ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE
ENGINEERING AND TECHNOLOGY

ADAPTIVE LEARNING OF SYMBOLIC NUMERICAL CONSTRAINTS IN THE REAL-WORLD

M.Sc. THESIS

Gökhan Solak

Department of Computer Engineering

Computer Engineering Programme

July 2017

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

ADAPTIVE LEARNING OF SYMBOLIC NUMERICAL CONSTRAINTS IN THE REAL-WORLD

M.Sc. THESIS

Gökhan Solak
(504141511)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Assoc. Prof. Sanem Sarıel

July 2017

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

SEMBOLİK SAYISAL KISITLARIN GERÇEK DÜNYADA UYARLANIR ÖĞRENİLMESİ

YÜKSEK LİSANS TEZİ

Gökhan Solak
(504141511)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Assoc. Prof. Sanem Sarıel

Temmuz 2017

Gökhan Solak, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504141511 successfully defended the thesis entitled "ADAPTIVE LEARNING OF SYMBOLIC NUMERICAL CONSTRAINTS IN THE REAL-WORLD", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Sanem Sarıel** Istanbul Technical University	....
Jury Members :	**Asst. Prof. Yusuf Yaslan** Istanbul Technical University	....
	Asst. Prof. Emre Uğur Boğaziçi University	....
		....
		....
		....

Date of Submission :	**5 May 2017**
Date of Defense :	**19 June 2017**

Anneme ve eşime,

FOREWORD

Bu tezin oluşumunda dolaylı ya da doğrudan katkısı olan herkese teşekkür ederim. Öncelikle bu tezin yeni oluşmaya başladığı zamanlarda kaybettiğim, beni ben yapan annem Seher Solak'a, bu tezi yazarken evlendiğim her koşulda yanımda olan eşim Monika Solak'a, bu tezi yazabilmek için gerekli altyapıyı kazanmamda bana rehberlik eden danışman hocam Sanem Sarıel'e teşekkür ederim. Beni destekleyen babama, kardeşime, tüm aileme ve yakın arkadaşlarıma teşekkür ederim. İTÜ Yapay Zeka ve Robotik Laboratuvarı'ndan yol arkadaşlarıma, özellikle Ongun Kanat'a, Arda İnceoğlu'ya, Abdullah Cihan Ak'a, Türker Ünlü'ye benimle paylaştıkları her şey için teşekkür ederim.

Bu tez 115E-368 numaralı TÜBİTAK projesi kapsamında desteklenmiştir.

July 2017 Gökhan Solak

TABLE OF CONTENTS

Page

ABBREVIATIONS

LP	**:** Logic Programming
ILP	**:** Inductive Logic Programming
CLP	**:** Constraint Logic Programming
FOL	**:** First-order logic
LOF	**:** Local Outlier Factor
CLP(FD)	**:** Constraint Logic Programming over Finite Domains

SYMBOLS

$\mathcal{H}$	: Hypothesis set
$\mathcal{B}$	: Background knowledge
E	: Example set
E^+, E^-	: Positive examples, Negative examples
e_p	: Current positive example
$\perp$	: Bottom clause
`b1`, `b2`, `b3`	: Object symbols
`A`, `B`, `C` ...	: Variable symbols
x_{upper}	: Upper bound of the interval
x_{lower}	: Lower bound of the interval
v_i^+, v_j^-	: Positive and negative values of a numerical variable
m_c	: Current sample
m'_i	: i^{th} neighbour sample
d_{m_i}	: Density indicator value of sample m_i
$\hat{p}_i$	: Expected value of error rate at i^{th} observation
$\hat{\sigma}_i$	: Standard deviation of error rate at i^{th} observation
w_1, w_2	: Warning point set by drift detector
d_1	: Drift point set by drift detector

LIST OF TABLES

Page

LIST OF FIGURES

ADAPTIVE LEARNING OF SYMBOLIC NUMERICAL CONSTRAINTS IN THE REAL-WORLD

SUMMARY

This thesis presents an adaptive learning system which can learn symbolic hypotheses representing numerical constraints using the observations of a robot. The system is based on a framework which deals with real-world conditions of noise and concept drift.

Inductive Logic Programming (ILP) is used as the base learning method. ILP is a machine learning approach that uses first-order logic (FOL) for knowledge representation. FOL representation allows expressing relational features, which is not possible with attribute-based machine learning methods. Another advantage of using FOL is that it allows reasoning to derive facts from the given observations and background knowledge. FOL is also used in planning and reasoning systems, which makes the learned rules easily adaptable to these systems.

The core learning method extends a well-known ILP system with a constraint solver and lazy evaluation. This extension deals with the limitation of ILP in learning numerical constraints. The extended learning method imposes constraints on the domains of the numerical variables in a hypothesis clause. If an appropriate constraint exists, addition of it transforms a previously inconsistent hypothesis into a useful one. Hence, the extended method combines the relational learning capability of Inductive Logic Programming and the numerical reasoning capability of Constraint Logic Programming.

Finding appropriate constraints on the domains of variables is achieved using a constraint solver. During the ILP hypothesis generation, if a hypothesis contains a numerical variable, then a constraint satisfaction problem (CSP) is solved to find the largest interval in the domain of the variable, such that, when the values in this interval are substituted with this variable in the hypothesis, the hypothesis does not entail any of the negative examples but it entails at least one positive example. If such an interval exists, constraining the domain of this variable makes the hypothesis consistent.

Since robot applications are targeted, learning symbolic numerical constraints should be robust under real-world conditions. The proposed system integrates a noise removal module to cope with uncertainties arising from the robot and its environment. Local outlier factor (LOF) method is used for noise removal since it does not require a prior cluster scheme and it takes densities into account. Noise removal is applied on the inputs of the constraint solver since it does not handle the noise itself. Another addition is a concept drift detector which makes the system adaptive to external and internal changes. A concept drift may render the previously learned hypotheses obsolete. Hence, the drift detection module continuously monitors the prediction success of the learned model, and requests the update of the learned hypotheses if necessary.

The system is evaluated with robot experiments in the real-world and computer generated scenarios. The results of the experiments show that the enhancements increase the robustness and the effectiveness of the learner. It is observed that the presented system can learn from noisy data that is collected in the real-world environment, and it can improve the prediction performance by detecting concept drifts. The system is expected to be useful in lifelong learning scenarios and in compromising between low-level and high-level robot learning.

SEMBOLİK SAYISAL KISITLARIN GERÇEK DÜNYADA UYARLANIR ÖĞRENİLMESİ

ÖZET

Robotlar çalışmaları esnasında birçok kısıtla karşılaşır. Görevlerini başarıyla tamamlayabilmek için robotların bu kısıtların farkında olması gerekir. Bu tezde robotların etkin bir biçimde ortam kısıtlarını öğrenebilmesi için bir ilişkisel öğrenme yöntemi sunulmaktadır. Ayrıca, yöntemin gerçek dünya koşullarında uygulanabilmesi için gerekli bileşenleri içeren bir uyarlanır öğrenme mimarisi önerilmektedir.

Robotların karşılaştığı kısıtlar çevreden, çalışma amaçlarından ya da robotun kendi yapısından kaynaklanıyor olabilir. Ortamın kısıtları nesnelerin uzamsal yerleşiminden, görevlerin gereksinimlerinden ya da robotun eylem yürütmesini etkileyen diğer dış faktörlerden kaynaklanabilir. Robotun kısıtları robot donanımının fiziksel özelliklerinden ya da robot yazılımlarının limitlerinden ötürü oluşabilir. Kısıtların ihlal edilmesi yürütülen eylemlerin istenmeyen şekilde sonuçlanmasına sebep olabilir. Güvenli ve verimli bir robot sistemi kendi içsel kısıtlarına ve ortam kısıtlarına dikkat etmelidir.

Kısıt bilgisi tasarımcı tarafından robota yüklenebileceği gibi, robotun kendisi tarafından da öğrenilebilir. Kısıtların robotun kendi gözlemlerinden öğrenilmesi sistemin esnekliğini ve uyarlanırlığını artırır. Genel amaçlı robotlar birçok farklı ortamda çalışabilir, dolayısıyla mümkün olan bütün kısıtların önceden tespit edilmesi zordur. Ortamın ve görevin koşulları zamanla değişiklik gösterebilir, bu yüzden robotun kısıt bilgisinin güncellenmesi gerekebilir. Robotun kısıtları kendi gözlemlerinden öğrenmesi bu sorunlara çözüm oluşturabilir.

Bu tezde önerilen öğrenme sistemi ortam kısıtlarının sembolik seviyede öğrenilmesini amaçlar. Yöntemin temelinde Tümevarımlı Mantık Programlama (TMP) vardır. TMP bilginin ifadesi için sembolik mantık kullanan bir makine öğrenmesi yöntemidir. Bilginin bu şekilde ifade edilmesi ilişkisel özniteliklerin kullanılmasını mümkün kılmaktadır. Sembolik mantık kullanmanın bir diğer yararı da verilen gözlemleri ve artalan bilgisini kullanarak yeni bilgiler çıkarsama yeteneğidir. Bu sayede, öğrenme yöntemine, alan uzmanı tarafından tespit edilen kuralların ve önceden elde edilmiş bilgilerin verilmesi mümkün olur. Sembolik mantık, robotikte planlama ve çıkarsama sistemlerinde de kullanıldığı için TMP ile öğrenilen kuralların kolayca bu sistemlere aktarılması mümkündür.

Ancak TMP'nin sayısal kısıtları öğrenmek konusunda eksiklikleri vardır. Bu yüzden öğrenme yöntemi tembel değerlendirme ve kısıt çözme ile geliştirilmiştir. Sayısal kısıtların ilişkisel bilgi korunarak öğrenilmesi bir kısıt çözme problemi olarak modellenmektedir. Geliştirilen yöntem, eğer mümkünse, tutarsız bir hipotezdeki sayısal değişkenlerin tanım kümelerini kısıtlayarak doğru hipotezler üretir. Geliştirilen yöntem Tümevarımlı Mantık Programlama'nın ilişkisel öğrenme kabiliyeti ile Kısıt Mantık Programlama'nın (KMP) sayısal çıkarsama kabiliyetini bir araya getirmektedir.

Sayısal kısıtlar bir kısıt çözücü kullanılarak belirlenir. TMP'nin hipotez üretimi sırasında, eğer bir hipotez sayısal bir değişkene sahipse, bir kısıt sağlama problemi (KSP) çözülür. Bu KSP çözülerek sayısal değişkenin tanım kümesi içerisindeki bir aralık tespit edilir. Bu aralıktaki değerler hipotezdeki sayısal değişkenin yerine koyulduğunda hipotez olumsuz örneklerin hiçbirini gerektirmez ve olumlu örneklerin en az birini gerektirir. Bulunan aralık mümkün olan en geniş aralık olmalıdır. Eğer böyle bir aralık varsa, değişkenin tanım kümesinin bu aralığa kısıtlanması hipotezi tutarlı hale getirir.

Yöntemin gerçek dünya ortamında çalışan robotlarda kullanılması amaçlandığı için zorlu dünya koşullarına uygun hale getirilmesi gerekmektedir. Bu çalışmada ele alınan koşullar gürültülü bilgi ve kavram kaymasıdır. Gürültülü bilgi robotun donanımsal zaafiyetlerinden ve dünya ortamının stokastik olmasından kaynaklanır. Robot amaçlanan davranışları yanlış yürütebilir ya da robotun topladığı sensör verileri kusurlu olabilir. Bir diğer zorlu dünya koşulu olan kavram kayması, bir kavramı açıklayan hipotezlerin zamanla değişmesidir. Eski hipotezler geçersiz hale geldiğinde sistemin hata oranı artar. Kavram kaymasının sebebi, genellikle, kavramın aslında gözlemlenemeyen bir bağlamla ilişkili olmasıdır. Gözlemlenemeyen bağlam değiştiğinde, onunla ilişkili olan kavramın da değişmesine neden olur. Gözlem yoluyla bu değişim fark edilemediği için kavram kaymasının tespiti için özel yöntemler geliştirilmiştir. Hatalar hem gürültü hem de kavram kayması tarafından oluşabileceği için tespit yönteminin bu ikisini ayırt edebiliyor olması gerekmektedir.

Öğrenme sistemini gerçek dünya zorluklarına dayanıklı hale getirmek için bir uyarlanır öğrenme mimarisi tasarlanmıştır. Bu mimari sembolik sayısal kısıt öğrenme yöntemini temel alacak şekilde geliştirilmiştir. Gürültü zorluğunun aşılabilmesi için öğrenme yöntemine bir gürültü filtreleyici eklenmiştir. Gürültünün filtrelenmesi için veri yoğunluğuna duyarlı ve genel amaçlı bir yöntem seçilmiştir. Gürültü filtreleme, kısıt çözücünün girdilerine uygulanır. Öngörücü bileşen, öğrenme yönteminin çıktısı olan hipotezleri, artalan bilgisini ve gözlemlenen dünya durumunu değerlendirerek tahminlerde bulunur. Doğrulayıcı bileşen, yapılan tahminleri gerçekleşen çıktı ile karşılaştırarak hataları takip eder. Kayma saptayıcı bileşen, doğrulayıcıdan gelen bilgileri değerlendirerek kavram kaymasını tespit eder. Bu çalışmada kullanılan tespit yöntemi, daha önce öğrenilmiş olan hipotezlerin tahmin etme hatasındaki değişmeleri takip ederek çalışır. Kavram kayması tespit edildiğinde, gözlem geçmişinden yeni bir eğitim kümesi oluşturarak öğreniciyi tetikler. Öğrenilen yeni hipotezler öngörücü tarafından gelecekteki çıkarsamalarda kullanılır.

Sunulan sistemin etkinliği, gerçek dünya ortamında yapılmış robot deneyleri ve bilgisayar ortamında oluşturulmuş deneyler ile değerlendirilmiştir. Bilgisayar ortamında oluşturulan deneylerde temel öğrenme sisteminin tek ve çok boyutlu kısıtları öğrenebildiği doğrulanmıştır. Gerçek dünya deneylerinde bir robot kolunun nesneleri yanyana koyabileceği yakınlık kısıtı ve bir insansı robotun bir kasedeki malzemeleri dökerken dikkat etmesi gereken açı ve yükseklik kısıtları öğrenilmektedir. Deney sonuçları, yapılan geliştirmelerin öğrenicinin etkinliğini ve gürbüzlüğünü artırdığını göstermiştir. Önerilen sistemin gerçek dünya ortamında elde edilen gürültülü veriden öğrenebildiği ve kavram kaymasını yakalayarak öngörü performansını artırdığı doğrulanmıştır.

Bu tezde, sembolik hipotezler ve sayısal kısıtları bir arada öğrenebilen bir öğrenme yöntemi ve bu yöntemin gerçek dünya koşullarına uyum gösterebilmesi için yapılan gürültü filtreleme ve kavram kaymasına uyarlanırlık geliştirmeleri sunulmaktadır. Gelecekte birçok farklı geliştirme seçeneği bulunan bu sistemin, ömür boyu robot öğrenmesi senaryolarında, robotların yüksek seviye öğrenme ve düşük seviye öğrenme sistemleri arasında geçiş sağlamasında faydalı olacağı düşünülmektedir.

1. INTRODUCTION

Robots working in real-world environments need to deal with various constraints for accomplishing their goals. Learning constraints in a relational learning setting is useful because this type of learning is suitable for exploiting the spatial features of environment. The learning system should be robust under real-world conditions such as noise and concept drift. This may be achieved with additional modules working together with the base learner.

Constraints may be inherent in the environment or the robot itself. Constraints of the environment may arise from the spatial layout of objects, special requirements of tasks or other external conditions that affect the action execution of the robot. For example, assume a sample task of pouring small pieces from one container to another. In Figure 1.1, parameters of the task are illustrated. A typical constraint of this task may depend on the rotation angle or height of the held container. A constraint may involve the relations of multiple variables, such as the depth or the diameter of the container on the table. Constraints of the robot originate from the physical properties of the robot's body or the computational limits of its sofware. Violation of the constraints may result in action failures that lead to undesired or inferior outcomes. A safe and efficient robot system should be aware about the constraints of itself and its environment to handle them effectively.

Awareness of constraints can be provided initially by the designer or obtained by the robot from its own experiences. Learning of the constraints by the robot has certain advantages in flexibility and adaptability. General purpose robots can operate in a wide range of environments, hence determining all of the possible constraints is a challenging task. Conditions of the environment and the tasks are subject to change, thus the constraints of the robot system should be updated when necessary. An effective method for lifelong learning of constraints has significant benefits in these manners.

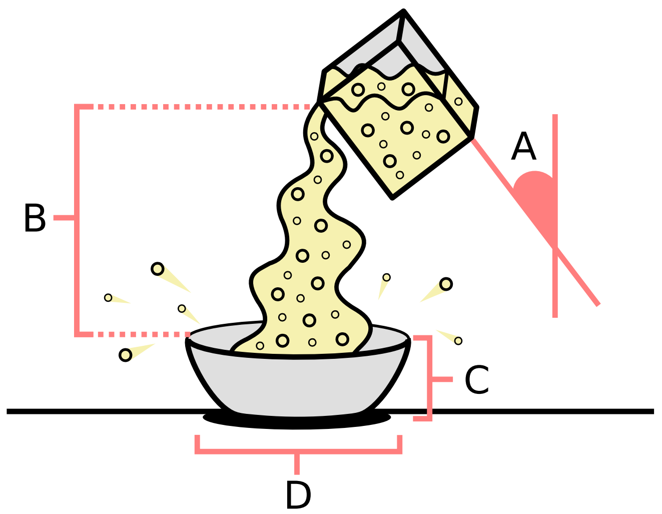

Figure 1.1 : Pouring action may have constraints on the rotation of the held container (A) or the distance between the containers (B). The constraints may be related to multiple variables such as the depth (C) or the diameter (D) of the container on the table.

The learning system proposed in this thesis aims to learn constraints in unstructured environments in the form of symbolic-level rules. Inductive Logic Programming (ILP) is used as the base learning method because it can exploit relational knowledge which can express complex world states of object manipulation tasks. An ILP method learns the general rules that explain given examples of a target concept. In [1], an ILP approach is used for learning the conditions which cause failures in action executions by robots. However, ILP is specialized in learning logical relations among objects and it has limitations in learning numerical constraints.

ILP can use a given numerical constraint as a background knowledge, and it can include the constraint in a rule it creates. For example, constraints of *short* and *long* distances are defined by Prolog clauses in Table 1.1. According to the definitions, a numerical term `X` is considered short if its value is lesser than 5, otherwise it is long. An ILP system provided with these background definitions can use them in a rule to distinguish short distances from long ones. This approach requires a domain expert to specify all relevant constraints before learning.

Table 1.1 : Example definitions for numerical constraints.

```
short(X)     :- X < 5.
long(X)      :- X >= 5.
```

Using prior definitions of constraints has some disadvantages. Possible constraints that can be learned by the robot are limited by the supervisor's intuititon. They are prone to mistakes that can be made by the supervisor in determining the constraints. Also, generality of the constraint definitions are limited. For example, the definition of *short* distance in one context may not be valid in another. For a humanoid robot manipulating objects on a table, *5 cm* is a short distance. However, for a smart car, the constraint of short distance is much larger. Therefore, two different definitions should be made for these two domains. These disadvantages are surpassed when the learning system can derive the numerical constraints from data.

Since the robots are subject to real world conditions, the learning system should be able to handle the noise and adapt to the drifting concepts. Noise can occur due to the uncertainty of the robot hardware or unpredicted external factors. Uncertainty of the robot hardware is either originated by the sensor errors or the actuator insensitivity of the robot. The stochasticity of the real-world environment also causes misleading observations. Concept drift is the condition that acquired hypotheses may become obsolete due to the changes in the environment or the robot itself. A concept drift may occur if the software or hardware of the robot system is updated in a way to change the constraints of the environment. For example, sensitivity of the robot may increase with the new hardware. Sometimes a difference in the environment or the task may change the concept. Coping with these conditions require extra efforts to remove the noise and monitor the knowledge validity over time. Both of these efforts are tackled in various ways in the literature of machine learning.

The robot can use the learned constraints to update its action model or predict the potential failures in order to avoid execution of unsafe actions. Planning and reasoning is done in symbolic level, hence constraints learned in the symbolic level can be easily integrated with these systems.

1.1 Purpose of Thesis

In this thesis, a method is proposed to learn symbolic hypotheses with numerical constraints from the actual observations of the robot. This method improves the flexibility of existing ILP approaches by supporting the derivation of numerical constraints from observations. Derivation of numerical constraints is integrated in a well-known ILP system [2]. The resulting system can bring relational features together with numerical constraints in a learned model. Relational learning allows acquiring general knowledge in an unstructured environment.

Experience-based learning of constraints of a robot and its environment has more flexibility and adaptability comparing to providing the robot with constraints. Flexibility is gained by the ability to extract constraints from the observations of the robot. Therefore, the robot can learn the appropriate constraints in the present environment. Adaptability is improved, since the constraint learning can be revoked if necessary, compared to manually coding of the constraints by human experts. Moreover, updating the constraints can be automated using a meta-learning system.

The proposed method is suitable for real-world conditions. The method can generalize rules from noisy and incomplete observations without over-fitting. It is suitable for lifelong learning scenarios as it allows knowledge transfer and usage of prior knowledge. The learning system is integrated with a mechanism to verify and update the learned hypotheses when they lose correctness due to concept drift.

1.2 Literature Review

Robots should make good decisions to execute their tasks effectively and safely. Satisfying the given constraints is a vital part of making good decisions [3]. The set of constraints change according to contexts and tasks. Learning the constraints of the robot and the execution environment is important since the set of constraints directly affect the decisions of the robot.

Robot systems which learn from data overcome the dependency on domain experts. Attribute-based learning methods are used to learn numerical constraints of robot task execution. Reinforcement learning methods are used for learning from experience [4], while supervised learning methods are used for learning from demonstration [5]. However, attribute-based representations are insufficient in expressing the various relations among objects. Learning from relational features using an attribute-based learning algorithm requires a mapping from relations to attributes by human intuition which makes the solution more problem-specific.

ILP overcomes these limitations by deriving hypotheses using relational features [6,7]. It does so using a first-order logic representation with integrated logic reasoning that allows the usage of background knowledge. ILP is applied in many domains which has relational features such as predicting chemical mutagenity [8], recognizing symbols by their spatial features [9]. In robotics, some applications of ILP are learning the failure contexts in robot task execution [1] and learning relational affordances in object manipulation tasks [10].

However, ILP does not handle the numerical features so well as the attribute-based learning methods. Early ILP algorithms introduced definition of numerical relations in background knowledge [11]. Progol [12] and Aleph [2] involve the constant values that appear in examples and background knowledge in searched hypotheses, which sometimes lead to useful hypotheses. This method is referred as the *guessing method* in the rest of the thesis. Finally, the lazy evaluation approach is used to integrate attribute-based learning algotihms into ILP induction [13]. Lazy evaluation is proven to be useful in regression of variables in a first-order logic clause, however it has a limitation in classification scenarios as discussed in this thesis. This limitation is eliminated using a constraint solver.

The constraint solver, however, is not robust under noise. Hence a noise filtering module is necessary to satisfy real-world conditions. Noise detection is a well-studied field [14]. Local Outlier Factor (LOF) method [15] is suitable for the proposed learning system since it can detect noise in data with various densities and it does not require a clustering scheme to be specified beforehand.

The problem of concept drift is studied under machine learning. Concept drift handling approaches depend on the applications [16] and the learning system [17]. Some approaches rely on detecting when the drift occurs [18–20], others train multiple learners and use an ensemble model [21]. The proposed learning system integrates a method which observes the error rate and detects knowledge drifts [18]. There exists a concept drift application in robotics domain [22]. Differently from the proposed system, it follows the ensemble approach. Multiple models are trained and the appropriate subset of the models are activated for different contexts. The learning system in [22] benefits from the fact that in their problem, a model can be trained using a single image, using stereo labeling. It adds a new model to ensemble for each observation. But this approach is not suitable for the proposed system.

1.3 Hypothesis

In this thesis, an ILP learning algorithm is used to exploit relational features for deriving hypotheses. Using a relational knowledge representation enables expressing complex object manipulation observations properly. Since features are not mapped to a problem-specific attribute set, more general conclusions are derived.

Limitation of ILP in learning numerical constraints is resolved by extending the ILP search with a constraint induction procedure [23]. Numerical constraints are derived using a constraint solver as described in [24]. Integration of constraint induction into the ILP algorithm is done with a slight modification of lazy evaluation [13].

The constraint solver cannot handle noisy inputs, hence its inputs should be filtered from noise. A density-based noise removal method [15] is included into the learning system in order to answer the needs of real-world environments.

The learning system is equiped with a module to adapt dynamic conditions of the real-world environment. A drift detector module continuously monitors the validity of hypotheses and analyses the error rate statistically [18]. When the acquired knowledge become obsolete, the learning system is triggered to update its hypotheses.

The proposed learning system overcomes the limitations of previous high-level learning systems by learning symbolic numerical constraints from the experience of a robot. The system is equipped with noise removal and drift detection mechanisms to deal with real-world conditions.

1.4 Organization

The next chapter presents the background information on the topics of Inductive Logic Programming, constraint induction, noise removal and concept drift. Then in Chapter 3, details of the proposed method are explained. In Chapter 4, constraint induction, noise reduction and drift detection methods are evaluated with hypothetical and real-world robot experiments. In the final chapter, outcomes of this thesis are summarized and the directions for future works are given.

2. BACKGROUND

This section introduces the background concepts that are used in the lifelong learning system. The base learning approach, Inductive Logic Programming (ILP) is described with its problem definition and motivation. The details of the adopted ILP method are presented. Earlier methods on learning numerical constraints in ILP are detailed in the next section. Then, discussion of noise removal which is used for filtering the input of the constraint solver is given. Basis of the concept drift detection method is presented in the last section.

2.1 Inductive Logic Programming

Inductive Logic Programming (ILP) is a class of machine learning algorithms which use first-order logic as their knowledge representation language. ILP algorithms benefit from the expressive power and reasoning capabilities of first-order logic. Knowledge represented in first-order logic allows the exploitation of relational features of data. This makes ILP a suitable approach for problems involving spatial or structured features [6]. Examples of relational features in the blocks-world domain are shown in Figure 2.1.

Relations of *on*, *in* and *distance* are very likely to be observed in a tabletop block manipulations scenario. This makes it a suitable representation language for robot applications.

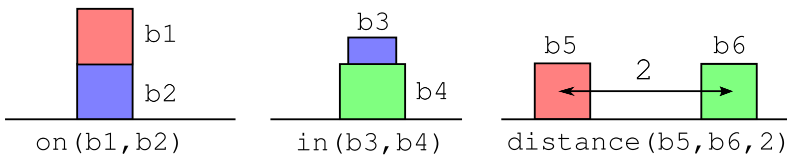

Figure 2.1 : Sample relational features in the blocks-world domain.

ILP methods derive hypotheses which predict positive examples of a given concept given a background knowledge. Derived hypotheses are consistent with the provided negative examples. Derivation of hypotheses is done by searching the space of possible hypotheses. Starting from an initial hypothesis, new candidate hypotheses are explored using a refinement operator. ILP approaches can be classified as *top-down* and *bottom-up* according to their search direction. Top-down approaches start from the most general clause and use refinement to create more specific clauses while bottom-up approaches start with the most specific clause and generalize it.

The ILP learner used in the proposed system is Aleph [2]. Aleph is an experimental ILP system which includes many concepts developed in ILP literature. Its default learning method is based on mode-directed inverse entailment [12]. This method uses a top-down approach that bounds the search space from bottom using a language bias defined with *mode declarations*. Mode declarations are explained in Section 2.1.3.

2.1.1 Problem

An ILP learning problem has the goal of deriving a set of hypotheses $\mathcal{H}$ that explains a concept given a set of positive and negative examples $E = E^+ \cup E^-$ and a background knowledge $\mathcal{B}$. Background knowledge and examples are commonly encoded as logic programs. Background knowledge $\mathcal{B}$ includes the world facts and the domain knowledge while set of examples E includes the grounded instances of the target concept. ILP learning problem is formally expressed as follows:

$$\mathcal{B} \wedge \mathcal{H} \models E. \tag{2.1}$$

This expression indicates that the set of hypotheses $\mathcal{H}$ and background knowledge $\mathcal{B}$ together explain the set of examples E. After learning $\mathcal{H}$, it can be used together with $\mathcal{B}$ to predict the observations.

In a robotic domain, the world facts are the world states before the execution of corresponding actions. Domain knowledge is the set of common-sense rules. Examples are observations of the robot in specific states.

2.1.2 Inverse entailment

Inverse entailment [12] is the basis of the ILP system adopted in our method. This approach bounds the hypothesis space from below by building the most specific clause that entails a given positive example. This clause is named as *bottom clause* $\perp$. It is built according to the language limitations defined by *mode declarations* as explained in the next subsection. Any hypothesis that covers the current positive example must subsume the bottom clause $\perp$, hence each candidate hypothesis is a subset of it.

ILP learning has two procedures; *saturation* and *reduction*. These procedures are applied to each positive example that remains uncovered. *Saturation* is the phase that builds the bottom clause $\perp$ and *reduction* is the phase that searches the subsets of the bottom clause $\perp$. In the *reduction* phase, candidate hypotheses are continuously explored and evaluated according to a performance criterion. This criterion is commonly correlated with the covered positive and negative examples and the complexity of the clause. The covered positive examples increase the chance of this hypothesis to be chosen while the covered negative examples and the clause complexity introduce a penalty. The hypothesis with the best performance is chosen as a result of the reduction procedure. Other positive examples that are covered by this hypothesis are removed from the search queue. Saturation and reduction are repeated for every positive example which is not yet covered.

2.1.3 Mode declarations

Mode declarations impose limits on the structure of literals which can be used in a candidate hypothesis. The mode of a literal determines the types, roles and recalls of its arguments. An argument can have the input role that requires a variable with the same type already exists in the body of the hypothesis clause, or it can have the output role allowing a new variable to be introduced to the clause, or it can be a constant that can take a single value in accordance to the background knowledge and examples. *Recall* limits the possible number of outputs of a literal given a set of inputs. For example, assume a language to express a table-top block world scenario (Section 3.1). The language contains predicates to describe the spatial features and the colors of the objects. In Aleph, modes of such language are declared as shown in Table 2.1.

Table 2.1 : Mode declarations for the literals in blocks-world scenario.

```
mode(*, on_table(-object)).
mode(1, in_hand(-object)).
mode(1, position(+object, -integer)).
mode(1, distance(+integer, +integer, -integer)).
mode(1, paint(+object, #color)).
```

The leftmost number indicates the recall of the literal. Types of the arguments are written in the parantheses after the literal together with a symbol indicating their roles. Role symbols are '+' for input, '−' for output and '#' for constant arguments. Table 2.1 indicates that the recall of `on_table` predicate is indefinite ($\star$) and recall of all others are `1`. That is because there can be an indefinite number of objects on the ground. However, there can be only one object in the hand of the robot, or an object can have only one color. Since the arguments of `on_table` and `in_hand` literals assume the output role, addition of these literals to the candidate hypothesis introduces a new variable with `object` type to the clause. However, `position` and `paint` literals can only be associated with an `object` variable that is already in the clause.

2.2 Constraint Induction

The proposed method for learning numerical constraints with ILP requires some extensions to the basic ILP learner [23]. The proposed constraint induction extension uses two existing approaches as its basis. Firstly, the lazy evaluation procedure [13] for integrating attribute-based algorithms in ILP generalization scheme is used. Lazy evaluation allows branching to an attribute-based learning procedure when necessary. Secondly, a definition of generalization among numerical constraints is adopted from [24]. Details of these adoptions are presented in this section.

2.2.1 Lazy evaluation

Lazy evaluation method is used for learning numerical concepts with ILP [13]. This approach allows using attribute-based machine learning methods as lazy evaluated predicates in the background knowledge. Integration of lazy evaluated predicates into ILP search requires a special procedure since there are basic differences between ILP and attribute-based learning algorithms.

In Aleph ILP learner, each positive example is generalized one by one. Candidate hypotheses are derived using the bottom clause $\perp$ which is specific for the chosen positive example. After a hypothesis is added to the theory, all positive examples that are entailed by it are removed, and the learner chooses a remaining positive example to generalize. On the other hand, attribute-based learning algorithms consider all positive and negative examples together. They create models which minimize the overall prediction error. Lazy evaluation procedure creates a bridge between these two approaches.

Predicates associated with attribute-based learning methods are marked as lazy evaluated in the background knowledge $\mathcal{B}$. Outputs of lazy predicates are left uncertain in the saturation phase. They are evaluated lazily during the reduction phase.

Main ILP search loop is not changed by lazy evaluation. Space of candidate hypotheses are explored from general to specific by considering a subset of the bottom clause $\perp$, gradually increasing the number of literals in the clause. Whenever the candidate hypothesis clause has a lazy evaluated predicate in its body, a special procedure for lazy evaluation is initiated. Evaluation is done with the current input arguments of the lazy predicate that are derived from the prior part of the clause. This requires the *argument collecting* step to collect all input values following from the positive examples and the negative examples. This is achieved by unifying the head of the candidate hypothesis with every positive and negative example. Following the unification and the background knowledge, variables in the body of the clause are grounded. Each possible value of the input argument of the lazy predicate is added to the set of positive or negative values with respect to the label of the example. Then, the evaluation is done according to the definition of the attribute-based algorithm with provided set of input values. Result of the attribute-based algorithm is assigned as a constant to the output of the lazy evaluated predicate.

Performance of a candidate hypothesis is calculated as usual in reduction step, using the result of the lazy evaluation as the output value of the lazy evaluated predicate. However, evaluation of a lazy predicate is specific for the candidate hypothesis, since the input arguments will vary according to the body of the clause. For that reason the lazy evaluation is repeated for every clause.

2.2.2 Maximally discriminant domain constraints

Constraint Logic Programming (CLP) provides better numerical reasoning capabilities than basic Logic Programming (LP). [24] uses CLP as its representation language. A constraint solver is used for modifying the domains of variables in order to cover the positive examples and exclude the negative examples. A generality relation among the CLP clauses is defined in this work.

A *CLP clause* is different from LP clauses by that it contains constraint terms as function symbols. An example of a CLP clause is given in Table 2.2. This clause defines that object `B` is carriable if its weight is constrained below 1500. Here, CLP allows the addition of the constraint `W < 1500`. A generality definition to determine which constraints are more general another is obtained by a modification of θ-subsumption.

Table 2.2 : A CLP clause defining a carrying limit.

`carriable(B)`	`:- weight(B, W), W < 1500.`

A *maximally discriminant domain constraint* is defined as the domain constraint which is consistent with the negative examples and θ-subsumes all other domain constraints. Consistency is the requirement that the domain constraint covers none of the negative example values. An example of a maximally discriminant domain is shown in Figure 2.2. In the figure, the maximally discriminant domain is shown for a variable which takes the values $\{2,5\}$ for negative examples. In other words, the hypothesis does not satisfy the ILP requirements if the variable is assigned with these values. Hence the maximal domain, excluding these values is $(-\infty,1] \vee [3,4] \vee [6,\infty)$.

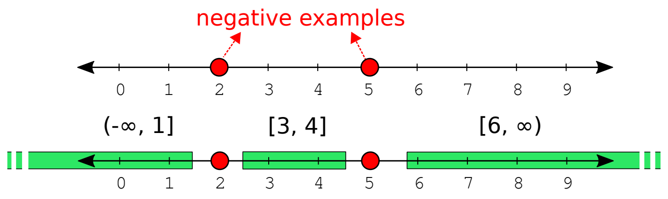

Figure 2.2 : Maximally discriminant domain for a simple set of negative example values $N = \{2,5\}$.

2.3 Noise Removal

Noise is defined by the examples that stand diverse from the general pattern of the data [14]. In robot task execution, the noise can occur because of the imperfections of the perceptors or actuators. For example, the scene modeling module of the robot may misinterprete the locations of the objects, or the motors of the robot may not be sensitive enough to produce the exact goal.

There are various noise detection methods for different needs [14]. We are interested in unsupervised approaches, since the data sent to the constraint solver are without labels. Also, generality of the approach is important to work in unstructured environments. Distance-based [25], density-based [15] and cluster-based methods [26] are appropriate, since they do not presume a given model. Clustering-based noise removal is effective only when the cluster number is given correctly [27]. However, in our setting, the cluster count is not certain and it should not be limited for the generality of the method. Distance-based methods do not take the clusters or densities into account, therefore they have limitations in distinguishing noise within clustered data. We use the density-based LOF method [15] as it is independent from the number of the clusters and it works better with data having clusters of different densities.

2.4 Concept Drift

Concept drift is the problem that the rules explaining a concept changes over the time. The change occurs often due to a *hidden context* [17] which is not observed, therefore not expressed in the language. The drift may occur as incrementally or suddenly as shown in Figure 2.3. Concept drift may be confused with noise as both result in a deviation from the learned hypotheses. Hence, the learning system should be able to differentiate between noise and concept drift [17].

The proposed system uses a forgetting-based method [18] which continuously monitors the prediction error rate and detects drift in case a large deviation occurs. It does not maintain a window as other forgetting-based methods [19, 20]. Instead it cuts the observations from a past point when a concept drift is detected. This approach is more suitable to the proposed learning system since it follows a batch learning

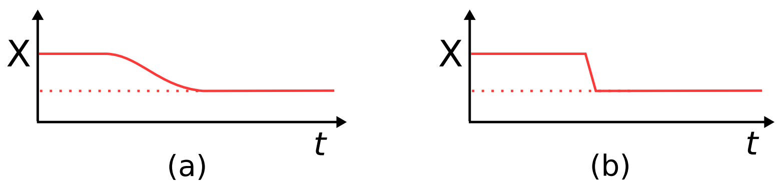

Figure 2.3 : Concept drift of a variable X. (a) An incemental drift (b) A sudden drift.

tradition, instead of an incremental one. An incremental learner updates its theory with every example, instead of learning from a set of examples altogether. Hence, it requires a special base learning method for this purpose. Since the presented learning method does not estimate the probability distribution of the data [19], the prediction accuracy is used as the indicator of the drift.

3. LEARNING SYMBOLIC NUMERICAL CONSTRAINTS

The method proposed for learning of symbolic numerical constraints uses the Aleph ILP system [2] as the base learner. Aleph is an open-source ILP implementation which bears many different strategies as its components. The adopted strategy in the proposed system is the inverse entailment [12] which bounds the search space from bottom using mode declarations. Aleph is extended with a constraint induction system that uses a constraint solver to find the *maximally discriminant domains* of variables. The constraint solver is integrated into Aleph using a slightly modified *lazy evaluation* [13] method. Aleph implementation is available as a YAP-Prolog program, hence the YAP-Prolog implementation of the *CLP(FD)* library [28] is used for constraint solving. Since, the expected application domain of this learning method is robotic, it should be suitable for real-world conditions such as noisy observations and concept drift. An adaptive learning framework is presented in Figure 3.1, for improving the applicability of the learning method under these conditions. This framework is built around the base learning method, which consists of the ILP method and the constraint solver. Noise filtering is applied on the data sent from ILP to the constraint solver. The LOF method [15] is used for this purpose. The LOF method is implemented

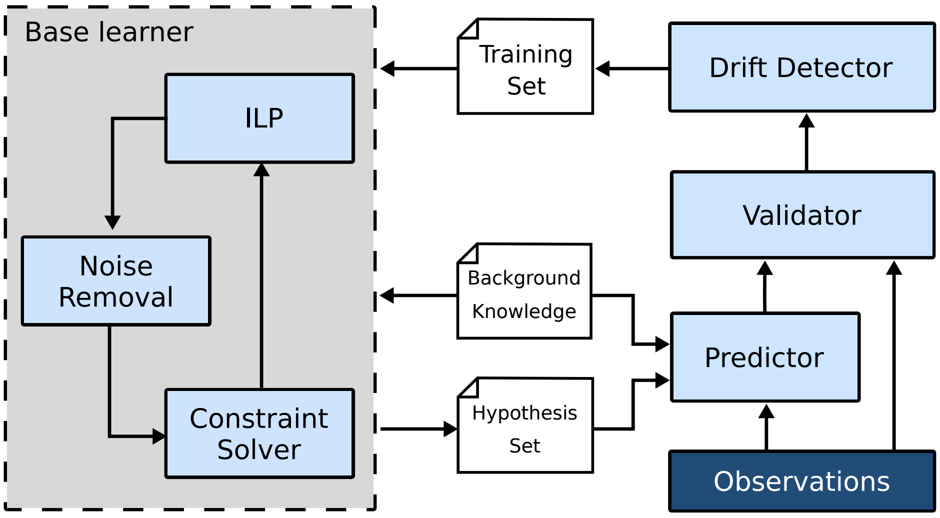

Figure 3.1 : The architecture of the adaptive learning system.

as a program that is run from the extended Aleph learner before constraint solving. Concept drift handling is achieved by dividing the responsibilities among multiple modules: *drift detector*, *predictor* and *validator*. *Drift detector* module implements the forgetting-based method from [18]. *Predictor* is a Prolog script that reasons about the current state using the learned hypotheses $\mathcal{H}$ and the given background knowledge $\mathcal{B}$. *Validator* is a program that evaluates the output of *predictor* as soon as the label is available. For example, before the robot takes any action, it predicts a certain outcome. After executing the action, the robot can observe the outcome of its action, and compare it to the predicted outcome. The output of *validator* is used by *drift detector* in order to estimate the error rate. *Drift detector* selects the training set among the past observations and calls the learning method.

In the following section, a running example is defined for better demostration of the methodology. Then, the proposed method for symbolic learning of the constraints is presented in detail. Section 3.3 explains the application of noise removal in order to generalize data obtained from real-world environments. Section 3.4 details the drift detector and the knowledge verification architecture for adaptive learning.

3.1 Blocks-world Scenario

A simple blocks-world scenario is designed for the demonstration of the learning system. Assume a humanoid robot, which has the task of putting blocks on a table. There are already few objects present on the table. These objects are represented by b_i symbols($i = 1,2,...$). Space is discrete and one dimensional for simplicity.

In this scenario, the robot holds object `b1` in its hand. There are two objects, `b2` and `b3`, on the ground. Object `b2` is positioned at location `1` and object `b3` is positioned at `8`. The state of the world is illustrated in Figure 3.2. Symbols of the background knowledge and their explanations are given in Table 3.1. The corresponding background knowledge $\mathcal{B}$ and examples E for this problem instance are shown in Table 3.2. Here, the background knowledge has the necessary `distance` definition and facts describing the current world state.

The robot aims to learn the distance constraint which causes a failure. Putting action of the robot has two arguments; the name of the block and an integer denoting the target location. The target concept is denoted as `put_on_fail(b1, 2)`, i.e., putting the object `b1` on location `2` fails.

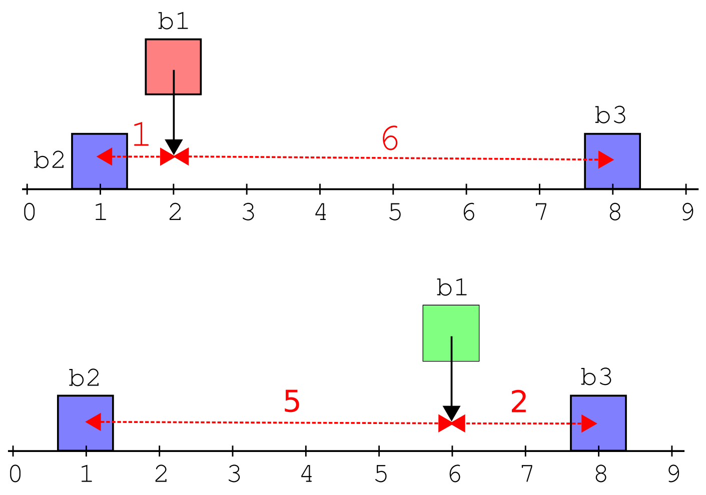

Figure 3.2 : Examples of a failure and success cases of put action. Putting on location 2 fails above, putting on location 6 is successful below.

Table 3.1 : Symbols used in the blocks-world scenario and their explanations.

Symbol	Definition
`put_on_fail(b1,2)`	Putting the block `b1` on position `2` fails
`on_table(b2)`	`b2` is on table
`in_hand(b2)`	`b2` is at the hand of robot
`position(b2,1)`	`b2` is positioned at `1`
`distance(A,B,C)`	`C` = |`A-B`|

Assume that a failure of put action occurs when the distance of the carried object to any object which is already on table is *less than 2 units*. Possible cases of a failure are illustrated in Figure 3.2. In the first case, the failure occurs because the distance of object `b1` to object `b2` is `1`. In the second case, distances of object `b1` to other objects are `5` and `2` which are both not lesser than 2. Thus, the object is put successfully. Failure of the action is a positive example while the success is a negative example in this scenario.

3.2 Constraint Induction with Lazy Evaluation

The proposed method brings an ILP learner and a constraint solver together to learn relational rules explaining the target concepts which require numerical constraints.

Table 3.2 : Background knowledge, positive and negative examples of the blocks-world scenario.

Positive example E^+
`put_on_fail(b1,2)`

Negative example E^-
`put_on_fail(b1,6)`

Background knowledge $\mathcal{B}$

`in_hand(b1).`
`on_table(b2).`
`on_table(b3).`
`position(b2,1).`
`position(b3,8).`
`distance(A,B,C) :- C = |A-B|.`

The method combines the relational learning advantages of an ILP learner and the numerical inference ability of a constraint solver.

Integration of the constraint induction procedure does not alter the main loop of the ILP algorithm as explained in Section 2.2.1. Integration is achieved by a slightly modified lazy evaluation [13] approach. The difference of the lazy evaluation application in the porposed learning method comparing to the original one is in the argument collecting step. Lazy evaluation collects the input arguments following from all positive and negative examples. However, in the proposed method, only the argument values from the current positive example are collected. Negative input arguments following from all of the negative examples are collected as the original approach. That means, in our method, a constraint satisfies the consistency requirement by excluding all of the negative examples. However, a constraint does not need to be complete itself alone. It is expected to satisfy only the current positive example which is resembling the relational learning approach of ILP.

Every variable that is *constrainable* are specified in the background knowledge $\mathcal{B}$. A special term for each of these variables are added to the bottom clause $\perp$ in the *saturation* phase (See Section 2.1.2). The special term for constrainable variables has the form `discr(V,D)`. This term has a general definition of '`discr(V,D) :- V in D.`' meaning that this expression is true when the value of variable `V` is in the domain `D`. In the blocks-world scenario, the second arguments of `put_on_fail` and `pos` literals and the third argument of the `distance` literal are marked as constrainable. In the blocks-world scenario, bottom clause $\perp$ is derived as shown in Table 3.3. Terms written in bold face style are added by the proposed extension.

Table 3.3 : Blocks-world scenario's bottom clause $\perp$.

Bottom clause $\perp$:
`put_on_fail(A,B) :- on_table(C), in_hand(A),`
`distance(B,B,D), position(C,E), distance(D,D,D),`
`distance(D,B,B), distance(E,E,D), distance(E,D,E),`
`distance(E,B,F),` **`discr(B,0), discr(D,0),`**
`discr(E,0), discr(F,0).`

After the bottom clause $\perp$ is built, ILP searches the hypothesis space in the *reduction* phase (See Section 2.1.2). Lazy evaluation procedure is called whenever a candidate clause includes one of these `discr` terms. Lazy evaluation of constraint induction procedure returns the maximally discriminant domain constraints that entail the current positive example and exclude all of the negative examples. Finally, these constraints are substituted to the body of the candidate hypothesis and the ILP search continues as usual.

Learning symbolic numerical constraints is achieved by modifying the saturation and reduction phases of Aleph. Extension on saturation procedure is shown in Algorithm 1. This extension adds `discr` terms to the bottom clause $\perp$. Extension on reduction procedure is shown in Algorithm 2. Reduction is extended to lazy evaluate the derivation of the maximally discriminant domain constraints when necessary. The values of the constrainable terms are collected in lines 11-12. Then the set of collected values are sent to the constraint solver to induce the maximally discriminant domain in line 13. The maximally discriminant domain is used to form constraints and it is substituted into the candidate clause.

Algorithm 1 Algorithm extension in the saturation phase.

```
Require: $\mathcal{B}$, $e_p$
 1: Create $\perp$ using $\mathcal{B}$, $e_p$
 2: Get constrainables $Cs$ from $\mathcal{B}$
 3: for all constrainable $c_i \in Cs$ do
 4:    Find variable terms $V_c$ in $\perp$ which matches $c_i$
 5:    for all variable $v_i \in V_c$ do
 6:       if $v_i$ is not already added then
 7:          Add discr($v_i$, 0) to $\perp$
 8:       end if
 9:    end for
10: end for
11: Remove redundant terms from $\perp$
```

Algorithm 2 Algorithm extension in the reduction phase.

```
Require: $\mathcal{B}$, $e_p$, $E^-$
 1: Initialize node $n_s = head(e_p)$
 2: while termination criteria is not met do
 3:    Expand node $n_s$ to successor nodes $Succ$
 4:    for all node $n_i \in Succ$ do
 5:       for all term $t_i \in n_i$ do
 6:          if $functor(t_i)$ is not discr then
 7:             Continue
 8:          end if
 9:          Unify $t_i$ with $discr(var_i, D_i)$
10:          Let $D_c = (-\infty, \infty)$
11:          Find values $NegVals$ of $var_i$ following from $E^-$
12:          Find values $PosVals$ of $var_i$ following from $e_p$
13:          $D_c = induce\_domain(NegVals, PosVals)$
14:          if $D_c$ is not empty then
15:             Create term $t_n =$ discr($var_i$, $D_c$)
16:             Replace term $t_i$ with $t_n$ in node $n_i$
17:          end if
18:       end for
19:    end for
20:    Calculate scores of every node in $Succ$
21:    Choose next node to be $n_s$
22: end while
```

The function *induce_domain(NegVals, PosVals)* derives the maximally discriminant domain for the given negative and positive values. For each positive value $v_i^+ \in$ *PosVals*, a constraint satisfaction problem is defined as shown in Table 3.4. This is a maximization problem that finds the *largest* interval which includes the positive value v_i^+ and does not include any of the negative values $v_j^- \in NegVals$. The interval is expressed as $[x_{lower}, x_{upper}]$. The complete discriminant domain is the disjunction of the intervals found for each positive value.

Table 3.4 : Constraint satisfaction problem for finding the interval $[x_{lower}, x_{upper}]$ that discriminates a positive value v_i^+ from all negative values v_j^-.

maximize	$area(x_{lower}, x_{upper})$	
subject to	$x_{lower} \leq v_i^+ \wedge x_{upper} \geq v_i^+$	Include the positive value
	$\neg \bigwedge_{v_j^-} (x_{lower} \leq v_j^- \wedge x_{upper} \geq v_j^-)$	Exclude all of the negative values

For the candidate hypothesis '`put_on_close(A,B) :- on_table(C), position(C,E), distance(E,B,F), discr(F,0)`' constraint induction process in the reduction phase runs as follows. Since there is a `discr(F,0)` term in the candidate hypothesis, the domain of `F` is subject to constraint induction. Negative input arguments of the variable `F` are collected as {2, 5}, since the target location of the negative example is 5 and 2 units far to blocks `b2` and `b3`. The positive values of `F` are collected as {1, 6}. Then the constraint solver is called to find the maximal domain which excludes {2, 5} and includes {1, 6}. The resulting domain is $dom_c = (-\infty, 1] \vee [6, \infty)$. The final hypothesis is obtained by substituting `discr(F,0)` with `discr(F,(−∞,1]∨[6,∞))`.

3.2.1 Multi-dimensional constraints

There are situations when a concept depends on the conjunction of constraints on multiple variables. For example, some areas of the terrain may be impassable for a rover robot. In such case, a hypothesis that imposes constraints on the location should be constrained in 2 dimensions.

The proposed method can be generalized to induce multi-dimensional constraints. This is achieved by taking the positive and negative values as tuples in the argument collecting step. Every collected tuple combines the values of multiple constrainable variables which follow when the head of the hypothesis clause is unified with examples. Then for each positive tuple, a constraint satisfaction problem is solved to find the largest area, or volume, that includes the positive tuple and excludes all of the negative tuples. The final discriminant domain is the disjunction of intervals for all positive tuples.

A rectangular area (or volume) is represented as two tuples $\mathbf{x}_{lower}$ and $\mathbf{x}_{upper}$. These two tuples express the bounds or the corner points of the area (or volume) as illustrated in Figure 3.3. Let x^i_{lower} and x^i_{upper} indicate the bound values coming from the i_{th} variable. Then, the area can be calculated as $\prod_i (x^i_{upper} - x^i_{lower})$. Hence, it is straightforward to generalize the constraint satisfaction problem defined in Table 3.4 to induce multi-dimensional constraints.

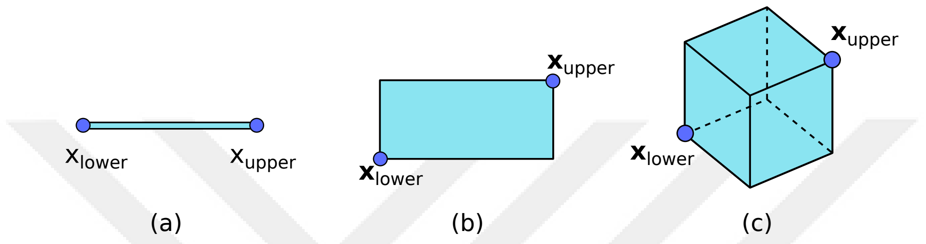

Figure 3.3 : Bounds of an interval (a), an area (b), a volume (c).

3.3 Noise Removal

We use the Local Outlier Factor (LOF) method [15] to remove the noisy inputs among the data given to the constraint solver. An input sample is considered as noisy if its local density is lower than the densities of its neighbors. The LOF method produces an *outlier factor* value for each sample that indicates how noisy the sample is. This allows a tuning mechanism on the degree of noise removal.

For each sample m_c, the set of k-nearest neighbors $m'_1 ... m'_k$ are found. Given the Euclidian distance between m_c and m'_i as $dist(m_c, m'_i)$, the density indicator value d_{m_c} is assigned as $d_{m_c} = 1/\max_i(dist(m_c, m'_i))$. That is the inverse of the maximum distance from m_c to any m'_i. Using the densities, the LOF value of m_c is calculated as in Equation (3.1). The sample m_c is marked as a noise if $LOF_k(m_c)$ is greater than a given threshold parameter t. The output of noise removal depends on parameters k and t.

$$LOF_k(m_c) = \frac{\sum_{m'_i} d_{m'_i}}{k \cdot d_{m_c}} \qquad (3.1)$$

Noise removal module is activated before the constraint solver call during the ILP learning (Figure 3.1). Input of the constraint solver is passed to this module and filtered from noisy data points.

One weakness of this noise removal approach is that the success of filtering depends on the chosen k and t parameters. To obtain a general method, different strategies can be employed. The optimal value of k is dependent on the densities of the clusters in data which is unknown a priori. Also too small and too large values of k are problematic. The authors of [15] recommended running the algorithm for multiple values of k and assuming the maximum outlier factor for each data point. However, since the noise reduction step is repeated for every constrainable hypothesis, this approach is expensive. For this reason, another strategy is used within the proposed learning system. A k value that is proportional to the number of total samples N, is chosen as $k = N/8$. The possible k values are also bounded from below and above ($k \in [15, 100]$), preventing extreme values. This strategy was effective in the experiments. A constant value is used for t as 1.25.

3.4 Drift Detection

The system for learning symbolic numerical constraints is augmented with a meta-learning architecture to gain the ability of hypothesis verification (Figure 3.1). The learning system works together with helper modules in order to detect a concept drift and re-learn hypotheses that are compatible with the latest observations. A drift detection method that monitors the error rates of the existing hypothesis set is implemented [18]. *Drift detector* listens to the *validator* which reports the success of *predictor* in guessing the labels of observations. If a drift is detected, then the drift detector triggers the base learner with the new training data.

Predictor module depends on the *background knowledge* provided by the domain expert and the *hypothesis set* on predicting the class of an observation. It uses the logic reasoning provided by Prolog. Validator records the prediction and assesses the success when the corresponding class becomes available, i.e., the robot observes the outcome of its action. Validator informs the drift detector about the observations and the correctness of predictions. Drift detector selects a trail of observations and sends it to the base learner when a drift is identified. Base learner uses the training set and background knowledge to produce a new hypothesis set.

Beginning and ending of the new training data is determined statistically by the drift detector. Drift detecting method models the event of an incorrect prediction as a Bernoulli trial [18]. Consequently, the error rate is a binomial random variable. Expected value $\hat{p}_i$ and the standard deviation $\hat{\sigma}_i$ of the error rate is then estimated with respect to the i^{th} prediction-observation matching. The method keeps track of the minimum observed error rate as $\hat{p}_{min} = min_i(\hat{p}_i), \hat{\sigma}_{min} = min_i(\hat{\sigma}_i)$. $\hat{p}_{min}$ and $\hat{\sigma}_{min}$ are used as a reference point. Whenever, the current error rate measure $\hat{p}_i + \hat{\sigma}_i$ exceeds $\hat{p}_{min} + 2.\hat{\sigma}_{min}$ the drift detection method sets a *warning point*. If the rise in the error rate measure reaches to a value that exceeds $\hat{p}_{min} + 3.\hat{\sigma}_{min}$, then, the method confirms the drift. The new training set is built with all of the observations from the warning point until the drift point [18]. Examples of warning points and a drift point are shown in Figure 3.4.

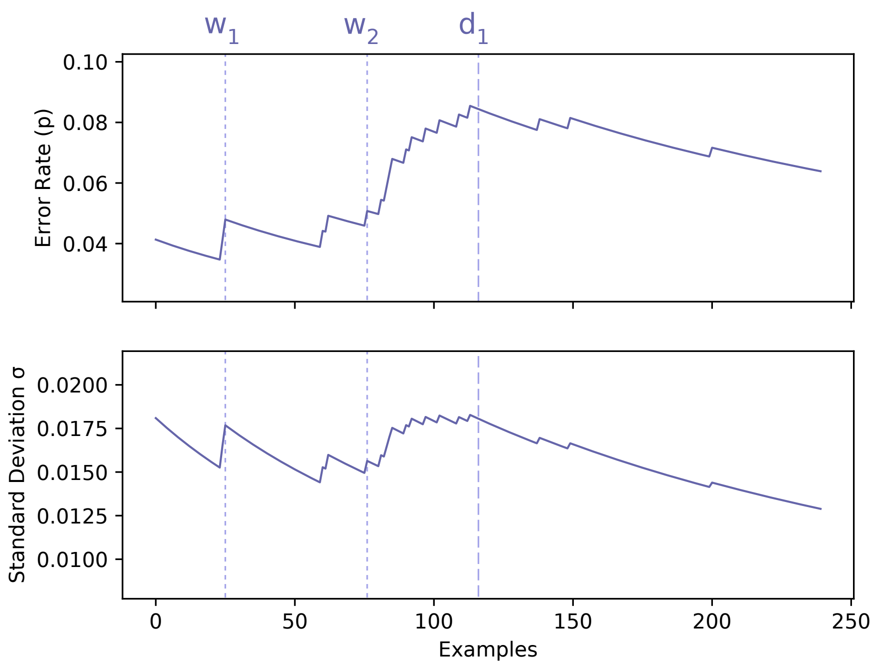

Figure 3.4 : The error rate over time in an experiment with a concept drift. w_1 and w_2 are the warning points set by the drift detector. d_1 is the point where the drift detector decides to update the hypothesis set.

There is a possibility that the warning point is set falsely because of noise. Drift detector can reset a warning point if the error rate falls below. This mechanism is shown to differentiate the noise from concept drift effectively.

Figure 3.4 shows an example of the error rate change during a concept drift. The drift is observed when the distance criteria is changed during the course of blocks-world object placement scenario (See Section 3.1). The warning point w_1 is set before the concept drift due to noise. It was reset at w_2 when a real concept drift starts. Then, the hypotheses are updated at point d_1. Only the examples between w_2 and d_1 are included in the new training set.

An issue observed during the development is that when a drift is detected, the size of the training set may be insufficient to learn effectively. To alleviate this issue, a *minimum window size* parameter is added to the method. If the number of examples in the new training set is lesser than the minimum window size, the systems delays updating the hypotheses until necessary number of examples are collected. This approach yiels better results in the experiments, hence it is adopted as the default approach.

4. EVALUATION

Multiple experiments are done to evaluate different aspects of the proposed learning system. The core aspect of the system is learning numerical constraints in a relational setting. At first, the learning results are evaluated with generated data and shown that it can induce constraints of both single and multi-dimensions while the existing methods cannot. The real-world experiments are conducted with a robot arm and a humanoid robot on the table-top environment. The real-world experiments point to the necessity of a noise filtering step. It is observed that the addition of noise removal module increases the generality of the learned hypotheses. Finally, both the generated and the real-world experiments are done to simulate concept drift. The drift detection enhancement is analysed in comparison to a baseline method which updates the hypotheses in a fixed period.

The next section presents the experiments done with the generated data to test the core learning aspect. The real-world experiments follows in Section 4.2.

4.1 Core Learning Experiments

The evaluation of the core learning method is done by two experiments. In the first one, the blocks-world *object placement* scenario is implemented (See Section 3.1). In the second experiment, a scenario with a *mobile robot* collecting objects in a 2-dimensional area is assumed. This experiment requires the learning system to be able to derive multi-dimensional constraints.

In addition to the proposed method, two other learning methods are applied on the data. The first one is referred as the *guessing method.* The guessing method depends on the ability of Aleph on introducing constants to the hypotheses. The second one is the original lazy evaluation method together with a attribute-based linear discriminator algorithm. The learning setting is the same for all three approaches. The used background knowledge is also the same except some differences in each method. The guessing method has the definitions of comparison operators ($\leq$, $\geq$, $=$), the lazy evaluation method has the definition of an attribute-based linear discriminator algorithm and the proposed method has the specification of constrainable terms.

4.1.1 Object placement scenario

Data generation

A dataset with 96 examples is generated for the object placement scenario. Differently from Section 3.1, a *color* attribute is added for every object. A color condition is added to the target concept defined in the blocks-world scenario. The target concept is the failure of put action. If the target location is *too close* to any block on the table and the carried object is *red*, then a failure occurs. The purpose of the color condition is to test the compatibility of our extension with the classical induction of ILP.

Results

Basic lazy evaluation cannot derive the required rules because it tries to find a discriminator which is complete for all input arguments collected from positive examples. To better express its limitation, assume the examples presented in Section 3.1. In the hypothesis ‘`put_on_close(A,B) :- on_table(C), position(C,E), distance(E,B,F), linear_discriminator(F,G)`’, `F` variable is the input of linear discriminator procedure. Argument collection yields {2,5} as the positive values and {1,7} as the negative values. There exists no linear discriminator for discriminating these two sets of values. The problem occurs, because the method tries to find a complete hypothesis which covers for every positive value. But some of the values are not relevant.

The guessing method can find a hypothesis which states that the failure occurs if the distance of the put object to any other object is equal to either 1 unit or 12 units.

The proposed learning method finds the desired hypotheses successfuly as shown in Table 4.1. This confirms that the method works as expected in ideallistic conditions.

4.1.2 Mobile robot scenario

Data generation

In the *mobile robot* scenario, the target concept is the failure of collecting an object on the ground. The robot is assumed to be unable to pick objects in a rectangular area, if the object color is red. 8 positive and 32 negative examples are generated as shown in Figure 4.1. Object positions are discretized in space. Since there may be more than one observations made in the same location, in some locations more than one markers are shown. For example, picking up a green object was successful in position $(7,3)$ while picking up a red object was failed in the same place. As seen in the figure, picking up action fails in the area defined as $5 \leq x_i \leq 8$, $y_i \leq 4$, where (x_i, y_i) is the position of i_{th} object. The robot can pick the green and blue objects even when they are in this area.

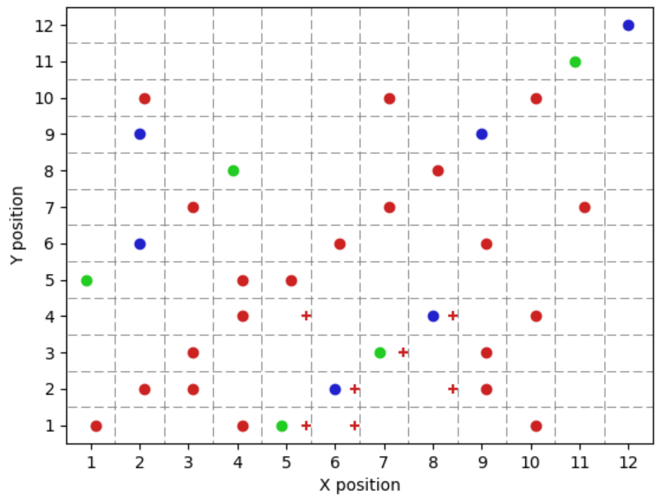

Figure 4.1 : Object picking observations in mobile robot scenario. '+' markers indicate failures, '•' markers indicate successes. Markers colors indicate the object colors.

Results

In this scenario both the lazy evaluation and guessing method cannot learn the concept. Limitation of the lazy evaluation with a linear discrimination method is explained in the previous scenario. The guessing method cannot succeed since deriving multi-dimensional constraints requires a special effort to consider the domains of both variables together.

The proposed learning method can learn the concept as shown in Table 4.1. This validates that the method can learn multi-dimensional hypotheses with idealistic data collection.

Table 4.1 : Hypotheses found in the core learning experiments.

Object placement experiment	
`put_on_fail(A,B) if:`	`on_table(C) and position(C,D) and distance(D,B,E) and E ≤ 1 and color(A,red).`
Mobile robot experiment	
`pickup_fail(A) if:`	`positionX(A,B) and positionY(A,C) and 5 ≤ B ≤ 8 and C ≤ 4 and color(A,red).`

4.2 Real-world Experiments

The real-world experiments are done on two different scenarios; *object placement* and *pouring pieces*. The first scenario is the blocks-world object placement experiment (Figure 4.2). The target concept of the experiment is the same as the other object placement experiments which appeared in this thesis. This experiment includes the uncertainty of the real-world environment and the robot hardware. The second scenario, *pouring pieces*, aims to test both the multi-dimensional constraint induction and the drift detection aspects of the proposed system. This scenario involves a robot that is responsible to pour the tiny pieces from a jar into a container (Figure 4.3). If the robot cannot pour enough pieces or it spills many pieces to the table, then the pouring action is assumed to be failed.

In both experiments, the robot learns hypotheses which explain the causes of action failures in a supervised setting.

4.2.1 Object placement scenario

Data collection

A scenario similar to the blocks-world domain (Section 3.1) is experimented with a real robot system. A 7-DOF Cython Veta robot arm is used for actuation and an ASUS Xtion Pro RGB-D camera is used for sensing. In this scenario, the robot has the task

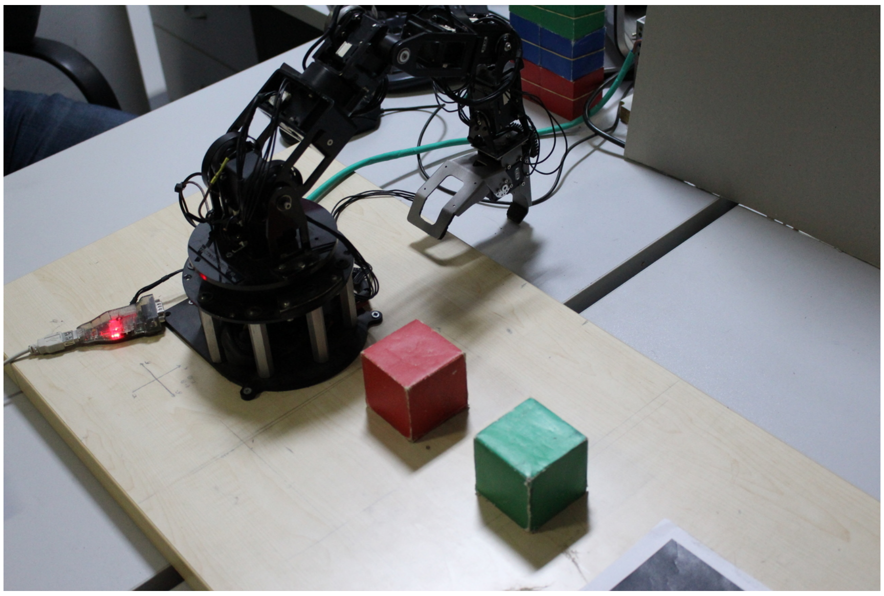
Figure 4.2 : Real-world experiment environment with the robot arm and two blocks.

of putting a block near to another one with a given distance. The robot interpretes the scene with an existing scene interpretation system [29]. Cases of success and failure are labeled by the scene interpretation module of the robot. Every block is a cube with 5.8 cm size in each dimension. The experiment is repeated with various distances and object colors and a total of 48 observations are collected. Every observation has the world state facts and the labels that are extracted by the scene interpreter. The observations involve noise from the sensors and actuators of the robot, and the stochastic environment.

Results

The core constraint induction method has finds the distance constraint as *5.1 cm*. The found hypothesis is given in Table 4.2. This constraint is erroneous since the size of a cube is 5.8 cm, and therefore the distance constraint should be larger than that. However, this hypothesis is supported by the noisy data collected from the real-world. This result shows that a noise removal procedure is necessary to prevent over-fitting.

Table 4.2 : Hypothesis learned for putting objects near each other in the real-world.

```
put_on_fail(A,B) if:    on_table(C) and
                        position(C,D) and
                        distance(D,B,E) and E ≤ 51.
```

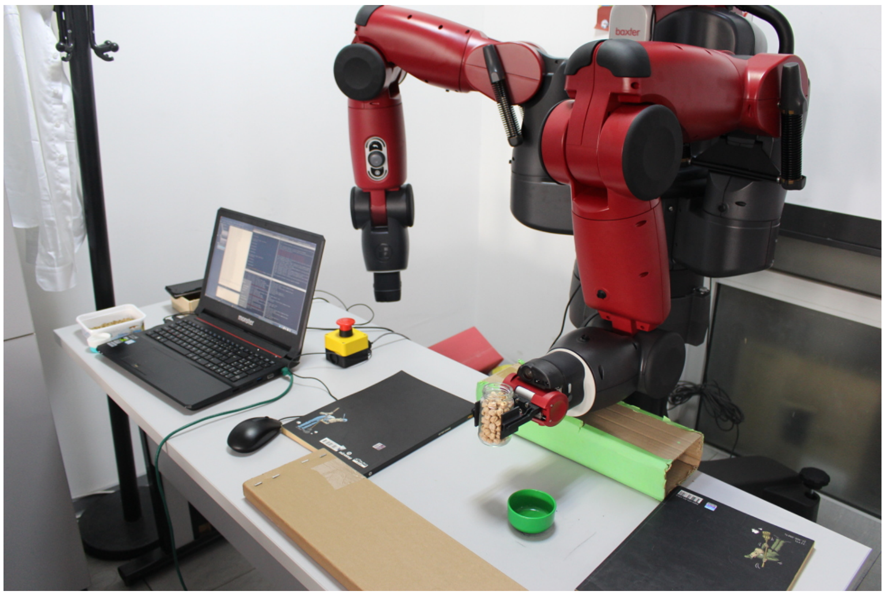

Figure 4.3 : Pouring experiment is done with a humanoid robot.

Figure 4.4 shows the distribution of distances and their labels. Intersections of distances among positive and negative examples indicate the high-level noise. Some of the noisy examples are removed using the LOF approach. The result of the noise filtering is also shown in Figure 4.4. The learning system enhanced with the noise removal module learns the distance constraint as *5.5 cm*.

4.2.2 Pouring pieces scenario

Data collection

In this experiment, pouring action is done by a Baxter humanoid robot. A small jar is used to hold 111 pieces of chicken peas. The robot holds the jar and pours it into a container on the table as shown in Figure 4.5. There are two types of containers used in the experiment, a rectangular package of cottage cheese ($14 \times 12 \times 4cm$) and a cylindrical sauce bowl ($7 \times 7 \times 3.5cm$). Pouring action is repeated from various heights with different angles between 0° to 180°. A total of 505 examples are collected. Unlike the previous real-world experiment, labeling is done by a human supervisor. Hence, the source of the noise is not perception, but actuation and stochasticity of the environment. The pouring action is assumed successful if a sufficient number of chicken peas are landed into the container while not many are spilled around. 318 of the examples are

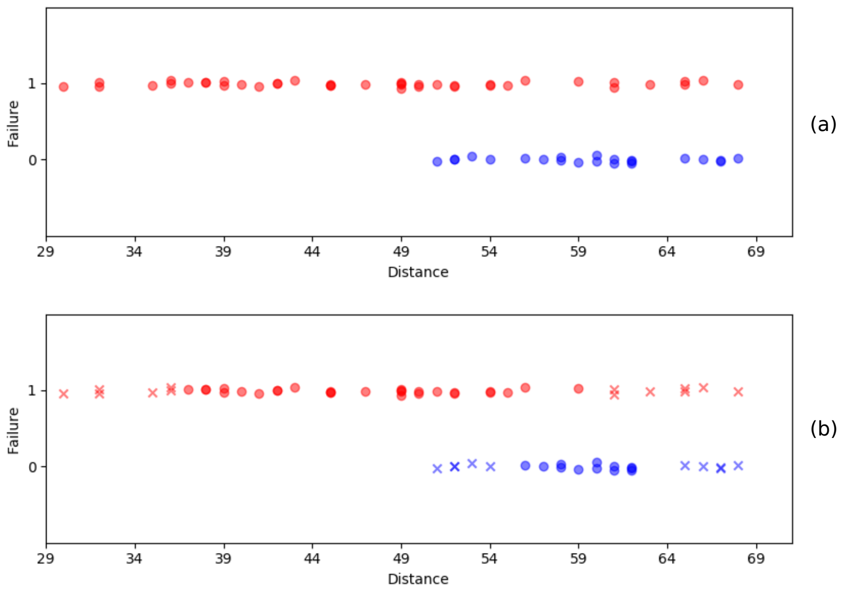

Figure 4.4 : Scatter graphs of distances in the real-world experiment with the robot arm. Y axis indicates the label of examples. (a) Original data. (b) After LOF noise filtering.

positive, indicating that the pouring action is failed. The remaining 187 examples are negative.

The bowl type is changed from cheese package to sauce bowl during the data collection to create a concept drift. Height and angle constraints are expected to be different since it is harder to pour into the sauce bowl. Preceding 299 of the examples are collected with the cheese package and 206 of them are collected with the sauce bowl.

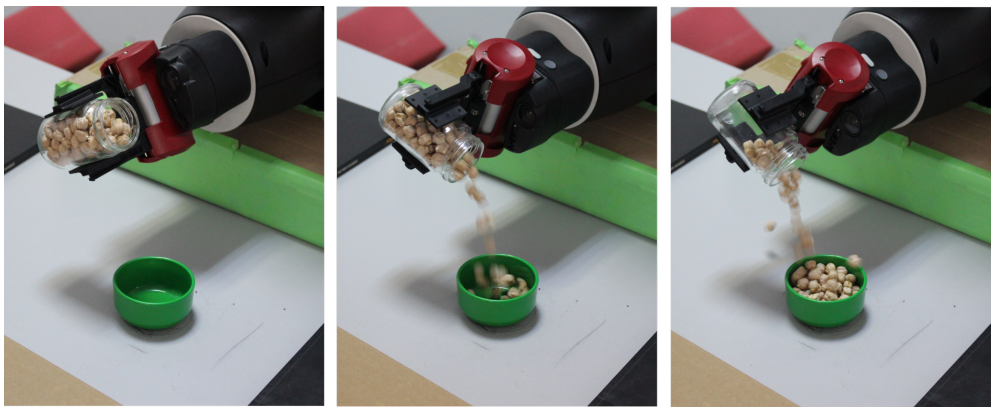

Figure 4.5 : The robot tries to pour enough pieces without spilling out.

Results

The pouring experiment is used in two evaluations. Firstly, a 6-fold cross validation is applied on the subset of examples that are collected only with the cheese package. Results show that the learned hypotheses generalize sufficiently, as the testing accuracy has the expected value of *0.85* with standard deviation of *0.03*. Generalization is achieved with a disjunction of hypotheses, some of which contain multi-dimensional constraints as shown in Table 4.3. This hypothesis set indicates that a pouring action fails if the height is higher than 26 cm, or the difference between the pouring angle and the height is less than or equal to 90, or the diference is lesser than or equal to 99 *and* the height is lesser than or equal to 12. This result is obtained with no parameter tuning or problem specific treatments. It indicates that the learning method is appropriate for noisy data collected in the real-world environment. The learned hypotheses show that the system can create complex relations between multiple variables when necessary.

Table 4.3 : Some of the hypotheses learned by pouring pieces into the cheese package.

`pour_fail(A) if:`	`height(A,B) and angle(A,C) and distance(C,B,D) and D` $\leq$ `84.`
`pour_fail(A) if:`	`height(A,B) and B` $\geq$ `32.`
`pour_fail(A) if:`	`height(A,B) and angle(A,C) and distance(C,B,D) and D` $\leq$ `93 and B` $\leq$ `18.`

The second experiment evaluates system on the concept drift problem. In this evaluation, 161 examples collected with the cheese package are used to train the initial model, and the rest of the data are used for testing. First 120 examples of the testing data are still with the cheese package. However, after these examples, the container type is assumed to be changed. The rest of the data consist of 206 examples collected with only the sauce bowl. Two methods are compared in this experiment. One is the proposed learning system and the other is the base learning system with noise removal, but without the drift detection module. The minimum window size of the drift detector is given as *80*. The aim of this comparison is to observe the effects of the drift detection enhancement.

The testing data are given as a stream to both learning systems. The learning systems make predictions for each observation, and the total error rates are recorded. This process is repeated 8 times by shuffling the examples in the testing set folding the examples of the same container types together. In every test, the system with drift detection identified a drift and updated its hypothesis set at least once. Accuracy of the learning with drift detection is 0.81 ± 0.017 while without this enhancement it is 0.76 ± 0.03. The error rate plot graphs of some of the repetitions are given in Figure 4.6. The error rates are estimated only with the predictions done on the test data. The dotted lines represent the warning points and the dashed lines represent the drift points similar to igure 3.4. In all repetitions, new hypothesis sets improved the overall prediction accuracy of the system. As it is seen on Figure 4.6, the error rate keeps decreasing after the drift detection. As the number of examples after the drift increases, the difference between the error rates become larger. This means that a learning system without a

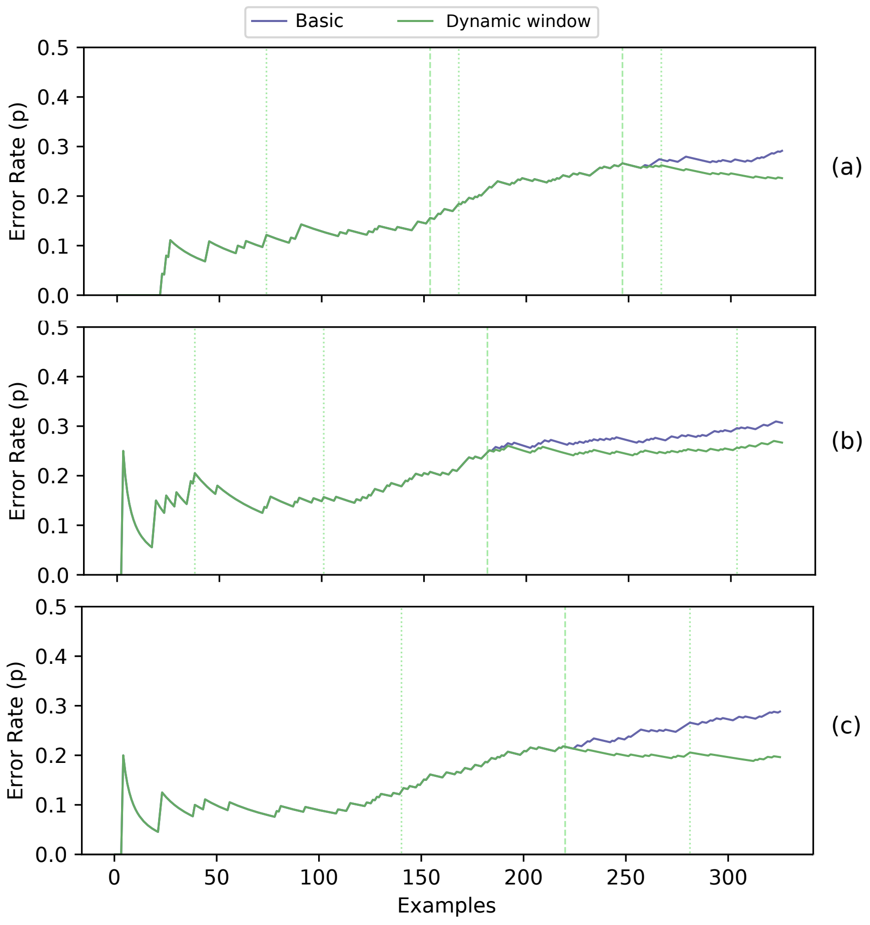

Figure 4.6 : Error rates of the drifting experiment on the pouring action.

drift detection improvement would have worse error rates as time passes until the drift is detected. The adaptive learning system clearly has an advantage against the basic system. The hypotheses learned before and after the drift, for the experiment whose error rates depicted in Figure 4.6.c are given in Table 4.4.

Table 4.4 : Hypotheses learned before and after drift

Initial hypotheses
`pour_fail(A) if:` `angle(A,B) and B =< 108`
`pour_fail(A) if:` `height(A,B) and B >= 32`
Hypotheses after drift
`pour_fail(A) if:` `height(A,B) and B >= 20`
`pour_fail(A) if:` `angle(A,B) and B >= 131`
`pour_fail(A) if:` `angle(A,B) and B =< 106`

5. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, a learning system which can learn symbolic-level numeric constraints under real-world conditions is presented. The method combines an existing ILP system with a constraint solver to achieve constraint induction. This method is suggested for robots to learn numeric constraints of predicates in lifelong learning scenarios using their own observations. Modeling the physical limitations of a robot, its tasks and its environment as relational rules with numerical constraints can increase the task execution performance.Noise removal and concept drift detection modules enhance the robustness of the system. In this aspect, the presented system is among the very few robot systems that take concept drifts into account.

The results of the experiments show that the proposed method can induce the concepts which cannot be induced by the alternative ILP methods. It is shown that with the applied extensions, the learning system can generate more robust hypotheses on the numerical constraints from the observations of the robot. Robot experiments show that the system is applicable in real-world environments. Therefore, this method alleviates the need for the domain experts to extract the numerical constraints of the environment and the robot. This gives robots flexibility in life-long learning scenarios as they may re-learn the constraints themselves. Also the system makes it possible to solve the scenarios where domain experts have limited knowledge about the environment.

The current system is not be able to learn from small observation windows which contain insufficient number of examples to be generalized by ILP. In such cases, more examples can be collected with *active learning* methods.

Recurring concept drift is a change in the concept to a previous state. Keeping the old hypotheses can be more efficient than learning again, in case of recurring concepts. The system can benefit from such a feature.

Integration of probabilistic reasoning with constraint induction may further increase the robustness of the system in real-world conditions. Furthermore, the system can be enhanced with incremental learning capabilities for better adaptation to changing environment conditions. Predicate invention capability may be useful in long term scenarios.

REFERENCES

[1] **Karapinar, S. and Sariel, S.** (2015). Cognitive robots learning failure contexts through real-world experimentation, *Autonomous Robots*, *39*(4), 469–485.

[2] **Srinivasan, A.**, (2001), The aleph manual.

[3] **Stansbury, R.S. and Agah, A.** (2012). A robot decision making framework using constraint programming, *Artificial Intelligence Review*, *38*(1), 67–83.

[4] **Peters, J.**, **Vijayakumar, S. and Schaal, S.** (2003). Reinforcement learning for humanoid robotics, *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp.1–20.

[5] **Argall, B.D.**, **Chernova, S.**, **Veloso, M. and Browning, B.** (2009). A survey of robot learning from demonstration, *Robotics and autonomous systems*, *57*(5), 469–483.

[6] **Bratko, I. and Muggleton, S.** (1995). Applications of inductive logic programming, *Communications of the ACM*, *38*(11), 65–70.

[7] **Džeroski, S.**, **De Raedt, L. and Driessens, K.** (2001). Relational reinforcement learning, *Machine learning*, *43*(1-2), 7–52.

[8] **Srinivasan, A.**, **Muggleton, S.H.**, **Sternberg, M.J. and King, R.D.** (1996). Theories for mutagenicity: A study in first-order and feature-based induction, *Artificial Intelligence*, *85*(1-2), 277–299.

[9] **Santosh, K.**, **Lamiroy, B. and Ropers, J.P.** (2009). Inductive logic programming for symbol recognition, *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, IEEE, pp.1330–1334.

[10] **Moldovan, B.**, **Moreno, P.**, **van Otterlo, M.**, **Santos-Victor, J. and De Raedt, L.** (2012). Learning relational affordance models for robots in multi-object manipulation tasks, *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, pp.4373–4378.

[11] **Botta, M. and Giordana, A.** (1993). SMART+: A multi-strategy learning tool, *IJCAI*, Citeseer, pp.937–945.

[12] **Muggleton, S.** (1995). Inverse entailment and Progol, *New generation computing*, *13*(3-4), 245–286.

[13] **Srinivasan, A. and Camacho, R.** (1999). Numerical reasoning with an ILP system capable of lazy evaluation and customised search, *The Journal of Logic Programming*, *40*(2), 185–213.

[14] **Hodge, V. and Austin, J.** (2004). A survey of outlier detection methodologies, *Artificial intelligence review*, *22*(2), 85–126.

[15] **Breunig, M.M.**, **Kriegel, H.P.**, **Ng, R.T. and Sander, J.** (2000). LOF: identifying density-based local outliers, *ACM sigmod record*, volume 29, ACM, pp.93–104.

[16] **Žliobaitė, I.**, **Pechenizkiy, M. and Gama, J.**, (2016). An overview of concept drift applications, Big Data Analysis: New Algorithms for a New Society, Springer, pp.91–114.

[17] **Tsymbal, A.** (2004). The problem of concept drift: definitions and related work, *Computer Science Department, Trinity College Dublin*, *106*(2).

[18] **Gama, J.**, **Medas, P.**, **Castillo, G. and Rodrigues, P.** (2004). Learning with drift detection, *Brazilian Symposium on Artificial Intelligence*, Springer, pp.286–295.

[19] **Bifet, A. and Gavalda, R.** (2007). Learning from time-changing data with adaptive windowing, *Proceedings of the 2007 SIAM International Conference on Data Mining*, SIAM, pp.443–448.

[20] **Salganicoff, M.** (1997). Tolerating concept and sampling shift in lazy learning using prediction error context switching, *Artificial Intelligence Review*, *11*(1-5), 133–155.

[21] **Wang, H.**, **Fan, W.**, **Yu, P.S. and Han, J.** (2003). Mining concept-drifting data streams using ensemble classifiers, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, AcM, pp.226–235.

[22] **Procopio, M.J.**, **Mulligan, J. and Grudic, G.** (2009). Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments, *Journal of Field Robotics*, *26*(2), 145–175.

[23] **Solak, G.**, **Ak, A.C. and Sariel, S.** (2016). Experience-based learning of symbolic numerical constraints, *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, IEEE, pp.1264–1269.

[24] **Sebag, M. and Rouveirol, C.** (1996). Constraint inductive logic programming.

[25] **Knorr, E.M.**, **Ng, R.T. and Tucakov, V.** (2000). Distance-based outliers: algorithms and applications, *The VLDB Journal—The International Journal on Very Large Data Bases*, *8*(3-4), 237–253.

[26] **Ertöz, L.**, **Steinbach, M. and Kumar, V.** (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data, *Proceedings of the 2003 SIAM International Conference on Data Mining*, SIAM, pp.47–58.

[27] **Xiong, H.**, **Pandey, G.**, **Steinbach, M. and Kumar, V.** (2006). Enhancing data analysis with noise removal, *IEEE Transactions on Knowledge and Data Engineering*, *18*(3), 304–319.

[28] **Triska, M.**, (2014), Library (clpfd): Constraint Logic Programming over Finite Domains.

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

CURRICULUM VITAE

Name Surname: Gökhan Solak

Place and Date of Birth: Eskisehir, Turkey - 17.12.1991

E-Mail: solakg@itu.edu.tr

EDUCATION:

- **B.Sc.:** 2013, Anadolu University (AU), Electrical/Electronics Engineering (English)

PUBLICATIONS:

- Gokhan Solak, Abdullah Cihan Ak, and Sanem Sariel, 2016. Experience-based learning of symbolic numerical constraints. *In Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on Humanoids Robots*.
- M Akif Gunes, Gokhan Solak, Ugur Akin, Omer Erden, and Sanem Sariel, 2016. A generic approach for player modeling using event-trait mapping and feature weighting, *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.