Proactive Middleware for Fault Detection and Advanced Conflict Handling in Sensor Fusion

Gilles Neyens and Denis Zampunieris

University of Luxembourg,
2, avenue de l'Université
L-4365 Esch-sur-Alzette, Luxembourg
{gilles.neyens, denis.zampunieris}@uni.lu

Abstract. Robots traditionally have a wide array of sensors that allow them to react to the environment and make appropriate decisions. These sensors can give incorrect or imprecise data due to malfunctioning or noise. Sensor fusion methods try to overcome some of these issues by using the data coming from different sensors and combining it. However, they often don't take sensor malfunctioning and a priori knowledge about the sensors and the environment into account, which can produce conflicting information for the robot to work with. In this paper, we present an architecture and process in order to overcome some of these limitations based on a proactive rule-based system.

Keywords: Rule-based systems · Sensor fusion · Conflict handling

1 Introduction

Sensor fusion is used in IT and robotic systems to combine the data coming from different sensors in order to reduce the uncertainty the resulting information would have in case the sensors were used individually.

A lot of the existing work in sensor fusion is carried out on a set of homogeneous sensors and while there exist multi-sensor fusion techniques that are used on heterogeneous sensors, used for example to calculate a more accurate position of the robot[6, 21], most do not take into account the known relations between sensors of different types. Also often they do not provide means for handling noise, malfunctioning or corrupted sensors.

Similarly, there exists work on integrating a trust model into the data fusion process[8]. 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 gives trust or confidence values to sensors based on historical data, it is only based on data of the same sensor. Other methods try to improve this by taking context into account like [29] or [1] 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 reason about the current context of the system, and further refine the confidence for each sensor and potentially improve the data. The notion of proactive computing has been introduced in 2000[28] for systems that work for and on behalf of the user on their own initiative[24]. In this case, the robot will be the user and the proactive system can be tailored to handle a range of different scenarios in which data coming from different sensors is conflicting with each other.

In the next section, we are going to present related work in the domain of classification algorithms, sensor fusion, and conflict handling as well as the concept of proactive computing that our rule-engine is based on. In section 3, we introduce the general architecture of our system along as the structure for the a priori knowledge that is required by our system. Section 4 contains the method for resolving the conflicts between different sensors. In section 5 the evaluation of our system is described, and we will finish with a conclusion and possible future work.

2 Related work

2.1 Time series classification algorithms

In our system, we are going to mainly use 2 approaches for time series classification: Hidden Markov Models (HMMs) and Artificial Neural Networks (ANNs).

The mathematical foundations for HMMs have been developed in 1966 and the following years by Baum[3–5]. Over the years they have been used in a lot of different fields, be it speech recognition[11, 14, 22], failure detection[23] or healthcare applications[13]. More recent studies also have successfully used them for the diagnosis of industrial components[7].

The beginnings of ANNs go back to 1943[19]. Their use in sensor failure detection started in the 90s[20]. During the last years they have been successfully used in failure detection for gyroscopes, either on their own[15] or in combination with a fuzzy voter and fuzzy rules[18].

While the work in failure detection for some sensors already gave good results, it did not take other sensors of different types into account that could have provided necessary information to detect if something was wrong. In our system, we will, therefore, use the described approaches only as a first step of a whole sensor fault detection, sensor fusion, and conflict handling process.

2.2 Sensor fusion and conflict handling

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[16] along with its variants the extended Kalman filter or the unscented Kalman filter who are often used in navigation systems[12, 25], but also other methods like the particle filter. On a decision level other methods like the Dempster Shafer theory[9, 26] is used.

In this work, we are going to focus on work done on Kalman and particle filters. Studies that 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[30]. This solution managed to correctly identify situations in which sensors were failing but did not yet take sensor noise into account. In [2] 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 [32] 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 [17], 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.

Our system will provide this fault handling in order to provide better data for the robot, by using a proactive system in combination with classification algorithms.

2.3 Proactive computing

The concept of proactive computing was introduced in 2000 by Tennenhouse[28] as systems working for and on behalf of the user on their own initiative[24]. Based on this concept a proactive engine (PE) which is a rule-based system was developed[33]. The rules running on the engine can be conceptually regrouped into scenarios with each scenario regrouping rules that achieve a common goal[34, 27]. 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[10].

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[33]. These five steps have each a different role in the execution of the rule.

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

Fig. 1. The algorithm to run a 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 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. 1. 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.

3 System architecture

The general architecture of the system is shown in Fig. 2. The sensors of the robot send their data to our system where the classifiers attribute a confidence value for each sensor individually. 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.

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.

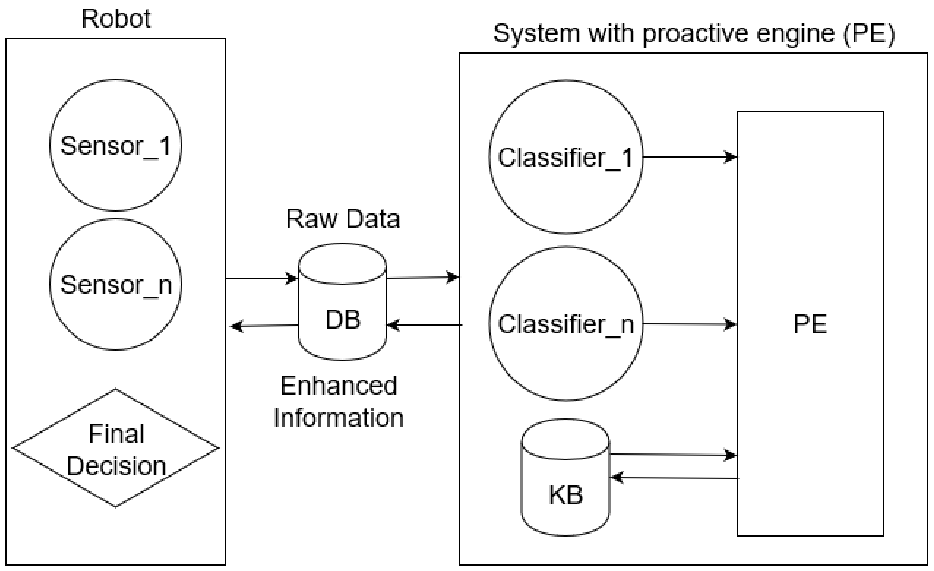


Fig. 2. System Structure

In our system, the data will first get processed by some classification algorithms, like for example Hidden Markov Models or Neural Networks, in order 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 to detect and handle potential conflicting information. This process will be described in section 4.

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.

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
...	...	...	...	...

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.

4 Conflict detection and handling

The whole process used by our system to handles potential conflicting information from the different sensors can be separated into 3 steps:

1. Updating the sensors confidence levels based on known interactions between sensors
2. Estimating values for sensors that can be calculated based on data from other sensors.
3. Deciding which sensors to trust

The first step is described in the pseudo code shown in Fig. 3. For every sensor, we fetch the property or list of properties they provide. For each of these properties, we then check whether this property can be influenced by other properties. If this is the case we fetch the sensor with the currently highest confidence that provides this property. This sensor also has to have a greater confidence value than its specified minimum confidence. Finally, we check if the data for this sensor is outside the acceptable range defined in the knowledge base and if this is the case we update the confidence for the initial sensor.

The second step is described in the pseudo code in Fig. 4. For every sensor, we again get the list of properties they provide along with the last known data. The system then checks in the knowledge base whether it is possible to calculate an estimate of this property based on data from other sensors. If this is the case

```
for sensor in sensorList{
        propertyList=GetListOfProperties(sensor, DB);
        for property in propertyList{
                relationList=CheckRelations(property, KB);
                for relation in relationList{
                        updateConfidence(sensor, relation, DB);
                }
        }
}
```

Fig. 3. Update confidence (KB=Knowledge base, DB=Data base)

```
for sensor in sensorList{
        propertyList=GetListOfProperties(sensor, DB);
        currentData=getCurrentData(sensor, DB);
        for property in propertyList{
                relatedPropList=getRelatedProperties(property
                    ,KB);
                action=getAction(property, KB);
                for relatedProp in relatedPropList{
                        relatedSensorWithHighestConfidence=
                            getMostTrustedSensor(relatedProp,
                             DB);
                        dataRelatedProp.add(
                            relatedSensorWithHighestConfidence
                            .getData());
                }
                pastDataOfProperty=getPastData(property, DB);
                estimate=calculateEstimate(action,
                    pastDataOfProperty, dataRelatedProp);
                saveEstimate(sensor, property, currentData,
                    estimate, DB);
        }
}
```

Fig. 4. Estimate calculation (KB=Knowledge base, DB=Data base)

it does so based on the past data of the sensor and the current data of related sensors and saves the results.

Finally, Table 2 shows the results from the 2 previous steps. In the last step, we compare the estimates to the actually delivered data from the sensors. If the difference between the two is larger than the predefined error margin we check which data should be trusted based on the current confidence values of the sensor and of the related sensors used to calculate the estimate. For the related sensors, we first check whether all of them meet their minimum confidence criteria. If this is the case we take the average of the confidences and check whether it is higher than the current confidence of the initial sensor. The information passed to the robot will then include the final confidence level for every sensor and either the original raw data or the estimated data based on the previous computation.

Table 2. Estimate calculation results table

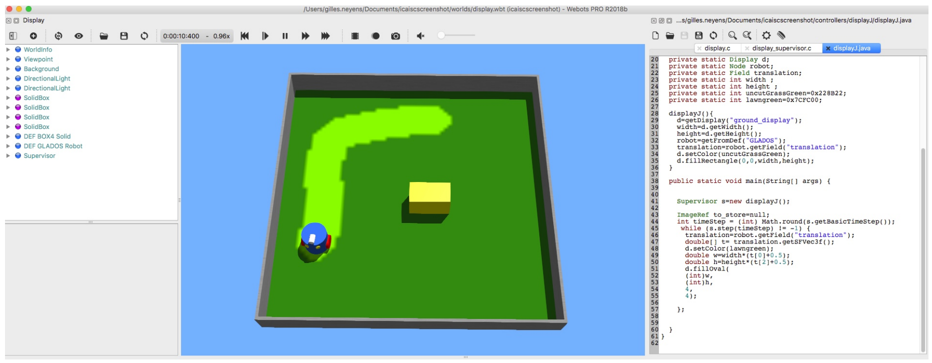
Sensor name	property	Confidence	Estimate	Related sensors confidence
GPS	position	0.2	49.82, 6.13, 321.36	0.8, 0.9, 0.78
...	...	...	...	...

5 Experimental Evaluation

The evaluation for our system is work under development via a proof of concept in the robotics simulation software Webots[31] (Fig. 5) which provides us with a range of different sensors (GPS, Accelerometer, Sonar, etc.) and gives us the possibility to add custom sensors to the robot. The robot will be evaluated for a specific scenario in which it has to mow the lawn in an efficient way while avoiding obstacles and keeping itself safe. In the scenario the robot will be confronted with a range of random dynamic events that can happen, e.g. rain, animals running over the lawn, etc.. Two versions of the robot will be compared: one without our system that takes its decisions based on the aggregated data from sensor fusion techniques and one with our system in addition to these sensor fusion techniques. Both versions of the robot will be tested for a list of parameters:

1. Speed
   The robot should mow the lawn as fast as possible.
2. Efficiency
   The robot should mow the lawn with as little battery usage as possible. This will be split into two parts: The distance traveled by the robot and the internal performance of the systems.
3. Safety
   The robot should keep itself safe and avoid any danger while also not endangering people and animals crossing the lawn while it is trying to fulfill its task.

For both versions of the robot, we will inject faults and noise for the different sensors. In a first step, we will only inject fault for sensors individually and then, progressively, we will inject faults for more and more sensors in order to detect how many simultaneous faults the system can handle. This will also allow us to verify different properties like robustness and resilience for the system.



```java
  private static Display d;
  private static Node robot;
  private static Field translation;
  private static int width ;
  private static int height ;
  private static int uncutGrassGreen=0x228B22;
  private static int lawngreen=0x7CFC00;

  displayJ(){
    d=getDisplay("ground_display");
    width=d.getWidth();
    height=d.getHeight();
    robot=getFromDef("GLADOS");
    translation=robot.getField("translation");
    d.setColor(uncutGrassGreen);
    d.fillRectangle(0,0,width,height);
  }

  public static void main(String[] args) {


    Supervisor s=new displayJ();

    ImageRef to_store=null;
    int timeStep = (int) Math.round(s.getBasicTimeStep());
     while (s.step(timeStep) != -1) {
      translation=robot.getField("translation");
      double[] t= translation.getSFVec3f();
      d.setColor(lawngreen);
      double w=width*(t[0]+0.5);
      double h=height*(t[2]+0.5);
      d.fillOval(
      (int)w,
      (int)h,
      4,
      4);

    };


  }
}
```

Fig. 5. Webots example

It was also considered to compare our system to other similar systems, but a lack of very precise information on the implementation of other systems would make this comparison unreliable.

6 Conclusion

In this paper, we have introduced a generic middleware model for handling conflicting data coming from heterogeneous and possibly defective sensors in a robotic system. The process is used alongside state-of-the-art sensor fusion techniques in order to improve the information the robot receives from its sensors. This allows the robot to make better decisions overall. We also have presented our ongoing experiment to test and evaluate this model.

7 Future work

In the future, we would like to extend this system in different ways. First, as it is important for robots to dynamically adapt to the environment, we want to add self-learning capabilities to our system in order to improve the conflict handling done.

Secondly, we want to provide additional information to the robot in order for it to be able to make even better decisions. This could be done by providing virtual sensors that will provide more abstract information to the robot based on the data received from the sensors, like for example a virtual sensor for the danger level. These virtual sensors would behave in the same way as the normal sensors from the perspective of the robot.

References

1. Akbari, A., Thomas, X., Jafari, R.: 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). pp. 178–182. IEEE (2017)
2. Bader, K., Lussier, B., Schön, W.: A fault tolerant architecture for data fusion: A real application of kalman filters for mobile robot localization. Robotics and Autonomous Systems **88**, 11 – 23 (2017). https://doi.org/https://doi.org/10.1016/j.robot.2016.11.015, http://www.sciencedirect.com/science/article/pii/S0921889015302943
3. Baum, L.E., Eagon, J.A.: An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. Bull. Amer. Math. Soc. **73**(3), 360–363 (05 1967), https://projecteuclid.org:443/euclid.bams/1183528841
4. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state markov chains. Ann. Math. Statist. **37**(6), 1554–1563 (12 1966). https://doi.org/10.1214/aoms/1177699147, https://doi.org/10.1214/aoms/1177699147
5. Baum, L.E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. Ann. Math. Statist. **41**(1), 164–171 (02 1970). https://doi.org/10.1214/aoms/1177697196, https://doi.org/10.1214/aoms/1177697196
6. Bostanci, E., Bostanci, B., Kanwal, N., Clark, A.F.: Sensor fusion of camera, gps and imu using fuzzy adaptive multiple motion models. Soft Computing **22**(8), 2619–2632 (2018)
7. Cannarile, F., Compare, M., Baraldi, P., Di Maio, F., Zio, E.: Homogeneous continuous-time, finite-state hidden semi-markov modeling for enhancing empirical classification system diagnostics of industrial components. Machines **6**(3), 34 (2018)
8. Chen, Z., Tian, L., Lin, C.: Trust model of wireless sensor networks and its application in data fusion. Sensors **17**(4), 703 (2017)
9. Dempster, A.P.: A generalization of bayesian inference. Journal of the Royal Statistical Society. Series B (Methodological) **30**(2), 205–247 (1968), http://www.jstor.org/stable/2984504
10. Dobrican, R.A., Neyens, G., Zampunieris, D.: 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 (2016)
11. Gales, M., Young, S.: The application of hidden markov models in speech recognition. Foundations and trends in signal processing **1**(3), 195–304 (2008)
12. Hide, C., Moore, T., Smith, M.: Adaptive kalman filtering for low-cost ins/gps. The Journal of Navigation **56**(1), 143–152 (2003)
13. Hu, S., Shao, Z., Tan, J.: A real-time cardiac arrhythmia classification system with wearable electrocardiogram. In: Body Sensor Networks (BSN), 2011 International Conference on. pp. 119–124. IEEE (2011)

14. Huang, X.D., Ariki, Y., Jack, M.A.: Hidden Markov models for speech recognition, vol. 2004. Edinburgh university press Edinburgh (1990)
15. Hussain, S., Mokhtar, M., Howe, J.M.: Sensor failure detection, identification, and accommodation using fully connected cascade neural network. IEEE Transactions on Industrial Electronics **62**(3), 1683–1692 (March 2015). https://doi.org/10.1109/TIE.2014.2361600
16. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering **82**(Series D), 35–45 (1960)
17. Kordestani, M., Samadi, M.F., Saif, M., Khorasani, K.: A new fault prognosis of mfs system using integrated extended kalman filter and bayesian method. IEEE Transactions on Industrial Informatics pp. 1–1 (2018). https://doi.org/10.1109/TII.2018.2815036
18. Kwon, S., Ahn, H.: Sensor failure detection, identification and accommodation using neural network and fuzzy voter. In: 2017 17th International Conference on Control, Automation and Systems (ICCAS). pp. 139–144 (Oct 2017). https://doi.org/10.23919/ICCAS.2017.8204431
19. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (1943)
20. Naidu, S.R., Zafiriou, E., McAvoy, T.J.: Use of neural networks for sensor failure detection in a control system. IEEE Control Systems Magazine **10**(3), 49–55 (April 1990). https://doi.org/10.1109/37.55124
21. Nemra, A., Aouf, N.: Robust ins/gps sensor fusion for uav localization using sdre nonlinear filtering. IEEE Sensors Journal **10**(4), 789–798 (2010)
22. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE **77**(2), 257–286 (1989)
23. Salfner, F., Malek, M.: Using hidden semi-markov models for effective online failure prediction. In: Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on. pp. 161–174. IEEE (2007)
24. Salovaara, A., Oulasvirta, A.: Six modes of proactive resource management: a user-centric typology for proactive behaviors. In: Proceedings of the third Nordic conference on Human-computer interaction. pp. 57–60. ACM (2004)
25. Sasiadek, J., Wang, Q.: Sensor fusion based on fuzzy kalman filtering for autonomous robot vehicle. In: Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on. vol. 4, pp. 2970–2975. IEEE (1999)
26. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press (1976)
27. Shirnin, D., Reis, S., Zampunieris, D.: 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. pp. 253–258. SciTePress–Science and Technology Publications (2012)
28. Tennenhouse, D.: Proactive computing. Communications of the ACM **43**(5), 43–50 (2000)
29. Tom, K., Han, A.: Context-based sensor selection (Oct 14 2014), uS Patent 8,862,715
30. Turan, M., Almalioglu, Y., Gilbert, H., Araujo, H., Cemgil, T., Sitti, M.: 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). pp. 1–8. IEEE (2018)
31. Webots: http://www.cyberbotics.com, http://www.cyberbotics.com, commercial Mobile Robot Simulation Software

32. Yazdkhasti, S., Sasiadek, J.Z.: Multi sensor fusion based on adaptive kalman filtering. In: Advances in Aerospace Guidance, Navigation and Control. pp. 317–333. Springer International Publishing, Cham (2018)
33. Zampunieris, D.: Implementation of a proactive learning management system. In: Proceedings of" E-Learn-World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education". pp. 3145–3151 (2006)
34. Zampunieris, D.: 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 (2008)