A Generalized Extended Kalman Filter Implementation for the Robot Operating System

Thomas Moore
Sensor Processing and Networking Division
Charles River Analytics, Inc.
Cambridge, Massachusetts, USA
tmoore@cra.com

Daniel Stouch
Sensor Processing and Networking Division
Charles River Analytics, Inc.
Cambridge, Massachusetts, USA
dstouch@cra.com

***Abstract*—Accurate state estimation for a mobile robot often requires the fusion of data from multiple sensors. Software that performs sensor fusion should therefore support the inclusion of a wide array of heterogeneous sensors. This paper presents a software package, *robot_localization*, for the Robot Operating System (ROS). The package currently contains an implementation of an extended Kalman filter (EKF). It can support an unlimited number of inputs from multiple sensor types, and allows users to customize which sensor data fields are fused with the current state estimate. In this work, we motivate our design decisions, discuss implementation details, and provide results from real-world tests.**

Keywords—sensor fusion; extended Kalman filter; localization; Robot Operating System

I. Introduction

A critical challenge for all mobile robots is the ability to answer the question, "Where am I?" [1] The answer to this question can be obtained through the robot's sensors, which can provide both proprioceptive and exteroceptive data. However, sensors are imperfect, and their measurements are prone to errors. By fusing the data from multiple sensors, we can obtain an overall position estimate whose error is less than would be possible by using a single sensor in isolation. It is often the case that a greater amount of sensor input data will produce more accurate position estimates. It is therefore critical that any software that performs sensor fusion on a mobile robot platform is able to take in any and all available data on the platform. Additionally, the software should be easy to use and highly customizable, thereby providing users with greater of flexibility while allowing them to focus on higher-level behaviors.

In this paper, we introduce our software package, *robot_localization*, for the Robot Operating System (ROS) [2]. ROS is an open-source robotic framework that has been widely adopted across academia, industry, and militaries around the world. Our software addresses the sensor fusion needs of a broad range of mobile robots and allows for rapid integration with those platforms.

In Section II, we detail our motivation for the creation of the *robot_localization* package. In Section III, we describe our extended Kalman filter (EKF) [3] ROS node, *ekf_localization_node*. Section IV details experiments that we performed on a Pioneer 3 mobile robot. Section V provides a summary discussion of the experiments, and Section VI concludes with applications to other platforms and details plans for extending both *ekf_localization_node* and the *robot_localization* package.

II. Motivation

The ROS community has developed and contributed a wealth of software to facilitate robotic development and practice, with over 2,000 packages available to date. While other packages exist that perform state estimation, they are often difficult to apply to new problems for a variety of reasons:

- **Limited sensor inputs**. Robots are being equipped with an increasing number of sensors, and existing ROS packages require a lot of user effort to successfully integrate the data from all of them.
- **Limited to 2D estimation**. For some unmanned ground vehicles (UGVs) operating in planar indoor environments, ROS packages that estimate the vehicle's state in 2D are sufficient for the intended application. However, these packages are insufficient for estimating the state of platforms that operate in 3D, such as unmanned aerial vehicles (UAVs), unmanned underwater vehicles (UUVs), and UGVs operating outdoors.
- **Limited ROS message support**. Sensor data in ROS often originates from hardware driver packages over which the user has no control. If a state estimation node does not support a given message type, the user must either modify the driver's source or create an intermediary node to copy the message data into a supported message type.
- **Limited control over sensor data**. Accurate state estimates often require only a subset of the available sensor messages because of faulty sensors or sensor drivers that fail to properly fill out covariance values. This requires users to modify the data messages, e.g., by artificially inflating covariances.

We developed our *robot_localization* package from the ground up to overcome these limitations and be as general-

purpose as possible. It performs state estimation in 3D space, allows for an unlimited number of sensors, supports multiple standard ROS message types, and allows per-sensor control of which message fields are fused with the state estimate.

III. Extended Kalman Filter Node

We developed *ekf_localization_node*, an EKF implementation, as the first component of *robot_localization*. The *robot_localization* package will eventually contain multiple executables (in ROS nomenclature, *nodes*) to perform state estimation. These nodes will share the desirable properties described in Section II, but will differ in their mathematical approaches to state estimation. In this section, we describe the implementation details for *ekf_localization_node*.

A. Extended Kalman Filter Algorithm

The EKF formulation and algorithm are well-known [3, 4, 5]. We detail them here to convey important implementation details. Our goal is to estimate the full 3D (6DOF) pose and velocity of a mobile robot over time. The process can be described as a nonlinear dynamic system, with

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}) + \boldsymbol{w}_{k-1}\,, \tag{1}$$

where $\boldsymbol{x}_k$ is the robot's system state (i.e., 3D pose) at time k, f is a nonlinear state transition function, and $\boldsymbol{w}_{k-1}$ is the process noise, which is assumed to be normally distributed. Our 12-dimensional state vector, $\boldsymbol{x}$, comprises the vehicle's 3D pose, 3D orientation, and their respective velocities. Rotational values are expressed as Euler angles. Additionally, we receive measurements of the form

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k\,, \tag{2}$$

where $\boldsymbol{z}_k$ is the measurement at time k, h is a nonlinear sensor model that maps the state into measurement space, and $\boldsymbol{v}_k$ is the normally distributed measurement noise.

The first stage in the algorithm, shown as equations (3) and (4), is to carry out a prediction step that projects the current state estimate and error covariance forward in time:

$$\hat{\boldsymbol{x}}_k = f(\boldsymbol{x}_{k-1}). \tag{3}$$

$$\hat{\boldsymbol{P}}_{\boldsymbol{k}} = \boldsymbol{F}\boldsymbol{P}_{k-1}\boldsymbol{F}^T + \boldsymbol{Q}. \tag{4}$$

For our application, f is a standard 3D kinematic model derived from Newtonian mechanics. The estimate error covariance, $\boldsymbol{P}$, is projected via $\boldsymbol{F}$, the Jacobian of f, and then perturbed by $\boldsymbol{Q}$, the process noise covariance.

We then carry out a correction step in equations (5) through (7):

$$\boldsymbol{K} = \hat{\boldsymbol{P}}_k\boldsymbol{H}^T\left(\boldsymbol{H}\hat{\boldsymbol{P}}_k\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1}. \tag{5}$$

$$\boldsymbol{x}_{\boldsymbol{k}} = \hat{\boldsymbol{x}}_k + \boldsymbol{K}(\boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}_{\boldsymbol{k}}). \tag{6}$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\hat{\boldsymbol{P}}_k(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T. \tag{7}$$

We calculate the Kalman gain using our observation matrix, $\boldsymbol{H}$, our measurement covariance, $\boldsymbol{R}$, and $\hat{\boldsymbol{P}}_k$. We use the gain to update the state vector and covariance matrix. We employ the Joseph form covariance update equation [6] to promote filter stability by ensuring that $\boldsymbol{P}_k$ remains positive semi-definite.

The standard EKF formulation specifies that $\boldsymbol{H}$ should be a Jacobian matrix of the observation model function h. To support a broad array of sensors, we make the assumption that each sensor produces measurements of the state variables we are estimating. As such, $\boldsymbol{H}$ is simply the identity matrix. A core feature of *ekf_localization_node* is that it allows for partial updates of the state vector, which is also a requirement of any future state estimation nodes that are added to *robot_localization*. This is critical for taking in sensor data that does not measure every variable in the state vector, which is nearly always the case. In practice, this can be accomplished through $\boldsymbol{H}$. Specifically, when measuring only m variables, $\boldsymbol{H}$ becomes an m by 12 matrix of rank m, with its only nonzero values (in this case, ones) existing in the columns of the measured variables.

Because the process noise covariance, $\boldsymbol{Q}$, can be difficult to tune for a given application [7], *ekf_localization_node* exposes this matrix as a parameter to users, allowing for an additional level of customization.

IV. Experiments

We designed and executed two experiments to evaluate the performance of *ekf_localization_node*. Our test platform is a MobileRobots Pioneer 3 (Fig. 1). The robot is equipped with wheel encoders that provide raw odometry estimation. Additionally, we have mounted a sensor suite on the platform with two Microstrain 3DM-GX2 IMUs and two Garmin GPS 18x units. The sensors are mounted on a custom rig that aids in magnetic interference reduction for the IMUs and increases signal quality for the GPS units. We have configured *ekf_localization_node* to take in roll, pitch, yaw, and their respective velocities from each of the IMUs, and x and yaw velocity from the wheel encoders. For each GPS, we define a transform that converts the robot's world frame coordinates (i.e., the frame with its origin at the robot's start position) to the GPS's UTM coordinates, as

$$\boldsymbol{T} = \begin{bmatrix} c\theta c\psi & c\psi s\Phi s\theta - c\Phi s\psi & c\Phi c\psi s\theta + s\theta s\psi & x_{\mathrm{UTM}_0} \\ c\theta s\psi & c\Phi c\psi + s\Phi s\theta s\psi & -c\psi s\Phi + c\Phi s\theta s\psi & y_{\mathrm{UTM}_0} \\ -s\theta & c\theta s\Phi & c\Phi s\theta & z_{\mathrm{UTM}_0} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{8}$$

where Φ, θ, and ψ are the vehicle's initial UTM-frame roll, pitch, and yaw, respectively. c and s designate the cosine and sine functions, respectively, and x_{UTM_0}, y_{UTM_0}, and z_{UTM_0} are the UTM coordinates of the first reported GPS position. At any subsequent time t, we transform the GPS measurement into the robot's world coordinate frame, *odom*, by

$$\begin{bmatrix} x_{\text{odom}} \\ y_{\text{odom}} \\ z_{\text{odom}} \\ 1 \end{bmatrix} = \boldsymbol{T}^{-1} \begin{bmatrix} x_{\text{UTM}_t} \\ y_{\text{UTM}_t} \\ z_{\text{UTM}_t} \\ 1 \end{bmatrix}. \quad (9)$$

We then configure *ekf_localization_node* to fuse the transformed position with the state estimate. This process is carried out for each GPS independently.

We collected raw sensor data from the platform in the parking lot of the authors' building. Its path is depicted in Fig. 2. The experiment environment measures approximately 110 meters from the robot's origin to the most distant point traveled. The robot was joystick controlled and driven so that its final position was exactly where it started. The collection lasted approximately 777 seconds. The data was then played back through ROS's *rosbag* utility, allowing us to run multiple experiments on the same collected dataset[1]. While the state was estimated in 3D space (i.e., taking into account roll and pitch and incorporating the GPS altitude measurements), we report our results in 2D, as the nearly planar environment made the reporting of 3D information superfluous (Section VI provides an example of *ekf_localization_node* applied to a 3D state estimation problem for a UAV).

A. Loop Closure Accuracy

For our first experiment, we are interested in the distance between the robot's start and end positions, as reported by *ekf_localization_node*. Ideally, we would like the end position (*x*, *y*) values to be as close to the origin at (0, 0) as possible. We repeat the experiment in multiple *ekf_localization_node* sensor configurations: (1) dead reckoning via the platform's odometry, (2) fused odometry with a single IMU, (3) fused odometry with two IMUs; (4) fused odometry with two IMUs and a single GPS, and (5) fused odometry with two IMUs and two GPS units.

As previously mentioned, *ekf_localization_node* affords the ability to configure which variables from each of its sensors are actually fused in the final state estimate. We list the sensor configurations in Table I. Note that for these experiments, we hold each configuration constant. In reality, however, some configurations would likely change depending on the sensor package. See Section V for further discussion.

The results are listed in Table II. For each sensor configuration, the loop closure error is reported for *x* and *y* position. Our experiment design is such that this error is simply *ekf_localization_node*'s last reported state estimate before the bag file playback stops. We also report the filter's last estimated standard deviation values for *x* and *y*. This gives an indication of how closely the EKF's error matches reality. Statistics are reported in the robot's world coordinate frame (in keeping with ROS standards, we denote this frame *odom*). The robot's starting orientation and origin are depicted in Fig. 2.

We also present graphical depictions of the results in Fig. 2 through Fig 7. In Fig. 2, we give the robot's path as an average of the GPS tracks. This serves only as a visual guide and is not considered to be ground truth. In Fig. 3 through Fig. 7, we overlay the estimated paths on top of this visualization for reference purposes. It also showcases how the configurations in rows four and five of Table II (corresponding to Fig. 6 and Fig. 7) improve upon the average GPS track.

The results of the experiment largely follow intuition. Dead reckoning yields the worst performance, with the robot's final reported position being more than 174 meters from the origin (Fig. 3). Our Pioneer's wheel encoders are biased in such a way that straight lines get reported as mild right turns, leading to a highly inaccurate position estimate. Including one IMU

[1] The bag file generated from this experiment is available at http://www.cra.com/robot_localization_ias13.zip

Fig. 1: Our test platform is a Pioneer 3 with a custom sensor mounting rack. It has two IMUs and two GPS units.

TABLE I. SENSOR CONFIGURATIONS

Sensor	*Configuration* Vector *0 = false, 1 = true*											
	x	***y***	***z***	**Φ**	**θ**	**ψ**	***x′***	***y′***	***z′***	**Φ′**	**θ′**	**ψ′**
Odometry	0	0	0	0	0	0	1	1	1	0	0	1
IMU 1	0	0	0	1	1	1	0	0	0	1	1	1
IMU 2	0	0	0	1	1	1	0	0	0	1	1	1
GPS 1	1	1	1	0	0	0	0	0	0	0	0	0
GPS 2	1	1	1	0	0	0	0	0	0	0	0	0

TABLE II. ERRORS FOR FIVE DIFFERENT SENSOR CONFIGURATIONS

Sensor Set	*Loop Closure Error x, y (m)*	*Estimate Std. Dev. x, y (m)*
Odometry (dead reckoning)	69.65, 160.33	593.09, 359.08
Odometry + one IMU	10.23, 47.09	5.25, 5.25
Odometry + two IMUs*	12.90, 40.72	5.23, 5.24
Odometry + two IMUs* + one GPS	1.21, 0.26	0.64, 0.40
Odometry + two IMUs* + two GPSs	0.79, 0.58	0.54, 0.34

* IMU 2 failed after approximately 45% of the collection

Fig. 2: The robot's path as a mean of the two raw GPS paths is shown in red. Its world coordinate frame is shown in green.

Fig. 3: Output of *ekf_localization_node* (yellow) when fusing only raw odometry data.

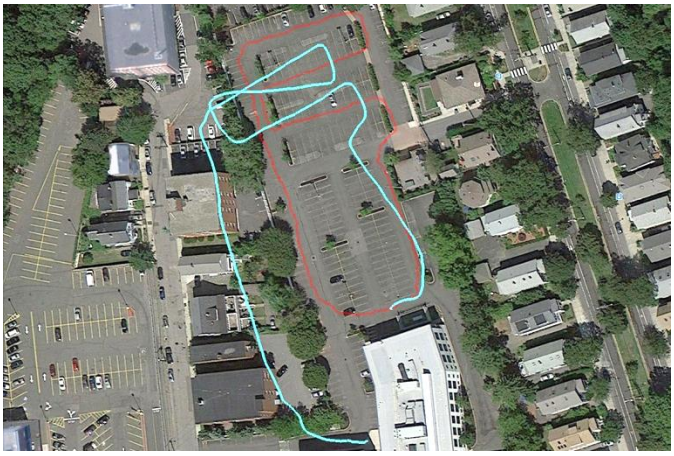


Fig. 4: Output of *ekf_localization_node* (cyan) when fusing data from odometry and a single IMU.

Fig. 5: Output of *ekf_localization_node* (orange) when fusing data from odometry and two IMUs. Note that the second IMU stopped reporting data midway through the run.

Fig. 6: Output of *ekf_localization_node* (blue) when fusing data from odometry, two IMUs, and one GPS.

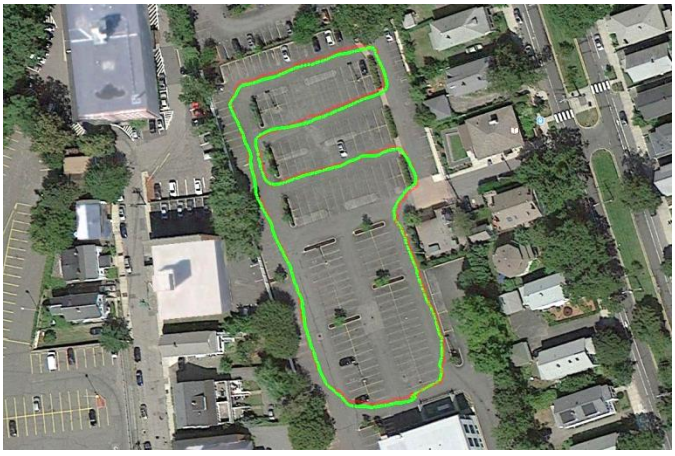


Fig. 7: Output of *ekf_localization_node* (green) when fusing data from odometry, two IMUs, and two GPS units.

aids in correcting this problem, owing to the order-of-magnitude improvement in the IMU's yaw velocity error over that of the Pioneer's odometry, as well as the fusion of absolute orientation (Fig. 4). However, the collection area contains many areas with strong electromagnetic interference, resulting in inaccurate headings being reported by the magnetometer. The addition of a second IMU improves the final position error only slightly, because it is subject to the same interference, and because it actually stopped reporting data halfway through the collection (Fig. 5). While normally a cause to repeat the experiment, this sensor failure serves as an example of why the fusion of multiple sensors is so powerful, as the system can more gracefully cope with faulty or infrequent sensor data.

We can further refine our estimate through the inclusion of a single GPS (Fig. 6). This aids in constraining the effects of both the odometry's inaccurate linear velocity estimates and eliminates the effect of the poor heading estimate resulting from IMU interference. Adding a second GPS provides a less drastic improvement in the final position error, but showcases the ability of *ekf_localization_node* to successfully fuse data from a large number of sensor inputs (Fig. 7).

B. *Infrequent GPS*

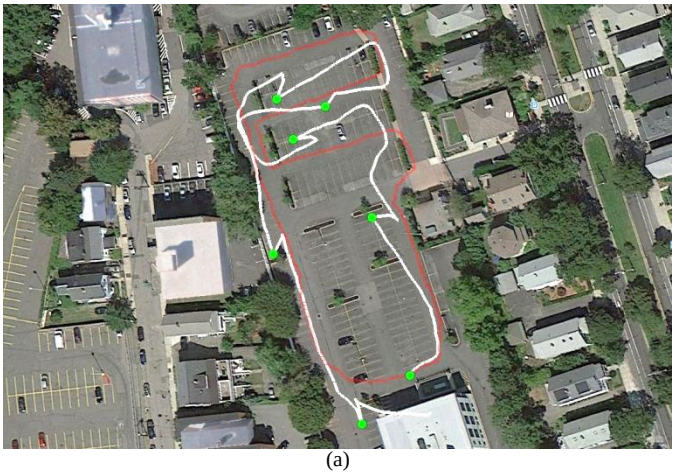
Many robots receive infrequent absolute position measurements and must maintain a state estimate when these signals are absent. For our second experiment, we want to evaluate the performance of the filter when GPS signals arrive infrequently. We run the experiment with the same sensor configuration as in row four of Table II, i.e., with odometry, both IMUs, and one GPS. However, we filter the collection log file such that GPS data is only available once every 120 seconds. Our aim is to determine how gracefully the filter handles the fusion of sensor data that varies greatly from its current state estimate.

The results are shown in Fig. 8. The locations at which GPS fixes occur are displayed on the map, and result in noticeable instantaneous position changes. These jumps clearly pull the state estimate towards the GPS track, but the Kalman gain gives some weight to the current state estimate, resulting in the new position being in between the current state and measurement. Despite the large difference between state estimate and measurement, the filter's covariance matrix retains its stability, and the x and y variance values decrease considerably. At the end of the run, the vehicle's loop closure (x, y) absolute error is (12.06, 0.52) meters.

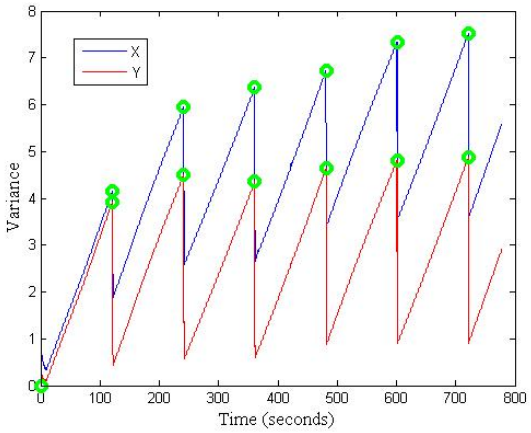
V. Discussion

Referring again to the first row of Table II (dead reckoning), it is clear that the estimate variance for x and y was very large. For this particular test, the condition number of the covariance matrix grew rapidly, indicating filter instability. This is due in part to the sensor's configuration. The strong correlation between yaw and x and y position means that, without an absolute measurement of yaw or (x, y), the errors on these values will grow rapidly. Clearly, this problem is solved by the inclusion of IMUs, which provide absolute yaw measurements.

The estimated standard deviations of the (x, y) positions for rows two and three in Table II are much smaller than the true



(a)



(b)

Fig. 8: (a) Output of *ekf_localization_node* (white) when fusing odometry, IMU, and infrequently reported GPS data. GPS fixes occur at the green circles. (b) x and y position variances for the same run. Green circles denote the reception of GPS data.

position estimation errors. This is partially due to both the Pioneer odometry and IMU data being noisier than its covariance values reported. We also did not tune the process noise covariance matrix, $\boldsymbol{Q}$ [7].

It is worth noting that despite the fact that the odometry only truly measures x and yaw velocity, we can infer more information. The platform is not going to obtain any instantaneous z or y velocity due to platform constraints, i.e., it cannot fly and is nonholonomic. We can therefore fuse the zero values in those data fields with our estimate, providing that the measurement's covariances are set appropriately (Table I). In general, if the measurement of a quantity is implied through kinematic constraints, it is best to treat that quantity as a measured value.

Although our experiments utilized only proprioceptive sensors, the design of our software is such that inputs from exteroceptive sensors such as laser scanners or cameras could be used as well, provided that they produce supported ROS message types. For example, the iterative closest point (ICP) algorithm [8] could be used with data from an RGBD sensor

such as the Microsoft Kinect [9] to generate an additional source of odometry [10].

While the number of sensor permutations possible for this experiment was not large, the sensor data customization parameters for *ekf_localization_node* (and future nodes in *robot_localization*) can yield a much larger set of possible configurations. For example, one of our IMUs is known to have a faulty gyroscope. In that case, we can use that IMU to report only orientation and use the second IMU to give us both orientation and orientation velocity. This kind of fine-grained control is useful for dealing with known faulty sensors and for troubleshooting.

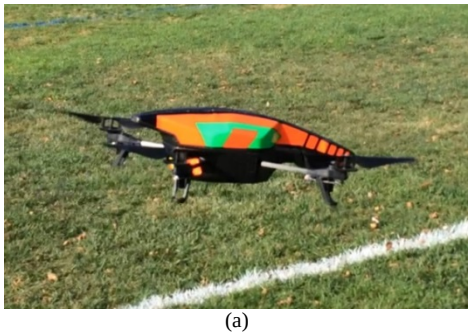
VI. Conclusion and Future Work

While this work focused on a robot operating in a near-planar environment, we have also successfully applied *ekf_localization_node* across multiple projects involving both ground and aerial robots [11]. In particular, we have integrated the software with a Parrot AR.Drone 2.0 quadcopter via the *ardrone_autonomy* ROS package [12] (Fig. 10). The drone has camera-based velocity sensing, camera- and barometry-based altitude sensing, an IMU, and GPS.

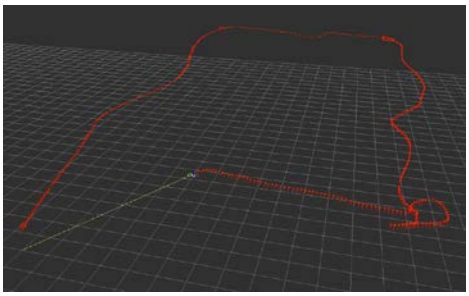
We plan to improve both *ekf_localization_node* and *robot_localization* in a number of ways:

- **Covariance override**. Some ROS nodes for specific robot platforms or sensors assign arbitrary values for certain covariance matrix entries so as to signify that the quantity in question is not measured or is not trustworthy. However, as discussed in Section V, users may wish to incorporate sensor data for values that aren't actually measured by the sensor. Currently, the sensor data preprocessing logic, common to all nodes in *robot_localization*, assigns a small variance value to any sensor that is fused with a variance of zero. While this allows the measurement to be fused without breaking the filter, the values should be parameterizable.
- **Support for linear acceleration**. We do not currently fuse linear acceleration in our state estimate or account for it in our kinematic model. Doing so will further increase filter accuracy.
- **Additional state estimation nodes**. The *robot_localization* package is meant to contain multiple nodes for carrying out state estimation. We plan to add new nodes in the future, such as an unscented Kalman filter [13] node and a particle filter [14] node.

In this paper, we introduced a generalized extended Kalman filter node, *ekf_localization_node*, for our *robot_localization* ROS package. Its support for multiple sensors and high level of customizability make it well suited to the problem of state estimation for a variety of robot platforms. It is the authors' hope that *robot_localization* will benefit from the feedback and contributions of the ROS community.



(a)



(b)

Fig. 9: (a) Parrot AR.Drone 2.0 quadcopter, (b) 3D path flown by the AR.Drone and estsimated by *ekf_localization_node*.

References

[1] J.J. Leonard and H.F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *Robotics and Automation, IEEE Transactions on* vol. 7, no. 3, pp. 376-382, 1991.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A.Y. Ng, "ROS: an open-source robot operating system," *ICRA workshop on open source software* vol. 3, no. 3.2, 2009.

[3] G.L. Smith, S.F. Schmidt and L.A. McGee, "Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle," 1962.

[4] R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME* pp. 35-45, 1960.

[5] G. Welch and G. Bishop, "An introduction to the Kalman filter," 1995.

[6] G.J. Bierman and C.L. Thornton, "Numerical comparison of Kalman filter algorithms: Orbit determination case study," *Automatica* vol. 13, no. 1, pp. 23-35, 1977.

[7] B.J. Odelson, M.R. Rajamani and J.B. Rawlings, "A new autocovariance least-squares method for estimating noise covariances," *Automatica* vol. 42, no. 2, pp. 303-308, 2006.

[8] P. Besl and N. McKay, "Method for registration of 3-D shapes," *IEEE PAMI*, 14, pp. 239-256, 1992.

[9] Microsoft Corporation, "Kinect for Windows,". http://www.microsoft.com/en-us/kinectforwindows/, 2014

[10] A. Milella, and R. Siegwart, "Stereo-based ego-motion estimation using pixel tracking and iterative closest point," *Computer Vision Systems, 2006 ICVS'06. IEEE International Conference on*., 2006.

[11] D. Stouch, A. Ost, T. Moore and C. Monnier, "Robust Tactical Communications Relay using Visual Object Detection on an Autonomous Mobile Robot," *International Advanced Robotics Programme's 7th International Workshop on Robotics for Risky Environments - Extreme Robotics (IARP RISE-ER 2013)* 2013.

[12] M. Monajjemi, "Autonomylab/ardrone autonomy. github. com," *AutonomyLab/ardrone autonomy* 2013.

[13] S.J. Julier and J. Uhlmann, "A New Extension of the Kalman Filter to Nonlinear Systems," *AeroSense, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management II* 1997.

[14] S. Thrun, D. Fox, W. Burgard and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots," *Artificial Intelligence* vol. 128, no. 1-2, pp. 99-141, 2000.