Methods for Failure Detection for Mobile Manipulation

Anastassia Küstenmacher

Publisher: Dean Prof. Dr. Wolfgang Heiden

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

March 2012

Technical Report 04-2012

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

Abstract

The work presented in this paper focuses on the comparison of well-known and new fault-diagnosis algorithms in the robot domain. The main challenge for fault diagnosis is to allow the robot to effectively cope not only with internal hardware and software faults but with external disturbances and errors from dynamic and complex environments as well. Based on a study of literature covering fault-diagnosis algorithms, I selected four of these methods[1] based on both linear and non-linear models, analysed and implemented them in a mathematical robot-model, representing a four-wheels-OMNI robot. In experiments I tested the ability of the algorithms to detect and identify abnormal behaviour and to optimize the model parameters for the given training data. The final goal was to point out the strengths of each algorithm and to figure out which method would best suit the demands of fault diagnosis for a particular robot.

[1] The algorithms chosen were: Parity Space (PS), Hidden Markov Model (HMM), Particle Filter (PF) and

Acknowledgments

I would like to thank my supervisor Paul G. Plöger who took a lot of time to discuss the topic with me. He gave me new ideas and helped me understand the theory behind the algorithms.
Kai Pervölz supported me with his teachings about mobile manipulators. The discussions with him about fault diagnosis have been interesting and helpful.
Thanks to my family for being patient with a robot-crazy mom and wife.

Contents

1. Introduction

Fault diagnosis plays an important part in the development of complex systems.
The ability to monitor and diagnose complex physical systems is critical for constructing efficient autonomous systems that can perform their tasks robustly in dynamic environments over a long period of time.
Fault Diagnosis allows complex systems to efficiently cope not only with internal faults but with external faults as well. Since a robot must closely interact with its environment, there is high probability for external faults for the environment may by unknown or changing. Consider a robot system consisting of a number of components, each of which is responsible for detection internal faults. If a robot performs the task of grabbing a pencil from a table, the respective manipulator component can detect if it grabs the object or if it fails to do so. Would it grab another object from the same table, the manipulator would not recognise an error. This action however could be detected by an external device like a camera. The new data would then be translated to the central controller for analysis and eventual recovery. Fault monitoring provides a fault report which will enable the system to adapt to the situation and avoid errors in the future.
In many cases the physical model of the system is known. This means that we are able to calculate the correct outputs, compare them to the ones we received from the sensors, and draw conclusions about faults. The main problem is the handling of the physical model of the robot, as it is hard to exactly determine. Since the world is dynamic, the robot's environment can always change, therefore the robot model has to be customized accordingly. If the physical model is unknown, it has to be estimated from a set of training data.
To a given or an estimated model we can apply model-based *fault detection and isolation* (FDI) algorithms. The general principle is to compare the expected behaviour of the system given by the model with actual behaviour, known through on-line observations.

1. 1 Problem Statement

The objective of the work is to recognize errors and abnormal behaviour in complex systems with large number of heterogeneous modules and devices which interact with dynamic environments.

The interest in fault diagnosis has been increasing in the last three decades. Building a flexible system is of importance especially in automobile, aircraft and chemical industries. The increasing complexity of robot systems attracted the attention of universities and scientists for Fault diagnosis. Research in fault-tolerant control has created a large variety of algorithms and ways of implementing them. Unfortunately there is no universal technique that could be easily applied to any model. To develop a fault tolerant system we first need to survey existing fault diagnosis methods in dynamic systems. We then chose the appropriate algorithms, test them and narrow them down to the best possible solution for implementing in a mobile manipulator. The implementation itself is not a part of this work.

1. 2 Motivation and Challenges

The increasing complexity of robot systems influences the probability of component and system faults. Fault tolerant behaviour in robots is desirable for a variety of rather obvious reasons. Timely fault diagnosis increases the ability to complete tasks satisfactory and improves performance and safety – the robot becomes more efficient economically.

The ability of a system to recognise errors and draw conclusions about future actions according to the situation enables it to avoid failures such as mission abortion, material damage and human accidents. After estimating the severity of a fault and determining its location, the system can be easily repaired. Fault tolerant robots are more flexible to new circumstance and environment.

Fault tolerant behaviour not only identifies unsatisfactory performance and defines the location of an error, but it also enables the system to keep on performing its task. The majority of the approaches in the fault detection and isolation literature deal with internal faults such as defects in hardware or software. State-of-the-art researches for fault diagnosis focus on dealing with external influences. This might be a manipulator grapping the wrong object, but one with a similar shape or suddenly switching off the light in a bright room, etc.

Most existing literature on Fault monitoring is concerned with detecting abnormal behaviour in mobile robots or fix-based manipulators separately. Only few written works investigate fault diagnosis in the field of mobile manipulation. I assume this lack of scientific debate is due to the infancy of the topic and the complexity of the matter – system redundancy, cooperative coordination between vehicle and arm platform and the control structure design.

To select a fault diagnosis algorithm adequate for a mobile manipulator it has to be taken into account that the environment of the mobile manipulator is dynamic, that sensor measurements can be disturbed by noise, that actuators are imprecise and that the system state can depend on various operating conditions. Furthermore we have to accept that there is only limited computational power, that some information of interest is unobservable and that some faults demand a sequence of observations in order to be detected. Real-time detection of faults is essential for the robot.

1. 3 Thesis Statement

In this work I intend to discuss and compare four fault diagnosis approaches, for a better understanding of their theoretical basics and their practical application. The final goal is to point out the strengths of each algorithm and to figure out which method best suits the demands of fault diagnosis for a robot.

After comparing the algorithms *Parity Space* to *Particle Filter* and *Hidden Markov Model* (*HMM*) to *Observable Operator Model* (*OOM*), I came to the following conclusions:

- For a linear state-space model with additive faults analytical results can be derived by the *Parity space* approach [1]. If a model is unknown but known to be linear, the *principle component analysis* can be used [2].
- In systems with temporal dependencies the *Hidden Markov Model* (*HMM*) can be applied to describe a model and to solve the fault diagnosis problem. The *HMM* uses the *Expectation-Maximization* (EM) algorithm for training, but it leads to local maxima only, and in the most points of interest, the optimization surface is very complex and has many local maxima. [3]

- Some problems can be solved using state estimation, where the fault is an unknown state among other states in the process. In this case, fault detection and isolation may be tracked with *Particle Filters*. Using this algorithm the developer can influence the accuracy of results and computational resources by adjusting the number of particles. [4]
- The *Observable Operator Model* (*OOM*) is an alternative new approach to *HMM*. Its theory is expressed in terms of linear algebra. *OOM* is applicable to a broader range of processes. This new algorithm does not have a local maxima problem as is the case with the EM approach. Most datasets obtained via the *OOM* learning algorithm are more accurate than HMM models. OOM is stable in the detection phase, but suffers from the *negativity probability problem* [5].

1. 4 Related Works

There are various classified approaches of the existing fault diagnosis methods. The usual classification is shown in the papers [6], [7], [8]. In these papers the authors broadly divide fault diagnosis methods into three general categories: quantitative model-based methods, qualitative model-based methods, and history-based methods. The authors define the methods for the industrial domain; unfortunately an application for robots is not of concern here.

In his thesis work [9], P. Sundvall chiefly considers the model-based diagnosis. He focuses on the fault handling methods in general and relates how they have been implemented in different robots. In the model-based approach the dependency between inputs and outputs is mathematically defined, which means that there has to be a process model.

In contrast to the model-based methods, in history-based methods [12] only a large amount of input and output data is available. The primary benefits of model-based diagnosis over other techniques are that it does not need pre-computation (it is entirely online) and that it uses less computational power. It can provide very accurate results (the exactness of the underlying model improves the accuracy of the algorithm).

R Isermann in his book [10] has shown the theoretical and experimental research of new ways to detect and diagnose faults. This book, which aptly introduces the matter of fault diagnosis, is based on the results of the author’s own research projects during the last 25 years and on publications by many other research groups.

M. Staroswiecki in his tutorial paper [11] about model-based fault diagnosis techniques introduces the mathematics of constructing sufficient models for various kinds of faults.

I selected four methods based on both linear and non-linear models. F. Gustafsson [1] [2] and J. Gertler [13] analyse the *parity space* approach for fault detection and identification. They present the *parity space* algorithm based on the well known *Chow-Willsky scheme* [14]. It can take advantages of the linear state space system.

The other approach has been described by L. Rabiner [3]. His work focuses on the theoretical description of *HMM* and its implementation in speech recognition applications.

The works on model-based fault diagnosis include *GDE/Sherlock* [15], [16], *Livingstone 1, 2* [17], [18] and *Titan mode estimation* which maintain reliability using a variant of the *Viterbi algorithm* [3]. Unfortunately the authors only take into account discrete states and known models (the model estimation was not provided).

For many robot applications a diagnosis with a discrete model is inadequate. To overcome this problem the model needs to be a hybrid system. Such a system consists of a set of discrete states which correspond to functional modes, fault conditions and continuous states which represent the observable state of the robot (e.g. wheel speed, motor current etc.).

Particle Filters are the solution for such kind of problems. They belong to a family of sequential *Monte Carlo methods* for approximate inference in parity observable *Markov Chains* [4]. They represent the probability of the system states by a set of particles.[20]

The classical *Particle Filter* approach has several disadvantages in the fault diagnosis domain, such as a high improbability value of faults and an increasing amount of samples that require a lot of computing power. There are various approaches to addressing this problem in the literature. The goal of the *Risk Sensitive Particle Filter* algorithm [21], [22] by S. Thrun is to increase the amount of particles in “risky” or important states (e.g. the manipulator breaking its joint).

V. Verma presents a further algorithm with the *Variable resolution Particle Filter* [21], [23]. It is based on the observation that some faults have similar symptoms so that they can be grouped together. E. Benanzera [24] combines *Livingstone* and *Particle Filter*. Plagermann [25] applies the *Gaussian process classification* for learning effective proposal distributions of *Particle Filter*. As a result, the efficiency and robustness of the state estimation is improved. The *Rao-Blackwellized Particle Filter* [26] is motivated by problems of low prior fault probabilities and restricted computational resources.

H. Jaeger [5] introduces a new alternative approach in the robot field – the *Observable Operator Model* (*OOM*). This theory seems to be similar to the *Hidden Markov Model* (*HMM*) as both can be expressed in matrix formalisms. The matrixes and state vectors of *OOM*s may contain negative elements, whereas the *HMM* matrixes include only non-negative probability values. Jaeger gives a theoretical comparison of the *OOM* to the *HMM* – but no published material on the comparison of the two theories with results based on an experimental fundament exists yet. In my work I will attempt to close this gap.

1. 5 Reader's Guide

In this paper I describe the activities carried out while working on the Thesis. The second chapter presents the theoretical fundamentals of fault diagnosis in general. The diagnosis methods of the industrial domain are grouped and exemplified. Mobile manipulator theory and fault handling in the robot domain are described in the third chapter. In chapter four I acquaint the reader with the four selected fault diagnosis algorithms. Besides introducing the theoretical background, I evaluate the efficiency of these methods also by means of practical examples. The results of implementing the methods in the four-wheel OMNI robot are illustrated at the end of each section. Based on these solutions the advantages and disadvantage are discussed. Eventually, I provide outcomes and conclusions of the research work in chapter five.

2. Theoretical Background of Fault Diagnosis (in general)

Consider a mobile manipulator in a room. To perform a task, it needs to move to a table and take an object. The mobile manipulator commands a fault diagnosis system for controlling its behaviour. It sets a defined speed to its wheel and measures its location every two seconds. From this information, the diagnosis system of the robot calculates the distance for each time unit and defines the expected location. The fault diagnosis system can now generate a diagnosis statement that will point out a fault if the expected location does not correspond to the measured location. When a fault is detected, the diagnosis system tries to identify its nature. For example, if a wheel gets stuck, the system records all the information about the occurrence and passes it on to the "central controller", which should come up with a solution for the problem.
From a general perspective [27] fault diagnosis can be explained as follows: The task is to generate a diagnosis that states whether a fault arises or not. If a fault is determined, its location has to be identified.
Hence there are three main challenges of fault diagnosis: The generation of the diagnosis statement, the choice of the relevant parameters and the representation of expected or normal behaviour. The observations or measurements are chiefly output data obtained from the sensors, but can also be observations made by humans.

2. 1 Basic Definitions

The terminology used in this paper field is based on definitions of the IFAC Technical Committee SAFEPROCESS.
"

- ***Fault***

  Unpermitted deviation of at least one characteristic property or variable of the system from acceptable/usual/standard behaviour.
- ***Fault Detection***

  Determination of faults present in a system and time of detection.
- ***Fault Isolation***

Determination of kind, location, and time of detection of a fault. Follows fault detection.

- ***Fault Identification***

  Determination of the size and time-variant behaviour of a fault. Follows fault isolation.
- ***Fault Diagnosis***

  Determination of kind, size, location, and time of detection of a fault. Follows fault detection. Includes fault detection, isolation and identification."

For this work we will use an abstract version of these definitions, as fault identification is not of concern for our thesis:

- Fault detection defines whether a fault has occurred.
- Fault isolation sets where and when a fault has occurred.
- Fault diagnosis contains both fault detection and fault isolation.

Most fault diagnosis methods are based on the concept of redundancy (extra resources) in the system, so that a parameter can be calculated in more than one way. If, for example, several sensors are available to measure the same quantity, such type of redundancy is called hardware redundancy [10]. Hardware redundancy is a classical approach of fault diagnosis methods. Obvious disadvantages of using the hardware redundancy concept are higher costs, increased weight and complexity. The trend of current fault diagnosis techniques is based on the analytical redundancy concept.

> "*There exist analytical redundancy if there exists two or more (but not necessarily identical) ways to determinate a variable, where one way uses a mathematical process model in analytical form.*" [42]

If two different sensors measure the same parameter according to the following relation: $y_1 = \sqrt{x} \wedge y_2 = x$, then the accuracy of parameter x can be validated [42].

The third kind of redundancy concept used is the concept of hybrid redundancy. It includes hardware and analytical redundancies.

2. 2 Classification of Fault Diagnosis Methods

There are various approaches to classify the existing fault diagnosis methods.
One popular classification of industrial fault detection and isolation (FDI) methods is shown in Figure 1.

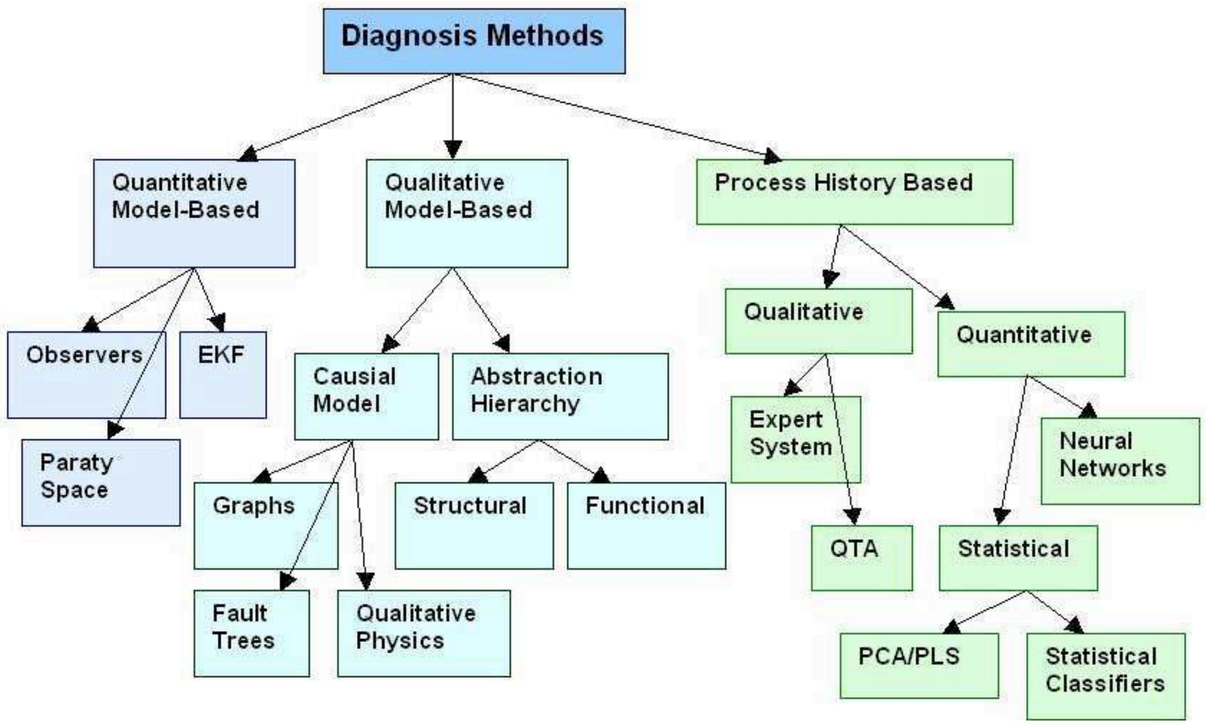

Figure 1 Classification of diagnostic algorithms [6]

In "*A review of process fault detection and diagnosis part1: Quantitative model-based methods*" [6] the authors broadly divide the most frequently used approaches to fault diagnosis in engineering into three general categories: quantitative model-based methods, qualitative model-based methods, and process history-based methods. In the model-based approach the relation between inputs and outputs is mathematically defined, which means the process model is assumed. In contrast to model-based methods, in history-based methods only a large amount of input and output data is available. As depicted above, the model-based approaches can be classified as quantitative [7] or qualitative [8]. In quantitative methods the underlying model is expressed in terms of a mathematical relationship between inputs and outputs of the system, e. g. differential equations, transfer functions, state-space models, etc. In contrast, qualitative methods are based on artificial-intelligent techniques, such as fuzzy logic and neural networks, using qualitative reasoning and modelling such as causalities and IF-THEN rules. They predict the behaviour of the system in normal and faulty conditions and then compare predicted and actual behaviour to diagnose the faults [43].

2. 3 Model-Based Scheme

There is an increasing interest in theory and applications of model-based fault diagnosis algorithms. The simple diagram in Figure 2 displays an example on how these models are usually integrated into a control system as its diagnosis constituent [45], [46].

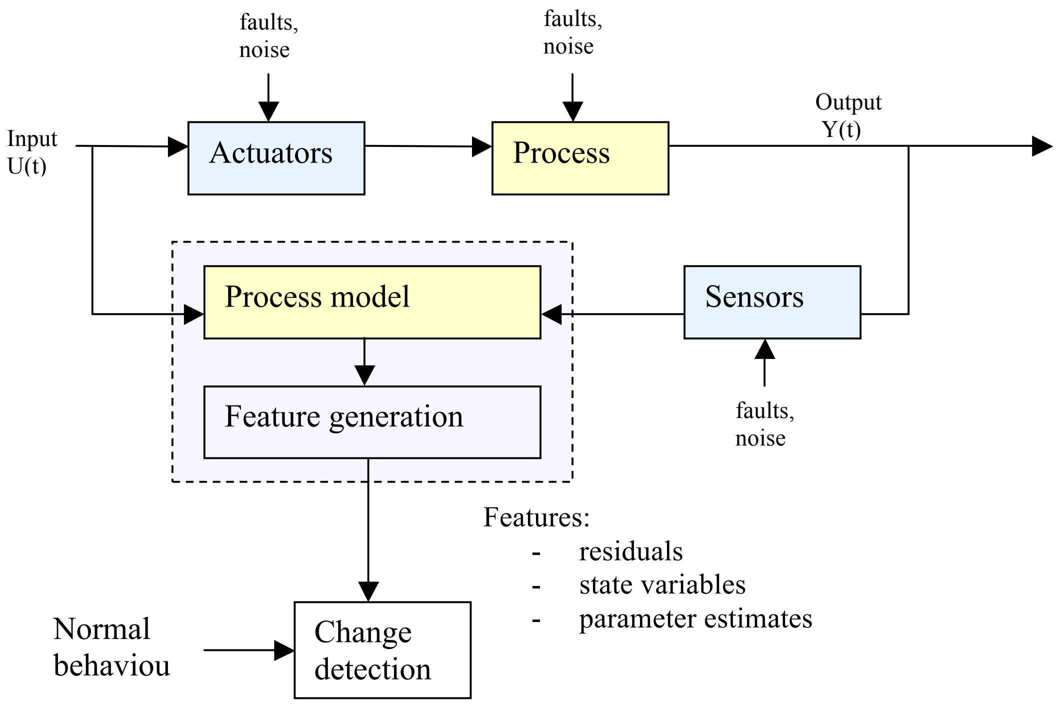

Figure 2 Scheme for the model-based diagnosis [45], [46].

Usually the model-based fault diagnosis scheme consists of two steps: detection and isolation. During the first step the actual behaviour is generated and compared to the one of the process model. Together with the process model the detection algorithm calculates corresponding features. The *Parity Space* method for example will generate residuals (the deviation of the actual behaviour from the nominal one), *Particle Filter* will generate state variables (for the details see chapter 3 and 4), etc. During the second step, the *isolation process*, the faults are evaluated.

2. 4 Fault Modelling

The knowledge about the modelling of faults is important for the right choice of suitable fault diagnoses methods. “Fault” was defined as a deviation of any property of a variable [10]. Faults can be classified as follows:

- Additive Faults. These faults are additive to input or output of the process. The process model is fixed even when faults occur.
- Multiplicative Faults. These faults appear as changes in the process model.

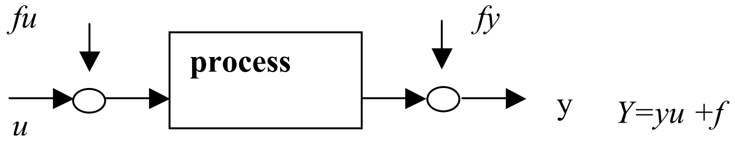

Figure 3 Additive faults [10]

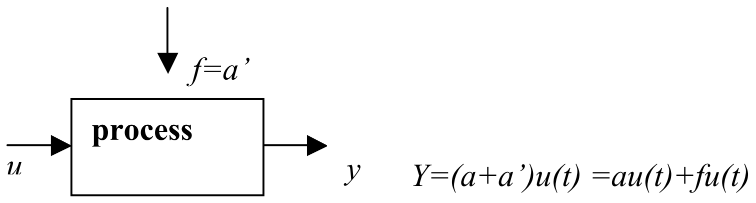

Figure 4 Multiplicative faults [10]

R. Isermann groups possible failure situations by their nature [10], [46], [58]:

- Abrupt Faults (sudden faults) are unexpected faults, which appear as a quick change from normal to abnormal behaviour, for example the sudden breakage of a wheel motor.

- Incipient Faults (slowly developing) are represented by drift-type changes. A typical example is the degradation of a tool. The faults are typically small and not easy to detect.

- Intermittent Faults (periodic faults) repeatedly occur and disappear with different deviations between normal and abnormal behaviour value.

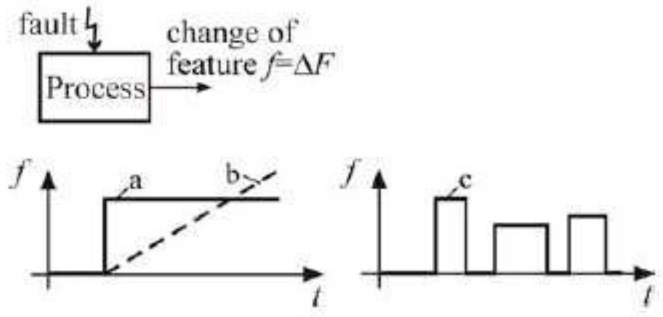

Figure 5 Time-dependency of faults: (a) abrupt; (b) incipient; (c) intermittent [10]

2. 5 Process Modelling

Within the bounds of this work we intend to describe and compare the four model-based techniques[2]. Since comprehensive and accurate mathematical models of dynamic processes are necessary for a model-based diagnosis, in this chapter we will introduce some examples of mathematical models. They are obtained by either theoretical modelling or experimentally [10].

In theoretical modelling the model is set up on the basis of mathematically formulated physical laws.

During experimental modelling we obtain the mathematical model of a process from measurements. Input and output signals are measured and evaluated by identification methods in such a way that the relationship between input and output signal are expressed in a mathematical model [46].

We can distinguish the following types of mathematical models: algebraic equations, difference equations, finite state automata and differential equations. At any time moment the state of the system (*state x*) is described by a set of variables. For example, coordinates and Euler angles describe a state of a mechanical system. The input commands which control the state of the system are *control inputs u*, the sensor output is an *observation y.*

Model Examples:

A large class of engineering systems can be modelled by differential equations of state-space representations.

[2] Parity Space, Hidden Markov Model, Particle Filter, Observable Operator Model

Consider the system is working around nominal operating conditions and its behaviour can be represented by a linear state-space model.

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)x(t) + D(t)u(t) \end{aligned}$$

x(t) – state vector, *y(t)* – output vector and *u(t)* – control vector

A(t), B(t), C(t) and *D(t)* are matrixes of appropriate dimensions. The matrixes are time-variant (the elements of them depend on time), in the time-invariant case the elements of matrixes are not changed over time.

The model for a discrete-time system is governed by the difference equations. Otherwise for continuous-time system the model is defined by deferential questions, Depending on the system type, the state-space model representations have the following forms (see for details [71]):

System type	**State-space model**
Continuous time-invariant	$\dot{x}(t) = Ax(t) + Bu(t)$ $y(t) = Cx(t) + Du(t)$
Continuous time-variant	$\dot{x}(t) = A(t)x(t) + B(t)u(t)$ $y(t) = C(t)x(t) + D(t)u(t)$
Discrete time-invariant	$x(t+1) = Ax(t) + Bu(t)$ $y(t) = Cx(t) + Du(t)$
Discrete time variant	$x(t+1) = A(t)x(t) + B(t)u(t)$ $y(t) = C(t)x(t) + D(t)u(t)$

Faulty operations in the state-space model are:

Multiplicative faults (changes in the system parameters)

The system parameters for the state-space representation are model matrixes *A, B, C* and *D* their changes can be presented as

$(A+ \Delta A(t)), (B+\Delta B(t)), (C+\Delta C(t))$

Additive faults (Faults are input signals to the model)

Additive faults in the sensors and actuators can be modelled by two additive signals:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + F_x\varphi(t)\\ y(t) &= Cx(t) + F_y\varphi(t)\end{aligned}$$

When $\varphi(t) \in R^f$ is some unknown fault vector, F_x, F_y are matrixes of suitable dimensions, whose entries are real numbers, which trace the fault influence, respectively on state and measurement equations.
If we want the model to better depict its real world counterpart we need to add the uncertainty about unknown inputs or process disturbances and measurement noise.
The general form of linear state-space representations is

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + Ev(t) + F_x\varphi(t)\\ y(t) &= Cx(t) + F_y\varphi(t) + \varepsilon(t)\end{aligned}$$

v(t) is the vector of unknown inputs or disturbances acting on the process, and $\varepsilon(t)$is the measurement noise which corrupts the sensors.

2. 6 Applications in the Industrial World

Rama K. Yedavalli [41] presents the tutorial overview of the literature in the area of fault diagnosis of dynamical systems
The model-based FDI methods have been used for various applications, such as helicopter rotors [28], aircrafts [29], automotive vehicles [31], space shuttle main engines [39], actuators/sensors [30], industrial furnaces [32], electro-hydraulic cylinders [33], diesel engines [34], induction motors [35], [36], satellite systems [37], UAVs (Uninhabited Aerial Vehicles) [38] and rocket engines [40].

3. Fault Diagnosis in the Robot Domain

It is obvious that a robot includes a large amount of components controlled by various software programs and hardware devices. As an example we take a look at the mobile manipulator of the German Service Robotics Initiative project (DESIRE) [44]. The robot consists of components such as *head control*, *perception*, *drive unit*, *manipulation* and others. Each of the components is responsible for a concrete functionality. The *perception* component includes three various cameras: two RGB cameras which record from a position at the left and the right side of the robot's heads and one 3D-camera at the front of the torso. These devices are controlled by software of the *perception* component which gives commands to cameras and processed delivered data. The components interact with each other, e.g. the *perception* component provides data to the *drive unit* from which it can construct, for example, a map about the environment. The other way for component communication is via *Eigenmodel. Eigenmodel* is a component of the robot for managing the collaboration of components. It collects information from various components, analyses them and predicts the future actions of the robot. If one component needs to provide the data for another, it will first inform *Eigenmodel* about its action. Besides the coordination work the *Eigenmodel* is responsible for recognising abnormal data. Fault diagnosis for such robot systems is a complex task, because the number of possible faults is huge. It includes not only faults which could arise in hardware and software applications but also faults during component interaction and even worse faults caused by the environment..

3. 1 Fault Classification

Based on this example we can classify the fault field in the robot domain (Figure 6).

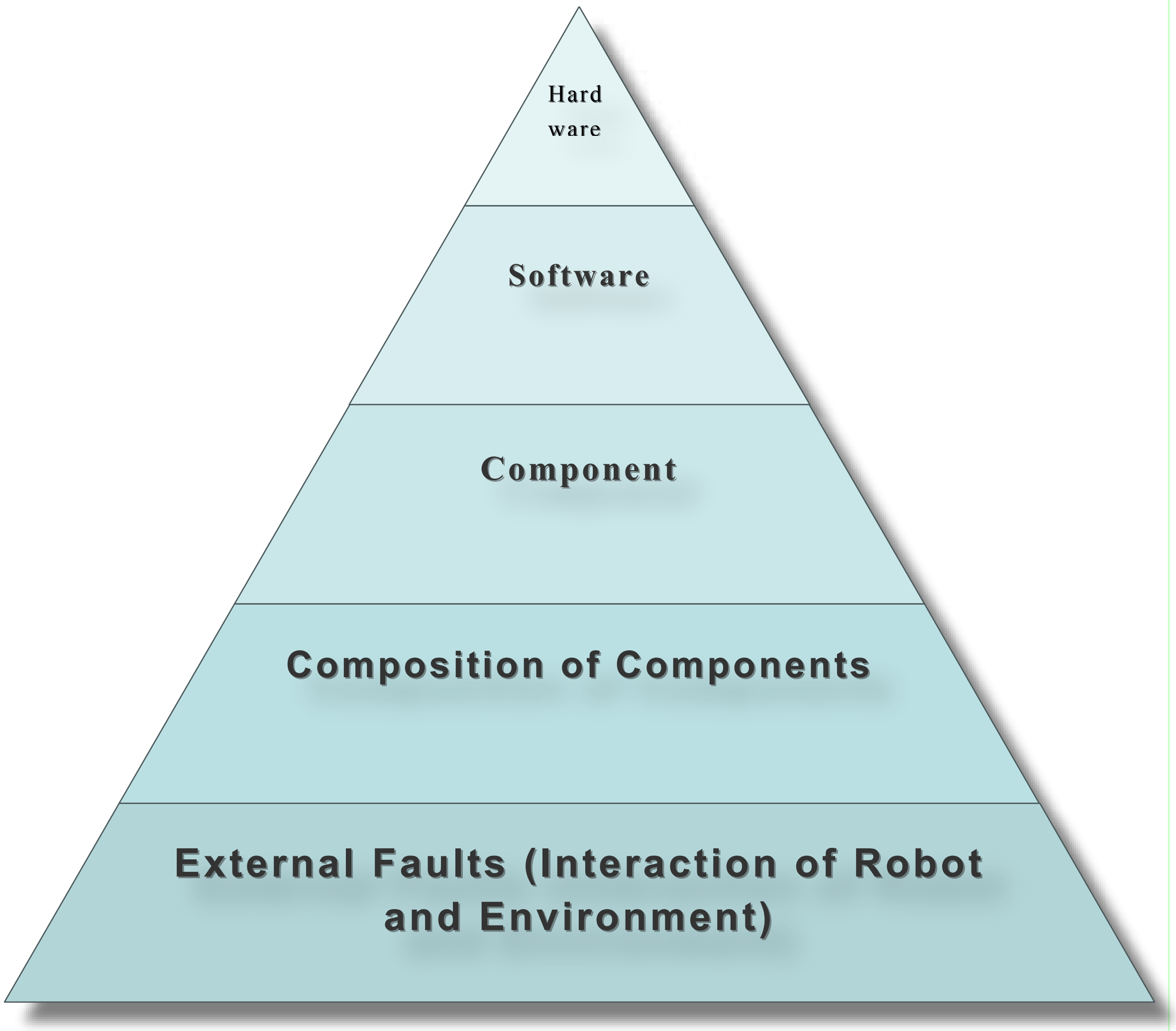
Hard
ware
Software
Component
Composition of Components
External Faults (Interaction of Robot and Environment)

Figure 6 Fault classification

The triangle depicted in Figure 6 consists of five levels of fault's classes. On the top of the triangle the "easiest" fault class is located. The complexity of faults increases from top to bottom, the lowest level corresponding to the most complex fault class. "Fault complexity" stands for the difficulty to detect and identify a certain group of faults.

Hardware faults: are mechanical breakdowns of inner devices e.g. the camera switches off when it should be recording, motors suddenly breaking down, etc.

Software faults: result from program/data corruption. This could be a fault inside a function like the division by zero, etc. By sending error messages, the procedure may inform the diagnosis system about existing faults.

Component faults: include faults which take place during the collaboration of a software program and a hardware device. This could be an error following from a transmission of a wrong format to a device. The diagnosis system may monitor component faults via following a sequence of steps: preconditions - > transitions –> post-conditions (See Figure 7)

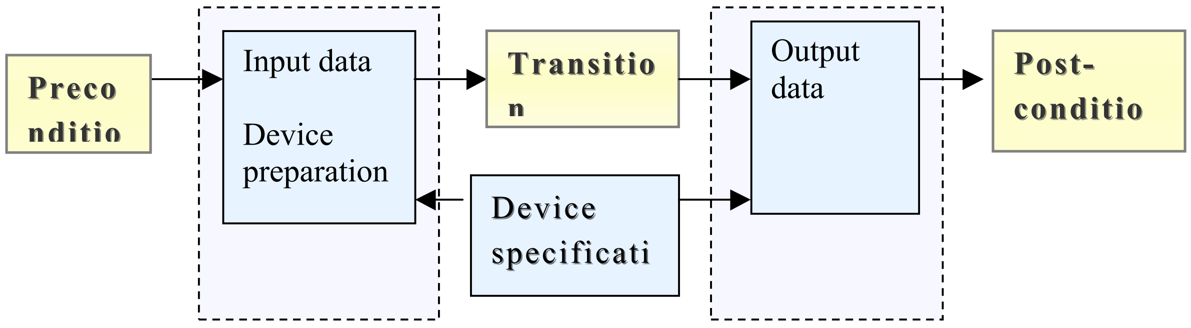

Figure 7 Diagnosis scheme for component's faults

During precondition, the diagnosis system checks serviceable conditions of the device. The device must be prepared for the input commands. The device must be able to understand the format. The output of devices will be controlled during the post-condition step. The diagnosis system prepares a protocol about the fault for recovery.

Composition of components faults: are faults resulting from the interaction of several components to achieve a task. In this case the result of interaction is faulty, although the single components may work fault-free separately. If, for example, a mobile manipulator should grasp a desired object, it first has to move until the object will lie in the reachable work space. It then pulls its mobile part up and performs the grapping action. Would the manipulator now start the grapping action while still on the move to the reachable work space, we would call this kind of fault a "composition of components fault" (see Figure 8).

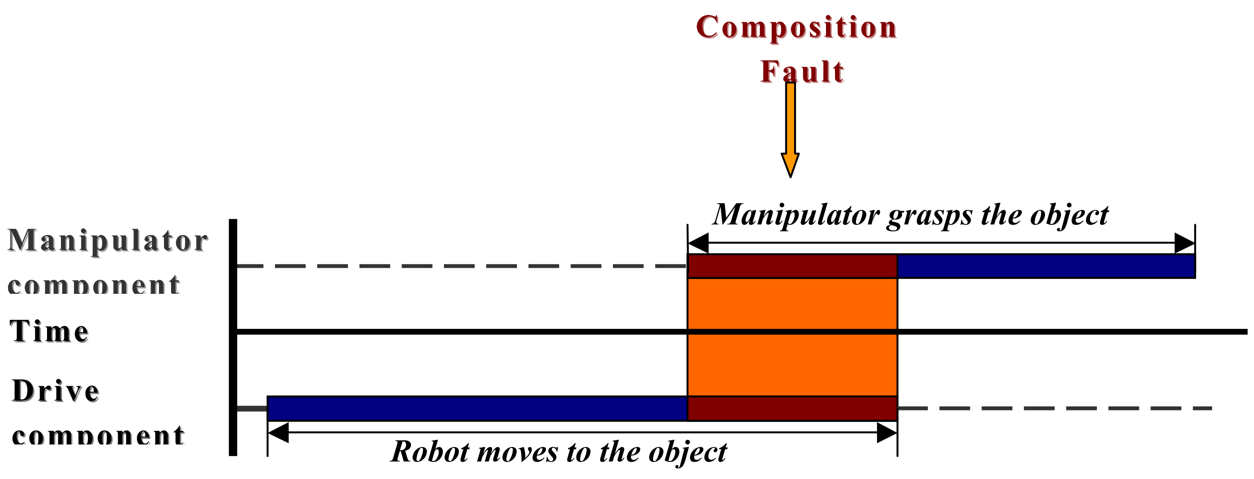

Figure 8 Example of "composition of components fault" in Mobile Manipulator

<u>External faults</u>: take place during the interaction of a robot and its environment. External faults could result from changing environments, interaction with human beings or other autonomous systems. All these criteria involve deep analysis of the fault situation, which might be beyond the scope of the fault diagnosis inside the system.

Imagine a robot running in a room filled with various objects. The robot needs to move carefully to avoid collision with these objects. If the light in the room is suddenly switched off (external fault), then the robot must understand the situation: It's not his sensors that cease to function but a change in the environmental conditions.

The monitoring of external faults is especially important for mobile manipulators, because they closely interact with a dynamic environment.

3. 2 Robot Model (Design Example)

To fully understand the fault diagnosis techniques, they have to be implemented in practical applications. Among a wide range of diagnosis techniques we selected four distinct algorithms[3]. We are going to illustrate their functional efficiency on a particular practical example – a four-wheel OMNI robot [72]. OMNI directional robots have become popular mobile robots for the use in indoor environments, because they may drive in any directions without having to rotate first. An OMNI drive mobile robot frequently serves as a moving platform for mobile manipulators. The geometry of the

[3] Parity Space, Hidden Markov Model, Particle Filter, Observable Operator Model

OMNI driver robot is depicted in Figure 13. It has two intersecting axes, the horizontal axis corresponding to the x-direction and the vertical axis corresponding to the *y*-direction. φ is the angle between the wheel and the x-axis, R is the radius of the robot platform and F_1, F_2, F_3 F_4 are forces from the motors. Assumed the robot does not slip on the flow, it will translate along the x-axis when F_1=-F_2, F_1=F_4 and F_2=F_3 and along the y-axis when F_1=F_2, F_3=F_4 and F_1=-F_3. The wheels work against each other when F_1=-F_2, F_1=-F_4 and F_2=-F_3.

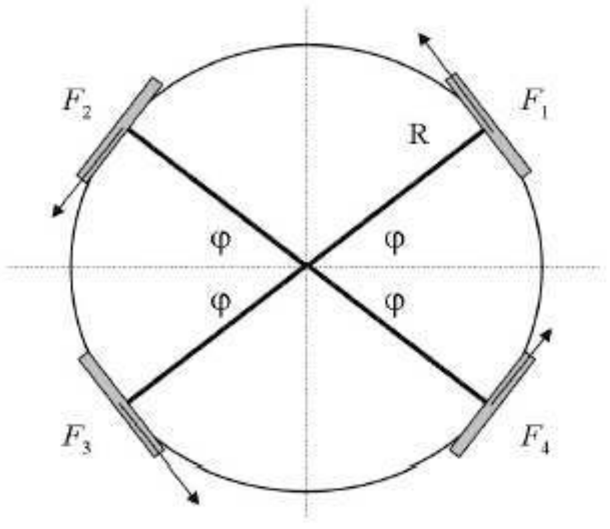

Figure 9 Arrangement of the wheels and distribution of forces [72]

The forward kinematics for obtaining the robot velocities from the given wheel velocities are given by the following expression:

$$(v_x, v_y, w)^T = \begin{pmatrix} \sin\phi & -\sin\phi & -\sin\phi & \sin\phi \\ -\cos\phi & -\cos\phi & \cos\phi & \cos\phi \\ \frac{1}{4R} & \frac{1}{4R} & \frac{1}{4R} & \frac{1}{4R} \end{pmatrix} (v_{1,} v_2, v_3, v_4)^T \qquad (3.2.1)$$

Wheel velocities are presented through the (v_1, v_2, v_3, v_4) vector and the robot velocities are described by the Euclidean velocity (v_x, v_y) of the robot on the ground and its angular velocity (ω). ϕ and R are fixed parameter which are defined for the robot: ϕ=0.588 radian and *R*=0.25 m.

The localisation problem is chosen for a test application of the selected fault diagnosis techniques. The localisation problem is formulated as an estimation of the robot position from the given sensor data. Although the localisation seems a simple problem compared to others, it seems to be a good starting point. After testing various fault diagnosis techniques, conclusions not only about the algorithms' performance but also about the possibility to apply them in more complex systems, e.g. mobile manipulators, can be drawn.

Since the mobile platform is a mandatory part of a mobile manipulator, the task of localisation for a mobile robot can be extended to the localisation problem of a mobile manipulator.

In general, the localisation, calculating the changing position over time, is a dynamic process. At least two models are required to conduct the process. One model is needed to describe the evolution of the state with time (system model), and the other to relate observable measurements of the state (measurement model).

There are two known forms available to build these models: state-space form and probabilistic form.

The state-space model for fault diagnosis is presented in section 2. 5, hence we will focus our attention on probabilistic models.

Many real-world applications are able to analyze their own data by estimating unknown quantities from some given observations. The prior knowledge of the modelled phenomenon is available. This knowledge allows us to formulate Bayesian (probabilistic) models. A Bayesian model includes prior distributions of unknown quantities and probability functions, which expresses the relationship between the unknown quantities and observable events [4]. The main drawback of probabilistic modelling is the long-time model estimation.

To build an appropriate probability and state-space model for a four-wheel OMNI robot, we define a group of states [73]:

- normal behaviour (no fault)
- broken motor (the output of the motor is fixed to the value "zero", regardless of input)
- stuck motor (the output of the motor is a fixed constant, regardless of input)
- gradual degradation of performance (the output of the motor grows with a negative exponential function)

3. 3 Mobile Manipulators

A mobile manipulator is a robot arm build on top of a moving base. It is becoming more and more popular in our days since it extends the performance ability of mobile robots and manipulators. Mobile manipulators appear in a broad spectre of robot applications, ranging from underwater and space robots to service robots. They obtain wide application in fields where robots typically interact with the environment.

The key problem of model-design of mobile manipulators is the coordinated work between vehicle platform and arm. The investigations on mobile robots were successful in the fields of localisation, navigation and learning environment. Most work on manipulators focuses on the properties of specific objects to be manipulated, rather than on moving in or understanding the global environment.

There are three important reasons for the existence of mobile manipulators, one of them being its superior dexterous manipulation.

Secondly a mobile manipulator is able to execute more complicated tasks, for example operating a door inwards, towards itself. By performing this action the mobile base has to move to avoid getting hit by the door while the arm has to grasp the door-handle and move simultaneously with the base.

Thirdly it extends the reachable workspace of the manipulator. The robot can for example grasp an object lying on a table and put it to a shelf which is situated far from the table.

One of the characteristics of mobile manipulators is the high degree of kinematics redundancy (more than six degrees of freedom) created by the addition of the mobile platform to the manipulator. The redundancy gives the mobile manipulator several advantages. Joint torques can be optimized, singular configurations of the manipulator can be avoided and decoupled force/position control along the same task direction can be achieved [75]. The higher degree of redundancy allows various fault recovery possibilities that improve the performance of the robot. If, for example, one of the manipulator joints is broken, the robot may be able to continue the task by recalculating the inverse kinematics of the manipulator for using all joints except the broken one.

Note that not every mobile manipulator is redundant, for example in the paper [76], the authors introduce mobile robots equipped with low degree-of-freedom “palm manipulators” (see Figure 13).

As the authors of A. Petrovskaya and A. Ng [77] note, the main advantage of mobile manipulators is the combination of both navigation and manipulation. Most of the works on mobile manipulation nonetheless treat the problem as two tasks to be solved separately: The mobile platform navigates to an appropriate point and then the manipulator separately performs actions with an object. A. Petrovskaya and A. Ng [77] created an algorithm based on the probabilistic approach that models the position of the robot within the environment and simultaneously manipulates the object.

Using probabilistic models for fault diagnosis techniques brings the advantage of a broad spectrum of already existing inference algorithms. We will discuss some of them in chapter four (Hidden Markov model, Particle Filter and OOM).

3. 3. 1 Mobile Manipulator Examples

German Service Robotics Initiative project (DESIRE)

The project DESIRE [44], which was established by the German ministry of research involves partners in the German robotics community. The goal of the DESIRE project is to research and implement methods and algorithms for the development of service robots for domestic applications.

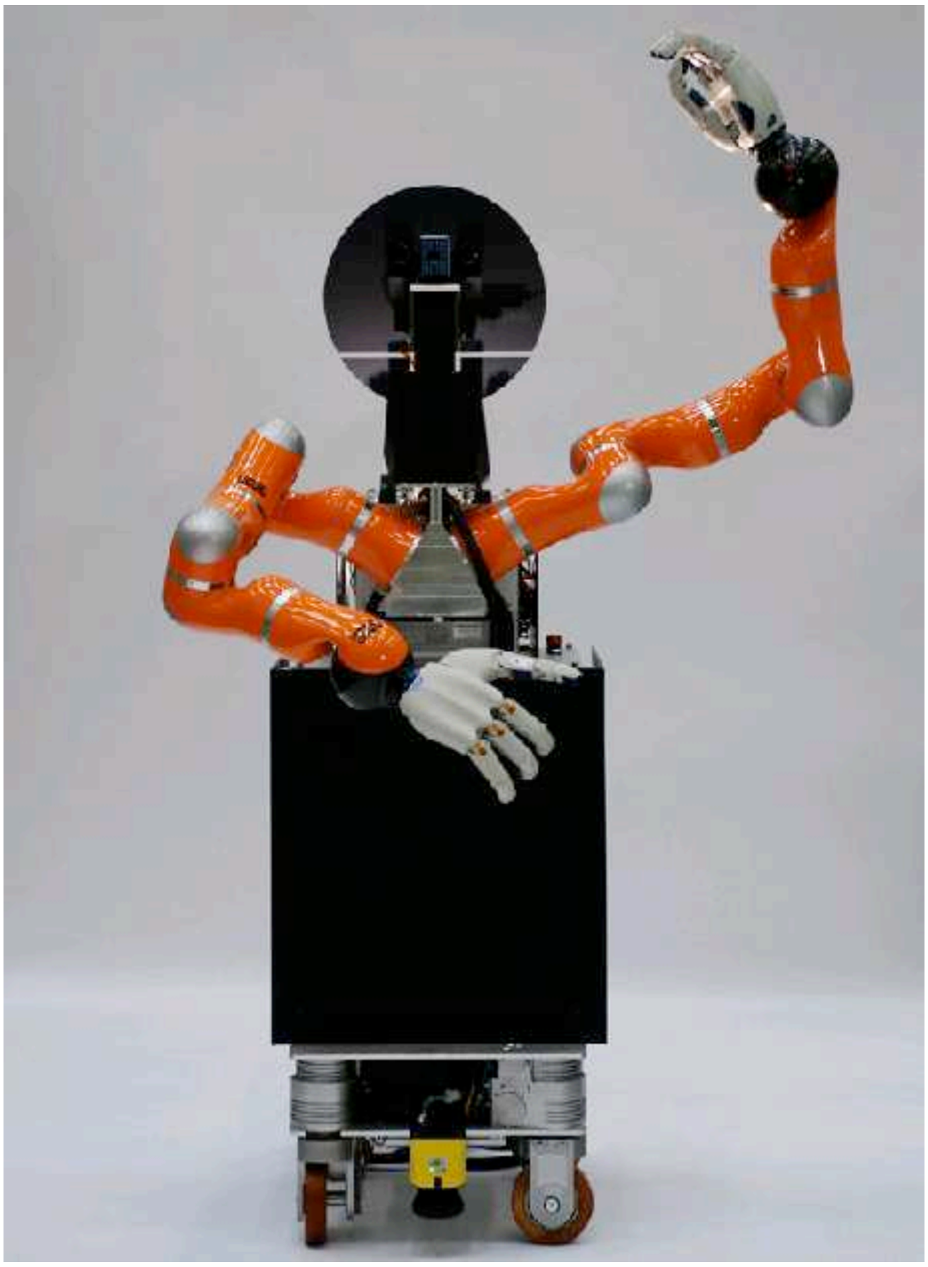

Figure 10 DESIRE robot [44]

Care-O-Bot

The Care-O-Bot [78] is a mobile manipulator service robot built by the Fraunhofer Institute. It operates in indoor environments with tasks such as fetch-and-carry and being a walking aid.

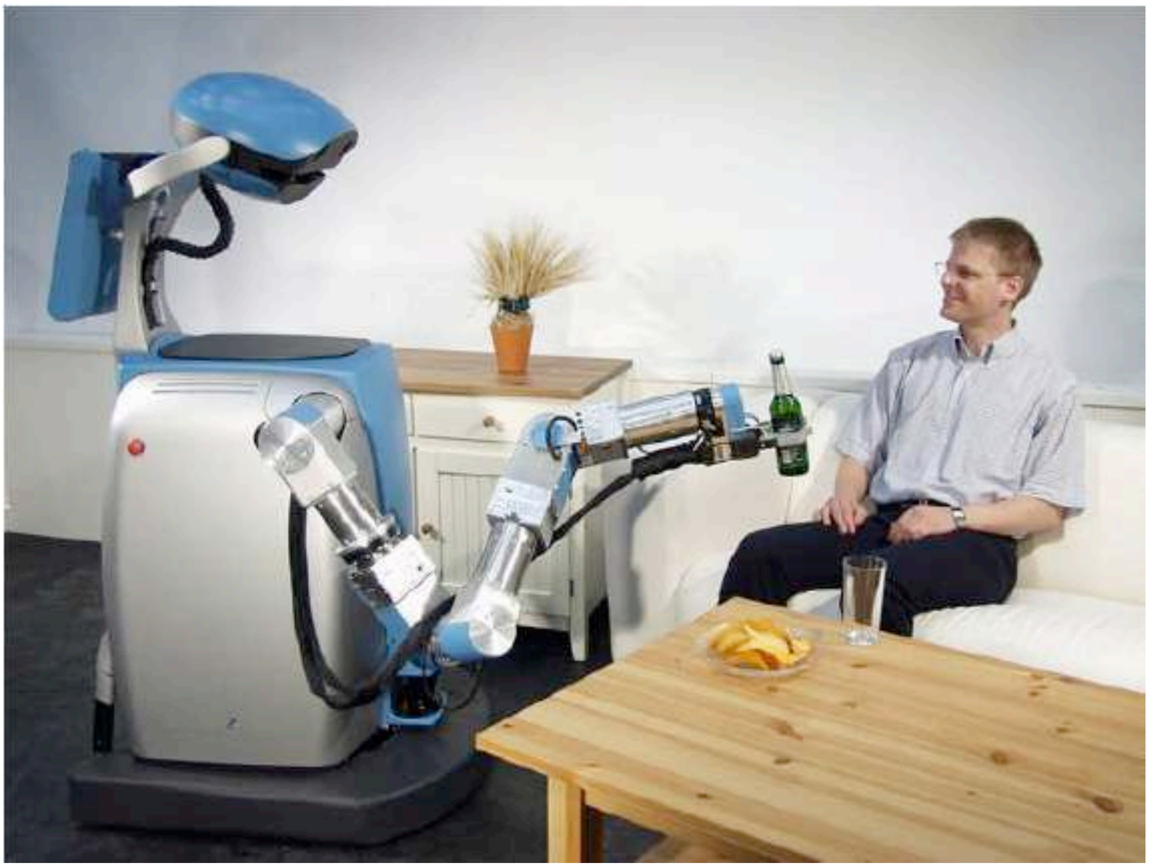

Figure 11 The Care-O-Bot II from the Fraunhofer Institute.
Picture: http://www.care-o-bot.de/

Stanford Artificial Intelligence Robot (STAIR)

Palmbots

The Palmbot [76] is equipped with a simple two degree-of-freedom nongrasping "palm manipulator" as shown in Figure 13. The palm can slide under, push, support, roll, or topple objects. Since the manipulator does not grasp the objects, the object can have a wide variety of shapes and sizes. A single robot can manipulate small objects, and two robots can cooperatively manipulate large objects.

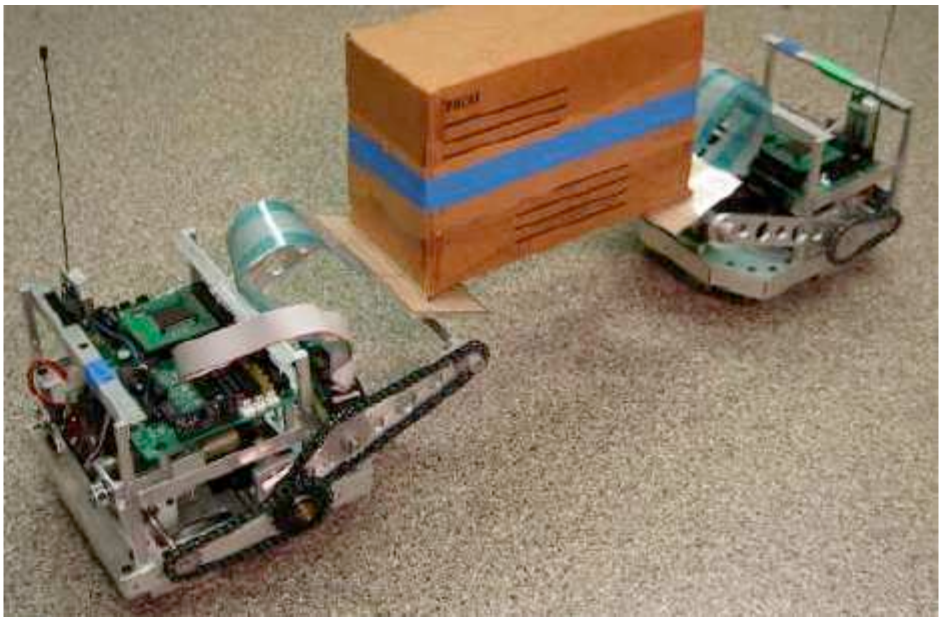

Figure 13 Palmbots [76]

4. Fault Diagnosis Methods

Fault diagnosis is a relatively new field of research in the robot domain. I met no techniques developed especially for the robot's needs. Developers mostly borrow the methods used for industrial applications and modify them for a particular robot. In this chapter we introduce and discuss four different fault diagnosis techniques, parity space (PS), hidden Markov model (HMM), particle filter (PF) and observable operator model (OOM) and examples of how they might be applied to the model of a four-wheel OMNI robot (see section 3. 2.).

4. 1 Parity Space and Principle Component Analysis

The idea of the parity space approach [9], [47], [14], [13], [48] for fault diagnosis is to deliver a technique for computing residual vectors which become non-zero if the actual system differs from the ideal system due to faults. From the analysis of the residuals, the fault diagnosis can conclude fault locations. Parity space is simple in computation and a straightforward method [14] which is applied to the linear state-space model with additive faults.

If no model is given a priori, but the relationship between input and output signals is known to be linear, then the principle component analysis (PCA) can be used as the tool to estimate a state-space model from the data.

4. 1. 1 Background Theory

The background theory presented by F. Gustafsson [1], [2] and J. Gertler [13], [48] will be summarized in this section.

A linear state-space representation (see section 2. 5) of the system data is observable and a data vector from the sliding window over time is constructed in the form (4. 1. 1)

$$Z_t = \begin{pmatrix} Y_t \\ U_t \end{pmatrix} \tag{4. 1. 1}$$

The aim is to compute a residual vector (4. 1. 2)

$$r_t = P^T Z_t \tag{4. 1. 2}$$

This residual vector is insensitive to the states and disturbances, but reacts on faults. The detection is performed based on the size of a residual and the isolation is achieved via direction of a residual.

Figure 14 presents the schematic representation of parity space as a fault diagnosis technique. The residual generates a deviation between the output y(t) and the model's "original output" computation. The conclusion about the fault location is postulated during the residual evaluation step.

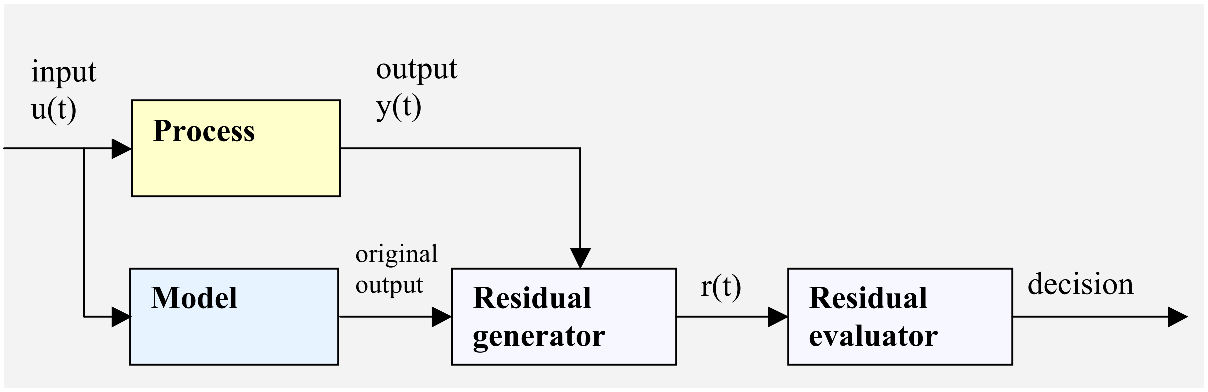

Figure 14 Structure of parity space algorithm for fault diagnosis system

To outline the basic idea of parity space methodology we consider a mixed stochastic-deterministic model represented by linear state-space equations

$$\begin{aligned} x_{t+1} &= A_t x_t + B_{u,t} u_t + B_{d,t} d_t + B_{f,t} f_t + B_{v,t} v_t \\ y_t &= C_t x_t + D_{u,t} u_t + D_{d,t} d_t + D_{f,t} f_t + e_t \end{aligned} \tag{4. 1. 3}$$

u_t – deterministic known inputs

d_t – deterministic unknown disturbance

f_t – deterministic unknown additive faults ($f_t = m_t * f^i$, f^i is all zero except for the element *i* which is one)

v_t – stochastic unknown state disturbance with zero mean and covariance matrix Q

e_t – measurement noise with zero mean and covariance matrix R

There are many approaches to derive the parity space. One of them is based on the discrete-time state-space model (4.1.3.), which uses data from a sliding window of size L. The measurements can be expressed explicitly in matrix form as

$$Y_t = Ox_{x_{t-L+1}} + H_u U_t + H_d D_t + H_v V_t + H_f F_t + E_t \tag{4. 1. 4}$$

H (Hankel matrix) is a function of the state-space matrixes defined as

$$H_s = \begin{pmatrix} D_s & 0 & \dots & 0 \\ CB_s & D_s & \dots & 0 \\ . & . & & . \\ . & & . & . \\ . & & & . \\ CA^{L-2}B_s & . & .. & CB_sD_s \end{pmatrix}$$

for all signals $s = u, d, f, v$

and the observation matrix

$$O = \begin{pmatrix} C \\ CA \\ . \\ . \\ . \\ CA^{L-1} \end{pmatrix}$$

The stuck measurement vector

$$Y_t = (y_{t-L+1}^T, \dots, y_t^T)^T \tag{4. 1. 5),}$$

is sampled from several time instants (normally over the sliding window of size L).

The inputs u_t, deterministic and stochastic disturbances d_t and vt are stacked into U, D and V accordingly.

The fault stacked vector for unity fault with constant magnitude m is defined as

$F_t = mF^i$ (for details, see [50]).

Residual Generator

If Y and U are stacked outputs and inputs from the sliding window, we can define a residual as

$$\begin{aligned} r_t &= w^T(Y_t - H_u U_t) \\ &= w^T(O_{x_{t-L+1}} + H_d D_t + H_f F_t + H_v V_t + E_t) \\ &= w^T(H_f F_t + H_v V_t + E_t) \end{aligned} \tag{4.1.6}$$

The residual r_t has to satisfy the following properties:

1. It should belong to parity space (w) defined by

   $w^T O = 0$ and $w^T H_d = 0$

   That implies insensitivity of the residual r to any initial state and disturbance.
2. The parity space also should satisfy

   $w^T H_f \neq 0$

   it means

   $r_t = w^T(Y_t - H_u U_t) = w^T H_f F_t \neq 0$

   whenever $F_t \neq 0$

Residual Analysis

The aim of a diagnosis system is to determine which fault(s) occurred. The residual vector r_t should have a different form for each fault.

If there are *m* different faults *f1, f2,..., fm*, the task is to define which *fi* has occurred. If m residuals can be designed in that way that the *i*-th residual is only affected by the *i*-th fault, then the fault isolation can be achieved easily [50].

This implies that the residual vectors should form a certain pattern, called *residual structure R*. Table 1 shows a residual structure *R* consisting of three faults. Here a 0 on position *(i, j)* means the residual in the row *i* is insensitive to a fault in column *j*, while a 1 means the residual *i* reacts on the fault *j*.

fault	F1	F2	F3
R1	1	0	0
R2	0	1	0
R3	0	0	1

Table 1 Residual structure R

There are two possible approaches to solve the isolation problem for the parity space algorithm:

a) Transformation matrix:

The transformation matrix T can be defined based on the residual structure R so that

$$Tr_t = T\omega^T H_f = R^i \qquad (4.1.7)$$

The isolation design is done by first choosing a residual structure R and then calculating the transformation matrix T from equation (4. 1. 7). This design assumes that the fault magnitude is constant within the sliding window.

b) Fault decoupling

In this algorithm each residual is designed separately by the condition

$W_i^T [OH_d H_f F^{-i}] = 0$

Here F^{-i} is a fault vector that includes all faults except for fault i. The advantage of this isolation technique is the insensitivity in residuals to measurement noise. The disadvantage is that more measurements are needed and that one projection W_i is needed for each fault.

4. 1. 2 The Algorithm

The parity space algorithm includes following steps :

Given: State-space model (4.1.3), input data U, measured data Y

Design parameters**:** sliding window size L, h- detection threshold and residual structure R.

Computation**:**

1. Compute the Model Matrixes O, H_d, H_u, H_f
2. Compute data-vectors Y_t and U_t over sliding window
3. Compute a Parity Space W
4. Compute a residual r_t
5. Perform detection
6. Perform isolation

In the fragment of the MATLAB-code we illustrate the computational steps 3, 4, 5 and 6.

We have defined design parameters and calculated model matrixes:

Design parameters: *R*-residual structure matrix, *h* – threshold

Computed matrixes: O, H_d, H_u, H_f

```
%    COMPUTE PARITY SPACE
%    Define the Null space N of (O, Hd)
[U,D,V]=svd([Q Hd]);
n=rank(D);
N=U(:,n+1:end);

%    calculate transformation matrix T $w^T H_f F_t^i = N^T T^T H_f F_t^i = R_{:,i}$
%    there to define the vectors f1, f2, f3 which are columns of R matrix
%    kron is Kronecker product
T = R / (N*Hf*kron(ones(L,1),[f1 f2 f3]));

%    Caclculate parity space
w = (T*N)’;

%    RESIDUAL ANALYSIS
%    Compute residual $r_t = w^T(Y_t - H_u U_t)$
r=w’*(Y-Hu*U) ;

%    Detection:
if r’r>h
% Isolation. Fault i in direction fi where: $i = \arg\max_i r^T R_i$, Ri is column i of R
   [val,i]=max(r’*R);
end
```

When there is no model available, we need an alternative approach to compute a correspondence to a parity space residual. If the relationship between input and output data is known to be linear the principle component analysis can be used.

4. 1. 3 Principle Component Analysis (PCA)

"Principal components analysis (PCA) is a technique used to reduce multidimensional data sets to lower dimensions for analysis." [52]

The detailed algorithm can be found in A. Hagenblad andF. Gustafsson paper [51]. Below we give a brief description of the PCA method.

As with parity space we transform the input and output into U and Y vectors. This is going to be our training data; it has been obtained from measured or simulated data. The data is combined into the vector

$$Z_t = \begin{pmatrix} Y_t \\ U_t \end{pmatrix}.$$

The principle component analysis is used to split up the data vector into parts, model $\hat{Z}_t$ and residual $\tilde{Z}_t$

$$Z_t = \begin{pmatrix} Y_t \\ U_t \end{pmatrix} = \hat{Z}_t + \tilde{Z}_t = P_x x_t + P_r r_t$$

P_r is a basis for the residual space (c.f. w^T in section 4.1.1.)

PCA Procedure

Given: input data U, measured data Y

Design parameters**:** sliding window size L, h- detection threshold and residual structure R, number of components.

Computation**:**

1. Estimate mean value μ and covariance matrix Σ from training data
2. Calculate singular value decomposition (SVD) of Σ.

   $\Sigma = PDP^T$

   P is a projection matrix that contains the principal components which are the eigenvectors associated to the eigenvalues λ_i.

$D = diag(\lambda_1 ... \lambda_m)$ is a diagonal matrix with eigenvalues as diagonal elements in a decreasing magnitude order.

3. Split the SVD into two parts as:

   $$P = (P_x P_r), \qquad D = \begin{pmatrix} D_x & 0 \\ 0 & D_r \end{pmatrix}$$

   The largest singular values are considered to be part of the model P_x and the other small singular values to the residual part P_r. Hence for a noise-free system the elements of D_r will be zero. The order of the model is a design parameter, i.e. the number of principle components which is needed to build a model.

4. Compute the model and residual

   $$\hat{Z}_t = P_x P_x^T Z_t$$

   $$\tilde{Z}_t = P_r P_r^T Z_t$$

Fault Diagnosis with PCA

The new data is depicted as stack vectors $Z_t = \begin{pmatrix} Y_t \\ U_t \end{pmatrix}$. They are projected into the projection matrix P_r. The result of the projection is the residual for this data set from which the diagnosis can be generated.

$$r_t = P_r^T Z_t$$

Since there is no model available, the isolation of the fault is a more difficult process compared to the parity space algorithm. If data about a particular fault is known then the fault can be estimated by calculating the corresponding residual and by estimating its mean and covariance [51].

4. 1. 4 Applications

We attempt to build a state-space model for a four-wheel OMNI robot and if possible (prerequisite: linear model), implement the parity space algorithm.

State-Space Model

The parity space method is applied to a mixed stochastic-deterministic model represented by linear state-space equations. The creation of state-space models for fault diagnosis is described in section 2.5.

To represent the localisation problem in a four-wheel OMNI robot we need a model of the robot to describe how a robot moves and turns and a sensor model, describing the sensor output as a function of the environment.

The position of the moving robot can be described with the following formula:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}\Bigg|_{t+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}\Bigg|_{t} + T \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}\Bigg|_{t} \begin{bmatrix} \upsilon_x \\ \upsilon_y \\ \omega \end{bmatrix}\Bigg|_{t}$$

x, y are the positions and θ is the orientation in a rectilinear two-dimensional coordinate system. The sample interval between time *t* and *t+1* is *T*. For the simplicity we assume that *T* is 1.

$\begin{bmatrix} \upsilon_x \\ \upsilon_y \\ \omega \end{bmatrix}$ is the robot speed vector which can be generated by using the formula (3.2.1.).

φ is the angle of deviation between robot frame and world frame. The angle is not constant – it depends on the robot angular velocity as the sum of the current angle and its calculated angular velocity: $\varphi|_t = \varphi|_{t-1} + \omega$. This property means that our state-space system is non-linear hence we can not apply parity space to the four-wheel OMNI robot example.

Nonetheless particle space and PCA were applied with success in various applications. PCA is widely used in chemical plants and as an on-board car-engine diagnosis for fault monitoring [53]. It seems parity space is applied in the car industry, introduced by GM and Daimler (only mentioned by Gertler [54]). In the robotics domain V. Filaretov and M. Vukobratovic apply non-linear parity space to a manipulator robot [55].

Toolbox: There are MATLAB frameworks in fault diagnosis for various methods which also support the parity space algorithm such as *A Fault Detection Toolbox for MATLAB* [56] and *MATLAB-based FDI-toolbox* [57].

4. 1. 5 Summary

The proposed algorithm is shown to be able to detect and identify faults for a linear state-space model with additive faults. The method computes a residual to detect that a fault has occurred. The vector is zero in no-fault case, and non-zero otherwise. To be able to identify a fault location, residuals are created in such a way, that each fault gets its own residual. In case no model is available a priori, training data in combination with PCA can be used. We can split the data in two parts, model and residual, by applying the *singular value decomposition* (SVD) of the covariance matrix for the given training data.

In spite of its simplicity in computation, the parity space approach has several disadvantages. The main disadvantage of this algorithm is its sensitivity to noise. Residuals become quite noisy even with low levels of measurement noise, or when the design model deviates from the original system. This problem is scarcely treated in the literature, and there are no design rules to be found. This is a critical point for the robot domain, because it is a difficult task to build a model that fully incarnates the original. Another drawback is that faults are not always suitably modelled as additive faults. PCA and parity space are restricted to the linear model but a lot of available model descriptions are non-linear, for instance the OMNI example presented in section 3. 2. With PCA no model is needed, but fault isolation will be more difficult. In case outliers corrupt the data, traditional PCA proves to be ineffective. Y. Tharrault, G. Mourot, J. Ragot, and D. Maquin [59] developed a robust, alternative version of PCA that seems to arrive at completely satisfactory results.

4. 2 Hidden Markov Model

In recent years, probabilistic models were successfully applied in the industrial and the robot domain. The probabilistic state-space formulation and the requirement for updating the states with new measurements are ideally fitted for the Bayesian model, which provides a general framework for the dynamic state estimation problem.
Hidden Markov model uses the probability calculus for modelling and reasoning actions and perceptions. The probabilistic model of a system is state-of-the-art in the robot domain because it is the right tool to represent the uncertainty in the robot's environment, in its perceptions and of its actions.

4. 2. 1 Background Theory

We replicate and summarize the insights of the tutorial paper [3] to describe the HMM theory.

In a stochastic system which can occupy one of N states x_t (state at time t), the state evolution is random. Any joint distribution can be factored into a series of conditional distributions:

$$p(x_0, x_1, \ldots, x_T) = p(x_0)\prod_{t=1}^{T} p(x_t \mid x_0, \ldots, x_{t-1})$$

This formula is the mathematical expression for the temporal process.

For a Markov process, the next state depends only on the current state:
$p(x_{t+1} \mid x_0, x_1, \ldots, x_t) = p(x_{t+1} \mid x_t)$.
Often, the term Markov chains is used to describe a discrete-time Markov processes [60].

$$p(x_0, x_1, \ldots, x_T) = p(x_0)\prod_{t=1}^{T} p(x_t \mid x_{t-1})$$

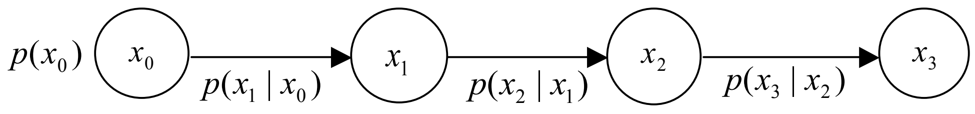

Figure 15 Graphical interpretation of Markov process [79].

We have a stationary Markov chain, if a process of change defined by some law is not changed over time. If this process has *N* states, then it can be described by a *NxN* transition matrix with elements defined as condition distributions:
$a_{ij} = p(x_{t+1} = i \mid x_t = j)$.

So far we have considered Markov models with directly visible states, but in a real-world application it is too restrictive an assumption and states might be only partially observable.
If the system is a Markov chain with unknown variables and observable evidence variables, then it can be described by a hidden Markov model (see Figure 16).

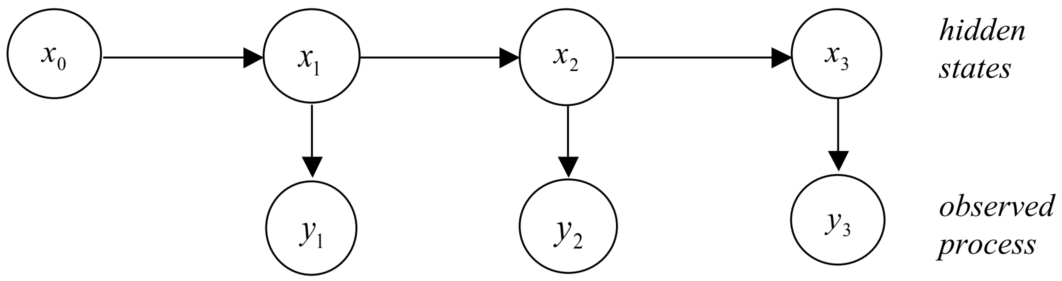

Figure 16 Architecture of hidden Markov model [79]

4. 2. 2 HMM Representation

Formalization of HMM is defined as a tuple $\lambda = \langle S, O, A, B, \pi \rangle$, satisfying the following conditions:

- $S = \{s_0, \ldots, s_{N-1}\}$ set of *N* system states
- $O = \{o_0, \ldots, o_{M-1}\}$ set of *M* observations
- *A* is a *N x N* transition probability matrix, its entries describing the probability that one state becomes another state $A_{i,j} = P(q_{t+1} = s_j \mid q_t = s_i)$, $1 \le i \le N$, $1 \le j \le N$
- *B* is a *N x M* observation matrix, its elements are the probability of observing an event related to the given state $B_{j,k} = P(v_t = o_k \mid q_t = s_j)$, $1 \le k \le M$, $1 \le j \le N$

- π is an initial distribution vector i.e. the start state of the system. $\pi_i = P(q_1 = S_i)$, $1 \le i \le N$

The system at any time step lays in one of the state. This state is hidden and not directly observable, but some observable variables about the state, are obtained.

4. 2. 3 HMM Problems and Solutions

Three Fundamental Problems

HMM provides a formal mathematical solution to three fundamental problems [3]:

1. Definition of probability of observable sequence for given HMM $P(O|\lambda)$
2. Definition of sequence of states leading to sequence of observations for HMM
3. Definition of HMM based on sequence of observations

Solutions

1. Definition of the probability of an observable sequence

The solution of the problem is Forward procedure (for details [3])
The idea is to define the forward variable for the given sequence of observations $O_1, O_2, ...O_t$ which ended up in state S_i:

$\alpha_t(i) = P(O_1 O_2 ... O_t \wedge q_t = S_i)$ where $1 \le t \le T$

The $\alpha_t(i)$ can be computed recursively:

a.) Initialization:

$\alpha_1(i) = P(O_1 \wedge q_1 = S_i) = P(q_1 = S_i) P(O_1 \mid q_1 = S_i)$

$P(q_1 = S_i)$ is the initial probability of being in state S_i and

$P(O_1 \mid q_1 = S_i)$ the element in observation matrix.

b.) Induction:

$\alpha_{t+1}(j) = P(O_1 O_2 ... O_t O_{t+1} \wedge q_{t+1} = S_j)$

$$= \sum_i P(q_{t+1} = S_j \mid q_t = S_i) P(O_{t+1} \mid q_{t+1} = S_j) \alpha_t(i)$$

$P(q_{t+1} = S_j \mid q_t = S_i)$ is the element of the transition matrix and

$P(O_{t+1} \mid q_{t+1} = S_j)$ the given element of observation matrix.

2. Most probable path (MPP)

This class of problems is solved using the Viterbi algorithm [3],[80].
The MPP algorithm is a recursive relationship between the most likely path to each state x_{t+1} followed by the transition $x_t \to x_{t+1}$.

3. Learning algorithm

The third problem of HMMs is to determine a method to adjust the model parameters $(A,\ B, \pi)$ to maximize the probability of the observation sequence. To choose $\lambda = (A, B, \pi)$ in such a way that $P(O \mid \lambda)$ is maximized. The method is known as Baum-Welch method or expectation-modification (EM) method [3], [80] .
In order to describe the procedure of an iterative update and an improvement we first introduce the auxiliary parameters:
$\gamma_t(i) = P(q_t = S_i \mid O_1 O_2 ... O_T, \lambda)$
$\varepsilon_t(i, j) = P(q_t = S_i \wedge q_{t+1} = S_j \mid O_1 O_2 ... O_T, \lambda)$
$\gamma_t(i)$ is the probability of being in state S_i at time *t*, given the observation sequence and the model
$\varepsilon_t(i, j)$ is the probability of being in state S_i at time *t*, and state S_j at time *t+1*, given the model and the observation sequence
If we sum up $\gamma_t(i)$ and sum up $\varepsilon_t(i, j)$ over a certain time period, we get quantities which can be interpreted as:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{Expected number of transitions} \qquad (4.\ 2.\ 1)$$

out of state *i* during the path

$$\sum_{t=1}^{T-1} \varepsilon_t(i,j) = \text{Expected number of transitions from state } i \text{ to state } j \text{ during the path} \qquad (4.\,2.\,2)$$

Using the (4. 2. 1) and (4. 2. 2) formulas we can give a method for re-estimation of the model parameters of a HMM:

$$\bar{\pi}_i = \text{expected frequency in state } S_i \text{ at time (t=1)} = \gamma_1(i) \qquad (4.\,2.\,3)$$

$$\bar{a}_{ij} = \text{(expected number of transitions from state } S_i \text{ to state } S_j\text{)/}$$
$$\text{(expected number of transitions from state } S_i\text{)}$$
$$= \frac{\sum_{t=1}^{T-1} \varepsilon_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \qquad (4.\,2.\,4)$$

$$\bar{b}_j(k) = \text{(expected number of time in state } S_j \text{ and observing symbol } v_k\text{)/}$$
$$\text{(expected number of times from state } S_j\text{)}$$
$$= \frac{\sum_{\substack{t=1 \\ s.t.O_t = v_k}}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)} \qquad (4.\,2.\,5)$$

So if we know λ, we can estimate the expectation of quantities such as the expected number of times in state *i* and the expected number of transitions from state *i* to state *j*. If we know the quantities such as the expected number of times in a state and as an expected number of transitions from state *i* to state *j*, we can estimate the maximal likelihood of $\lambda = \langle \{a_{ij}\}, \{b_j(k)\}, \pi_i \rangle$.

Algorithm scheme

1. Get the observation sequence $O_1..O_T$
2. Define the initial model as $\lambda = (A, B, \pi)$.
3. Compute new estimates $\bar{A}, \bar{B}\, and\, \bar{\pi}$ based on equations (4. 2. 3), (4. 2. 4), (4. 2. 5) using the model λ so the re-estimated model $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ is found.
4. Analyze the re-estimated model $\bar{\lambda}$. It can be either
   a) identical to the initial one $\bar{\lambda} = \lambda$ (**termination criteria**), i.e. λ defines a critical point of the likelihood function or
   b) more likely than model λ in the sense that $P(O \mid \bar{\lambda}) > P(O \mid \lambda)$, i.e., we have found a new model $\bar{\lambda}$ from which the observation sequence is more likely to have been produced. Then we need again **go to step 2** of the algorithm.

If we iteratively use $\bar{\lambda}$ in place of λ and repeat the re-estimation calculation, we then can improve the probability of O being observed from the model until some limited point is reached.

EM (expectation-modification) is successfully applied for problems such as the speech recognition, in biology the recognition of an albumen structure.
The essential drawback of the learning algorithm is that it gets stuck in a local maximum and leads to a wrong estimated model.
The EM approach is guaranteed to converge to a local maximum of the likelihood. There is no guarantee that the algorithm will find the global maximum. Often the value of the local maximum critically depends on the initial settings of the parameters. According to Rabiner [3] the best parameter initialization is a thorny task:

"*Experience has shown that either random (subject to the stochastic and the nonzero value constraints) or uniform initial estimates of the π and A parameters is adequate for giving useful re-estimates of these parameters in almost all cases. However, for B parameters, experience has shown that good initial estimates are helpful in the discrete* *case, and are essential in the continuous distribution case. Such initial estimates can be obtained in a number of ways, including manual segmentation of the observation sequence(s) into states with averaging of observations within states, maximum likelihood segmentation of observations with averaging, and segmental k-means segmentation with clustering.*"

4. 2. 4 Application of HMM in Fault Diagnosis

For systems based on discrete states, the applications *GDE/Sherlock* [15], [16], *Livingstone1, 2* [17], [18] and *Titan* (reactive model-based programming) [18], [19] provide a framework that can be used efficiently for both diagnosis and recovery. These algorithms obtain reliability by estimating the system state from the set of measurements as a "most probable path" [3].

4. 2. 5 Numerical Example

The objective of this subsection is to apply the HMM forward-algorithm to the four-wheel OMNI robot for fault diagnosis. In order to analyse and make an inference about the dynamic system of the robot, two models are required. We need a system model to describe the evolution of the states and a measurement model which describes the relation between states and measurements. These models can be designed in probabilistic form, where states correspond to normal and faulty conditions of the robot. The groups of possible faults are described in section 3. 2. Each group consists of one or several fault modes.

1. Normal operation **N** (no fault);
2. The group of abrupt motor faults consists of four faults namely **W1, W2, W3** and **W4**. They correspond to a wrong output value of the wheel motors 1, 2, 3 and 4.
3. The stuck motor group is comprised of the faults **M1s, M2s, M3s** and **M4s.** They describe the output of wheel motors 1, 2, 3 and 4 being fixed to a constant value regardless of the input.
4. Gradual degradation of performance includes the faults **M1d, M2d, M3d** and **M4d**. They correspond to the output of wheel motors 1, 2, 3 and 4 being multiplied with a negative exponential function.

Altogether we have 13 system states. Following the HMM theory, we construct a modification of states as a probability table (the notations for the table were taken from [81].

i	$p(q_{t+1} = s_1 \mid q_t = s_i)$	$p(q_{t+1} = s_2 \mid q_t = s_i)$	…	$p(q_{t+1} = s_{12} \mid q_t = s_i)$	$p(q_{t+1} = s_{13} \mid q_t = s_i)$
1	$a_{1,1}$	$a_{1,2}$	…	$a_{1,12}$	$a_{1,13}$
2	$a_{2,1}$	$a_{2,2}$	…	$a_{2,12}$	$a_{2,13}$
:	:	:	:	:	:
12	$a_{12,1}$	$a_{12,2}$	…	$a_{12,12}$	$a_{12,13}$
13	$a_{13,1}$	$a_{13,2}$	…	$a_{13,12}$	$a_{13,13}$

Table 2 Transition matrix

Notation $a_{i,j}= p(q_{t+1} = s_j \mid q_t = s_i)$ is the probability distribution for the next state given current.

According to the table we can construct the following system model for four-wheel OMNI robot:

[0.8 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0 0 0 0;
0.075 0.6 0.025 0.025 0.025 0.1 0.1 0.025 0.025 0 0 0 0;
0.075 0.025 0.6 0.025 0.025 0.025 0.025 0.1 0.1 0 0 0 0;
0.5 0.2143 0.2143 0.6 0.2143 0.2143 0.2143 0.2143 0.2143 0.1 0.1 0 0;
0.5 0.2143 0.2143 0.2143 0.6 0.2143 0.2143 0.2143 0.2143 0 0 0.1 0.1;
0.025 0.2 0.025 0.025 0.0083 0.5 0.2 0.0083 0.0083 0 0 0 0;
0.025 0.2 0.025 0.025 0.025 0.2 0.5 0 0 0 0 0 0;
0.025 0.025 0.2 0.025 0.025 0 0 0.5 0.2 0 0 0 0;
0.025 0.025 0.2 0.025 0.025 0 0 0.2 0.5 0 0 0 0;
0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.5 0.2 0 0;
0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.2 0.5 0 0;
0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.5 0.2;
0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.2 0.5]; (4. 2. 6)

The robot has some sensors and it can gather state information. We assume that in our four- wheel OMNI robot example a measurement relates to each state. Table 3 shows a compact representation of probabilities of measurements depending on a system-state.

i	$p(O_t = 1 \mid q_t = s_i)$	$p(O_t = 2 \mid q_t = s_i)$	$\cdots$	$p(O_t = 12 \mid q_t = s_i)$	$p(O_t = 13 \mid q_t = s_i)$
1	$b_1(1)$	$b_1(2)$	$\cdots$	$b_1(12)$	$b_1(13)$
2	$b_2(1)$	$b_2(2)$	$\cdots$	$b_2(12)$	$b_2(13)$
:	:	:	:	:	:
12	$b_{12}(1)$	$b_{12}(2)$	$\cdots$	$b_{12}(12)$	$b_{12}(13)$
13	$b_{13}(1)$	$b_{13}(2)$	$\cdots$	$b_{13}(12)$	$b_{13}(13)$

Table 3 Probability of measurements

Notation $b_i(k) = p(O_t = k \mid q_t = s_i)$

The measurement probability matrix is:

[0.8 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0 0 0 0;
0.075 0.6 0.025 0.025 0.025 0.1 0.1 0.025 0.025 0 0 0 0;
0.075 0.025 0.6 0.025 0.025 0.025 0.025 0.1 0.1 0 0 0 0;
0.075 0.025 0.025 0.6 0.025 0.025 0.025 0 0 0.1 0.1 0 0;
0.075 0.025 0.025 0.025 0.6 0.025 0.025 0 0 0 0 0.1 0.1;
0.025 0.2 0.025 0.025 0.025 0.5 0.2 0 0 0 0 0 0;
0.025 0.2 0.025 0.025 0.025 0.2 0.5 0 0 0 0 0 0;
0.025 0.025 0.2 0.025 0.025 0 0 0.5 0.2 0 0 0 0;
0.025 0.025 0.2 0.025 0.025 0 0 0.2 0.5 0 0 0 0;
0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.5 0.2 0 0;
0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.2 0.5 0 0;
0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.5 0.2;
0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.2 0.5]; (4. 2. 7)

Notice that since neither the real robot nor its simulator was available the probabilistic models were constructed approximately by hand.

The diagnosis of the behaviour modes for the robot, based on probabilistic models, can be conducted by the HMM forward algorithm. K. Murphy developed the MATHLAB toolbox which supports the inferences and a learning algorithm for HMMs. We applied the methods of this toolbox to perform fault diagnosis in the four-wheel OMNI robot. The script hmm4wheelOMNI.m gives the example (see Appendix A.) The diagnosis was executed in the following steps:

1. Set given parameters: model matrixes, forward kinematics of the robot
2. Define design parameters: time *T*, start state prior0
3. Load/generate input and measurement data
4. Preprocess measurement data
5. Apply forward-backward algorithm
6. Plot results

Given Parameters

The example is applied to the probabilistic model described in the matrixes above (4. 2. 6) and (4. 2. 7). Besides system and measurement matrixes the forward kinematics matrix of the four-wheel OMNI robot is given.

```
controlMat(:,:)=[sin(angle) -sin(angle) -sin(angle) sin(angle);...
                 -cos(angle) -cos(angle) cos(angle) cos(angle);...
                 1/(4*R) 1/(4*R) 1/(4*R) 1/(4*R)];
```

We will need it for preprocessing the sensor data.

Design Parameters

A user can modify these design parameters to check experimental results under various conditions like time steps, accuracy and start state of the robot.

Load/generate input and measurement data

The sensor measurements and input wheel velocities for each time step are saved in the *RobotPoseData.mat* file. The user can load all the variables from this file or create new ones or generate data inside of the script. The information is given in the following variables:

- *u(:,:) – 4xT* matrix of input data. To each time step a vector of robot motor velocities corresponds $(v_1, v_2, v_3, v_4)^T$. By collecting the vectors we receive the input matrix u.
- *velObs(:,:) – 3xT* matrix of measurements. As measurements data we use linear and angular velocities of the robot.

Preprocess Measurement Data

The data from the matrix *velObs(:,:)* are analysed to construct measurements which can be observed in the system states. As we postulated in the observation matrix, one measurement corresponds to each robot state (see the observation matrix (4. 2. 7)). Now we consider the part of the script which encodes this analysis.

Normal state at time t

A robot operating in a normal condition means that its measurement vector *velObs(:,t)* is identical to the original one *robotVelN(:,t)* which is generated from the following equation:

*robotVelN(:,t)=controlMat(:,:,1)*u(:,t);*

Note: Since all fault cases belonging to one group can be deduced in a similar manner the only case with motor 1 for each group will be considered.

Sudden motor fault at time step t

A fault has occurred when

velObs (:, t) ~= robotVelN (:, t)

We define that it is wheel motor 1 which produces the wrong output.

The forward kinematics for obtaining the robot velocities from the given wheel velocities are given in section 3. 2 by the expression (3. 2. 1) .

Applying this expression for our notation we receive following equations:

$$\begin{gathered} u(1,t)\sin(angle) - u(2,t)\sin(angle) - u(3,t)\sin(angle) + u(4,t)\sin(angle) = robotVelN(1,t) \\ -u(1,t)\cos(angle) - u(2,t)\cos(angle) + u(3,t)\cos(angle) + u(4,t)\cos(angle) = robotVelN(2,t) \\ \frac{u(1,t)}{4R} + \frac{u(2,t)}{4R} - \frac{u(3,t)}{4R} + \frac{u(4,t)}{4R} = robotVelN(3,t) \end{gathered} \tag{4. 2. 8}$$

If only wheel 1 produces the wrong output and the other wheels' outputs are correct, then the following statement holds true:

*robotVelN(1,t) - velObs(1,t) = u(1,t)sin (angle) - ErrValue*sin (angle)*

*robotVelN(2,t) - velObs(2,t) =- u(1,t)cos (angle) + ErrValue*cos (angle)*

robotVelN(3,t) - velObs(3,t) =- u(1,t)/4R+ ErrValue/4R

ErrValue is the wrong value of wheel motor 1 output.

We express the *ErrValue* from the three equations and save the results in the *velMatrix(:,1)*

```
velMatrix(:,1)=[(velObs(1,t)-robotVelN(1,t))/sin(angle)+u(1,t)+smallErr;
                (-velObs(2,t)+robotVelN(2,t))/cos(angle)+u(1,t)+smallErr;
                4*R*(velObs(3,t)-robotVelN(3,t))+u(1,t)+smallErr];
```

If the differences *(velMatrix(1,1)-velMatrix(2,1))*, (*velMatrix(2,1)-velMatrix(3,1)*) and *velMatrix(1,1)-velMatrix(3,1)*) are equal to zero then a fault with wheel motor 1 occurred.

Stuck motor

To diagnose this group of faults we need to analyse the sequence of length *L* of the latest outputs. If from *L* latest observations we can conclude that for example the wheel motor 1 produces the same wrong output (*ErrValue*) then the measurement “stuck wheel 1” (W1s) is observed.

“Gradual degradation” of motor

The fault attacking the system grows in proportion to time, for example if the wheel velocity differs from the expected velocity only slightly but increases with time, then a “gradual degradation” of the motor occurred.

To diagnose this scenario, the sequence of L latest outputs has to be observed and a “decay rate” has to be generated as *1-4*R*((robotVelN(3,t)-velObs(3,t))/u(1,t))*.

If the “decay rate” belongs to segment (0, 1) and decreases with each time step, we have a “gradual degradation” scenario.

Forward-Backward Algorithm

After preprocessing the measurement data the observation matrix can be constructed.

If, for example, the measurement sequence corresponds to the observations of states 1, 1, 1, 8, 8, 8 (1 = normal behaviour, 8 = wheel motor 2 stuck) then the observation matrix for our example is the 13x6 matrix. Each column in this matrix corresponds to a column from *obsmat* with a number from the sequence (1, 1, 1, 8, 8, 8). Hence the observation matrix for the given sequence is [*obsmat(:,1) obsmat(:,1) obsmat(:,1) obsmat(:,8) obsmat(:,8) obsmat(:,8)*]

```
% Create observation matrix
for i=1:T
  obsmat1(:,i)=obsmat(:,obserVal(i));
end
```

Now the forward method (a function of HMM toolbox) to estimate the probability of the current state is executed.

```
[alpha, beta, gamma, loglik] = fwdback(prior0, transmat0, obsmat1, 'act', act);
```

The arguments of the function are the system matrix (4. 2. 6) transmat0, the generated observation matrix obsmat1, the initial state prior0 and the control inputs (in this example we do not use the parameter *act*, therefore it is zero). Only the return value *alpha* includes the probability distribution of system states. The other parameters are outside the scope of the task.

Plot Results

The following results have been simulated by this script:

1. Diagnosis of the "gradual degradation" fault of wheel motor 4

Given: T=10, L=4.
Original robot behaviour: The robot starts to run in normal conditions but from time step 2 onwards a "gradual degradation" of wheel motor 4 occurs.

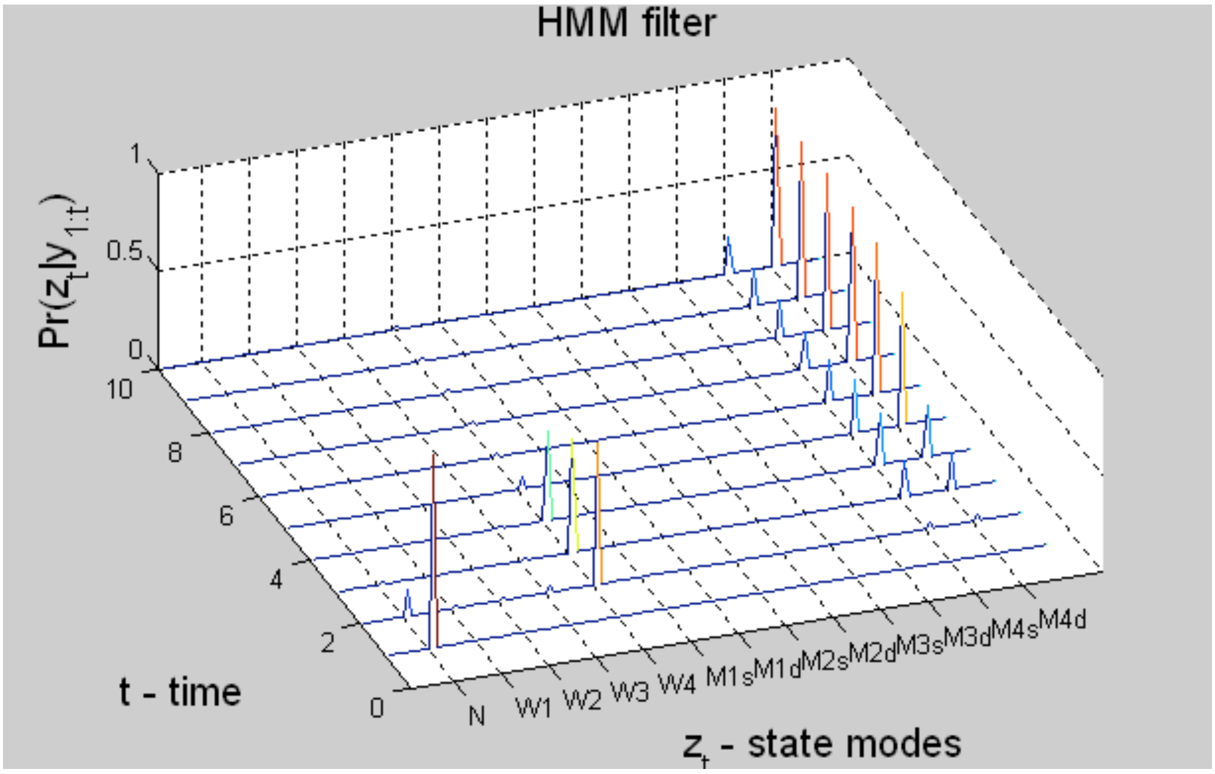

Figure 17 Simulation results for the "gradual degradation of wheel motor 4" scenario

Notice that in this example the parameter, *L*=4 thus the first three time steps the system diagnose only *Sudden motor4* faults

2. Diagnosis of the "gradual degradation" fault of wheel motor 1

Given: *T*=10, *L*=4

Original robot behaviour**:** Robot starts to run in normal conditions but from time step 2 "gradual degradation of motor 1" arise.

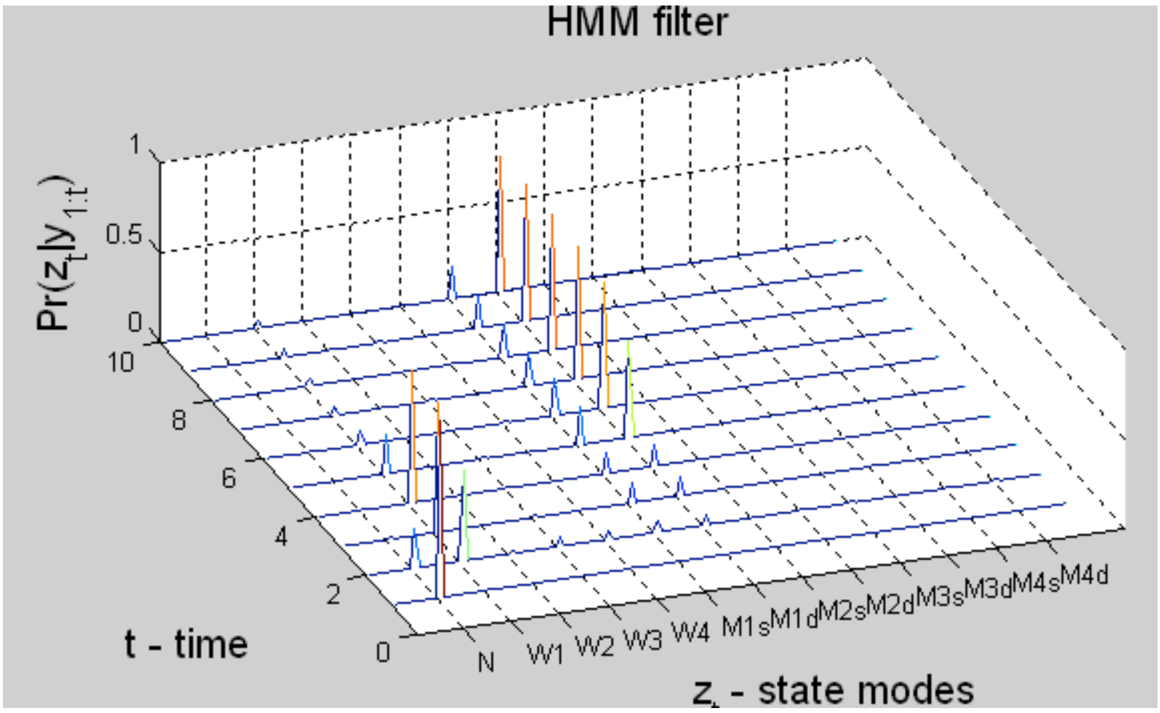

Figure 18 Simulation results for the "gradual degradation of motor 1" scenario

3. Diagnosis of the stuck motor 1

Given: T=90, L=4

Original robot behaviour: Until time step 50 robot runs in normal condition, then wheel motor 1 gets stuck.

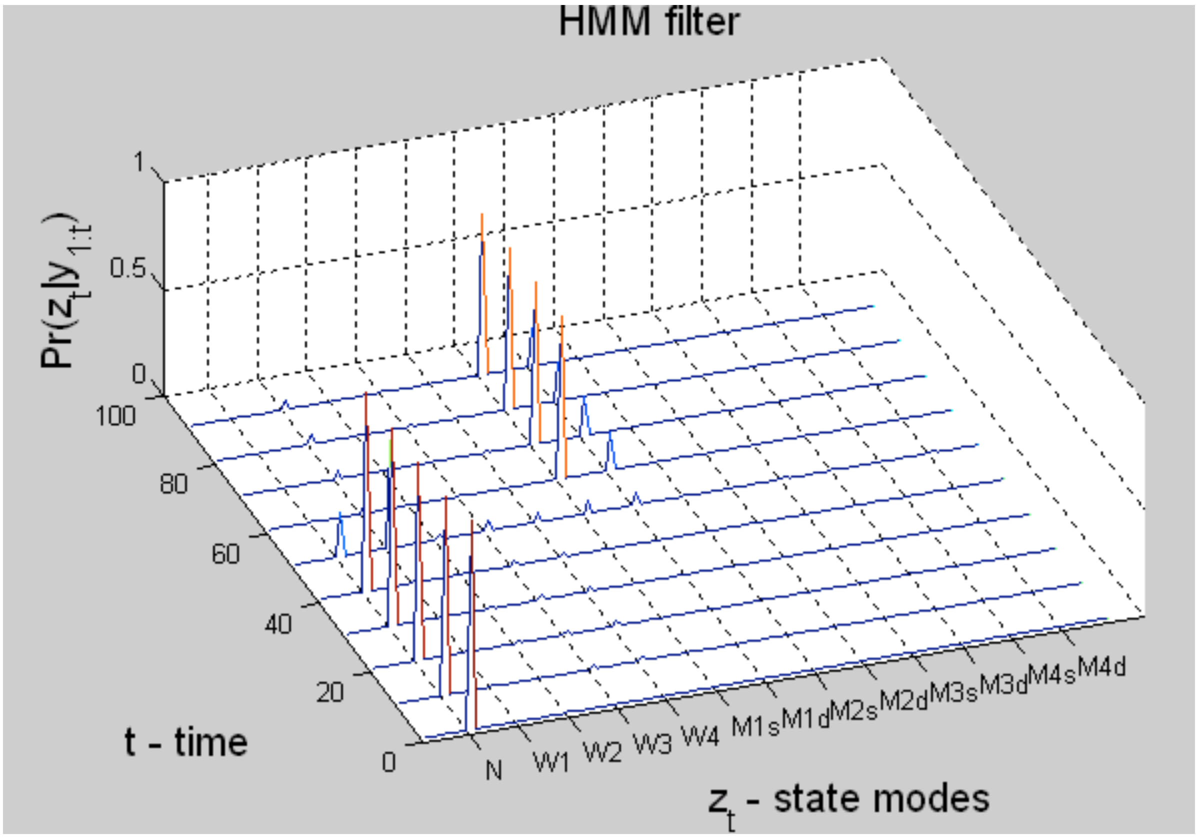

Figure 19 Simulation results for the stuck motor1 scenario

4. Diagnosis of "sudden faults" of motor 1

Given: T=10, L=4

Original robot behaviour: Sudden faults of the motor 1 in time step 5,9,10

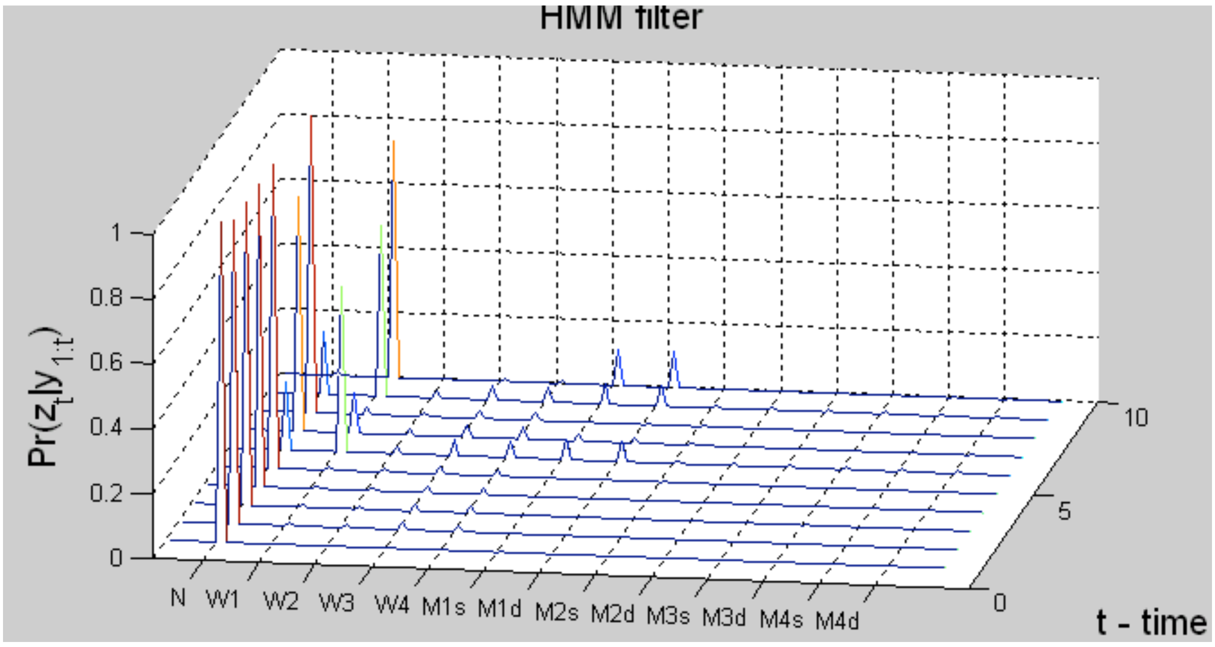

Figure 20 Simulation results for the "sudden faults of motor 4" scenario

4. 2. 6 Summary

The Parity space approach described in the previous section could not satisfy all needs of fault diagnosis for robot systems, since it requires a well-defined system model and is only applicable for additive faults. The other fault diagnosis model-based technique is the hidden Markov model. HMM is a temporal probability model of stochastic processes composed of a transition model describing the evolution and a sensor model describing the observation process. It solves inference problems with forward-backward algorithms; the practical examples given in this section illustrated the accuracy of the method. Williams [18], [19] presented the successful implementation of HMM for fault diagnosis. The disadvantage of HMM as a fault diagnosis method is that it supports only discrete states. To increase the robustness of the system over a long period of time, one needs to use models that describe both the discrete stochastic behaviour and the continuous dynamics of it.

HMMs are usually trained using the expectation-maximization (EM) algorithm (the practical example will be given in subsection 4. 4. 4). As a learning algorithm, HMM is not entirely satisfactory due to slow convergence and the presence of many suboptimal solutions.

In spite of some drawbacks, HMM takes a leading role in application areas such as speech recognition, bio sequence analysis and control engineering.

4. 3 Particle Filter (PF)

Particle Filters [1] are powerful methods to track probability distribution over state variables of complex systems with mixtures of discrete and continuous variables. Particle Filters are the techniques for implementation of recursive Bayesian filters by Monte Carlo sampling.

4. 3. 1 Background Theory

Bayesian filtering is a general tool used for estimating the states of a dynamic system from sensor measurements based on a predict/update cycle. The estimation of the probability about the current state based on a sequence of observations and input data (see figure 21) can be calculated recursively using the Bayesian filter [25].

$$p(s_t \mid z_{0:t}, u_{0:t-1}) = \eta_t p(z_t \mid s_t) \int p(s_t \mid s_{t-1}, u_{t-1}) p(s_{t-1} \mid z_{0:t-1}, u_{0:t-2}) ds_{t-1} \qquad (4.\ 3.\ 1)$$

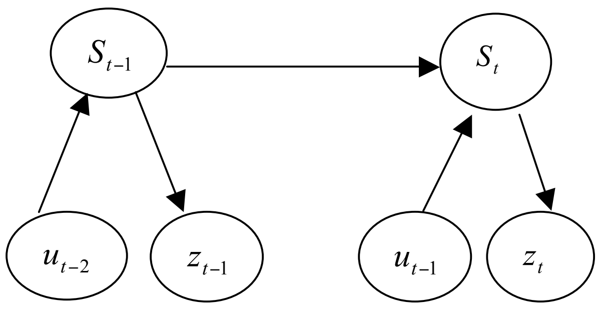

Figure 21 Graphical model for the dynamic system in an abstract view [25]

S_t - state at time t

u_t - input or control command at time t

z_t - observation at time t

The prediction stage uses the system model to predict the state probability distribution from the current estimation onwards.

$$p(s_t \mid z_{1:t-1}, u_{t-1}) = \int p(s_t \mid s_{t-1}, u_{t-1}) p(s_{t-1} \mid z_{0:t-1}, u_{0:t-2}) ds_{t-1}$$

The update operation uses the latest measurement to modify the prediction [4].

$$p(s_t \mid z_t, z_{0:t-1}, u_{0:t-1}) = \eta_t p(z_t \mid s_t) p(s_t \mid z_{1:t-1}, u_{t-1})$$

Particle filter is used as a sample based representation of the Bayesian filter (4. 3. 1) The underlying theory is based on the work of "Architectures for Efficient Implementation of Particle Filters" presented by M.Bolic [82]

The principle idea behind particle filters is to represent the posterior probability by a set of random particles with associated weights and then compute estimates based on these sampling and weights.

More specifically, at every time instant *n* a random measure $\{s_{0:n}^{(m)}, w_n^{(m)}\}_{m=1}^{M}$ is defined, where $s_n^{(m)}$ is the *m*-th particle of the state at time *n,* $s_{0:n}^{(m)}$ is the *m*-th trajectory of the state, and $w_n^{(m)}$ is the weight of the *m*-th particle (or trajectory) at time instant *n*. If these particles are obtained from the observations $z_{0:n}$ and the trajectories are drawn from the conditional probability $p(s_{1:n} \mid z_{1:n})$, then the particles approximate this probability by

$$p(s_{0:n} \mid z_{0:n}) \approx \sum_{m=1}^{M} w_n^{(m)} \delta(s_{0:n} - s_{0:n}^{(m)}).$$

The implementation of Particle Filters involves three important operations:

1. Generation of particles (sample step),
2. Computation of the particle weights (importance step)
3. Resampling

There is a family of particle filters that is based only on the first two steps (*Sequential Importance Sampling Filter*). The filters that perform all three operations are called *Sample Importance Resampling Filters* (SIRF).

1. Generation of particles

The generation of particles $s_n^{(m)}$ is performed by drawing them from an importance density function $\pi(s_n)$. If we choose an importance density function

$$\pi(s_{0:n}) = \pi(s_1 \mid z_1) \prod_{1}^{n} \pi(s_k \mid s_{0:k-1}, z_{0:k}),$$

we can compute the weights of the particles recursively:

$$s_n^{(m)} \sim \pi(s_n \mid s_{n-1}^{(m)}, z_{0:n})$$

The importance density $\pi(s_n \mid s_{n-1}, z_{1:n})$ plays a basic role in the design of particle filters, because it generates particles that have to represent a desired probability. If the drawn particles are in regions where the probability has small values, the estimates obtained from the particles and their weights would be poor and subsequent tracking of the signal would very likely diverge. By contrast, if the particles are from regions where the probability mass is significant, the Particle Filter will have improved performance.
Various strategies have been proposed for design density functions [64], [65]. One might argue that the optimal importance density function should be designed as a target distribution

$$\pi(s_n \mid s_{n-1}, z_{0:n}) = p(s_k \mid s_{k-1}, z_k).$$

However, the drawbacks of this strategy are the difficulties to sample and perform weight calculation. Another strategy for drawing particles is to use a transition prior as an important density function

$$\pi(s_n \mid s_{n-1}, z_{0:n}) = p(s_k \mid s_{k-1})$$

2. Computation of the particle weights

The importance step consists of two steps: computation of the weights and normalization. In the former step the weights are evaluated up to a proportionality constant and subsequently, in the latter they are normalized. If the importance function has the form, the weights are updated via

$$w_n^{*(m)} = w_{n-1}^{(m)} \frac{p(z_n \mid s_n^{(m)}) p(s_n^{(m)} \mid s_{n-1}^{(m)})}{\pi(s_n^{(m)} \mid s_{0:n-1}^{(m)}, z_{0:n})}$$

After applying this formula the weights should normalized.

3. Resampling

While time progresses, few weights become very large and some of the particles decrease in weight so that they become negligible. The resampling is the procedure for removing the trajectories that have small weights and focus on dominating trajectories.
There are various standard algorithms used for resampling, such as residual resampling (RR), branching corrections [67] and systematic resampling (SR) [66].

4. 3. 2 Particle Filter Enhancements

The Particle Filter approach becomes several modifications. The reason for this is that classical filters have some drawbacks applied to the problem of fault diagnosis.
Authors in their work define some challenges for online diagnosis problems which are difficult to address only by classical Particle Filter algorithm [63]. There are

1. Very low prior fault probabilities
2. Restricted computational resources
3. High dimensional state space (number of samples grows exponentially with the dimensionality of a problem)
4. Non-linear stochastic transitions and observations. Ability to apply the algorithm to non-linear models
5. Multimodal system behaviour

There are various approaches to addressing theses problems in the literature.
The goal of the *Risk Sensitive Particle Filter* algorithm [21] [22] by S. Thrun is to increase the amount of particles in "risky" or important states. The concept is to identify a risk function which binds the low probability states (which are most probable fault states) with high costs whereas the states with high probability with low cost. The states get few particles but the cost miscalculating their probability is high. Particle filter sample from the product of risk function and original distribution.
V. Verma presents a further algorithm with the *Variable resolution Particle Filter* [21] [23]. It is based on the observation that some faults have similar symptoms so that they can be grouped together. If some fault from this group occurred the algorithm will breaks apart the group and diagnose the received states.
E. Benanzera [24] combines two approaches *Livingstone* and *look-ahead RaoBlackwellized filter* in aim to reduce computational complexity associated with particle filter technique and extend Livingstone approach to handle stochastic hybrid system.
The low a priory probability of fault states supplements challenges for detection algorithm. Plagermann [25] applies the *Gaussian process classification* and regression techniques for learning effective proposal distributions of particle filter.

4. 3. 3 Numerical Example

This section presents software that implements particle filtering for fault diagnosis in the four-wheel OMNI robot (the full version of this example is given in Appendix B). The objective is to estimate and illustrate the fault states of the robot. The groups of states are described in section 3. 2. Each group consists of one or several fault modes. For this example the number of faults has been reduced compare particle filter to the HMM example.

1. Normal operation **N** (no fault);
2. The group of abrupt motor faults consists of the four faults **W1, W2, W3** and **W4.** They correspond to the output value zero of wheel motors 1, 2, 3 and 4.
3. The stuck motor group is comprised only of the fault **M1s**. It describes the output of wheel motor 1 being fixed to a constant value regardless of the input.
4. Gradual degradation of performance includes only the fault **M1d.** It corresponds to the output of wheel motor 1 being multiplied with a negative exponential function.

A robot might need given measurements of robot velocities to automatically diagnose whether any of the faults occur. In this example the discrete state can only be one fault of the listed fault groups or the normal mode (no fault). Once the robot knows its discrete state it can generate a control action to solve its velocity problem.
The transition matrix of discrete states is:

```
par.T = [0.75 0.05 0.05 0.05 0.05 0.05 0;
        0.025 0.7 0.025 0 0.05 0.05 0.05;
        0.1 0.05 0.8 0.05 0 0 0;
        0.1 0 0.05 0.8 0.025 0 0;
        0.1 0.05 0 0.05 0.8 0 0;
        0.05 0.05 0.025 0.025 0.025 0.8 0.025;
        0.05 0.05 0.025 0.025 0.025 0.025 0.8];
```

The control action to solve the velocity problem is described as:

$$\begin{pmatrix} v_x \\ v_y \\ \theta \end{pmatrix} = K \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}, \qquad (4.\ 3.\ 2)$$

$$K=\begin{pmatrix} \sin\varphi & -\sin\varphi & -\sin\varphi & \sin\varphi \\ -\cos\varphi & -\cos\varphi & \cos\varphi & \cos\varphi \\ \frac{1}{4R} & \frac{1}{4R} & \frac{1}{4R} & \frac{1}{4R} \end{pmatrix} \qquad (4.\ 3.\ 3)$$

The matrix K is modified depending on the discrete state. If, for example, the system is in state W1 then the first column of the matrix K would be zero whereas the others columns stay unchanged. To implement the particle filter, I used the software package from Nando de Freitas, which uses classical particle filters and *Rao-Blackwellised particle filters* [83]. The software also includes efficient state-of-the-art resampling routines.

The script was executed in the following steps:

1. Initialisation of parameters
2. Generation of data
3. PF estimation
    a) Sequential importance sampling step
    b) Resampling step
2. Summery and plots

Initialisation of Parameters

Parameters such as the number of particles, time steps, transition matrix for a discrete state, four-wheel OMNI robot constants and the control matrix *K* are defined in this part of the numerical example.

Generation of Data

Compared to the numerical example in subsection 4. 2. 5 (HMM), in this example only the notation of *velObs(:,:)* is changed to *y(:,:)*.
The sensor measurements and input wheel velocities for each time step are saved in the *RobotPoseData.mat* file.

- *u(:,:) – 4xT* matrix of input data. To each time step a vector of robot motor velocities corresponds $(v_1, v_2, v_3, v_4)^T$. By collecting the vectors we receive the input matrix *u*
- *y(:,:) – 3xT* matrix of measurements. As measurements data we use linear and angular velocities of the robot.

PF Estimation

In this part of the script we attempt to track the current state and diagnose the faults.

In each time step *t = 1,…, T*.

- the important sampling procedure is performed:
    - for each particle *i = 1,…, N*
        - the new discrete state is generated from the previous one: *z(t)~p(z(t)|z(t-1))*
        - then the new continuous state is obtained from the currently generated discrete state and the previous continuous state: *x(t)~p(x(t)|z(t),x(t-1))*
    - for each particle *i = 1, …, N* the importance weights based on the observations $w(t) = p(y(t) \mid x(t))$ are evaluated
    - importance weights are normalised
- the particles with replacement N according to the importance weights are resampled.

<u>Sampling of the continuous state in detail</u>

To calculate the continuous state we use the formula (4.3.2). Matrix K is used to generate the control action (robot velocities), its view depending on the current discrete state. Sometimes we will modify the input vector *u(:,:)* in order to achieve the desired result for the robot velocities.

Abrupt motor fault at time step t

The diagnosis of faults from the second group W1, W2, W3 and W4 is pretty simple. We need to modify the columns of the matrix *K* accordingly to the wheel order number so that for the state W1 the first column is zero, for W2 the second column is zero, for W3 and W4 the third and fourth columns are zero.

Stuck wheel motor 1 (M1s)

To diagnosis the fault M1s we need to find out the stuck value of wheel 1. If the latest values of the wheel 1 velocities have the same value and it differs from the expected value, then M1s has occurred.

If, for example, input wheel vector *u(:,t)=[-2,-2,2,2]*, but wheel 1 of the robot gets stuck to 5m/c, then the true values of the robot's velocities could be derived from (4. 3. 2) by replacing the given vector *u(:,t)* with *new_u(:,t)=[5,-2,2,2]*.

We attempt to find the *new_u(:,t)*:

*new_u=[4*R*x_pf(3,t-1,i)-u(2,t-1)-u(3,t-1)-u(4,t-1);u(2,t);u(3,t);u(4,t)]*

where *x_pf(3,t-1,i)* is the robot angular velocity θ in time *t-1*.

Instead of the expression

*4*R*x_pf(3,t-1,i)-u(2,t-1)-u(3,t-1)-u(4,t-1)*

we can use

-x_pf(2,t-1,i)/cos φ *-u(2,t-1)-u(3,t-1)-u(4,t-1)* or

x_pf(1,t-1,i)/sin φ *-u(2,t-1)-u(3,t-1)-u(4,t-1)*.

"Gradual degradation" of wheel motor 1 (M1d)

To diagnose the M1d fault we need to find the decay rate and analyse its modification over time. The decay rate of the last two time steps reads:

*pred_val(1)=u(1,t-1)/(4*R*x_pf(3,t-1,i)-u(2,t-1)-u(3,t-1)-u(4,t-1));*
*pred_val(2)=u(1,t-2)/(4*R*x_pf(3,t-2,i)-u(2,t-2)-u(3,t-2)-u(4,t-2));*

Then the current value of the motor wheel 1 output can be estimated:

```
new_u=[u(1,t)/(pred_val(1)+abs(pred_val(1)-pred_val(2)));u(2,t);u(3,t);u(4,t)];
% compute robot velocities
v_pf(:,t,i) = par.K(:,:,1)* new_u;
```

Plot Results

In this paragraph various experimental results of the implementation of fault diagnosis using the classical particle filter are illustrated. Each plot has three axes: t – time, z_t – system states and $p(x_t \mid y_{1:t})$, where units of the z_t axis correspond to the following states: 1-Normal condition, 2-W1, 3-W2, 4-W3, 5-W4, 6-M1s, 7-M1d.

1. Diagnosis of the stuck motor 1

Given: N (number of particles) =200 and N (number of particles) =50, T(time)=10
Original behaviour: Two experiments run using 200 and 50 particles for the same fault scenario "motor 1 gets stuck". The motor 1 gets stuck from the second time step onwards. The initial distribution of particles is chosen randomly.

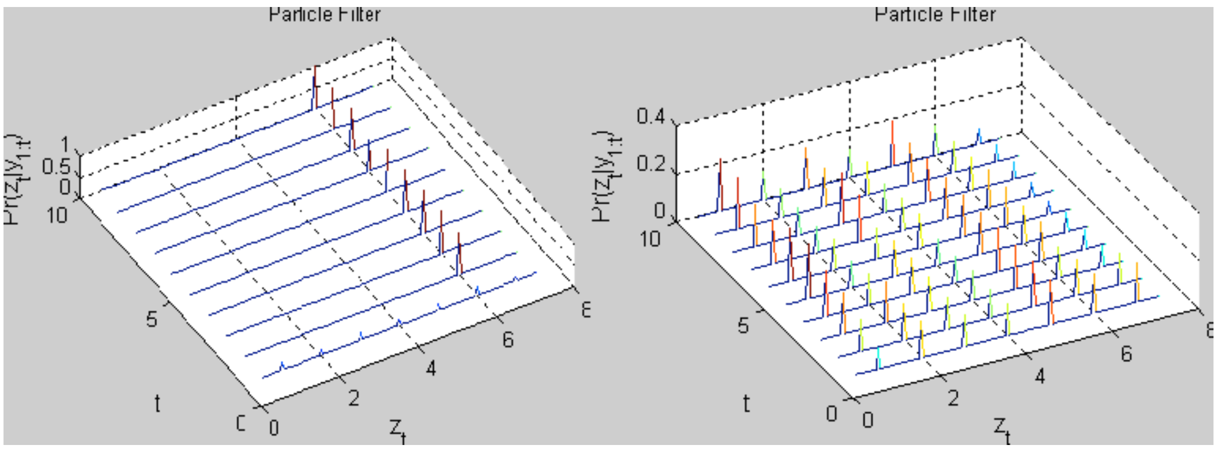

Figure 22 The estimated filtering distribution for "motor 1 gets stuck" scenario, left plot using 200 particles and right plot using 50 particles.

2. Diagnosis of the "gradual degradation" fault of wheel motor 1

Given: N =200 and T=10

Original behaviour: The "gradual degradation" fault of wheel motor 1 from the fourth time step onwards. The initial distribution of particles is chosen randomly.

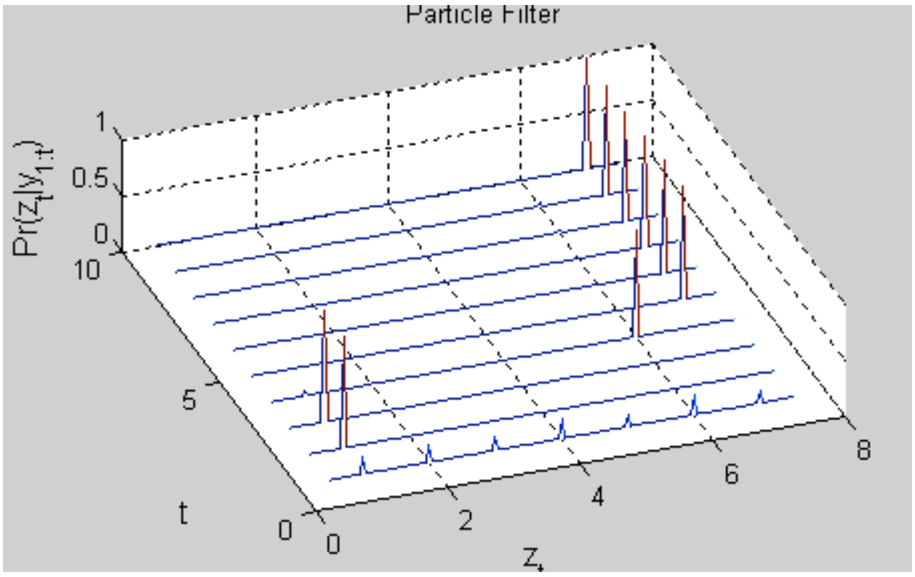

Figure 23 Estimated filtering distribution for "gradual degradation" of wheel motor 1.

3. Diagnosis of the stuck wheel motor 1 to zero value

Given: *N* =200 and *T*=30

Original behaviour: Wheel motor 1 stuck with the value "zero" from the twentieth time step onwards. The initial distribution of particles is chosen randomly.

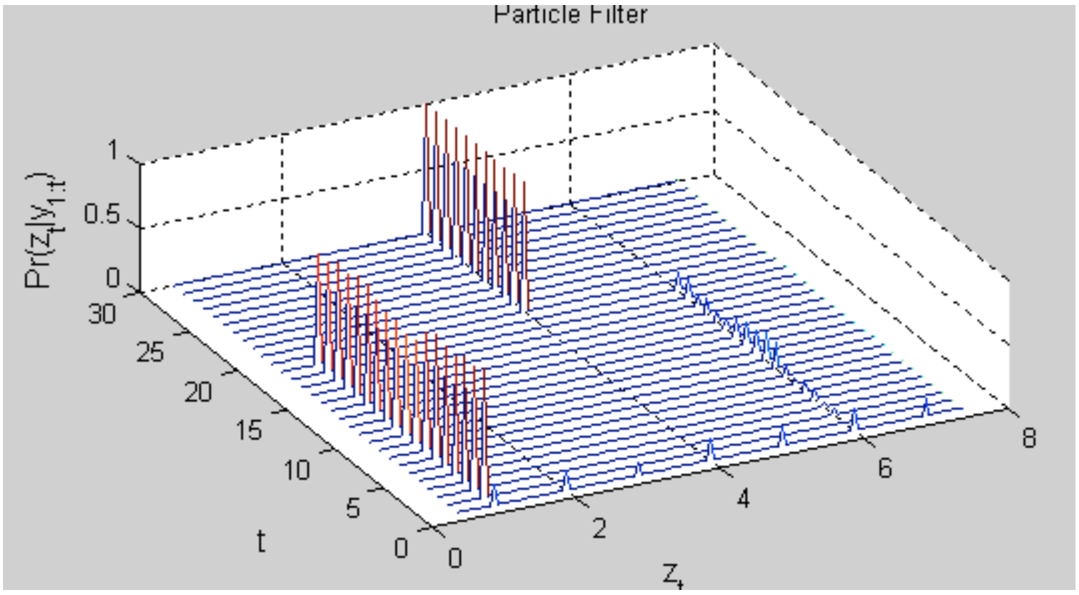

Figure 24 Estimated filtering distribution for the "motor 3 gets stuck with output zero" scenario.

Notice: The input vector *u(:,:)* in this experiment has the same values in the sequence from step1 to step 19. This condition influences the estimate of the "stuck motor 3" state.

During the experiments it occurred that particle filter produced unexpected results: all states had the same probability distribution value or were described as normal states in the M1d scenario, when the decay rate was calculated for the last six steps instead of only the last two.

In my opinion it happened because the decay value did not constantly decrease over time. I conclude that the particle filter needs knowledge about the noise process for the fault mode.

4. 3. 4 Summary

The hidden Markov models described in the previous section operate on discrete modes and use monitors to translate continuous variables into discrete values. The monitors only once determine consistent value from the measurements, and hence this system cannot generally diagnose a temporal event [17]. To overcome this problem we need a hybrid model. Particle filter is a technique for reasoning with hybrid models. The idea of particle filters is to represent the posterior density by a set of random particles with associated weights. The advantages of particle filter algorithms are that they support complex, non-linear, non-Gaussian models. It is an attractive algorithm for fault diagnosis for more than one reason: First, it can be applied to almost any probabilistic robot model that can be formulated as a Markov chain. Particle filter's computational time is independent of model complexity only for a certain number of particles. A developer can design an amount of particles to match the available computational resources. Finally, they are relatively easy to implement.

However, there are various problems with using particle filters for implementing in a fault diagnosis technique. The number of particles in improbable states (that are often faulty states) are few and the obvious solution of increasing the number of particles leads to increasing computational requirements. There are several approaches to address this problem in literature which improve fault detection while keeping the computational complexity low. The other drawback of particle filter is that the number of particles grows exponentially with state-space dimensionality. The significant disadvantage with particle filters for fault diagnosis is the need to know models for the state transition and noise process apriory,

The practical application of low-dimensional state-space with seven discrete modes and one continuous state shows sufficient diagnosis results for various experimental setups. The experiment proved that results become inaccurate for a quantity of particles below hundred (see Figure 22).

4. 4 Observable Operator Model (OOM)

The *Observable Operator Model* (*OOM*) is an alternative new approach to HMM. Its theory, developed by H. Jaeger of the international university Bremen, is expressed in terms of linear algebra. OOM looks almost like HMM: both can be expressed in matrix formalisms, although the matrixes and state vectors of OOMs may contain negative components, whereas the elements of HMM matrixes include only non-negative probability values.

4. 4. 1 Background Theory

This subsection gives a tutorial introduction to OOM by summarising and recapitulating the material of original works [5],[84],[85]. In our thesis we are only going to present the OOM theory for discrete time and discrete value processes. Non-stationary, continuous-time and arbitrary-valued processes are sketched in "*Characterizing distributions of stochastic processes by linear operators*" by H. Jaeger [68].

Abstract Form of OOM

To perform any task a robot needs to make predictions about the effects of its actions. In other words the robot should build a number of future trajectories and follow them to achieve a certain goal. Imagine for instance a robot in a room which needs to move from a door to a window. There are various possible paths which the robot could use to get to the window. The expectations about future trajectories depend not only on the location of the goal but also on the robot's current observations (everything with informational value for the expected future). While performing the task, the robot has to generate information about the future of the system based on its observations during each time step.

Based on this knowledge we want to introduce the basics of OOM. It is a mathematical model of constantly updating operations, where every possible observation is presented by one operator called *observable operator*. The key insight about OOM is *"(...) the observation that these observable operators are linear."* [5].

We are going to present an interpretation of OOM a discrete time, finite-value and stationary stochastic process.

Let $(X_n)_{n\in N}$ be such a process with values in a finite set $O=\{a^1,...,a^\alpha\}$ of possible observations. Consider a set O^* that denotes the set of all finite strings over O including the empty string.

For every $\bar{a}\in O^*$ (where $\bar{a}$ is a sequence of $a_0...a_r$), we define a real-valued function

$$f_{\bar{a}} : O^* \rightarrow R \tag{4. 4. 1}$$

$f_{\bar{a}}$ is a *prediction function* of the process that describes the future distribution of the process after an initial observation $\bar{a}$. In our robot illustration $\bar{a}$ would correspond to the robot's path that it had in short-time till the current position, and $f_{\bar{a}}$ would correspond to the distribution of future trajectories started at that moment.

F is the space of future distributions of the process (X_n), namely *vector space*.

t_a is a linear *observable operator* for every $a\in O$ $t_a : F \rightarrow F$ by

$$t_a(f_{\bar{a}_t}) = P(a\,|\,\bar{a})f_{\bar{a}_t\,a} \tag{4. 4. 2}$$

$\bar{a}\,a$ is concatenation of sequence $\bar{a}$ with a

$P(a\,|\,\bar{a})$ is the short form of $P(a\,|\,a_0...a_{s-1})$ or as full formulation

$P(X_{n+1} = a\,|\,X_{n-s} = a0,...,X_{n+s} = a_s)$

This leads to the following definition:

"Let $(X_n)_{n\in N}$ be a stationary stochastic process with values in a finite set O. The structure $(F,(t_a)_{a\in O},f_\varepsilon)$ is called the observable operator model of the process. The vectors $f_{\bar{a}}$ are called states of the process; the state f_ε is called the initial state. The vector space dimension F is called the dimension of the process." [5]

Matrix OOM

Section 4. 2 introduces HMM techniques for analysing discrete-time, discrete-state and stochastic process $(Y_n)_{n \in N}$. The outcomes of the random variables (Y_n) are given in set $O = \{a^1,...,a^\alpha\}$. A Markov chain $(X_n)_{n \in N}$ produces a sequence of hidden states from the set $\{s_1,...,s_m\}$. Now we will show how HMM can be generalised to serve as a basis for creating OOM.

Assume we have a hidden Markov model with the parameters
- *m x m* stochastic matrix M collecting state transition probabilities
- set of *m x m* observation matrixes O_a for every $a \in O$, each O_a consisting of elements with the value zero except for the diagonal elements which are the observation probabilities $P(Y = a \mid X = s_j)$
- initial distribution $w_0 = (P(X_0 = s_1),..., P(X_0 = s_m))^T$

The matrixes *M*, O_a and w_0 are used to compute the probability of finite observation sequences.
Let 1 = (1, . . . , 1) is the *m*-dimensional row vector of units, and let $T_a = M^T O_a$.
Then the probability to have the sequence $a_0...a_r$ is

$$P(a_0...a_r) = 1T_{a_r}...T_{a_0 w_0} \qquad (4.\ 4.\ 3)$$

This is a matrix representation of the forward algorithm for determining probabilities of observation sequences in HMMs (see section 4. 2. 3). It shows that the distribution of the process (Y_n) is specified by the operators T_a and the initial vector w_0.
At this point we can derive the matrix definition of a finite-dimensional OOM by first relaxing the requirement that M^T includes only non-negative elements to the weaker requirements a) that the sum of elements of each column of M^T is 1, and b) that the sum of the w_0 component is 1, meaning that negative entries are allowed. The symbol τ in OOMs stands in the places, where T appears in HMMs. Now we can get the matrix definition of OOM [5]:

“An m-dimensional (matrix) OOM is a triple $A = (R^m, (\tau_a)_{a\in O}, w_0)$ where $w_0 \in R^m$ and $\tau_a : R^m \to R^m$ are linear maps represented by matrixes, satisfying three conditions:

1. $1w_0 = 1$,
2. $\mu = \sum_{a\in O} \tau_a$ has column sums equal to 1
3. for all sequences $a_0 \ldots a_r$ it is holds that $1\tau_{a_r} \ldots \tau_{a_0} w_0 \geq 0$.”

For more details and numerical examples about the generation of OOM from HMM, see [69].

Condition 1 and 2 were mentioned in relaxation a) and b) while the condition 3 ensures that calculated probabilities obtain non-negativity values. Note that for the given operator $(\tau_a)_{a\in O}$ no known way exists to decide whether the condition 3 holds true [5].

If $\tau_{\bar{a}}$ is concatenations of operators $\tau_{a_r} \ldots \tau_{a_0}$ then we can compute the probabilities of a finite-length sequence by

$$P_0(\bar{a}) = 1\tau_{\bar{a}} w_0 \tag{4. 4. 4}$$

In this section we have described OOM as the matrix structure $(R^m, (\tau_a)_{a\in O}, w_0)$. In the previous section we have discovered the abstract OOM structure $(F, (t_a)_{a\in O}, f_\varepsilon)$.

The two structures are related via dimension of process and dimension of a OOM matrix. If a process has the dimension *m*, then a concrete matrix OOM of the matrix dimension *m* exists. A matrix *m*-dimensional OOM specifies a process with the dimension *k*, $k \leq m$. An *m*-dimensional process has no matrix OOM with a dimension smaller than *m*.

Thus, if a process has the dimension *m*, and we have a *k*-dimensional OOM *A* describing this process, then an *m*-dimensional OOM *A'* exists which is equivalent to *A*. Furthermore, *A'* is minimal-dimensional in its equivalence class. A minimal-dimensional OOM *A'* can be constructively obtained from *A* in several ways which are described in [5].

Generation Procedure

To solve the diagnosis problem (determine the current system state) we describe techniques of how to generate *state vector* $w_{\bar{a}}$ of OOM $A = (R^m, (\tau_a)_{a \in O}, w_0)$ after history $\bar{a}$ has been observed.

The entire generation procedure is executed as follows:

1. Define initial state vector as $w = w_0$
2. Choose next observation a_n

   a_n is the observation at time *n* after $a_0, \ldots, a_{n-1}$ have already been produced.

   At time *n=0*, the probability of producing a is $P(X_0 = a\ )$. To generate the symbol a_0 with the correct distribution we need to consider the probabilities for each observation $a \in O$ and then to choose one with the highest value. For this we introduce probability vector $p_0 = (P(X_0 = a^1) \ldots p(X_0 = a^\alpha))^T$. This is done by calculating $P(X_0 = a) = 1\tau_a w_0$ using (4. 4. 4) for all $a \in O$. A faster way to do this is to calculate the row vector $1\tau_a$ for all *a*, and collect them in the matrix

$$\sum = \begin{bmatrix} 1\tau_{a^1} \\ \cdot \\ \cdot \\ \cdot \\ 1\tau_{a^\alpha} \end{bmatrix} \qquad (4.\ 4.\ 5)$$

and derive

$$p_0 = \sum w_0 \, . \qquad (4.\ 4.\ 6)$$

At every time step *n>0* the observation a_n can be chosen according to the probability vector $p = (P(a^1 \mid \bar{a}) \ldots P(a^\alpha \mid \bar{a}))^T = \sum w_{\bar{a}}$.

3. Having the observation a_n we can take the corresponding operator τ_{a_n} and update the state vector by $w_{n+1} = \tau_{a_n} w_n \big/ 1\tau_{a_n} w_n$ (for details see [5]) and continue at step 2.

HMMs and OOMs

The conceptual difference between the representation of HMM and OOM lies in the display of their theories in the stochastic system. HMM views stochastic systems as trajectories in a state-space, where observations are locations in that state-space, while OOM understands trajectories as a sequence of (linear) operations. Each observation corresponds to the sequence of operations built on the previous observation.

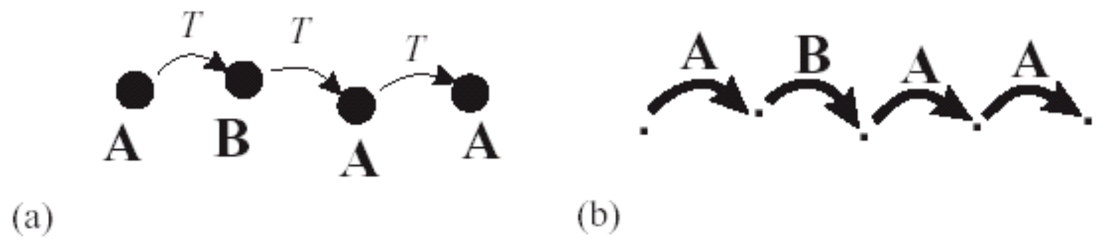

Figure 25 (a) The standard view of trajectories. A time step operator T yields a sequence ABAA of states. (b) The OOM view. Operators A and B are concatenated to yield a sequence of observations. [69]

OOM and HMM have different understandings of their states. HMM states denote the set of physical states of the target system. By contrast, OOM states represent the expectation about the system's future and the observable development provided by an observed past.

OOMs are more general than HMMs since OOM can express every HMM, but HMM can not express every OOM.

Learning Algorithm

The learning algorithm for OOM estimates linear operators from a sequence of observations. Before presenting the basic OOM learning algorithm, we quickly provide an overview the most important properties of OOM.

Model equivalence

The central theorem of the OOM theory is about the equivalences (describing the same stochastic process) of two minimal-dimensional OOMs:

"Two minimal-dimensional OOMs $A = (R^m, (\tau_a)_{a \in O}, w_0)$ and $A' = (R^m, (\tau'_a)_{a \in O}, w'_0)$ are equivalent if and only if there exist an bijective linear map $\rho : R^m \to R^m$, satisfying the following conditions:

1. $\rho(w_0) = w_0^{'}$,
2. $\tau_a^{'} = \rho \tau_a \rho^{-1}$ for all $a \in O$,
3. $1w = 1\rho w$ for all $w \in R^m$."[5]

A matrix ρ satisfies condition 3 only if each column of ρ sums up to one. Having one minimal-dimensional OOM *A*, we can derive the other equivalent OOMs by applying any transformation matrix ρ with the columns' sum = 1.

Indicative and characteristic events

The key concepts of the OOM learning algorithm are based on indicative and characteristic events. These events are received from dividing the process trajectories into past and future.

We have already defined set O^* as a collection of all strings with elements from set *O*. O^k is the set of strings with length $k O^k = A_1 \cup ... \cup A_m$. If for some sequences $\bar{b}_1, ..., \bar{b}_m$ a non-singular *m x m* matrix with elements $(P[A_i \mid \bar{b}_j])_{i,j}$ (where $P[A_i \mid \bar{b}_j]$ denotes $\sum_{\bar{a} \in A_i} P[\bar{a} \mid \bar{b}_j]$) exists, then A_i (*i=1,...,m*) is a set of *characteristic events.*

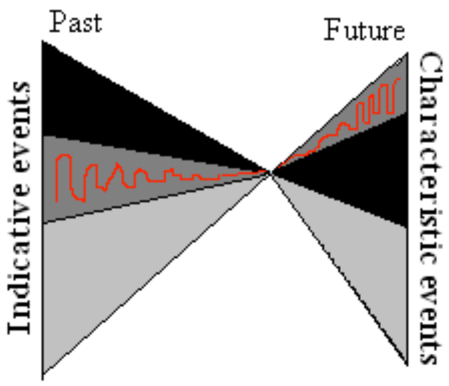

Figure 26 Storing out a process realisation into indicative and characteristic events [70]

Interpretability

Based on the theory of characteristic events we can estimate the probability of OOM producing a certain characteristic event when it is started in state w. We assume that for a m-dimensional process with a set of visible possible observations O the characteristic events $B_1,\ldots,B_m$ and OOM $A=(R^m,(\tau_a)_{a\in O},w_0)$ exist. A is interpretable with respect to $B_1,\ldots,B_m$, if the states w of A have the property $w=(P(B_1\mid w)\ldots P(B_m\mid w))^T$ [5]. The idea behind interpretable OOM is to take the next event probabilities from the current state vector. An important property of the interpretability summed up by H. Jaeger reads:

"*In an OOM that is interpretable with respect to* $B_1,\ldots,B_m$ *it holds that*

1. $w_0=(P(B_1)\ldots P(B_m))^T$,

2. $\tau_{\bar{a}}w_0=(P(\bar{a}B_1)\ldots P(\bar{a}B_m))^T$," [5]

This property allows us to estimate the state vector w from the sequence of observations using frequency counts. With the estimated state vectors we construct the operators using linear algebra. This property is the key to the computational advantage associated with OOMs.

OOM basic learning algorithm

It is often a problem in the robot domain that a model of a specific robot system is not available, only a sequence of observations $S=\{a_0a_1\ldots a_N\}$, produced by some hidden process. Learning OOM is a technique for estimating or computing a model from the given sequence. In this part of the paragraph the fundamental steps of the learning algorithm and a simple example of its implementation are presented. A detailed description can be found in [5].

Step 1	Design model dimension m and length of characteristic event k
Step 2	Choose characteristic events $B_1,\ldots,B_m$ and indicative sequences $\bar{a}_1,\ldots,\bar{a}_m$ so that matrix $V^{\#} = (P_S^{\#}(\bar{a}_j\, B_i))_{i,j=1,\ldots,m}$ is non-singular, where $P_S^{\#}(\bar{a}\,B_i) = \dfrac{frequency\quad of\ \bar{a}\,\bar{b}(where\,\bar{b}\ \in B_i)\,within\quad S}{N-\lvert \bar{a}\,B_i \rvert +1}$
Step 3	Compute matrix $W_a^{\#} = (P_S^{\#}(\bar{a}_j\, aB_i))_{i,j=1,\ldots,m}$ for every $a \in O$
Step 4	Obtain $\tilde{\tau}_a = W_a^{\#}(V^{\#})^{-1}$.

A simple example can be found in the lecture H. Jaeger (Discrete-time, discrete-valued observable operator models: a tutorial) [reference]

4. 4. 2 Learning with OOMs: Challenges and Their Solutions

The basic OOM learning algorithm does not have a local minima problem from which the EM algorithm (see subsection 4. 2. 3) suffers and it is computationally cheap. Besides these advantages it has the following drawbacks:

1. The statistical efficiency (model variance) depends on the choice of indicative and characteristic events. For the infinite sequence, the selection of indicative and characteristic events may be randomly chosen to estimate a correct model operator. With finite training data however, the selection of indicative and characteristic events is difficult. Several methods are determined to overcome the problem. One of them is the extension of the basic learning algorithm by the *efficiency sharpening* (ES) method, which solves the problem by only using these events for the estimation of an initial model A_0. The better models $A_1, A_2, \ldots$ are iteratively obtained from A_0 without using such events at all [5].

2. As there are only heuristic solutions to the problem that the OOM learning algorithm needs to know the "correct" model dimension in advance, we have to define criteria for choosing the dimensionality. The dimension m should be chosen large enough, because the model needs to capture all the properties of the training sequence distribution, and small enough to prevent overfitting.

3. Even with efficient characteristic and indicative events, the basic OOM learning algorithm has limited statistical efficiency. Since only the substrings of some determined length are considered in the learning algorithm, the other information contained in the training data is ignored. This problem can be solved by using a *suffix tree* [5] to represent the state sequence.

4. The most critical issue with the OOM learning algorithm is the negative probability problem (NPP). It is unknown whether an OOM-like system is indeed a valid OOM or not. Thus the learning algorithms of OOMs can obtain invalid models which assign negative numbers to probabilities of some (rare) events instead of small positive numbers. The problem is still not solved. H. Jaeger describes an unexplainable trick to overcome it. The idea is to transform the reverse $A'^{r(n)}$ (ES-method) matrix (before using it) into a valid OOM by inserting all negative elements in the operator matrixes of $A'^{r(n)}$, set them to zero and renormalize their columns [5]. Another solution of this problem is using an alternative version of OOM – norm-OOM (see the next subsection).

4. 4. 3 OOM Flavours

Input-output observable operator models (IO-OOMs) [86] are extensions of the basic OOM theory with added input data to control the system output.

The IO-OOM is defined as set of observable operators represented by matrixes of real-valued elements and an initial state vector. Note that for every fixed input the structure is just an ordinary OOM (Figure 27). So an IO-OOM is a set of classical OOMs, one for each possible input, where the given input switches between these OOMs. All incoming OOMs share the same state-space. The detailed description of IO-OOM theory and examples is presented in [86].

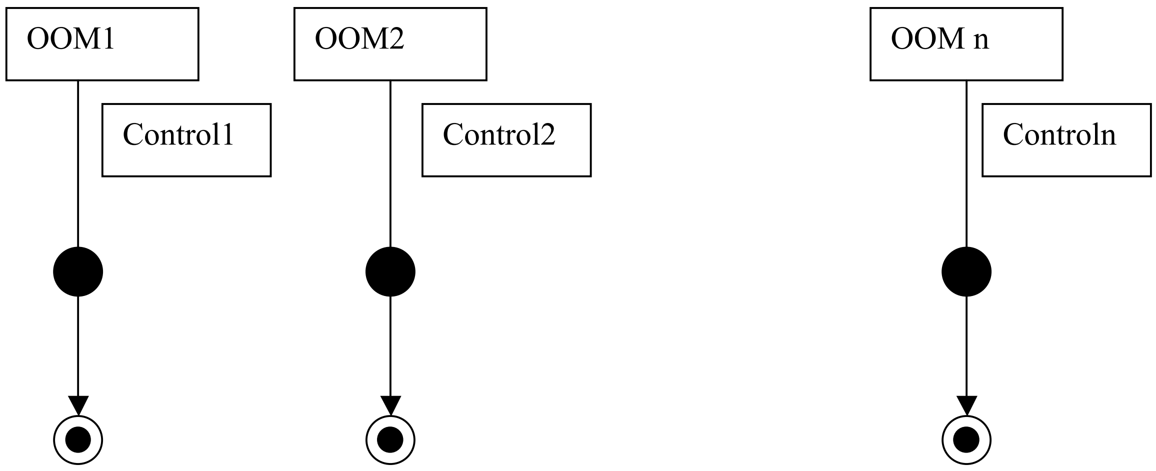

Figure 27 IO-OOM structure [OOM slides]

As already mentioned, the critical issue of OOM is the negativity probability problem, which remains unsolved in OOM theory. To avoid NPP, M. Zhao and H. Jaeger in one of theirs latest works [86] introduce another similar model class, *norm observable operator models* (norm-OOM). The idea of norm-OOMs is the following: in the way an OOM model stochastic process can be extended for describing numerical functions, so can NPP be avoided by applying a nonnegative function on the state vectors of OOM.
In particular, for norm-OOMs, we can compute the probability of an initial sequence $\bar{a} \in O^*$ by $P(\bar{a}) = \left\| \tau_{\bar{a}} w_0 \right\|^2$, where ||.|| is Euclidian norm. Although the NPP-problem is solved, norm-OOM still suffers from the complexity of the theory and calculation problems.

4. 4. 4 Numerical examples

In this section practical implementation of the OOM theory will be conducted. There are two main tasks: checking the efficiency of the OOM probability estimation algorithm and comparing the model learning techniques of OOM and HMM. The first example illustrates the ability of the OOM generation procedure to estimate the system behaviour for the given model. The objective of this implementation is to show how it can be applied in fault diagnosis.
The second example presents the experimental study of learning algorithms for two different schemes of modelling dynamic systems without control: HMM and OOM. For

the OOM representation, we use the Matlab package developed by H. Jaeger [4]. To evaluate how well an HMM can learn the data, K. Murphy's Matlab toolbox is available [87].

Example for State Estimation

For demonstration purposes we apply the prediction algorithm to the four-wheel OMNI robot described in section 3. 2. The values for the transition matrix and the observation matrix are obtained from the original matrixes given in the HMM example, subsection 4. 2. 5, only with a reduced number of fault states (only four faults were considered: wheel 1, 2, 3 and 4 broken).

Transition matrix:

```
markovMat = [0.8 0.05 0.05 0.05 0.05;
             0.1 0.75 0.05 0.05 0.05;
             0.1 0.05 0.75 0.05 0.05;
             0.1 0.05 0.05 0.75 0.05;
             0.1 0.05 0.05 0.05 0.75];
```

Observation matrix:

```
obsmat = [0.8 0.05 0.05 0.05 0.05;
          0.1 0.75 0.05 0.05 0.05;
          0.1 0.05 0.75 0.05 0.05;
          0.1 0.05 0.05 0.75 0.05;
          0.1 0.05 0.05 0.05 0.75];
```

Jaeger’s Matlab package uses an HMM representation to create an interpretable OOM from it. To test the OOM diagnosis algorithm we assume a sequence of states which reflects normal and fault states. The sequence was generated by hand. According to the sequence the output data can be simulated. Commanding sensor measurements and a probabilistic model, we can apply the diagnosis algorithm to estimate system behaviour and compare it with the given real behaviour.

We assume four various robot behaviour scenarios: the robot starts in the normal state and at some moment in time one of its wheels breaks. For the first scenario it is wheel 1, for the second wheel 2, etc. The duration of each scenario was divided into ten moments and diagnosis was performed for each one. For the first three recorded moments the behaviour is fixed to be normal, for all following moments it is defined that the corresponding faults take place. The algorithm results are depicted in Figure 28.

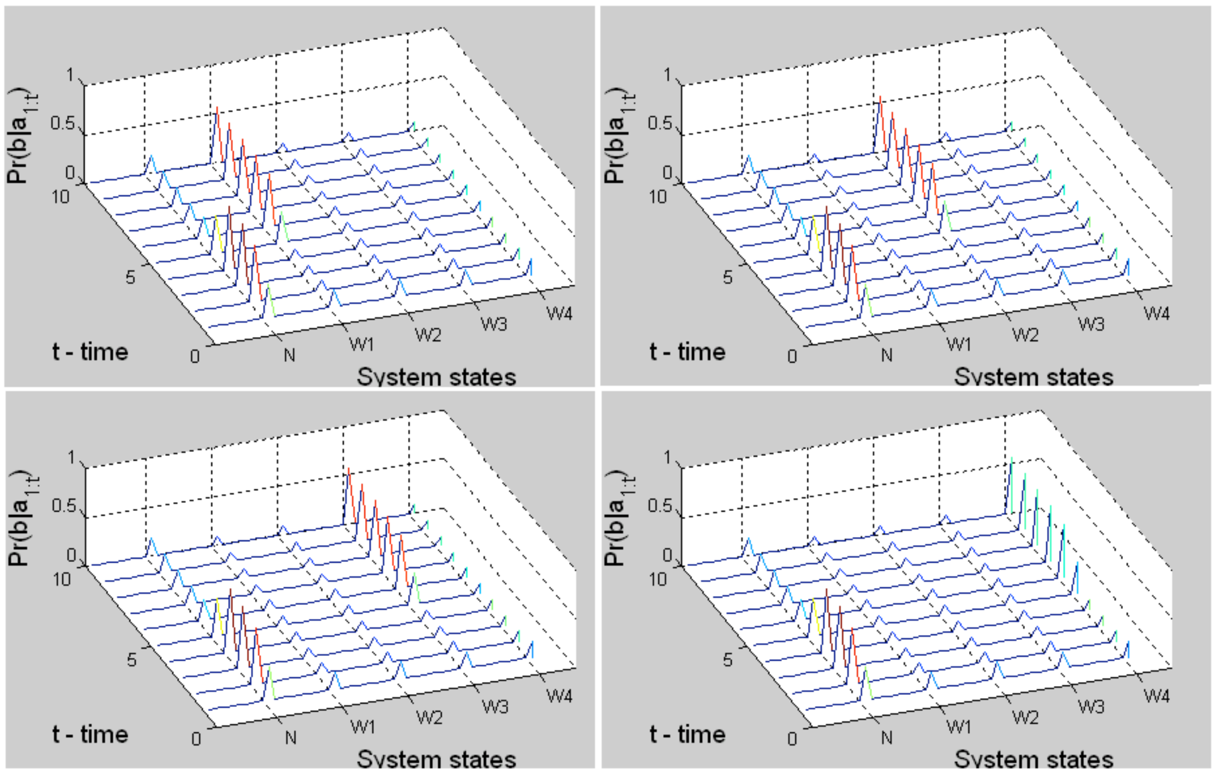

Figure 28 Estimated probability distribution using OOM

In all cases we can see that faults were detected with a high probability value. It lies between 0.5 and 0.6 although in the fifth (transient moment) it is hard to draw conclusions about the system state since both normal and fault states have a similar probability value. The full version of the example is given in Appendix C.

After this simple example we can conclude that the OOM algorithm achieves good estimation results for a well-defined model. So the second task will be to test how well the OOM learning algorithm can estimate a model of the robot from training data.

Implementation and Comparison of HMM and OOM Learning Algorithms

In this section the performance of OOM and HMM learning algorithms on the same dataset are checked. The aim of the example is to test the HMM and OOM learning algorithms, to compare their output for 1.000 states and to check the OOM validation for 100.000 states.

The code of this example is a slight update of *tutorialDemo.m* file from the original package [88].

We will first consider the main steps of the exemplary application and then provide the details. Finally the experimental result will be explained.

Steps of the example:

1. Generate training dataset S
2. Design parameters:
   a) Model dimension,
   b) Block length
3. Application of OOM learning algorithm
   a) Learn model from dataset S with OOM algorithm
   b) Draw results for 1000 generated states of OOM
4. Application of HMM learning algorithm
   a) Learn model from dataset S with EM algorithm
   b) Draw results for 1000 generates states of HMM
5. Draw results for 100.000 generates states of OOM

<u>1. Generate training dataset S</u>

In order to apply learning algorithms we need the training dataset. Since neither a real robot nor simulators were available to receive a training sequence, it has to be generated from a defined model. In our case an OOM was created from an HMM. Design parameters are the number of states *stateNr* (OOM dimension) and the number of observations *obsNr*.

Transition matrix:

```
markovMat = [0.8 0.1 0.1;
             0.1 0.8 0.1;
             0.1 0.1 0.8];
```

Observation matrix :

```
obsmat = [0.8 0.1 0.1;
          0.1 0.8 0.1;
          0.1 0.1 0.8];
```

The values for transition matrix and observation matrix are obtained from the original matrixes given in the subsection 4. 2. 5 with a reduced number of fault states and normalized matrix rows. The OOM package supports only a three-dimensional model and no more than three observations.

2. Design parameters

Based on these matrixes we create an OOM for use as a training data generator. The package provides two adjustable parameters: the dimension M of the vector space and the length L of the sequences to be sampled from the data.

3. Application of OOM learning algorithm

learnOOM (trainData, modelDimension, sampleBlockLength) is an implementation of the learning algorithm (see Appendix D). It learns an OOM model from training data *trainData*, which must be a single sequence. The dimension of the model is set by the *modelDimension* parameter and is learnt by sampling from *trainData* statistics of subsequences of length *sampleBlockLength*.

To present the results of the learning algorithm we generate a state sequence of length 1000 for the generator OOM and plot it as a set of points with different colours. Actually we will use an interpretable version of the generated OOM.

The generated OOM can be transformed into many different equivalent, interpretable OOMs depending on the choice of characteristic events. As mentioned by H. Jaeger [5], the interpretability enables us to visualize the state dynamics of an OOM. The three dimensions of OOM, its interpretable states being probability vectors, are non-negative and thus lie in the intersection of the positive orthant of R^3 with the hyperplane $H = \{x \in R^3 \mid 1x = 1\}$. This intersection is a triangular surface, its corners marking the three unit vectors of R^3 [5].

Figure 30 depicts three plots of states obtained from generating runs of three 3-dimensional OOMs over an observation alphabet of size 3, which was made interpretable with reference to the same characteristic events as the original generator. Note that if this plot shows points outside the triangular area, the model is not a valid OOM in that it would predict negative probabilities for some events in some states. In the appendix B function *plotStates3DColored* the algorithm of state evaluation is conducted.

4. Application of HMM learning algorithm

The same probabilistic data is used to receive a HMM model trained by the EM algorithm. A similar graphical representation of states for HMM [5] is applied to plot states.

5. Draw results for 100.000 generated states of OOM

It has already been mentioned that the plot depicts triangle and lying insight of it system states if one or several states outside the model are invalid. Even if all 1000 states are situated in the triangle of OOM, we can not be sure whether the model is valid or not. For this proposal the number of states is enlarged to 100.000 and plotted again.

Plot Results

To evaluate the generated hidden Markov model and observable operator model and compare them with each other, their plots can be used. The plot of original model depicted in Figure 29.

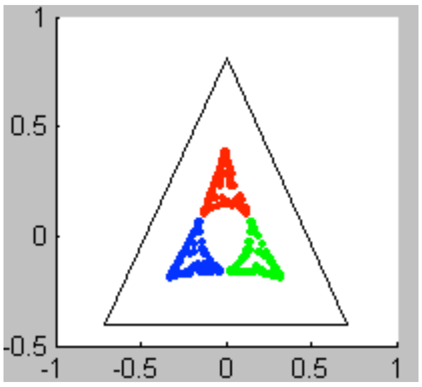

Figure 29 State sequence of length 1.000 of the original model

The points on the plot correspond to the states of the model. There are three kinds of states each one with its own colour. It is not important to know which colour corresponds to which kind of state. The goal is to show invalid OOM or HMM which stuck to local minima.

OOM includes a set of linear operators and an initial system state. To generate the other states we randomly choose an observation, select the corresponding operator and apply the operator to current state – the result is the next state. The information about generated states and corresponding observations is saved. By repeating the process 1.000 times, we obtain the set of states with corresponding observations. Based on this information the plot in Figure 30 can be created.

Efficiently estimated models:

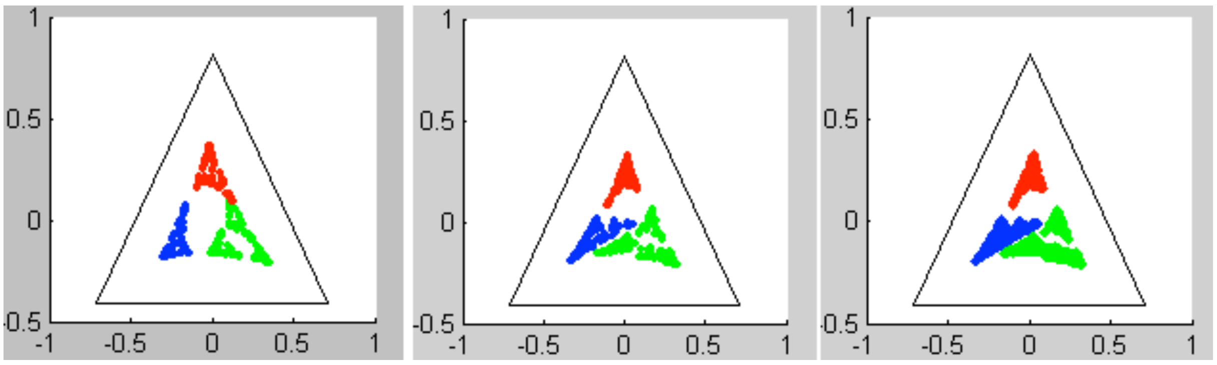

Figure 30 HMM and OOM estimated models

The first plot shows an estimated HMM with 1.000 points (states), the second plot presents an estimated OOM with 1.000 states and the third plot corresponds to an OOM with 100.000 states

By applying two different learning schemes HMM and OOM to the given dataset we receive learning models. To plot them, the technique described above is applied. If the generated model is perfect its plot should be identical to the original one that is depicted in figure 29. Plots in Figure 30 introduce well learned models for HMM and OOM accordingly.

Bad HMM (stuck in local minima)

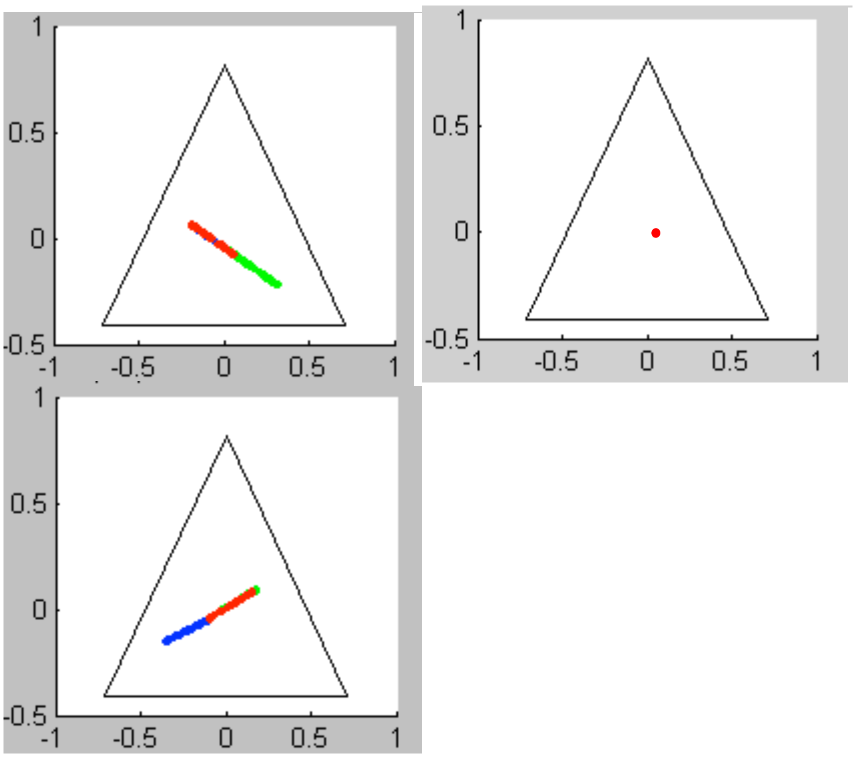

Figure 31 Variants of bad HMM

During the experimental performance the generated learned HMMs got stuck in bad local minima. The generation of bad models depends on a training dataset. In several cases we had a training sequence of length 1.000 on which EM always got stuck in a process model that had no memory at all (dim = 1, see second plot in Figure 31).

Invalid OOM

Not only the learning HMM algorithm shows unwelcome results, the learning OOM method generates an invalid model too. As described in subsection 4. 4. 2, the critical problem of the OOM learning algorithm is the negative probability problem when the generated states include negative values. Graphically this problem is visualised by the points lying outside of the triangular area.

a) *The invalid OOM generates states with negative probabilities on both cases for the sequences of length 1000 and 100000.*

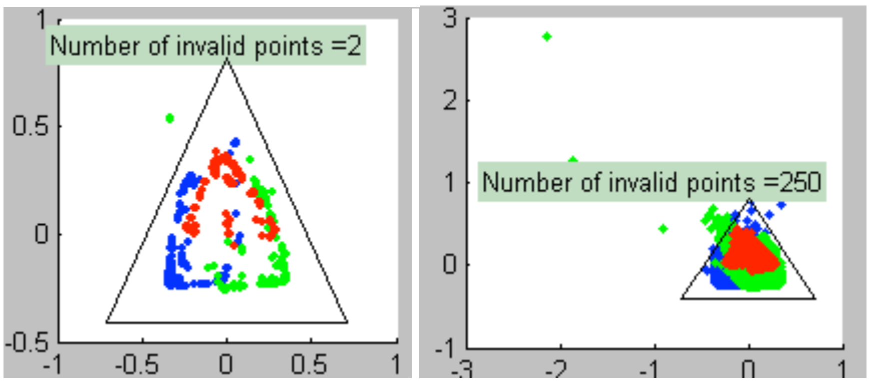

Figure 32 States distribution generated by invalid OOM.

The first plot presents the 1.000 points (states) generated by the invalid OOM, the second plot depict 100.000 points of the same OOM.

Figure 32 introduces the state distribution of invalid OOM model. In plot a) only two states with negative probability appear. The generated state sequence with 100.000 entries already presents 250 invalid states for the same OOM. Note that the amount of wrong states does not influence the model quality. Even if one million states would be generated and only one comprises negative values the model would be invalid.

b) *The learned OOM generates states with normal entries for the sequence of length 1.000 but states with negative probabilities with sequence of length 100.000.*

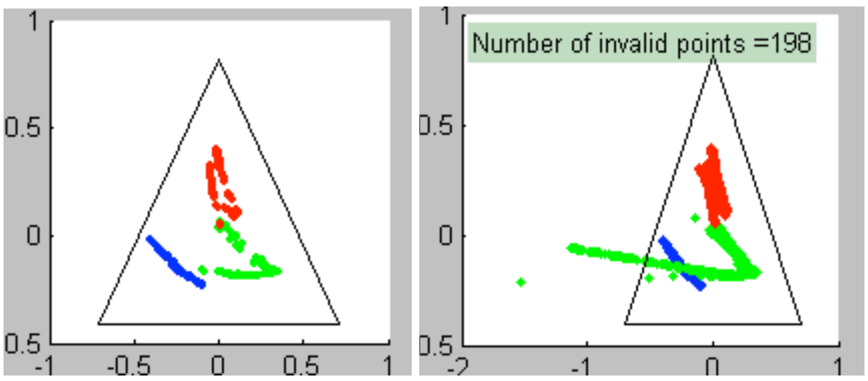

Figure 33 States distribution generated by invalid OOM

The first plot shows 1.000 points (states) generated by the invalid OOM, the second plot shows 100.000 points of the same OOM.

Unfortunately we can not be sure about the accuracy of the learned model even if it has only valid states, invalid states could always occur in the future. Figure 33 depicts this case: The model seems to be valid in the beginning, but when the sequence is enlarged the model becomes invalid.

Statistical results

In the example described above the OOM learning algorithm and the EM algorithm for HMM are applied to the generated training data. After repeating the experiment a thousand times we analysed the received plots for learned HMM and OOM state distribution. Based on the analysis the following conclusions about learning algorithm efficiency could be drawn:

- In 553 of 1.000 cases the EM algorithm for HMM got stuck to local minima (Figure 31). Moreover, in thirty of the 553 ”stuck” cases the process model had dimension 1 (see second plot in Figure 31).
- The learned OOMs achieved better results, as only in 215 of 1.000 cases the models were invalid (Figure 32). But this number rose to 331 when 100.000 states were checked.

The results of these experiments prove that the hidden Markov models were not able to learn the given training dataset in a satisfactory manner. This lack in the HMM learning algorithm is due to the fact that the EM algorithm reaches a local minimum and is unable to move away from it. The OOM learning algorithm on the other hand does not have a local minima problem and always generates a sufficient model. But in spite of these facts it is not really an alternative since it suffers from invalid models which produce states with negative values. Although the OOM algorithm possesses invalid models the total experimental result shows that it performs better, at least as far as this data set was concerned.

4. 4. 5 Summary

In this section we have established the basic theory of observable operator models (OOM) and compared the OOM learning algorithm with the EM algorithm for the hidden Markov model (HMM).

OOM is a recently developed class of models to describe a linear decision process. It is presented as a mathematical model of linear operators, which updates the future expectations based on observations. OOM sets the observable events a of a process to the linear observable operator τ_a acting on a real-value vector space of system states w (probability distributions). OOM is an alternative, more general approach to the hidden Markov model. Its theory is expressed in terms of linear algebra and its learning algorithm ES estimates models more accurate.

The OOM learning algorithm has a crucial remaining unsolved problem - the negative probability problem (NPP), since no algebraic criterion is available to control whether an OOM-like system is a valid model or not. Using norm observable operator models (norm-OOMs) allows avoiding the NPP. But this approach is in its infancy stage and many questions are still to be answered.

Based on the statistical data of the experimental results, we conclude that the OOM algorithm achieves a more accurate model than the HMM method for the given dataset, but the NPP problem is a great obstacle for implementing OOM in practical applications.

5. Comparison of Solutions

The investigations about the various fault diagnosis approaches are summarized in a comparison table, which shows assumptions strengths and weaknesses for each algorithm.

		Parity Space	**Hidden Markov Model**	**Particle Filter**	**Observable Operator Model**	
Modelling of Data		Linear Gaussian State-space model	Partially observed finite state-space Markov chain	Markovian nonlinear, non-Gaussian state-space model	Exhaustive collection of probabilities of every possible observation sequence describing discrete time discrete value stationary process.	
Modelling of Faults		Additive	Fault modes (fault causes process to behave according to fault model)	Fault modes	Fault modes	
Advantages		Simplicity	- Established both in theoretical and practical application fields - Comfortable structure (hidden states and observations) - Well interpreted hidden states in term of application	- Does not require fixed computation time, since it depends on the number of particles - Non-Gaussian distributions - Non-linear state and observation model - Mixtures of discrete and continuous states	- Model class richer than for HMM - Theory expressed in terms of linear algebra	
Disadvantages		- Good model representation obligatory - Sensitivity measurement errors and state noise - Linear model - Gaussian noise	- Supports only discrete system states	Computational power needed	Lack of practical applications	
Learning Algorithms		Principle Component Analysis (PCA)	Expectation Maximization (EM) Algorithm	-	Basic Learning Algorithm	Efficiency Sharpening (ES) Algorithm
	Advan-tages	Few parameters to tune: window size L and number of components	Wide range of applications	-	More effective than learning algorithm of HMM	- Generates more accurate model than HMM EM-algorithm - Needs less runtime than HMM algorithm
	Disadvan-tages	- Restricted to linear model - Order of model needs to be decided	Gets trapped in local maxima		- Invalid generated model (NPP) - Statistical inefficiency	Invalid generated model (NPP)

Table 4 Comparison of Algorithms

For a linear state-space model with additive faults, analytical results can be derived by the parity space approach. If a model is unknown but known to be linear, the principle component analysis can be used. The method has too many restrictions to be applied in fault diagnosis

If the process describing system behaviour is a Bayesian model with Markov assumptions then the data can be modelled as hidden Markov process. The HMM filter solves diagnosis problems for discrete-state finite HMM. It is one of the most general and widely-used filters in practice. It has a well established structure to represent a diagnosis model of faulty and normal behaviours and is therefore simple to implement. The HMM training algorithm (EM) is not completely satisfactory due to slow convergence and the presence of many local solutions (local maxima problem).

If data are modelled as a continuous-state Markov model and continuous-states are expressed via linear or nonlinear equations, with particle filter it is possible to compute approximately the developing sequence of posterior distributions. Approximation errors and the need for computational power can be managed by setting the number of particles. This property makes the use of particle filter especially attractive in practice. Unfortunately, it has a drawback for fault diagnosis since the filter needs to have a state transition and noise model for the faulty modes. If a non-modelled fault occurs its filter response is unpredictable.

The observable operator model (OOM), an alternative to HMM, is a mathematical model of linear operators for describing stochastic time-series processes. Compared with HMM OOM has several attractive properties: its theory presented in terms of linear algebra is easier to work with. OOM is able to express a broader range of processes than HMM. The available learning techniques are more accurate and do not get stuck to local maxima. Unfortunately, the basic version of the OOM learning algorithm is statistically inefficient and suffers from the *negative probability problem* (NPP). The novel approach to OOM estimation *efficiency sharpening (ES)* has a better statistical efficiency, but the NPP problem is still not solved. A variation of OOM, the norm-OOM allows avoiding the NPP. OOM for non-stationary processes could be used to diagnose environment faults. As the process of changing robot states depends on the environment and therefore is non-stationary, the design of a model for a non-stationary process can more exactly describe the normal and faulty behaviour of a robot with respect to changes in the environment.

6. Conclusions

The goal of this paper was to attempt a comprehensive evaluation of fault diagnosis methods in the robot domain. The objective of this work was to provide a kind of manual which should help a user to decide which of the four presented algorithms would best fit the demands of a particular autonomous system. The complex of the presented algorithms included one of the pioneering techniques in the fault diagnosis field – parity space, one of the most ubiquitous and famous – the hidden Markov model, a state-of-the-art algorithm – particle filter and a novel, developing approach – observable operator model.

The work can only be a first step in laying a foundation for an extended evaluation of diagnosis methods in the future. The apparent next step is to check the efficiency of the presented methods by applying them to the datasets provided by real robots, increasing the range of possible fault situations. Its complexity makes the mobile manipulator a good choice as a robot platform for testing the algorithms.

The OOM learning algorithm ES estimates more accrued models of stationary processes than HMM. This fact makes the OOM techniques very promising for future implementation as fault diagnosis methods. In my opinion norm-OOM should be the next diagnosis algorithm for evaluation. The negative probability problem being solved, an OOM for non-stochastic processes, could be a very promising tool for an efficient fault diagnosis of robots in a dynamic environment.

Hopefully in the future these achievements will help not only to select appropriate diagnosis algorithms for the demands of a given robot, but also to develop a complete fault handling system including fault recovery.

Appendices

A. State Estimation with Hidden Markov Model

```
% FILE NAME   :  HMM4WheelOMNI.m

% PURPOSE     :  HMM for Four Wheel Omni Driver.

% TOOLBOX     : Murphy, Kevin. Hidden Markov Model Toolbox for Matlab
% http://www.ai.mit.edu/~murphyk/Software/HMM/hmm_download.html



T = 10;                         % Number of time steps.
O = 13;                         % Number of observations
Q = 13;                         % Number of states
smallErr=0.000001;              % Value of an error

% ====================================================================
%             INITIALISATION AND PARAMETERS
% ====================================================================

% =====PROBABILISTIC MODEL REPRESENTATION=============================
% Transition matrix for discrete state depending on observations

transmat=zeros(Q,Q);
transmat(:,:,1) = ...
[0.8 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0 0 0 0;
 0.075 0.6 0.025 0.025 0.025 0.1 0.1 0.025 0.025 0 0 0 0;
 0.075 0.025 0.6 0.025 0.025 0.025 0.025 0.1 0.1 0 0 0 0;
 0.5 0.2143 0.2143 0.6 0.2143 0.2143 0.2143 0.2143 0.2143 0.1 0.1 0 0;
 0.5 0.2143 0.2143 0.2143 0.6 0.2143 0.2143 0.2143 0.2143 0 0 0.1 0.1;
 0.025 0.2 0.025 0.025 0.0083 0.5 0.2 0.0083 0.0083 0 0 0 0;
 0.025 0.2 0.025 0.025 0.025 0.2 0.5 0 0 0 0 0 0;
 0.025 0.025 0.2 0.025 0.025 0 0 0.5 0.2 0 0 0 0;
 0.025 0.025 0.2 0.025 0.025 0 0 0.2 0.5 0 0 0 0;
 0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.5 0.2 0 0;
 0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.2 0.5 0 0;
 0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.5 0.2;
 0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.2 0.5];

act=ones(1,T);

% measurement matrix
obsmat = zeros(Q,Q);
obsmat = ...
[0.8 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0.025 0 0 0 0;
 0.075 0.6 0.025 0.025 0.025 0.1 0.1 0.025 0.025 0 0 0 0;
 0.075 0.025 0.6 0.025 0.025 0.025 0.025 0.1 0.1 0 0 0 0;
 0.075 0.025 0.025 0.6 0.025 0.025 0.025 0 0 0.1 0.1 0 0;
 0.075 0.025 0.025 0.025 0.6 0.025 0.025 0 0 0 0 0.1 0.1;
 0.025 0.2 0.025 0.025 0.025 0.5 0.2 0 0 0 0 0 0;
 0.025 0.2 0.025 0.025 0.025 0.2 0.5 0 0 0 0 0 0;
 0.025 0.025 0.2 0.025 0.025 0 0 0.5 0.2 0 0 0 0;
 0.025 0.025 0.2 0.025 0.025 0 0 0.2 0.5 0 0 0 0;
 0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.5 0.2 0 0;
 0.025 0.025 0.025 0.2 0.025 0 0 0 0 0.2 0.5 0 0;
 0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.5 0.2;
 0.025 0.025 0.025 0.025 0.2 0 0 0 0 0 0 0.2 0.5];

L=4;                  % sliding window size

% ================== Four Wheel OMNI Robot ==========================
% Robot parameters
angle = 0.588;        % angle between wheels
R=0.25;               % robot radius

% Forward kinematics matrix the product of this matrix and
% wheel velocities vector is robot velocities vector

controlMat(:,:)=[sin(angle) -sin(angle) -sin(angle) sin(angle);...
                 -cos(angle) -cos(angle) cos(angle) cos(angle);...
                 1/(4*R) 1/(4*R) 1/(4*R) 1/(4*R)];

obserVal=ones(1,T);    % sequence of observations
threshold=0.1;

% ================== Load data ======================================
% - u(:,T)-matrix of 4xT size consists set of wheel velocities
% - velObs(:,T)-matrix of 3xT size includes set of robot velocities

load('C:\KA\Master_Thesis\MATLAB\PF\4wheelOMNI\RobotPoseData.mat');

% ===================================================================
%             SEQUENCE OF OBSERVATIONS
% ===================================================================

% ======= Generation of the sequence of observations ===
stuckDecayArr=zeros(3,T);
robotVelN=zeros(3,T);
velMatrix=zeros(3,4);
for t=1:T
   % calculate robot velocity "robotVelN" for the given control "u"
   % "u" is vector of wheel velocities
   robotVelN(:,t)=controlMat(:,:,1)*u(:,t);

   % compare measured and calculated velocities
   if(velObs(:,t)~=robotVelN(:,t))

       % fault is here
       velMatrix(:,1)=[
          (velObs(1,t)-robotVelN(1,t))/sin(angle)+u(1,t)+smallErr;
          (-velObs(2,t)+robotVelN(2,t))/cos(angle)+u(1,t)+smallErr;
           4*R*(velObs(3,t)-robotVelN(3,t))+u(1,t)+smallErr];
       velMatrix(:,2)=[
          (-velObs(1,t)+robotVelN(1,t))/sin(angle)+u(2,t)+smallErr;
          (-velObs(2,t)+robotVelN(2,t))/cos(angle)+u(2,t)+smallErr;
           4*R*(velObs(3,t)-robotVelN(3,t))+u(2,t)+smallErr];
       velMatrix(:,3)=[
          (-velObs(1,t)+robotVelN(1,t))/sin(angle)+u(3,t)+smallErr;
          (velObs(2,t)-robotVelN(2,t))/cos(angle)+u(3,t)+smallErr;
           4*R*(velObs(3,t)-robotVelN(3,t))+u(3,t)+smallErr];
       velMatrix(:,4)=[
          (velObs(1,t)-robotVelN(1,t))/sin(angle)+u(4,t)+smallErr;
          (velObs(2,t)-robotVelN(2,t))/cos(angle)+u(4,t)+smallErr;
           4*R*(velObs(3,t)-robotVelN(3,t))+u(4,t)+smallErr];
       % motor 1 faults
```

```
if(-0.00001<(velMatrix(1,1)-velMatrix(2,1))&
            (velMatrix(1,1)-velMatrix(2,1))<0.00001)&&...
  (-0.00001<(velMatrix(2,1)-velMatrix(3,1))&
            (velMatrix(2,1)-velMatrix(3,1))<0.00001)
   obserVal(t)=2;

   % save each value 1-4*R*((robotVelN(3,t)-
   %                        velObs(3,t))/u(1,t)) of
   % the motor1 with corresponding obserVal(t)=2 and
   % vector velMatrix(1,1)

   stuckDecayArr(:,t)=[obserVal(t);
                       velMatrix(1,1);
                       1-4*R*((robotVelN(3,t)-
                               velObs(3,t))/u(1,t))];

   if(t>=L)

       % copy last L data to stuckVal
       stuckVal=(stuckDecayArr(2,t-L+1:t));
       dacayVal=(stuckDecayArr(3,t-L+1:t));
       k=find(stuckDecayArr(1,t-L+1:t)==2);
       % check weither last L faults happend with the motor1

       s=find(0<=dacayVal&dacayVal<1);
       if(length(k)==L)

           %Motor 1 stuck
           copyArr(1,1:L)=stuckVal(1);
           if(-0.00001<sum(copyArr-stuckVal)& …
                       sum(copyArr-stuckVal)<0.00001)
                obserVal(t)=6;

           %Motor 1 dacay
           elseif (length(s)==L)
               sortArr=sort(dacayVal,1);
               if(-0.00001<sum(sortArr-dacayVal)&
                           sum(sortArr-dacayVal)<0.00001)
                   obserVal(t)=7;
               end
           end
       end
   end

% motor 2 faults
elseif (-0.00001<(velMatrix(1,2)-velMatrix(2,2))&
         velMatrix(1,2)-velMatrix(2,2))<0.00001)&&...
       (-0.00001<(velMatrix(2,2)-velMatrix(3,2))&
              (velMatrix(2,2)-velMatrix(3,2))<0.00001)
   obserVal(t)=3;
   stuckDecayArr(:,t)=[obserVal(t);
                       velMatrix(3,2);
                       1-4*R*((robotVelN(3,t)-
                               velObs(3,t))/u(2,t))];
   if(t>=L)
       stuckVal=(stuckDecayArr(2,t-L+1:t));
       dacayVal=(stuckDecayArr(3,t-L+1:t));

       % check weither last L faults happend with the motor2
       k=find(stuckDecayArr(1,t-L+1:t)==3);
       s=find(0<=dacayVal&dacayVal<1);

       if(length(k)==L)
```

```
            %Motor 2 stuck
            copyArr(1,1:L)=stuckVal(1);
            if(-0.00001<sum(copyArr-stuckVal)&
                        sum(copyArr-stuckVal)<0.00001)
                 obserVal(t)=8;

            %Motor 2 dacay
            elseif (length(s)==L)
                sortArr=sort(dacayVal,1);
                if(-0.00001<sum(sortArr-dacayVal)&
                    sum(sortArr-dacayVal)<0.00001)
                    obserVal(t)=9;
                end
            end
        end
    end

% motor 3 faults
elseif (-0.00001<(velMatrix(1,3)-velMatrix(2,3))&
        (velMatrix(1,3)-velMatrix(2,3))<0.00001)&&...
        (-0.00001<(velMatrix(2,3)-velMatrix(3,3))&
        (velMatrix(2,3)-velMatrix(3,3))<0.00001)
    obserVal(t)=4;
    stuckDecayArr(:,t)=[obserVal(t);
                        velMatrix(3,3);
                        1-4*R*((robotVelN(3,t)-
                        velObs(3,t))/u(3,t))];

     if(t>=L)
        stuckVal=(stuckDecayArr(2,t-L+1:t));
        dacayVal=(stuckDecayArr(3,t-L+1:t));
        % check weither last L faults happend with the motor3

        k=find(stuckDecayArr(1,t-L+1:t)==4);

        s=find(0<=dacayVal&dacayVal<1);
        if(length(k)==L)

            %Motor 3 stuck
            copyArr(1,1:L)=stuckVal(1);
            if(-0.00001<sum(copyArr-stuckVal)&
                sum(copyArr-stuckVal)<0.00001)
                 obserVal(t)=10;

            %Motor 3 dacay
            elseif (length(s)==L)
                sortArr=sort(dacayVal,1);
                if(-0.00001<sum(sortArr-dacayVal)&
                    sum(sortArr-dacayVal)<0.00001)
                    obserVal(t)=11;
                end
            end
        end
    end

% motor 4 faults
elseif (-0.00001<(velMatrix(1,4)-velMatrix(2,4))&
        (velMatrix(1,4)-velMatrix(2,4))<0.00001)&&...
        (-0.00001<(velMatrix(2,4)-velMatrix(3,4))&
        (velMatrix(2,4)-velMatrix(3,4))<0.00001)
    obserVal(t)=5;
    stuckDecayArr(:,t)=[obserVal(t);
```

```
                              velMatrix(3,4);
                              1-4*R*((robotVelN(3,t)-
                              velObs(3,t))/u(4,t))];
          if(t>=L)
              stuckVal=(stuckDecayArr(2,t-L+1:t));
              dacayVal=(stuckDecayArr(3,t-L+1:t));
              k=find(stuckDecayArr(1,t-L+1:t)==5);
              % check weither last L faults happend with the motor4

              s=find(0<=dacayVal&dacayVal<1);
              if(length(k)==L)

                  %Motor 4 stuck
                  copyArr(1,1:L)=stuckVal(1);
                  if(-0.00001<sum(copyArr-stuckVal)&
                      sum(copyArr-stuckVal)<0.00001)
                       obserVal(t)=12;

                  %Motor 4 dacay
                  elseif (length(s)==L)
                      sortArr=sort(dacayVal,1);
                      if(-0.00001<sum(sortArr-dacayVal)&
                         sum(sortArr-dacayVal)<0.00001)
                          obserVal(t)=13;
                      end
                  end
              end
          end
      end
  else
      % normal mode
      obserVal(t)=1;
      stuckDecayArr(:,t)=[1;u(1,t);1];
  end
end

%==================== Plot mode states =============================
figure(1)
clf
plot(1:T,z,'r','linewidth',2);
ylabel('Observation modes','fontsize',15);
xlabel('Time','fontsize',15);
axis([0 T+1 0 Q+1]);
grid on;
% ==================================================================
%                          HMM ESTIMATION
% ==================================================================

% This part of code involve the functions from K. Murphy's HMM toolbox

tic;                 % Initialize timer for benchmarking
flops(0);
prior0 = normalise(transmat(1,:,1));
transmat0 = mk_stochastic(transmat);
obsmat0 = mk_stochastic(obsmat);
obsmat1=zeros(Q,T);

% Create observation matrix
for i=1:T
  obsmat1(:,i)=obsmat(:,obserVal(i));
end

% estimation of fault states forward-backward algorithms
% HMM toolbox
[alpha, beta, gamma, loglik] = fwdback(prior0, transmat0, obsmat1,
'act', act);
time_pf = toc;

% ======================================================================
%                           SUMMARIES AND PLOTS
% ======================================================================
disp(' ');
disp('Overlooked errors');
disp('-----------------------------');
disp(' ');
disp('Execution time  (seconds)');
disp('-------------------------');
disp(' ');
disp(['HMM filter     = ' num2str(time_pf)]);
disp(' ');
filtDistPlot=[zeros(10,T)
            alpha(1,:)
              zeros(10,T)
            alpha(2,:)
                zeros(10,T)
            alpha(3,:)
              zeros(10,T)
            alpha(4,:)
              zeros(10,T)
            alpha(5,:)
              zeros(10,T)
            alpha(6,:)
              zeros(10,T)
            alpha(7,:)
              zeros(10,T)
            alpha(8,:)
              zeros(10,T)
            alpha(9,:)
              zeros(10,T)
            alpha(10,:)
              zeros(10,T)
            alpha(11,:)
              zeros(10,T)
            alpha(12,:)
              zeros(10,T)
            alpha(13,:)
              zeros(10,T)
            ];     % Zero pad to make plots look nice.

figure(2)
clf;
hold on
ylabel('t - time','fontsize',15)
zlabel('Pr(z_t|y_{1:t})','fontsize',15)
xlabel('z_t - state modes','fontsize',15)
title('HMM filter','fontsize',15)
for t=1:1:T,
 waterfall([1:153],t,filtDistPlot(:,t)');
end;
view(-20,60);
rotate3d on;
set(gca,'ygrid','off');
set(gca,'xtick',10:11:145);
set(gca,'xticklabel',{'N','W1','W2','W3','W4','M1s','M1d','M2s','M2d',
                      'M3s','M3d','M4s','M4d'})
```

B. State Estimation with Particle Filter

```
% FILE NAME   :  PF4omniDriver.m

% PURPOSE     :  PF for Four Wheel Omni Driver.

% TOOLBOX     : N. de Freitas, software for classical particle filters
%               and Rao- %Blackwellised particle filters
%               http://www.cs.ubc.ca/~nando/sofware.html


clear;
echo off;

% ====================================================================
%              INITIALISATION AND PARAMETERS
% ====================================================================

N = 200;                    % Number of particles.
T = 30;                     % Number of time steps.

% Here, we give you the choice to try three different types of
% resampling algorithms: multinomial (select 3), residual (1) and
% deterministic (2). Note that the code for these O(N) algorithms is
generic.

resamplingScheme = 2;

n_x = 3;                    % Continuous state dimension.
n_z = 7;                    % Number of discrete states.
n_y = 3;
n_u = 4;                    % Number control values

par.A = zeros(n_x,n_x,n_z); % Control matrix for state equation
par.B = zeros(n_x,n_x,n_z); % State noise
par.C = zeros(n_y,n_x,n_z); % observationmatrix
par.D = zeros(n_y,n_y,n_z); % observation noise
par.E = zeros(n_x,n_x,n_z);
par.K = zeros(3,n_u,n_z);   % State control matrix
par.G = zeros(n_y,3,n_z);
for i=1:n_z,
  par.A(:,:,i) = eye(n_x,n_x);
  par.C(:,:,i) = eye(n_y,n_x);
  par.B(:,:,i) = 0.01*eye(n_x,n_x);
  par.D(:,:,i) = 0.01*eye(n_y,n_y);
  par.G(:,:,i) =  eye(n_y,n_x);
end;

% Transition matrix for discrete state.
par.T = [0.75 0.05 0.05 0.05 0.05 0.05 0;
         0.025 0.7 0.025 0 0.05 0.05 0.05;
         0.1 0.05 0.8 0.05 0 0 0;
         0.1 0 0.05 0.8 0.025 0 0;
         0.1 0.05 0 0.05 0.8 0 0;
         0.05 0.05 0.025 0.025 0.025 0.8 0.025;
         0.05 0.05 0.025 0.025 0.025 0.025 0.8];

for i=1:n_z,
  par.T(i,:) = par.T(i,:)./sum(par.T(i,:));
end;
```

```
par.pz0 = [0.75; 0.05; 0.05; 0.05; 0.05; 0.05]';
par.pz0 = par.pz0./sum(par.pz0);

par.mu0 = zeros(n_x,1);                 % Initial Gaussian mean.
par.S0  = 0.1*eye(n_x,n_x);             % Initial Gaussian covariance.
par.fd = [0.01; 0.001];

%================== Four Wheel OMNI Robot ==========================

% Robot parameters
angle = 0.588;       % angle between wheels
R=0.25;              % robot radius
stuck_val = 0;
% Control matrix for the state equation
par.K(:,:,1)=[sin(angle) -sin(angle) -sin(angle) sin(angle);
              -cos(angle) -cos(angle) cos(angle) cos(angle);
              1/(4*R) 1/(4*R) 1/(4*R) 1/(4*R)];
for i=2:5
    par.K(:,:,i)=par.K(:,:,1);
    par.K(:,i-1,i)= par.K(:,i-1,i)*stuck_val;
end

% ====================================================================
%                          GENERATE THE DATA
% ====================================================================

% Load data from the file.
load('C:\KA\Master_Thesis\MATLAB\PF\4wheelOMNI\RobotPoseData.mat');

% The data includes input vector and true fault modes which happand
% with robot

% Initialization
x = zeros(n_x,T);
y = zeros(n_y,T);
z = ones(1,T);

% A sequence of observable states
for i=20:T
    z(i)=4;
end
u=zeros(4,T);

% Set input wheel velocities for each time step
for i=2:T
    u(:,i)=[2;-2;-2;2];
end

x(:,1) = [0;0;0];
angle=zeros(T,1);
robot_vel=zeros(3,1);

% create observations for the given state sequence
for t=2:T,
   stuckVal=1;
   if(z(t)==6)
       robot_vel = par.K(:,:,1)*[3;u(2,t);u(3,t);u(4,t)];
   elseif (z(t)==7)
       par.K(:,:,7)=par.K(:,:,1);
       par.K(:,1,7)=par.K(:,1,7)*1/(1.01^(t-1));
       % define velocities
       robot_vel = par.K(:,:,7)*u(:,t);
   else
```

```
        % define velocities
        robot_vel = par.K(:,:,z(t))*u(:,t);
    end

    x(:,t) = robot_vel;
    y(:,t) = par.C(:,:,z(t))*x(:,t);
    if(t>2)
    pred_val=[1 1];
    pred_val(1)=1/((4*R*x(3,t)-u(2,t)-u(3,t)-u(4,t))/u(1,t));
    pred_val(2)=1/((4*R*x(3,t-1)-u(2,t-1)-u(3,t-1)-u(4,t-1))/u(1,t-1));
    new_u=[u(1,t)/(pred_val(1)+abs(pred_val(1)-pred_val(2)));
                                                     u(2,t);
                                                     u(3,t);
                                                     u(4,t)];

    n_u1=par.K(:,:,1)*new_u;
   end
end;

% ============== Plot the discrete modes ===========================
figure(2)
clf
plot(1:T,z,'r','linewidth',2);
ylabel('z_t','fontsize',15);
axis([0 T+1 0 n_z+1])
grid on;

% ===================================================================
%                              PF ESTIMATION
% ===================================================================

% INITIALISATION:
% ===================================================================
z_pf = ones(1,T,N);             % These are the particles for the
                                % estimate of z. Note that there's no
                                % need to store them for all t. We're
                                % only doing this to show you all the
                                % nice plots at the end.
z_pf_pred = ones(1,T,N);        % One-step-ahead predicted values of z.
x_pf = 10*randn(n_x,T,N);       % These are the particles for the
estimate x.
x_pf_pred = x_pf;
y_pred = 10*randn(n_y,T,N);     % One-step-ahead predicted values of y.
w = ones(T,N);                  % Importance weights.

initz = 1/n_z*ones(1,n_z);
for i=1:N,
  z_pf(:,1,i) = length(find(cumsum(initz')<rand))+1;
end;
v_pf = zeros(3,T,N);
k=zeros(3,T);
disp(' ');
tic;
rot_angle=zeros(T,N);
L=4; %sliding window size
% Initialize timer for benchmarking
angl = 0.588;
for t=2:T,
  fprintf('PF :  t = %i / %i  \r',t,T); fprintf('\n');


% SEQUENTIAL IMPORTANCE SAMPLING STEP:
% ===================================================================
  for i=1:N,
    % sample z(t)~p(z(t)|z(t-1))
    z_pf_pred(1,t,i) = length(find(cumsum(par.T(z_pf(1,t-
1,i),:)')<rand))+1;
    z_val=z_pf_pred(1,t,i);

    % sample x(t)~p(x(t)|z(t|t-1),x(t-1))
    if(z_pf_pred(1,t,i)==6)
    % Wheel 1 stuck.
    % Calculation of correct velocities for the preveous state
       correct_vel=par.K(:,:,1)*u(:,t-1);
       if(correct_vel~=x_pf(:,t-1,i))
           if((u(2,t)==u(2,t-1))&&
              (u(3,t)==u(3,t-1))&&
              (u(4,t)==u(4,t-1)))
               v_pf(:,t,i) = x_pf(:,t-1,i);
           else
              new_u=[
               4*R*x_pf(3,t-1,i)-u(2,t-1)-u(3,t-1)-u(4,t-1);
               u(2,t);
               u(3,t);
               u(4,t)];
               v_pf(:,t,i) = par.K(:,:,1)*new_u;
           end
       else
          v_pf(:,t,i) = par.K(:,:,2)*u(:,t);  % define velocities
       end
    % Wheel 1 decay scenario
    elseif(z_pf_pred(1,t,i)==7)
       if(t>2)
           pred_val=[1 1];
           %calculate the decay value for previous
           pred_val(1)=u(1,t-1)/(4*R*x_pf(3,t-1,i)-
                                 u(2,t-1)-u(3,t-1)-u(4,t-1));
           pred_val(2)=u(1,t-2)/(4*R*x_pf(3,t-2,i)-
                                 u(2,t-2)-u(3,t-2)-u(4,t-2));

           new_u=[u(1,t)/(pred_val(1)+abs(pred_val(1)-pred_val(2)));
                  u(2,t);
                  u(3,t);
                  u(4,t)];
           n_u=par.K(:,:,1)*new_u;
        else
           v_pf(:,t,i) = par.K(:,:,2)*u(:,t);
       end

    elseif(z_pf_pred(1,t,i)==8)
          new_u=[u(1,t);
                 4*R*x_pf(3,t-1,i)-u(1,t-1)-u(3,t-1)-u(4,t-1);
                 u(3,t);
                 u(4,t)];
          v_pf(:,t,i) = par.K(:,:,1)*new_u;
    else
        v_pf(:,t,i) = par.K(:,:,z_pf_pred(1,t,i))*u(:,t);
    end

        x_pf_pred(:,t,i) = v_pf(:,t,i) + ...
                           par.B(:,:,z_pf_pred(1,t,i))*randn(n_x,1);

  end;

  % Evaluate importance weights.
  % ===================================================================
  for i=1:N
    y_pred(:,t,i) = par.C(:,:,z_pf_pred(1,t,i))*x_pf_pred(:,t,i) + ...
                    par.D(:,:,z_pf_pred(1,t,i))*randn(n_y,1);
    Cov = par.D(:,:,z_pf_pred(1,t,i))*par.D(:,:,z_pf_pred(1,t,i))';

    w(t,i) =  (det(Cov)^(-0.5))*exp(-0.5*(y(:,t)-y_pred(:,t,i))'* ...
                         pinv(Cov)*(y(:,t)-y_pred(:,t,i))) + 1e-99;
  end;
  w(t,:) = w(t,:)./sum(w(t,:));             % Normalise the weights.

% SELECTION STEP:
% ====================================================================
  if resamplingScheme == 1
    outIndex = residualR(1:N,w(t,:)');        % Higuchi and Liu.
  elseif resamplingScheme == 2
    outIndex = deterministicR(1:N,w(t,:)');   % Kitagawa.
  else
    outIndex = multinomialR(1:N,w(t,:)');     % Ripley, Gordon, etc.
  end;
  z_pf(1,t,:) = z_pf_pred(1,t,outIndex);
  x_pf(:,t,:) = x_pf_pred(:,t,outIndex);

end;   % End of t loop.
time_pf = toc;     % How long did this take?

% ====================================================================
%                          SUMMARIES AND PLOTS
% ====================================================================

z_plot_pf = zeros(T,N);
for t=1:T,
  z_plot_pf(t,:) = z_pf(1,t,:);
end;

z_num_pf = zeros(T,n_z);
z_max_pf = zeros(T,1);
for t=1:T,
  for i=1:n_z,
    z_num_pf(t,i)= length(find(z_plot_pf(t,:)==i));
  end;
  [arb,z_max_pf(t)] = max(z_num_pf(t,:));
end;

detect_error_pf = sum(z~=z_max_pf');
if(z(1)~=z_max_pf(1))
    detect_error_pf=detect_error_pf-1;
end


disp(' ');
disp('Overlooked errors');
disp('-----------------------------');
disp(' ');
disp(['PF      = ' num2str(detect_error_pf)]);
disp(' ');
disp(' ');
disp('Execution time  (seconds)');
disp('-------------------------');
disp(' ');
disp(['PF      = ' num2str(time_pf)]);
disp(' ');
```

```
figure(3)
clf;
domain = zeros(N,1);
range = zeros(N,1);
thex=[0.5:0.05:n_z+.5];
hold on
ylabel('t','fontsize',12)
zlabel('Pr(z_t|y_{1:t})','fontsize',12)
xlabel('z_t','fontsize',12)
for t=1:1:T,
 [range,domain]=hist(z_plot_pf(t,:)',thex);
  waterfall(domain,t,range/sum(range))
end;
view(-30,80);
rotate3d on;
set(gca,'ygrid','off');
title('Particle Filter')
```

C. State Estimation with Observable Operator Model

```
% FILE NAME   :  OOM4wheelOMNI.m

% PURPOSE     :  OOM for Four Wheel Omni Driver.

% TOOLBOX     : Jaeger, Herbert. OOM Matlab implementation.
%            http://www.faculty.iu-bremen.de/hjaeger/OOM/OOMGeneric.zip


% ====================================================================
%              INITIALISATION AND PARAMETERS
% ====================================================================

global dim alphabetsize charEvLength charEvents w0Int  tlInt;

%===================== BUILD A MODEL==================================
stateNr = 5;                  % number of states
obsNr = 5;                    % number of observations
Msparsity = 0.3;
Osparsity = 0.5;

procureOOM4OMNI(stateNr, obsNr, Msparsity, Osparsity);

% generate run of lenght numberOfPoints, collect states in 2 or 3
% lists according to observable producing the state
tl=tlInt;
w0=w0Int;
modDim = length(tl(1,:,1));
numberOfPoints=10;
statePlotLength = numberOfPoints;
modStatePL3D = zeros(statePlotLength,3,min([alphabetsize 6]));
modStateCounters = ones(alphabetsize);
%w = w0;
squeezedTL = zeros(alphabetsize,modDim);
for i = 1:alphabetsize
        squeezedTL(i,:) = sum(tl(:,:,i));
end

%====================================================================
load('C:\KA\Master_Thesis\MATLAB\PF\4wheelOMNI\RobotPoseData.mat');

pvec=zeros(5,numberOfPoints);
w=zeros(5,numberOfPoints+1);
w(:,1)=w0;
for n = 1:numberOfPoints
        % update model
        pvec(:,n) = squeezedTL * w(:,n); % probability vector
        choice = obsArr(n);
        w(:,n+1) = tl(:,:,choice) * w(:,n);
        w(:,n+1) = w(:,n+1) / sum(w(:,n+1));
end

% ====================================================================
%                          SUMMARIES AND PLOTS
% ====================================================================

filtDistPlot=[zeros(10,numberOfPoints)
            pvec(1,:)
              zeros(10,numberOfPoints)
            pvec(2,:)
                zeros(10,numberOfPoints)
            pvec(3,:)
              zeros(10,numberOfPoints)
            pvec(4,:)
              zeros(10,numberOfPoints)
            pvec(5,:)
          ];     % Zero pad to make plots look nice.
figure(2)
clf;
hold on
ylabel('t - time','fontsize',14)
zlabel('Pr(b|a_{1:t})','fontsize',14)
xlabel('System states','fontsize',14)
%title('OOM','fontsize',15)
for t=1:1:numberOfPoints,
 waterfall([1:55],t,filtDistPlot(:,t)');
end;
view(-20,60);
rotate3d on;
set(gca,'ygrid','off');
set(gca,'xtick',10:11:58);
set(gca,'xticklabel',{'N','W1','W2','W3','W4'})
```

Reference:

1. F. Gustafsson. *Adaptive filtering and change detection*. JohnWiley & Sons, Ltd, 2001.
2. F. Gustafsson. *Statistical signal processing approaches to fault detection.* Annual Reviews in Control (JARAP), 31(1):41-54, 2007.
3. L.R. Rabiner. *A tutorial on hidden Markov model and selected applications in speech recognition.* Proceedings of the IEEE 77(2):257-286, February 1989.
4. A. Doucet, N. de Freitas, N. Gordon (editors). *Séquentiel Monte Carlo Methods in Practice.* Springer. 2001.
5. H. Jaeger, M. Zhao, K. Kretzschmar, T. Oberstein, D. Popovici, A. Kolling *Learning observable operator models via the ES algorithm*. In S. Haykin, J. Principle, T. Sejnowski, and J. McWhirter, editors, *New Directions in Statistical Signal Processing: from Systems to Brain*, MIT Press, Cambridge, MA., pages 417-464 2006.
6. V. Venkatasubramanian, R.Rengaswamy, Kewen Yin, and Surya N. Kavuri. *A review of process fault detection and diagnosis part1: Quantitative model-based methods*. Computers and Chemical Engineering, 27:293–313, 2003.
7. V. Venkatasubramanian, R.Rengaswamy, Kewen Yin, and Surya N. Kavuri. *A review of process fault detection and diagnosis part 2: Qualitative models and search strategies*. Computers and Chemical Engineering, 27:313– 326, 2003.
8. V. Venkatasubramanian, R.Rengaswamy, Kewen Yin, and Surya N. Kavuri. *A review of process fault detection and diagnosis part 3: Process history based methods.* Computers and Chemical Engineering, 27:327–346, 2003.
9. P. Sundvall. *Mobile robot fault detection using multiple localisation modules*. Master's thesis, KTH School of Electrical Engineering, Stockholm, Sweden, 2006.
10. R. Isermann *Fault-diagnosis systems: An introduction from fault detection to fault tolerance.* Berlin: Springer-Verlag, 2006.
11. M. Staroswiecki. *Model based FDI: the control approach. Plenary lecture*. Bridge Workshop, Sancicario. Italy. March 2001.
12. Steven. X. Ding *Model-Based Fault Diagnosis Techniques: Design Schemes, Algorithms, and Tools*. Springer, May 2008.
13. J. Gertler. *Fault detection and isolation using parity relations*. Control Engineering Practice, 5(5):653-661, 1997.

14. E. Chow, A Wilsky. *Analytical redundancy and the design of robust failure detection systems.* Automatic Control, IEEE Transactions on, 29(7): 603-614 July 1984.
15. J de Kleer and B.C. Williams. *Diagnosing multiple faults.* Artificial Intelligence, 32(1):97– 130, 1987.
16. J de Kleer and B.C. Williams. *Diagnosis with behaviour models*. In Proceedings of IJCAI-89, pages 1324-1330, 1989
17. J. Kurien and P. Nayak. *Back to the future for consistency-based trajectory tracking*. In Proceedings of AAAI-00, ppages 370-377, 2000
18. B.C. Williams and P. Nayak. *A model-based approach to reactive self-configuring systems*. In Proceedings of AAAI-96, 2:971-978, Portland, OR, August 1996.
19. B.C. Williams, M.D. Ingham, S.H. Chung, and P.H. Elliott. *Model-based programming of intelligent embedded systems and robotic space explorers*. In Proceedings of the IEEE, 91(1): 212—237, January 2003.
20. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
21. V Verma, G Gordon, R Simmons, S Thrun. *Particle Filters for Rover Fault Diagnosis*, Robotics and Automation Magazine, June 2004
22. S. Thrun, J. Langford, and V. Verma. *Risk sensitive Particle Filters*. In Neural Information Processing Systems (NIPS), December 2001.
23. V. Verma, S. Thrun, and R. Simmons. Variable resolution Particle Filter. In *International Joint Conference of Artificial Intelligence*, 2003.
24. E. Benazera, R. Dearden, and S. Narasimhan. *Combining Particle Filters and consistency-based approaches*. In 15th Int. Workshop on Principles of Diagnosis, Carcassonne, France, 2004.
25. C. Plagemann, D. Fox, W. Burgard. *Efficient Failure Detection on Mobile Robots Using Particle Filters with Gaussian Process.* Proposal European Robotics Symposium 2006, Springer-Verlag Serlin Heidelberg, Germany, pages. 93-107
26. R. Dearden, T. Willeke, F. Hutter, R. Simmons, V. Verma and S. Thrun. *Real-time Fault Detection and Situational Awareness for Rovers: Report on the Mars Technology Program Task.* In Proceedings of IEEE Aerospace Conference, pages 826-840, 2004.

27. M. Nyberg. *Model Based Fault Diagnosis Methods, Theory, and Automotive Engine Application.,* PhD, Department of Electrical Engineering, Linköping University, Linkoeping, Sweden, 1999

28. R. Ganguli. *Health monitoring of a helicopter rotor in forward flight using fuzzy logic*. AIAA Journal, 40(12):2373–2381, September 2002.

29. A. Marcos, S. Ganguli, and G. Balas. *Application of H¥ fault detection and isolation to a boeing 747-100/200 aircraft.* In Proceedings of AIAA-2002-4944, Monterey, CA, August 2002.

30. M. Borairi and H. Wang. *Actuator and sensor fault diagnosis of nonlinear dynamic systems via genetic neural networks and adaptive parameter estimation technique.* In Proceedings of IEEE International Conference on Control Applications, pages. 278–282 , September 1998.

31. Y.-W. Kim, G. Rizzoni, and V. Utkin. *Automotive engine diagnosis and control via nonlinear estimation* IEEE Control Systems Magazine, 18:84–99, October 1998.

32. D.-L. Yu. *Diagnosing simulated faults for an industrial furnace based on bilinear model* IEEE Transactions on Control Systems Technology, 8:435–442, May 2000.

33. P. Garimella and B. Yao. *Model based fault detection of an electro hydraulic cylinder.* In Proceedings of American Control Conference, 1:484-489, June 2005.

34. E. Mesbahi. *An intelligent sensor validation and fault diagnostic technique for diesel engines.* Journal of Dynamic Systems, Measurement, and Control, 123: 141–144, March 2001.

35. Z. Ye and B. Wu. *A review on induction motor online fault diagnosis.* In Proceedings of the 3rd International Power Electronics and Motion Control Conference, 3:1353-1358, August 2000.

36. S. Lee, M. D. Bryant, and L. Karlapalem. *Model and information theory-based diagnostic method for induction motors*. Journal of Dynamic Systems, Measurement, and Control, 128:584–591, September 2006.

37. R. H. Chen, H. K. Ng, J. L. Speyer, L. S. Guntur, and R. Carpenter. *Health monitoring of a satellite system*. Journal of Guidance, Control, and Dynamics, 29(3):593–605, 2006.

38. H. Rotstein, R. Ingvalson, T. Keviczky, and G. J. Balas. *Fault detection design for uninhabited aerial vehicles*, Journal of Guidance, Control, and Dynamics, 29(5):1051–1060, 2006.
39. A. Duyar and W. Merrill, *Fault diagnosis for the space shuttle main engine.* Journal of Guidance, Control, and Dynamicss, 15(2):384–389, 1992.
40. Y. Zhang, J. Wu, M. Huang, H. Zhu, and Q. Chen, *Liquid-propellant rocket engine health-monitoring techniques*. Journal of Propulsion and Power, 14(5):657–663, 1998.
41. R. K. Yedavalli *Robust Estimation and Fault Diagnostics for Aircraft Engines with Uncertain Model.* Data Proceedings of the 2007 American Control Conference Marriott Marquis Hotel at Times Square New York City, USA, July 2007.
42. Y. Zhang. *Detection and diagnosis in dynamic systems. Lecture 1. Introduction to Fault Detection and Diagnosis (FDD)*, Department of Computer Science and Engineering, Aalborg University Esbjerg, 2006.
43. C. Bonivento, A. Isidori, L. Gentili, L. Marconi, and A. Paoli. *Fault detection and isolation and fault tolerant control.* Research work. Online at http://www.casy.deis.unibo.it/files/fdiftc.pdf (Accessed: September 06, 2008), 2001.
44. The Desire Consortium. *Deutsche service robotik initiative*. Online at http://www.projekt-desire.de/ (Accessed: September 03, 2008), 2008.
45. A. Shakhimardanov. *Report on the Research and Development Project 2 Fault Tolerance and Robustness in Robotics: A Survey.* R&D2 project, Fachhochschule Bonn-Rhein-Sieg, Sankt Augustin, Germany, 2006 .
46. R. Isermann. *Model-based fault-detection and diagnosis – status and applications.* Annual Reviews in Control 29:71-85, 2005.
47. M. Basseville and I.V. Nikiforov. *Detection of abrupt changes: theory and application*. Information and system science series. Prentice Hall, Englewood Cliffs, NJ., 1993.
48. J. Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker, Inc, 1998.
49. P. Frank. *Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy- A survey and some new results*, Automatica 26(3): 459–474, 1990.

50. M. Kinnaert *Fault diagnosis based on analytical models for linear and nonlinear systems – A tutorial.* In Proceedings Safeprocess, Washington, U.S.A. pages 133-139, 2003
51. A. Hagenblad, F. Gustafsson and I. Klein. *A comparison of two methods for stochastic fault detection: The parity space approach and principal components analysis*. Proceedings of 13th IFAC Symposium on System Identification, pages 1090–1095. 2003
52. *Wikipedia*. Online at http://en.wikipedia.org/ (Accessed: September 04, 2008), 2008.
53. H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. *Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring*. Proceedings of SIAM International Conference on Data Mining, 2004.
54. Home Page of Fault Detection and Diagnosis in Engineering Systems. Online at http://teal.gmu.edu/~jgertler/lab/paper.html (Accessed: September 04, 2008), 2008.
55. V. Filaretov, M. Vukobratovic and A. Zhirabok. *Parity relation approach to fault diagnosis in manipulation robots.* Mechatronics 13(2):141-152, March 2002.
56. A. Varga. A *Fault Detection Toolbox for MATLAB* Computer-Aided Control Systems Design, 2006 IEEE International Symposium on, 4(6): 3013–3018 October 2006.
57. S. X. Ding, E. Atlas, S. Schneider, Y. Ma, T. Jeinsch and E. L. Ding: *An introduction to a MATLAB-based FDI-toolbox,* Proc. of IFAC Symposium SAFEPROCESS, Beijing, 2006.
58. P. Frank. *Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy- a survey and some new results*. Automatica 26(3):459–474, 1990.
59. Y. Tharrault, G. Mourot, J. Ragot, D. Maquin. *Fault detection and isolation with robust principal component analysis.* Control and Automation, 2008 16th Mediterranean Conference on, Ajaccio, France, pages 59-64, June 2008
60. A.T. Bharucha-Reid *Elements of the Theory of Markov Processes and Their Applications.* New York: McGraw-Hill, 1960

61. D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble Jr., B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, and B. C. Williams, *Design of the remote agent experiment for spacecraft autonomy.* In Proceedings of the IEEE Aerospace Conference, 1998.
62. L. Fesq, M. Ingham, M. Pekala, J. V. Eepoel, D. Watson, and B. Williams, *Model-based autonomy for the next generation of robotic spacecraft.* In Proceedings of 53rd International Astronautical Congress, October 2002.
63. R. Dearden and T. Willeke and F. Hutter and R. Simmons and V. Verma and S. Thrun *Real-time Fault Detection and Situational Awareness for Rovers: Report on the Mars Technology Program Task.* In Proceedings in IEEE Aerospace Conference pages 826- 840, 2004
64. T. Clapp and S. Godsill, *Improvement strategies for Monte Carlo particle filters*, in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds., New York: Springer Verlag, 2001.
65. A. Doucet, S. J. Godsill, and C. Andrieu, *On sequential Monte Carlo sampling methods for Bayesian filtering.* Statistics and Computing, pages. 197-208, 2000.
66. J. Carpenter, P. Clifford, and P. Fearnhead, *Improved particle fillter for non-linear problems*. IEEE Proceedings on Radar and Sonar Navigation, 146(1):2-7, 1999.
67. D. Crisan, P. Del Moral, and T. J. Lyons, *Non-linear filtering using branching and interacting particle systems*. Markov processes and Related Fields, 5(3):293-319, 1999.
68. H. Jaeger. *Characterizing distributions of stochastic processes by linear operators.* GMD Report 62, German National Research Centre for Information Technology, 1999. http://www.faculty.iubremen.de/hjaeger/pubs/oomdistributionsTechRep.pdf. (Accessed: September 04, 2008),
69. H. Jaeger. *A short introduction to observable operator models of stochastic processes.* Proceedings of the Cybergenetics and Systems 1998 Conference, 1, Austrian Society for Cybergenetics Study, 38-43 1998
70. T. Oberstein. *Efficient Training of Observable Operator Models using Context Graph*. Master's thesis, Institute for autonomous intelligent systems, Fraunhofer AIS Mathematical Institute / ZAIK, University Cologne, Gremany, 2002.
71. C. Chen. *Linear system theory and design.* 3rd ed., Oxford University Press, 1999

72. R. Rojas. *Omnidirectional Control*, Freie Universität Berlin; Institut für Informatik; Veranstaltung „Robotik“ http://www.inf.fu-berlin.de/lehre/WS04/Robotik/omnidrive.pdf (Accessed: September 04, 2008), 2008
73. M. Mladenov, M. Mock, K.-E Grosspietsch. *Fault monitoring and correction in a walking robot using LMS filters.* Intelligent Solutions in Embedded Systems, 2008 International Workshop on pages: 1-10 July 2008.
74. J. Tan and N. Xi *Integrate task planning and control for mobile manipulators Robotics and Automations*. Proceedings ICRA 02, IEEE Internetional Conference on 1:382-387
75. J. Tan, N.Xi and Y. Wang *Integrated Task Planning and Control for Mobile Manipulators.* The international Journal of Robot Research, 22(5): 337-354, May 2003.
76. W.H.Huang, G.F. Holden, *Nonprehensile palmar manipulation with a mobile robot*. Intelligent Robots and Systems, Proceedings IEEE/RSJ International Conference on 1:114-119, 2001.
77. A. Petrovskaya and A. Y. Ng *Probabilistic mobile manipulation in dynamic environments, with application to opening doors*. In IJCAI, 2007
78. M. Hans, B. Graf, and R. Schraft. *Robotic home assistant Care-O-bot: past-present-future.* Robot and Human Interactive Communication, 2002. Proceedings.11th IEEE International Workshop on page 380–385.
79. E. Sudderth. *Hidden Markov models, graphical models*. Slides http://www.eecs.berkeley.edu/~pliang/cs294-spring08/lectures/hmm (Accessed: September 24, 2008)
80. S. Russell, P. Norvig. *Artificial Intelligence - A Modern Approach.* 2nd Edition. Prentice Hall. 2003.
81. A. W. Moore. *Hidden Markov Models*. Slides from a tutorial presentations http://www.cs.cmu.edu/~awm/tutorials (Accessed: September 30, 2008)
82. M. Bolic. *Architectures for Efficient Implementation of Particle filters*, Ph.D. thesis, Department of Electrical Engineering, State University of New York at StonyBrook, 2004.

83. N. de Freitas, *Software for classical particle filters and Rao-Blackwellised particle filters* http://www.cs.ubc.ca/~nando/sofware.html. (Accessed: September 24, 2008)

84. H. Jaeger, M. Zhao and A. Kolling. *Efficient estimation of OOMs*. Advances in Neural Information Processing Systems 18 (Y. Weiss, B. Schölkopf and j. Platt, eds.), MIT Press, Cambridge, MA pages 555-562, 2005.

85. H. Jaeger *Discrete-time, discrete-valued observable operator models: a tutorial* , Slides, International University Bremen, July, 2003 (Accessed: September 30, 2008)

86. M. Zhao, H. Jaeger. *Norm observable operator models.* Technical report 8, School of Engineering and Science, July, 2007.

87. Murphy, Kevin. *Hidden Markov Model Toolbox for Matlab.* http://www.ai.mit.edu/~murphyk/Software/HMM/hmm_download.html (Accessed: September 4, 2008)

88. Jaeger, Herbert. *OOM Matlab implementation.* http://www.faculty.iu-bremen.de/hjaeger/OOM/OOMGeneric.zip (Accessed: September 23, 2008)