Fault detection for mobile robots using redundant positioning systems

Paul Sundvall and Patric Jensfelt
Centre for Autonomous Systems, KTH
SE-100 44 Stockholm, Sweden

***Abstract*— Reliable navigation is a very important part of an autonomous mobile robot system. This means for instance that the robot should not lose track of its position, even if unexpected events like wheel slip and collisions occur. The standard approach to this problem is to construct a navigation system that is robust in itself. This paper proposes that detecting faults can also be made outside the normal navigation system, as an additional fault detector. Besides increasing the robustness, a means for detecting deviations is obtained, which can be important for the rest of the robot system, for instance the top level planner. The method uses two or more sources of robot position estimates, and compares them to detect unexpected deviation without getting deceived by drift or different characteristics in the position systems it gets information from. Both relative and absolute position sources can be used, meaning that existing positioning systems already implemented can be used in the detector. For detection purposes, an extended Kalman filter is used in conjunction with a CUSUM test. The detector is able to not only detect faults, but also give an estimate of when the fault occurred, which is useful for doing fault recovery. The detector is easy to implement, as it requires no modification of existing systems. Also the computational demands are very low. The approach is implemented and demonstrated on a mobile robot, using odometry and a scan matcher as sources of position information. It is shown that the system is able to detect wheel slip in real-time.**

I. Introduction

For mobile service robots, a crucial part of the system is to navigate reliably, meaning to move the robot between places while avoiding losing track of its position. Such functionality is needed in e.g. fetch-and-carry tasks and supervising tasks. For such tasks, it is necessary for the robot not to lose track of its position in order to be successful. Losing position means that the risk of collision is increased, as there might be obstacles marked on the robot's map that are not possible to detect with normal sensors and thus not captured by obstacle avoidance mechanisms. An example of such an obstacle is descending stairs. Clearly, the success of the robot is highly dependent on the integrity of the navigation system. The importance of the navigation system for a mobile robot can be compared to how much an industrial robot is depending on its joint encoders. If a collision occurs, the robot should minimize the damage by either doing an emergency stop or take recovery actions. Such collisions might hurt people or damage the robot, which should be avoided to the greatest possible extent.

Faults typically have a low probability of occurring, but the costs associated with them are high. Most systems work well under normal operating conditions, but their performance degrade significantly upon unexpected events, such as failing sensors or undetected obstacles. For applications in a domestic setting, the environment is changing and there are people moving in the close proximity of the robot. For such applications, users may not be supervising or may not even be able to help the robot if an unwanted situation occurs.

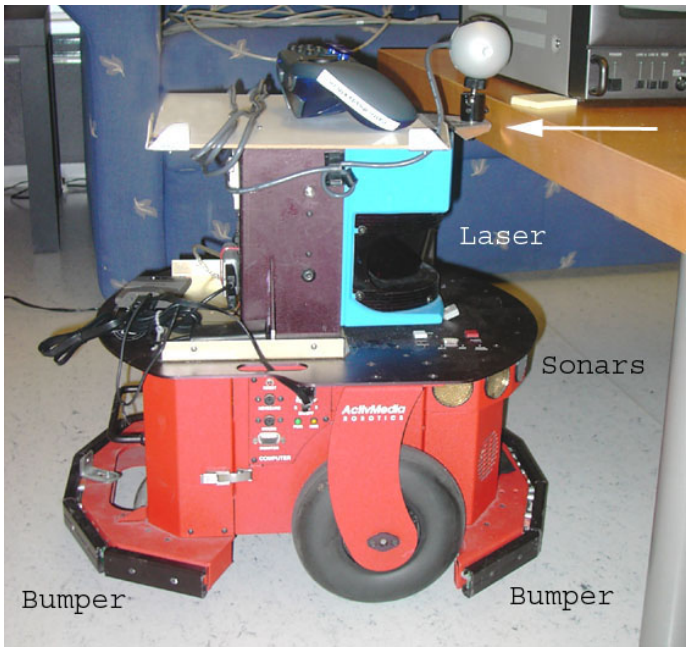

Fig. 1. A Pioneer robot in collision (see arrow in top right corner) with a table. This is not observable by the laser scanner (blue), sonars or the bumper switches since they are vertically displaced compared to the table. At the back of the robot (left in the picture), there is a caster wheel. As the robot drives forward against the table, the weight transfers to the caster wheel and the wheels slip against the floor.

There are many things that can make the navigation system fail, e.g. collisions or sliding against something which makes the robot rotate, slippage when running over a cable or a threshold, or because users push the robot. Earlier studies have shown that users are prone to test the limits of the robot capabilities, trying to destroy it or harassing it on purpose [1]. Detecting faults in the navigation system improves the performance in such situations. An example of a situation where the navigation will fail is shown in Fig. 1. This situation could for instance occur if the robot is exploring the world to build a map of the environment, or if the obstacle is not

included in the existing map. These two examples are valid for what situations a service robot can encounter. In the case shown in the figure, there are very few (if any) systems that can handle the situation, and the robot will almost certainly lose track of its position. Regardless of how the robot got into the situation, there is in this case no sensor on the robot that can detect that the top of the robot is in contact with the table. As the robot strives forward, the wheels will start spinning on the floor because of the contact with the table. This situation could be mitigated by adding more sensors, until the point that there would be bumpers all over the robot. Adding sensors increases complexity, space and cost, and does not seem to be a promising solution.

For the operation of the robot, it might be of equally high importance to know that there has been a collision as it is to maintain navigation performance. This implies that robust navigation is not enough for successful operation, but should be accompanied with a fault detection system.

II. Approach and outline

In the present paper, the approach is to use any two or more (redundant) localization methods, and compare their outputs to detect unexpected large deviation. Small deviations between localization methods are normal due to sensor noise and algorithm imperfections, while large deviations indicate that a fault has occurred.

An extended Kalman filter is used to track the outputs from the localization methods, and the residual of the filter is used to determine if an unexpectedly large deviation has occurred. To reduce the risk of false alarms, a CUSUM test is used to monitor the residual of the Kalman filter.

In the next section, other approaches to the problem are discussed. The details of the current approach presented in this paper are provided in Section IV. Experimental results from an implementation of the approach are shown in Sections V and VI.

III. Related work

Increasing robustness has been a driving force for developing navigation methods. In [2], movable doors are detected and removed from the map in order not to entice the navigation system when doors are opening or closing. Having people standing around the robot while building a map of the environment can confuse the robot. In [3], laser echoes from people are detected and removed prior to feeding the laser data into the map building system.

A common approach for navigation is to use a Kalman filter. As the Kalman filter only can approximate a unimodal probability distribution, methods to handle ambiguous situations have been developed. One example is [4], where multiple hypothesis are used to represent the uncertainty of the robot position. Another example is to use particle filters to represent the uncertainty [5].

Common to the approaches mentioned above is that the robustness is built into the navigation system, meaning that changes and tuning needs to be done within those systems. The present approach is different, as the detection of anomalies is performed *outside* the navigation system modules. Implementing detection of faults using an external fault detector might be easier than changing existing localization modules, unless you are a domain expert.

The idea of using Kalman filtering for fault detection is not new. For robot localization, the Kalman filter is very often used to fuse different sensors. Several contributions for detecting faulty sensors exist. In [6], a bank of Kalman filters, each one tuned to a specific fault, is used to do detection. Fault isolation is obtained by studying which of the residuals are large. The approach is demonstrated on a robot equipped with wheel encoders (odometry) and a rate gyro. In this approach, the detection is implemented in close conjunction with the sensors. Another approach is presented in [7]. Multiple sensors are combined to obtain several estimates of robot position, orientation and speed based on different sensors for each estimate. The pairwise differences between the estimates are then used as residuals. Based on what residuals are small and large, a table mapping high residuals to specific faults is used to isolate faults. Navigation is then performed with the subset of sensors that are considered functioning. It is not clear how the algorithm handles accelerometer and gyroscope drift, which would cause the position estimates to drift away compared to the odometry based estimates. An important difference between the proposed method and the methods in [6] and [7] is that those methods operate on a sensor level, while the proposed method operates on a higher level.

A way to increase the possibility to detect and accommodate faults is to add more sensors. Hardware may be doubled or more (*Hardware redundancy*), or redundancy can be added by observing the same thing with different sensors. One approach based on the latter is "gyrodometry", presented in [8]. In this case, the difference of yaw rate reported by a gyro and odometry is thresholded to decide which source of rotational speed should be used for positioning.

In [9], a particle filter is used for fault detection and isolation. Several discrete states are used, one for each fault mode, and an associated model for the continuous states is used for every mode. Particle filters are very powerful for tracking systems, and output a probability distribution over the states, given as samples (particles). It can be used for nonlinear, non-Gaussian and multi modal distributions and outperforms the Kalman filter in such cases. The drawback is that the computational demands increase fast with the state dimension, even if there are means for increased efficiency [9]. Another drawback is that a fault model is needed to track the probability of being in a fault state. With a Kalman filter tracking the normal state, it is possible to test for deviation from the normal (fault free) model, thus requiring a model only for normal behaviour.

IV. Modelling

As previously mentioned, the approach is to use existing systems or sensors ("pose providers") for obtaining updates on robot position or movement. All these systems have their own

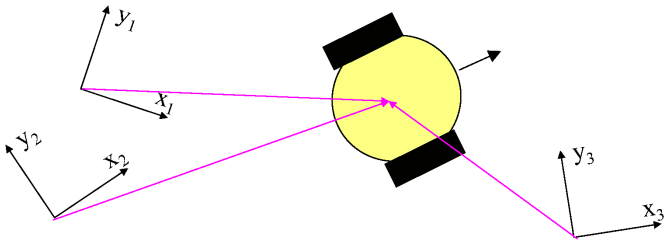

Fig. 2. The pose providers might use different coordinate systems, as indicated by the figure. Drift in the pose providers will make the coordinate systems move around with time.

characteristics of drift and noise. In the following sections, a model is presented that makes it possible to decide what is a normal deviation and what is not.

A. Pose providers

The term *pose provider* is used for sensors or systems that deliver estimates of the robot's position. The estimates do not need to be in the same coordinate system or have the same update rate. See Fig. 2 for a visualization of different pose providers having different meanings of where the robot is. Examples of pose providers are odometry, which can be regarded as a sensor, and SLAM which is an algorithm. Other examples are scan matching based odometry [10], visual odometry [11] and wireless localization [12]. It is important that the pose providers provide redundancy. Many navigation modules assume that the odometry is quite reliable and will fall back to odometry when the sensor readings do not match. In that case, odometry and the navigation module will behave equal. Since there is no redundancy left, there is no way to see that a fault has occurred.

As mobile robots often are equipped with many different types of sensors, and there are several algorithms that use a subset of the sensors, it is not difficult finding independent pose providers. An example of a basic system is to use odometry as one pose provider and an integrator that integrates motion control commands to position as a second pose provider.

An important property of a pose provider is if the estimation error is bounded or not. Inertial navigation systems and odometry have position errors that grow over time, while systems like SLAM and map based navigation systems typically have position errors that remain bounded. The approach presented here can handle both types of systems.

B. System model

To detect deviation between the pose providers, a model is used to predict the readings. A set of three state variables $r_i = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ is used for every pose provider i, which corresponds to rectilinear position and orientation of the robot, in the frame of each pose provider.

The particular choice of the state vector leads to a nearly linear system. Another possibility is to choose the state to be the true position augmented with a coordinate transform for each pose provider. Having a pose provider that slowly drifts, as for instance odometry, does however lead to cross couplings between noise sources and robot speed and is thus more difficult to track.

A standard first order dynamic model is used for the evolvement of the output from the pose providers:

$$\begin{aligned} x_{t+1} &= x_t + G_t(x_t)w_t \quad & cov(w_t) = Q_t(x_t) \\ y_t &= x_t + v_t & cov(v_t) = R_t \end{aligned} \tag{1}$$

where x is the aggregated state vector $\begin{bmatrix} r_1^T & r_2^T & \dots & r_p^T \end{bmatrix}^T$ for p pose providers. The process noise w corresponds to commanded motion as well as noise inherent within the pose providers. The model assumes that the control signal (speed reference u^c) of the robot is an unknown stochastic variable, and included in the process noise vector w. One can argue that the commanded motion of the robot is known and should be included as a deterministic input signal in (1). However, experiments have shown that this does not contribute significantly to detection performance. The increase in performance does not weigh up the added complexity of implementing the deterministic signal. On some systems, it might not even be possible to get this signal, for instance when the robot is controlled by the hardware directly or by another piece of software. If the control signal is available (speed reference), it can be externally integrated into pose using an appropriate dynamic model and then used as a pose provider in the present framework. The dynamic model for the relation between control signal and pose could be arbitrarily complex without affecting the structure of the fault detector, as only the output of the algorithm is read into the fault detector.

The measurements are the readings of the pose providers, and the influence of sensor noise is captured in the process noise w, and not the measurement noise v. This is further discussed in the remainder of the section.

C. Model input and process noise

The process noise w is a $2+(2+3)p$ vector which is used to model both the commanded motion of the robot as well as disturbances and imperfection within each pose provider. The latter will give rise to drift in the pose providers. In the following equations, the matrices are given for $p = 2$ pose providers. Extending to use more pose providers is straight forward.

$$w = \begin{bmatrix} u^c \\ u^{e,1} \\ u^{e,2} \\ u^{c,1} \\ u^{c,2} \end{bmatrix} \quad Q = \begin{bmatrix} Q^c & & & & \\ & Q^{e,1} & & & \\ & & Q^{e,2} & & \\ & & & Q^{c,1} & \\ & & & & Q^{c,2} \end{bmatrix} \tag{2}$$

$$G_t = \begin{bmatrix} G^1 & G^1 & 0 & I_3 & 0 \\ G^2 & 0 & G^2 & 0 & I_3 \end{bmatrix} \quad G^i = \begin{bmatrix} \Delta T \cos(\theta_i) & 0 \\ \Delta T \sin(\theta_i) & 0 \\ 0 & \Delta T \end{bmatrix} \tag{3}$$

Each part of the process noise vector represents different sources for driving the pose providers. The noise is assumed to be zero mean and white.

Common robot speed: The first and dominating part of the process noise vector is u^c, the common robot speed $\begin{bmatrix} u_x & u_\omega \end{bmatrix}^T$. As the robot moves, u^c is the part of the model that captures this. In absence of model errors, pose provider errors and noise, it would be the only nonzero part of the process noise. As previously mentioned, the commanded motion is considered to be an unknown stochastic variable. If the speed reference is known, or there is a model for how the robot speed evolves, there might be room for improvement on this part of the model. See the previous section for a discussion. There is however an advantage in considering the robot speed to be an unknown disturbance, as there is no need to implement a coupling from the reference speed signal (fed to the robot hardware) to the fault detector.

Pose provider robot frame drift: A part of the drift $u^{e,i}$ of the pose provider is modeled as additive input of the robot speed $u^{e,i} = \begin{bmatrix} v^{e,i} & \omega^{e,i} \end{bmatrix}^T$. Having a system that integrates signals from robot fixed speed sensors will be subject to this type of drift.

Pose provider Cartesian frame drift: As $u^{e,i}$ models drift relative to the robot frame, $u^{c,i}$ models drift in a Cartesian frame. $u^{c,i} = \begin{bmatrix} v_x^{c,i} & v_y^{c,i} & v_\theta^{c,i} \end{bmatrix}^T$ represents noise that models a drift in all states simultaneously, and has the effect of alleviating problems arising from linearization and model errors. An important assumption that is made, is that the noise is uncorrelated. The process noise gain matrix G is composed of blocks G^i which are linearized around the current heading angle θ_i of the corresponding pose provider. ΔT is the time elapsed since the last update.

The characteristics of the system changes with speed. For instance odometry has a larger drift the higher speed the robot operates at[1]. Other types of pose providers, like wireless localization, or inertial navigation, might have drift nearly independent of speed. To accommodate for these effects of changing speed, some elements of the noise intensity matrix Q are scaled with a factor k defined in (4). A similar scaling can be found in [13]. In this paper, a small offset α is added to the scaling factor k, to avoid unreasonable low noise at standstill. The offset also alleviates problems of having a slight delay in calculating the factor. Normalization constants $\bar{v}_i, i = x, y, \theta$ (see 4) are set to normal driving speed of the robot, and make k approximately 1 at normal speed, and $\alpha \ll 1$ at standstill. The reason for scaling the covariance is that adding independent uncertainty Q/n in n steps obtains Q total increase of uncertainty, independent of n.

$$k = \sqrt{\left(\frac{v_x}{\bar{v}_x}\right)^2 + \left(\frac{v_y}{\bar{v}_y}\right)^2 + \left(\frac{v_\theta}{\bar{v}_\theta}\right)^2} + \alpha \tag{4}$$

D. Measurement noise

As the imperfection of the pose providers is captured in the drift model, there is no regular measurement noise v, as all effects from sensor noise and algorithm shortcomings in the pose providers are represented in the process noise w. There will however be effects from scheduling jitter, which is especially important when implementing the fault detector in a non real-time system. Also, communication delays between sensors and receiving system might cause jitter. Effects from finite machine precision and interpolation errors might also be captured by the measurement noise. The measurement noise covariance matrix has been selected diagonal, where the elements have been chosen as a fraction of the typical motion between two time steps and the fraction corresponds to the size of the jitter compared to the sampling time. Having a nonzero R is also beneficial for the numerical properties of tracking the state, as $R = 0$ indicates that the measurements are exact, and can cause inconsistencies in the estimate.

E. Tracking the state

The pose providers are monitored using an extended Kalman filter (EKF). In this way, an estimate $\hat{x}$ and an associated covariance matrix P is calculated every time step. The purpose of tracking is not to get the estimate of the output, but instead to tell whether the pose providers agree or not. In the ideal case, the innovation (predicted output error) $e = y - C\hat{x}$ from the Kalman filter is Gaussian and white, given that the model assumptions hold and that linearization effects are negligible. When something abrupt happens like sensor malfunction, wheel slip or collision, the model will not be valid, and the innovation e will not be Gaussian and white. Thus, the innovation can be used to monitor the system.

At startup, the filter must be initialized. By setting the initial covariance $P_{0|-1}$ to a high value $\beta I,\ \beta \gg 1$, the filter will rapidly converge to the correct values. Having one or more pose providers start at a large nonzero position is thus not a problem. If the pose providers deliver data at different rates or readings are missing, the Kalman filter provides means to handle such situations. The Kalman filter theory is well developed, and provides several tools. See for instance [14]. An alternative to use EKF might be to compare the speeds from the pose providers directly, as done in [7]. This is however not straight forward, as different noise characteristics make it difficult to reason if changes are large or not. Comparing absolute sensors (GPS-like) and relative sensors like accelerometers is not easy. The naive approach of comparing, by differentiating the absolute position and comparing it to the integral of the accelerometer signal would suffer from noise amplification and drift.

F. Detecting abrupt changes

From the Kalman filter, the innovation e is obtained. The associated covariance will be $R_e = P + R$. A test statistic is obtained by the Mahalanobis distance $s_t = e^T R_e^{-1} e$ which will be χ^2 distributed with $3p$ degrees of freedom. All of the above is true when the model assumptions hold. A simple test of when to alarm for faults is to set a threshold on s_t. This can be done using a standard table, or from monitored levels of the test statistic.

However, this is sensitive to outliers in s_t, which can cause false alarms. To avoid this, s_t is fed into a detector that takes

[1]This reflects that the uncertainty per traversed distance is constant

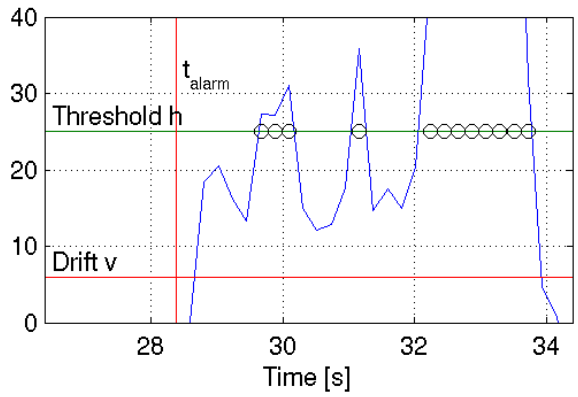

Fig. 3. Detail of an alarm from the CUSUM test: The blue curve is the test variable g_t. The alarms are marked with circles at the times they are raised, and the reported time t_{alarm} of the first alarm is marked with a vertical line.

the size and duration of the test statistic into account.

G. CUSUM test

The CUSUM test (see for instance [15]) is a test for detecting positive changes in a positive (noisy) scalar signal. In short, the test will alarm for a single sample being extremely large, or several consecutive samples being unexpectedly large. The algorithm is as follows:

$$g_t = g_{t-1} + s_t - v$$
$$if\ (g_t < 0)\ then\ t_{alarm} = t,\ g_t = 0$$
$$if\ (g_t > h)\ then\ raise\ alarm\ and\ set\ g_t = 0$$

The test will thus be less sensitive to outliers than simply thresholding on the test statistic directly. There are two parameters in the test, the drift value $v > 0$ and the alarm threshold $h > v$. A simplified description is that v is related to what level the input to the detector normally has, and h is adjusted to trade off false alarms and risk of missed detection. Since the Kalman filter after a disturbance will slowly adapt to the new data, faults will only give residuals under a limited time. Therefore, the detector must not be too slow. It is also important that the alarm is not delayed more than necessary after an unexpected event, in order not to worsen the situation.

The CUSUM test can estimate when the change in the signal has occurred [15]. This is given by the last reset time t_{alarm}. An example of an alarm time obtained with this method is shown in Fig. 3. Upon alarm, this information can be used to perform recovery actions on the navigation system. For instance, estimation of the current position can be carried out on stored sensor data from the time of when the fault was believed to occur and forward, where faulty sensors are excluded from the analysis. The memory of old sensor data can be of finite length, as a fault alarm probably will come rather quickly or not at all.

H. Setting filter parameters

The approach has shown to be quite insensitive to the parameters. For the parameter values used in the experiments in this paper, see [16], which also includes some guidelines for how to select them.

V. Implementation

To test the proposed method, it has been implemented on a PowerBot robot from ActivMedia. The robot is equipped with odometry and a SICK laser scanner among other sensors. The odometry system can directly be used as a pose provider. As a second provider, a scan matching routine was implemented which provides a source of position information independent of odometry.

A. Scan matching

A memoryless scan matching algorithm is implemented, inspired by [17]. As laser scans come in, they are matched to the previous scan and the relative motion is extracted. The pose of the robot is integrated using the relative motion. Using it this way, one can regard the scan matcher as laser based odometry. Since the match of two consecutive scans will have an error that is mainly independent of the robot motion, the error model with Cartesian noise dominates the drift. The size of the scan matcher error is for the speed range considered here nearly independent of the speed, and the corresponding parts of Q can thus be held constant.

VI. Experimental results

A. Experiment setup

During the experiments, the robot moved around autonomously using the Nearness Diagram algorithm [18]. The experimental environment is a room furnished with sofas, tables and bookshelves to resemble a domestic living room in size and materials. As the robot is moving around in the room, it has been pushed to induce wheel slip. The algorithm was run in realtime on a standard laptop computer, connected via wireless network to the robot and its sensors.

B. Results

The filter has been tested during several sessions, with both fault free and faulty data. No false alarms were triggered during the fault free sessions, even if the robot was forced to do fast moves with the obstacle avoidance behaviour. This was triggered by suddenly walking into the field of view of the laser scanner, close to the robot. Faults were injected by pushing the robot manually, while it was moving autonomously between waypoints. An example of output from such an experiment with faulty data is shown here. The Mahalanobis distance s_t from the Kalman filter is shown in Fig. 4. One can clearly see when the robot has been pushed (multiple times). One can also see that the residual is very low during the beginning of the test, when no fault is present. The corresponding test variable g_t and its threshold is shown in Fig. 5. Each time it exceeds the threshold, g_t is reset to zero and an alarm is raised. The alarm times are marked with circles in the figure. Several alarms are raised after each other. A detail of the first alarm is shown in Fig. 3.

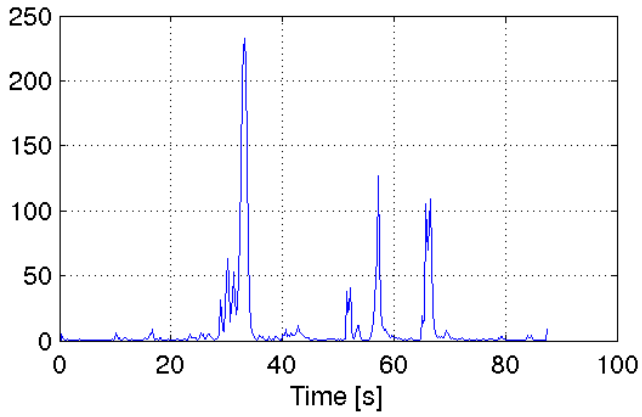

Fig. 4. Weighted residual s_t from the Kalman filter from test with robot being pushed several times. The pushes are clearly visible as peaks in s_t. The first 25 seconds are fault free and the residual is small.

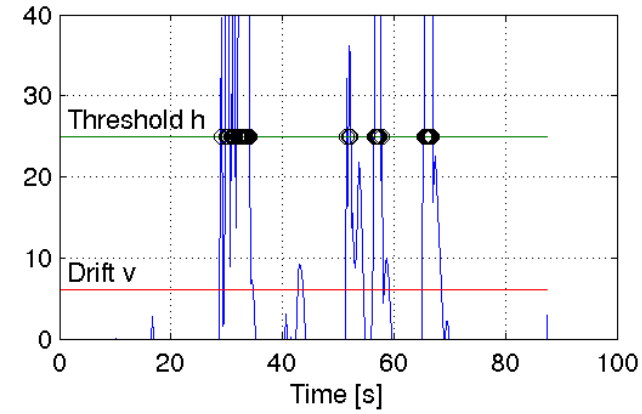

Fig. 5. Output from CUSUM test with robot being pushed several times. When the test variable g_t exceeds the threshold, an alarm is generated (marked with circles) and it is reset. The weighted residual from Fig. 4 is used as input for the detector.

VII. Summary and conclusions

The proposed system is shown to be able to detect wheel slip in realtime. The system handles initially unknown coordinate system and different noise characteristics. The approach is demonstrated using two pose providers, odometry and scan matching. It is shown that it is possible to detect faults at a higher level than processing the sensor data directly. Existing navigation modules can be used, and the parameters that are needed for the module are quite easy to obtain. By reusing existing navigation modules, domain knowledge built into these systems can be utilized. This way, modularity of the system is kept.

No specific fault model is needed, which is beneficial regarding the effort needed for implementation, but does not take advantage of information about faults that may be known. The algorithm does not isolate faults (indicate the source of fault), which maybe can be achieved by combining pose providers pairwise.

Returning to the situation shown in Fig. 1 (the robot skidding on the floor), the proposed fault detector would have detected the abnormal situation.

VIII. Future work

The crucial part for detection using the proposed framework is that the fault model is accurate. In some situations, pose providers can adjust their position estimate abruptly, which might not be captured by the drift model. Such cases may be handled better, if the pose provider can give information of its uncertainty.

References

[1] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence*, vol. 114, 1999.

[2] D. Avots, E. Lim, R. Thibaux, and S. Thrun, "A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments," in *International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[3] D. Hähnel, D. Schulz, and W. Burgard, "Map building with mobile robots in populated environments," in *International Conference on Intelligent Robots and Systems IROS*, 2002.

[4] P. Jensfelt and S. Kristensen, "Active global localisation for a mobile robot using multiple hypothesis tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, Oct. 2001.

[5] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[6] S. Roumeliotis, G. Sukhatme, and G. Bekey, "Sensor fault detection and identification in a mobile robot," in *International Conference on Intelligent Robots and Systems (IROS)*, 1998.

[7] Y. Lu, E. Collins, and M. Selekwa, "Parity relation based fault detection, isolation and reconfiguration for autonomous ground vehicle localization sensors," in *24th Army Science Conference*, 2004.

[8] B. J. and L. Feng, "Gyrodometry: A new method for combining data from gyros and odometry in mobile robots," in *International Conference on Robotics and Automation (ICRA)*, 1996.

[9] V. Verma, R. Simmons, G. Gordon, and S. Thrun, "Particle filters for fault diagnosis," *IEEE Robotics and Automation Magazine*, 2004.

[10] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2d range scans," in *CVPR94*, 1994, pp. 935–938.

[11] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[12] A. M. Ladd, K. E. Bekris, G. Marceau, A. Rudys, L. E. Kavraki, and D. S. Wallach, "Using wireless ethernet for localization," in *International Conference on Intelligent Robots and Systems IROS*, 2002.

[13] K. S. Chong and L. Kleeman, "Accurate odometry and error modelling for a mobile robot," in *International Conference on Robotics and Automation (ICRA)*, 1997.

[14] M. S. Grewal and A. P. Andrews, *Kalman Filtering, Theory and Practice*. Prentice Hall, 1993.

[15] F. Gustafsson, *Adaptive filtering and change detection*. Wiley, September 2000.

[16] P. Sundvall, "Parameters for an experiment with a fault detection routine," Tech. Rep., 2006, Available at www.ee.kth.se.

[17] C. Früh and A. Zakhor, "Fast 3D model generation in urban environments," in *Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, 2001.

[18] J. Minguez and L. Montano, "Nearness diagram navigation(ND): A new real-time collision avoidance approach," in *International Conference on Intelligent Robots and Systems (IROS)*, 2000.