Proactive Model for Handling Conflicts in Sensor Data Fusion Applied to Robotic Systems

Gilles Neyens, Denis Zampunieris
University of Luxembourg
Department of Computer Science
{*gilles.neyens, denis.zampunieris*}*@uni.lu*

Keywords: Decision Support System, System Software, Proactive Systems, Robotics

Abstract: Robots have to be able to function in a multitude of different situations and environments. To help them achieve this, they are usually equipped with a large set of sensors whose data will be used in order to make decisions. However, the sensors can malfunction, be influenced by noise or simply be imprecise. Existing sensor fusion techniques can be used in order to overcome some of these problems, but we believe that data can be improved further by computing context information and using a proactive rule-based system to detect potentially conflicting data coming from different sensors. In this paper we will present the architecture and scenarios for a generic model taking context into account.

1 INTRODUCTION

Robots generally have to be able to adapt to all kinds of different situations. In order to manage this task, they have to combine data from several different sensors which will then be used by the robot to make decisions. Sensor fusion is used to achieve this and reduce the uncertainty the resulting information would have in case the sensors were used individually.

A lot of these techniques are carried out on a set of homogeneous sensors while others exist that are used on heterogeneous sensors, like for example to calculate a more accurate position of the robot (Bostanci et al., 2018; Nemra and Aouf, 2010).

Similarly, there exists work on integrating a trust model in to the data fusion process(Chen et al., 2017). The approach is calculating trust values for each sensor based on historical data of a sensor, the current data, and a threshold representing the upper bound of the maximum allowed difference between the current data and the historical data.

While the previous method attributes trust or confidence values to sensors based on historical data, it is only based on the data of the same sensor. Other methods try to improve this by taking context into account like (Tom and Han, 2014) or (Akbari et al., 2017) in which they use rules to adapt the parameters of a kalman filter based on the current context.

The model we propose in this paper will be set between the sensors and the final decision making of the robot, and will work together with other fusion methods in order to improve the information that the robot receives. The goal is to reduce the probability of malfunctioning or corrupted sensors negatively affecting the robot. The system will in a first step use classification algorithms tailored for each sensor in order to obtain confidence/trust values for each sensor and in a second step use a priori knowledge about relations between sensors and a proactive rule-based system to further refine the confidence for each sensor and potentially improve the data based on the current context.

In the next section, we are going to give a quick overview of existing works in the field. In section 3 we will propose our model and in section 4 the internal workings of the model will get explained in detail. Finally, we will conclude and give some prospects for future work.

2 STATE OF THE ART

In the past years several sensor fusion methods have been used to aggregate data coming from different sensors. Depending on the level of fusion, different techniques have been used. In robotics the most popular method for low-level data is the Kalman filter which was developed in 1960 (Kalman, 1960) along with its variants the extended Kalman filter or the un-

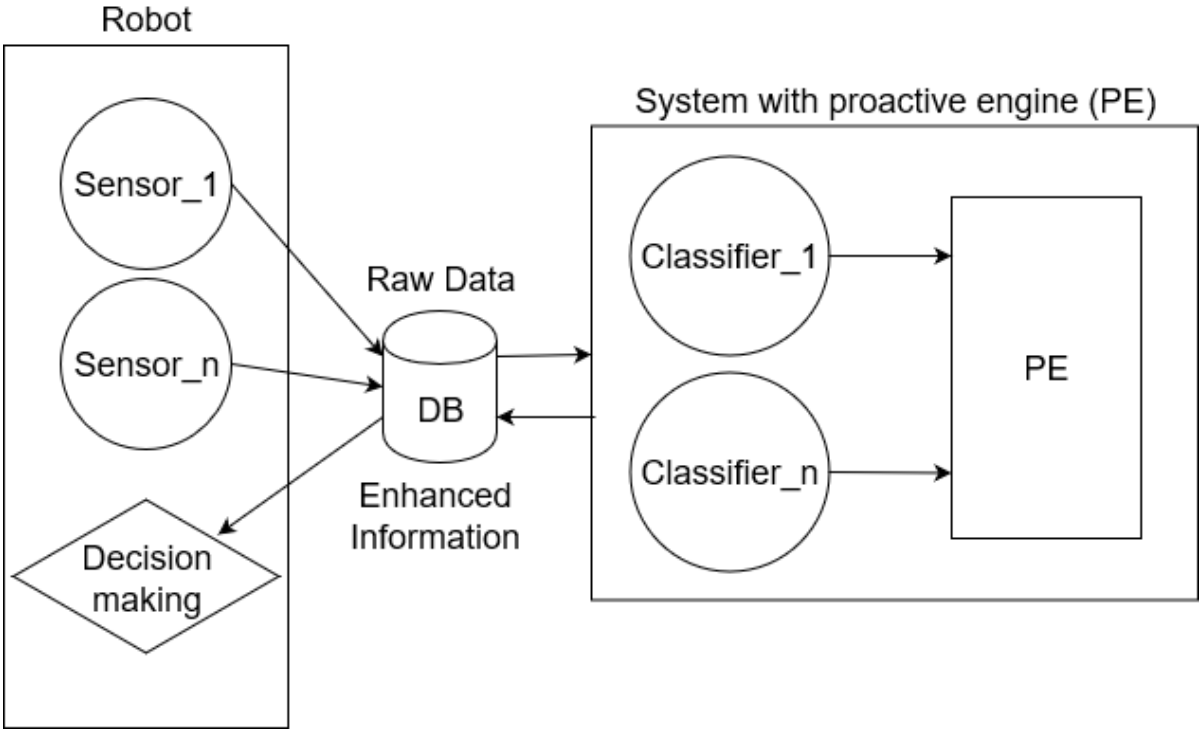

Figure 1: System architecture

scented Kalman filter who are often used in navigation systems (Hide et al., 2003; Sasiadek and Wang, 1999), but also other methods like the particle filter. On a decision level other methods like the Dempster Shafer theory (Dempster, 1968; Shafer, 1976) is used.

For the purpose of this paper, we are going to focus on work done on Kalman and particle filters. Some time ago, studies which used either of these, often did not take sensor failures and noise into account, which made these solutions vulnerable in harsh and uncertain environments. A recent study partly addressed these issues by using a particle filter method in combination with recurrent neural networks (Turan et al., 2018). This solution managed to correctly identify situations in which sensors were failing but did not yet take sensor noise into account. In (Bader et al., 2017) the authors propose a fault tolerant architecture for sensor fusion using Kalman filters for mobile robot localization. The detection rate of the faults injected was 100%, however the diagnosis and recovery rate is lower at 60%. The study in (Yazdkhasti and Sasiadek, 2018) proposed two new extensions to the kalman filter, the Fuzzy Adaptive Iterated Extended Kalman Filter and the Fuzzy Adaptive Unscented Kalman Filter in order to make the fusion process more resistant to noise. In (Kordestani et al., 2018), the authors used the extended kalman filter in combination with bayesian method and managed to detect and predict not only individual failures but also simultaneous occurring failures, however no fault handling was proposed.

3 MODEL

3.1 Architecture

The general architecture of our model is shown in Fig. 1. The sensors of the robot send their data to our system where the classifiers attribute a confidence value for each sensor individually (Neyens and Zampunieris, 2017). The proactive engine then uses the results from the classifiers as well as the data in the knowledge base in order to further check the trustworthiness of the different sensors by detecting and resolving potential conflicts between sensors (Neyens, 2017; Neyens and Zampunieris, 2018).

The robot itself will pass the data coming from the sensors to our system where it will get processed. It then will get enhanced data along with information on which sensors to trust from our system, which it will then use to make a decision.

In our system the data will first get processed by some classification algorithms, like for example Hidden Markov Models or Neural Networks, in or-

Table 1: Sensor registration table

Sensor name	List of properties	Minimum confidence	Grace period	History Length
GPS	position, speed	0.8	15s	1000
Accelerometer	acceleration	0.8	12s	800
Light	lightlevel	0.5	30s	50
...	...	...	...	...

der to determine if a sensor is failing or still working correctly, along with a confidence value for each sensor. The results from the classification will then be passed to our rule-based engine where it will get passed through a series of steps which will be described in Fig. 2.

First we will describe what information our system needs in order to fulfill its tasks. For every sensor we need to know different variables as shown in table 1. The name of the sensor is the identifier for a given sensor and will be unique. The list of properties are the properties that can be computed from the data from a given sensor, for example for a GPS it would be position and speed. The minimum confidence is the minimum value for the confidence computed based on the classifiers for a sensor in order to trust a sensor. Grace period is the time a sensor will still be used for computations after deemed as untrustworthy. The history length is the amount of past data kept for a given sensor.

In order for the system to know how different properties affect each other or how they can be calculated we need a knowledge base for these properties. The properties can either affect other properties (e.g. a low light level could affect the camera and thus image recognition modules) or they can be calculated based on past data and other properties (e.g. position). For the first case, a range of values is defined for each property that defines the allowed values for a property before it starts to affect other properties. For the second case, a list of operations is needed, that allows the system to know how to use past data and other properties to calculate an estimate of the given property.

Finally, the scenarios described in section 4 should be able to synchronize between themselves and operate on the same data as the scenarios in the previous steps of the flow did. For this, we divide the data into chunks and attribute a number called execution number to each chunk. This number along with the execution step number then allows each scenario to decide on which data it should currently operate.

3.2 Proactive system

We use Artificial Neural Networks (ANNs) to first analyse the data, but only basing the model on them would make it very difficult to follow decisions. Also, as we want the system to be able to react to all kinds of different contexts, training the ANNs would be very difficult. Therefore we believe that using a rule-based system is better suited for this task. In our model we use the proactive engine developed in our team over the years.

The concept of proactive computing was introduced in 2000 by Tennenhouse (Tennenhouse, 2000) as systems working for and on behalf of the user on their own initiative (Salovaara and Oulasvirta, 2004). Based on this concept a proactive engine (PE) which is a rule-based system was developed (Zampunieris, 2006). The rules running on the engine can be conceptually regrouped into scenarios with each scenario regrouping rules that achieve a common goal (Zampunieris, 2008; Shirnin et al., 2012). A Proactive Scenario is the high-level representation of a set of Proactive Rules that is meant to be executed on the PE. It describes a situation and a set of actions to be taken in case some conditions are met (Dobrican et al., 2016).

The PE executes these rules periodically. The system consists of two FIFO queues called currentQueue and nextQueue. The currentQueue contains the rules that need to be executed at the current iteration, while the nextQueue contains the rules that were generated during the current iteration. At the end of each iteration the rules from the nextQueue will be added to the currentQueue and the next Queue will be emptied.
A rule consists of any number of input parameters and five execution steps (Zampunieris, 2006). These five steps have each a different role in the execution of the rule.

1. Data acquisition
   During this step the rule gathers data that is important for its subsequent steps. This data is provided by the context manager of the proactive engine, which can obtain this data from different sources such as sensors or a simple database.
2. Activation guards
   The activation guards will perform checks based on the context information whether or not the conditions and actions part of the rule should be executed. If the checks are true, the activated variable of this rule will be set to true.
3. Conditions
   The objective of the conditions is to evaluate the

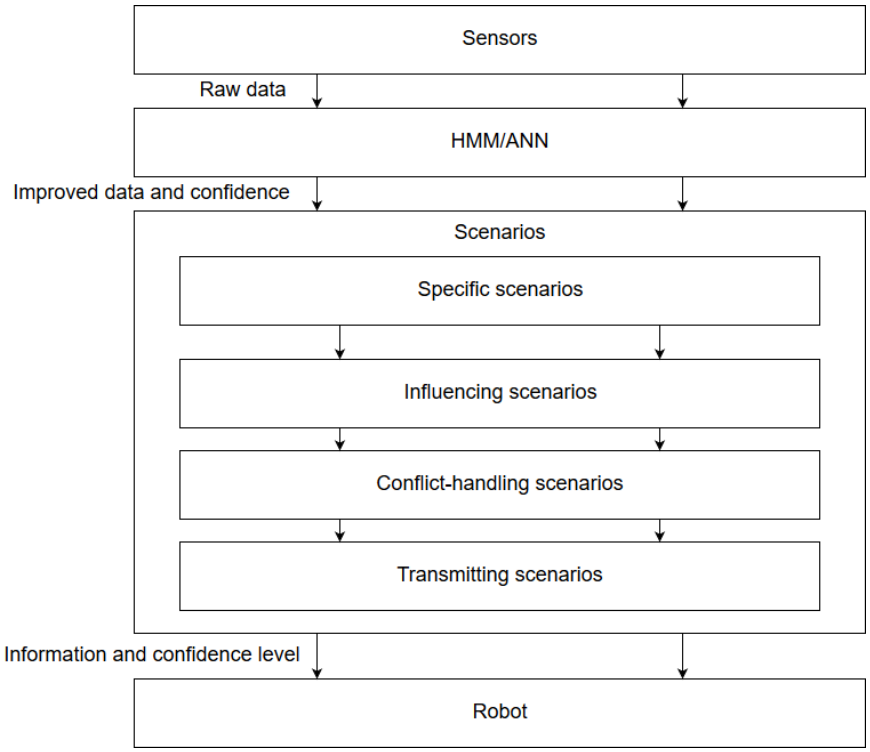

Figure 2: Scenario Flow

context in greater detail than the activation guards. If all the conditions are met as well, the Actions part of the rule is unlocked.

4. Actions
   This part consists of a list of instructions that will be performed if the activation guards and condition tests are passed.

5. Rule generation
   The rule generation part will be executed independently whether the activation guards and condition checks were passed or not. I this section the rule creates other rules in the engine or in some cases just clones itself.

During an iteration of the PE, each rule is executed one by one. The algorithm to execute a rule is presented in Fig. 3. The data acquisition part of the rule is run first and if it fails none of the other parts of the rule is executed. The rules can then be regrouped into scenarios. Multiple scenarios can be run at the same time and can trigger the activation of other scenarios and rules.

```
data=getData(dataRequest);
activated=false;
if(checkActivationGuards()){
   activated=true;
   if(checkConditions()){
      executeActions();
   }
}
generateNewRules();
discardRuleFromSystem(currentRule);
```

Figure 3: The algorithm to run a rule

4 SCENARIOS

In Fig. 2, the flow of the data in the system is described. It gets first passed from the sensors of the robot to our system where the classifiers assign a confidence value to the sensors. It will then get passed on to the rule-based system where the confidences will get updated and the data will get improved using different scenarios. Finally, the data and confidence level will get sent back to the robot. This section is dedicated to the different types of scenarios running on the rule-based engine.

4.1 Scenario flow

1. Specific scenarios
   During this step scenarios specifically designed for a single sensor try to further improve the data and/or confidence of the sensor.
2. Influencing scenarios
   This is the first step in which context information will be taken into account in order to adapt the confidence of some sensors. Data coming from some sensors as well as their confidence will be used in order to compute the context and to decide whether the confidence of related sensors should be adapted.
3. Conflict-handling scenarios
   In a third step, the system will solve the conflicts that can occur between the data/information coming from different sensors. Those conflicts can either occur because the some sensors give data for the same property (e.g.: distance to object based on infrared and ultrasonic sensors) or because the property that a sensor gives the data for can be computed based on several other sensors (e.g. position). Based on the confidence attributed to the sensors in the previous steps it will then be decided which sensors to trust.
4. Transmitting scenarios
   Finally the output information and confidence levels coming from our system will then be transferred to the robot, either through the initial sensor channels or through an additional virtual sensor channel if the robot's configuration allows it.

4.2 Implementation

In the system itself, the scenarios are implemented as a set of rules. They can be put into 3 main categories.

1. Check for data
   These rules check whether a Scenario has any data to operate on and triggers the next rules in the scenario if this is the case. The skeleton for these rules is generic for which the pseudo code can be seen in Fig. 4.
2. Computation
   These rules are specific for every step described in the previous section and can be very different depending on the step, sensor and property they are operating on. An example of a computation rule is shown in Fig. 5 which computes how the confidence of an infrared sensor is changed based on the amount of light around the robot which could potentially affect the accuracy of the infrared distance sensor.

```
dataAcquisition() {
  dataExists=checkDataExists(sensor
      ,property,execution_number,
      EXECUTION_STEP);
}

boolean activationGuards() {
  return dataExists;
}

boolean conditions() {
  return true;
}

actions() {
 this.execution_number++;
}

boolean rulesGeneration() {
  addRule(self);
  startScenarios(execution_number,
      EXECUTION_STEP);
  return true;
}
```

Figure 4: Generic check data rule

3. Save data
   These rules simply save the results from the computation in an appropriate manner. The skeleton for these rules is generic as well and the pseudo code is shown in Fig. 6.

4.3 Example

In this section, we are going to discuss an example with multiple sensors that could lead to conflicting information, that need to be resolved. The situation is the following: A GPS has been working correctly until recently, but now it has started to malfunction giving a slightly inaccurate position as the GPS antenna is coated in ice. The system executes its loop:

1. As the position is only slightly inaccurate, the classifiers still attribute a high confidence value to the GPS.
2. For the same reason, the specific sensor scenarios do not detect that anything is wrong and do not change the confidence value.
3. One of the influencing scenarios uses the temperature sensor and camera, detects that the tempera-

```
dataAcquisition() {
  lightdata=getData("lightsensor","
      lightlevel",execution_number,
      EXECUTION_STEP);
  infrareddata=getData("
      infraredsensor","distance",
      execution_number,
      EXECUTION_STEP);
}

boolean activationGuards() {
  return true;
}

boolean conditions() {
  return getLightLevel(lightdata)>
      threshold;
}

actions() {
  infraredconfidence=getConfidence(
      infrareddata);
  lightconfidence=getConfidence(
      lightdata);
  lightlevel= getLightLevel(
      lightdata);
  newconfidence=updateConfidence(
      infraredconfidence,
      lightconfidence, lightlevel);
  addRule(SaveRule(getValue(
      infrareddata),"infraredsensor
      ","distance",newconfidence,
      execution_number,
      EXECUTION_STEP+1));
}

boolean rulesGeneration() {
  this.execution_number++;
  return true;
}
```

Figure 5: Influencing rule

ture is below 0 and that there might be snow and because the confidences of these sensors are high enough concludes that it might have an effect on the GPS and thus updates the confidence of the GPS by calling the updateConfidence() function like shown in Fig. 5.

4. One of the conflict handling scenarios compares the current values given by the GPS to a value calculated based on the old position, speed, acceleration and direction and detects that the discrepancy between the two is above a threshold. As the GPS confidence currently is quite low and the average of the confidences of the sensors used to calculate an estimated position is higher and every single one of them is above a minimum threshold, the values for the position given by the GPS are replaced by the calculated ones.

```
dataAcquisition() {
}

boolean activationGuards() {
  return true;
}

boolean conditions() {
  return true;
}

actions() {
  insertData(value,sensor,property,
      confidence,execution_number,
      EXECUTION_STEP));
}

boolean rulesGeneration() {
  return true;
}
```

Figure 6: Save rule

5. Finally the calculated values are passed to the robot.

5 CONCLUSION

In this paper, we proposed a generic model for context-based handling in sensor data fusion. In this model, we use proactive scenarios running on a rule-based engine in addition to of a layer of classifiers in order to improve quality and the trust level of the information that the robot finally uses to make his decisions for reaching its objectives.

The proposed model takes the context of the robot and its environment into account, which allows it to better adapt to changes in the overall situation than traditional sensor fusion techniques on their own.

6 FUTURE WORK

As some robots in the real world have to be able to operate in different situations, in the next steps we want to verify and validate different properties for our generic model like robustness and resilience as these properties are amongst the main ones requested for these robots and are two important properties that our model tries to provide them with respect to decision-making based on multiple sensors data flows. These properties can be tested by deliberately making sensors fail or malfunction and checking whether the output is correct. To allow us to have full control and knowledge about the experiments, they will be done in the simulation environment Webots. It also allows us to generate the amount of data necessary to train the first layer of the model.

REFERENCES

Akbari, A., Thomas, X., and Jafari, R. (2017). Automatic noise estimation and context-enhanced data fusion of imu and kinect for human motion measurement. In *2017 IEEE 14th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 178–182. IEEE.

Bader, K., Lussier, B., and Schön, W. (2017). A fault tolerant architecture for data fusion: A real application of kalman filters for mobile robot localization. *Robotics and Autonomous Systems*, 88:11 – 23.

Bostanci, E., Bostanci, B., Kanwal, N., and Clark, A. F. (2018). Sensor fusion of camera, gps and imu using fuzzy adaptive multiple motion models. *Soft Computing*, 22(8):2619–2632.

Chen, Z., Tian, L., and Lin, C. (2017). Trust model of wireless sensor networks and its application in data fusion. *Sensors*, 17(4):703.

Dempster, A. P. (1968). A generalization of bayesian inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, 30(2):205–247.

Dobrican, R.-A., Neyens, G., and Zampunieris, D. (2016). A context-aware collaborative mobile application for silencing the smartphone during meetings or important events. *International Journal On Advances in Intelligent Systems*, 9(1&2):171–180.

Hide, C., Moore, T., and Smith, M. (2003). Adaptive kalman filtering for low-cost ins/gps. *The Journal of Navigation*, 56(1):143–152.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

Kordestani, M., Samadi, M. F., Saif, M., and Khorasani, K. (2018). A new fault prognosis of mfs system using integrated extended kalman filter and bayesian method. *IEEE Transactions on Industrial Informatics*, pages 1–1.

Nemra, A. and Aouf, N. (2010). Robust ins/gps sensor fusion for uav localization using sdre nonlinear filtering. *IEEE Sensors Journal*, 10(4):789–798.

Neyens, G. (2017). Conflict handling for autonomic systems. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 369–370. IEEE.

Neyens, G. and Zampunieris, D. (2017). Using hidden markov models and rule-based sensor mediation on wearable ehealth devices. In *Procedings of the 11th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Barcelona, Spain 12-16 November 2017*. IARIA.

Neyens, G. I. F. and Zampunieris, D. (2018). A rule-based approach for self-optimisation in autonomic ehealth systems. In *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, pages 1–4. VDE.

Salovaara, A. and Oulasvirta, A. (2004). Six modes of proactive resource management: a user-centric typology for proactive behaviors. In *Proceedings of the third Nordic conference on Human-computer interaction*, pages 57–60. ACM.

Sasiadek, J. and Wang, Q. (1999). Sensor fusion based on fuzzy kalman filtering for autonomous robot vehicle. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 4, pages 2970–2975. IEEE.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Shirnin, D., Reis, S., and Zampunieris, D. (2012). Design of proactive scenarios and rules for enhanced e-learning. In *Proceedings of the 4th International Conference on Computer Supported Education, Porto, Portugal 16-18 April, 2012*, pages 253–258. SciTePress–Science and Technology Publications.

Tennenhouse, D. (2000). Proactive computing. *Communications of the ACM*, 43(5):43–50.

Tom, K. and Han, A. (2014). Context-based sensor selection. US Patent 8,862,715.

Turan, M., Almalioglu, Y., Gilbert, H., Araujo, H., Cemgil, T., and Sitti, M. (2018). Endosensorfusion: Particle filtering-based multi-sensory data fusion with switching state-space model for endoscopic capsule robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE.

Yazdkhasti, S. and Sasiadek, J. Z. (2018). Multi sensor fusion based on adaptive kalman filtering. In *Advances in Aerospace Guidance, Navigation and Control*, pages 317–333, Cham. Springer International Publishing.

Zampunieris, D. (2006). Implementation of a proactive learning management system. In *Proceedings of" E-Learn-World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education"*, pages 3145–3151.

Zampunieris, D. (2008). Implementation of efficient proactive computing using lazy evaluation in a learning management system (extended version). *International*

Journal of Web-Based Learning and Teaching Technologies, 3:103–109.