ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE
ENGINEERING AND TECHNOLOGY

# ISOLATION OF ACTION EXECUTION FAILURES FOR COGNITIVE ROBOTS

**M.Sc. THESIS**

**Doğan ALTAN**

**Department of Computer Engineering**

**Computer Engineering Programme**

**MAY 2014**

**ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY**

# ISOLATION OF ACTION EXECUTION FAILURES FOR COGNITIVE ROBOTS

**M.Sc. THESIS**

**Doğan ALTAN**
**(504111511)**

**Department of Computer Engineering**

**Computer Engineering Programme**

**Thesis Advisor: Asst. Prof. Dr. Sanem SARIEL TALAY**

**MAY 2014**

**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**

# BİLİŞSEL ROBOTLAR İÇİN EYLEM YÜRÜTME HATALARININ TANISI

**YÜKSEK LİSANS TEZİ**

**Doğan ALTAN**
**(504111511)**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Bilgisayar Mühendisliği Programı**

**Tez Danışmanı: Asst. Prof. Dr. Sanem SARIEL TALAY**

**MAYIS 2014**

**Doğan ALTAN**, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504111511 successfully defended the thesis entitled **"ISOLATION OF ACTION EXECUTION FAILURES FOR COGNITIVE ROBOTS"**, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

| | | |
|---|---|---|
| **Thesis Advisor :** | **Asst. Prof. Dr. Sanem SARIEL TALAY**<br>Istanbul Technical University | ..... |
| **Jury Members :** | **Asst. Prof. Dr. Yusuf YASLAN**<br>Istanbul Technical University | ..... |
| | **Asst. Prof. Dr. Sırma YAVUZ**<br>Yıldız Technical University | ..... |

**Date of Submission :** **5 May 2014**
**Date of Defense :** **30 May 2014**

***Anneme ve babama,***

## FOREWORD

I am grateful to my supervisor, Asst. Prof. Dr. Sanem SARIEL TALAY, for her guidance and support during this thesis.

I also want to thank the members of Artificial Intelligence and Robotics Laboratory (AIR Lab.), Melis KAPOTOĞLU, Çağatay KOÇ, Mustafa ERSEN, Melodi Deniz ÖZTÜRK, Mehmet BİBERCİ, Sertaç KARAPINAR and Burak TOPAL for their contributions on different parts of the robot software.

Finally, I would like to thank my mother and father for their endless support. This thesis is dedicated to them.

This thesis is funded by a grant from the Scientific and Technological Research Council of Turkey (TUBITAK), Grant No. 111E-286.

May 2014

Doğan ALTAN
(Computer Engineer)

**TABLE OF CONTENTS**

**Page**

**FOREWORD** ..... **ix**
**TABLE OF CONTENTS** ..... **xi**
**ABBREVIATIONS** ..... **xiii**
**LIST OF TABLES** ..... **xv**
**LIST OF FIGURES** ..... **xvii**
**SUMMARY** ..... **xix**
**ÖZET** ..... **xxi**
**1. INTRODUCTION** ..... **1**
1.1 Purpose of Thesis ..... 2
1.2 Hypothesis ..... 2
**2. BACKGROUND** ..... **3**
2.1 Failure Detection ..... 3
2.2 Failure Diagnosis ..... 3
2.2.1 HMM-based diagnosis methods ..... 7
**3. HIDDEN MARKOV MODELS** ..... **9**
3.1 Hidden Markov Models (HMMs) ..... 9
3.1.1 Markov processes ..... 9
3.1.2 Hidden Markov Models (HMMs) ..... 10
3.1.2.1 Basic problems of HMMs ..... 11
3.2 Hierarchical Hidden Markov Models (HHMMs) ..... 14
**4. PROBABILISTIC FAILURE ISOLATION** ..... **17**
4.1 A Taxonomy of Robot Failures in Cognitive Tasks ..... 17
4.2 HHMM-based Failure Isolation ..... 20
4.2.1 Model parameters ..... 24
4.2.1.1 World state definition ..... 25
4.2.1.2 Transition probabilities ..... 26
4.2.1.3 Observation probabilities ..... 26
4.3 A Running Example ..... 29
**5. EXPERIMENTS** ..... **33**
5.1 Experimental Setup ..... 33
5.2 Experimental Results ..... 34
5.2.1 Case scenarios ..... 35
5.2.1.1 Scenario I ..... 35
5.2.1.2 Scenario II ..... 36
5.2.1.3 Scenario III ..... 38
5.2.1.4 Scenario IV ..... 39
5.2.1.5 Scenario V ..... 39

5.2.1.6 Scenario VI ..... 40
5.2.2 Performance evaluation ..... 41
5.2.2.1 Performance analysis of the method with maximum failure probability transfer values ..... 41
5.2.2.2 Performance analysis of the method with average of the failure probability transfer values ..... 42
5.2.2.3 Performance analysis of the method with the hybrid approach..... 43
5.2.2.4 Performance analysis of the method with Particle Filters ..... 43
5.2.3 Analysis of memory usage ..... 44
**6. CONCLUSION ..... 47**
6.1 Future Work ..... 47
**REFERENCES ..... 49**
**CURRICULUM VITAE ..... 53**

## ABBREVIATIONS

| | |
|---|---|
| **HMM** | **:** Hidden Markov Model |
| **HHMM** | **:** Hierarchical Hidden Markov Model |
| **POMDP** | **:** Partially Observable Markov Decision Process |
| **CPT** | **:** Conditional Probability Table |
| **DBN** | **:** Dynamic Bayesian Network |
| **PF** | **:** Particle Filter |
| **FDI** | **:** Fault Detection and Identification |
| **AR** | **:** Autoregressive Model |

## LIST OF TABLES

**Page**

**Table 4.1** : Action-Failure type relations used in the model. ..... 21
**Table 4.2** : Predicates and the related sensory data. ..... 25
**Table 4.3** : CPT for the vision model. ..... 27
**Table 4.4** : CPT for the external event model. ..... 28
**Table 4.5** : CPT for the gripper model. ..... 28
**Table 4.6** : CPT for the localization model. ..... 29
**Table 5.1** : Performance analysis of the method with maximum probability transfer values. ..... 42
**Table 5.2** : Performance analysis of the method using average approach values.. 42
**Table 5.3** : Performance analysis of the method using hybrid approach..... 43
**Table 5.4** : Performance analysis of the method with Particle Filters. ..... 44
**Table 5.5** : Analysis of state generation on HHMMs. ..... 45

## LIST OF FIGURES

**Page**


**Figure 3.1** : A three state Markov model [1]. ..... 10
**Figure 3.2** : An example HMM unfolded in time. The HMM structure builds a lattice showing all possible hidden state sequences [1]. ..... 11
**Figure 3.3** : Forward-backward procedure [1]. ..... 13
**Figure 3.4** : A four-level HHMM [2]. ..... 15
**Figure 4.1** : A taxonomy of robot failures in cognitive tasks. ..... 18
**Figure 4.2** : An illustration of an HHMM failure model corresponding to the actions of a given plan. ..... 21
**Figure 4.3** : Transition probabilities of the proposed method. ..... 26
**Figure 4.4** : An example model for a manipulation scenario. On the top, the plan generated by the planner is illustrated. In the lower level, hierarchical failure models are given. ..... 31
**Figure 5.1** : (Left) Pioneer 3-AT robot; (right) objects that are used in the experiments. ..... 33
**Figure 5.2** : (Left) Outputs of the LINEMOD and (right) segmentation algorithms [3]. ..... 34
**Figure 5.3** : (Left) The vision algorithm detects a small ball on the boundary of the big ball. (Right) The segmentation algorithm correctly clusters the point cloud of the big ball. ..... 35
**Figure 5.4** : A consistent vision failure for a particular object results in an increase in the corresponding probability. ..... 36
**Figure 5.5** : A hand-made gripper failure is injected into the system. Probabilities change accordingly. ..... 37
**Figure 5.6** : The resulting probabilities when an external event exists in the scene. ..... 38
**Figure 5.7** : The resulting probabilities when a localization failure occurs. ..... 39
**Figure 5.8** : Both an external event and vision failure occur in the environment. Probabilities change accordingly. ..... 40
**Figure 5.9** : An external event, a gripper failure and a vision failure occur in the environment. ..... 41

**ISOLATION OF ACTION EXECUTION FAILURES FOR COGNITIVE ROBOTS**

## SUMMARY

Robots execute actions in the physical world in order to reach their goals. They may face several failures during the execution of actions in their plans due to dynamic and partially observable structure of the physical world. For example, an object may not be grasped successfully by the robot due to a wrong grasp position. Another failure may occur when vision algorithm of the robot fails to recognise an object. Furthermore, the vision algorithm may falsely classify an object. Yet in another case, state of an object may be changed by another agent. These failures should be detected by the robot, and a reasoning procedure should take place to diagnose the causes of the failures. During the execution, deciding on an abnormal situation about the execution is called *Failure Detection*. In order to detect a failure, a monitoring procedure must be applied in both plan and action level. Finding the root cause of a failure situation is called *Failure Isolation*. Although failure detection is an important issue for robotics, failure isolation is also needed for robust execution. To do so, the robot should apply an isolation procedure.

A robot should interpret one or more sensors (touch, pressure, vision etc.) in order to monitor the execution and infer about failure cases. Therefore, the robot should interpret the scene and extract related information about the current situation. In order to accurately isolate a failure, the robot should have the models of the possible failures that can occur in the environment during the execution. The cause of the failure may not be related to the action in execution at an instant time step. It may be depending on a previously executed action's undesired effects or an external event. For example, in the blocks world domain, if a base block is not properly constructed, a stack action may destroy the unbalanced tower. Analysis of these kind of failures should be done from a temporal dimension. Furthermore, there may be more than one cause of a failure. In such cases, a temporal and probabilistic model is required in order to isolate these type of failures.

A *Hierarchical Hidden Markov Model (HHMM)*-based failure isolation method is presented in this thesis in order to explain the failure cases with relevant facts after they are detected. Failure models are represented as distinct HHMMs. During the execution of the actions in the plan, these models are updated while the scene is interpreted. Relations between the actions and the failure models are used to eliminate unnecessary model updates during the execution. In case of a failure, Viterbi algorithm is invoked to find the most likely explanation of the failure. Considering the calculated probabilities, the candidate models responsible for the failure are found.

Several failure cases on a real robot are investigated in order to analyse the performance of the proposed method. The results show that using a probabilistic method allows to diagnose failures when there are more than one cause of a failure. Using the

relations between the actions and the failure types decreases memory requirement of the proposed method by eliminating unnecessary model updates during the plan execution. The main contribution lies in representing each failure type hierarchically in parallel and using the action-failure type relations.

**BİLİŞSEL ROBOTLAR İÇİN**
**EYLEM YÜRÜTME HATALARININ TANISI**

## ÖZET

Yapay Zeka konusundaki gelişmeler ile birlikte günümüzdeki robot sistemleri daha akıllı hale gelmiştir. Robot teknolojilerindeki son gelişmeler ile de, robotların gerçek dünya uygulamalarında kullanımları yaygınlaşmıştır. Günümüzde artık birçok probleme robotlar aracılığı ile çeşitli çözümler getirilmektedir.

Bilişsel bir robot, planlama, yürütme, yürütme gözlemleme ve hata tanısı görevlerini yerine getirebilmelidir. Planlama, bir başlangıç durumundan istenen herhangi bir hedef durumuna robotu ulaştıracak planın üretilmesidir. Yürütme, planlayıcı tarafından üretilen sembolik plandaki eylemlerin yürütülmesi iken yürütme gözlemleme, planda yer alan herhangi bir eylemin yürütülmesi sırasında ortaya çıkabilecek beklenmedik hata durumlarının sezilmesi işidir. Hata tanısı ise, yürütme anında oluşabilecek hata durumlarının nedenlerine açıklama getirme işine karşılık gelir.

Gerçek dünyanın dinamik ve kısmi gözlemlenebilir yapısı nedeniyle, robotik uygulamalarında yürütme hatalarının oluşması söz konusu olabilir. Bu hata durumları, robot tarafından sezilmeli ve bu durumların nedenlerini belirlemek için bir çıkarsama yordamı yürütülmelidir. Bu hataların sezilebilmesi için yürütme, hem eylem hem de plan seviyesinde gözlemlenmelidir. Gürbüz eylem yürütme için her ne kadar hata sezme önemli bir problem olsa da, oluşan hatanın nedeni de (hata tanısı) belirlenebilmelidir. Hata tanısı yapılabilmesi için de bir çıkarım mekanizmasının yürütülmesi gerekmektedir. Bu çalışmada, olasılıksal bir hata tanı yöntemi önerilmektedir. Çalışma kapsamında, donanım/duyarga kısıtları, kısıtlı bilgi veya çevresel dış etkenler nedeniyle oluşabilecek eylem yürütme hataları ele alınmaktadır.

Yürütme gözlemleme ve hata durumları hakkında çıkarım yapma bir veya daha fazla duyargadan (görüntü, kuvvet, dokunma veya basınç) gelen bilgiyi yorumlamak ile mümkün olabilmektedir. Bu nedenle, robotun hata sezme için ortamı yorumlaması ve bir çıkarsama yordamı yürütmesi gerekir. Benzer bir yordam, robotun hata tanısı yapabilmesi için de gereklidir. Doğru bir hata tanısı yapabilmek için, robotun ortamda oluşabilecek hata modelleri hakkında bilgi sahibi olması gerekmektedir. Bazı durumlarda, hatanın nedeni o anda yürütülmekte olan eylem ile doğrudan ilintili olmayabilir. Hatanın nedeni, robot tarafından daha önce yürütülmüş olan bir eylemin istenmeyen bir sonucuna veya çevresel bir etkene ilişkin olabilir. Örneğin küp dünyası domeninde, alt blok düzgün bir şekilde yerleştirilmemiş ise, hali hazırda dizilmiş olan kulenin üzerine yeni bir blok konması işlemi sırasında bütün yapı yıkılabilir. Bu tür hata durumlarının tanısını koyabilmek için zamansal çıkarım yapılması gerekir. Dahası, bir hatanın birden fazla nedeni olabilir. Bu gibi durumlarda hatanın nedenini belirleyebilmek için olasılıksal ve zamansal bir modele ihtiyaç duyulmaktadır.

Bu çalışmada, hata durumlarının nedenlerini belirleyebilmek için Hiyerarşik Saklı Markov Modellerine (HSMM) dayalı bir hata tanı yöntemi önerilmektedir. Önerilen yöntem, paralel olarak işlenen ve farklı tipte hata durumlarını temsil eden HSMM'lerini içermektedir. Herbir hata tipi, ayrı bir SMM (Saklı Markov Modeli) ile hiyerarşik bir yapıda modellenmiştir. HSMM'lere dayalı olan model, durumların zamansal boyutta analizinin yapılmasına olanak vermekte ve beklenen durumlardan bir sapma gerçekleştiğinde zamansal hata bilgisinin durumlar arasında yayılmasını sağlamaktadır. Birden çok hipoteze göre hata durumunun olasılıksal olarak tanısını yapabilmek için farklı durumların aynı anda takip edilmesi gerekir. Bu çalışmanın ana katkısı, her bir hata tipinin eylem-hata tipi ilişkilerini de kullanarak ayrı bir hiyerarşik SMM olarak temsil edilmesidir. Bu temsil, bir nesne veya çevresel olay üzerine kalıcı olan hata durumları da dahil olmak üzere birçok hata durumunun tanısını mümkün kılmaktadır.

Bu çalışmada incelenen eylem yürütme hataları şunlardır: Bütün nesneler için *Görüntü işleme*, *Konumlama*, *Donanım kısıtları (eyleyici/efektör)* ve bütün nesneler için *Dış olay* hatalarıdır. *Görüntü(nesne)* hata modeli, görüntü işleme algoritmasının belirli bir nesneyi doğru olarak tanıyamadığı durumu temsil eder. *Donanım kısıtları (eyleyici/efektör)* hata tipi, robotun fiziksel kapasitesi dışındaki durumları temsil eder. *Dış olay(nesne)* hata tipi ise robotun kontrolü dışında, dış dünyada oluşan değişiklikler nedeniyle oluşan hataları temsil eder. *Konumlama* hata tipi robotun ortamda konumunu belirleyemediği durumları temsil eder.

Önerilen sistem, planlayıcı tarafından üretilen ve eylem dizisi içeren sembolik planı ve ayrı ayrı paralel olarak temsil edilen hata tipi modellerini girdi olarak kabul etmektedir. Sistemin çıktısı ise, hata durumunun nedeninden sorumlu olan olası hata modelleri listesidir. Robot planında yer alan bir eylem üç durumda olabilir: *başarı*, *hata* ve *yürütmede*. *başarı* durumu robotun bir eylemi beklenen sonuçları ile sonlandırdığını, *hata* durumu eylemin beklenmedik etkilerle sonlandığını ve *yürütmede* ise robotun eylemi yürütmeyi henüz bitirmediğini temsil eder. Her eylemin yürütülmesi sırasında, Hiyerarşik SMM'lerin alt katmanında o eyleme karşı düşen modeller daha önce tanımlanmış olan eylem-hata tipi ilişkileri göz önüne alınarak güncellenir. Eğer hata tipi modeli ve eylem ilgili ise model güncellenir, değil ise o eylem süresince ilgili hata modeli pasif kalır. Yürütme aşamasında robotun duyargaları aracılığı ile alınan gözlemler, duyargaların istatistiksel analizine bağlı olan bir olasılık dağılımına karşılık düşürülür ve bu değere göre yeni durum oluşturulur.

İlgili eylemin yürütülmesi sona erince, alt katman modelinde Viterbi algoritması yürütülür ve alt katmandaki durumlardan hata nedeni olma olasılığı en yüksek olan durumun değeri üst katmana aktarılır. Üst katmandaki model, yürütülen eylem ile ilgili ise güncellenir. Aksi takdirde model pasif olarak işaretlenir ve ilgili modelin şimdiki zaman düğümü yaratılmadan bir önceki durumun zaman vektörü genişletilir. Bu durum, robot planındaki bütün eylemleri başarılı bir şekilde yürütünceye veya hata durumu ile karşılaşıncaya kadar devam eder. Hataların, gözlemlemeye dayalı bir yöntem ile sezildiği varsayılmaktadır. Hata durumu oluştuğunda, üst katman modellerinde Viterbi Algoritması yürütülerek hataya neden olabilecek modeller belirlenir. Bunun için, modellerdeki durumlar incelenir ve *hata* ile işaretlenmiş durumlar saptanarak olasılıklarına bakılır. Bu olasılıklardan belli bir eşik değerinin üzerinde olan modeller aday listesine eklenir. Çalışma sonunda bu liste olası hata nedenleri olarak döndürülür.

Yöntemin başarım analizi Pioneer 3-AT gezgin robotu ile farklı tipte hatalar içeren senaryolar üzerinde yapılmıştır. Elde edilen sonuçlar, geliştirilen yöntemin hataların birden fazla nedene bağlı olduğu durumlarda da olasılıksal olarak belirlenebileceğini göstermektedir. Ayrıca, hata tipleri ve eylemler arasındaki ilişkilerin kullanılması da durum uzayını daraltarak, yöntemin bellek gereksinimini azaltmaktadır.

# 1. INTRODUCTION

Robot technologies have shown great progress in recent years. Such advances make robot systems applicable to many domains. Therefore, many hardware systems and algorithms have been proposed for the robots. However, due to the complex nature of the robotic environments, failures may occur during the execution of robotic missions.

An autonomous robot may face several failure situations during the execution of its actions to achieve a goal [4] due to non-deterministic actions or different sources of uncertainty in physical dynamic environments. A monitoring system is pivotal in order to achieve goals robustly in the face of uncertainties. While detecting failures is one of the central problems for robust execution [5–7], the robot should also identify the reasons of the failure for efficient recovery. Isolation of a failure requires an inference process to find the underlying reason behind the failure. In this research, the main focus is on a probabilistic failure isolation method. We address action execution failures that may arise due to hardware/sensor limitations, limited knowledge on some environmental features [8] or external events.

Monitoring and reasoning about failures requires interpreting data from one or more sensors (e.g., vision, force, touch, pressure). Therefore, in order to detect a failure, the robot needs to interpret the scene and apply certain reasoning tools to come up with correct conclusions. To achieve complete isolation of the failure, the robot needs to maintain a priori information on the models of failures that are likely to occur in the environment. In some cases, the reason of a failure may not be directly related to the action that is being executed. The reason of the failure may be depending on a previous action that is executed by the robot or an external event. For example, in the blocks world domain, when the base block structure is not properly formed, the execution of a stack action on the existing tower may fail, and the whole structure may be completely destroyed. In order to isolate these types of failures, a temporal reasoning model is

needed. Furthermore, there may be more than one cause of a failure. To deal with such cases, a probabilistic temporal model is useful to identify the possible failure cases.

## 1.1 Purpose of Thesis

In this thesis, the main goal is to isolate the causes of the action execution failures for robust execution. For efficient recovery, causes of action execution failures should be indicated by the robot. Otherwise, the robot may not be able to reach its desired goal state. Therefore, an inference mechanism should be applied in order to isolate possible action execution failures.

## 1.2 Hypothesis

In this thesis, a Hidden Markov Model (HMM) based isolation method is proposed to determine causes of the failures. The method includes parallel Hierarchical HMMs (HHMMs) running for tracking modelled failure types. The HMM model provides temporal analysis of states and propagates temporal failure information over time after a deviation occurs. Multiple hypotheses are tracked at the same time to identify underlying causes in a probabilistic manner. The main contribution lies in modelling each failure type as a distinct HHMM running in parallel considering action-failure type relations in a robotic planning domain. Several failure types are modelled and isolation of these failures is done in a probabilistic isolation framework. Failure situations are investigated in real robot scenarios. Furthermore, using the relations between the failure types and the actions decreases the number of generated states during the execution.

## 2. BACKGROUND

Failure detection and isolation (diagnosis) is an intensively investigated issue for robot systems due to the need for safe plan execution. In this section, literature review on the failure detection/diagnosis is presented. First, failure detection methods are examined. Afterwards, failure diagnosis methods are presented.

### 2.1 Failure Detection

Failure detection is applied to intuit the inconsistencies during the execution. In [5], sources of uncertainty classified under four categories, namely, missing information, unreliable sources, stochastic phenomena and inherently vague concepts. After this categorization, execution monitoring methods in robotics are presented under three categories: analytical approaches, data-driven approaches and knowledge-based approaches. Monitoring can take place both in plan level and action level. A common approach to detect failures is using an observer-based approach [5]. In this approach, predefined models and inconsistencies between the expected and observed outcomes are used to detect failures.

A comprehensive survey in execution monitoring is presented in [9]. Monitoring is surveyed in terms of external monitoring, expectation-based monitoring and model based monitoring (Diagnosis and state evaluation).

### 2.2 Failure Diagnosis

Failure diagnosis is applied in order to find the reasons behind the failures. In a knowledge-based data driven fault diagnosis approach [10], failures are diagnosed using Fuzzy Logic in three steps: acquiring information, making inferences and taking actions. In the first phase, process trend analysis is used to extract information from the real data. A system that makes fuzzy inference is given in the second step. In that step, if-then rules that are gathered from the experts are used. After making inferences,

some actions to return the system to its normal conditions are done in the third step. However, the performance of the method is highly dependent on the quality of expert knowledge.

Failure causes are listed in three headings in [11]. First, an action fails to produce its desired effect. Second, the world evolves different from it is modelled. Last, an exogenous event happens. However, autonomous robots may also fail because of errors in sensing, interpretation of sensor data and action execution. Monitoring and reasoning are achieved to detect and cope with failures. The idea of this paper is to detect inconsistencies between the theory and the model of the real world, plan execution and actual observations. Experiments are done on a delivery robot. It delivers letter, calculator and folder to their final destinations. From representation of states, logical inferences are made due to the failures. For example, if the robot detects an object and believes that it belongs to another room, it first tries to grasp it to verify if it really exists or not. If the robot can successfully grasp the object, it updates its belief.

Uncertainties may arise over the reasons of failures. This is handled with semantic-knowledge based execution monitoring where the robot estimates a probability distribution according to its expectations [12]. Structural abstraction is used for model-based diagnosis in earlier works [13, 14].

Model-based failure diagnosis in technical processes has been investigated by many researchers previously [15]. Dynamic Bayesian Networks (DBN) and Particle Filters are used for failure diagnosis [16]. In this work, a two-phased approach is used for failure diagnosis. DBNs are used to find the most possible candidate electric machines that failed. DBNs are learned off-line from raw data. After selecting the most possible candidate electric machines, Particle Filters (PFs) are used to track potential failed machine. Furthermore, a number of complementary algorithms are presented in [17]. These algorithms are for improving the accuracy of Fault Detection and Identification (FDI) with a Particle Filter.

In another work, a hierarchical representation with Particle Filters is used for failure diagnosis [18] to handle faults. A tree structure is used to represent faults where leaf nodes are faults and internal nodes are intermediate nodes that aggregate similar fault

nodes. Due to the hierarchical representation in modelling faults, only single faults are identified. In the same work, Partially Observable Markov Decision Processes (POMDPs) are also proposed for failure diagnosis in indoor mobile robot navigation. The policy for POMDP is gathered off-line.

Extended action models are also proposed in a control loop that integrates monitoring, diagnosis and recovery [19]. Actions of a robot are represented with their conditions both in time $t$ and $t+1$. Furthermore, degraded versions of these situations are also defined in the extended action model. These models are used to detect failures. After detection of a failure, the robot diagnoses the failure by replanning from current situation to $R$ where $R$ is the repaired status. This status is defined with three conditions: (1) all the functionalities in faulty situation is fixed, (2) preconditions of the next action is satisfied (3) for the further actions, no open preconditions are left. If an empty plan returns, the robot plans to another safe situation $S$ in order not to effect other robots in the environment.

Meta-cognitive loop (MCL) is a method to detect failures and respond them [20]. Process of dealing with a failure consists of three steps. First, the agent notes an anomaly. After noting the anomaly, it tries to assess it. Indications are represented in a graph structure. Failure ontology is also represented in a graph structure which is connected to indication event nodes via inter-ontological links called diagnostic nodes. These inter-ontological links allow meta-cognitive loop to move into guide state. In this state potential responses are found and after calculating their utilities, one of them is chosen and implemented. If a failure occurs again, then the response found before is marked and indications are updated with the new failure.

A history-based belief management framework is proposed in [21]. IndiGolog is used as agent programming language. This framework is used to detect and handle failures that may be encountered during execution. Failures that happen during picking up an object, picking up a wrong object and sensing failures are recovered. A delivery robot simulation in an office is used for experimental setup. The robot has a tactile sensor for deciding if it is carrying an object or not and a vision sensor which registers objects in a room. Hypotheses are used to overcome failures. Using history based diagnosis, action sequences which are suitable with the current situation are generated. Situation calculus is used to encode every activity and change is encoded as actions.

Proposed method includes four main steps: updating hypotheses, checking consistency of favourite hypotheses, diagnosis of hypotheses and deciding on new hypotheses.

In [22], logic programming with situation calculus [23] is studied in order to explain the unexpected deviations in task execution using hypotheses and their costs. Situation calculus is used to reason about an actions and their effects. External events, wrong knowledge and sensor failures are handled. A hypothesis pool is used in order to explain such situations. During the execution, each hypothesis is updated considering the actual sensor result and relations among fluent values. In case of inconsistencies, the actual hypothesis is removed from the pool and explaining diagnoses are generated. Pool has a maximum hypothesis capacity. A cost value is defined for each hypothesis. The cost is calculated by using the additional cost of insertions and variations on the new hypothesis plus the original cost of the original hypothesis. Finally, a hypothesis with the lower cost is selected to be processed again until inconsistencies do not exist on it.

Robots can learn new actions and their preconditions and effects from natural language instructions [24]. After learning new actions, robot updates its knowledge base and replans. As an example, a room with a locked door and a robot in it is given. The goal for the robot is to be in the outside of the room but the robot does not know pushing the door one meter has an effect of opening the door. An architecture namely DIARC is proposed and this framework has such components namely speech recognizer, vision processor, laser processor, goal manager, SapaReplan planner and motion planner. Such a statement "If you are at a closed door and you push it one meter, you will be in the room." causes the robot to parse the sentence and create a new action to open a door. The robot adds this action to its domain definition and when it faces a door again, it executes the new learned action. First Order Logic is used to represent knowledge. SapaReplan planner is an extended version of Sapa planner and it can make plans with considering costs, durations and partial satisfaction of goals.

In another work, situations where a planner-based agent is not able to find a plan to achieve a goal state is investigated [25]. Causal graphs and domain transition graphs are used to analyse the relations between fluent symbols. Excuses (causes of why a planner can not find a plan) are classified under three categories: acceptable excuses, good excuses and perfect excuses. Acceptable excuses are defined with having less

initial facts in the explanation. When two acceptable excuses are given, one of them is said to be good if it subsumes the other. An excuse is said to be perfect if its cost is minimal. The cost defines the change rate in the initial state to explain the absence of the plan. A* search is used with two different heuristics. One of them is LM Cut heuristic and the other one is enhanced-additive heuristic. The proposed method can introduce excuses in such situations where the planner is not able to find a plan.

### 2.2.1 HMM-based diagnosis methods

In some model-based fault detection and isolation systems, HMMs are used to monitor processes [26]. Force/torque signals are used to feedback the proposed models. Models are trained with Baum-Welch algorithm to estimate the parameters.

HMMs are also previously proposed to diagnose failures in different domains. Accidents in nuclear power plants are diagnosed using HMMs [27]. In this work, Baum-Welch algorithm is used to re-estimate the parameters of the model. Eight accident types and a normal state are represented as a single HMM and accidents are identified calculating the probability of each model after gathering observations.

In another work [28], HMMs are used to diagnose systems with partial and imperfect tests. A fault dictionary matrix is provided to associate faults and tests. If a test can detect a failure, the relation is established in the matrix. HMM with known parameters, adaptive HMM (parameters of the HMM is learned with Baum-Welch algorithm) and a hamming distance based diagnosis method is compared.

A bearing fault detection and isolation system is proposed in [29]. Vibration signals are used and using a polynomial transfer function, features are extracted. Failures are detected with pre-trained HMM for the normal case. Diagnosis is made by the HMM that has the maximum probability. Method is evaluated on the data derived from drive and ball bearing of an induction motor.

HMMs are used in a mobile robot system to estimate the most appropriate mode for the robot in execution [30]. Three modes are defined: plan, stop and react. In the plan mode, the robot plans a path from its current position to a location. It uses a motion planning algorithm. Modality react is used to avoid obstacles in the environment using the laser range finder data. Mode stop corresponds to the modality where the robot

stops. During the execution of a task, the robot uses the a priori information and monitoring to select and switch among the modalities.

HMMs are also used in fault diagnosis of metal stamping processes [31]. Feature extraction is done by using autoregressive model (AR). After extraction of the features, HMMs are used for classification.

Continuous HMMs (CHMMs) are employed for diagnosing mechanical fault signals [32]. CHMMs are HMMs of a vector sequence. Baum-Welch algorithm is used to train HMMs. Monitoring signals are gathered and feature vector sequence is extracted. After this phase, log-likelihood of each model is calculated and the model which has the maximum probability is selected. A rotor simulator is used in the experiments and rotor fault signals are gathered.

In another work [33], fast Fourier transform (FFT), autoregressive (AR) time series, wavelet transform and bi-spectrum are used as feature extraction methods. Then, feature vectors are used the train HMMs with Baum-Welch algorithm. After training phase, fault type decision is made considering calculated probabilities.

# 3. HIDDEN MARKOV MODELS

In this chapter, Hidden Markov Models (HMMs) and Hierarchical Hidden Markov Models (HHMMs) are explained.

## 3.1 Hidden Markov Models (HMMs)

In this section, first Markov Processes are explained. Afterwards, Hidden Markov Models (HMMs) are examined [1].

### 3.1.1 Markov processes

In a Markov Process a system can be in $N$ distinct states. At a given time $t$, the system can be in one of the states: $S_1$, $S_2$, $S_3$,...,$S_N$. At each time step $t$, the system may change its state with a probability. The state at time $t$ is also denoted with $q_t$. This definition forms a general *Markov Process*. The system moves into a new step by considering previous states.

In Markov Processes, the current state of the system ($q_t$) is defined as a function of the history of the system. Depending on the number of past states considered, the order of Markov processes are determined accordingly. For example, if the state of the system only depends on the intermediate previous state, it is called a *first-order Markov Process* (Equation 3.1).

$$P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k ...) = P(q_t = S_j | q_{t-1} = S_i) \tag{3.1}$$

Similarly, a *second-order Markov Process* would be defined for a system that considers the last two previous states in order to define the current state (Equation 3.2).

$$P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k ...) = P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k) \tag{3.2}$$

It is also assumed that transition between any two state is independent of time (Equation 3.3). An example three state Markov model is shown in Figure 3.1. In the figure, each circle represents a single state. The system can be in one of these states at any time step. $\pi_i$ represents the probability of the situation where the system is in state $i$ initially. $a_{ij}$ defines the probability of transferring from state $i$ to state $j$.

$$a_{ij} = P(q_t = S_j | q_{t-1} = S_i) s.t. \begin{cases} a_{ij} \geq 0 \\ \sum_{j=1}^{N} a_{ij} = 1 \end{cases} \tag{3.3}$$

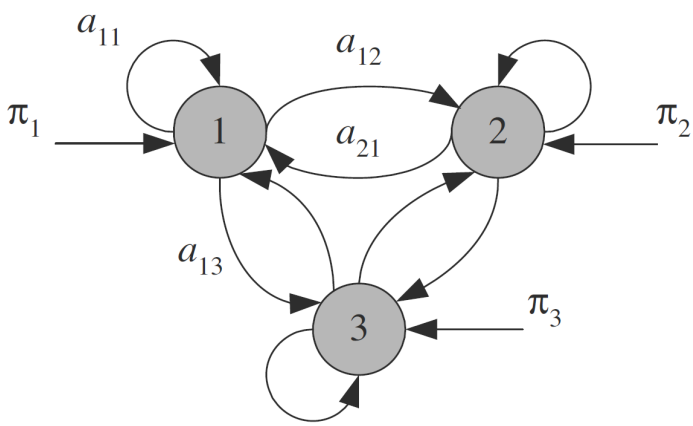


**Figure 3.1**: A three state Markov model [1].

### 3.1.2 Hidden Markov Models (HMMs)

Hidden Markov Models (HMMs) [34] are probabilistic temporal structures that include hidden states. The underlying model of the HMMs are Markov processes. The next step of the system is defined only considering the current state of the system. Therefore, Markov property holds for Hidden Markov Models (HMMs).

An HMM consists of five components, namely states, observations, transition probabilities, observation probabilities and the initial state distribution.

- There are $N$ hidden states in an HMM. Each hidden state is denoted with $S_i \in S$ where $S$ is the set of hidden states.
- Transition model, $A = a_{ij}$, defines the probability of transferring from state $S_i$ to state $S_j$ where $S_i, S_j \in S$.

- Observations, denoted by the $O_t$, represent the sensory information gathered at time $t$.

- Observation model, denoted by $B = b_{S_i}(O_t)$, defines the probability of gathering the observation $O_t$ at state $S_i$.

- Initial state probability distribution is represented with $\pi = \pi_i$. This distribution defines the probability $P(q_1 = S_i)$.

Given the components above, an HMM is represented with a tuple; $\lambda = (A,B,\pi)$. The model can be used to generate arbitrary number of observation sequences given $\lambda$.

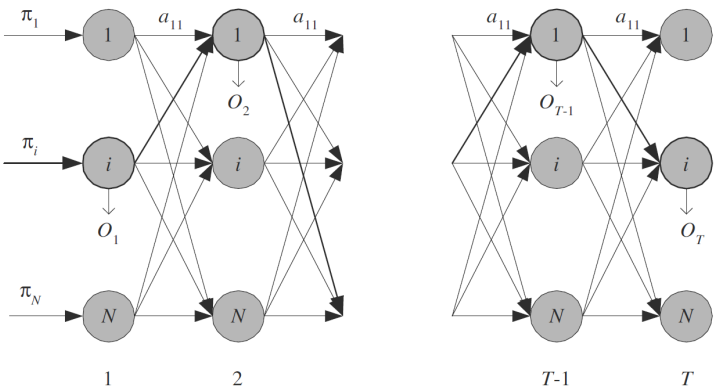


**Figure 3.2**: An example HMM unfolded in time. The HMM structure builds a lattice showing all possible hidden state sequences [1].

An example HMM is shown in Figure 3.2. In this figure, each column represents a time step and each node is a hidden state. Once an observation is gathered, the system passes another state. All possible hidden state sequences are shown in Figure 3.2.

#### 3.1.2.1 Basic problems of HMMs

There are three basic problems of HMMs given a number of sequence of observations.

- Calculating the likelihood of a given observation sequence, $O = \{ O_1, O_2 \ldots O_t \}$, $P(O|\lambda)$. The problem is called the *Evaluation Problem*.

- Finding the most likely state sequence of a given observation sequence, $O = \{ O_1, O_2 \ldots O_t \}$, given $\lambda$, $P(Q|O,\lambda)$. The problem is called the *Finding State Sequence*.

- Given a training set of observation sequences $X = \{O^k\}_k$, learning the model parameters $\lambda^*$ with maximizing $P(X|\lambda)$. This problem is called *Learning Model Parameters*.

Considering an observation sequence, $O = \{ O_1, O_2 \ldots O_t \}$, and a state sequence, $Q = \{ S_1, S_2 \ldots S_t \}$, probability of observing the given observation sequence is defined as follows:

$$P(O|\lambda, Q) = \prod_{t=1}^{T} P(O_t|S_t, \lambda) = b_{S_1}(O_1).b_{S_2}(O_2).b_{S_2}(O_2) \tag{3.4}$$

However, this production can not be calculated since the state sequence is unknown. The probability of the state sequence is defined as:

$$P(Q|\lambda) = P(S_1)\prod_{t=2}^{T} P(S_t|S_{t-1}) = \pi_{S_1}.a_{S_1S_2}\ldots a_{S_{t-1}S_t} \tag{3.5}$$

Then the joint probability is:

$$P(O,Q|\lambda) = P(S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})\prod_{t=1}^{T} P(O_t|S_t) = \pi_{S_1}.b_{S_1}(O_1).a_{S_1S_2}.b_{S_2}(O_2)\ldots b_{S_t}(O_t).a_{S_{t-1}S_t} \tag{3.6}$$

$P(O|\lambda)$ can be calculated by summing up over all possible $Q$:

$$P(O|\lambda) = \sum_{allpossibleS} P(O,Q|\lambda) \tag{3.7}$$

However, there are $N^T$ possible states in $Q$ where $Q$ is the possible state sequences. Therefore, it is not efficient to calculate all these possible states. In order to calculate $P(O|\lambda)$, *forward-backward procedure* is proposed. The observation sequence is divided into two parts. A forward variable ($\alpha_t(i)$) is defined to represent the probability of observing the observation sequence until time $t$ and being at hidden state $S_i$. Similarly, a backward variable ($\beta_t(i)$) is defined to represent the probability of observing the observation sequence from time $t+1$ to $T$. These probability values can be calculated recursively (Figure 3.3).

Forward variable is calculated as follows:

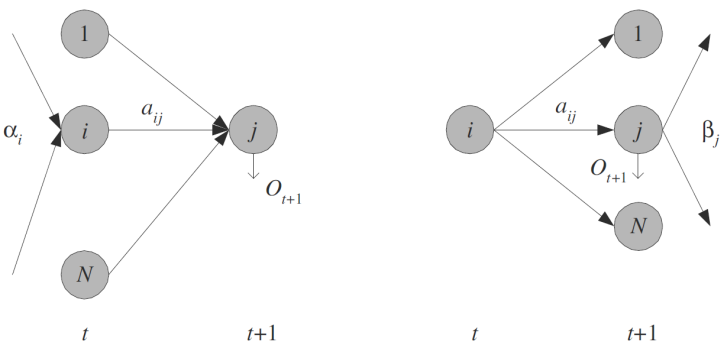


**Figure 3.3**: Forward-backward procedure [1].

- Initialization

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{3.8}$$

- Recursion

$$\alpha_{t+1}(j) = P(O_1 ... O_{t+1}, q_{t+1} = S_j | \lambda) \tag{3.9}$$

Backward variable is calculated as follows:

- Initialization

$$\beta_T(i) = 1 \tag{3.10}$$

- Recursion

$$\beta_t(i) = \sum_j P(O_{t+1} | q_{t+1} = S_j, \lambda) \tag{3.11}$$

Finding the state sequence problem is solved by using the Viterbi algorithm [35] which is based on dynamic programming. The algorithm finds a single state sequence. The algorithm recursively finds the optimal state sequence with backtracking at time $T$ by choosing the most probable state at each time step.

In order to find the most probable states, $\delta_t(i)$ is defined to represent the path with the highest probability at time $t$ that is calculated for the first $t$ observations. The last state of this path is $S_i$. After calculating these probabilities, the most probable path can be found by backtracking from the last time step $T$. $\psi_t(j)$ is defined to represent the state index $i$ that maximizes the multiplication of the previous step's $\delta_{t-1}(i)$ and $a_{ij}$.

Viterbi algorithm consists of three steps. The steps are given as follows:

- Initialization

$$\delta_1(i) = \pi_i b_i(O_1) \tag{3.12}$$

$$\psi_i(i) = 0 \tag{3.13}$$

- Recursion

$$\delta_t(j) = max_i \delta_{t-1}(i) a_{ij}.b_j(O_t) \tag{3.14}$$

$$\psi_t(j) = argmax_i \delta_{t-1}(i) a_{ij} \tag{3.15}$$

- Termination

$$p^* = max_i \delta_T(i) \tag{3.16}$$

$$q_t^* = argmax_i \delta_T(i) \tag{3.17}$$

Learning the model parameters problem is solved with Baum-Welch algorithm which is an EM algorithm. However, this is out of scope in this thesis.

### 3.2 Hierarchical Hidden Markov Models (HHMMs)

Hierarchical HMMs are proposed to represent each hidden state as a probabilistic method on its own [2]. The primary motivation of this structure is to model natural hierarchical structures better. Therefore, the HHMMs allow representing models in a hierarchical way. Furthermore, the models allow arbitrary number of activations in the sub-models.

In an HHMM, each hidden state has its own probabilistic model. Therefore, each state is another HHMM on its own. Beacuse of this property, states in an HHMM emit sequences instead of a single observation symbol. Once a state is activated in an HHMM, its probabilistic model is activated and transitions occur inside that sub-HMM. These transitions are called *horizontal transitions*. Recursively, a state in

that sub-model may also activate its sub-model. This is called a *vertical transition*. In a sub-model, when a special state is reached, control of the model passes to the upper level model. This special state is called *production state*. Hidden states that do not emit observations directly are called *inner states*.

A four level HHMM is given in Figure 3.4. Gray and black edges represent vertical and transitions. Dashed edges show the return of the control of the models to the upper level model. It is also assumed that all states can be reached by a finite number of states from the root state. This makes the model strongly connected. Note that it is also possible to represent an HHMM as a plain HMM. However, hierarchical structure would be absent in that representation.

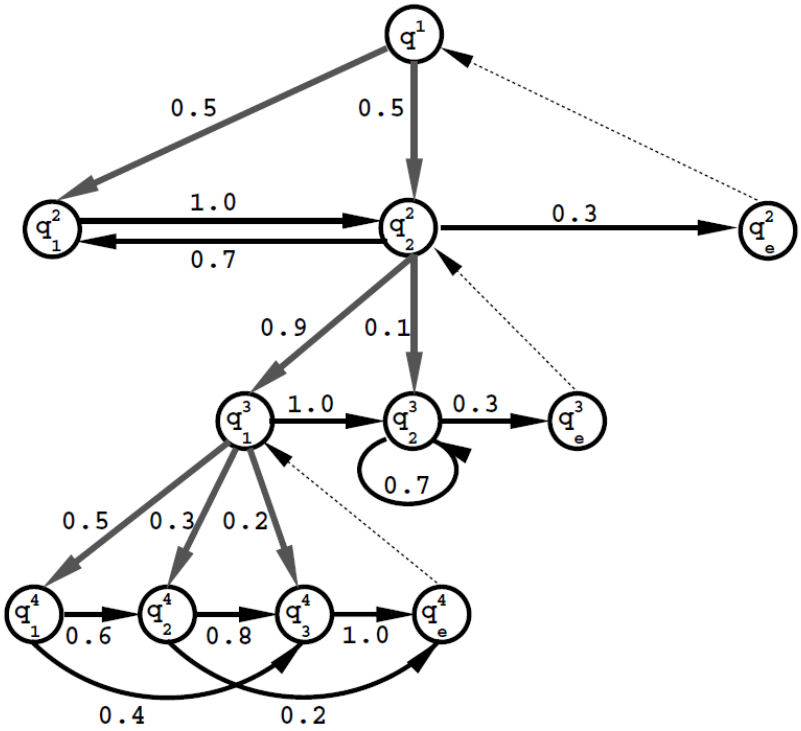


**Figure 3.4**: A four-level HHMM [2].

In this thesis, a similar idea of representing HMMs in a hierarchical way is adopted to the problem of failure isolation. Different failure types are modelled with hierarchy levels to isolate failures in a robotic planning domain. This representation will be explained in detail in the next section.

# 4. PROBABILISTIC FAILURE ISOLATION

In this chapter, a taxonomy of robot failures is presented. Afterwards, the proposed method for the failure isolation problem is explained.

## 4.1 A Taxonomy of Robot Failures in Cognitive Tasks

A taxonomy of robot failures in cognitive tasks [4] is given in Figure 4.1. The taxonomy presents two main categories, namely, external and internal failures. External failures are related to environmental issues whereas internal failures are more about the robot's internal state. Internal failures are divided into two subcategories as hardware and software failures. Hardware issues are about component failures of the robot. This branch includes five sub-branches: (1) Actuator/ effector failures (2) Sensor failures (3) Power supply failures (4) Lack of hardware resources. The actuators or the effectors of the robot may fail due to mechanical or electrical issues. Sensor failures may represent a state where sensory information is partially or completely lost. A robot may also fail to perform any tasks due to its dead battery. When the robot lacks the required hardware resources to execute an action, it may fail to achieve it. Assume that the robot is assigned a pick up action. If the object is too heavy for the robot to lift, the robot may fail to achieve this task. The other branch of internal failures -software failures- include three main categories, namely, localization, sensor and general failures. Localization failures are about errors in localization. While sensor failures are about incomplete sensor model, general failures are about other failures which occur due to the other software issues of the robot.

The second main category of failures includes external failures. A similar classification for external failures is given in an earlier work [36]. The taxonomy presents five sub-branches for external failures: (1) Communication failures (2) Misbeliefs on the facts and the rules in the KB (3) Lack of knowledge (4) Conflicting goals (5) Impossible goals.

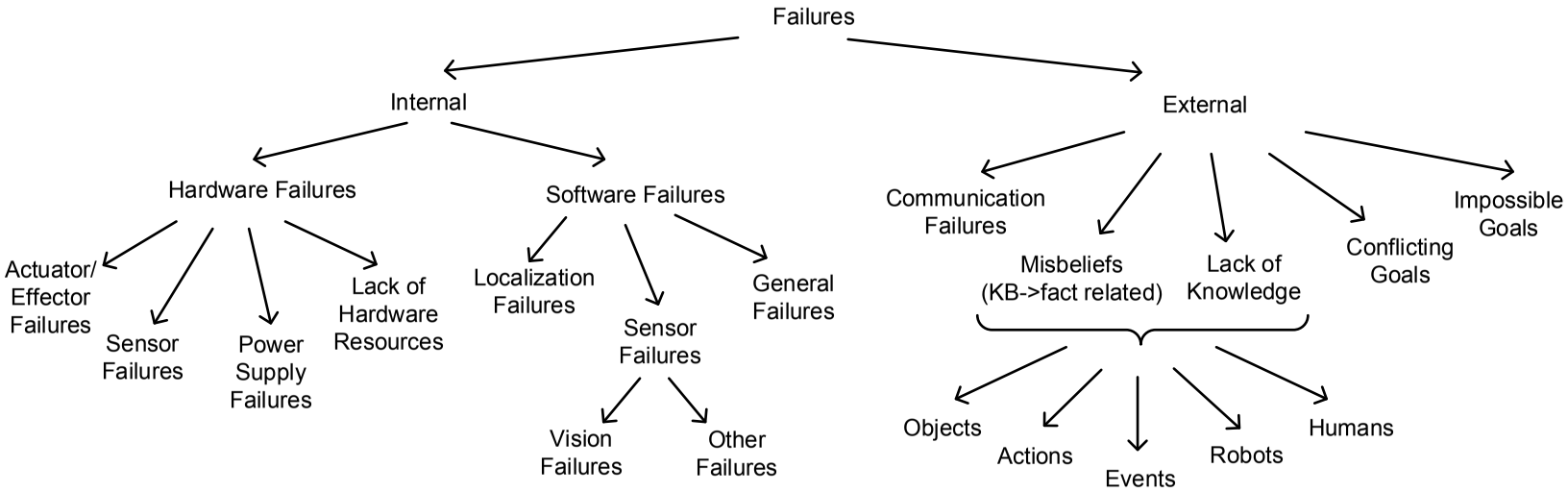


**Figure 4.1**: A taxonomy of robot failures in cognitive tasks.

Communication failures occur when the robot is not able to communicate with other robots or with an operation center in the environment. The same instance may occur when robots collaborate with each other through communication. The robot may also have misbeliefs in its KB of the pre/post conditions of actions or events. That may result in failures in achieving tasks. Action-related failures may occur due to incomplete representation of actions in the KB. For instance, actions may lead to unexpected outcomes in the environment. Similarly, the robot may not be aware of some events and their effects in the environment. Object-related failures arise from false beliefs about an object's existence, location or its attributes. These types of failures may arise from visual features (e.g., SIFT, VPFH etc.) as well as the mobility-related features such as weight, grasp position and center of mass (COM). For example, when the robot tries to pick up an object, it may detect that the object is wider than the size of its gripper. However, if the robot detects that the object has a handler, it may recover from the failure by grasping it from its handler. Misbeliefs about other robots may cause timing problems and this leads to failures especially in cooperation. In such a case, when the other robot delays or somehow fails, the robot may probably fail. The robot may also fail if it has misbeliefs about humans in the environment. When a domestic service robot has erroneous preference model for its companion, the robot may fail in achieving the desired goal. Lack of knowledge about objects, actions and events may also result in failures. Robots may also fail in achieving their tasks when their goals conflict with each other. Finally, when the robot detects that its goal is impossible to achieve, there is no way to attain the given objective. For instance, if the goal of the robot is going outside of a room without a door, the robot will not be able to reach its goal under this circumstance.

Note that failures in path planning are not presented in this taxonomy since failures in higher level cognitive tasks are addressed. Although there seems that failure types are distinct from each other, simultaneous failure causes or chain of events may result in failures.

These failure modes can also be classified under two main categories: complete and partial failures. Complete failures model the situations where the robot is not able to execute any actions. Assume that the software system of the robot completely crashes. In this case, the robot may not even be able to find out the reason of the failure. On the

other hand, whenever a partial failure occur (e.g., a sensor failure) the robot may find a way to handle the failure.

In this thesis, vision failures, localization failures, misbeliefs about external events and actuator/effector failures are handled. In the next section, representation of these failure types within a model based system is presented.

## 4.2 HHMM-based Failure Isolation

In the proposed method [37, 38], the type of a failure that may occur during the execution of an action is modelled as a distinct HHMM. In this representation, two-level HMMs are employed where parent states -top level HMMs- correspond to actions in a given plan. Lower level HMMs corresponds to execution states of an action (Figure 4.2). Each model is denoted by $M_i$ where $i$ is the index of the corresponding failure type. Causes of failures that are addressed in this thesis are: *vision failures for all objects*, *localization failures*, *hardware limitations (gripper)* and *external events for all objects*. *Vision(X)* failure model for a specific *object(X)* represents a faulty situation in the vision algorithm for that object (i.e., error on detection the object itself or its relations). Model *HardwareLimitation* indicates the situations that are beyond the capabilities of the robot. *ExternalEvent* stands for external events that change the world outside the control of the robot. *Localization* failures model the faulty situations where the robot cannot localize itself. Depending on the action that is being executed at a given world state and its parameters, the related failure models are activated. In both levels, each state has a latent variable which is either *success* or *failure*. Therefore, there are two hidden states in that representation. During the execution of an action in the plan, a lower level HMM is activated under the action state, and its model is updated. When the execution of an action ends, this lower level HMM is used to return a result to the upper level of the corresponding HHMM. The problem is to find models that include latent variables with the value *failure* and have failure probabilities over a given threshold. A generic illustration for the proposed method is given in Figure 4.2.

Hierarchical HMMs are used to represent failure models instead of classical HMMs because a hierarchy between the plan, its actions and its execution states is needed in order to isolate persistent failures on a specific object or event. For instance, the robot's

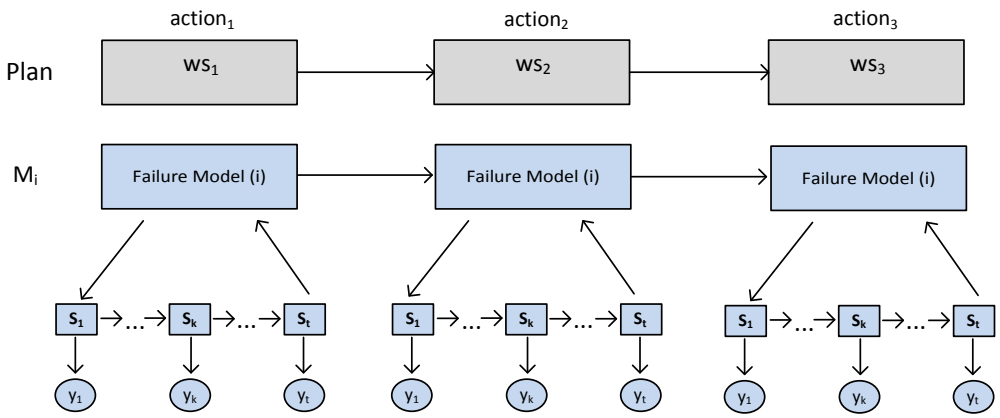


**Figure 4.2**: An illustration of an HHMM failure model corresponding to the actions of a given plan.

vision system may fail to recognize a specific object in the environment all the time. In such cases, faulty model can be easily separated from the rest of the models.

The relations that define which action is related to which failure type are given in Table 4.1 for a specific *object(objX)* called object *A*. Using the relations listed in Table 4.1, only related models are updated during the execution of an action. Therefore, irrelevant failure models are not updated during the execution of an action. For example, according to the table, during the execution *put-down(objA)*, only the gripper failure model is activated where other models become passive.

**Table 4.1**: Action-Failure type relations used in the model.

| *Action* | *Failure Type* |
|---|---|
| move-to-obj(objX) | Vision(objX), ExternalEvent(objX)<br>Localization(Robot) |
| move-to-loc(destination) | Localization(Robot) |
| pick-up(objX) | ExternalEvent(objX), Gripper(Robot)<br>Vision(objX), Localization(Robot) |
| put-down(objX) | Gripper(Robot) |

At the very beginning of the execution of the plan, the first state for each failure type is activated considering observations in the initial state. During the execution of each action, if that action and a failure type are related, the corresponding failure type's HHMM is activated for that time step, and this model is considered as an *active model*. Perceptions are gathered from the sensors of the robot (e.g., pressure sensor, RGB-D sensor, etc.) and these observations are mapped into a probability distribution which

depends on the statistical analysis of the sensory input. Depending on the observations gathered at the current state, the new state's contents are calculated considering the previous state with the following equations where $v_t$ is the Viterbi value which is calculated for each time step $t$, and $\delta_t$ is the state index that maximizes the given statement.

$$v_t(S_i) = max_k(v_{t-1}(k) * a(k, S_i)) * b_{S_i}(O_t) \tag{4.1}$$

$$\delta_t(S_i) = argmax_k(v_{t-1}(k) * a(k, S_i)) \tag{4.2}$$

If a failure model is not related to the action in execution, it is considered as a *passive model*, and the last state of the corresponding HHMM's time interval is augmented in such a way that the corresponding state also includes that time step without generating a new state. This procedure goes on until the robot reaches the goal state or a failure is detected. In case of a failure, the Viterbi algorithm [35] is invoked to label latent variables with either *success* or *failure* in each HHMM.

**Algorithm 1** FailureIsolation($P$,$M$)

```
Input: Plan P, failure models M, Action-Failure type relations R
Output: list, the list of the candidate causes of a failure
while P != ∅ and status == success do
    action = POP(P)
    activeModels = extractActionFailureRelations(action, R)
    while (status = execute(action)) == inExecution do
        predicateList = observeTheScene()
        probabilityList = calculateObservationProbabilities(predicateList, activeModels)
        updateLowerLevelModels(M, activeModels, probabilityList)
    end while
    labelLowerLevelLatents(M, t)
    transitToUpperLevel(M, t)
    updateModels(M, activeModels) // Algorithm 2
end while
for all M_i do
    labelUpperLevelLatents(M_i)
end for
list = isolateModels(M) // Algorithm 3
return list
```

Algorithms 1-3 are presented to summarize the proposed failure isolation approach. Algorithm 1 accepts a plan, failure models and action-failure type relations as

parameters. Plan $P$ includes a sequence of actions to be executed by the robot. An action can be in one of the following states: *inExecution*, *success* and *failure*. *inExecution* corresponds to the state in which the robot executes an action. *success* stands for the state that the robot reached the expected outcomes of a corresponding action. *failure* corresponds to the situations where the expected outcomes of an action are not met.

**Algorithm 2** updateModels($M, activeModels$)

```
Input: Failure models M, extracted related failure models activeModels
Output: M_i is updated according to observations
for all M_i do
    if isRelated(M_i, activeModels) then
        addNewState(M_i)
    else
        extendPreviousState(M_i)
    end if
end for
```

First, an action is popped from the plan which is generated by the planner. According to the action and its parameters, predefined action-failure type relations are used to extract *active models* from the failure model set. Other failure models that are irrelevant with the action to be executed, are labelled as *passive model*. With executing *extractActionFailureRelations(action,R)* action, active and passive models are defined. Once the execution of the action is started, the robot observes the world state through its sensors and related predicates are extracted with the *observeTheScene()* method. Using *calculateObservationProbabilities(predicateList,activeModels)* procedure, this world state is mapped into a probability distribution in the Conditional Probability Tables (CPTs). After calculating all probabilities for active models, all related failure models' lower levels are updated with *updateLowerLevelModels(M,activeModels,probabilityList)*. Note that passive models are not updated since they are eliminated from the current action's time interval. When the action comes to an end, Viterbi algorithm [35] is invoked in the lower level updated models with the *labelLowerLevelLatents(M,t)* procedure. By running the Viterbi algorithm, each HHMM's lower levels' hidden states for that executed action are labelled with success or failure. A transition to the upper level occurs with the *transitToUpperLevel(M,t)* function by using calculated maximum failure probability in the related model's lower level for the corresponding

action's time interval. That maximum value is passed to the upper level with *transitToUpperLevel(M,t)* model. At the end of the Algorithm 1, a list that contains the possible causes of the failure with a probability exceeding a given threshold is returned with *isolateModels(M)*. The content of this procedure is explained in Algorithm 3.

Algorithm 2 explains the update procedure of upper level models. In Algorithm 1, active models are extracted considering predefined action-failure type relations, and this list is used as a parameter of Algorithm 2. For all active models, a new state is generated considering Equations 4.2-4.1, and the context of the newly generated state are updated. Otherwise, if a failure model $M_i$ is not in the active model list, the previous state's time interval is extended without generating a new state.

**Algorithm 3** isolateModels($M$)

```
Input: M, Failure models
Output: list, the list of the candidate causes of a failure
list = ∅
for all M_i do
    for all ws_j do
        if M_i.getHiddenState(ws_j) == failure and M_i.getProbability(ws_j) ≥ threshold then
            list.add(M_i)
            break
        end if
    end for
end for
return list
```

Algorithm 3 returns a list that contains candidate causes of a failure. In order to find this list, all the models are examined, and probability and latent variable values are considered. If a state is labelled with *failure* and its normalized probability value is over a given threshold, this model is added to the cause list. The threshold is predefined as 0.5 since there are two hidden states in the models. Otherwise, the model is not added to the cause list. At the end of the Algorithm 3, the list is returned to the Algorithm 1.

### 4.2.1 Model parameters

In this section, parameters for the HMMs are presented for the proposed method. First, world state definition (observations) are given. Then, transition and observation models are presented, respectively.

#### 4.2.1.1 World state definition

The world state of the robot is maintained by using sensory and motor information of the robot. Table 4.2 lists the observable predicates of the world state that are considered and the related sensors providing the relevant data.

*object* predicate is for representing the verified existence of an object. *clear* is for stating that the object has nothing on it. *onground* stands for the situation that an object is on the ground. *segment* corresponds to a point cloud clustered in the scene by the segmentation algorithm but not detected by the vision algorithm. *isConflicted* defines that the vision algorithm simultaneously detected more than one object on the same location. *inconsistentObjectModel* states that the vision algorithm's output and the segmentation algorithm's output conflict for a specific object. These predicates are observed with the on-board RGB-D camera using a template-based vision algorithm in the system.

*handempty* states that the robot's gripper is available to hold an object whereas *holding* is for stating that an object is grasped by the robot. *offset* represents the offset of the robot's current location considering the feature based localization algorithm and the robot's odometer. Observation probabilities are defined regarding to the statistical analysis on the outputs of the sensors and the related predicate computations. A scene interpretation [3] approach maintains these predicates in the robot's knowledge base, and updates them according to new observations.

**Table 4.2**: Predicates and the related sensory data.

| Predicate | Source |
|---|---|
| object | RGB-D Camera |
| clear | RGB-D Camera |
| onground | RGB-D Camera |
| segment | RGB-D Camera |
| isConflicted | RGB-D Camera |
| inconsistentObjectModel | RDB-D Camera |
| handempty | Pressure sensor |
| holding | Pressure sensor |
| offset | Laser Range Finder |

#### 4.2.1.2 Transition probabilities

Since the model has two hidden states, transition probabilities that define the probability of transferring from one hidden state to another should be determined.

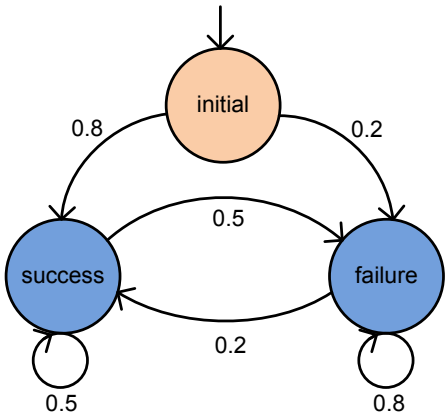


**Figure 4.3**: Transition probabilities of the proposed method.

In Figure 4.3, initial state distribution and hidden states are illustrated. Initially, probability of being at state *success* is 0.8 where probability of being at state *failure* is 0.2 since it is assumed that the system is in a success state. In a *success* state, probability of staying in the same state and passing through the other state is uniformly distributed. However, since it is more probable to stay in *failure* state when the robot is already in *failure* state, the probability of transferring from state *failure* to state *failure* is higher than the probability of transferring from state *failure* to state *success*. Therefore, this probability is defined as 0.8 where transferring from state *failure* to state *success* is lower with value 0.2.

#### 4.2.1.3 Observation probabilities

Observation probabilities are the components of HMMs that defines the probability of observing an observation in a hidden state $s_i$. Observation probabilities are defined as *Conditional Probability Tables (CPTs)* for each failure type.

Note that in the hierarchical structure, all actions have different observations in each model for different failure types. Therefore, for different failure models, observation probabilities should be defined.

Observation models of the proposed method are defined considering the sensory outputs and statistical information. These probabilities will be explained in detail in this section.

**Conditional Probability Table for the Vision Model**

CPT for the vision model is constructed considering the outputs of the vision algorithm (Table 4.3). For instance, if an object is detected with the RGB-D camera of the robot, the vision algorithm gives a similarity value considering the template of the object. This value is used as a probability value for that model. The output of the segmentation algorithm is also used in the CPT. If an object is not detected with the vision algorithm, the output of the segmentation algorithm is used. If a point cloud is detected by the segmentation algorithm, it is more probable that the vision algorithm failed to detect the object since the performance of the segmentation algorithm is better.

Furthermore, it is also possible that the vision algorithm may falsely classify an object. In these cases, if the detected size by the vision algorithm and the segmentation algorithm is quite different, *inconsistentObjModel(objX)* predicate is observed. This situation yields to a failure situation more probably. Furthermore, the vision algorithm may simultaneously detect two different objects in the same location. This situation is represented as another row in the CPT. Corresponding probabilities can be seen in Table 4.3. In the CPT, $s_{success}$ and $s_{failure}$ rows denote the probability of observing the corresponding observations in the rows in a success and failure state respectively.

**Table 4.3**: CPT for the vision model.

| Observation | $s_{success}$ | $s_{failure}$ |
|---|---|---|
| object(objX) | similarity(objX) | 1 - similarity(objX) |
| ¬object(objX) ∧ segment(objX) | 0.3 | 0.7 |
| ¬object(objX) ∧ ¬segment(objX) | 0.7 | 0.3 |
| inconsistentObjModel(objX) | 0.45 | 0.55 |
| isConflicted(objX) | 0.65 | 0.35 |

**Conditional Probability Table for the External Event Model**

CPT for the external model is also constructed considering the outputs of the vision algorithm (Table 4.4). Situations where the robot detects an object use the similarity measure as well. Other situations where the object is not detected by the vision

algorithm but detected with the segmentation algorithm is defined in Table 4.4. Since the segmentation algorithm is more accurate, probabilities are defined accordingly.

**Table 4.4**: CPT for the external event model.

| Observation | $s_{success}$ | $s_{failure}$ |
|---|---|---|
| object(objX) | similarity(objX) | 1 - similarity(objX) |
| ¬object(objX) ∧ segment(objX) | 0.8 | 0.2 |
| ¬object(objX) ∧ ¬segment(objX) | 0.35 | 0.65 |

According to the table, if the object is not recognized by the vision algorithm but the segmentation algorithm clusters a segment in the object's expected position, this situation is more likely a vision failure and the probability of gathering this observation in a success state is defined as 0.8. Probability of gathering this observation in a failure state is defined as 0.2 as well. If both of the algorithms are not able to find a clue about an object, probability of gathering this observation in a failure state is defined as 0.65.

**Conditional Probability Table for the Gripper Model**

CPT for the gripper model is defined in Table 4.5. The parameters of this model is tuned by empirical robot experiments. Several experiments take place in order to define the parameters of this model. Table 4.5 shows the following probabilities for the gripper model.

**Table 4.5**: CPT for the gripper model.

| Observation | $s_{success}$ | $s_{failure}$ |
|---|---|---|
| holding(robot,objX) | 0.75 | 0.25 |
| handempty(robot) | 0.55 | 0.45 |

In the table, it is seen that a fix probability is defined for the situations that the robot can not pick an object up. However, the difference between these two probabilities is not high since it is not obvious that the gripper is faulty. Therefore, the probability of observing *handempty(robot)* after a *pick-up* action in a success state is 0.55. Similarly, for the action *put-down*, same probability values in the CPT are used considering the natures of the action *pick-up* and *put-down*.

### Conditional Probability Table for the Localization Model

CPT for the localization model is defined (Table 4.6) considering the offset between the odometer and landmark based localization algorithm. Error in the localization is defined as follows [39]:

$Error = \frac{|expectedLocation - actualLocation|}{totalDistanceTravelled}$

The robot's localization error value is investigated in both faulty scenarios and normal scenarios. The error ratio defined above is observed between 0.05 and 0.35. Therefore, error value is normalized using 2.86 as a normalizing factor (1/0.35 = 2.86). According to this investigation, error value is mapped into a probability distribution.

**Table 4.6**: CPT for the localization model.

| Observation | $s_{success}$ | $s_{failure}$ |
|---|---|---|
| Error | 1 - (Error*2.86) | (Error*2.86) |

## 4.3 A Running Example

To clarify the proposed method, an object manipulation scenario is illustrated in Figure 4.4. On the top of the figure, the plan generated by the planner is given. In this scenario, only vision failure model for object A, vision failure model for object B, hardware limitation (particularly for the gripper) for the robot and localization failure models are shown.

All of the models are initialized in the beginning. The robot first executes action*move-to-obj(A)*. During the execution of action *move-to-obj(A)*, the localization failure model and the vision failure model for object A are updated since they are related to the action. However, the hardware failure model and the vision failure model for object B are not updated since they are not related to the action in execution.

The next action to be executed in the plan is action *pick-up(A)*. Since this action is related to the vision model for object A, hardware limitation and localization models, these models are updated during the execution. These models are labelled as *active models*. Vision failure model for object B is not updated since the executed action is irrelevant for object B. These models are labelled as *passive models*. At the end of each action, the corresponding lower level HMM is finalized, a transition is taken,

and the upper level HMM is updated. Sequentially, an action is popped from the plan and update procedure for the contents of the states are done. Action-failure relations are checked considering the parameters and active models are found. Similarly, these operations are applied for all actions in the given plan.

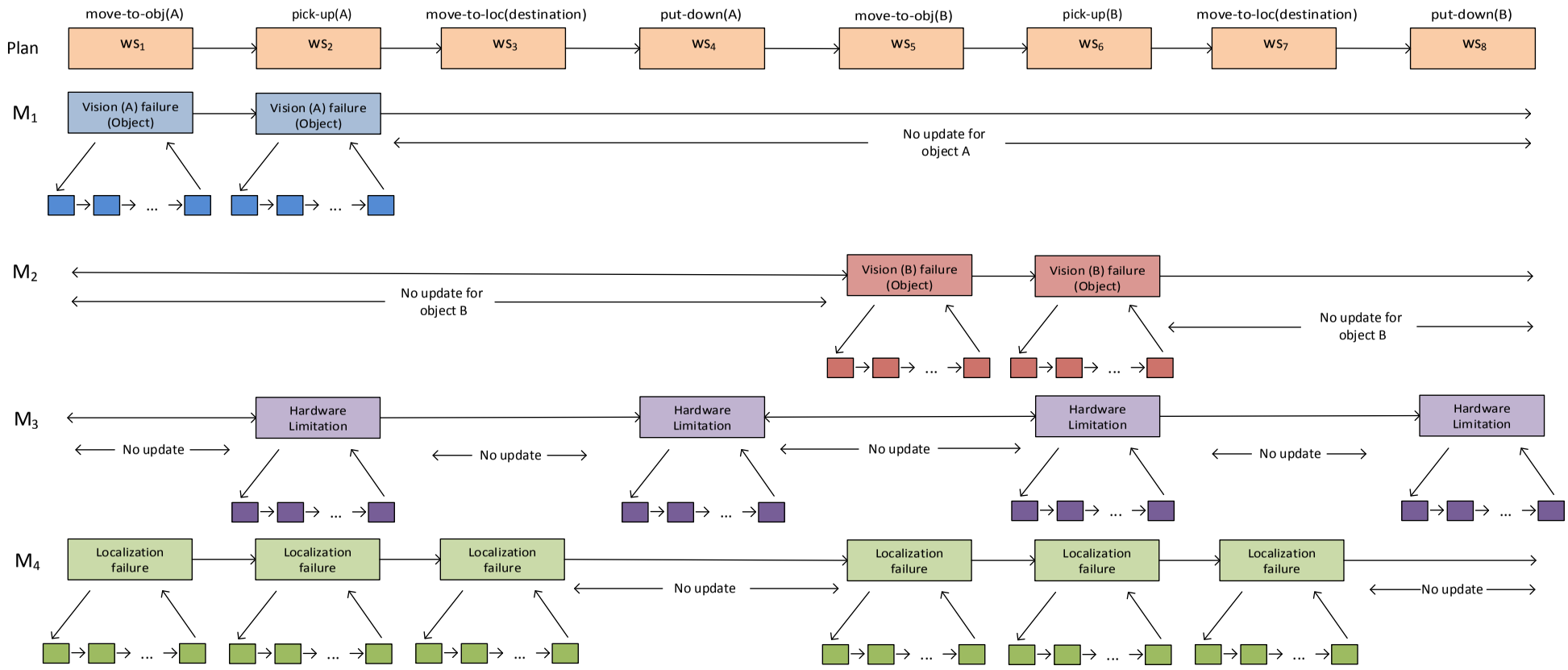


**Figure 4.4**: An example model for a manipulation scenario. On the top, the plan generated by the planner is illustrated. In the lower level, hierarchical failure models are given.

# 5. EXPERIMENTS

In this chapter, experimental setup is first presented. Then, the evaluation of the system on real robot scenarios is analysed.

## 5.1 Experimental Setup

In the experiments, Pioneer 3-AT (Figure 5.1) robot is equipped with a gripper, a sonar ring, and an RGB-D camera is used. Objects are recognized using LINEMOD algorithm [40,41] with RGB-D camera which gathers depth information of the objects with color information (Figure 5.2). Templates of the objects are stored in the knowledge base (KB) of the robot and during the execution of a task, point clouds from the scene are compared with the templates of the robot. If a point cloud is matched with a template, object is recognized with a similarity measure.

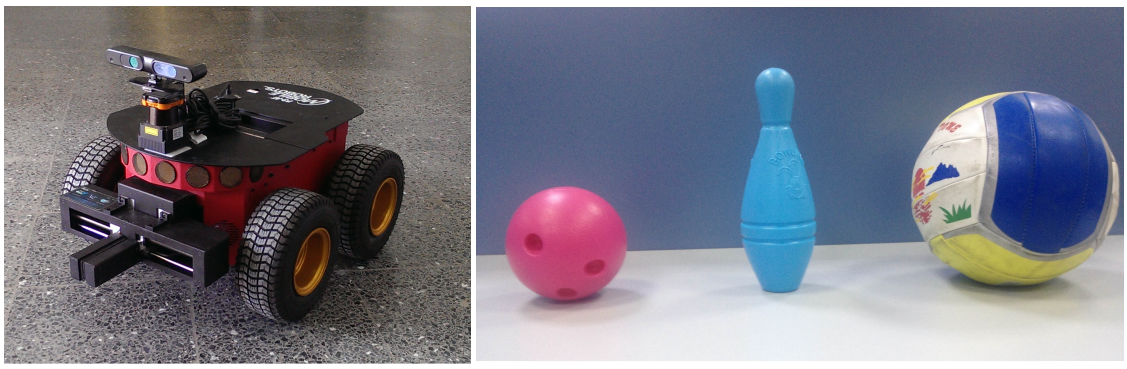

**Figure 5.1**: (Left) Pioneer 3-AT robot; (right) objects that are used in the experiments.

A depth-based segmentation algorithm is also run for detecting point clusters (segments) of the scene beside the template-based object recognition algorithm. Organized Point Cloud Segmentation [42] is used to extract clusters/segments from the scene gathered with the RGB-D camera (Figure 5.2). Clusters that are larger or smaller than given a threshold are discarded. Note that segments clustered with the segmentation algorithm are classified as unknown objects, these segments do not have an object type.

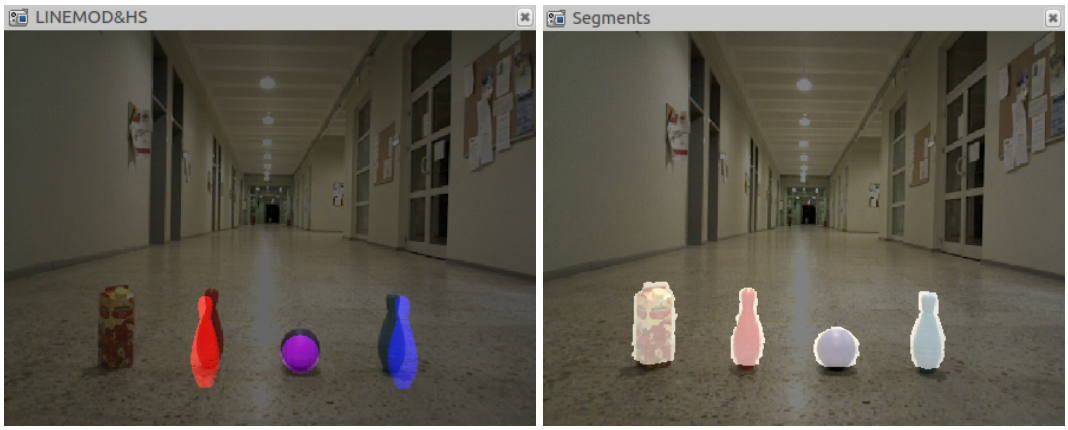


**Figure 5.2**: (Left) Outputs of the LINEMOD and (right) segmentation algorithms [3].

Experiments take place in a laboratory environment that contains objects placed on the ground. The experiments are performed on three objects: a blue plastic bowling pin, a beach volleyball, and a small plastic purple ball (Figure 5.1).

Given an initial and goal state, the robot takes consecutive actions in order to reach the goal. In the symbolic level, the robot maintains the models of its operators corresponding to the *actions* that it can execute in the real world. Each action is represented by a set of facts to be satisfied before executing it, namely, preconditions, and effects that occur in the world after the action is executed by the robot. In an object manipulation scenario, the robot's goal is to transport objects to a destination. Initially, all objects are in their initial positions. The robot is capable of executing some actions namely *move-to-loc*, *pick-up*, *put-down* and *move-to-obj*. Action *move-to-loc* is executed in order to move the robot to a location, whereas action *move-to-obj* moves the robot to a desired object location. Action *pick-up* is to pick up an object by the robot's gripper. Similarly, action *put-down* is executed for releasing an object from the gripper on the ground.

## 5.2 Experimental Results

In this section, the performance of the proposed method is analysed. First, some case scenarios and the corresponding probabilities are investigated. Afterwards, the performance analysis of the proposed method is given on a more extended set of experiments. Then, results of the failure isolation problem using Particle Filters are given. At the end, memory analysis of the proposed method is investigated.

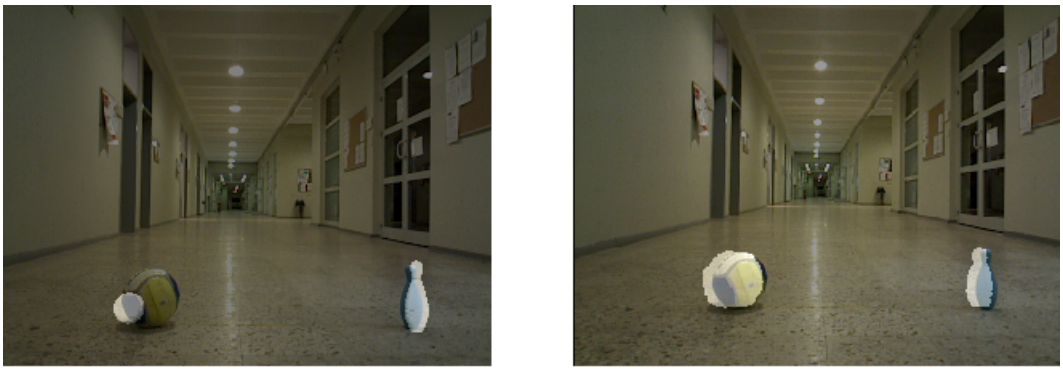

**Figure 5.3**: (Left) The vision algorithm detects a small ball on the boundary of the big ball. (Right) The segmentation algorithm correctly clusters the point cloud of the big ball.

### 5.2.1 Case scenarios

The performance of the proposed method is first analysed on some case scenarios.

In vision failure scenarios, object *B* is misrecognised by the vision algorithm due to the missing template model of the object. Therefore, the vision algorithm classified the volleyball as a small ball. In gripper failure scenarios, the controller of the robot is changed in such a way that the robot is not able to close its gripper. In external event scenarios, object *A* is taken out of the scene by a human from the environment during runtime. In localization failure scenarios, the robot's location is manipulated by changing its position during the execution of an action.

#### 5.2.1.1 Scenario I

In this scenario, the robot is tasked to move the pin (*objA*) and the beach ball (*objB*) to their desired destinations. Although the robot can correctly localize the pin, it fails in correctly recognizing the ball. Since the vision algorithm searches the objects by considering their surface normals, it falsely detects a small ball on the left boundary of the ball (Figure 5.3) due to the similarity of the objects' surface normals. This affects the parameters of the action *pick-up*. Therefore, the robot fails in picking up the ball with its gripper. After the failure, the robot is intentionally allowed to retry the action three more times. Since the pin is successfully recognized, action *pick-up* on this object is taken without any failure. The robot directly tries to move the object to the destination and puts it down.

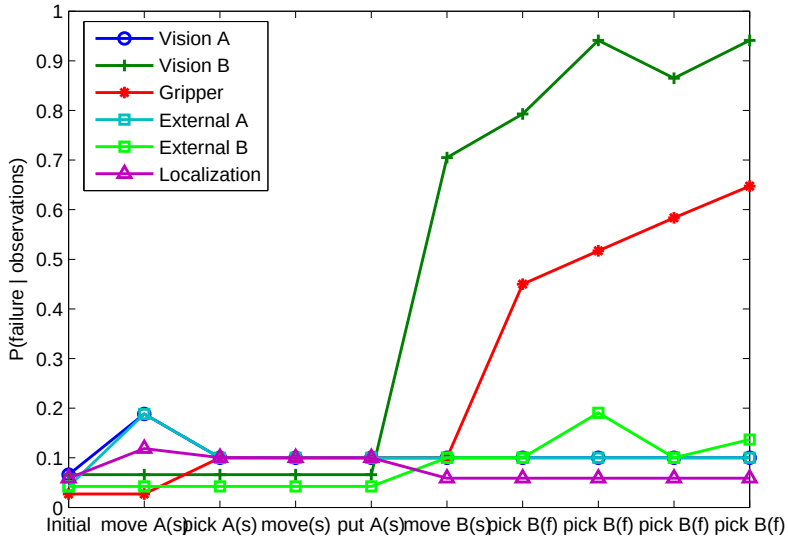


**Figure 5.4**: A consistent vision failure for a particular object results in an increase in the corresponding probability.

Figure 5.4 illustrates how the posteriors on failure models change during the execution of each action. Six failure types are considered. As clearly seen from the figure, during the execution of action *move-to-obj(A)* and *pick-up(A)*, vision failure is not observed, therefore, the probabilities are stable during these time steps. Note that in the plots, letters $s$ and $f$ next to the actions represent the action is succeeded or failed, respectively.

The vision failure probability for $B$ starts to increase after the execution of action *move-to-obj(B)* due to the conflicts in the results of the vision algorithm and the detected sizes of the object by segmentation. This situation may result with setting the parameter of the action *pick-up* wrong. Note that even when there is this type of vision failure, it is possible to successfully pick up the object if the center of recognition intersects with the center of the original object, but this is not the case in this scenario.

It is also seen from the figure that while the robot attempts to pick up $B$, the probability of having a hardware failure (gripper) increases due to the propagation of the failure over time.

#### 5.2.1.2 Scenario II

To test the performance of the system to detect a hardware malfunction, a hand-made gripper failure is injected to the robot. This is simulated by preventing the gripper from

closing completely and by closing the sensory feedback in software to detect the state of the gripper.

In the beginning of the execution, the planner generates a plan in order to transfer the objects to their desired locations. The planner generates a plan as follows: {move-to-obj(A) - pick-up(A) - move-to-loc(dest) - put-down(A) - move-to-obj(B) - pick-up(B) - move-to-loc(dest) - put-down(B)}.

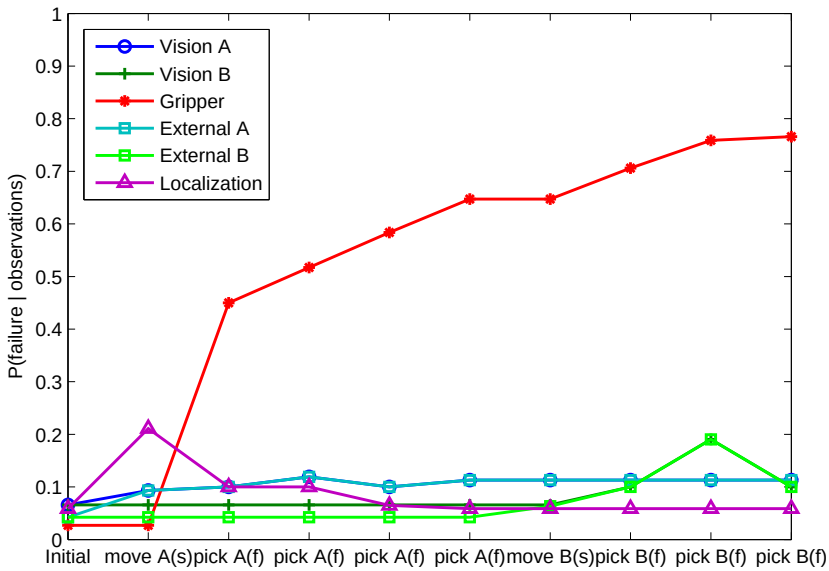


**Figure 5.5**: A hand-made gripper failure is injected into the system. Probabilities change accordingly.

The robot tries to pick up the plastic pin (*objA*) and the small ball (*objB*) but fails all the time for both objects due to the simulated failure in its gripper. Resulting probability values can be seen in the Figure 5.5. According to the figure, the robot's belief on gripper fault increases for each trial.

Other models' probability values are stable during the execution of the given plan since no indication of a failure is observed. Only small changes in the localization model occurs due to the increase in the offset of the robot considering its odometer and feature based localization algorithm. However, this situation is fixed in future steps of the plan. At the end of execution, it can be seen from the figure that probability of having a gripper failure is 0.76. Other models' probability values are stable during the execution of the plan. As a result, the method proposes the gripper model as the cause of the failure.

#### 5.2.1.3 Scenario III

In this scenario, failures due to external events are analysed. For this purpose, the object is manipulated by a human. Two objects (the pin and the small ball) are used and the pin (*objA*) is taken out of the environment by the human while the robot moves towards that object.

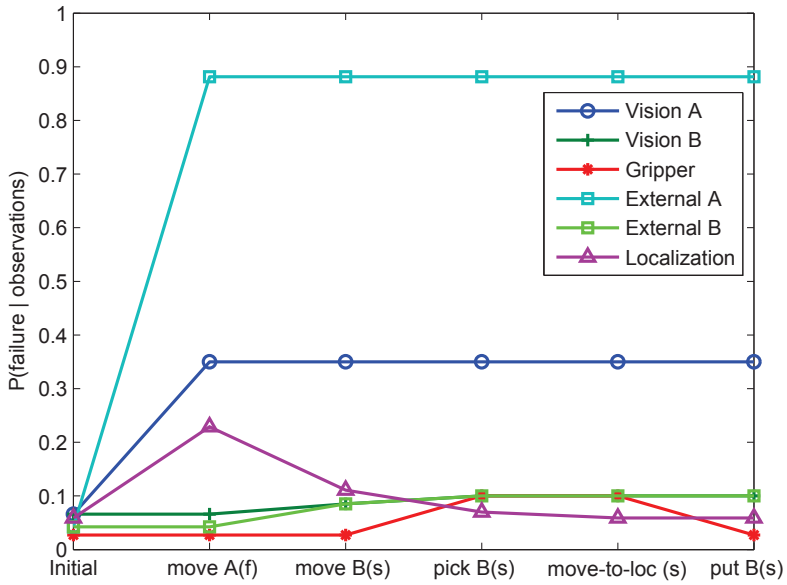


**Figure 5.6**: The resulting probabilities when an external event exists in the scene.

In order to collect the object in their desired positions, the planner generates a sequence of actions to be executed. The planner generates a plan as follows: {move-to-obj(A) - pick-up(A) - move-to-loc(dest) - put-down(A) - move-to-obj(B) - pick-up(B) - move-to-loc(dest) - put-down(B)}.

Since the object is taken out of the environment, action *move-to-obj(objA)* fails. According to Figure 5.6, the robot increases its belief on an external event when the object disappears from the robot's view. The segmentation algorithm's output also suggests this hypothesis. However, the vision failure for the object *A* is also in one of the likely hypotheses.

During the execution of action *move-to-obj(A)*, belief on a localization failure is increased due to the offset between the feature based localization algorithm and the odometer. However, this is fixed in the future steps of the plan execution.

#### 5.2.1.4 Scenario IV

In this scenario, failures due to the situations where the robot can not localize itself is presented. During the execution of *move-to-obj(A)* action, the robot is moved to another location by a human. A plastic pin (*objA*) and a small ball (*objB*) is used in this experiment.

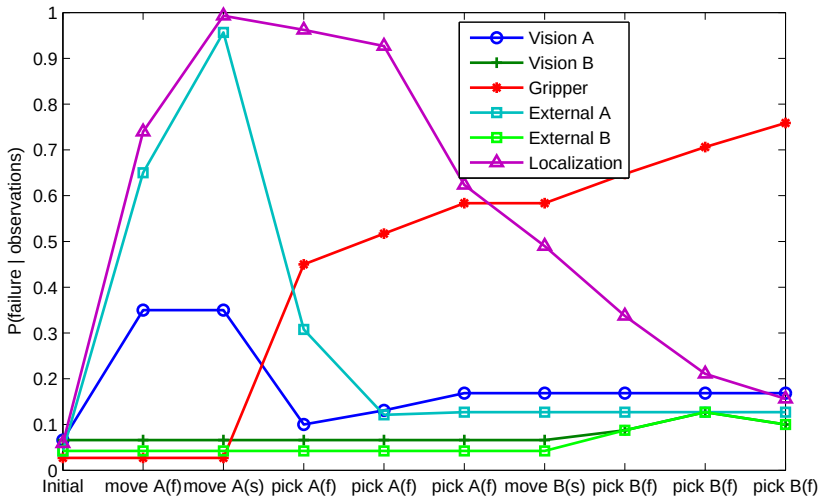


**Figure 5.7**: The resulting probabilities when a localization failure occurs.

According to Figure 5.7, the robot increases its belief on a localization failure due to the offset between the odometer and localization algorithm due to the increase in the offset of the robot that the localization algorithm returns. This offset causes an increase in the corresponding probability during the execution of actions *move-to-obj(A)* and *pick(A)*. However, this posterior drops later since the robot tries to fix its localization error during execution. Probability of a malfunction in the gripper is also possible since there is a fixed probability for each trial when the robot can not pick the object up with its gripper. The robot also increases its belief on an external event when its location is changed due to the disappearance of the object from the point of view. However, this posterior is dropped when the object is detected by the robot later.

#### 5.2.1.5 Scenario V

In this scenario, two objects are used. The first object is taken out of the environment during the robot moves towards it. The other object is misrecognised by the vision algorithm. The resulting probabilities can be seen in Figure 5.8.

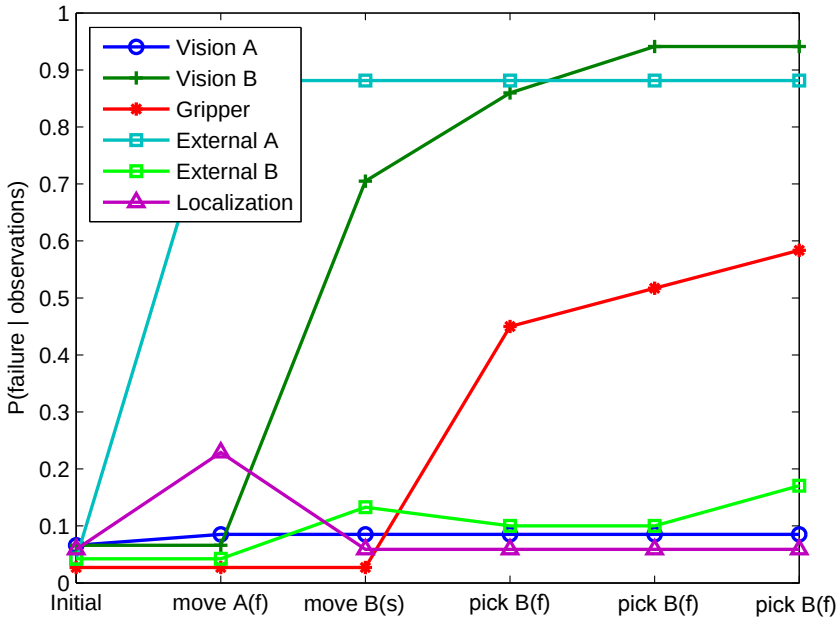


**Figure 5.8**: Both an external event and vision failure occur in the environment. Probabilities change accordingly.

The robot can not detect object *A* while executing action *move-to-obj(A)*. Therefore, the belief on an external event on this object increases. Then, the robot starts to execute the actions related to object *B*. Since the sizes of the object detected by the vision algorithm and segmentation algorithm conflict, the belief of a vision failure on object *A* increases. This scenario shows that the HHMM-based failure isolation method can track more than one failure since the failure models are independent from each other.

#### 5.2.1.6 Scenario VI

In this scenario, a complex scenario with three objects and multiple failures is investigated. The first object (*objA*) is taken out of the environment while the robot moves towards the object. A gripper failure is also injected to the software of the robot. Moreover, the third object (*objC*) has a vision failure.

Figure 5.9 shows the changes in the calculated probability values during the execution of the given plan. During the robot executes action *pick-up(A)*, the belief on an external event increases since the robot can not detect the object with the vision algorithm. Vision failure probability on that object also increases since this situation may be

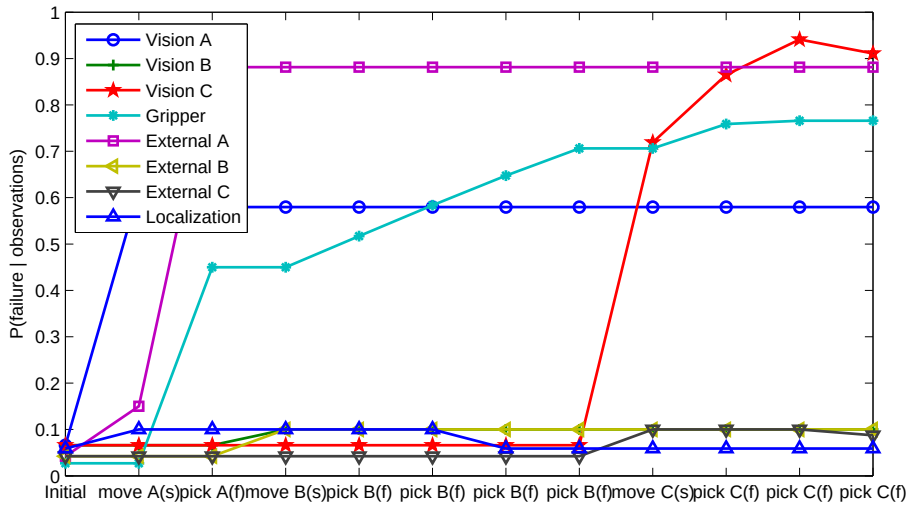


**Figure 5.9**: An external event, a gripper failure and a vision failure occur in the environment.

because of an error on the vision algorithm. For each trial of action *pick-up*, the belief of a gripper failure also increases.

### 5.2.2 Performance evaluation

Performance of the proposed method is analysed on several failure scenarios. 40 real robot scenarios are investigated. Each of 10 scenarios include vision failure on an object, gripper failure, external event on an object and localization failure.

Since the proposed method uses a hierarchy, the following three tables show the analysis of different probability transfer strategies from the lower level models to upper level models.

#### 5.2.2.1 Performance analysis of the method with maximum failure probability transfer values

In Table 5.1, each row represents the means ($\mu$) of the probability values and the standard deviations ($\sigma$) for each failure model in the corresponding 10 scenarios. For example, in the first row, a vision failure on object $B$ is investigated. According to the table, the proposed method suggests that the cause of the failure in the first 10 scenarios is due to a vision failure on the object $A$ with 0.11 probability on average, a vision failure on the object $B$ with 0.93 probability, a gripper failure with 0.61 probability, an external event on object $A$ with 0.12 probability, an external event on object $B$

with 0.15 probability and localization failure with 0.10 probability. Standard deviation values are 0.01, 0.02, 0.05, 0.02, 0.03 and 0.09 for the failure models, respectively. Similarly, the other rows represent the means of the probability values ($\mu$) and the standard deviations ($\sigma$) of each of the 10 scenarios that the proposed method suggests for each failure type. Standard deviations of vision models are high for some failure cases (in gripper scenarios for Vision A and in localization scenarios for Vision B) since there are also some vision failures in these scenarios.

**Table 5.1**: Performance analysis of the method with maximum probability transfer values.

| Failure Type | Vision A | | Vision B | | Gripper | | External A | | External B | | Localization | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scenarios | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Vision B | 0.11 | 0.01 | **0.93** | 0.02 | 0.61 | 0.05 | 0.12 | 0.02 | 0.15 | 0.03 | 0.10 | 0.09 |
| External A | 0.35 | 0.02 | 0.13 | 0.03 | 0.09 | 0.03 | **0.90** | 0.08 | 0.15 | 0.08 | 0.17 | 0.17 |
| Gripper | 0.24 | 0.22 | 0.18 | 0.11 | **0.77** | 0.00 | 0.16 | 0.08 | 0.26 | 0.22 | 0.14 | 0.04 |
| Localization | 0.22 | 0.10 | 0.11 | 0.02 | 0.69 | 0.19 | 0.37 | 0.28 | 0.11 | 0.00 | **0.98** | 0.02 |

From the table, it can be seen that HHMM-based failure isolation strategy efficiently explains failure cases when a failure occurs. When the rows are interpreted, it can be inferred that the proposed method explains faulty situations with the highest probability on average.

#### 5.2.2.2 Performance analysis of the method with average of the failure probability transfer values

In the analysis above, instant failure situations such as misrecognizing an object just for a time step are also transferred to the upper level in the hierarchy since it uses the highest failure probability value during taking a transition to the upper level.

**Table 5.2**: Performance analysis of the method using average approach values.

| Failure Type | Vision A | | Vision B | | Gripper | | External A | | External B | | Localization | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scenarios | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Vision B | 0.10 | 0.01 | **0.44** | 0.15 | 0.13 | 0.01 | 0.10 | 0.01 | 0.11 | 0.01 | 0.12 | 0.05 |
| External A | 0.32 | 0.02 | 0.11 | 0.02 | 0.09 | 0.03 | **0.82** | 0.10 | 0.11 | 0.02 | 0.12 | 0.02 |
| Gripper | 0.14 | 0.06 | 0.12 | 0.03 | **0.17** | 0.02 | 0.11 | 0.02 | 0.11 | 0.01 | 0.12 | 0.01 |
| Localization | 0.15 | 0.07 | 0.08 | 0.02 | 0.16 | 0.03 | 0.20 | 0.21 | 0.08 | 0.02 | **0.95** | 0.02 |

In order to eliminate these instant failures, average of the failure values may be transferred to the upper level. In Table 5.2, the corresponding results can be seen when averages of the failure probabilities are used while transferring to the upper level.

It can be seen from the table that although instant failures can be eliminated, the overall performance of the system decreases. Therefore, it is not efficient to use the average probabilities all the time.

#### 5.2.2.3 Performance analysis of the method with the hybrid approach

The maximum probability approach can not eliminate instant failures. However, average probabilities can eliminate these instant failures. Therefore, the sequence of the hidden states should be considered in order to decide which probability value to transfer to the upper level. Considering the sequence of the hidden states, if failure situations are followed with success states, then the average of the failure probabilities is transferred to the upper level. Otherwise, the maximum failure probability value is transferred to the upper level.

In the Table 5.3, it can be seen that hybrid approach eliminates instant failures. For example, for all scenarios, instant vision failures on object B are eliminated. Similarly, the mean probability of the external event on the object B decreases.

**Table 5.3**: Performance analysis of the method using hybrid approach.

| Failure Type | Vision A | | Vision B | | Gripper | | External A | | External B | | Localization | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scenarios | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Vision B | 0.11 | 0.01 | **0.80** | 0.16 | 0.61 | 0.05 | 0.11 | 0.01 | 0.15 | 0.03 | 0.10 | 0.2 |
| External A | 0.35 | 0.02 | 0.13 | 0.03 | 0.08 | 0.04 | **0.90** | 0.03 | 0.15 | 0.08 | 0.17 | 0.05 |
| Gripper | 0.17 | 0.11 | 0.18 | 0.11 | **0.77** | 0.00 | 0.16 | 0.08 | 0.15 | 0.04 | 0.14 | 0.05 |
| Localization | 0.22 | 0.11 | 0.11 | 0.02 | 0.69 | 0.19 | 0.34 | 0.28 | 0.11 | 0.03 | **0.98** | 0.02 |

#### 5.2.2.4 Performance analysis of the method with Particle Filters

In order to compare the proposed method, a Particle Filter based method is applied to the failure isolation problem.

Particle Filters (PFs) are probabilistic models and their underlying model is Bayesian Filters [43]. A Particle Filter consists of a number of particles. Each particle represents the system's belief on the state of the system. Therefore, belief of the system is represented with the particles. Higher density of the particles on a region means that the belief on that state is more probable. A general Particle Filter algorithm includes 3 main steps, prediction, importance sampling and resampling. First, particles are distributed considering the initial probability distribution. After, prediction step takes place. Particles are drawn among the states considering the transition model of the

system. When an observation is gathered, importance sampling is done, particles are weighted considering the observation probabilities. Then a resampling process is applied and particles are drawn considering their weights. After resampling process, the number of particles is protected. Therefore, at the end of the algorithm, the number of particles is the same as the initialization step.

In order to represent failure isolation problem with Particle Filters, each failure type is modelled as a distinct Particle Filter. For instance, there are six Particle Filters for the object manipulation scenario with two objects, namely, vision failure on object $A$, vision failure on object $B$, external event on object $A$, external event on object $B$, localization and hardware (gripper). After an action is executed, the maximum failure observation probability is used to weight the related PF. Since the failure isolation problem includes two hidden states, in each step of the Particle Filter algorithm, particles are drawn between these two states. Same parameters are used to model the Particle Filters. In the experiments, the number of particles is set to 5000. The results from the robotic experiments with Particle Filtering method is given in Table 5.4.

**Table 5.4**: Performance analysis of the method with Particle Filters.

| Failure Type | Vision A | | Vision B | | Gripper | | External A | | External B | | Localization | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scenarios | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Vision B | 0.14 | 0.02 | **0.70** | 0.02 | 0.64 | 0.01 | 0.15 | 0.06 | 0.17 | 0.03 | 0.15 | 0.01 |
| External A | 0.39 | 0.04 | 0.17 | 0.04 | 0.12 | 0.02 | **0.71** | 0.06 | 0.18 | 0.05 | 0.19 | 0.05 |
| Gripper | 0.24 | 0.17 | 0.21 | 0.12 | **0.65** | 0.01 | 0.18 | 0.07 | 0.18 | 0.04 | 0.18 | 0.09 |
| Localization | 0.30 | 0.12 | 0.12 | 0.02 | 0.59 | 0.16 | 0.36 | 0.22 | 0.15 | 0.01 | **0.97** | 0.02 |

It can be clearly seen from the Figure 5.4 that Hierarchical Hidden Markov Models outperforms Particle Filters for failure isolation problem. Since HMMs are represented hierarchically, the problem of failure isolation can be modelled more efficiently than Particle Filters.

### 5.2.3 Analysis of memory usage

In all the investigated scenarios, the size of the state space is analysed. It is shown that use of action-failure type relations reduces the number of states generated in our method (Table 5.5). In the table, the number of states with and without using relations (Table 4.1) and the average percentage of gain in terms of memory utilization are

presented for 40 real robot experiments. The overall memory reduction is 43.36% for these scenarios.

**Table 5.5**: Analysis of state generation on HHMMs.

| Scenario | # of States w/o relations | # of States w/ relations | Gain |
|---|---|---|---|
| Vision Failure | 457 | 245 | %46.47 |
| Gripper Failure | 491 | 320 | %34.93 |
| External Event | 260 | 118 | %54.61 |
| Localization Failure | 539 | 338 | %37.32 |

# 6. CONCLUSION

In this thesis, a temporal model for failure isolation that employs HHMMs is presented. Propagation property of the HMMs are used to model the continuousness of the failures. HHMMs are for modelling the possible failure types that a cognitive robot faces. Each failure type is represented as a distinct HHMM and relevant models are updated during the execution of the plan. Action-failure type relations and parameters of the actions are used to reduce the size of the generated state space.

Moreover, HHMMs ensure isolation of multiple faults and propose explanations for possible failure situations. Since the failures are modelled in parallel, each model is able to generate a probability in case of a failure. This property makes it possible to isolate multiple faults.

Due to hierarchical structure of the HMM, each action has its own probabilistic model for each failure type. This property also provides the opportunity to isolate persistent failures on a specific object or event. Modelling each failure type as a distinct model provides a simple yet effective representation.

Experimental results on several failure cases show that the proposed method can successfully suggest reasonable causes for detected failures. Considering the calculated probability for each model, candidate models can be interpreted considering their probabilities.

The main drawback of the proposed method is its model-based structure like all other model-based methods. Failures that don't have a model can not be isolated. Therefore, failure models should be defined for the system to isolate these type of failures.

## 6.1 Future Work

As a future work, isolation results will be used for the future plans of the robot. Therefore, the outcomes of the isolation process will be used as an input to a learning phase.

Since the proposed method is model-based, isolating non-modelled failures is hard. Therefore, new failure types will be added to the system in order to isolate more failures.

## REFERENCES


[1] **Alpaydin, E.** (2004). *Introduction to machine learning*, MIT press.

[2] **Fine, S.**, **Singer, Y. and Tishby, N.** (1998). The Hierarchical Hidden Markov Model: Analysis and Applications, *Machine Learning*, ***32***(1), 41–62.

[3] **Ozturk, M.D.**, **Ersen, M.**, **Kapotoglu, M.**, **Koc, C.**, **Sariel-Talay, S. and Yalcin, H.** (2014). Scene Interpretation for Self-Aware Cognitive Robots, *AAAI-14 Spring Symposium on Qualitative Representations for Robots*.

[4] **Karapinar, S.**, **Altan, D. and Sariel-Talay, S.** (2012). A Robust Planning Framework for Cognitive Robots, *Proceedings of the AAAI-12 Workshop on Cognitive Robotics (CogRob)*.

[5] **Pettersson, O.** (2005). Execution monitoring in robotics: A survey, *Robotics and Autonomous Systems*, ***53***, 73–88.

[6] **Usug, U.C.**, **Altan, D. and Sariel-Talay, S.** (2012). Robots That Create Alternative Plans against Failures, *10th IFAC Symposium on Robot Control*.

[7] **Karapinar, S.**, **Sariel-Talay, S.**, **Yildiz, P. and Ersen, M.** (2013). Learning Guided Planning for Robust Task Execution in Cognitive Robotics, *The AAAI-13 Workshop on Intelligent Robotic Systems*, Bellevue, USA.

[8] **Bouguerra, A.**, **Karlsson, L. and Saffiotti, A.** (2008). Monitoring the execution of robot plans using semantic knowledge, *Robotics and Autonomous Systems*, ***56***(11), 942–954.

[9] **Fritz, C.** (2005). Execution Monitoring – A Survey, **Technical Report**, University of Toronto.

[10] **Nan, C.**, **Khan, F. and Iqbal, M.T.** (2008). Real-time fault diagnosis using knowledge-based expert system, *Process Safety and Environmental Protection*, ***86***(1), 55 – 71.

[11] **Steinbauer, G. and Wotawa, F.** (2009). Robust Plan Execution Using Model-Based Reasoning, *Advanced Robotics*, ***23***(10), 1315–1326.

[12] **Bouguerra, A.**, **Karlsson, L. and Saffiotti, A.** (2007). Handling uncertainty in semantic-knowledge based execution monitoring, *IROS*, pp.437–443.

[13] **Friedrich, G.**, **Stumptner, M. and Wotawa, F.** (1999). Model-based diagnosis of hardware designs, *Artificial Intelligence*, ***111***(1–2), 3 – 39.

[14] **Chittaro, L. and Ranon, R.** (2004). Hierarchical model-based diagnosis based on structural abstraction, *Artificial Intelligence*, ***155***(1–2), 147 – 182.

[15] **Frank, P.M.**, **Ding, S.X. and Marcu, T.** (2000). Model-based fault diagnosis in technical processes, *Transactions of the Institute of Measurement and Control*, ***22***(1), 57–101.

[16] **Flores-Quintanilla, J.**, **Morales-Menendez, R.**, **Ramirez-Mendoza, R.**, **Garza-Castanon, L. and Cantu-Ortiz, F.** (2005). Towards a new fault diagnosis system for electric machines based on dynamic probabilistic models, *Proceedings of the 2005 American Control Conference*.

[17] **Verma, V.**, **Gordon, G.**, **Simmons, R. and Thrun, S.** (2004). Real-time fault diagnosis [robot fault diagnosis], *Robotics Automation Magazine, IEEE*, ***11***(2), 56–66.

[18] **Verma, V.**, **Simmons, R. and Fernandez, J.** (2002). Probabilistic models for monitoring and fault diagnosis, *The Second IARP and IEEE/RAS Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*, pp.43–67.

[19] **Micalizio, R.** (2013). Action Failure Recovery via Model-based Diagnosis and Conformant Planning, *Computational Intelligence*, ***29***(2), 233–280.

[20] **Schmill, M.**, **Josyula, D.**, **Anderson, M.L.**, **Wilson, S.**, **Oates, T.**, **Perlis, D. and Fults, S.** (2007). Ontologies for reasoning about failures in AI systems, *Proceedings from the Workshop on Metareasoning in Agent Based Systems at the Sixth International Joint Conference on Autonomous Agents and Multiagent Sytems*.

[21] **Gspandl, S.**, **Pill, I.**, **Reip, M. and Steinbauer, G.**, (2011). Belief Management for Autonomous Robots Using History-Based Diagnosis, Developing Concepts in Applied Intelligence, Springer, pp.113–118.

[22] **Gspandl, S.**, **Podesser, S.**, **Reip, M.**, **Steinbauer, G. and Wolfram, M.** (2012). A dependable perception-decision-execution cycle for autonomous robots, *IEEE International Conference on Robotics and Automation (ICRA)*, pp.2992–2998.

[23] **McCarthy, J.** (1963). Situations, actions, and causal laws, **Technical Report**, DTIC Document.

[24] **Cantrell, R.**, **Talamadupula, K.**, **Schermerhorn, P.**, **Benton, J.**, **Kambhampati, S. and Scheutz, M.** (2012). Tell me when and why to do it! Run-time planner model updates via natural language instruction, *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, pp.471–478.

[25] **Göbelbecker, M.**, **Keller, T.**, **Eyerich, P.**, **Brenner, M. and Nebel, B.** (2010). Coming Up With Good Excuses: What to do When no Plan Can be Found., *ICAPS*, pp.81–88.

[26] **Hovland, G.E. and McCarragher, B.J.**, (1996), Hidden Markov Models as a Process Monitor in Robotic Assembly.

[27] **Kwon, K.C. and Kim, J.H.** (1999). Accident identification in nuclear power plants using hidden Markov models, *Engineering Applications of AI*, ***12***(4), 491 – 501.

[28] **Ying, J.**, **Kirubarajan, T.**, **Pattipati, K. and Patterson-Hine, A.** (2000). A hidden Markov model-based algorithm for fault diagnosis with partial and imperfect tests, *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, ***30***(4), 463–473.

[29] **Ocak, H. and Loparo, K.** (2001). A new bearing fault detection and diagnosis scheme based on hidden Markov modeling of vibration signals, *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*, pp.3141–3144.

[30] **Peynot, T.**, **Fitch, R.**, **McAllister, R. and Alempijevic, A.** (2011). Autonomous Reconfiguration of a Multi-Modal Mobile Robot, *Workshop on Automated Diagnosis, Repair and Re-Configuration of Robot Systems, IEEE International Conference on Robotics and Automation (ICRA).*

[31] **Ge, M.**, **Du, R. and Xu, Y.** (2004). Hidden Markov Model based fault diagnosis for stamping processes, *Mechanical Systems and Signal Processing*, ***18***(2), 391 – 408.

[32] **Lee, J.M.**, **Kim, S.J.**, **Hwang, Y. and Song, C.S.** (2004). Diagnosis of mechanical fault signals using continuous hidden Markov model, *Journal of Sound and Vibration*, ***276***(3–5), 1065 – 1080.

[33] **Li, Z.**, **Wu, Z.**, **He, Y. and Fulei, C.** (2005). Hidden Markov model-based fault diagnostics method in speed-up and speed-down process for rotating machinery, *Mechanical Systems and Signal Processing*, ***19***(2), 329 – 339.

[34] **Baum, L.E. and Petrie, T.** (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains, *The Annals of Mathematical Statistics*, ***37***(6), pp. 1554–1563, `http://www.jstor.org/stable/2238772`.

[35] **Viterbi, A.** (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *Information Theory, IEEE Transactions on*, ***13***(2), 260–269.

[36] **Beetz, M.** (2000). *Concurrent reactive plans: anticipating and forestalling execution failures*, Springer-Verlag.

[37] **Altan, D. and Sariel-Talay, S.** (2014). Hierarchical HMM-based Failure Isolation for Cognitive Robots, *Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART)*, Angers, Loire Valley, France.

[38] **Altan, D. and Sariel-Talay, S.** (2014). Probabilistic Failure Isolation for Cognitive Robots, *Proceedings of the 27th International FLAIRS Conference*, Pensacola Beach, Florida, USA.

[39] **Bekey, G.A.** (2005). *Autonomous robots: from biological inspiration to implementation and control*, MIT press.

[40] **Hinterstoisser, S.**, **Cagniart, C.**, **Ilic, S.**, **Sturm, P.**, **Navab, N.**, **Fua, P. and Lepetit, V.** (2012). Gradient response maps for real-time detection of textureless objects, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ***34***(5), 876–888.

[41] **Ersen, M.**, **Sariel-Talay, S. and Yalcin, H.** (2013). Extracting Spatial Relations Among Objects for Failure Detection, *on Visual and Spatial Cognition*, 13.

[42] **Trevor, A.J.B.**, **Gedikli, S.**, **Rusu, R.B. and Christensen, H.I.** (2013). Efficient Organized Point Cloud Segmentation with Connected Components, *3rd Workshop on Semantic Perception, Mapping and Exploration (SPME)*, Karlsruhe, Germany.

[43] **Thrun, S.**, **Burgard, W. and Fox, D.**, (2005), Probabilistic robotics. 2005.

**CURRICULUM VITAE**

**Name Surname:** Doğan Altan

**Place and Date of Birth:** Orhangazi, 05/04/1989

**E-Mail:** daltan@itu.edu.tr

**B.Sc.:** Istanbul Technical University, Department of Computer Engineering

**Professional Experience and Rewards:** Department of Computer Engineering, Istanbul Technical University, Research Assistant (2011 - ongoing)

**List of Publications and Patents:**

**PUBLICATIONS/PRESENTATIONS ON THE THESIS**

▪ **Altan D.** and Sariel-Talay S., 2014: Probabilistic Failure Isolation for Cognitive Robots *The 27th International FLAIRS Conference*, May 21-23, 2014 Pensacola Beach, Florida, USA.

▪ **Altan D.** and Sariel-Talay S., 2014: Bilişsel Robotlar için Eylem Yürütme Hatalarının Tanısı *IEEE 22. Sinyal İşleme ve İletişimi Uygulamaları Kurultayı (IEEE 22nd Signal Processing and Communications Applications Conference (SIU 2014))*, April 23-25, 2014 Trabzon, Turkey.

▪ **Altan D.** and Sariel-Talay S., 2014: Hierarchical HMM-based Failure Isolation for Cognitive Robots *6th International Conference on Agents and Artificial Intelligence (ICAART)*, March 6-8, 2014 Angers, France.

**OTHER PUBLICATIONS/PRESENTATIONS**

▪ Usug C.U., **Altan D.** and Sariel-Talay S., 2012: Robots That Create Alternative Plans against Failures *10th IFAC Symposium on Robot Control (SYROCO)*, September 5-7, 2012 Dubrovnik, Croatia.

▪ Karapinar S., **Altan D.** and Sariel-Talay S., 2012: A Robust Planning Framework for Cognitive Robots *AAAI-2012 Workshop on Cognitive Robotics (CogRob)*, July 22-26, 2012 Toronto, Ontario, Canada.

▪ Usug C.U., **Altan D.**, Karapinar S., Ersen M., and Sariel-Talay S., 2012: An Integrated Planning and Learning Framework for Autonomous Mobile Robots *5th International Conference on Cognitive Systems*, February 22-23, 2012 Vienna, Austria.