GFR Team 61/62 Technology Review

Nicholas Milford

November 9, 2018

1 Problem Definition

Autonomous vehicles require robust representations of the world in order to make correct driving decisions. The driverless vehicle must utilize sensor input from lidar, camera, accelerometer and GPS in order to construct a virtual representation of the world around it as well as localize itself within that world. Due to the inherent error and low sampling rates of the sensors, high speed motion can cause data about the environment to be sparse and noisy. Robust statistical analysis is needed to evaluate the confidence of the vehicle's internal representation of the world around it. Our team is tasked with constructing the vision, mapping, and localization systems for a driverless formula style racecar for the Global Formula Racing (GFR) team. The driverless vehicle is expected to compete in the Formula SAE (FSAE) competition in August 2019.

I'm working on sensor filtering for odometry. This involves filtering multiple sensor sources and using that data along with known kinematics and controller inputs to determine where the car is. The output of this program will then be used in localization and motion planning. The major decisions for this part of the project can be broken down into 3 segments: two design decisions, and one implementation decision. The first is sensor fusion, where data from multiple sensors will be used to construct a best guess of the state of the vehicle. An existing particle filter library will be used to implement the sensor fusion, and so possible options will be considered second. Finally the filtered data will be cached and an interpolation program will be used to determine vehicle states between actual sensor data points for use in mapping and localization. Because the odometry data structure itself will be relatively small in size, efficient caching methods are not necessary, leaving only an interpolation method to be decided.

2 Sensor Fusion

Because the systems on the vehicle are nonlinear and can't be approximated well by a linear function across the entire range of inputs, a nonlinear filter is needed. This rules out the normal Kalman filter because linearity assumptions are made through its use of strictly gaussian distributions, but nonlinear variants of it have been developed, which will be considered alongside entirely different statistical filtering methods. The most important criteria is accuracy, followed by convergence time and stability.

2.1 Available Technologies

2.1.1 Extended Kalman Filter

The extended Kalman filter is the defacto standard for nonlinear filtering and state estimation. It linearizes the nonlinear equations in the model by using the first derivative of the taylor series [9]. Advantages include inheriting all benefits of the normal kalman filter, which includes proof of convergence and stability. Disadvantages are increased error margins, because while the gaussian covering the actual probabilities is more accurate and assumes higher error, it still may not match the shape of the distribution of the actual system.

2.1.2 Unscented Kalman Filter

The unscented Kalman filter is another nonlinear variant of the standard Kalman filter. Instead of taking a linear approximation of the system, the unscented Kalman filter uses an unscented transform, which samples points around the probability distribution of the input and then applies the nonlinear system to those samples, resulting in an output gaussian distribution [5]. Unscented Kalman filters have better performance for more highly nonlinear systems than extended Kalman filters because it does not linearize the underlying model [4]. It still suffers from all the same drawbacks as the extended Kalman filter: it assumes a gaussian input and output distribution over each variable in the system.

2.1.3 Particle Filter

The simplest description of the difference between a Kalman filter (or its nonlinear variants) and a particle filter is that a Kalman filter is a probabilistic process while a particle filter is a stochastic process. While the Kalman filters use probability theory to iteratively reduce error in a single measurement model, particle filters work by simulating many individual particles in the system, and re-sampling those particles in accordance with how likely each particle is to be the true measurement. The particle filter solves our main issue with Kalman filters: because there is no math being done on the expected values and variance of the system, the type of probability distribution that the system best fits does not matter. This results in the best approximation, and therefore highest accuracy of the three candidates in highly nonlinear models [2]. The disadvantages of the particle filter are increased computation time because many particles are needed for a robust measurement, and the randomness of the process removes any guarantee of convergence time, however that issue may be solved simply by using more particles.

2.2 Selected Technology

For the odometry filter, a particle filter will be used as a first pass to fuse all the sensor data, and an extended Kalman filter will then be used to further refine the odometry measurement before being output to other systems.

The reasoning behind using the particle filter for only odometry is because sensor information from three accelerometers each at different locations on the car and a GPS will be difficult to model. In addition, absolute knowledge of expected values and errors in accelerometer readings is vital, because any error in the measurement system will be double integrated to get position.

After a rough model of the car has been built by the particle filter, the extended Kalman filter will provide the numerical stability necessary to output reliable information to other systems. The dynamics of the vehicle are all well known and linearizable enough, therefore the additional complexity of the unscented Kalman filter is not necessary.

3 Filter Libraries

Because of the prevalence of particle filters and Kalman filters in industry, many implementations already exist in the languages we will be using (C++/Python). In addition, most Bayesian filtering libraries provide both filters. It is therefore wasteful to try to re-implement either of these processes ourselves, and so we must consider which Bayesian filtering library to use. Selection criteria will include interface complexity and available documentation.

3.1 Available Technologies

3.1.1 Bayesian Filtering Library (BFL)

The Bayesian Filtering Library for C++ is the standard for bayesian filtering in ROS [3]. BFL is general, yet complex, largely due to the features and limitations of C++: nonlinear system and measurement distribution models require their own classes, and so the code becomes fragmented across files. Due to this, it will take longer to make a simple, usable first solution.

3.1.2 PyBayes

PyBayes is “an object-oriented Python library for recursive Bayesian estimation (Bayesian filtering) that is convenient to use” [7]. PyBayes implements particle filters, but not extended Kalman filters, so it . There are options for compiling the Python library into a binary file for greater speed if necessary, so the performance concerns of using python can be mitigated.

3.1.3 FilterPy

FilterPy is another python filtering library with all of the same features, in addition to an extended Kalman filter implementation [6]. It is more popular and therefore likely better tested and more robust than PyBayes, however support for python 2.7, which we will be using, will be dropped in December 2018 at the latest.

3.2 Selected Technology

For the particle filter implementation, PyBayes will be used. Due to the proximity of the preliminary design review, it is very important to have something that works by November 17, and an easier to use Python library has more value in that regard than a more complex C++ library. FilterPy will not be used because of the lack of support for development continued into 2019.

Because PyBayes does not implement the extended Kalman filter, a second choice needs to be made. Either BFL or FilterPy can be chosen, because each of these will be implemented as separate ROS nodes, which run as independent process and so can interoperate between languages. Due to the issues stated before, BFL will be chosen over FilterPy. The time constraints are looser for this portion of the project because it is just data refinement, and the parameters to the extended Kalman filter are less general, reducing the effects of the drawbacks to using C++ and BFL.

4 Interpolation

Because of the decentralized and asynchronous nature of the publisher/subscriber architecture implemented in ROS, many data sources may provide that data at different time intervals. As accuracy is vital to the performance of the localization and mapping system, which uses odometry data, it is necessary to infer the intermediate vehicle state given multiple data points. There is plenty of research on numerical solutions to differential equations, however most only involve known initial conditions. The known end conditions in the interpolation problem provide unique constraints and possibilities to the problem. For example, whereas stability is normally an important topic in numerical approximation, knowledge of the end point can correct for an approximation that has diverged from the real value.

4.1 Available Technologies

4.1.1 Linear Multistep Methods

Linear multistep numerical methods involve taking single steps through the domain using a linear combination of previous states. The simplest example is the Euler - or first order - method, which only uses the previous state. Higher order methods, such as the Adams-Bashforth method, are more accurate while not significantly increasing computational complexity [8].

4.1.2 Runge-Kutta Methods

Runge-Kutta methods are the other main family for approximating differential equations. Instead of each evaluation resulting in a full step, a Runge-Kutta implementation will refine values using partial steps before calculating the next full step. This results in better accuracy over time. Its additional complexity allows the Runge-Kutta method to be parameterized further, but requires additional work to verify the parameters chosen. The standard Runge-Kutta method is the 4th order explicit Runge-Kutta algorithm (RK4), which would be used unless extremely specific requirements need otherwise [1].

4.1.3 Spline Interpolation

Because the problem is interpolation and not extrapolation, a possible solution is to use the kinematic relationships of the vehicle model to create a basic motion profile between the known endpoints using a cubic spline. A cubic spline is defined by a set of points and derivatives, and generates a smooth curve that crosses through those points with their respective slopes [10]. The advantage of this method over the others mentioned is that the desired intermediate value can be calculated in constant time (that is, regardless of the step size). The disadvantage is the increased complexity necessary to develop the spline parameters, and a more drastic failure if implemented improperly.

4.2 Selected Technology

Vehicle state interpolation will be done with the Runge-Kutta method. The slightly increased accuracy compared to the linear multistep is worth the slightly increased implementation complexity. Spline interpolation was not chosen because of the increased design and implementation complexity for only marginal computational gains, and because it is not proven to be more accurate.

References

[1] Erik Cheever. Fourth order runge-kutta, 2017.

[2] Jesús Fernández-Villaverde. Kalman and particle filtering. In *Macroeconometrics and Time Series Analysis*, pages 151–157. Palgrave Macmillan UK, 2010.

[3] Klaas Gadeyne. BFL: Bayesian Filtering Library. `http://www.orocos.org/bfl`, 2001.

[4] Simon J. Julier and Jeffrey K. Uhlmann. New extension of the kalman filter to nonlinear systems. In Ivan Kadar, editor, *Signal Processing, Sensor Fusion, and Target Recognition VI*. SPIE, jul 1997.

[5] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, mar 2004.

[6] Roger R Labbe. Filterpy, 2018.

[7] Matej Laitl. Pybayes, 2012.

[8] Varun Shankar. Linear multistep methods i: Adams and bdf methods, 2016.

[9] Harveen Singh. Extended kalman filter: Why do we need an extended version?, 2018.

[10] Eric W Weisstein. Cubic spline, 2018.