Generalized Architecture for Simultaneous Localization, Auto-Calibration, and Map-building

Eric M. Foxlin
InterSense Incorporated, Burlington, MA, USA, ericf@isense.com

Abstract

This paper discusses the design of a very general architectural framework for navigation and tracking systems that fuse dead-reckoning sensors (e.g. inertial or encoders) with environment-referenced sensors, such as ultrasonic, optical, magnetic, or RF sensors. The framework enables systems that simultaneously track themselves, construct a map of landmarks in the environment, and calibrate sensor intrinsic and extrinsic parameters. The goals of the architecture are to permit easy configuration of numerous sensor combinations including IMUs, GPS, range sensors, inside-out bearing sensors, outside-in bearing sensors, etc., and to provide compatibility with multiple sensor networking standards, distributed sensor fusion algorithms, and implementation strategies. A decentralized Kalman filter based on Carlson's Federated filter algorithm is used to decouple the auto-mapping, auto-calibration and navigation filters to produce a more flexible and modular architecture.

1. Introduction and motivation

In mobile robotics, telematics and other applications, there is much interest recently in systems for tracking (or navigating) a moving entity and mapping it's surrounding environment. This is usually justified because the map-building is necessary for the navigation function, which operates by making relative measurements between the vehicle and certain reference points in the environment. In some applications, the environment map is also wanted for other purposes, or may be the only goal.

When navigating in uncharted territory with no GPS availability, such as Mars rover missions, underground mining or undersea operations, often the only means of navigating is to make use of landmark observations, and the map of the landmarks is not available in advance. This has led to extensive research on the topic of simultaneous localization and map-building (SLAM) [e.g. 1, 3, 8, 9, 10, 11]. Given the extreme difficulty and expense of specialized surveying to obtain landmark locations in preparation for tracking or navigation activities, SLAM is likely to become the predominant approach for applications where satellite navigation is insufficient.

Although most theoretical research on SLAM algorithms is relatively unconcerned with the specific types of sensors used, every application requires different types of sensors adapted for the particular operating environment and performance requirements. Each sensor has its own measurement model with unique distortions and biases that must be included in the tracking filter algorithms for maximum accuracy. Thus we are really interested not just in SLAM, but in simultaneous localization, auto-calibration and map-building (SLACAM), for which appropriate notation will be introduced in Section 2.3.

1.1 Structure of centralized KF solution to SLAM

Smith, Self and Cheeseman [1] introduced a solution to the SLAM problem called Stochastic Mapping based on an Extended Kalman Filter (EKF). For a map with N fiducial points, the state vector consists of the vehicle state, $\mathbf{x}_v$, augmented with the position states of all the fiducials:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_v^T & \mathbf{f}_1^T & \mathbf{f}_2^T & \cdots & \mathbf{f}_N^T \end{bmatrix}^T,$$

where $\mathbf{f}_i$ is the position of the i[th] beacon in the map. The associated estimation error covariance matrix is written

$$\mathbf{P}_{\mathbf{XX}} = \begin{bmatrix} \mathbf{P}_{\mathbf{xx}} & \mathbf{P}_{\mathbf{xf}_1} & \mathbf{P}_{\mathbf{xf}_2} & \cdots & \mathbf{P}_{\mathbf{xf}_N} \\ \mathbf{P}_{\mathbf{f}_1\mathbf{x}} & \mathbf{P}_{\mathbf{f}_1\mathbf{f}_1} & \mathbf{P}_{\mathbf{f}_1\mathbf{f}_2} & \cdots & \mathbf{P}_{\mathbf{f}_1\mathbf{f}_N} \\ \mathbf{P}_{\mathbf{f}_2\mathbf{x}} & \mathbf{P}_{\mathbf{f}_2\mathbf{f}_1} & \mathbf{P}_{\mathbf{f}_2\mathbf{f}_2} & \cdots & \mathbf{P}_{\mathbf{f}_2\mathbf{f}_N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{\mathbf{f}_N\mathbf{x}} & \mathbf{P}_{\mathbf{f}_N\mathbf{f}_1} & \mathbf{P}_{\mathbf{f}_N\mathbf{f}_2} & \cdots & \mathbf{P}_{\mathbf{f}_N\mathbf{f}_N} \end{bmatrix},$$

in which the diagonal blocks represent the covariances of the vehicle position estimate and each fiducial position estimate. The off-diagonal blocks represent the cross-covariances between the vehicle and each fiducial, and between each fiducial and every other fiducial. These cross-covariances are essential to the stochastic mapping approach. If they are ignored (i.e. treated as zero), each successive measurement will be incorrectly interpreted as new independent information and the filter will diverge [9]. Conversely, when the full covariance matrix is maintained and updated using the standard EKF procedures, the algorithm has been shown to perform very well.

Because the fiducials are assumed to have constant position, the state transition and process noise matrices for the EKF time update step will have diagonal structure:

$$\mathbf{\Phi}(k) = \begin{bmatrix} \mathbf{\Phi}_x(k) & & & \\ & \mathbf{I} & & \\ & & \ddots & \\ & & & \mathbf{I} \end{bmatrix} \quad \mathbf{Q}(k) = \begin{bmatrix} \mathbf{Q}_{\mathbf{x}}(k) & & & \\ & \mathbf{0} & & \\ & & \mathbf{0} & \\ & & & \mathbf{0} \end{bmatrix}, \qquad (0.1)$$

which reduces the computational load of the time update from the usual $O(N^3)$ to $O(N)$. Each time the vehicle makes a measurement to one of the fiducials, it is processed by the standard EKF measurement update to obtain new estimates of the vehicle state $\hat{\mathbf{x}}_v$ and all the fiducial positions $\hat{\mathbf{f}}_i$. Updating the full covariance matrix $\mathbf{P}$ requires $O(N^2)$ operations and becomes the rate limiting step for large N.

1.2 Limitations of centralized EKF approach

The centralized EKF algorithm summarized above is optimal, but it tends to constrain the architecture of systems that use it.

The Kalman filter must be specifically designed for a certain set of sensors. The measurement function **h** depends on the type of sensor and how it is positioned on the vehicle. The state vector must be augmented to include any sensor biases. By sensor biases we mean any constant parameters that affect the measurement, even in complex nonlinear ways, such as camera lens distortion parameters. A given type of sensor also has its own rules governing the frequency with which it can acquire data or trigger it's sources, so a fairly specific measurement scheduler and data screening algorithm are also coupled to the KF implementation. Because the details of every sensor are relied upon in the design of the Kalman filter, it is usually not a straightforward task to replace one type of sensor with another. Any time the hardware is changed, the filter must be retuned at a minimum, and reprogrammed if the sensor has a structurally different measurement model or triggering policy. The reprogramming and revalidation effort can be significant if new states need to be added to reflect new or different bias terms.

The stochastic mapping formulation is intended to address the problem of map-building, not sensor calibration, and no explicit provisions are made for sensor bias states. Either all the sensors must be calibrated by other means in the factory and their biases compensated before the measurements enter the Kalman filter, or the bias states can be appended to the vehicle state vector. In the latter case, the designer must decide which bias states to include, and the filter will continue estimating these parameters even after they become well-known. The user cannot reconfigure the system with more or fewer sensors, because that would change the dimension of the vehicle state vector and require reprogramming the function which generates the matrices $\mathbf{\Phi}_x(k)$ and $\mathbf{Q}_x(k)$.

Finally, the algorithm as described is designed for tracking a single vehicle in a fairly limited area. Additional vehicle states could be appended to the state vector to create an optimal algorithm for collaborative mapping, but then the tracking code for all the vehicles would have to run on the same computer, which might be a problem if each vehicle requires its own on-board navigation processor. New fiducials can be added to the map as the vehicle enters a new area, but the computational burden is $O(N^2)$, and after a few hundred fiducials the system may slow down to an unacceptable update rate.

1.3 Goals of this work

This paper presents an architecture that tries to overcome these limitations and provide a framework in which one can rapidly implement a wide variety of user-configurable multi-sensor fusion systems for tracking, auto-calibration, auto-mapping or any combination of these.

The goals of the architecture are:

- Sensor versatility: fuse any combination of types and qualities of sensors -- inside-out, outside-in, mixed, with or without inertial or dead-reckoning sensors.
- "Plug & Track": allow users to configure systems by plugging together self-identifying and self-describing smart sensor modules.
- Scalability: enable systems with large numbers of tracked vehicles and/or large maps.
- Permit implementation of either navigation systems with one on-board processor per vehicle, or tracking systems in which a regional processor tracks all vehicles in the area.
- Algorithm versatility: define the interfaces between modules, but allow changing individual modules to take advantage of the newest large-scale SLAM algorithms. In particular, the architecture shall neither force nor prevent the use of sub-map partitioning.
- Function versatility: the system may switch between simultaneous tracking and auto-calibration, simultaneous tracking and map-building, tracking only, or all at once.
- Object-oriented design: facilitate re-use of the modules even if the top-level system architecture needs to change.

In order to achieve some of these goals the stochastic mapping algorithm is recast from its standard form into a complimentary EKF formulation with explicit states for intrinsic and extrinsic parameters of sensors and targets attached to the vehicle or to the map. This is done in Section 2.3 based on a sensor/target object extraction made in Section 2.1 and 2.2. In section 2.4, the sensor fusion process for a given vehicle is decomposed into several separable modules by making use of a distributed Kalman filtering strategy called the Federated filter [4], modified to fit our complimentary EKF architecture. All of the sensor-specific modeling is separated out from the generic sensor fusion algorithms into two modules called the dead-reckoning unit (DRU) and the measurement management unit (MMU), which are described in Sections 2.5 and 2.6. Figure 1 shows an overview of the architecture.

2. Sensor Fusion Core Architecture

2.1 The sensor/target abstraction

There are a huge variety of sensors used in tracking and navigation applications, each with different performance, operating environments, range, size, cost, update rate, penetration capabilities, and so on. From the sensor fusion algorithm's point of view, however, most sensors belong to one of a few basic categories according to the type of measurement observable they produce:

Dead-reckoning continuous observables

- **Angular rate** (e.g. gyros)
- **Linear acceleration** (e.g. accelerometers)
- **Relative displacement** (wheel encoder, optical flow)

Map-point-referenced discrete update observables

- **Range** (e.g. ultrasonic, Lidar, radar, raw GPS)
- **1-D Bearing** (laser scan, linear CCD, phased array)
- **2-D Bearing** (imaging sensor, quadcell, pan/tilt servo)
- **Range rate** (Doppler radar, phase-coherent acoustic)
- **Range difference** (e.g. TDOA acoustic or radar)
- **Dipole field component** (e.g. active source magnetic tracker, electric field tracker)

Earth-referenced discrete update observables

- **Cartesian position** (e.g. cooked GPS, altimeter)
- **Heading** (e.g. magnetic compass, gyrocompass)
- **Speed** (optical flow, airspeed, waterspeed sensors)
- **Direction** (e.g. optical flow)
- **Homogeneous field** (magnetometers, gravitometers)

Generally, the sensor types listed under dead-reckoning are used for that purpose, and therefore enter into the Kalman filter as control inputs to the system dynamics model, which is implemented in the DRU. The rest of the observable types are treated as measurements and their data is fetched and interpreted by the MMU.

Only the map-point-referenced observables contribute directly to the environment mapping process. All these observables make a measurement between two points; call them a sensor and a target. For convenience we will define the sensor for a bearing angle measurement to be the device at the fulcrum of the bearing angle measurement, even though in the case of the laser scanner and related structured light projectors, this means the "sensor" is the light projector, and the "target" is the photodetector. Each measurement, $\mathbf{z}_{st}$, is a function of the pose $\boldsymbol{\lambda}_{st}$ between a particular sensor and target as well as the biases $\mathbf{k}_s$ and $\mathbf{k}_t$ of the sensor and target respectively:

$$\mathbf{z}_{st} = \mathbf{h}_{st}(\boldsymbol{\lambda}_{st}, \mathbf{k}_s, \mathbf{k}_t) \tag{0.2}$$

For an inside-out measurement, the sensor is attached to the vehicle and the target is fixed in the environment, while for an outside-in measurement a sensor fixed in the environment observes a target fixed to the moving vehicle.

2.2 The main object classes: PSEs, maps, vehicles

The environment map consists of a collection of sensors and targets, each having a pose with respect to the map reference frame, also called the navigation frame or n-frame. Likewise the vehicle has a number of sensors and targets rigidly attached to it, each with a pose relative to the vehicle frame, called the body-frame or b-frame. We define a class of objects called pose-sensing endpoints (PSEs) that may be sensors or targets of any type. Every PSE has the following attributes:

- Pose relative to the environment or vehicle map to which it is attached.
- Bias parameters vector **k.**
- Basic type. (2-D bearing, 1-D bearing, range, range rate or dipole sensor or target)
- Specific type. (Mfg. and model, used to look up details such as range, update rate, etc.)
- Status (ready, busy, etc.)

The position of the PSE is always important, since it is impossible to make use of a measurement between two points without knowing where they are. The orientation parameters may not all be important for certain PSEs, in which case no corresponding filter states need be allocated. For example, for a circular fiducial mark, the orientation is only used to determine visibility, and perhaps to calculate a centroid correction for perspective distortion. It should suffice to enter an approximate orientation and leave it alone. For a camera, however, aim is critical and the orientation must be included in the state vector so that the EKF can refine the alignment.

The PSEs are the atomic objects in this architecture. Each vehicle has a Sensor Fusion Core (SFC) associated with it to perform it's tracking and auto-calibration, and to build it's own private version of the part of the map it sees, which may or may not be shared or merged with map fragments developed by other vehicles. The SFC may either be a dedicated processor running on-board the vehicle, or just a process or thread running inside a server somewhere. Each SFC has a DRU containing the vehicle state, a vehicle map containing pointers to all the PSEs attached to the vehicle, and an environment map containing pointers to all the environment-fixed PSEs the vehicle is currently using.

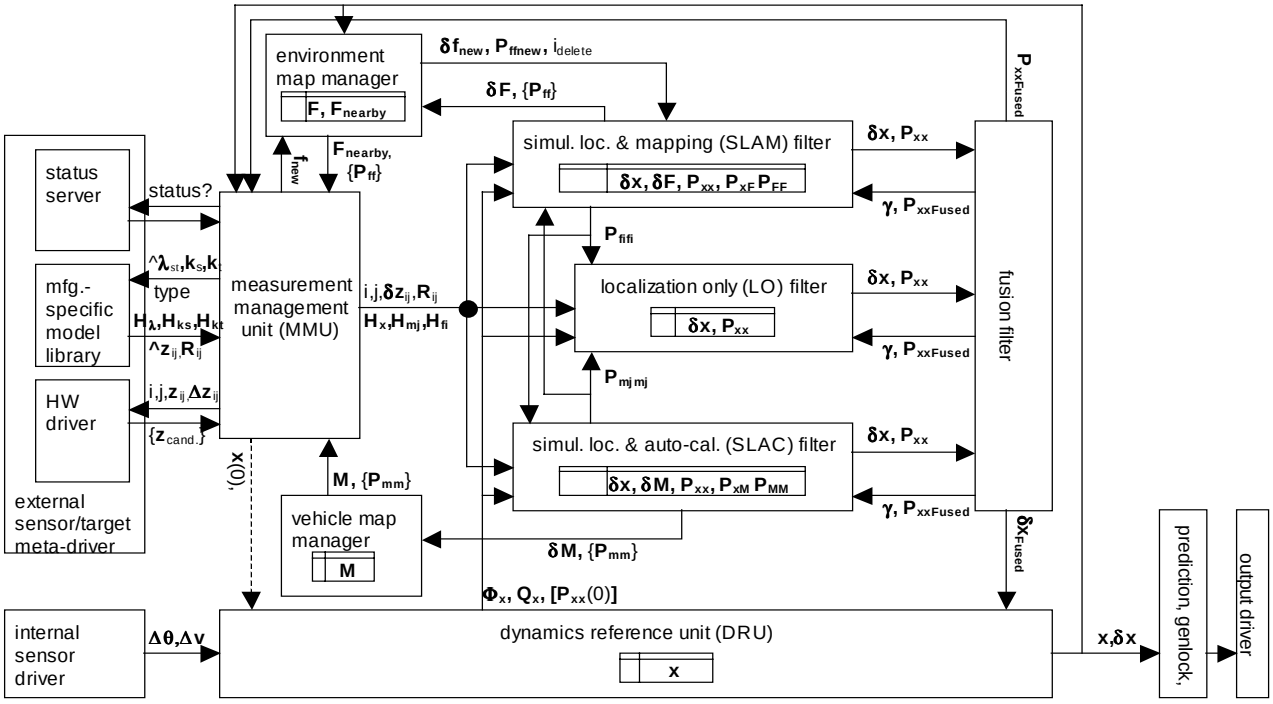

Figure 1: Block diagram of the vehicle-level sensor fusion core (SFC) architecture showing data flow during tracking. Blocks which maintain state information show the names of their internal states in a data store icon.

2.3 Complementary EKF for SLACAM

We now introduce some expanded notation to handle full SLACAM with a complementary filter. The complementary filter [2] implements the extended Kalman filter (EKF) by using an ordinary linear Kalman filter to estimate errors in the underlying states of interest. A linear model for the propagation of error states is obtained by perturbation analysis from the nonlinear dynamics model of the underlying system. The underlying states are not part of the filter, and are stored and propagated separately. In the proposed architecture, the underlying states consist of the vehicle states $\hat{\mathbf{x}}$ which are stored in the DRU, the states $\mathbf{M} = \begin{bmatrix} \mathbf{m}_1^T & \mathbf{m}_2^T & \cdots & \mathbf{m}_M^T \end{bmatrix}^T$ of the M mobile PSEs (MPSEs) that are stored in the vehicle map manager, and $\mathbf{F} = \begin{bmatrix} \mathbf{f}_1^T & \mathbf{f}_2^T & \cdots & \mathbf{f}_N^T \end{bmatrix}^T$ of the N fixed PSEs (FPSEs) that belong to the environment map manager. Each individual PSE "vector" $\mathbf{f_i}$ is composed of a position, orientation and intrinsic parameter (bias) fields:

$$\mathbf{f_i} = \begin{bmatrix} \mathbf{f}_i^{\rho} \\ \mathbf{f}_i^{\varphi} \\ \mathbf{f}_i^{k} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i^{\chi} \\ \mathbf{f}_i^{k} \end{bmatrix} \tag{0.3}$$

where $\mathbf{f}_i^{\chi}$ stands for the combined pose states.

Corresponding to the underlying system "state vector"[1] $\mathbf{X} = \begin{bmatrix} \mathbf{x}^T & \mathbf{M}^T & \mathbf{F}^T \end{bmatrix}^T$ there is an estimation error vector $\boldsymbol{\delta}\mathbf{X} = \begin{bmatrix} \boldsymbol{\delta}\mathbf{x}^T & \boldsymbol{\delta}\mathbf{M}^T & \boldsymbol{\delta}\mathbf{F}^T \end{bmatrix}^T$ that serves as the state of the complementary KF (CKF) with covariance matrix

$$\mathbf{P_{XX}} = \begin{bmatrix} \mathbf{P_{xx}} & \mathbf{P_{xM}} & \mathbf{P_{xF}} \\ \mathbf{P_{Mx}} & \mathbf{P_{MM}} & \mathbf{P_{MF}} \\ \mathbf{P_{Fx}} & \mathbf{P_{FM}} & \mathbf{P_{FF}} \end{bmatrix}.$$

In general, boldface lowercase letters represent vectors and boldface uppercase letters represent matrices, but we use the uppercase **X**, **M**, and **F** to represent stacked vectors. Further abusing notation, write $\hat{\mathbf{X}} = \mathbf{X} \oplus \boldsymbol{\delta}\mathbf{X}$, where the circle around the plus sign reminds us that $\mathbf{X}$ (the "true" values of the underlying states) and $\hat{\mathbf{X}}$ (the estimated values of the underlying states) are not really vectors, and not of the same dimension as $\boldsymbol{\delta}\mathbf{X}$ (because underlying orientation requires at least 4 parameters for non-singular representation, whereas $\boldsymbol{\delta}\mathbf{X}$ contains just the three parameters of the small-angle error rotation vectors, so as to avoid redundant states in the KF). The position, orientation and bias components of all the individual PSE error vectors are stacked in one huge vector (real this time):

$$\boldsymbol{\delta}\mathbf{F} = \begin{bmatrix} \boldsymbol{\delta}\mathbf{f}_1^T & \boldsymbol{\delta}\mathbf{f}_2^T & \cdots & \boldsymbol{\delta}\mathbf{f}_N^T \end{bmatrix}^T$$

Corresponding notation applies to the MPSEs.

Now consider the time update of the CKF. As in Section 1.1, both the MPSE states and the FPSE states are assumed constant, so the simplified $\boldsymbol{\Phi}$ and $\mathbf{Q}$ matrices of Eq. (0.1) hold, and the computation time is linearly related to the number of MPSEs plus FPSEs:

$$\widehat{\boldsymbol{\delta}\mathbf{x}}(k+1) = \widehat{\boldsymbol{\delta}\mathbf{x}}(k) + \boldsymbol{\Phi}_x(k)\widehat{\boldsymbol{\delta}\mathbf{x}}(k)$$

$$\mathbf{P_{xx}}(k+1) = \boldsymbol{\Phi}_\mathbf{x}(k)\mathbf{P_{xx}}(k)\boldsymbol{\Phi}_\mathbf{x}(k)^T + \mathbf{Q}_\mathbf{x}(k)$$

$$\textit{for} \quad j = 1,\ldots,M:$$

$$\mathbf{P}_{\mathbf{x}\mathbf{m}_j}(k+1) = \boldsymbol{\Phi}_\mathbf{x}(k)\mathbf{P}_{\mathbf{x}\mathbf{m}_j}(k)$$

$$\mathbf{P}_{\mathbf{m}_j\mathbf{x}}(k+1) = \mathbf{P}_{\mathbf{x}\mathbf{m}_j}{}^T(k+1)$$

$$\textit{for} \quad i = 1,\ldots,N:$$

$$\mathbf{P}_{\mathbf{x}\mathbf{f}_i}(k+1) = \boldsymbol{\Phi}_x(k)\mathbf{P}_{\mathbf{x}\mathbf{f}_i}(k)$$

$$\mathbf{P}_{\mathbf{f}_i\mathbf{x}}(k+1) = \mathbf{P}_{\mathbf{x}\mathbf{f}_i}{}^T(k+1)$$

Note that to perform these operations, the CKF does not need to know anything about the structure or physical meaning of the underlying states or even the error state vector $\boldsymbol{\delta}\mathbf{X}$ that it is estimating. It just needs to be handed the matrices $\boldsymbol{\Phi}_\mathbf{x}(k)$ and $\mathbf{Q}_\mathbf{x}(k)$ which encapsulate everything about the performance and configuration of the inertial or dead-reckoning sensors, the integration algorithms applied to them and associated error propagation models. This is why the calculation of $\boldsymbol{\Phi}_\mathbf{x}(k)$ and $\mathbf{Q}_\mathbf{x}(k)$ is left to the DRU, which already has all this source information available because it has the job of reading the dead-reckoning sensor driver and integrating to keep track of the current vehicle state.

On each measurement cycle the MMU obtains one map-point-referenced measurement $\mathbf{z}_{ij}$ between the i[th] FPSE and the j[th] MPSE or one earth-referenced measurement $\mathbf{z}_{0j}$ between the j[th] MPSE and the "world". The MMU calculates and delivers the innovation $\boldsymbol{\nu}_{ij} = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}$, and the three Jacobians which form the only nonzero entries in the linearized measurement matrix:

$$\mathbf{H}_{ij} = \begin{bmatrix} \mathbf{H}_\mathbf{x} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{H}_{\mathbf{m}_j} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{H}_{\mathbf{f}_i} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \tag{0.4}$$

$$\mathbf{H}_\mathbf{x} \equiv \frac{\partial(\mathbf{z}_{ij})}{\partial(\boldsymbol{\delta}\mathbf{x}_v)} \qquad \mathbf{H}_{\mathbf{m}_j} \equiv \frac{\partial(\mathbf{z}_{ij})}{\partial(\boldsymbol{\delta}\mathbf{m}_j)} \qquad \mathbf{H}_{\mathbf{f}_i} \equiv \frac{\partial(\mathbf{z}_{ij})}{\partial(\boldsymbol{\delta}\mathbf{f}_i)}$$

with which the CKF computes its Kalman gains and updates its covariance matrix:

$$\mathbf{B}_{ij} \equiv \mathbf{P_{XX}}\mathbf{H}_{ij}{}^T = \begin{bmatrix} \mathbf{P_{xx}}\mathbf{H}_\mathbf{x}{}^T + \mathbf{P}_{\mathbf{x}\mathbf{m}_j}\mathbf{H}_{\mathbf{m}_j}{}^T + \mathbf{P}_{\mathbf{x}\mathbf{f}_i}\mathbf{H}_{\mathbf{f}_i}{}^T \\ \mathbf{P}_{\mathbf{m}_1\mathbf{x}}\mathbf{H}_\mathbf{x}{}^T + \mathbf{P}_{\mathbf{m}_1\mathbf{m}_j}\mathbf{H}_{\mathbf{m}_j}{}^T + \mathbf{P}_{\mathbf{m}_1\mathbf{f}_i}\mathbf{H}_{\mathbf{f}_i}{}^T \\ \vdots \\ \mathbf{P}_{\mathbf{f}_N\mathbf{x}}\mathbf{H}_\mathbf{x}{}^T + \mathbf{P}_{\mathbf{f}_N\mathbf{m}_j}\mathbf{H}_{\mathbf{m}_j}{}^T + \mathbf{P}_{\mathbf{f}_N\mathbf{f}_i}\mathbf{H}_{\mathbf{f}_i}{}^T \end{bmatrix}$$

$$\mathbf{S}_{ij} = \mathbf{H}_{ij}\mathbf{P_{XX}}\mathbf{H}_{ij}{}^T + \mathbf{R}_{ij} = \mathbf{H}_{ij}\mathbf{B}_{ij} + \mathbf{R}_{ij}$$

$$= \mathbf{H}_\mathbf{x}\mathbf{P_{xx}}\mathbf{H}_\mathbf{x}{}^T + \mathbf{H}_\mathbf{x}\mathbf{P}_{\mathbf{x}\mathbf{m}_j}\mathbf{H}_{\mathbf{m}_j}{}^T + \mathbf{H}_\mathbf{x}\mathbf{P}_{\mathbf{x}\mathbf{f}_i}\mathbf{H}_{\mathbf{f}_i}{}^T +$$

$$\mathbf{H}_{\mathbf{m}_j}\mathbf{P}_{\mathbf{m}_j\mathbf{x}}\mathbf{H}_\mathbf{x}{}^T + \mathbf{H}_{\mathbf{m}_j}\mathbf{P}_{\mathbf{m}_j\mathbf{m}_j}\mathbf{H}_{\mathbf{m}_j}{}^T + \mathbf{H}_{\mathbf{m}_j}\mathbf{P}_{\mathbf{m}_j\mathbf{f}_i}\mathbf{H}_{\mathbf{f}_i}{}^T +$$

$$\mathbf{H}_{\mathbf{f}_i}\mathbf{P}_{\mathbf{f}_i\mathbf{x}}\mathbf{H}_\mathbf{x}{}^T + \mathbf{H}_{\mathbf{f}_i}\mathbf{P}_{\mathbf{f}_i\mathbf{m}_j}\mathbf{H}_{\mathbf{m}_j}{}^T + \mathbf{H}_{\mathbf{f}_i}\mathbf{P}_{\mathbf{f}_i\mathbf{f}_i}\mathbf{H}_{\mathbf{f}_i}{}^T + \mathbf{R}_{ij}$$

$$\mathbf{K}_{ij} = \mathbf{B}_{ij}\mathbf{S}_{ij}{}^{-1}$$

$$\mathbf{P_{XX}}(+) = (\mathbf{I} - \mathbf{K}_{ij}\mathbf{H}_{ij})\mathbf{P_{XX}}(-)(\mathbf{I} - \mathbf{K}_{ij}\mathbf{H}_{ij})^T + \mathbf{K}_{ij}\mathbf{R}_{ij}\mathbf{K}_{ij}{}^T$$

$$= \mathbf{P_{XX}}(-) - \mathbf{B}_{ij}\mathbf{K}_{ij}{}^T - \left(\mathbf{B}_{ij}\mathbf{K}_{ij}{}^T\right)^T + \mathbf{K}_{ij}\mathbf{S}_{ij}\mathbf{K}_{ij}{}^T$$

[1] In practice the underlying states are stored as fields in the PSE structures, not in a grand unified vector as shown here.

Finally, the CKF can update it's states:

$$\widehat{\boldsymbol{\delta}\mathbf{x}}(+) = \widehat{\boldsymbol{\delta}\mathbf{x}}(-) + \mathbf{K}_{ij}^{\mathbf{x}}\mathbf{v}_{ij}$$
$$\widehat{\boldsymbol{\delta}\mathbf{M}}(+) = \widehat{\boldsymbol{\delta}\mathbf{M}}(-) + \mathbf{K}_{ij}^{\mathbf{M}}\mathbf{v}_{ij}$$
$$\widehat{\boldsymbol{\delta}\mathbf{F}}(+) = \widehat{\boldsymbol{\delta}\mathbf{F}}(-) + \mathbf{K}_{ij}^{\mathbf{F}}\mathbf{v}_{ij}$$

where $\mathbf{K}_{ij}^{\mathbf{x}}$, $\mathbf{K}_{ij}^{\mathbf{M}}$, $\mathbf{K}_{ij}^{\mathbf{F}}$ represent the appropriate partitions of the Kalman gain matrix. After each measurement cycle, the CKF transfers $\widehat{\boldsymbol{\delta}\mathbf{x}}$, $\widehat{\boldsymbol{\delta}\mathbf{M}}$, and $\widehat{\boldsymbol{\delta}\mathbf{F}}$ to the DRU, vehicle map manager and environment map manager respectively to correct the underlying states, and zeroes its own error state estimates.

2.4 Federated filtering algorithm for decoupling mapping and auto-calibration

The preceding section introduced the notation for a centralized CKF that performs simultaneous tracking, auto-calibration and map-building. The centralized filter produces optimal results, but it limits the flexibility of the architecture. The calibration and auto-mapping processes are intertwined through the cross-correlations $\mathbf{P}_{\mathbf{MF}}$ and cannot be used separately. In practice it may be desirable to first perform the auto-calibration of the vehicle's sensors using a small pre-surveyed map, then to freeze the vehicle map and set out to map the larger space using the pre-calibrated vehicle. Once the whole environment is mapped it may even be beneficial to turn off the mapping and just use the system for real time tracking. These things are cumbersome with the centralized filter. A second drawback of the centralized filter is that the computation cost of the measurement update is proportional to the square of the total number of states. If the number of auto-calibration parameters were comparable to the states in the map, decoupling them would almost halve the computation. Most importantly, the centralized filter does not scale well with large map sizes N. To overcome this, the architecture should facilitate the use of SLAM strategies based on breaking the map into submaps [3]. Removing the auto-mapping states from the SLAM filter means they don't have to be replicated in each submap.

The proposed architecture exploits the Federated Kalman filter algorithm [4], which was originally developed by Carlson to provide distributed and fault-tolerant filters for aircraft navigation systems. In that application, there was no environment map, just a vehicle with an INS and a handful of earth-referenced sensors (GPS, baro-altimeter, SAR, radar altimeter, etc). Each sensor has a few bias states, and the centralized filter stacks them all up with the vehicle INS states in the centralized state vector:

$$\mathbf{X}_{centralized} = \begin{bmatrix} \mathbf{x}^T & \mathbf{k}_1^T & \mathbf{k}_2^T & \mathbf{k}_3^T \end{bmatrix}^T.$$

In a federated system, separate local filters (LFs) are created for each of the three sensors, with state vectors:

$$\mathbf{X}_{LF1} = \begin{bmatrix} \mathbf{x}_1^T & \mathbf{k}_1^T \end{bmatrix}^T \quad \mathbf{X}_{LF2} = \begin{bmatrix} \mathbf{x}_2^T & \mathbf{k}_2^T \end{bmatrix}^T \quad \mathbf{X}_{LF3} = \begin{bmatrix} \mathbf{x}_3^T & \mathbf{k}_3^T \end{bmatrix}^T$$

Because there are redundant copies of the vehicle state vector **x** in each LF, the total information about **x** has to be split up amongst the LFs. This is done by multiplying **Pxx**(0) by a factor γ (in this case $\gamma = 3$) and initializing all the LFs with this inflated covariance. Each LF then performs a time update, using $\mathbf{Q}_{\mathrm{LFi}} = \gamma \mathbf{Q}$, and then processes the measurement from only its own sensor. Finally, a master filter (MF) fuses the common states of all the LFs using the information fusion form of KF update:

$$\mathbf{P}_{\mathbf{xx}Fused} = (\mathbf{P}_{\mathbf{xx}LF1}^{-1} + \mathbf{P}_{\mathbf{xx}LF2}^{-1} + \mathbf{P}_{\mathbf{xx}LF3}^{-1})^{-1}$$
$$\hat{\mathbf{x}}_{MF} = \mathbf{P}_{\mathbf{xx}Fused}(\mathbf{P}_{\mathbf{xx}LF1}^{-1}\hat{\mathbf{x}}_{LF1} + \mathbf{P}_{\mathbf{xx}LF2}^{-1}\hat{\mathbf{x}}_{LF2} + \mathbf{P}_{\mathbf{xx}LF3}^{-1}\hat{\mathbf{x}}_{LF3})^{-1}$$

At this point, all the information about x has been transferred into the MF, whose covariance $\mathbf{P}_{\mathbf{xx}Fused}$ represents the true system covariance, while all the information about each sensor's biases remains in its own LF. To start the next cycle, the vehicle information from the MF gets split up and distributed back to the LFs using the Fusion Reset equations. If the MF has dimension less than the centralized filter, as just described, there will be some loss of optimality, but in the examples shown the difference is not noticeable [4].

In the Federated filter, each measurement is processed by just one LF. However, the problem posed in Section 2.3 has an extra complication. If the simultaneous localization and autocalibration (SLAC) process is assigned to one LF and the simultaneous localization and mapping (SLAM) process to another LF, then each measurement will depend on states in both filters. We solve this by using the Principle of Measurement Reproduction [5], which states that a measurement $\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{v}, \quad \mathbf{v} \sim N(0, \mathbf{R})$ is exactly equivalent to two repetitions of the measurement, each with twice the original noise:

$$\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{H} \\ \mathbf{H} \end{bmatrix}\mathbf{x} + \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix}, \quad \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} \sim N\left(\mathbf{0}, \begin{bmatrix} 2\mathbf{R} & \mathbf{0} \\ \mathbf{0} & 2\mathbf{R} \end{bmatrix}\right)$$

Thus, each measurement is sent to all the LFs which contain a PSE involved in that measurement, and each LF multiplies the measurement noise matrix by γ so that the information is divided up between the LFs. Each LF then acts independently, as if the measurement were it's own, and bumps up it's noise to account for states not included within itself, just like an ordinary stand-alone filter tuned to account for unmodeled states by standard reduced-order KF design methods.

Figure 1 summarizes the architecture described above. The underlying states are maintained in the DRU, the vehicle map manager, and the environment map manager. The SLAC filter estimates errors in the underlying MPSE poses and biases, and sends the corrections vector $\boldsymbol{\delta}\mathbf{M}$ back to the vehicle map manager, which applies the corrections to the appropriate PSE fields with appropriate translation of orientation format. In like fashion, the SLAM filter sends error correction estimates back to the environment map manager. A third LF for simple localization only (LO) has been added, so that if the user or the system decides that both auto-calibration and mapping are done, the SLAC and SLAM filters can be turned off. Any one, two or three of the LFs may be active at a given time, with γ set to the number active. Figure 1 only shows what data is passed between the modules during active tracking. Much of the flexibility of the architecture stems from how it detects hardware and initializes itself, which will be touched on briefly in the following sections.

2.5 Dynamic Reference Unit (DRU)

At initialization, the DRU finds out from the internal sensor driver whether there are inertial sensors, wheel encoders, or no dead-reckoning sensors. Accordingly, the DRU determines

whether tracking is desired in 3D or 2D, and chooses an appropriate integration algorithm and error model. Next, the DRU asks the driver for a number of standard parameters appropriate to the integration model. For an IMU, for example, these would include white noise and random walk amplitudes and r.m.s. initial uncertainty estimates for gyro and accel biases, ramps, misalignments, scale factors, nonlinearities, etc. Based on the sizes of these initial uncertainties, the DRU decides whether to create CKF error states for each parameter.

Smart sensor modules designed for this architecture might use an IEEE 1451.2 interface [6], which specifies a transducer electronic data sheet (TEDS) resident in EEPROM that identifies to the driver the types of sensors present and provides calibration data. The driver then passes along to the DRU the part of the information it needs, and compensates for sensor biases, scale factors and misalignments for which factory calibration data is stored. For non-IEEE 1451.2 sensors, the driver must emulate this process with user-supplied files.

2.6 Measurement Management Unit (MMU)

The MMU is a complex system with many functions:

Initialization

1. Consult status server to discover all available sensors and pre-known targets (use IEEE 1451.2 TEDS or user-provided configuration and calibration files to auto-identify devices and their parameters.)
2. Pass data to vehicle and environment maps to initialize PSE states.

Acquisition

1. Request initial sensor scan, download reference positions from map.
2. Assemble set of measurements and reference positions, acquire pose.

Tracking

1. Identify sensors ready and targets in range. Status server may consult a regional server if there are resources shared with other vehicles.
2. Select sensor/target pair yielding greatest information gain.
3. Predict measurement, calculate search window, send trigger.
4. Retrieve measurement and perform data association.
5. If passed, send innovation and measurement model matrices to CKF.
6. If failed, sensor is MPSE, and target is plausible, initialize new FPSE.

Due to space limitations, we'll focus on tracking steps 3-5. Here is where we really exploit the sensor/target abstraction of Eq. (0.2) to partition the work of preparing the measurement matrices $\mathbf{H}_{\mathbf{x}}$, $\mathbf{H}_{\mathbf{mj}}$, and $\mathbf{H}_{\mathbf{fi}}$ into generic and model-specific parts, keeping the generic operations inside the MMU and pushing the sensor/target details out into the driver. As a result, the architecture is extensible by outside parties: a sensor manufacturer who understands their own product and can model its output, but knows nothing of Kalman filtering or SLAM theory, could follow some instructions and write a driver that would enable the sensor to work with the sensor fusion core and be optimally fused with the inertial sensors and any other aiding sensors that are attached.

The key to this partitioning is to let the MMU calculate the predicted pose, $\hat{\boldsymbol{\lambda}}_{st}$, of the target w.r.t. the sensor by first calculating the pose of the j[th] MPSE w.r.t. the i[th] FPSE, $\hat{\boldsymbol{\lambda}}_{ij} = \boldsymbol{\lambda}(\hat{\mathbf{x}}, \hat{\mathbf{m}}_j^{\chi}, \hat{\mathbf{f}}_i^{\chi})$, which is a function of the poses of the vehicle, the MPSE, and the FPSE. If the measurement is outside-in (oi), $\hat{\boldsymbol{\lambda}}_{st} = \hat{\boldsymbol{\lambda}}_{ij}$ and if it is inside-out (io), $\hat{\boldsymbol{\lambda}}_{st} = \hat{\boldsymbol{\lambda}}_{ji}$. The MMU sends $\hat{\boldsymbol{\lambda}}_{st}, \mathbf{k}_s$ and $\mathbf{k}_t$ to the mfg.-specific model server, which responds with a predicted measurement, $\hat{\mathbf{z}}_{st}$, and the Jacobians $\mathbf{H}_{\lambda_{st}}$, $\mathbf{H}_{\mathbf{k}_s}$, $\mathbf{H}_{\mathbf{k}_t}$ associated with measurement equation (0.2). The MMU uses the chain rule to produce the H matrices needed by the CKF:

$$\mathbf{H}_{\mathbf{x}} = \mathbf{H}_{\lambda_{st}} \boldsymbol{\Lambda}_{\mathbf{x}}$$

$$\mathbf{H}_{\mathbf{m}} = \begin{bmatrix} \mathbf{H}_{\mathbf{m}^{\chi}} & \mathbf{H}_{\mathbf{m}^{\mathbf{k}}} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{\lambda_{st}} \boldsymbol{\Lambda}_{\mathbf{m}^{\chi}} & \left(dir == io ? \mathbf{H}_{\mathbf{k}_s} : \mathbf{H}_{\mathbf{k}_t} \right) \end{bmatrix}$$

$$\mathbf{H}_{\mathbf{f}} = \begin{bmatrix} \mathbf{H}_{\mathbf{f}^{\chi}} & \mathbf{H}_{\mathbf{f}^{\mathbf{k}}} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{\lambda_{st}} \boldsymbol{\Lambda}_{\mathbf{f}^{\chi}} & \left(dir == io ? \mathbf{H}_{\mathbf{k}_t} : \mathbf{H}_{\mathbf{k}_s} \right) \end{bmatrix}$$

where

$$\boldsymbol{\Lambda}_{\mathbf{x}} = \left. \frac{\partial \boldsymbol{\lambda}_{st}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}, \hat{\mathbf{m}}_j^{\chi}, \hat{\mathbf{f}}_i^{\chi}} \quad \boldsymbol{\Lambda}_{\mathbf{m}_j^{\chi}} = \left. \frac{\partial \boldsymbol{\lambda}_{st}}{\partial \mathbf{m}_j^{\chi}} \right|_{\hat{\mathbf{x}}, \hat{\mathbf{m}}_j^{\chi}, \hat{\mathbf{f}}_i^{\chi}} \quad \boldsymbol{\Lambda}_{\mathbf{f}_i^{\chi}} = \left. \frac{\partial \boldsymbol{\lambda}_{st}}{\partial \mathbf{f}_i^{\chi}} \right|_{\hat{\mathbf{x}}, \hat{\mathbf{m}}_j^{\chi}, \hat{\mathbf{f}}_i^{\chi}}.$$

3. Implementation

Over the past year, a substantial portion of this architecture has been implemented using Simulink, DSP Blockset and Real-Time Workshop. As a test case, we have built a wearable vision/inertial hybrid self-tracker based on this architecture. The system consists of an InterSense InertiaCube[2] IMU and a DSP-based smart camera, both interfaced through serial ports to a 100 x 150 mm single-board computer running the SFC software. Figure 2 shows a block diagram of the system. Figure 3 is a photograph of the camera and IMU mounted on a head-mounted display looking up at a ceiling full of fiducial marks.

The InertiaCube[2] is a tiny (32x29x24mm) low-power (0.5W) MEMS IMU containing 3 rate gyros, 3 linear accelerometers, and 3 magnetometers (not used in this example).

The smart camera integrates a 640x480 CCD with a DSP that performs fiducial extraction image processing code at a 24 Hz frame rate, and communicates the final target coordinates to the SFC through a serial port. During acquisition, the image processor scans the entire image for candidate fiducials, attempts to read the barcode for each one, and returns the best

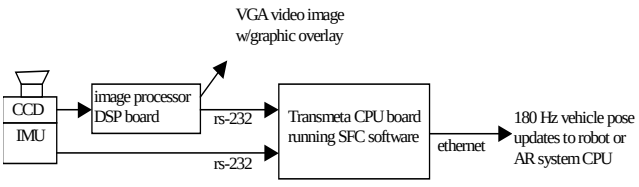

Figure 2: Block diagram of example implementation

Figure 3: IMU (blue) & camera (white) on HMD

four to the SFC for initial pose recovery. During tracking the SFC predicts the location of fiducial centers within a few pixels, so only a search box of typically 30x30 pixels needs to be processed, which can be completed during the 10 ms exposure of the subsequent image. The smart camera employs image processing techniques which make it quite robust in the face of varied lighting conditions [7], and the system gains further robustness because of the hybrid inertial-vision tracking methodology. We have found that it can keep on tracking well even when almost all of the fiducials are occluded by a hand waving in front of the camera. The accompanying video on the conference CD shows the CCD video output with graphical overlay by the image processing board showing a box where it searches for each fiducial and a "+" where it finds it during a simple tracking experiment using about 100 fiducials of the type shown in Figure 3.

4. Discussion

A broad-brush outline has been presented for a new sensor fusion architecture designed to facilitate rapid development and reconfiguration of systems that perform tracking, auto-calibration, and auto-mapping using a wide variety of different types of sensors. While we have not yet implemented all the described features of the architecture (notably the SLAC filter and the initial self-discovery of all the attached hardware devices), we are so far very encouraged by the modularity of the architecture. This paper was written and a top-level Simulink diagram similar to Figure 1 was created before any implementation of the subsystems began. Thus we have had a working system from a very early point, and then successively refined it by adding more and more algorithmic complexity within the subsystems, without ever needing to re-organize and rebuild the system. No re-organization is anticipated, as it appears we will accomplish most of the stated goals by filling in the missing components of the architecture, and interfacing different sensors, such as a computer vision system that can recognize natural features instead of printed paper fiducials.

There is much work left to do to fill in the details of the architecture and achieve some of its loftier goals, including scalable mapping algorithms and "Plug&Track" self-configuration. The Federated filter has been selected as a nice framework for experimentation with algorithms that involve a group of submaps [3] of which only a subset are active at a time. However, more work is required to identify the most efficient yet consistent algorithms for swapping the active set of submaps, or to find algorithms for combining and repartitioning submaps dynamically. In particular, we would want to explore the possible use of Covariance Intersection [8] and relative maps [10]. Also, future work must address the issues of how to merge the submaps generated by different vehicles at a regional coordination node, even though they may not be based exclusively on independent information sources, and how to re-integrate these merged maps back to the vehicles.

5. References

[1] Smith, R., Self, M., & Cheeseman, P. (1987). A Stochastic Map for Uncertain Spatial Relationships. Fourth Intl. Symposium on Robotics Research, MIT Press.

[2] Brown, R.G. & Hwang, P.Y.C. (1992). Introduction to Random Signals and Applied Kalman Filtering. New York: John Wiley & Sons, Inc.

[3] Leonard, J.J., & Feder, H.J.S. (2000). A Computationally Efficient Method for Large-Scale Concurrent Mapping and Localization. Robotics Research: The Ninth International Symposium, J. Hollerbach & D. Koditschek (Ed.s) London: Springer-Verlag.

[4] Carlson (1990). Federated Square Root Filter for Decentralized Parallel Processes. IEEE Transactions on Aerospace and Electronic Systems, AES 26:3

[5] Tupysev, V. A. (1998). A Generalized Approach to the Problem of Distributed Kalman Filtering. AIAA Guidance, Navigation & Control Conference, AIAA 98-4309, Boston

[6] IEEE Instrumentation and Measurement Society (1997). IEEE Std 1451.2-1997: Smart Transducer Interface for Sensors and Actuators-Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats.

[7] Naimark, L. & Foxlin, E. (2002) Circular Data Matrix Fiducial System and Robust Image Processing for a Wearable Vision-Inertial Self-Tracker. Submitted to IEEE/ACM Intl. Symp. Mixed and Augmented Reality, Darmstadt, Sept 30-31.

[8] Julier, S.J. & Uhlmann, J.K. (1997). A Non-Divergent Estimation Algorithm in the Presence of Unknown Correlations. Proc. of the American Control Conference.

[9] Castellanos, J.A., Tards, J.D. & Schmidt, G. (1997). Building a Global Map of the Environment of a Mobile Robot: The Importance of Correlations. Proc. IEEE ICRA.

[10] Csorba, M., Uhlmann, J.K. & Durrant-Whyte, H.F. (1997). A Sub Optimal Algorithm for Automatic Map Building. In Proceedings of the American Control Conference, Albuquerque, NM, pp. 537-541

[11] Guivant, J. & Nebot, E. (2001). Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation. IEEE Trans. on Robotics and Automation 17(3), pp. 242-257