TODO: Explain how the graph attaches to the trajectory.

## Discussion of Quadratic Forms

In this form, an algorithm that models a vector-valued measurement  at the instant  with normally-distributed error  would take the shape of a paraboloid centered at the mean . Its Hessian would be uniformly , where  is the covariance of the measurement noise projected on to the state space.

## Value to Users

Our modular approach serves the interests of several types of users. Here is a rough breakdown of users and their requirements:

System Integrator: i) Ability to easily swap sensors, simulators, and datasets; ii) Ability to easily swap measurement algorithms; iii) Tools to evaluate the navigation accuracy of combinations of sensors and algorithms.

Sensor Specialist: i) Abstraction of real and simulated sensors for offline testing; ii) Ability to customize for their own sensor hardware; iii) Automatic integration with sensors and algorithms developed by others.

GNC Specialist: i) A clear place to insert a canonical linear or nonlinear model of vehicle dynamics; ii) A means to access the body position, orientation, velocity, and rotation rates at any time.

## TrajectorySim

This component provides a method to create arbitrary smooth trajectories from a series of control points. It interpolates position using a Cardinal Spline and it interpolates orientation using a special form of a Hermite Spline for quaternions.

## MiddleburyData

This class automatically downloads images from the Middlebury College Stereo Vision Research Page from their website [5]. These are colorful, well-illuminated, high-resolution images from a perspective camera that translates to the right.

## ThesisDataDDiel

This class automatically downloads images and inertial data from David D. Diel’s thesis work from the web. This data includes both real and simulated imagery from fisheye cameras and three different grades of inertial sensors. Four scenes are available from this data source. The scenes comprise: i) grayscale video from a high-definition progressive scan camera using a calibrated fisheye lens; ii) inertial data from the MicroStrain 3DM-GX1; and iii) ground truth trajectory with 1mm relative position accuracy from a linear motion gantry.

## MacBookBuiltInSensors

SSCI has selected MacBook hardware to demonstrate real time trajectory optimization. This choice was primarily driven by the availability of consumer hardware that supports a built-in camera and inertial sensor, sufficient memory, and a relatively fast CPU for image processing. The key feature is the inertial sensor. According to Wikipedia, Apple uses the Kionix KXM52-1050 three-axis accelerometer chip, with dynamic range of +/- 2g and bandwidth up to 1.5 kHz. We are able to access this sensor through an adaptation of the BSD licensed SeisMac library. This class provides access to data from the built-in camera and three-axis accelerometer on most MacBook laptops. It depends on VLC Media Player for OS X, which is free software.

## Default implementations

The TOMMAS framework provides a default implementation for each class, as described below:

* DynamicModel – The default dynamic model places the body frame at the equator and prime meridian, with its axis orientation aligned with the ECEF frame. Its domain begins at the initial time supplied to the class constructor and ends at +Inf. It accepts no parameters of any kind. (Example: dynamicModel = tom.DynamicModel.create('tom', tom.WorldTime(0), ' ');)
* Measure – The default measure has no data and constructs no graph edges. (Example: measure = tom.Measure.create('tom', tom.WorldTime(0), ' ');)
* Optimizer – The default optimizer does nothing and provides no solutions. (Example: optimizer = tom.Optimizer.create('tom');)
* DataContainer – The default data container lists no sensors from which to get data. Neither does it contain a reference trajectory. All data containers are in the antbed namespace. (Example: dataContainer = antbed.DataContainer.create('antbed', tom.WorldTime(0));)

## MiddleburyTemple data container

UCF has selected the Middlebury Temple data set for multi-frame image data [4]. This was necessary in order to move forward without publicly released data from AFRL.

SSCI has developed a non-trivial data container for the Middlebury Temple data that allows a series of poses to be selected to represent the body trajectory. A smooth 6-DOF trajectory is fit through those poses to provide ground truth for the data container. Quaternion interpolation is handled using a sophisticated algorithm derived from [1][3]. We expect this to be a highly valuable data set for demonstrating results to the computer vision community.

## Fast Point-Based Measure:

The scene geometry and the ego motion are embedded in the visual information provided from a camera. Instantaneous motion estimation methods rely on a differential model to recover the velocity of the camera based on the optical flow. Point-based motion estimation methods are generally more recent and have been proved to perform better than instantaneous methods, since the latter requires a high-frame rate such that several approximations can be valid in the differential model???.

Alternatively, the matched sets of points  can also be directly used to estimate the residual without the need for complete motion estimation. This measure can be expressed in terms of optical flow. In that, we use the hypothesis relative translation  and relative rotation  to simulate a translational optical flow field, and a rotational optical flow field  respectively. Consequently, the matched points are represented as flow field (note that can also be obtained by regular optical flow estimation methods). The rotational field is then subtracted from the optical flow to obtain a rotation-free flow. The residual of the hypothesis can then be computed as

,

where  is the vector orthogonal to . It can be shown that the residual in equation 13 is equivalent to 11.

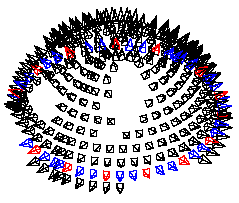


Figure . An example image and other selectable poses from the Middlebury Temple data set [4].

## GlobalSatData < DataContainer

Global Positioning System (GPS) devices, in conjunction with inertial sensors, have become the primary sources of location information used in modern navigation devices. While much of our current work focuses on navigation performance in GPS-denied environments, it is still important to provide users of TOMMAS the ability to use global position information, not only to test the performance of algorithms and aiding sensors in conjunction with GPS, but also to provide comparisons with a baseline GPS-enabled navigation system. Therefore, during this reporting period, we have developed an interface for a global position sensor, which is present in the Google Code repository.

In order to test the global position interface, we also developed a simulator for the Globalsat BU-353 SIRF II GPS receiver. The simulator takes a user-provided trajectory and, based on user queries through the global position sensor interface, returns estimates of the sensors position at the requested time instances. The primary challenge in developing the simulator is to characterize the noise in the sensor. Instead of attempting to model the noise characteristics of the sensor, we simply use real sensor data to find time-varying errors in lat-long measurements. These errors are then added to the user-provided trajectory to simulate receiver measurements. The real sensor data has been collected using an actual Globalsat BU-353 receiver, by placing it in a single location and recording measurements over a period of several minutes. GPS measurements at a single location over approximately a minute are shown in Figure 3.

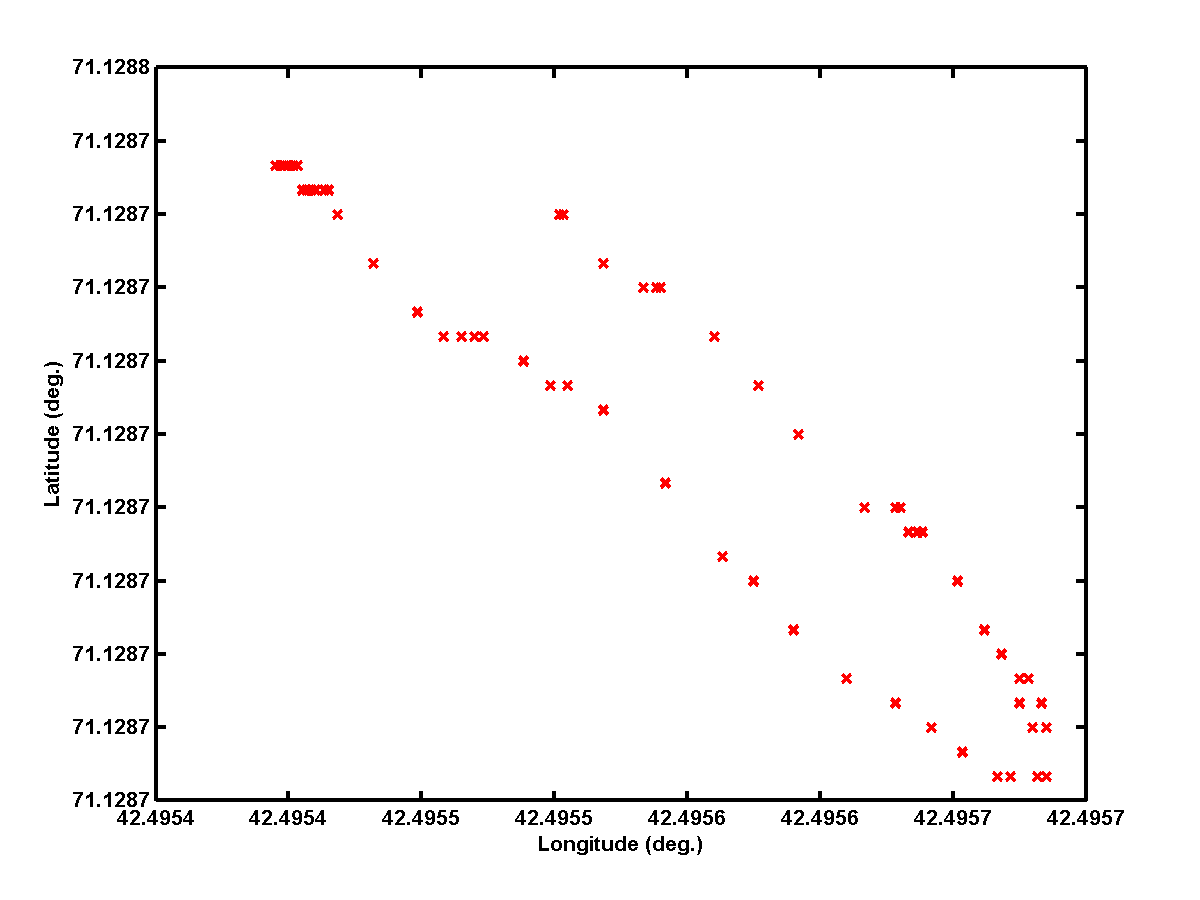


Figure . Lat-Long measurements from a stationary Globalsat BU-353 over a one minute period.

As part of the interface, a user may also request a GPS location at a time at which no GPS data is recorded. In such cases, it may be necessary to interpolate between existing GPS measurements.

## Interpolation Reference

All TOMMAS trajectories must satisfy the  continuity criterion, meaning that they are differentiable and have a continuous first derivative. In order to interpolate the trajectory, while adhering to this requirement, we use a Cardinal Spline function. The Cardinal Spline is a special case of the more general Cubic-Hermite Spline (Cspline) interpolation in which, given two points, *pk*and *pk+*1, and two tangents, *mk* and *mk+*1, at the beginning and end, respectively of a closed time interval [*Tk*, *Tk+*1], the point at some time, *Tk*<*T*<*Tk+*1 is given by [1]:

.



and  is in the time scaled to the interval between 0 and 1 for each segment. The Cardinal Spline is then the specific case where the tangents are computed using the following method [4]:



In the above equation, *c* represents a tension parameter between 0 and 1, which is set, by default, to 0 (i.e., the Catmull-Rom spline), but can be modified by the user via a configuration file.

Figure 4 shows an example of the cardinal spline interpolation for a two-dimensional trajectory, given only four points on the trajectory, that represent the four corners of a square. Note that smoothness of the interpolated trajectory, specifically while turning around the corners of the square, in order to maintain C1 continuity.

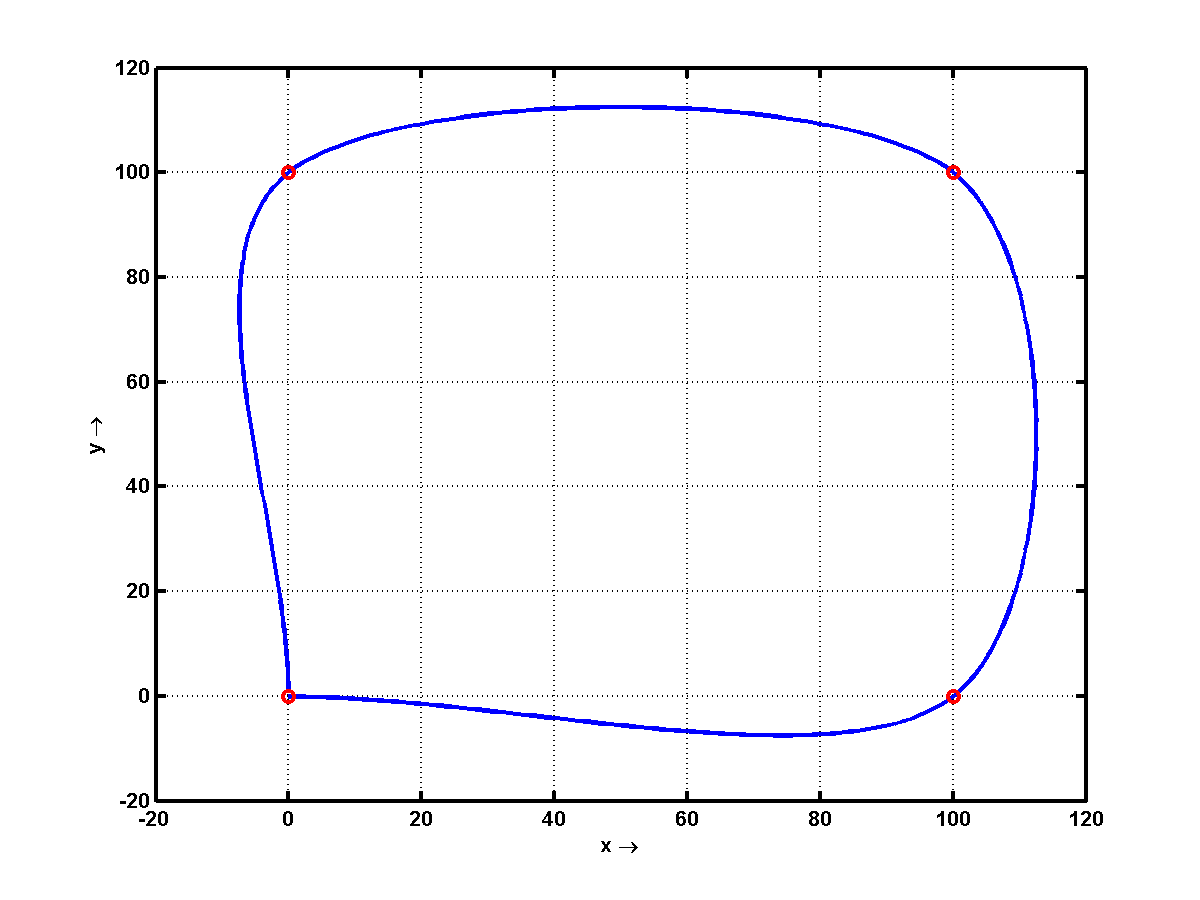


Figure . Example of cardinal spline interpolation given five points in clockwise order starting at the bottom left, where the first and last point are at the same location.

## Timing and Synchronization

A common problem that arises from combining multiple asynchronous sensors is maintaining indices of data elements that are ordered consecutively in time. Our solution is to introduce a communication interface called the Common Observation Model that defines the following functionality: i) Query the first and last indices in a consecutive list of data elements from an individual sensor; ii) Read a data element given a sensor index and a data index specific to that sensor; iii) Refresh the first and last data indices associated with an individual sensor. Reordering of data packets for an individual sensor occurs during a call to its refresh function, but between refresh requests, the available data is held static. This means that downstream processes can peek at the time stamps of the available data elements and choose which one to process next without modifying the stack.

# References

1. Paul Bourke. *Interpolation Methods*. December 1999. http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/interpolation/ Retrieved 01-18-2010.
2. Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. “A general construction scheme for unit quaternion curves with simple high order derivatives”. In proceedings of SIGGRAPH, pp. 369-376, 1995.
3. Myoung-Jun Kim and Myung-Soo Kim. “A Compact Differential Formula for the First Derivative of a Unit Quaternion Curve”. In Journal of Visualization and Computer Animation, pp. 43-57, 1996.
4. Microsoft Developers Network. *Cardinal Splines*. Retrieved 01-18-2010.
5. S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. “A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms”. CVPR 2006, vol. 1, pages 519-526.

## Modeling the Residual

In order to obtain a better data-driven residual representation, the residual  resulting from the discussed measures is compared to its characteristic distribution as determined from training data. For instance, we have modeled the tracking error in the Middleburry dataset using a single Gaussian, a Gaussian Mixture Model, and Kernel Density Estimation (KDE). In order to select the most appropriate model, we divided our dataset into two sets for training and testing. The training data is used in the model learning, and then the model response against the testing data is compared to the actual groundtruth response using the sum of squared error (SSE). Figure 1 illustrates the obtained results. In our experiments, we found that even as simple as a single Gaussian is appropriate for modeling the error, and that the SSE decreases with increasing the number of Gaussian components, while being generally close to the SSE obtained when using the KDE. In our framework, we use a single Gaussian component in order to avoid over-fitting. Given the residual  for each feature pair and the error model (eg.), we compute the normalized Sum of Squared Differences (SSD)

.

As discussed in the previous section, follows a chi-square distribution; therefore, the final data likelihood is given as

.

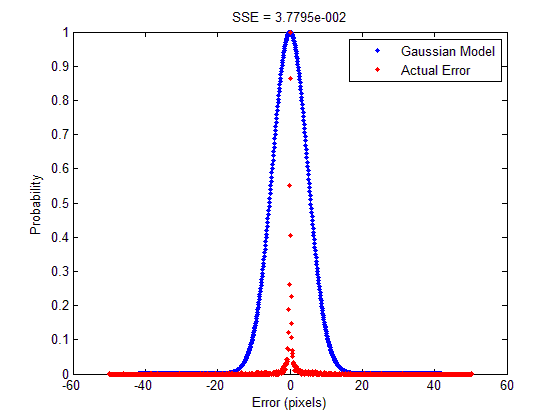


Figure . KLT feature tracking error distribution given Middlebury Art data. The blue curve represents the learned model, and the red dots represent the actual data.