Kalman Filtering with STDR Simulator and ROS

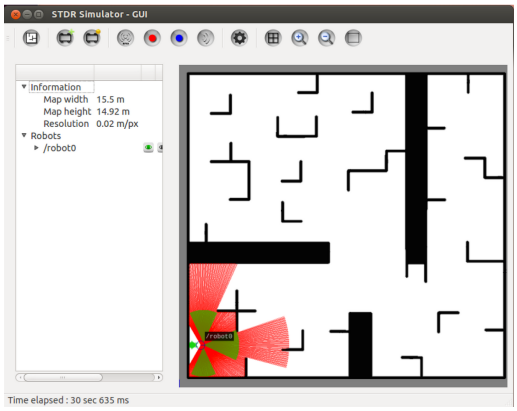
Luc Alexandre Bettaieb
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, Ohio 44106

Using software written for the Simple Two-Dimensional Robot Simulator (STDR) running with the Robot Operating System (ROS) as middleware, a Kalman filter was implemented to localize a simulated mobile robot equipped with a laser distance scanner. The Kalman filter used a sensor model using data gathered from measurements tolerance-matched against recorded data with respective states and a velocity model based off of equations of linear, zero-acceleration motion. Using these two models and their associated errors, the Kalman filter was able to accurately estimate the robot's position and orientation.

I. Background

The Robot Operating System, ROS, provides robotics hobbyists, researchers, and industry-workers with a modular framework for constructing robust software for use on robotic systems, with emphasis on code reusability. It provides a distributed framework so that it is can be implemented for us on robots that operate with multiple processors. Because of the modular nature of this middleware, it also distributes its commands to effectively "anyone who listens." This behavior allows for simulators to run using the same code that could be run on a real robot, perhaps with a few tuning variable adjusted.

Several simulators are available for use alongside ROS such as Gazebo, Stage, and Simple Two-Dimensional Robot Simulator, or STDR. STDR Simulator is designed to be an easy-to-use simulator that one can quickly develop software for. Because of a lack of documentation; it was initially a difficult task to figure out how to operate the program and its associated features. Several features of the simulator currently include multiple robots being controlled at once in simulation, the adding of sound and radiation sources that can be perceived by the robot using specialized virtual sensors, and the addition of LIDAR and sonar sensors to the robot. The radiation and sound sources could prove very useful in the future for the implementation of planning algorithms.



STDR Simulator GUI

The implementation of a Kalman filter in the STDR environment was a lengthy process, mainly due to the lack of concrete documentation and critical features such as noise in sensor measurements. Nonetheless the overall implementation was relatively simple despite the huge amount of overhead that came with the implementation.

A mathematical summary of the Kalman filter is as follows:

$$
\begin{aligned}
\mathrm{x}_p &= Ax_{n-1} + Bu_n \\
\mathrm{P}_p &= AP_{n-1}A^T + Q \\
\widetilde{y} &= z_n - Hx_p \\
\mathrm{S} &= \mathrm{H}\mathrm{P}_p H^T + R \\
\mathrm{K} &= \mathrm{P}_p H^T S^{-1} \\
\mathrm{x}_n &= x_p + K\widetilde{y} \\
\mathrm{P}_n &= (I - KH)P_p
\end{aligned}
$$

The filter first takes in a predicted state using the previous, or initial state along with the input to the system. These variables are then multiplied by matrices to tune them to be compatible with the vector

that defines the state. Next, the predicted covariance is obtained by multiplying the previous covariance matrix with the state transition matrix and its transpose and adding additional process error.

The next step is the most important in the process; where the innovation is found. The predicted state is multiplied against an observation matrix and then subtracted from the measurement vector. This step shows the difference of the system's 'real' measured state versus the estimated state. The associated innovation covariance is then calculated by multiplying the predicted covariance by the observation matrix and its transpose and summed with additional measurement error.

The Kalman gain is the calculated by multiplying the predicted covariance with the transpose of the observation matrix and then the inverse of the innovation covariance. Finally, the state update vector and its covariance matrix are calculated – the state by summing the predicted state with the innovation multiplied by the Kalman gain, and the covariance by multiplying the predicted covariance with the difference of the identity matrix and the state transition matrix multiplied by the Kalman gain.

II. Implementation and Results

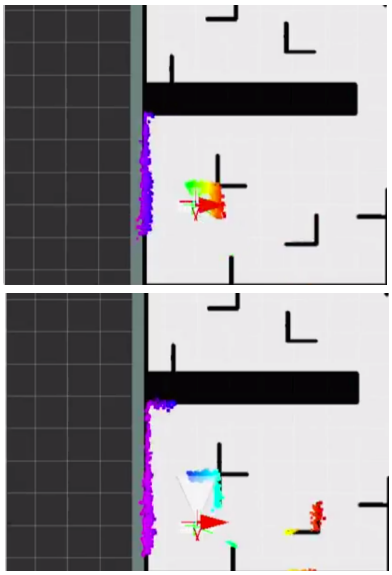
The lack of the simulator's ability to add noise into the environment was one of its major drawbacks. To combat this, a ROS node was created to subscribe to a topic of type sensor_msgs/LaserScan, extract the range vector, add gaussian noise to it, and republish the scan message for use by the sensor model. The node that does this is called "noisey" and could prove to be a valuable asset to other individuals developing applications for use in this simulator as well as others, therefore, plans on converting the utility into a standardized ROS library are underway.

Other noise present in the system is required for the full potential of the Kalman filter to be realized. The error that was present in the velocity model was propagated via the floating point imperfection present when performing operations with those kinds of variables. The imperfection present in these operations was assumed to be gaussian and offset from the true value by a small factor. The error in these calculated values proved to work well for the purposes of implementing the filter.

The Kalman filter requires two motion models; one from the predicted position using velocity commands, and one measurement model using gathered information from the sensors. The sensor model was able to accurately determine the position of the robot with some error by matching the current laser scan with a list of previously defined scans and associated poses. This scan matching technique was constructed by creating a large text file with distance information describing eight angles at each of the corresponding positions. The text file was generated programmatically and then batch edited for greater usability.

A node had the task of parsing through the file and creating an associated data structure for each of the range values. Once this data structure was created, it was sent to a separate node whose job it was to create the actual sensor model estimation of where the robot was in the environment. This node read the current noisey scan data from the robot and tolerance matched it using an average difference estimation between corresponding values in each of the range sets. An accurate pose was successfully extracted using this technique.

The velocity model operated on linear state space equations and the constraint that there would



KF pose (red arrow) overlaid
on the coordinate frames of the robot

always be a constant velocity command executed for a single second, as well as the ability to only control either the translational and rotational velocity independent of each other. This constraint made the

velocity model more accurate and satisfied the Kalman filter's linearity assumption.

$$\begin{aligned} \mathrm{x}_p &= x_{p-1} + \cos\omega * \Delta t * v_x \\ \mathrm{y}_p &= y_{p-1} + \sin\omega * \Delta t * v_y \\ \theta &= \theta_{p-1} + \Delta t * \omega \end{aligned}$$

The Kalman filter node subscribed to the two Pose2DWithCovariance topics that were published by the velocity model and the sensor model and then published another Pose2DWithCovariance message to reflect the predicted position of the robot. A separate PoseStamped message was generated so a marker could be placed in RViz to display where the filter predicted the measurement to be.

III. Conclusions

Given a linear control input and a good sensor model, a Kalman filter is a great way to localize a mobile robot in a 2D environment. However, if the linearity is broken in any way the Kalman filter will not produce accurate results. For this, the implementer may wish to use a more robust version of the filter such as the Extended Kalman filter, Unscented Kalman filter, or changing routes altogether and implementing a Particle filter.

STDR Simulator provides a relatively good infrastructure for the relatively quick implementation of simple robots. Unfortunately, the lack of documentation and example code made more overhead required to begin concrete work on the implementation of the models and filter. Hopefully, future work by myself and others will be able to use code written for this project as a basis for implementing other more complex simulations.