상황 관리를 위한 에피소딕 메모리의 설계 및 구현

이재윤,이기호,김인철
경기대학교 컴퓨터과학과
email: jaeyoon_95@kyonggi.ac.kr,rlgh9250@kyonggi.ac.kr,kic@kyonggi.ac.kr

Design and Implementation of Episodic Memory for Context Management

Jaeyun Lee, Giho Lee, Incheol Kim
Department of Computer Science, Kyonggi University

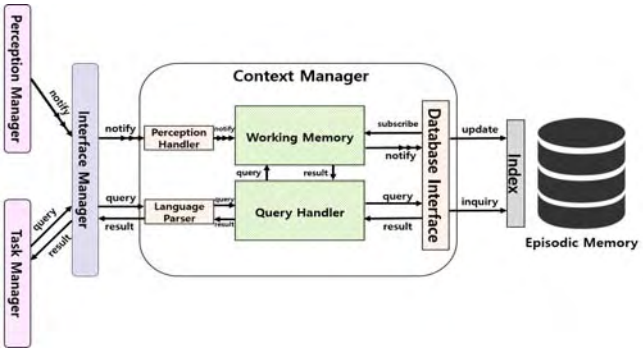
요 약

본 논문에서는 실생활 환경에서 동작하는 지능형 서비스 로봇의 효과적인 상황 관리를 위한 에피소딕 메모리의 설계와 구현을 제안한다. 지능형 서비스 로봇은 실시간 센서 데이터로부터 상황 정보를 끊임없이 생성, 갱신해야 할 뿐만 아니라, 추후 참조와 학습을 위해 상황 정보들을 데이터베이스나 파일과 같은 비휘발성 장기 메모리에 저장, 관리할 필요가 있다. 본 논문에서는 이러한 요구에 부응하여, 지능형 서비스 로봇의 실시간 상황 정보 관리에 효과적인 에피소딕 메모리의 구조와 기능들을 설계하고 구현하였다. 정량적 실험을 통해, 본 본문에서 제안한 에피소딕 메모리의 높은 성능과 유용성을 확인하였다.

1. 서론

지능형 서비스 로봇은 외부환경을 인식하고, 스스로 상황을 판단하여, 자율적으로 동작하는 로봇을 의미한다. 이러한 점에서 지능형 서비스 로봇은 기존의 로봇과 차별성이 존재한다. 상황 판단과 자율적인 동작을 위해 로봇은 기초 지식 또는 과거에 발생한 사건에 대한 지식이 필요하다. 이를 위해 메모리는 장기적으로 기초 지식과 하나의 장면의 시간, 공간의 맥락과 관련 있는 정보 등의 정보를 저장해야한다. 시간, 공간의 정보는 장면에서 인식된 물체의 속성 및 위치 정보, 인식된 시간 정보, 물체들 간에 관계 정보 등을 저장한다. 지능형 서비스 로봇은 과거 정보들을 저장함으로써 현재 상황을 판단할 수 있고, 자율적인 동작을 할 수 있다. 본 논문에서는 지능형 서비스 로봇의 지식을 장시간 저장하기 위해서 비휘발성 메모리를 활용한다.

지능형 서비스 로봇의 상황을 판단하기 위해 [4, 5]와 같은 로봇 컨트롤 시스템이 필요하다. 본 논문에서는 입력받은 상황 정보를 판단하기 위해 상황 관리자(Context Manager)를 사용하였다. 또한, 센서 데이터를 저장하기 위해 상황 관리자에 비휘발성 메모리인 EM(Episodic Memory)을 추가하였다. 상황 관리자의 구조는 (그림 1)과 같으며, PM(Perception Manager), TM(Task Manager) 등을 총괄하여 관리한다. 상황 관리자는 센서 데이터를 받아들여 WM(Working memory)에 전달하고, 일정 시간 이상 지나면 센서 데이터를 인코딩시켜 EM에 저장한다. 작업 관리자에서는 메모리에 저장된 정보에 대해 질의를 보낸다. 상황 관리자로 전달된 질의는 파싱을 거쳐 WM과 EM 중 어느 메모리에 전달해야 할지 판단한 뒤, 질의를 한다. 이때 현재 시점과 가까운 질의는 WM로, 현재 시점과 일정 시간 이상 차이가 나는 질의는 데이터베이스 인터페이스를 거쳐 EM으로 전달된다. 본 논문에서는 PM으로부터의 센서 데이터를 참조 방식을 이용하여 효율적으로 저장하는 구조와 저장한 데이터를 질의에 따라 필요한 인식 정보를 적절하게 WM 혹은 EM에서 검색하여 인출하는 방식의 시스템에 대하여 제안한다.

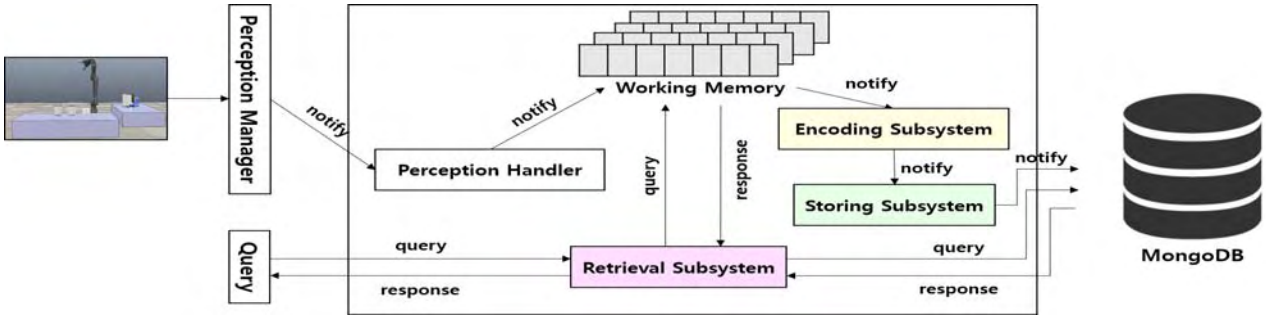


(그림 1) 상황 관리자 시스템 구조

2. 관련 연구

[1]에 따르면 사람의 메모리 시스템은 크게 작업 메모리, 단기 메모리, 장기 메모리 세 가지로 나누어진다. 작업 메모리는 일시적인 저장소로 인식, 추론, 계획 등의 정보를 저장하는 메모리이다. 현재 인식하고 있는 일시적인

* 본 연구는 산업통상자원부의 재원으로 기술혁신사업의 지원을 받아 수행한 연구과제 (No. 10060086, 개인 서비스용 로봇을 위한 지능-지식 집약, 개방, 진화형 로봇지능 소프트웨어 프레임워크 기술 개발)입니다.



(그림 2) 에피소딕 메모리 서브 시스템 구조

정보들 즉, 최소 몇 초에서 최대 몇 분까지의 정보를 저장한다. 단기 메모리는 작업 메모리와 유사한 일시적인 저장소이며, 정보들이 장기 메모리로 가기 전 거치는 메모리이다. 저장 시간은 최소 3시간에서 최대 6시간까지로, 작업 메모리보다 길다. 단기 메모리는 작업 메모리와 동일한 인식, 추론, 계획 등의 정보를 저장한다. 에피소딕 메모리는 하나의 에피소드 즉, 하나의 장면의 시간, 공간의 맥락과 관련 있는 정보를 저장한다. 상황 관리자에서 구현된 메모리는 작업 메모리와 장기 메모리 내에 있는 에피소딕 메모리이다.

에피소딕 메모리는 인코딩, 저장, 인출의 중요 3가지 시스템 형태로 구성되어진다. 인코딩 시스템은 물리적, 감각적인 입력 정보를 기억에 저장할 수 있는 표상의 한 형태로 전환하는 시스템이다. 저장 시스템은 인코딩된 정보를 메모리에 담아두고, 어떻게 유지를 시키고, 삭제를 시킬지에 대한 알고리즘을 적용시키는 시스템이다. 마지막으로 인출 시스템은 메모리에 저장되어 있는 정보에 대하여 접근하는 시스템이다.

[2]에서는 작업 계획 정보와 인식 정보를 인코딩 서브시스템을 통해 각각 다른 저장 장치에 저장한다. 작업 계획 정보는 로봇이 수행했던 작업의 계층적 정보로, 작업 중 수행한 서브 작업들의 정보 또한 포함한다. 작업 계획 정보는 서브 작업을 통해 상위 작업이 완료되는 일정한 구조를 가지며, 이를 저장하기 위해 OWL 표현 언어로 저장한다. 인식 정보는 물리적, 감각적인 입력 정보로, 이미지, 인식시간, 물체 종류 등의 정보이다. 인식 정보는 매 초마다 수십에서 수백에 달하는 정보가 들어오며, 정보의 형식이 다양해 NoSQL 데이터베이스인 MongoDB에 저장한다. 저장 서브시스템은 인식 정보를 통해 기존 지식과의 차이를 일정 이상 인지할 경우 지식을 갱신한다(예를 들어 새로운 지식에 없던 새로운 물체의 등장). 인출 서브 시스템은 복잡한 MongoDB 질의어 대신 PL(Prolog Language)를 사용하며, 저장된 작업 계획 정보와 인식 정보를 기반으로 상위 지식을 추론한다.

[3]은 [2]와 달리 인식 정보만을 인코딩하여 저장하며, 인식 장면과 장면에 대한 해석을 포함한다. 인식 정보는 MongoDB에 물체의 속성, 인식 장면, 부모 컬렉션 등 여러 종류의 컬렉션에 저장된다. 저장된 지식 정보의 갱신은 [2]와 유사하게 새로 받아들이는 인식 정보와 일정 임계값 이상 차이가 날 경우 이루어진다(예를 들어 새로운 물체의 인식). 인출 서브 시스템은 BNF문법을 선언하여 질의문을 간편화하였다. [2]에서는 인식 정보를 여러 컬렉션에 양방향 참조 방식을 사용하여 부모 컬렉션과 자식 컬렉션들의 형태로 저장한다. 이는 인출 서브 시스템에서 부모나 자식 컬렉션 어디에서도 정보에 접근이 편리하다는 장점이 있으나, 유사한 정보를 두 번 저장하는 결과를 가져와 메모리 공간 차지와 저장 속도에 있어 단점을 갖는다. 인코딩 측면에서는 [2]는 작업, 인식 정보를 모두 저장하기 때문에, [3]보다 인코딩 시간이 오래 걸릴 수 있다. 하지만 저장되는 정보의 종류가 다양하므로, 상황에 대한 깊은 이해를 할 수 있다. 저장 측면에서는 [2, 3]모두 저장된 정보와 새로운 정보를 비교하여 일정 값 이상 차이가 날 경우에만 정보를 갱신한다. 인출 측면에서는 [2, 3]모두 MongoDB 질의어가 아닌 다른 언어를 사용하여 데이터를 인출한다.

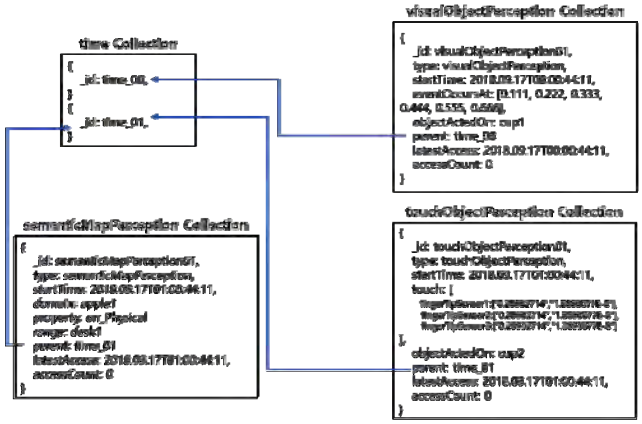
3. 에피소딕 메모리 시스템

본 논문이 제안하는 에피소딕 메모리 모델은 (그림 2)와 같이 인코딩 서브 시스템, 저장 서브 시스템, 인출 서브 시스템으로 구성된다. PM에서 받은 센서 정보를 WM로 전달한다. WM의 저장 기간은 수십 초에서 수 분으로, 매 초마다 약 16개의 정보를 EM에 전달한다. EM의 저장 장치로 NoSQL 데이터베이스인 MongoDB를 사용한다. 인식 정보는 인코딩 서브 시스템과 저장 서브 시스템을 거쳐 EM에 저장된다. 인코딩 서브 시스템은 WM에서 보낸 센서 데이터를 인코딩시켜 MongoDB에 저장할 수 있는 형태로 만든다. 인코딩된 정보는 저장 서브 시스템으로 전달되어 MongoDB에 직접적으로 인코딩 데이터를 전달한다. 저장된 데이터를 검색하기 위한 질의가 들어오면, 인출 서브 시스템에서 질의를 처리한다.

3.1 MongoDB설계

MongoDB의 구조 설계는 인코딩 서브 시스템과 인출 서브 시스템의 동작 방식에 큰 영향을 준다. 효율적으로 설계된 에피소딕 메모리 구조는 인코딩 방식과 지식 인출 방식에 영향을 주어 계산량을 줄여 속도를 향상시킨다. 또한 데이터 저장의 중복을 최소화시켜 메모리 공간을 효율적으로 사용한다. 에피소딕 메모리 구조의 설계 방법에는

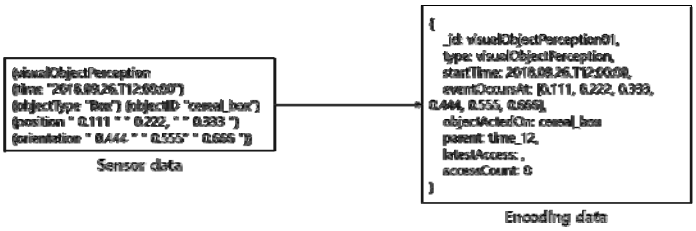
'One-to-Few', 'One-to-Many', 'One-to-Squillions' 등 여러 개의 형태가 존재한다. 'One-to-Few'는 참조하지 않는 설계 방법으로, 데이터의 중복이 발생할 수 있기 때문에 수많은 데이터를 저장하는 구조에는 적합하지 않다. 'One-to-Many'설계 방법은 부모 컬렉션에 참조되는 자식 컬렉션을 추가해 넣는 방법으로, 수많은 데이터를 저장하는 구조에 적합할 수 있다. 하지만 다큐먼트 하나의 크기가 16MB를 넘을 수 없는 MongoDB의 특성상 로그 데이터와 같은 대량의 데이터를 저장하기에는 해당 설계 방식은 적합하지 않다. 본 논문에서는 자식 컬렉션에서 참조하는 부모 컬렉션에 대한 정보를 저장하는 'One-to-Squillions'설계 방법을 채택한다. 앞서 설명한 설계 방법 외에도 [3]에서는 사용한 양방향 참조 방법이 채택하고 있으며, 이는 부모 컬렉션과 자식 컬렉션이 서로를 참조를 하는 방식이다. 이 방법을 사용하면 인출 서브 시스템에서 정보를 인출하기에 용이해지나, 불필요하게 많은 참조로 저장 공간이 낭비된다. 또한 양 방향에 참조를 정의해 주어야하기 때문에 데이터 저장시간이 증가할 수 있다. 본 논문에서는 이러한 단점을 'One-to-Squilions' 방식을 채택하여 극복한다. 본 논문의 스키마 구조는 (그림 3)와 같다. 질의에는 시간에 관련한 정보가 필수적으로 필요하다. EM은 에피소드를 저장하는 구조로 이는 더욱 두드러진다. 따라서 시간을 초기 기준으로 검색하는 것은 도큐먼트를 인출하는데 있어 도큐먼트 탐색 시간을 줄여준다. 본 논문에서는 시간을 24개의 도큐먼트로 나누어 구성하며, 이는 각 센서 데이터 도큐먼트의 부모 참조 도큐먼트로 이용된다.



(그림 3) MongoDB 스키마

3.2 인코딩 서브 시스템

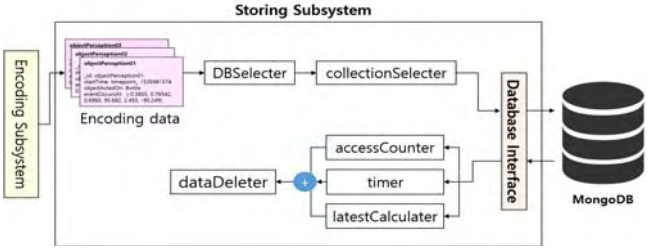
WM에 있던 센서 데이터가 인코딩 서브 시스템에 전달되면, 인코딩 서브 시스템은 3.1에서 설명한 데이터베이스의 구조에 맞게 Key-Value 형태로 데이터를 인코딩한다. (그림 4)에서 센서 데이터는 하나의 스트링 형태로 입력된다. 인코딩 서브 시스템에서는 스트링 형태의 센서 데이터를 타입 별로 나누고, 각 컬렉션 구조에 맞게 Key-Value 형태로 매칭 시킨다. 'type', 'startTime', 'eventOccursAt', 'parent'는 각각 데이터의 타입, 발생시간, 물체의 위치 값, 참조하는 'time Collection'의 '_id'값을 나타낸다. 각 데이터의 고유의 값인 "_id"는 데이터베이스 자체에서 생성하고, "latestAccess", "accessCount"는 데이터 검색을 할 경우에만 값이 갱신된다. 앞서 설명한 것처럼 센서 데이터를 (그림 4)와 같은 구조로 인코딩하면, 필요한 참조만을 하기 때문에 저장 시간과 저장 공간을 절약할 수 있다.



(그림 4) 인코딩된 센서 데이터

3.3 저장 서브 시스템

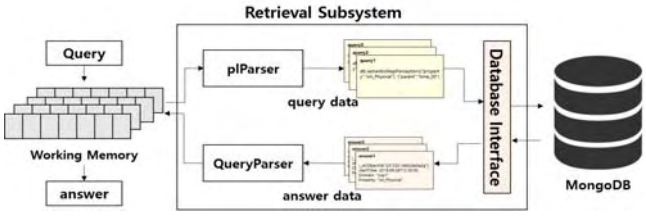
저장 서브 시스템은 인코딩 된 정보를 메모리에 저장, 유지, 삭제에 대한 알고리즘을 적용시키는 서브시스템이다. 데이터를 삭제하는 것이 아닌 갱신하는 [2, 3]의 구조와 다르게 본 논문에서는 삭제 알고리즘을 적용시켜 사용하지 않는 데이터를 삭제한다. 이의 장점은 정보의 저장량에 따라 증가하는 지식 인출 시간을 감소, 메모리 공간을 확보할 수 있다. 인간의 기억 시스템에서 망각은 기억 인출 주기와 겪은 에피소드의 노화에 따라 달라진다. 오래된 정보는 잊혀지기 쉬우며, 자주 인출한 지식은 잊혀지기 어렵다. (그림 5)는 본 논문의 저장 서브 시스템 구조이다. 인코딩된 데이터가 입력되면, DBSelecter는 저장할 데이터베이스를 선택한다. 데이터베이스가 선택된 후 collectionSelecter에 의해 해당 데이터가 들어갈 타입이 선택된다. 데이터베이스와 컬렉션이 모두 선택된 데이터는 MongoDB에 저장된다. 저장된 데이터는 검색할 수 있으며, 일정 시간이 지나면 삭제된다. accessCounter와 latestCalculater는 삭제에 필요한 정보인 accessCounter, latestAccess에 대한 계산을 해 주는 모듈이다. timer에서 일정 시간이 지나면, accessCounter와 latestCalculater는 각각 접근 횟수, 최근 접근 시간을 계산하며, 접근 횟수가 0이하이거나, 최근 접근 날짜가 하루 이상 차이 날 경우에만 dataDeleter에서 해당 데이터를 삭제한다. 데이터에 접근한 횟수가 많을수록 해당 데이터는 많이 사용하는 데이터이기 때문에 삭제를 하지 않고 남겨둔다. 또한 데이터에 많이 접근하더라도 자주 사용하지 않는다면, dataDeleter에서 해당 데이터를 삭제한다.



(그림 5) 저장 서브 시스템

3.4 인출 서브 시스템

본 논문에서는 [2]와 유사하게 PL형태로 질의 및 검색을 한다. MongoDB의 질의 언어는 SQL데이터베이스에 비해 매우 복잡하기 때문에 다음과 같은 방법을 사용하였다. 예를들어 책상위에 물체가 위에 있는지 확인하는 질의를 했을 때, 에피소딕 메모리 질의어는 “db.semanticMapPerception.find({‘property’:’on_Physical’}) “와 같다. 하지만 Prolog언어를 사용한 질의는 ”on_Physical (Top,‘table‘)“과 같이 매우 간단하다. (그림 6)은 인출 서브 시스템의 구조를 나타낸 그림이다. WM에 질의가 들어오면 해당 질의를 plParser를 통해 에피소딕 메모리 질의어로 변환해주고, 변환된 질의어는 MongoDB에서 직접적으로 정보를 인출한다. 인출된 정보는 WM에 전달되어 질의에 대한 응답을 추론하는데 사용된다. PL형태를 사용함으로써, 복잡했던 질의어를 간편화 할 수 있고, 동시에 시간 또한 단축할 수 있다.



(그림 6) 인출 서브 시스템

4.2 실험

4.2.1 저장 시간 측정

본 논문의 모델 저장 속도를 비교 하기 위하여 다음과 같은 실험을 진행하였다. ‘visualObjectPerception’, ‘touchObjectPerception’, ‘semanticMapPerception’에 대한 각각의 더미 데이터를 사용하였고, 1초마다 약 16개의 데이터를 MongoDB에 업로드 하였다. <표 1>의 ‘Ours’은 본 논문의 MongoDB모델이고, ‘OTO’, ‘OTM’, ‘BR’은 각각 ‘One-to-One Model’, ‘One-to-Many Model’, ‘Bidirectional Reference Model’를 의미한다. 총 데이터 수는 1000개 부터 5000개 까지 1000단위로 제공하였고, 모든 측정 단위는 초 단위 이다. <표 1>을 보면 데이터 개수가 달라지더라도 ‘Ours’가 저장하는데 가장 적은 시간이 걸린 것을 확인 할 수 있다. 특히 [3]의 모델과 비교했을 때 본 논문의 모델이 저장 속도가 더 빠르다는 것을 알 수 있다.

<표 1> 모델별 데이터 수에 따른 저장 속도[sec] 비교

Models	1000	2000	3000	4000	5000
Ours	**8.48**	**15.95**	**23.75**	**31.68**	**39.29**
OTO	8.96	71.50	26.00	34.34	42.60
OTM	10.63	20.61	31.66	42.32	46.25
BR[3]	9.87	20.85	29.27	41.46	50.34

4.2.2 저장 공간 측정

본 논문의 MongoDB 모델의 저장 공간을 비교 하기 위하여 다음과 같은 실험을 진행하였다. 실험 방법은 4.2.1과 같고, 모든 측정 단위는 MB이다.실험의 결과는 <표 2>와 같이 나타내었다. <표 2>에서 1000개의 데이터를 저장할 때부터 5000개의 데이터를 저장할 때 까지 모든 면에서 ‘Ours’가 가장 적은 메모리를 차지한다는 것을 확인 할 수 있다. 4.2.1의 실험과 마찬가지로 [3]의 모델과 비교했을 때 본 논문의 모델이 더 효율적으로 저장된다는 것을 알 수 있다.

<표 2> 모델별 데이터 수에 따른 저장 공간[MB] 비교

Models	1000	2000	3000	4000	5000
Ours	**0.36**	**0.73**	**1.09**	**1.45**	**1.86**
OTO	0.47	0.95	1.42	1.89	2.41
OTM	0.41	0.83	1.24	1.66	2.07
BR[3]	0.48	0.96	1.44	1.92	2.44

5. 결론

본 논문에서는 지능형 서비스 로봇의 지식을 저장할 수 있는 비휘발성 메모리인 에피소딕 메모리의 모델을 제안하였다. 본 논문에서 에피소딕 메모리의 세 가지 서브 시스템에 대한 설명을 하였고, ‘One-to-Few’, ‘One-to-Many’, ‘One-to-Squillions’ 와 ‘Bidirectional Reference’ 모델의 데이터 수에 따른 저장 속도와 저장 공간에 대한 실험을 통해 본 논문에서 제안한 모델의 성능을 분석하였다. 현재는 센서 데이터에 대한 정보만을 저장 가능 하지만 작업 정보를 저장하는 연구를 향후 진행할 예정이다.

참고문헌

[1] E. C. de Castro and R. R. Gudwin et. al., “An Episodic Memory for a Simulated Autonomous Robot,” *Proc of Robocontrol, 2010*.

[2] J. Winkler, M. Tenorth, and A. K. Bozcuoglu, et. al., “CRAMm－Memories for Robots Performing Everyday Manipulation Activities,” *Proc. of ACS-13, 2013.*

[3] F. Balint-Benczedi, Z. C. Marton, and M. Durner, et. al., “Storing and Retrieving Perceptual Episodic Memories for Long-Term Manipulation Tasks,” *Proc. of IJCAI-17*, 2017.

[4] M. Beetz, L. Mosenlechner, and M. Tenorth, et. al., “CRAM － A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments,” *Proc. of Intelligent Robots and Systems (IROS)*, 2010.

[5] M. Beetz, F. B´alint-Bencz´edi, and N. Blodow, et. al., “RoboSherlock: Unstructured Information Processing for Robot Perception,” *Proc. of Robotics and Automation (ICRA)*, 2015.