information from one or more sensors. It represents a novel concept in which algorithms, sensors, and their calibration data are packaged together. Neither raw data nor sensor-specific features are communicated through this interface.
3. An `Optimizer` is an algorithm that finds one or more elements in the set of maximum likelihood trajectories by intelligently generating and modifying the parameters of one or more `DynamicModel` instances and passing them through zero or more `Measure` instances.

A component implementation that meets any of the TOMMAS interface specifications is interchangeable with others of the same type at runtime. This extreme level of modularity comes at the expense of support for raw data access or cross-sensor mapping. However, TOMMAS places no limits on external interfaces beyond its use of the namespace `tom`.

B. Benefits to System Integrators

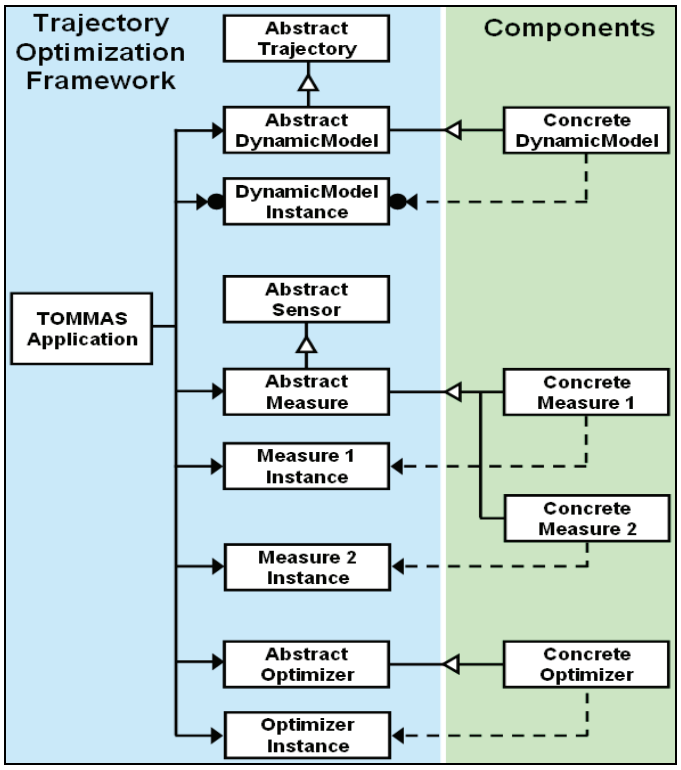

Fig. 1. Class inheritance diagram illustrating the use of the factory pattern to manage components. The graphical notation is from Design Patterns [9].

TOMMAS simplifies system integration by leaving the details of component implementation to specialists. A system integrator interacts with the framework by specifying a set of components that inherit from base classes as in Fig. 1. These components represent the physical system and define optimality for a specific application. The framework then provides an Application Programming Interface (API) that makes it easy for the system integrator to manage components and to optimize the scalar objective function derived in a later section.

The TOMMAS interface standard creates a level playing field for evaluation and competition between components. There are no framework parameters to tune at integration time, so new components can be developed and validated independently. Furthermore, the project code and documentation are Open Source and BSD Licensed in order to encourage widespread adoption of TOMMAS, which could lead to new markets for packaged navigation sensors and algorithms.

C. Related Work

Current state-of-the-art navigation algorithms include the Extended Kalman Filter (EKF) [38], Unscented Kalman Filter (UKF) [17], Ensemble Kalman Filter (EnKF) [6], other Particle Filters (PF) [32], and a variety of Simultaneous Localization and Mapping (SLAM) techniques such as Occupancy Grid Mapping [37], Atlas [2], Tree-Based Network Optimization (TORO) [11], Incremental Smoothing and Mapping (iSAM) [18], and general Graph Optimization [12].

This work identifies a universal structure shared by nearly all of the above algorithms. In many ways, TOMMAS represents the wrapper in which these algorithms can be packaged. It is not a centralized software repository for gathering navigation algorithms, like OpenSLAM [36], the Carnegie Mellon Robot Navigation Toolkit (CARMEN) [27], or Willow Garage's Robot Operating System (ROS) [31], but it has the potential to enhance interoperability within and across software repositories.

D. Comparison to Graph-Based SLAM

In order to illustrate structural similarities and differences between our problem formulation and an existing formulation in SLAM literature, we compare TOMMAS to Grisetti's description of Graph-Based SLAM (GBSLAM) [13]. Both formulations optimize trajectories in 6-DoF with respect to a graph-based objective function. However, GBSLAM and TOMMAS differ in the following ways:

In GBSLAM, the solution space is a vector of parameters that describe a set of instantaneous poses in 6-DoF. In TOMMAS, the solution space is a slightly more structured set of parameters that, together with an explicit dynamic model, describe a class of 6-DoF C^1 continuous trajectories.

Dynamic models are not explicit in GBSLAM. In contrast, TOMMAS defines an explicit dynamic model interface. From a theoretical perspective, this distinction may be trivial. However, from a design perspective, TOMMAS provides a clear placeholder for statistical knowledge and assumptions about the mobility of individual platforms. For example, if a vehicle has planar holonomic dynamics, then that self-knowledge can

be packaged into a TOMMAS component that is supplied with the vehicle.

GBSLAM formulates measurements as "zero noise observations" of transformations between poses, accompanied by an information matrix for each measurement. TOMMAS defines general nonlinear measures on the trajectory space, which includes the space of velocities and orientation rates. These measures do not need to fit a quadratic form or have a unique minimum corresponding to a zero noise observation.

In GBSLAM, there is a single graph with uniform indexing that lumps together information from several sensors and information from an implicit dynamic model. The TOMMAS objective has a more detailed structure. It allows for multiple graphs, usually one graph per sensor, and these are separate from the dynamic model. This clarifies the distinct roles of each component. Internal to a TOMMAS optimization method, the individual graphs can be combined into a single graph if the underlying algorithm does not recognize these distinctions.

II. Navigation by Cost Graph Optimization

TABLE I. BASIC SETS

$\mathbb{R}$	real numbers
$\mathbb{R}_+$	non-negative real numbers
$\mathbb{Z}_+$	non-negative integers
$\mathbb{S}^3$	quaternion rotation group

TABLE II. EMBELLISHMENTS

$\dot{}$	temporal derivative
*	optimal solution
$\hat{}$	intermediate estimate
^{T}	transpose

TABLE III. FRAMEWORK VARIABLES AND FUNCTIONS

$\mathbb{K} = \{ k : k \in \mathbb{Z}_+, k \le K \}$	discrete time index
$\mathbb{T} = \{ t : t \in \mathbb{R}, t_0 \le t \le t_K \}$	continuous time index
$\mathbb{M} = \{ m : m \in \mathbb{Z}_+, m < M \}$	measure index
$\mathbb{A} = \left\{ \begin{matrix} (a,b) : a,b \in \mathbb{Z}_+, \\ A_m \le a \le b \le B_m, m \in \mathbb{M} \end{matrix} \right\}$	ordered pair of data nodes
$\mathbb{U} = \{ \mathbf{u} : \mathbb{Z}_+ \to \{0,1\} \}$	raw sensor data
$\mathbb{V} = \{ \mathbf{v} : \mathbb{K} \to \mathbb{Z}_+^{D_k}, k \in \mathbb{K} \}$	dynamic model parameters
$\mathbb{X} = \{ \mathbf{x} : \mathbb{T} \to \mathbb{R}^3 \times \mathbb{S}^3 \}$	continuous body trajectory
$\mathbf{F} : \mathbb{V} \times \mathbb{U} \to \mathbb{X}$	functional dynamic model
$\mathbf{p} : \mathbb{T} \to \mathbb{R}^3$	body position
$\mathbf{q} : \mathbb{T} \to \mathbb{S}^3$	body orientation
$r : \mathbb{V} \times \mathbb{K} \to \mathbb{R}_+$	prior measure
$s : \mathbb{M} \times \mathbb{U} \times \mathbb{X} \times \mathbb{A} \to \mathbb{R}_+$	conditional measure

The mathematical objective of trajectory optimization is to find one or more elements in the set of maximum likelihood trajectories $\mathbf{x}^*$ given the data $\mathbf{u}$, as in

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{X}}{\operatorname{argmax}} \left\{ \mathrm{P}_{\mathbf{x}|\mathbf{u}} (\mathbf{x} | \mathbf{u}) \right\}. \tag{1}$$

The problem can be reduced to finding one or more elements in the set of maximum likelihood parameter vectors $\mathbf{v}^*$ subject to the dynamic model constraint $\mathbf{x} = \mathbf{F}(\mathbf{v}, \mathbf{u})$ given the data $\mathbf{u}$, as in

$$\mathbf{v}^* = \underset{\mathbf{v} \in \mathbb{V}}{\operatorname{argmax}} \left\{ \mathrm{P}_{\mathbf{v}|\mathbf{u}} (\mathbf{v} | \mathbf{u}) \right\}. \tag{2}$$

See Section III for more details about the role of dynamic models in our formulation.

The likelihood maximization problem can be converted to a cost minimization problem by taking the negative log of the distribution as follows:

$$\mathbf{v}^* = \underset{\mathbf{v} \in \mathbb{V}}{\operatorname{argmin}} \left\{ -\log\left(\mathrm{P}_{\mathbf{v}|\mathbf{u}} (\mathbf{v} | \mathbf{u}) \right) \right\}. \tag{3}$$

In order to impose structure onto the problem, we divide the total probability over the parameter space into marginal parts. By assuming that the probability mass associated with each $\mathbf{v}_k$ is independently identically distributed and that the probability mass functions associated with $\mathbf{u}$ are independent, the application of Bayes theorem leads to the following product of terms in a graph structure:

$$\mathbf{v}^* = \underset{\mathbf{v} \in \mathbb{V}}{\operatorname{argmin}} \left\{ -\log\left(\prod_{k \in \mathbb{K}} \mathrm{P}_{\mathbf{v}} (\mathbf{v} | k) \prod_{m \in \mathbb{M}} \prod_{(a,b) \in \mathbb{A}} \mathrm{P}_{\mathbf{u}|\mathbf{v}} (\mathbf{u} | \mathbf{x}, (a,b), m) \right) \right\}. \tag{4}$$

Any arbitrary function that does not depend on $\mathbf{v}$ can be added to the objective without affecting its overall shape or minimizing solution(s). We take advantage of this fact to utilize partially known mass functions (ie. those that are only known over a part of their domain or those with an unknown integral). For this reason and as a matter of convention, we divide each term inside the log function by its infinity norm, which ensures that the objective is non-negative:

$$\mathbf{v}^* = \underset{\mathbf{v} \in \mathbb{V}}{\operatorname{argmin}} \left\{ \begin{matrix} -\sum_{k \in \mathbb{K}} \log\left(\dfrac{\mathrm{P}_{\mathbf{v}} (\mathbf{v} | k)}{\left\| \mathrm{P}_{\mathbf{v}} (\mathbb{V} | k) \right\|_\infty} \right) \\ -\sum_{m \in \mathbb{M}} \sum_{(a,b) \in \mathbb{A}} \log\left(\dfrac{\mathrm{P}_{\mathbf{u}|\mathbf{v}} (\mathbf{u} | \mathbf{x}, (a,b), m)}{\left\| \mathrm{P}_{\mathbf{u}|\mathbf{v}} (\mathbb{U} | \mathbf{x}, (a,b), m) \right\|_\infty} \right) \end{matrix} \right\}. \tag{5}$$

Finally, we substitute the individual summands with the prior measure r and the conditional measures s_m, leading to the specific cost minimization problem

$$\mathbf{v}^* = \underset{\mathbf{v} \in \mathbb{V}}{\operatorname{argmin}} \left\{ \sum_{k \in \mathbb{K}} r(\mathbf{v}, k) + \sum_{m \in \mathbb{M}} \sum_{(a,b) \in \mathbb{A}} s_m (\mathbf{u}, \mathbf{x}, (a,b)) \right\}. \tag{6}$$

Once this problem has been solved, then the optimal trajectory can be computed by inserting the resulting parameters back through the dynamic model.

E. Hints for Efficient Optimization

The framework allows an optimizer to query r and s_m before summation takes place in order to search for global minima in Eq. (6). This formulation suggests automated learning of the relationship between time indexed parameters and time indexed cost graphs. In general, neither convexity nor uniqueness of solution is guaranteed; however, specific sets of components can be designed to offer these guarantees.

A. Time Domain Extension

The structure of the problem is fixed during a single optimization step, after which it is allowed to change. At that instant, the bounding indices K, M, A_m, and B_m are refreshed to match the current structure of available information. The framework places the optimization method in control of the refresh process, so that it can poll for new information and exclude information, if necessary, as computational complexity increases.

III. Dynamic Models for Deterministic Trajectory Generation

Consider the following general models for continuous or discrete nonlinear dynamic systems, where the vector-valued inputs and outputs are defined canonically (not using the notation in Table III):

$$\underbrace{\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t, \mathbf{v}_t, \mathbf{u}_t)}_{\text{continuous}} \qquad \underbrace{\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{v}_k, \mathbf{u}_k)}_{\text{discrete}}. \tag{7}$$

These equations have been established to model the dynamics of numerous physical systems, from Brownian particles to ground vehicles, fighter jets, and even animals. In the continuous case, the transition function $\mathbf{f}$ is typically placed in an integration loop, such that the instantaneous states $\mathbf{x}_t$ are computed incrementally at increasing time instants given an initial condition $\mathbf{x}_0$. The input $\mathbf{u}_t$ represents data that is known with exact precision, and the input $\mathbf{v}_t$ represents discrete parameters whose probability mass function is known. The implementation of the transition function and the integration method differ slightly in the discrete case, but the details do not affect our development.

If the modeled states comprise a rigid-body trajectory, and an interpolation method is specified in the discrete case, then the dynamics can be derived in functional form as follows (using the notation in Table III):

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} = \mathbf{F}(\mathbf{v}, \mathbf{u}). \tag{8}$$

In this form, a dynamic model generates a rigid-body trajectory from a structured set of parameters. This serves dual purposes: It reduces the space of all continuous trajectories to a substantially smaller parameter space, and it provides a natural way to constrain navigation results to achievable platform dynamics.

A single parameter can affect the entire trajectory function. This attribute of our model has benefits and drawbacks. Benefits include guaranteed continuity of the trajectory and the ability to apply functional measures (see Section IV). The main drawback is the lack of efficient optimization techniques designed to interact with this type of model.

A dynamic model is a deterministic function that depends on stochastic parameters. Assuming that the set of all possible parameter vectors comprises a probability space, and that parameter vectors are independently and identically distributed according to the probability mass function $\mathrm{P}_\mathbf{v}$, we identify a key component of the objective function:

$$r(\mathbf{v}, k) = -\log\left(\frac{\mathrm{P}_\mathbf{v}(\mathbf{v}|k)}{\|\mathrm{P}_\mathbf{v}(\mathbb{V}|k)\|_\infty} \right). \tag{9}$$

The TOMMAS `DynamicModel` class standardizes the interfaces to $\mathbf{F}$ and r. It stores and provides access to the stochastic parameters $\mathbf{v}$ that perturb the C^1 continuous 6-DoF rigid-body trajectory $\mathbf{x}$. Motion is defined relative to the Earth-Centered Earth-Fixed (ECEF) frame and time is defined in seconds since midnight on 1980 JAN 06 (GPS standard). This class provides a general interface for specifying the structure of $\mathbf{v}$ and enforces that the domains of $\mathbf{x}$ and $\mathbf{v}$ grow in a consistent manner as time moves forward. Finally, in order to support optimizers that expect to manipulate real parameters, methods for converting integer parameters to real numbers and vice versa are specified.

IV. Graph-Based Trajectory Measures

Consider the following general nonlinear sensor model in canonical form (not using the notation in Table III)

$$\mathbf{y}_t = \mathbf{g}(\mathbf{x}_t, \mathbf{w}_t), \tag{10}$$

where the function $\mathbf{g}$ returns an instantaneous measurement vector $\mathbf{y}_t$ whose value depends on an instantaneous state $\mathbf{x}_t$ and the error parameters $\mathbf{w}_t$. Many sensors have been modeled using this form. However, some sensors of interest simply do not produce instantaneous measurements.

A common example of a non-instantaneous sensor is a gyroscope. Although gyroscopes are often modeled as if they measure instantaneous rotation rates, they are most accurately modeled as measuring changes in orientation over discrete time periods [22].

Another example of a non-instantaneous sensor is a camera that identifies a single feature appearing in two images taken at different times. The feature displacement in image coordinates depends on the position and orientation of the sensor at both times, among other factors.

A. Graph-Based Measurement Model

Measurements may accumulate information over multiple times or may arise from integration over an interval of time. In

general, these time intervals can overlap, leading to a functional measurement model

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{w}, (a,b)), \tag{11}$$

where each pair of node indices (a,b) forms an edge in an incomplete graph, as illustrated in Fig. 2. Indices and time stamps must be related by a known monotonically increasing function, such that a given pair of indices associates a single measurement with the closed time interval $[t_a, t_b]$. The time stamps do not need to be regularly spaced or frequent.

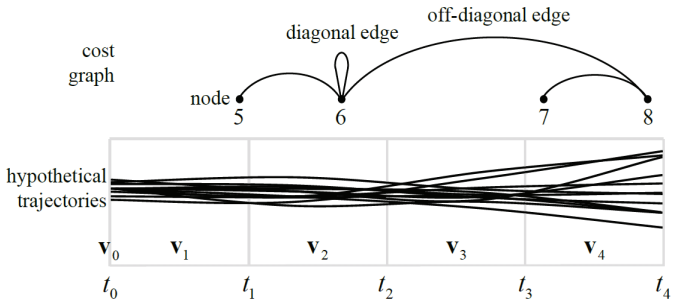

Fig. 2. Relationship of indices between a cost graph and a set of hypothetical trajectories. Each block corresponds to a discrete time period.

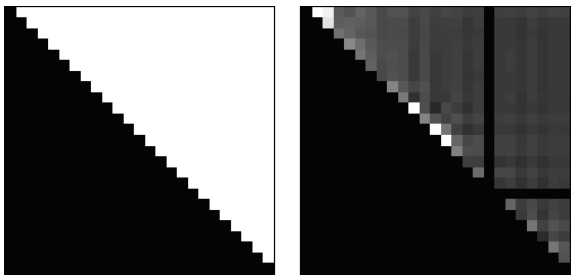

Fig. 3. Graph adjacency (left) and edge cost (right) for a single trajectory evaluated by a relative measure. The bright spots indicate data that is inconsistent with the trajectory. The dark horizontal and vertical lines indicate zero-cost edges associated with an invalid data node.

B. From Measurements to Trajectory Measures

In order to clarify terminology, let the *data* $\mathbf{u}$ be the set of raw unfiltered numerical values recorded by all available sensors, including calibration values. The data is assumed to be completely determined by the actual body *trajectory* $\mathbf{x}$ and a particular realization of the *error parameters* $\mathbf{w}$. Each *measurement* $\mathbf{y}$ can be any conceivable function of the data. We seek a *measure* s_m based on the relative likelihood of the observed data given any hypothetical *trajectory* $\mathbf{x}$.

Assuming that $\mathbf{g}$ is invertible with respect to $\mathbf{w}$, and remembering that $\mathbf{x}$ and $\mathbf{y}$ are functions of $\mathbf{u}$ and $\mathbf{v}$, the following error model is obtained:

$$\mathbf{w} = \mathbf{g}^{-1}(\mathbf{x}, \mathbf{y}, (a,b)) \sim \mathrm{P}_{\mathbf{u}|\mathbf{v}}(\mathbf{u}|\mathbf{x}, (a,b), m). \tag{12}$$

This provides the basis of a trajectory measure for each combination of sensors and algorithms and each graph edge as follows (using the notation in Table III):

$$s_m(\mathbf{u}, \mathbf{x}, (a,b)) = -\log\left(\frac{\mathrm{P}_{\mathbf{u}|\mathbf{v}}(\mathbf{u}|\mathbf{x}, (a,b), m)}{\left\|\mathrm{P}_{\mathbf{u}|\mathbf{v}}(\mathbb{U}|\mathbf{x}, (a,b), m)\right\|_\infty}\right). \tag{13}$$

The TOMMAS `Measure` class standardizes the interface to each s_m without requiring explicit computation of $\mathbf{y}$ or $\mathbf{w}$. This not only has the potential to reduce processor burden, but it also makes it possible to wrap a wide variety of sensors and algorithms with a uniform interface. For example, suppose s_1 represents a calibrated camera coupled with a sparse feature tracking algorithm, and s_2 represents a GPS unit. Since they both possess the same interface, they can be tested and their performance can be compared on a level playing field.

V. Examples of Optimizers

This section describes components that inherit from the `Optimizer` base class. The development of trajectory optimization methods that are both general and efficient remains an open problem. Therefore, we present a few methods that we have implemented, and then propose future work in this field.

A. Linear Kalman Filter

This TOMMAS component implements a parameter update step that is equivalent to the linear least squares update in the linear Kalman filter [19]. There is no guarantee that the objective function will match the underlying assumptions of the filter (i.e. linear system dynamics and normally distributed disturbances). Therefore, this component exists only to demonstrate the advantages and disadvantages of applying this filter to new problems. It is known to be efficient and optimal when the objective is quadratic, and it is known to produce suboptimal solutions otherwise.

To apply the classical Kalman filter equations in our framework, some modifications are necessary: 1) Since means and covariances are not directly accessible, the filter must query the Jacobian and Hessian of the functions r and s_m in the vicinity of a given trajectory hypothesis using finite differences in order to derive the first and second moments of equivalent normal distributions; and 2) Only diagonal edges in the measurement graph are included in the filter. Information in the off-diagonal edges is ignored.

B. Matlab Genetic Algorithm

The MATLAB Genetic Algorithm and Direct Search (GADS) toolbox contains several functional optimization methods, one of which is a customizable Genetic Algorithm (GA). To use the MATLAB GA, we needed to implement two transformations; one that converts a set of dynamic model parameters into a bit string, and another that converts a bit string back into a set of parameters. In terms of configuration, we selected built-in uniform functions for initial population creation, parent selection, and mutation, along with single-point crossover and proportional fitness scaling. This optimization method is guaranteed to converge to the optimal solution eventually, but it takes a relatively long time to do so.

C. Evolutionary Optimization with Linkage Learning

Using TOMMAS as a rapid development tool, we have begun to explore optimization techniques from the field of evolutionary computation, with a focus on linkage learning [3][10]. Linkage learning is a technique for implicitly discovering and exploiting correlations between various parameter inputs and the resulting costs. The cost graph structure in the TOMMAS framework provides explicit information about these correlations that can be utilized in linkage learning. In addition, most dynamic models introduce an implicit correlation between the values of individual parameters and the resulting trajectory during limited intervals of time.

To build a foundation for discovering and exploiting these correlations, we review Holland's fundamental theories of evolutionary computation below [16]:

- ***The K-Armed Bandit Argument:*** Given a probabilistic decision between a finite number of options, it is a near-optimal strategy to allocate exponentially increasing numbers of trials to the observed best alternatives.
- ***The Schema Theorem:*** A Genetic Algorithm (GA) assigns exponentially increasing number of copies to combinations of parameter values that are consistently observed to be better if those combinations also survive recombination and mutation operators with high probability. We call such highly survivable combinations of parameters building blocks.
- ***Implicit Parallelism:*** By processing a number of individuals using GA operators, a vastly larger number of building block combinations are implicitly processed in the near-optimal fashion discussed above.
- ***The Building Block Hypothesis:*** The core heuristic of the GA approach is that for many problems and encodings, recombination of building blocks yields high-quality solutions.

One way to apply these ideas is to design the objective function such that highly correlated parameters appear relatively close to one another in a string. As a population of strings are spliced and recombined at randomly selected crossover points, those parameters that are close together in the encoding will be more likely to survive than those that are far apart, thereby acting as building blocks. To some extent, this technique is already implicit in the TOMMAS definition of the dynamic model, because its input parameters are ordered by a time index.

Linkage learning offers a more advanced concept of building blocks in which parameter proximity only plays a minor role. In linkage learning algorithms, metrics and procedures determine combinations of parameters whose values are correlated with better solutions. This information is used to alter the genetic operators to ensure that these combinations survive with high probability. Often, a linkage model, such as a map of the strength of pair-wise parameter correlations, is built to aid in biasing the survivability of parameter combinations [34].

A straightforward method of testing this theory is as follows: When crossing-over two individuals, the probability of taking a parameter from one of the two parents can be taken in proportion to the relative (inverse) magnitude of the costs on the edge(s) of the cost graph that are most affected by that parameter. Over generations of the evolutionary optimization process, this should tend to preserve sets of parameters that consistently lower overall costs as building blocks, since these tend to be taken from a single parent.

In future work, we would like to explore this simple scheme, as well as methods that exploit more complex statistical analysis of cost graphs to build linkage models.

VI. Examples of Dynamic Models

This section describes components that inherit from the `DynamicModel` base class.

A. Brownian Planar Dynamic Model

This component is based on the dynamics of a free body restricted to planar 3-DoF motion under the effect of bounded forces in the range $[-\mathbf{\Phi}, \mathbf{\Phi}]$. We call it Brownian because the probability of the force taking on any particular value is modeled by a truncated normal distribution. Therefore, the cost function r is convex, quadratic, and minimized when all elements of $\mathbf{v}_k$ lie in the middle of their range.

B. Bounded Markov Dynamic Model

As the name implies, this component implements a Markov motion model. It is a simple second order model of a free body in 6-DoF that is driven by a forcing function. The body has unit inertia in translation and rotation. The forcing function is piecewise constant and bounded in the range $[-\mathbf{\Phi}, \mathbf{\Phi}]$, where the scale $\mathbf{\Phi}$ can be configured for a particular application. The cost function r is uniformly zero everywhere.

C. Strapdown Inertial Integration Model

One common method of modeling a dynamic system is to attach a 6-DoF inertial sensor to the physical system, and then treat the data from the inertial sensor, coupled with its error process and statistics, as the model. This method is also called strapdown mechanization.

Our strapdown inertial integration model inverts the model described above in order to generate a plausible trajectory $\hat{\mathbf{x}}$ given a hypothetical set of error parameters $\hat{\mathbf{v}}$. We assume the existence of error-compensation functions $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ that produce corrected gyroscope and accelerometer data, respectively, and a function $\boldsymbol{\gamma}$ that supplies an initial condition $\boldsymbol{\gamma}(\hat{\mathbf{v}}_0)$. These functions contribute to the discrete process model

$$\begin{bmatrix} \hat{\mathbf{p}}(t_{k+1}) \\ \hat{\mathbf{q}}(t_{k+1}) \\ \dot{\hat{\mathbf{p}}}(t_{k+1}) \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{p}}(t_k) + \Delta_{t,k}\dot{\hat{\mathbf{p}}}(t_k) + \frac{\Delta_{t,k}^2}{2}\mathbf{R}(\hat{\mathbf{q}}(t_k))\boldsymbol{\alpha}(\mathbf{u},\hat{\mathbf{v}},k) \\ \hat{\mathbf{q}}(t_k)\mathbf{Q}(\Delta_{t,k}\boldsymbol{\omega}(\mathbf{u},\hat{\mathbf{v}},k)) \\ \dot{\hat{\mathbf{p}}}(t_k) + \Delta_{t,k}\mathbf{R}(\hat{\mathbf{q}}(t_k))\boldsymbol{\alpha}(\mathbf{u},\hat{\mathbf{v}},k) \end{bmatrix} \tag{14}$$

where $\mathbf{R}(\bullet)$ converts rotation from quaternion to matrix form, $\mathbf{Q}(\bullet)$ converts from axis-angle form to quaternion form, and the time increment is $\Delta_{t,k} = t_{k+1} - t_k$. Finally, the discrete trajectory is interpolated using a piecewise quadratic spline for translation and a Hermite spline for rotation [20].

VII. Examples of Measures

This section describes components that inherit from the `Measure` base class. Each of our examples corresponds to a single sensor, although it is possible to create a measure that extracts information from multiple sensors.

A. Distance to the Nearest [Chain Restaurant]

Consider the simplistic verbal cue "I am near a [Chain Restaurant]." Given access to a database of landmarks in the contiguous United States, this phrase can be interpreted as a mathematical measure of the shortest distance along the Earth's surface between an observer and the named landmark. Fig. 4 shows what the function s might look like based on this information. This component illustrates the breadth of potential applications of our approach.

Fig. 4. Illustration of the distance to the nearest [Chain Restaurant] within the contiguous United States [39].

B. Visual SfM-Based Measure

This measure computes camera motion given any pair of images from which at least eight matching point features can be extracted. It solves for the extrinsic transformations between camera poses (egomotion) up to a translation scale factor, and it also solves for scene structure up to an initial offset and scale. The transformations between camera poses are then stored in order to evaluate subsequent trajectory hypotheses relative to them.

Our algorithm is an adaptation of recent work on Structure from Motion (SfM) similar to [15], [30] and [35]. Like other TOMMAS components, this measure is intended as a starting point for further development, also known as a reference implementation. It is publicly available and licensed under GPL.

SfM Algorithm: The algorithm begins by finding SURF interest points in a pair of images. Matching points are identified by extracting a SURF feature vector at each point and evaluating pairs using the nearest neighbor criterion [1].

Following the methodology in [15], RANSAC [8] detects and removes matches that are not mutually consistent with a set of known intrinsic calibration parameters while simultaneously fitting a geometric model to the data. This process iteratively employs the eight-point algorithm [14] to estimate the essential matrix in a linear least squares sense.

The linear estimate of the essential matrix is then further refined as follows [26]: The essential matrix is factored into the product of a skew symmetric matrix and a rotation matrix, resulting in four possible solutions for the extrinsic transformation. Then, each pair of matched points is triangulated using each of the four solutions, and the solution which results in a positive depth for the highest number of pairs is selected. This solution is then recomposed into a projection matrix that becomes the initial condition for Sparse Bundle Adjustment (SBA) [23], which refines the estimate via local nonlinear optimization.

Eight-Point Algorithm: Given a pair of matched interest points $\mathbf{d}_a$ and $\mathbf{d}_b$ in homogenous image coordinates observed at two times t_a and t_b, the geometric epipolar constraint can be written as

$$\left(\mathbf{H}_a^{-1}\mathbf{d}_a\right)^{\mathrm{T}} \mathbf{E}_{(a,b)} \left(\mathbf{H}_b^{-1}\mathbf{d}_b\right) = 0\,, \tag{15}$$

where $\mathbf{H}_a$ and $\mathbf{H}_b$ are the calibration matrices representing the camera intrinsic parameters at each of the two times, and $\mathbf{E}_{(a,b)}$ is the essential matrix that relates the two poses.

The interest points and the calibration matrices are derived from the raw data $\mathbf{u}$, but the essential matrix is unknown. In order to solve for the eight unknown elements of the essential matrix, epipolar constraint equation can be rearranged to fit the form

$$\boldsymbol{\eta}\,\mathrm{vec}\left(\mathbf{E}_{(a,b)}\right) = 0\,, \tag{16}$$

where $\mathrm{vec}(\bullet)$ produces a column vector made up of the entries of its argument in row-major order, and $\boldsymbol{\eta}$ is a row vector made up of the remaining variables. Each pair of matched interest points gives rise to one equation, and the linear least squares estimate $\hat{\mathbf{E}}_{(a,b)}$ is the null space of the matrix formed by stacking at least eight constraint equations in rows.

The numerical stability of this method can be improved by forcing the essential matrix to have exactly two equal singular

values. This can be done by replacing its singular values as follows

$$\hat{\mathbf{E}}_{(a,b)} \leftarrow \mathbf{\Psi}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}\mathbf{\Omega}^{\mathrm{T}}, \tag{17}$$

where $\mathbf{\Psi}$ and $\mathbf{\Omega}$ are the orthonormal matrices obtained by singular value decomposition (i.e. $\hat{\mathbf{E}}_{(a,b)} = \mathbf{\Psi\Theta\Omega}^{\mathrm{T}}$).

Proposed Measure: The essential matrix is related to the camera motion by its factorization [26],

$$\hat{\mathbf{E}}_{(a,b)} = \left[\hat{\boldsymbol{\tau}}_{(a,b)}\right]_{\times}\mathbf{R}\left(\hat{\boldsymbol{\varphi}}_{(a,b)}\right), \tag{18}$$

where $\hat{\boldsymbol{\tau}}_{(a,b)}$ is the relative translation direction and $\hat{\boldsymbol{\varphi}}_{(a,b)}$ is the relative quaternion rotation of the second camera frame as viewed in the first camera frame. The $\mathbf{R}(\bullet)$ function converts its argument from quaternion to matrix form, and the bracket operator $[\bullet]_{\times}$ converts its argument to the matrix operator form of the cross product. Each vector has a magnitude equal to one, as determined by the geometry and enforced by the factorization algorithm.

Given a trajectory hypothesis with nonzero translation between the two times of interest, and assuming that the camera frame is coincident with the body frame, the normalized camera motion is

$$\boldsymbol{\tau}_{(a,b)} = \mathbf{R}\left(\mathbf{q}(t_a)\right)^{\mathrm{T}}\frac{\mathbf{p}(t_b)-\mathbf{p}(t_a)}{\left\|\mathbf{p}(t_b)-\mathbf{p}(t_a)\right\|} \tag{19}$$

$$\boldsymbol{\varphi}_{(a,b)} = \mathbf{q}(t_a)^{-1}\mathbf{q}(t_b). \tag{20}$$

There are multiple ways to compare the trajectory hypothesis with the visual motion estimate. In this example, we choose a measurement space that represents the angular difference between each motion vector

$$\mathbf{y}_{(a,b)} = \begin{bmatrix}\cos^{-1}\left(\boldsymbol{\tau}_{(a,b)}\circ\hat{\boldsymbol{\tau}}_{(a,b)}\right)\\ \cos^{-1}\left(\boldsymbol{\varphi}_{(a,b)}\circ\hat{\boldsymbol{\varphi}}_{(a,b)}\right)\end{bmatrix}. \tag{21}$$

The distribution of this measurement depends on many factors beyond the scope of this paper. However, if we loosely assume a normal distribution,

$$\mathbf{y}_{(a,b)} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Lambda}_{(a,b)}\right), \tag{22}$$

then the SfM-based contribution to the navigation objective is the quadratic cost function

$$s_m\left(\mathbf{u}, \mathbf{x}, (a,b)\right) = \frac{1}{2}\mathbf{y}_{(a,b)}^{\mathrm{T}}\mathbf{\Lambda}_{(a,b)}^{-1}\mathbf{y}_{(a,b)}. \tag{23}$$

C. Visual Measure of Epipolar Tracking Error

We have developed a novel visual measure based on epipolar tracking error. This measure evaluates a trajectory given a pair of images. It finds salient point features in each image and matches them using either the Kanade-Lucas-Tomasei (KLT) Optical Flow algorithm or the Speeded Up Robust Features (SURF) algorithm. The KLT tracker exploits sparsity of features to expedite the matching process, while SURF offers greater robustness at a higher computational burden. Each feature match $j \in \{1,2,3,\ldots,J\}$ is returned as a pair of ray vectors $\left(\mathbf{c}_{j,a}, \mathbf{c}_{j,b}\right)$ in the camera frame, derived from the image data and the camera projection in $\mathbf{u}$.

Given a trajectory $\mathbf{x}$ that contains the position $\mathbf{p}$ and orientation $\mathbf{q}$ of the body frame, and assuming that the camera frame is coincident with the body frame, vectors in the camera frame are rotated into the world frame as follows:

$$\mathbf{z}_{j,a} = \mathbf{R}\left(\mathbf{q}(t_a)\right)\mathbf{c}_{j,a} \tag{24}$$

where $\mathbf{R}(\bullet)$ converts its argument from quaternion to matrix form. The two ray vectors in the world frame are then compared to the trajectory using

$$\varepsilon_{(a,b),j} = \left(\frac{\mathbf{p}(t_b)-\mathbf{p}(t_a)}{\left\|\mathbf{p}(t_b)-\mathbf{p}(t_a)\right\|}\times\mathbf{z}_{j,a}\right)\circ\mathbf{z}_{j,b}, \tag{25}$$

which is the sine of the angular difference between each feature ray vector and its corresponding epipolar plane. In practice, division by zero can be avoided by multiplying both sides by the denominator and propagating that scale factor through the rest of the development.

In order to create a measure in the form of Eq. (13) from a list of tracked visual features, a measurement function with a known distribution must be specified. There are numerous ways to parameterize the measurement, and each way has its own error statistics. If these statistics are unknown, then they must be determined from experimental data that includes ground truth, such as the Middlebury College stereo datasets [33]. We present a few options for consideration below.

Sum of Squared Differences (SSD): Assuming that the tracking error associated with each feature is independent, unbiased, and normally distributed as in $\varepsilon_{(a,b),j} \sim \mathcal{N}\left(0, \sigma^2\right)$, the measurement

$$y_{(a,b)} = \frac{1}{\sigma^2}\sum_{j=1}^{J}\varepsilon_{(a,b),j}^2 \tag{26}$$

follows the chi-square distribution

$$\mathrm{P}_{\mathbf{u}|\mathbf{v}}\left(\mathbf{u}\middle|\mathbf{x},(a,b),m\right) = \frac{y_{(a,b)}^{(J-2)/2}\exp\left(-y_{(a,b)}/2\right)}{2^{J/2}\Gamma(J/2)} \tag{27}$$

where Γ is the gamma function. In order to complete the measure, we also need to know the infinity norm of the distribution. In cases when there are multiple features $J \geq 2$, the

maximum occurs at $y_{(a,b)} = J - 2$, and its corresponding value is

$$\left\| \mathrm{P}_{\mathbf{u}|\mathbf{v}} \left(\mathbb{U} | \mathbf{x}, (a,b), m \right) \right\|_{\infty} = \frac{(J-2)^{(J-2)/2} \exp\left(-(J-2)/2\right)}{2^{J/2} \Gamma(J/2)}. \tag{28}$$

However, if there is only one visual feature, then the model collapses to the normal distribution, in which case

$$s_m(\mathbf{u}, \mathbf{x}, (a,b)) = \frac{y_{(a,b)}}{2}. \tag{29}$$

Half-Space Test Function: Alternatively, assuming that the tracking error associated with a set of features is equally distributed about zero, we introduce smooth a test function to measure asymmetry:

$$y_{(a,b)} = \frac{1}{2J} \sum_{j=1}^{J} \left(1 + \mathrm{erf}\left(\frac{\varepsilon_{(a,b),j}}{\mu} \right) \right) \tag{30}$$

where $\mathrm{erf}(\bullet)$ is the normal error function and μ is a smoothing parameter related to the pixel spacing. This measurement indicates the ratio of feature tracking errors that lie in the positive half-space to the total number of features. If we assume that $y_{(a,b)}$ follows a truncated normal distribution, then

$$s_m(\mathbf{u}, \mathbf{x}, (a,b)) = \frac{y_{(a,b)}^2}{2\sigma^2}. \tag{31}$$

This measure is less sensitive to large feature tracking errors than the SSD measure, but that does not necessarily make it a better choice. In general, each measure should be evaluated with regard to how well it models the errors of the specific combination of sensors and algorithms that it represents.

VIII. Open Source Implementation

In order to encourage widespread adoption, TOMMAS is provided online as an Open Source and BSD Licensed project [5]. The learning curve for creating a framework component is easily accessible to an individual engineer versed in navigation concepts and either ANSI C++ or MATLAB.

A. Software Engineering Considerations

TOMMAS utilizes several features of object-oriented programming. In particular, it uses class abstraction, encapsulation, inheritance, namespaces or packages, and passing data by value and by reference. Most importantly, modularity is accomplished through polymorphic inheritance from abstract interface classes that implement the factory design pattern.

The stack diagram in Fig. 5 shows the minimal requirements for TOMMAS to run on an embedded system. We selected ANSI C++ for our reference interface because the language is strongly typed, and because it can be compiled on a large variety of platforms across the size, weight, and power (SWaP) spectrum.

All TOMMAS class definitions are mirrored in a weakly typed MATLAB interface. In addition, MEX code is provided that automatically wraps native ANSI C++ components with a MATLAB interface. This special feature bridges the gap between the two languages and greatly simplifies unit testing, as shown in Fig. 6.

Adopting the ANSI C++ version of TOMMAS imposes no computational overhead. As a pure virtual interface, TOMMAS guides system development and integration. Yet, most compilers will optimize (inline) its functions such that the interface adds negligible processor cycles to the implementation. In other words, the computational burden is completely determined by the components that are selected at runtime.

TOMMAS has been successfully tested on several versions of Windows, Linux, and Mac using MSVC, g++, Xcode, and MATLAB.

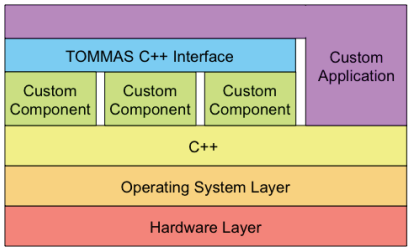

Fig. 5. Stack diagram for embedded TOMMAS components and applications.

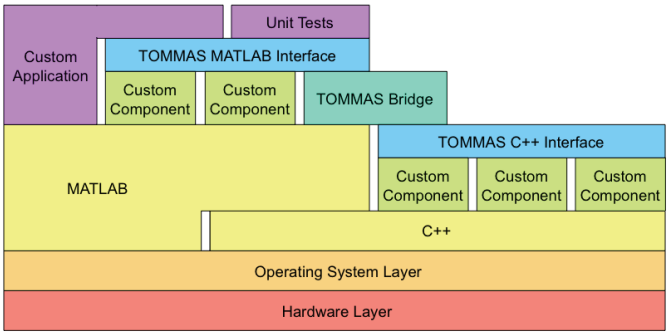

Fig. 6. Stack diagram for rapid development and testing of components through the TOMMAS Bridge between C++ and MATLAB.

Acknowledgments

D.D.D thanks Tony Falcone for his guidance during the early stages of this work, and Patrick Fenelon and Prince Gupta for their contributions to the code repository.

References

[1] H. Bay, A. Ess, T. Tuytelaars, L. van Gool. *SURF: Speeded Up Robust Features.* Computer Vision and Image Understanding (CVIU), 2008.

[2] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. *An Atlas framework for scalable mapping*. In ICRA, v. 2, pp. 1899-1906, Sept. 2003.

[3] D. Coffin and R. E. Smith. *Linkage Learning in Estimation of Distribution Algorithms*. Linkage in Evolutionary Computation: Studies in Computational Intelligence. Page(s) 141-156. Springer, 2008.

[4] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2000.

[5] D. Diel, O. Oreifej, P. Fenelon. *Functional Navigation*. Project on Google Code. http://code.google.com/p/functionalnavigation.

[6] G. Evensen. *The Ensemble Kalman Filter: Theoretical formulation and practical implementation*. Ocean Dynamics, 2003.

[7] M. Fessenden, C. New, J.E. Touma, T.J. Klausutis, D. Diel. *Precision Multi-Sensor Optical Navigation Test-bed Utilizing Ground-Truthed Data Set*. IEEE/ION PLANS. 2010.

[8] M. A. Fischler and R. C. Bolles. *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*. Communications of the ACM, 1981.

[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional Computing Series, 1995.

[10] D. E. Goldberg. *Design of Innovation*. Springer, 2002.

[11] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, W. Burgard. *Efficient estimation of accurate maximum likelihood maps in 3D*. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2007.

[12] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, C. Hertzberg. *Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping*. IEEE Int. Conf. on Robotics and Automation (ICRA), 2010.

[13] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. *A tree parameterization for efficiently computing maximum likelihood maps using gradient descent*. In Proc. of Robotics: Science and Systems (RSS}, 2007.

[14] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 19(6):580 - 593, October 1997.

[15] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. 2004.

[16] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[17] S. J. Julier and J. K. Uhlmann. *A new extension of the Kalman filter to nonlinear systems*. Int. Symp. Aerospace/Defense Sensing, Simulation and Controls, 1997.

[18] M. Kaess, A. Ranganathan, and F. Dellaert. *iSAM: Incremental Smoothing and Mapping*. IEEE Trans. on Robotics, vol. 24, no. 6, 2008.

[19] R. E. Kalman. *A new approach to linear filtering and prediction problems*. Transactions of the ASME–Journal of Basic Engineering, 82, Series D:35–45, 1960.

[20] M. J. Kim, M. S. Kim, and S. Y. Shin. *A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives*. In Proc. of SIGGRAPH, pp. 369-376, 1995.

[21] J. P. Lewis. *Fast normalized cross-correlation*. Vision Interface. 1995.

[22] Litton Guidance and Control Systems. *Product Description of the LN-200 Family*, Document No. 208961, September 1996.

[23] M.I. A. Lourakis, A.A. Argyros. *SBA: A Software Package for Generic Sparse Bundle Adjustment*. ACM Trans. Math. Software, v. 36, n. 1, 2009, New York, NY, USA.

[24] D. G. Lowe. *Distinctive image features from scale invariant keypoints*. International Journal of Computer Vision, 60:91–110, 2004.

[25] B. D. Lucas and T. Kanade. *An iterative image registration technique with an application to stereo vision*. In 7th International Joint Conference on Artificial Intelligence, 1981.

[26] Y. Ma, S. Soatto, J. Kosecka and S. Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. 2003.

[27] M. Montemerlo, N. Roy, S. Thrun, D. Haehnel, C. Stachniss, J. Glover. *Carnegie Mellon Robot Navigation Toolkit (CARMEN)*. http://carmen.sourceforge.net.

[28] D. Nister. *An efficient solution to the five-point relative pose problem*. In: IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 195–202, 2003.

[29] E. Olson, J. Leonard, and S. Teller. *Fast iterative optimization of pose graphs with poor initial estimates*. ICRA, pages 2262–2269, 2006.

[30] M. Pollefeys, L. van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops and R. Koch. *Visual modeling with a hand-held camera*. Int. J. of Computer Vision. 2004.

[31] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. *ROS: an open-source robot operating system*. International Conference on Robotics and Automation, Open-Source Software workshop, 2009.

[32] T. B. Schön, A. Wills, B. Ninness. *System Identification of nonlinear state-space models*. In Automatica, v. 47, pp. 39-49, 2011.

[33] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. *A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms*. CVPR, vol. 1, pages 519-526, 2006.

[34] R. E. Smith. *An Iterative Mutual Information Histogram Technique for Linkage Learning in Evolutionary Algorithms*. Proceedings of the Congress on Evolutionary Computation, IEEE, 2005.

[35] N. Snavely, S. M. Seitz, R. Szeliski. *Photo tourism: Exploring photo collections in 3D*. ACM Transactions on Graphics, 2006.

[36] C. Stachniss, U. Frese, G. Grisetti. *OpenSLAM*. http://openslam.org.

[37] S. Thrun. *Robotic Mapping: A Survey*. Exploring Artificial Intelligence in the New Millenium, 2002.

[38] G. Welch and G. Bishop. *An Introduction to the Kalman Filter*. University of North Carolina at Chapel Hill Technical Report 95-041, 1995.

[39] S. Von Worley. *The Contiguous United States Visualized by Distance to the Nearest [Chain Restaurant]*. http://www.datapointed.net. Data courtesy of AggData. http://www.aggdata.com. Used with permission.