Manipulation Primitives: A Paradigm for Abstraction and Execution of Grasping and Manipulation Tasks

J. Felip[a,*], J. Laaksonen[b], A. Morales[a], V. Kyrki[b,c]

[a]*Robotic Intelligence Laboratory, Universitat Jaume I, 12006 Castellón, Spain*
[b]*Department of Information Technology, Lappeenranta University of Technology, P.O. Box 20, 53851 Lappeenranta, Finland*
[c]*Department of Automation and Systems Technology, Aalto University, P.O. Box 15500, 00076 Aalto, Finland*

Abstract

Sensor-based reactive and hybrid approaches have proven a promising line of study to address imperfect knowledge in grasping and manipulation. However the reactive approaches are usually tightly coupled to a particular embodiment making transfer of knowledge difficult.

This paper proposes a paradigm for modeling and execution of reactive manipulation actions, which makes knowledge transfer to different embodiments possible while retaining the reactive capabilities of the embodiments. The proposed approach extends the idea of control primitives coordinated by a state machine by introducing an embodiment independent layer of abstraction. Abstract manipulation primitives constitute a vocabulary of atomic, embodiment independent actions, which can be coordinated using state machines to describe complex actions. To obtain embodiment specific models, the abstract state machines are automatically translated to embodiment specific models, such that full capabilities of each platform can be utilized.

The strength of the manipulation primitives paradigm is demonstrated by developing a set of corresponding embodiment specific primitives for object transport, including a complex reactive grasping primitive. The robustness of the approach is experimentally studied in emptying of a box filled

[*]Corresponding author.
[**]Current address: Aalto University, Department of Automation and Systems Technology, P.O. Box 15500, 00076 Aalto, Finland.

Email addresses: `jfelip@uji.es` (J. Felip), `jonna.e.laaksonen@gmail.com` (J. Laaksonen), `Antonio.Morales@uji.es` (A. Morales), `ville.kyrki@aalto.fi` (V. Kyrki)

with several unknown objects. The embodiment independence is studied by performing a manipulation task on two different platforms using the same abstract description.

Keywords: sensor-based grasping and manipulation, abstraction

1. Introduction

Robots operating in unstructured service scenarios need to operate robustly despite incomplete and uncertain information about their environment. Seminal works on reactive control [1, 2] demonstrated that the use of several low-level perception/actuation loops enable robots to adapt to unknown scenarios. These approaches were soon extended by incorporating high-level planners prioritizing the available reactive behaviours, giving birth to hybrid deliberative/reactive control [3, 4].

In complete contrast to reactive approaches, manipulation and grasping has been traditionally addressed through planning of contact states. Methods, such as grasp quality metrics based on form and force closure, are very powerful when the uncertainty in robot and environment models is minimal. However, in service robotics manipulation scenarios, uncertainties appear in many quantities, for example, inaccurate knowledge about the poses of objects and obstacles, incomplete models of object shape and physical properties, or inaccurate kinematics in flexible robots. To address these issues, there are a few works using the reactive paradigm. One of the earliest works in reactive grasping proposed the use of a light beam sensor to align the gripper with an unknown object [5]. More recently, solutions such as IR proximity sensors [6], tactile sensors [7, 8], and force and tactile feedback [9] have been proposed. In contrast to traditional grasp planning, these approaches aim to adapt the robot hand to the shape of the target object reactively instead of placing contacts in planned locations. As a consequence, exact models of objects are not required, which is a great strength of these approaches. Reactive manipulation approaches are usually, however, specific to a particular embodiment, which makes it difficult to transfer plans between different embodiments and even from humans to robots.

This paper proposes a paradigm for modeling and execution of reactive manipulation which describes manipulation tasks in terms of atomic primitives coordinated by a state machine. The approach extends earlier works by introducing an embodiment independent layer of abstraction. The abstraction offers several advantages. Firstly, complex actions can be described in terms of simple abstract primitives. Secondly, plans can be shared over different embodiments because the vocabulary of primitives is shared. Thirdly, manipulation primitives offer high-level planners a vocabulary of reliable actions onto which build manipulation plans, thus simplifying and robustifying planning. Finally, these abstract models can be translated to embodiment specific models, constituting of reactive sensor-based controllers, such that the full capabilities of each platform can be utilised.

While the idea of coordination of control primitives using a state machine has been known for a long time, few works have considered what kind of primitives would be needed for typical manipulation tasks. To flesh out the abstract primitives, we present a complete set of reactive manipulation primitives for an arm manipulator equipped with a wrist force sensor and a three-fingered hand equipped with tactile sensors. In particular, we present a complex reactive grasping primitive for which the robustness against inaccuracies in the planning of action is studied. Moreover, we study the use of several primitives in solving a complex task in the scenario of emptying a box filled with several unknown objects blindly. Finally, we look into the power of the embodiment independent abstract primitives in the scenario where two different platforms with different hardware capabilities are used to complete a manipulation task using the same abstract description.

The experiments demonstrate the potentials of manipulation primitives in two main aspects. Firstly, according to our experiments, primitives are an effective way to address complex manipulation tasks in a robust manner. Secondly, a primitive based vocabulary is an effective way of transferring knowledge and plans between embodiments. In practical terms, the developed manipulation primitives are demonstrated to work robustly in the presence of unknown conditions in the execution of a complex task, and the transferring of a manipulation task to two different embodiments following the paradigm of manipulation primitives is shown to be successful.

1.1. Related work

The idea of control primitives is not new in robotics, and particularly in robot grasping. Earlier works propose individual control primitives for different problems such as to control a hand [10], to define object movements [11] and its relations [12] and to control a manipulator [13]. Despite different definitions of primitives, all of them present a common trend, discretizing and reducing the complexity of controlling a robotic setup by reducing the search space for planning. Other similar approaches include Object Action Complexes [14] and the physical interaction framework of [15]. However, in contrast to this work, all of the above consider primitives which are specific to a particular embodiment.

An apparently similar concept are Dynamic Movement Primitives (DMPs) introduced by Schaal et al. [16]. DMPs describe motion trajectories by means of differential equations, which can be adapted to several situations by adjusting a few of the equation parameters and are deeply interlinked with motor control. This framework has proved to be very effective to represent standardized arm movements, which can be learned by human observation or demonstration[17], and can even be adapted reactively as the scene is observed to change. However, they are basically different idea of primitives presented in this paper. DMPs are focused on motion description in a lower level, while our primitives describe robot manipulation skills.

An alternative approach to address the problem of unknown environment is to use sensors, for example vision, to build the necessary models. Vision has been used to obtain the shape of unknown target objects [18, 19] and to determine the location and pose of objects [20]. In both cases, visual input was used to plan feasible grasps. Visual feedback can also be used during reaching for an object. Murphy et al. [21] uses visual techniques to correct the orientation of a four-finger hand while approaching an object to improve contact locations. Once contact between object and robot has been reached, tactile and force sensors can be applied. Tactile measurements can be used to estimate the quality of grasps [22, 23, 24, 25] or the shape of an object [26] with the purpose of reaching better contact locations through a sequence of grasping/regrasping actions. Contact information can also be used to program complex dexterous manipulation operations like finger repositioning while holding the object [22, 27]. Several works have combined the use of several sensors to complete the process of grasp planning and execution [28, 29]. An early version of the grasping primitive presented in this paper was presented in [9]. To our knowledge, this paper is the first to consider primitives for the whole object transport cycle. Moreover, the grasp primitive is the first to propose a reactive controller able to grasp any type of rigid object, without taking any types of assumptions. It does it by combining three different reactive phases: hand alignment, collision with supporting surface, and grasp force adaptation. Please note that some solutions to some of this problems have been recently presented [30].

Finally, few studies have addressed the issue of abstracting hardware from action. Petersson and Christensen presented a somewhat similar framework in [31] but to our knowledge that framework has never been demonstrated in practice with multiple embodiments. Earlier version of the framework presented here appeared in [32]. Finally, Ellenberg et al. studied how algorithms for humanoid robot walking can be transferred between embodiments [33]. Recently, the RoboEarth project has proposed a web platform for sharing environment models as well as action "recipes" between multiple robots using Ontology Web Language (OWL) [34]. However, to our knowledge, the work presented here is the most advanced in demonstrating in practice the performing of abstract actions across multiple embodiments.

1.2. Paper outline

The paper is structured as follows. We now continue by defining the concepts and terminology used through the paper in Sec. 2. Then, Sec. 3 describes the abstraction first conceptually and then its implementation. Sec. 4 presents manipulation primitives, that is sensor-based controllers, on a specific platform, the UJI robot embodiment. Experiments in Sec. 5 study the different aspects of the proposed approach. In Sec. 6 we discuss about the limitations and pitfalls. Finally, we conclude and give remarks for future work in Sec. 7.

2. Manipulation primitives framework

We define a *manipulation primitive* as a single reactive controller designed to perform a specific primitive action on a particular embodiment. Each primitive is parameterized to allow it to be used in different situations. A focused control policy which uses the available sensor feedback is then used to achieve predefined success or failure conditions.

Primitives are parameterizable, thus, a task planner requires some information (i.e. parameters) to tune and instantiate actions to the specific situation. The description of such a planner is out of the scope of this paper but its outcome must be a composition of primitive instances to address the current scenario. Finally, primitives can finish with several degrees of accomplishment, which in its more simple expression would be a value from the pair *Success/Failure.*

Primitives are the elementary symbols of a vocabulary that is used to describe *actions* and *tasks.* A *task* is a semantically meaningful goal, such as emptying a grocery bag, consisting of one or more *actions.* Each action

Abstract primitive	*Parameters*	*Meaning*
move	trajectory, move type	Move without object.
transport	trajectory, move type	Move with object.
place	trajectory, move type	Place down object.
push	trajectory, move type	Push object.
grasp	*preshape*	Grasp object.
release		Release object.

Table 1: Abstract primitives and parameters (optional parameters in *italic)*.

describes a single manipulation action, for example, moving an object from one location to another, as a Finite State Machine (FSM) where the states correspond to manipulation primitives. The transitions between states are triggered by predefined *events*, which correspond to perceptual or internal conditions. It must be noted that besides FSMs, other approaches exist for combining primitives in sequences or in parallel.

Primitives are by definition embodiment specific. However, embodiments with similar capabilities allow the definition of primitives with similar behavior and purpose, which can be thought as *abstract manipulation primitives*. The focused purpose of primitives simplifies the development of equivalent primitives on several embodiments. This equivalence also enables the transmission and execution of plans between different embodiments. The abstract manipulation primitives can then be used to describe *abstract actions*. We call this abstract representation of an action the Abstract State Machine (ASM), which is a FSM composed of abstract primitives as their states. The embodiment specific FSM can be automatically constructed from the ASM using a mechanism we term translation. The next section describes the abstraction and the translation mechanism in more detail.

3. Embodiment independence through abstraction

The set of abstract primitives proposed in this work is shown in Table 1. This set of abstract actions allows moving objects by grasping or pushing them and has been found to support many common manipulation actions. The primitives have required and optional parameters. When the primitives are instantiated, the required parameters which describe constraints need to be fulfilled but the optional parameters can be ignored if necessary as their purpose is to serve as hints how to perform the task.

All primitives except *grasp* and *release* are related to arm motions. The required parameters for these primitives define the target of the motion and

Abstract event	*Meaning*
success	Primitive successfully completed.
grasp_stable	Stable grasp detected.
grasp_lost	Grasp loss detected.
timeout	Timeout for specified time.
hardware_failure	Hardware failure detected.

Table 2: Abstract events.

the type of the motion. The motion target can be a single waypoint or a trajectory represented as a set of waypoints, but defining a strict trajectory to be followed should be avoided when not made necessary by the task to allow each embodiment to use its own capabilities in the best possible way. Defining only the end waypoint is usually sufficient from the task perspective and leaves the freedom for the embodiment to choose a collision free path for that particular embodiment. In addition to the motion target, the type of motion is specified. Supported motion types include free, guarded, and constrained motions. In free motion, the embodiment is free to use any path to reach the target. In a guarded motion, the embodiment is required to use a Cartesian straight line path. In a constrained motion, rotational degrees of freedom can be constrained to remain the same for the duration of the motion. This is useful, for example, to transport containers with liquid. The underlying idea in the required parameters is thus to constrain the effects of a primitive rather than the ways to achieve them.

The grasp primitive allows to use an optional parameter to choose the grasp preshape. Currently supported preshapes in the implementation include parallel and cylindrical grasps. Because the parameter is optional, it can be ignored by a platform which does not support a particular grasp type. In that case, the primitive is likely to be translated to the closest possible grasp available.

To allow the embodiment independent description of series of primitives, the state transitions need also to be described in an abstract fashion. This is done using abstract events shown in Table 2. The events are related to completing a primitive successfully (*success*), grasp stability (*grasp_stable, grasp_lost*), and failure conditions (*timeout, hardware_failure*). Each platform is again free to use the available sensor set in any possible way to detect these events.

It should be noted that the primitives and events at the abstract level are not coupled to any particular embodiment. An important note here is that the sets of abstract primitives and events need to be rich enough in order to allow wide use of sensors in the embodiment specific controllers, while at the same time it is important to keep the semantic meanings of the abstract entities clear to allow the mapping between abstract and platform specific sensor events and controllers.

3.1. Abstract State Machine

The abstract state machine is a hardware independent description of a manipulation action. XML (eXtensible Markup Language) is used to describe the relevant information, such as the states related to the abstract primitives and the transitions related to the abstract events. In addition to states and transitions, information about the environment such as obstacles and the location, mass and approach direction to the target object are included in the abstract state machine description. All the properties and definitions in XML are hardware independent.

The abstract state machine is described through definition of states and transitions between the states. Both states and transitions have properties that can be used to further inform of the intended action. The most important state property is *type*, corresponding to one of the primitives introduced above, *"success"*, or *"failure"*. The two latter types indicate end states of an action (terminating states of the state machine) with either success or failure reported to the higher level controller. In addition, the parameters of the primitives are specified as state properties. For example, the hand preshape for grasping or the target position of the end-effector can be set through state properties. The transition properties describe the set of abstract events which trigger the transition. For example, the loss of a grasp can trigger a transition to another state.

The attributes are the key factor in selecting the primitive controllers during the translation process depicted in 3.2. An example abstract state machine and its XML definition, describing a simple grasp and lift manipulation, is shown in Figs. 1 and 2. Some of the elements have been left out for brevity, e.g. properties of the object and some of the common transitions, e.g. timeout to the failure state. It should be noted that the state machine does not need to be a sequence or a tree but it can be any directed graph, however, the simple form in the example is used to limit the size of the associated XML code shown.

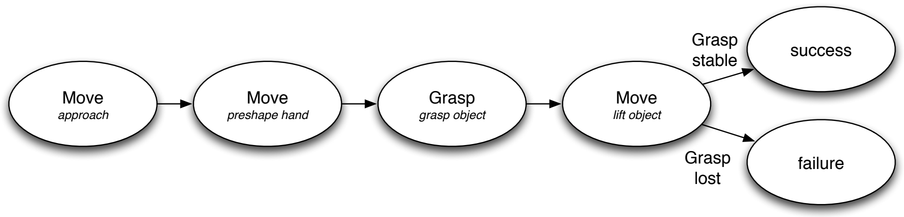
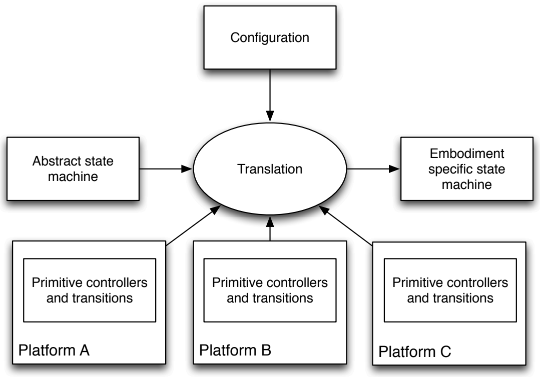


Figure 1: An abstract state machine.

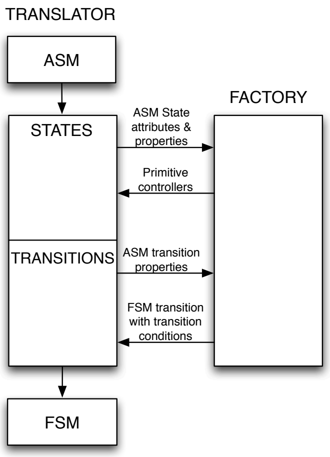
```xml
<statemachine>
  <state name="approach" type="move">
    <movement>free</movement>
    <hand_shape>open</hand_shape>
   </state>
  <state name="preshape_hand" type="move">
    <movement>guarded</movement>
    <hand_shape>pinch_grasp_preshape</
         hand_shape>
  </state>
  <state name="grasp_object" type="grasp">
    <movement>guarded</movement>
    <hand_shape>pinch_grasp</hand_shape>
  </state>
  <state name="lift_object" type="transport
       ">
    <movement>guarded</movement>
    <hand_shape>pinch_grasp</hand_shape>
    <path>
      <position>0.2 0.6 0.25</position>
    </path>
  </state>
  <state name="success_end" type="success">
  </state>
  <state name="fail_end" type="failure">
  </state>
```

```xml
  <transition origin="approach"
   destination="preshape_hand">
    <success/>
  </transition>
  <transition origin="preshape_hand"
   destination="grasp_object">
    <success/>
  </transition>
  <transition origin="grasp_object"
   destination="lift_object">
    <success/>
    <grasp_stable/>
  </transition>
  <transition origin="lift_object"
   destination="fail_end">
  <grasp_lost/>
  </transition>
  <transition origin="lift_object"
   destination="success_end">
    <success/>
    <grasp_stable/>
  </transition>
</statemachine>
```

Figure 2: An abstract state machine in XML.


Configuration
Abstract state machine
Translation
Embodiment specific state machine
Primitive controllers and transitions
Platform A
Primitive controllers and transitions
Platform B
Primitive controllers and transitions
Platform C

(a)


TRANSLATOR
ASM
STATES
TRANSITIONS
FSM
FACTORY
ASM State attributes & properties
Primitive controllers
ASM transition properties
FSM transition with transition conditions

(b)

Figure 3: (a) Translation process. (b) Relationship and communication of the translator and factory.

3.2. Translation from ASM to FSM

The translation process connects the abstract state machine and the embodiment specific state machine (FSM). The translation takes the abstract state machine as an input, and translates the abstract state machine into an embodiment specific state machine. The high-level translation process is depicted in Fig. 3(a).

As can be seen in Fig. 3(a), the translation component needs input defining the configuration of the translation process, i.e., the target platform and the platform specific transitions and primitive controllers used directly in the embodiment specific state machine. The benefit of this arrangement is that the only hardware dependent blocks shown in the figure are the primitive controllers and transitions that are platform specific. The critical requirement of real-time operation for sensor-based control is also fulfilled as the embodiment specific state machine can be run as is, without any additional overhead from maintaining hardware independence.

The translation process requires a mapping component which produces the embodiment specific state machine from the abstract automaton. In our case the mapping component is constructed from two sub-components, shown in Fig. 3(b). The first part, *translator*, consists of necessary book-keeping and the internal logic that is independent from the embodiment. The translator also constructs the final finite state machine which is used to execute the desired abstract action. The second part, *factory*, handles the embodiment specific construction of the states and the transitions of the finite state machine, i.e., the sensor based primitive controllers and transition conditions. This division was made to reduce the implementation time of the factory, which needs to be implemented for each different embodiment. The relation and the communication between these two sub-components are shown in Fig. 3(b). The factory also receives object and environment information of the ASM in addition to a particular state or event and its properties. This gives the factory the complete information needed for the primitive controllers and transition conditions. The translation process proceeds as shown in the figure. First, each of the abstract states is mapped independently by the factory to a suitable embodiment specific primitive controller. Then, the abstract events (transitions) are processed in a similar fashion.

The embodiment specific factory uses abstract primitive parameters and environment information to choose a suitable embodiment specific controller and its parameters. Typically, each type of abstract primitive is mapped to a certain corresponding embodiment specific primitive, although it is possible that this relation is not one-to-one or even static. For example, it is possible to map different abstract arm movement primitives to a single embodiment specific primitive if that primitive can be parametrized in a suitable fashion, as we show in Sec. 4. In addition to choosing the type of the controller, the factory can deliver embodiment specific parameters to the controller. These can be used, for example, to communicate a collision free path for that particular embodiment. Thus, in this case, the factory will also act as an embodiment specific path planner. A similar process is in place for the transition events, that is, the factory produces computation nodes for sensor processing which use the available sensors of each embodiment to detect the events.

For free motions in a collision free space and guarded motions, common primitive controllers can be used over several embodiments. This is possible by having common control and sensor interfaces for the arm, which in our case perform either Cartesian or joint space velocity control. Thus, we can use primitive controllers that use the arm velocity control for all hardware platforms without modifications just by setting appropriate parameters through the embodiment specific factory. The same applies to the transition conditions, for example a timeout transition condition can be used across all platforms as the condition relies on measurement of time which should be available in every platform.

The rules that are observed in the translation process are simple:

- Each state in the ASM must correspond to one state in the FSM.
- Each transition in the ASM must correspond to one transition in the FSM.
- Each transition condition in the ASM can be represented by one or more transition conditions in the FSM.

These rules ensure that the execution of the FSM can be traced back to the original abstract state machine. This allows the system to report back failures to higher level so that the higher level system operating on the abstract state machine is able to reason using the same concepts. The possibility to represent an abstract transition condition by more than one embodiment specific ones allows, for example, to check the success of multiple primitive controllers, such as separate arm and hand controllers, with a single success transition condition in the ASM.

While the translator component is universal across all embodiments, the factory component needs to be built specifically for each platform. The complexity of the factory affects the flexibility of the final system. A simple factory with fixed mappings between abstract and embodiment specific primitives and events is sufficient for many relatively simple tasks. Complex factories considering for example path planning for redundant manipulators or the choice of a grasping primitive among several are possible and discussed more in Sec. 6. If the factory is unable to find a suitable mapping for any reason, the mapping fails which is reported back to the task level. However, it should be noted that the factory is often fairly simple to implement because there are only a limited number of abstract primitives, event types and parameters, and the factory needs to consider only one primitive or event of an abstract action at a time.

4. Sensor-based primitives for manipulation

In order to allow primitives to transfer plans between embodiments, all the primitives appearing in Table 1 must be instantiated on each specific embodiment. This section describes such an instantiation for a platform consisting of an arm manipulator with a wrist force/torque sensor and a

Abstract	*UJI*	*Other parameters*	*Control and sensor requirements*
Grasp	Robust grasp	Pregrasp size, grasp preshape	Arm control, FT and tactile sensors
Move	Transport	-	Arm control
Push	Transport	-	Arm control
Transport	Transport	Obstacles, trajectory, constraints	Arm control
Place	Place	Contact threshold	Arm control, Force-torque sensor
Slide	Slide	Slide force threshold	Arm control, Force-torque sensor
Release	Release	Hand position	Arm control

Table 3: Mapping of abstract primitives to UJI's implementation, parameters and requirements.

three-fingered hand with tactile sensors. A more detailed description of the platform can be found in Sec.5.

The result of the instantiation of the abstract primitives defined in Table 1 is a set of sensor-based primitives which can be found in Table 3. In this table it can be noted that three abstract primitives *Move*, *Push* and *Transport* have been instantiated through a single embodiment specific primitive, *Transport*. The behaviour of the latter primitive can be modulated through the parameters to produce the behaviour expected for each abstract primitives. More details of this primitive are given in Sec. 4.2.

All of the embodiment specific primitives accept the parameters specified in Table 1. In addition, some of implemented primitives accept several optional parameters which have not been specified in the abstract description. The purpose of these parameters is to provide more complete and flexible primitives. In any case the translation process will not use them.

The abstract events shown in Table 2 are mapped one to one, there are no parameters that can be tuned to change their behavior or meaning. The platform dependent implementation, exploits where possible, the available sensors to detect the events. On the other hand, as explained in Sec. 3.2, there are events that can be platform independent and can use the same implementation for any platform, e.g. timeout.

The rest of the section is devoted to describing the sensor-based primitives in detail.

4.1. Grasp primitive

The simplest implementation of grasp primitive would consist of closing the robot hand. It has, however, been demonstrated that by using sensor based methods the success rate of this primitive can be increased significantly

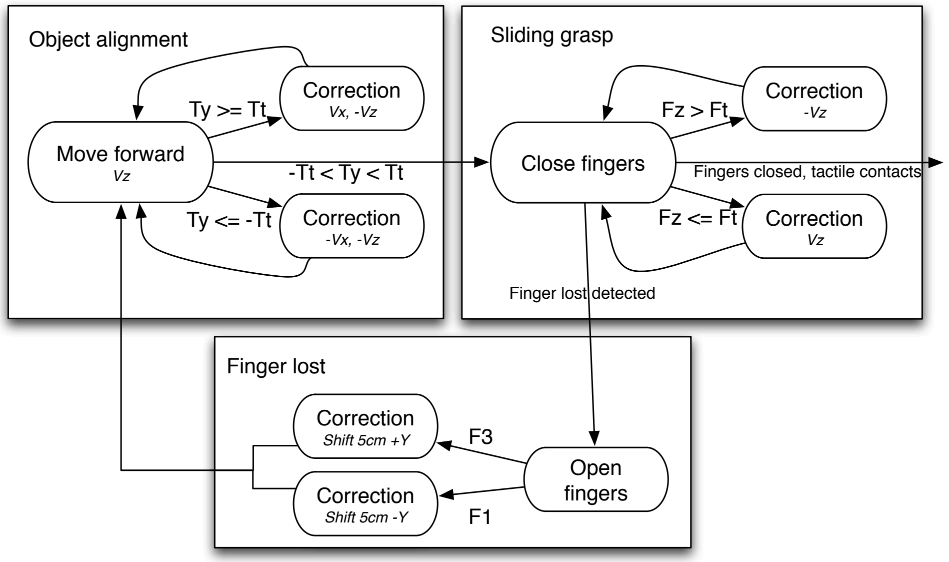


Figure 4: Robust grasp primitive. Fz is the force in the Z hand axis, Ty is the torque on the Y hand axis, Tt is the Y torque threshold, Ft is the Z Force threshold, Vz is a velocity in the Z hand frame. Vx is a velocity in the X hand frame

[9]. We propose a new sensor based controller that performs several corrective movements in order to get a stable grasp.

The optional parameters for the implemented grasp primitive are the pregrasp type (cylindrical, spherical, hook) and size. The instantiation of the abstract grasp primitive would directly convert the preshape parameter values into different combination of the pregrasp type and size.

The initial assumption of the grasp primitive is that the hand is facing the object, at a close distance. If the approaching direction of the hand would have been carefully planned and the positioning in the vicinity of the object executed accurately, the hand should only move forward and close to obtain a stable grasp of the object. This is not often the case, so a series of corrective movements are performed in order to obtain a robust grasp, see Fig. 4. These movements are divided into three phases: alignment, sliding grasp and force adaptation.

The corrections are performed depending on the estimation of the location of the detected contact, the implementation of this primitive is preshape independent.

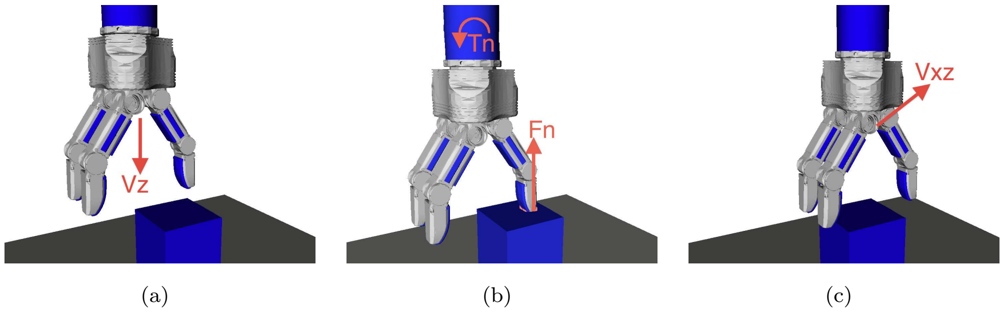


Figure 5: Grasp primitive: Alignment phase. (a)Arm moving towards the object. (b)Contact generates torque in the wrist. (c)Correction movement is performed.

4.1.1. Alignment

In some situations, the initial approach vector is not pointing to the center of the object, and thus there is a premature collision during the approaching. This contact can be detected using a force-torque sensor mounted on the wrist. Using the torque, the contact point is estimated and a correction is performed to center the object.

$$\vec{v} = \begin{cases} \vec{v_x} - \vec{v_z} & \text{if } T_y > T_{treshold} \\ -\vec{v_x} - \vec{v_z} & \text{if } T_y < -T_{treshold} \\ \vec{v_z} & \text{otherwise} \end{cases} \quad (1)$$

Eq. 1 shows the controller that performs the alignment phase of the robust grasp primitive. Where $\vec{v}$ is the resultant velocity that is applied by the controller to the hand (w.r.t hand frame) and is a combination of $\vec{v_x} = (V_x, 0, 0)$ and $\vec{v_z} = (0, 0, V_z)$. Where V_x and V_z are the parameters that control the speed of the corrections produced by the controller movements. T_y is the torque around y-axis from the force sensor. The alignment is finished when a contact is detected and $-T_{treshold} \leq T_y \leq T_{treshold}$

An example of this is depicted in Fig. 5. The contact can also be detected using tactile sensors. Alignment correction improves grasping of objects with location uncertainty by allowing the hand to align its center with the object.

4.1.2. Sliding grasp

When approaching, the hand makes contact in occasions with the supporting surface instead of the object (See Fig. 6(a)). In this case, closing the hand can result in unsuccessful grasps especially for small objects. To counter this problem, a sliding correction is used. The corrective movement consists of moving the hand forwards or backwards depending on the force sensed along its Z axis while the fingers are closing (see Fig. 6) to maintain stable, light contact with the supporting plane. When the fingers are no longer able to close, because the object is grasped or the fingers reach their joint limits, the sliding grasp control ends. The correction allows grasping small objects by sliding the fingers on the supporting plane until the object is securely grasped. Equation 2 describes the arm cartesian velocity control used meanwhile the fingers are closing where $\vec{v}$ is the velocity control sent to the arm, $\vec{v_z}$ is a velocity in the Z axis of the hand. Fz_{sensor} is the current force in Z axis of the hand read by the sensor and $Fz_{threshold}$ is the force threshold.

$$\vec{v} = \begin{cases} -\vec{v_z} & \text{if } Fz_{sensor} <= -Fz_{threshold} \\ 0 & \text{if } -Fz_{threshold} < Fz_{sensor} < Fz_{threshold} \\ \vec{v_z} & \text{if } Fz_{sensor} >= Fz_{threshold} \end{cases} \qquad (2)$$

The behavior of this correction phase is shown in Fig. 6. The hand starts closing and when the fingers make contact with the surface, the force they are applying is detected in the wrist, thus the arm moves back (Fig. 6(a)). The fingers continue closing and because no contact force is detected, the arm moves forward (Fig. 6(b)). In Fig. 6(c) the fingers are not able to close anymore and the sliding grasp ends. In a recent work a similar strategy is used by Kazemi et. al.[30]

It may happen, if the error is big enough, that the hand closes and one of the external fingers loose contact with the object. If this happens, the *finger lost* corrective movement is triggered, the hand opens and shifts towards the detected contacts in order to place all the fingers on the object.

4.1.3. Force adaptation

The force of the fingers is increased to improve grasp stability. The primitive ends with a success if at the end the object is still in the hand, detected with joint angles or contact information.

4.2. Transport primitive

The purpose of the transport primitive is to move the arm to a specified target position while the hand holds an object. The primitive can also be used to move the arm without an object.

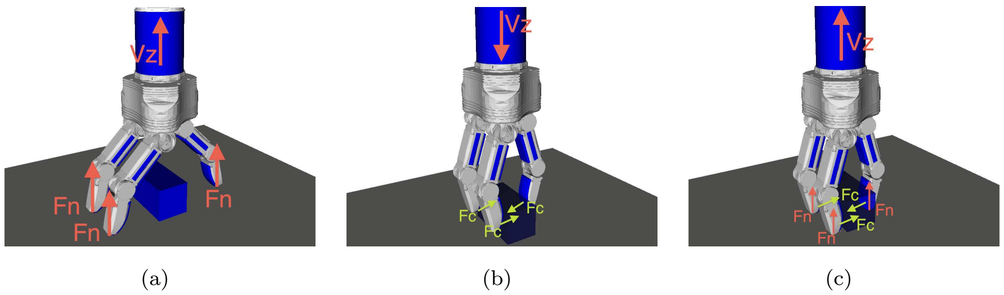


Figure 6: Grasp primitive: Sliding grasp phase. (a)The fingers contact the table while closing. Thus the controller sets the velocity to move the hand back. (b)The fingers are closing and the contact with the table is lost. Vz is set forwards. (c)The hand contacts the table again but the object is already grasped.

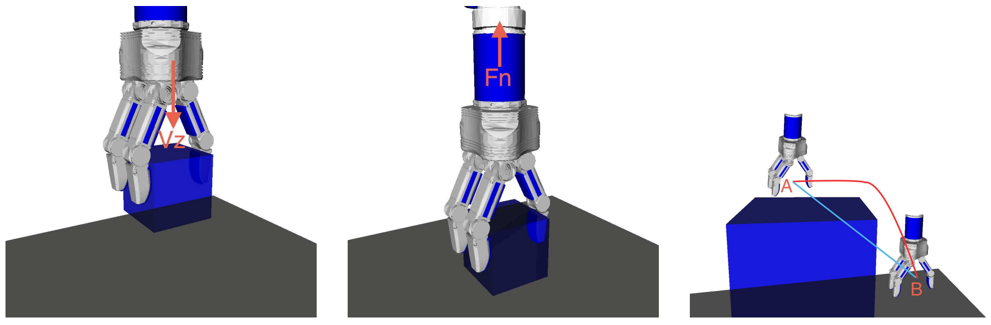
As discussed in Sec. 3.2 this primitive is used in *Tombatossals* to map the *move*, *transport* and *push* abstract states to the embodiment dependent states.

The trajectory to move the arm from the starting point to the target can be constrained by specifying optional parameters. A trajectory can be specified as a list of joint positions that define the state of each joint during all the transport primitive execution. A less restrictive constraint is to specify the end-effector Cartesian trajectory. Instead of defining the exact trajectory that the robot must follow, it is also possible to specify position, velocity or acceleration limits. Equation 3 shows the use of a force-torque threshold parameter that is used to stop the movement if a collision is detected where x_{target} and $x_{current}$ are the target and current 6D pose vectors.

$$\vec{v} = \begin{cases} x_{target} - x_{current} & \text{if } ||F_{sensor}|| < F_{threshold} \\ 0 & \text{if } ||F_{sensor}|| >= F_{threshold} \end{cases} \tag{3}$$

Optional parameters can also be used to describe environment obstacles as an obstacle point cloud, in which case a force-field [35] based collision avoidance strategy is used to generate a collision free trajectory from current to target position maintaining the hand orientation.

For instance, if the task is to transport a mug full of water without pouring the liquid, acceleration should be constrained to a low value on all axes and the rotation velocity of the table plane axes should be set to 0 to prevent tilting the mug. If the target position cannot be reached without breaking the specified constraints, the primitive ends with a failure. In Fig. 7(c) an



(a) (b) (c)

Figure 7: Place (a-b) and transport (c) primitives. (a)Arm moving the object towards the surface. (b)Contact is detected by force/torque sensor. (c)Example of execution of the constrained transport primitive from the starting point a to the target point b. Red line: Standard trajectory. Blue line: Position constrained trajectory.

example of a position constrained trajectory is shown. The convex hull of the box is defined as forbidden space to define position constraints.

4.3. Place primitive

The place primitive is used to place an object on a supporting plane while detecting the support on-line using sensor feedback. The arm moves downwards until a contact is detected with a force sensor. Equation 4 describes the control action taken depending on the force sensor readings where $\vec{x_z}$ represents a constant predefined velocity on the Z axis of the world. The primitive ends when the arm stops. This primitive can be configured with an optional parameter $F_{threshold}$ defining the force threshold needed to detect a contact. An example execution of this primitive is shown in Fig. 7.

$$\vec{v} = \begin{cases} -\vec{x}_z & \text{if } ||F_{sensor}|| < F_{threshold} \\ 0 & \text{if } ||F_{sensor}|| >= F_{threshold} \end{cases} \tag{4}$$

4.4. Release primitive

Releasing an object can be difficult because the fingers can, while opening, collide with the supporting plane or other parts of the object (see Fig. 8(a)). To handle this problem, the release primitive opens the hand slowly while the arm moves back. The movement of the arm is force-controlled and the arm only moves back if there is a contact detected between the opening

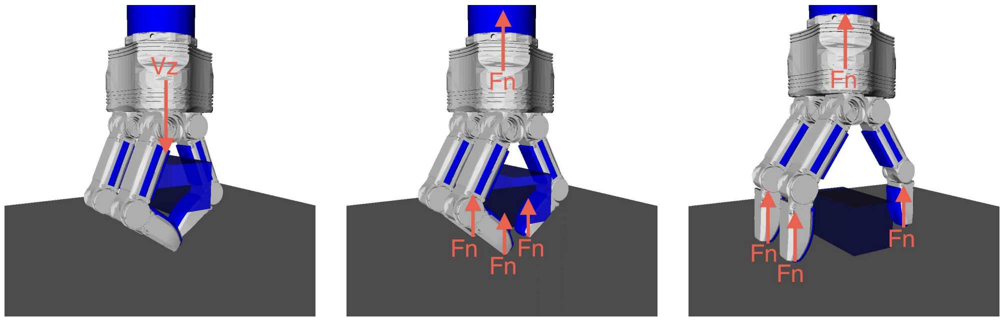


Figure 8: Release primitive. (a)Hand before opening the fingers. (b)The hand cannot release the object, the fingers are blocked by the surface. The normal force F_n in each finger propagates to the wrist. (c)The hand moves back and continues opening the fingers. The object is released successfully.

fingers and the surface (see Equation 5). The sequence of movements that this primitive performs is shown in Fig. 8. This primitive can be configured by setting the target hand position after release and the $F_{threshold}$ parameter.

$$\vec{v} = \begin{cases} -\vec{v}_z & \text{if } Fz_{sensor} < -F_{threshold} \\ 0 & otherwise \end{cases} \tag{5}$$

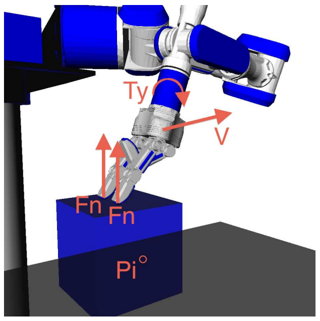
4.5. Slide primitive

The purpose of the slide primitive is to push an object from the top and slide it on a surface to a target position.

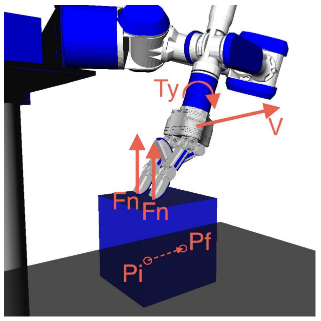
Equation 6 shows the control law that keeps the force applied to the object in a desired range while it moves the arm towards the target position. Only the target position is a required parameter, but the applied force can be configured by setting a desired force range defined by F_{min} and F_{max}.

$$\vec{v} = \begin{cases} x_{target} - x_{current} & \text{if } F_{min} < ||F_{sensor}|| < F_{max} \\ -\vec{x}_z & \text{if } ||F_{sensor}|| <= F_{min} \\ \vec{x_z} & \text{if } ||F_{sensor}|| >= F_{max} \\ 0 & otherwise \end{cases} \tag{6}$$

Fig. 9 shows more graphycally the behavior of this primitive: Using force control the arm applies a desired force (Fn) to the object, then moves towards



(a)



(b)

Figure 9: Slide primitive. (a)From the starting position with a hook preshape, the arm moves down until it touches the object, then it starts moving towards the target. (b)The object slides over the table from P_i to P_f. The primitive keeps the applied force stable.

a set target, keeping the applied force constant (Fig. 9(a)). The contact fixes the arm and object movement allowing the robot to slide the object on the surface from the starting to the target position (Fig. 9(b)).

5. Experimental results

Three experiments have been implemented to validate and illustrate the usefulness of the manipulation primitives paradigm: validation of the robust grasp primitive, completion of a manipulation task using a set of the primitives described, and the mapping of the same abstract state machine for different embodiments. The two first experiments are focused in illustrating the design of a manipulation primitive and the resolution of a complex task under this paradigm. The last case is focused on showing that action plans can be easily transferred between different platforms following the principles of the manipulation primitives paradigm.

All the experiments have been tested on real robot systems. In all of them the main experimental platform is *Tombatossals*, an anthropomorphic torso with 29 DOF shown in Fig. 13. The platform is composed of two 7 DOF Mitsubishi PA10 arms. The right arm has a 4 DOF Barrett Hand and the left arm a 7DOF Schunk SDH2. Both hands are endowed with Weiss Robotics tactile sensors on the fingertips. Each arm also has a JR3 force-torque sensor mounted on the wrist. The visual system is composed of a TO40 4 DOF pan-tilt-verge head unit with two Imaging Source DFK 31BF03-Z2 cameras and a Microsoft Kinect. For the third experiment a second robot platform consisting of Melfa RV-3SB 6-DOF arm with a Weiss Robotics WRT-102 parallel jaw gripper equipped with tactile sensors has been used.

In the attached video[1], we show all the experiments. First the highlights of the corrections performed by the robust grasp primitive, secondly the uncut execution of the empty the box task and finally the execution of an abstract state machine on two different platforms.

5.1. Validation of robust grasp primitive

In order to compare the robust grasp primitive with a non adaptive grasp controller, we have designed a naive grasping strategy that does not react to tactile and force data. The naive grasping primitive behaves as follows: The arm moves forward 10cm or until a contact is detected. Then the hand closes stopping each finger that detects contact. Finally force is slightly increased on the distal phalanxes to establish the final grasp.

In order to grasp the objects, the robot needs to detect their pose and plan a trajectory. For this experiment we have implemented a simple approach that is well known, fast, stable but not very precise. Using this object pose estimation will allow the controllers to show its performance under some uncertainty.

To determine the object position, Kinect RGBD images in combination with algorithms from the Point Cloud library have been used: The table plane is segmented out and the remaining points are clustered, the target object position is determined by the centroid of the leftmost cluster. The approach vector orientation and its roll are fixed to a top grasp, the approach vector position is shifted 20cm away from the highest point of the object.

The testbench for both controllers consisted on grasping 10 different household objects (see Figure 10) 20 times. 10 using the approach vector given by the visual system and 10 introducing a random uniform error to the approach vector generated by the visual system. Thus, each controller has tried 200 grasps.

The error that we have introduced to the approach vectors is uniformly distributed, 5cm in each axis and 15 degrees around each axis. Figure 11 shows a set of 10 approach vectors for an object after adding the uniform error.

[1]A high resolution version of the attached video can be found here: `http://jemma.hut.fi/irgroup/felip_laaksonen_kyrki_morales.mp4`

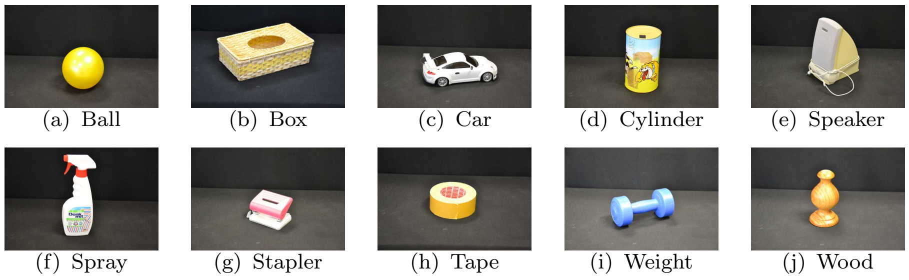

(a) Ball (b) Box (c) Car (d) Cylinder (e) Speaker (f) Spray (g) Stapler (h) Tape (i) Weight (j) Wood

Figure 10: Set of 10 objects used for testing the robust grasp primitive.

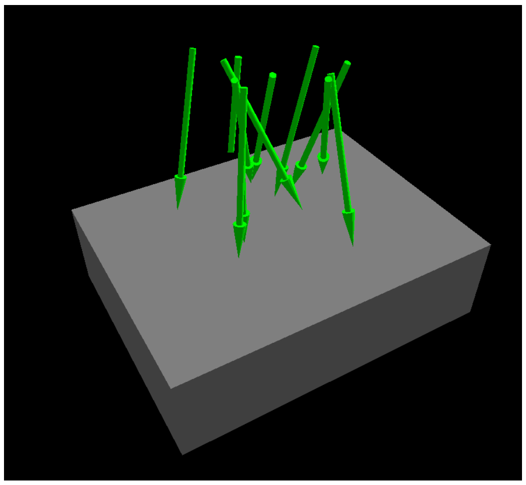


Figure 11: Set of 10 approach vectors after adding the uniform error.

	stable grasps	failures	%success
naive	88	12	88%
robust	99	1	99%
naive + error	19	81	19%
robust + error	59	12	59%

Table 4: Grasping experiments overall results after 100 attempts.

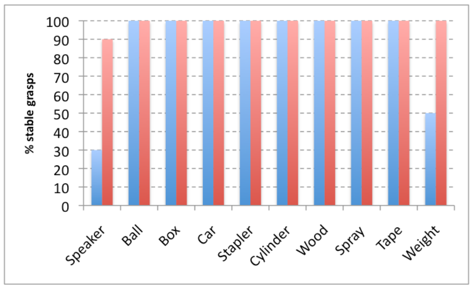
Table 4 shows the overall performance of each controller with and without the additional error. The performance of the robust controller has shown to be better especially under error conditions. Although the results without additional error are quite good for both controllers, the robust grasp primitive outperforms the naive one by more than 10% of success rate.

Figure 12 shows the performance of each controller, with and without error, for each object of the test set. Under controlled conditions both controllers are able to perform 10 successful grasps out of 10 attempts on eight of the objects. On the other hand, for some objects, e.g. speaker and weight, the performance of the naive controller is dramatically reduced (5/20) while the robust controller keeps its good performance almost intact (19/20). The main reason of that low performance is that those objects have some properties, assymetry and thinness, that make them difficult to grasp. On the other hand the robust grasp primitive is able to adapt to those properties that cause the naive controller to fail.

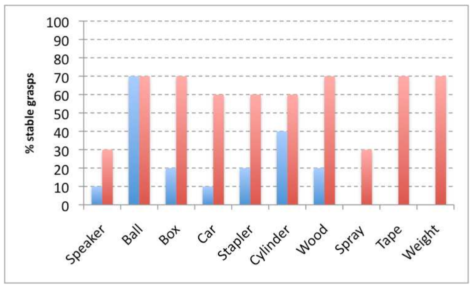
Under uncertainty conditions, the robust primitive is able to perform better than the naive one. As shown in Table 4 under error conditions the performance of the naive controller drops to 19% while the robust primitive holds a 59% of robust grasps achieved.

5.2. Emptying a box: Execution of a complex task

To demonstrate that the paradigm is valid for executing complex sensor-based tasks, we chose the task of emptying a box with no previous information about the number, location and pose of the objects inside. More precisely, the assumptions are that the object positions inside the box are not restricted, objects can be in any position and orientation inside the box, except that there is some clearance between the objects and the sides of the box. The object size is defined by the robot hand dimensions so that the objects fit inside the hand and are thus graspable. The box is set on an even plane inside the arm workspace. *Tombatossals* is used as the experimental platform.

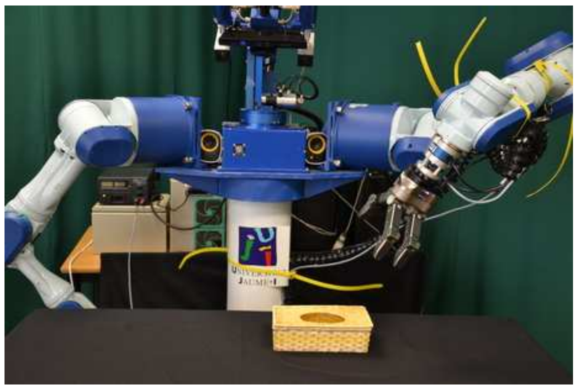


(a) Grasping performance without error.



(b) Grasping performance with error.

Figure 12: Grasping performance after 10 trials per object. Blue: Naive controller, Red: Robust controller



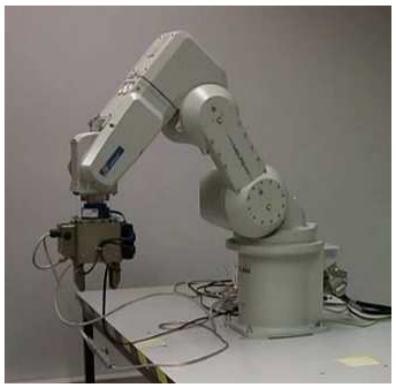


Figure 13: The experimental robotic platforms, Left: Tombatossals, the UJI humanoid torso. Right: MELFA robotic setup.

The task is solved using a *pick and place* loop executed for each object. This loop consists of a sequence of primitives structured and executed as a Finite State Machine (FSM) as described in Sec. 3.2. The FSM shown in Fig. 14 includes several of the sensor based manipulation primitives described in Sec. 2 which are instantiated to direct the robot to pick up an object from a starting position and place it to a destination position. The required parameters are the starting approach vector to a target object and the target position to place it. This procedure is repeated until the box is empty.

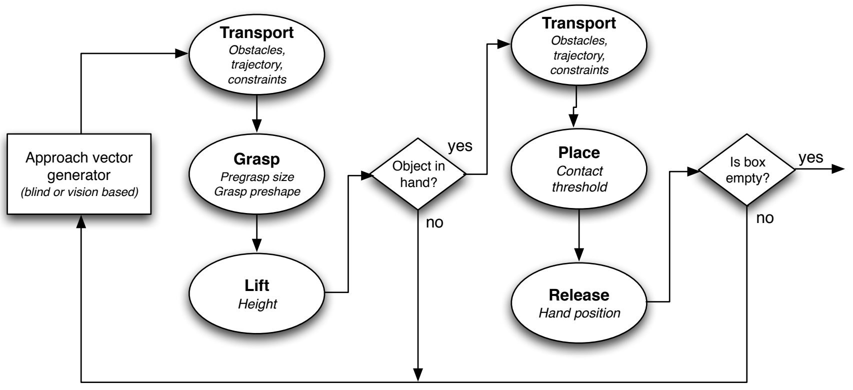
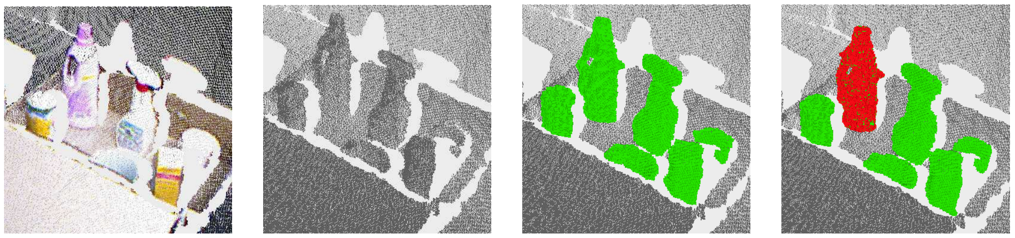


Figure 14: State machine for a pick and place task. Primitives are represented by circles. External processes are depicted using boxes. Diamond boxes represent conditions that are checked inside the parent primitive to determine the next transition. Inside each primitive, some examples of parameters are written in italics.

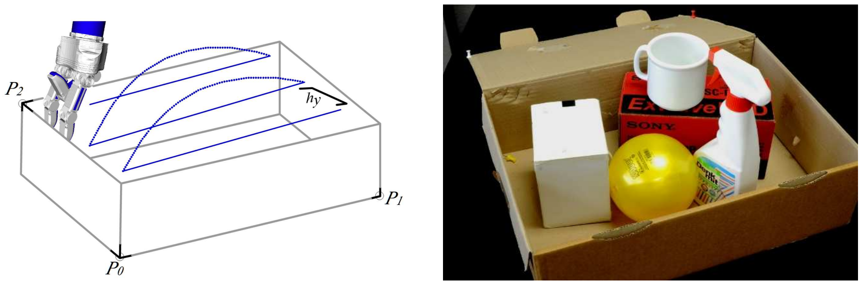
A key part of this loop is the generation of initial approach vectors. Three strategies were implemented: *random blind*, *blind exploration* and a *vision-based* method. In the first one, top-grasp approach vectors are generated uniformly at random inside the known location of the box. In this case, ending the whole process is decided by a human observer.

In the *blind exploration* strategy, the arm moves down until a contact is detected. If the contact is an object, the approach vector is generated over that contact location. If the contact is the box bottom the hand starts moving along the bottom until it detects a contact using the tactile and the force-torque sensor. As the position of the box is known, proprioception is used to determine whether the contact is with an object or with the box bottom. The exploration trajectory followed by the hand is shown in Fig. 16(a). The task ends after exploring the whole box without finding an object.



(a) Original 3D image. (b) Original 3D point cloud read from Kinect sensor. (c) Virtual box background filtering. Background points are colored in gray and objects are in green. (d) Object clustering and selection. Background points are marked in gray, objects in green, and the selected cluster is labeled in red

Figure 15: 3D point cloud segmentation phases.



(a) Hand preshape for exploration and exploration trajectory. (b) A possible object layout

Figure 16: Exploration trajectory and object layout.

In the *vision-based* strategy, the Kinect sensor is used in the same fashion as in Sec.5.1. Objects are segmented from the environment using a pass-trough filter using the known box boundaries and clustered as shown in Fig. 15. The approach vector is determined to approach the centroid of a randomly chosen cluster from the top. The task ends when there are no clusters left.

In order to validate the approach we carried out a total of 30 experiments of emptying a box filled with five unknown objects (see Fig. 16(b)). 10 experiments were performed for each approach vector generation method. All the methods were able to empty the box successfully 10 times out of 10. However, several attempts were sometimes needed to grasp an object. The number of attempts needed to lift an object was recorded. Fig. 17 shows the

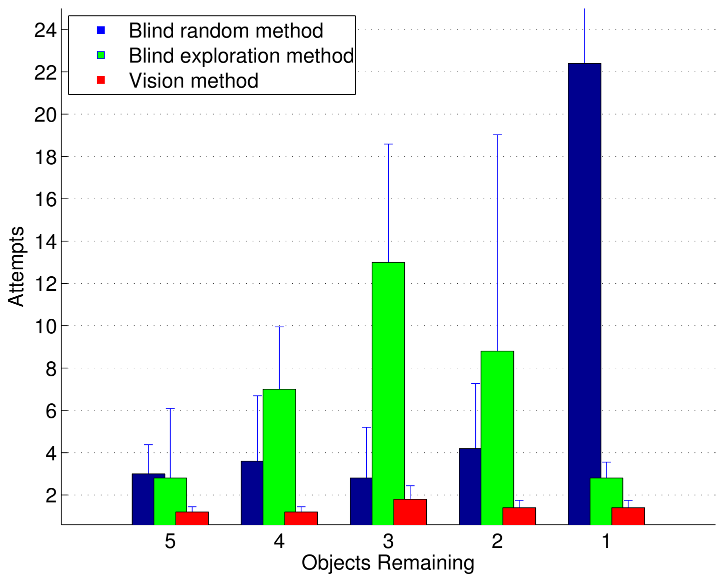


Figure 17: Average and standard deviation of required attempts depending on the number of objects remaining. The standard deviation for the blind random method when there is only one object left is truncated in the picture, its value is 39.32

average number of required attempts depending on the number of objects remaining in the box as well as the standard deviation.

It is evident from the figure that the vision-based approach vector generation improves the results over the blind methods, which is hardly surprising. However, the interesting result is that the blind methods were also able to complete the task successfully every time. It is important to note that the reactive grasping primitive plays a crucial role because the generated approach vectors are quite inaccurate.

5.3. Action mapping to different embodiments

We demonstrate the mapping of the abstract state machine by showing a pick and place task that needs first clearing the path to the object to be grasped. This is achieved by developing two simple abstract state machines, the first to push an object away (see Fig. 18(a)) to clear the path to perform the second action: a simple pick and place (see Fig. 18(b)). To enable mapping of the ASM, we implemented the translation component described in Section 3.2 for two different platforms, *Tombatossals* and a 6-DOF Melfa RV-3SB arm with a 1-DOF WRT-102 gripper from Weiss Robotics. The

Abstract primitive	*Control primitive*	*Other parameters*	*Control and sensor requirements*
Grasp	Grasp	Grasp preshape	Arm control, tactile sensors
Move	Move	Trajectory	Arm control
Push	Move	Trajectory	Arm control
Transport	Transport	Trajectory, constraints	Arm control, tactile
Place	Transport	Trajectory, constraints	Arm control, tactile
Slide	-		
Release	Release	Hand position	Arm control

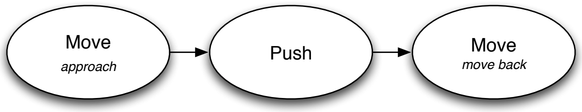
Table 5: Primitives for the Melfa platform.

WRT-102 gripper is based on the PG-70 parallel jaw griper from Schunk. The implementation included the required platform specific controllers for the different states in the ASM and the platform specific transitions, as well as the required configuration information.

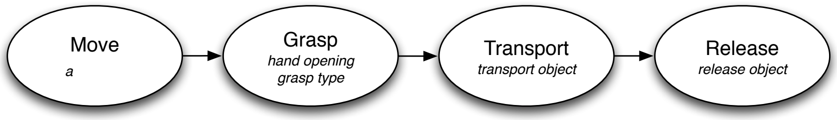
While the translation and the different requirements for the primitives are shown for Tombatossals in Table 3, the same information is available for the Melfa platform in Table 5. When comparing the two tables, the Melfa platform does not utilize as much sensor feedback in the primitives due to the difference in hardware. The primitives for the Melfa platform are, in general, different from the primitives presented in Section 4 as the SDH hand integrated into Tombatossals is much more capable in terms of DOFs for example. The effects of the use of different embodiments can be seen, for example, in the *grasp* primitive. The Melfa robot is not able to do any of the corrections that *Tombatossals* does, it is only possible to perform the force adaptation using the tactile sensors.

As a result, shown in Fig. 19, we were able to push away one object and grasp the second one based only on the sensor data from the hand and the arm, when given estimates of the pose of the objects. Using the same abstract state machines for both platforms shows clearly that we are able to use abstraction and then turn this abstract information to platform specific primitives and transitions used in the sensor-based control.

In the context of the demonstration we used the same Cartesian controllers for both arms. On the other hand, the hands are too different in terms of kinematics and sensors so that each hand had its own implementation of control. Also the transitions for grasp stability or instability were customized for each of the platforms in order to effectively use the different sensor capabilities available on the platforms. It should be noted that the task was nevertheless described using only the abstract description, without any embodiment specific information.



(a) Push object abstract state machine.



(b) Pick and place abstract state machine.

Figure 18: Abstract state machines tested on UJI and LUT platforms.

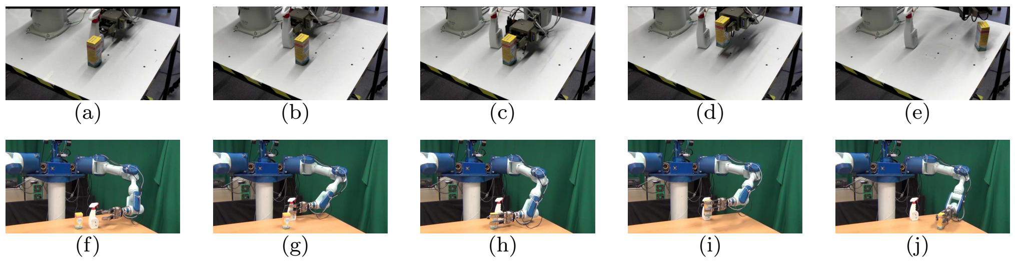


Figure 19: Action execution on different platforms, (a)-(e) Melfa RV-3SB with PG70, (f)-(j) *Tombatossals*: (a,f) Approach; (b,g) Push; (c,h) Grasp; (d,i) Lift; (e,j) Release.

6. Discussion

In the approach presented in this paper, the primitives have been implemented manually for each embodiment. Though all of them are intended to have the same behaviour and effects, following the principles described in Sec. 2, their implementation can vary significantly due to the mechanic, kinematic, and perceptual differences between embodiments. Especially in the case of the grasping primitives, the differences in hand construction (number of fingers, number of joints, number of actuators) and available sensors are on a level which makes automatic construction of primitives a grand challenge. In order to better exploit the advantages of each embodiment, it would be possible to use a more detailed abstract description of the grasping primitive, for example, differentiating between grasp types such as enveloping grasps, power grasps, precision grasps, or hook grasps. Nevertheless, the state-of-the art in grasping does not currently allow this level of abstract information to be used automatically and therefore each of the grasp flavors would need to be adapted manually, depending on the hand characteristics. This is the case especially if the full reactive capabilities of the embodiment are to be used.

On the other hand, primitives related primarily to control of arm motions can be general so that the same primitive controller can be used on multiple embodiments, as we demonstrated experimentally. However, in order to enable the full capabilities of an embodiment to be used, the path planning of the arm motions needs to consider the particular embodiment. This means that the factory component of the translation process is embodiment dependent, at least to some extent. Nevertheless, the planning collision free paths between end-effector poses can be performed using openly available software libraries and therefore the implementation of the factory is possible with reasonable effort.

The position of the factory component is central in the approach. While being outside the scope of this paper, the framework would allow things such as the choice of a grasp type to be performed by the embodiment specific factory. Differing capabilities of different embodiments, for example the size of the workspace, have the effect that there is no way to guarantee that an abstract plan would be translatable to any embodiment. Without requiring certain capabilities, it cannot be known with a certainty that a specific abstract plan can be executed on a specific embodiment. In the longer term, general principles on how embodiments could automatically instantiate sensor-based primitives would offer great benefits. This is a grand challenge in itself and possible solutions are outside the scope of the present paper. However, a complete solution would need to 1) analyse and abstract a skill performed by an existing system and 2) be able to map the abstract skill to the present embodiment.

7. Conclusion

This paper studied the capabilities of reactive control primitives in an abstraction framework allowing multiple embodiments. From a practical point of view, the main contributions are threefold. Firstly, a robust reactive grasp primitive was presented. Experiments verified that the reactive control was able to recover successfully from significant planning errors. Secondly, it was shown that the combination of several manipulation primitives could be used successfully to complete a complex task, emptying a box of unknown objects. The experimental results showed, not surprisingly, that increased perceptual capabilities improved the performance. However, a more interesting finding is that even under the worst conditions, in the blind grasping approach with only tactile feedback, the combination of reactive primitives was usually able to complete the task successfully, even though the time required was increased. The third and final main contribution is the abstraction framework and translation mechanism. We showed experimentally that the transfer of action plans was possible between different system setups while retaining the specific reactive capabilities of each embodiment.

The results support the paradigm based on reactive manipulation primitives as a good way to not only generate and execute plans in unstructured and uncertain scenarios, but also as a way to share plans, and more generally, knowledge, between different embodiments. These results complement recent results from the RoboEarth project, where similar results have been shown for the higher level planning without the viewpoint of reactive primitives presented in this paper. The results encourage us to believe that manipulation problems can be solved in complex, unstructured scenarios while retaining hardware independence on a higher level. However, immediate feedback capabilities seem essential in coping with the complexity of the world.

Many interesting open issues remain for the future. Firstly, the embodiment specific primitive controllers currently require careful design for each embodiment. Procedures which could automatically at least bootstrap the building of the controllers, or even construct the controllers, would be very valuable. It seems that the use of machine learning techniques would be an interesting and possibly profitable avenue of research in this direction. This approach would most likely require high quality simulations of the embodiment in order to provide training data for the learning approaches.

Secondly, unstructuredness and uncertainty can appear at different levels and in different aspects. The primitives presented here are mostly related with tolerating uncertainty in object pose and shape. In order to design primitives for other types of uncertainties and unexpected events, other primitive designs would be necessary, and most importantly a scheme to coordinate and group different strategies, for example hierarchically, would be necessary.

Over the years, the approaches of robot grasping have split into two groups of approaches. On one hand, object and planning based robot grasping focuses on considering a grasp as a set of contact locations on the object shape, through which manipulation forces are exerted on the object. On the other hand, hand and control based approaches rely on the capabilities and constraints of the robot embodiment, focusing on control aspects. The proposed manipulation primitives paradigm belongs to the latter approach, considering grasps as starting conditions for the action and letting the control loop and the real world itself guide the execution. It is the authors' firm belief that the inclusion of reactive capabilities is essential in coping with the whole scope of complexity present in the real world.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement ICT-215821 and from Academy of Finland under grant 250932.

References

[1] R. Brooks, A robust layered control system for a mobile robot, Robotics and Automation, IEEE Journal of 2 (1) (1986) 14 – 23. `doi:10.1109/JRA.1986.1087032`.

[2] J. Connell, A behavior-based arm controller, Robotics and Automation, IEEE Transactions on 5 (6) (1989) 784 –791. `doi:10.1109/70.88099`.

[3] J. Connell, Sss: a hybrid architecture applied to robot navigation, in: Robotics and Automation, 1992. Proceedings., 1992

IEEE International Conference on, 1992, pp. 2719 –2724 vol.3. doi:10.1109/ROBOT.1992.219995.

[4] R. C. Arkin, Behavior-Based robotics, The MIT Press, 1998.

[5] M. Teichmann, B. Mishra, Reactive algorithms for grasping using a modified parallel jaw gripper, in: Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, 1994, pp. 1931 –1936 vol.3. doi:10.1109/ROBOT.1994.351179.

[6] K. Hsiao, P. Nangeroni, M. Huber, A. Saxena, A. Y. Ng, Reactive grasping using optical proximity sensors, in: IEEE International Conference on Robotics and Automation, 2009, pp. 2098 –2105. doi:10.1109/ROBOT.2009.5152849.

[7] K. Hsiao, S. Chitta, M. Ciocarlie, E. Jones, Contact-reactive grasping of objects with partial shape information, in: IEEE International Conference on Robotics and Automation, 2010.

[8] D. Gunji, Y. Mizoguch, S. Teshigawara, A. Ming, A. Namiki, M. Ishikawa, M. Shimojo, Grasping force control of multi-fingered robot hand based on slip detection sing tactile sensor, in: SICE Annual Conference, 2008, 2008, pp. 894 –899. doi:10.1109/SICE.2008.4654781.

[9] J. Felip, A. Morales, Robust sensor-based grasp primitive for a three-finger robot hand, in: IEEE/RSJ International Conference on Intelligent Robots and Systems,, 2009, pp. 1811 –1816. doi:10.1109/IROS.2009.5354760.

[10] T. Speeter, Primitive based control of the utah/mit dextrous hand, in: Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, 1991, pp. 866 –877 vol.1. doi:10.1109/ROBOT.1991.131697.

[11] P. Michelman, P. Allen, Forming complex dextrous manipulations from task primitives, in: Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, 1994, pp. 3383 –3388 vol.4. doi:10.1109/ROBOT.1994.351050.

[12] J. Morrow, P. Khosla, Manipulation task primitives for composing robot skills, in: Robotics and Automation, 1997. Proceedings., 1997

IEEE International Conference on, Vol. 4, 1997, pp. 3354 –3359 vol.4. doi:10.1109/ROBOT.1997.606800.

[13] Y. Hasegawa, M. Higashiura, T. Fukuda, Object manipulation coordinating multiple primitive motions, in: Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on, Vol. 2, 2003, pp. 741 – 746 vol.2. doi:10.1109/CIRA.2003.1222273.

[14] N. Krger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wrgtter, A. Ude, T. Asfour, D. Kraft, D. Omrcen, A. Agostini, R. Dillmann, Object-action complexes: Grounded abstractions of sensori-motor processes, Robotics and Autonomus Systems 59 (2011) 740 – 757.

[15] M. Prats, P. Sanz, A. del Pobil, A framework for compliant physical interaction, Autonomous Robots 28 (2010) 89–111, 10.1007/s10514-009-9145-8.
URL http://dx.doi.org/10.1007/s10514-009-9145-8

[16] S. Schaal, Dynamic movement primitives. a framework for motor control in humans and huma
Adaptive Motion of Animals and Machines (2006) 261?280.
URL http://www.springerlink.com/index/k132j167h0825381.pdf

[17] S. Schaal, J. Peters, J. Nakanishi, A. Ijspeert, Learning movement primitives, in: P. Dario, R. Chatila (Eds.), Robotics Research, Vol. 15 of Springer Tracts in Advanced Robotics, Springer Berlin / Heidelberg, 2005, pp. 561–572.

[18] A. Morales, P. Sanz, A. del Pobil, A. Fagg, Vision-based three-finger grasp synthesis constrained by hand geometry, Robotics and Autonomus Systems 54 (6) (2006) 496–512.

[19] D. Aarno, J. Sommerfeld, D. Kragic, N. Pugeault, S. Kalkan, F. Wörgötter, D. Kraft, N. Krüger, Early reactive grasping with second order 3D feature relations, in: IEEE Conference on Robotics and Automation (submitted, 2007.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.2176

[20] P. Azad, T. Asfour, R. Dillmann, Combining appearance-based and model-based methods for real-time object recognition and 6D-localization, in: International Conference on Intelligent Robots and Systems, Beijing, China, 2006.

[21] T. Murphy, D. Lyons, A. Hendriks, Stable grasping with a multi-fingered robot hand: A behavior-based approach, in: IEEE/RSJ International Conference on Robotics and Intelligent Systems, Vol. 2, Yokohama, Japan, 1993, pp. 867–874.

[22] J. Coelho Jr., R. Grupen, A Control Basis for Learning Multifingered Grasps, Journal of Robotic Systems 14 (7) (1997) 545–557.

[23] R. Platt, A. H. Fagg, R. Gruppen, Nullspace composition of control laws for grasping, in: IEEE International Conference on Robots and Intelligent Systems, Lausanne, Switzerland, 2002, pp. 1717–1723.

[24] T. Mouri, H. Kawasaki, S. Ito, Unknown object grasping strategy imitating human grasping reflex for anthropomorphic robot hand, Journal of Advanced Mechanical Design, Systems, and Manufacturing 1 (1) (2007) 1–11.

[25] Y. Bekiroglu, J. Laaksonen, J. Jorgensen, V. Kyrki, D. Kragic, Assessing grasp stability based on learning and haptic data, IEEE Transactions on Robotics 27 (3) (2011) 616–629.

[26] P. Allen, K. Roberts, Haptic object recognition using a multi-fingered dextrous hand, Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on (1989) 342–347 vol.1`doi:10.1109/ROBOT.1989.100011`.

[27] M. Huber, R. Grupen, Robust finger gaits from closed-loop controllers, IEEE/RSJ International Conference on Robotics and Intelligent Systems 2 (2002) 1578–1584 vol.2. `doi:10.1109/IRDS.2002.1043980`.

[28] P. Allen, A. T. Miller, P. Oh, B. Leibowitz, Using tactile and visual sensing with a robotic hand, in: IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, 1997, pp. 677–681.

[29] B. J. Grzyb, E. Chinellato, A. Morales, , A. P. del Pobil, Robust grasping of 3D objects with stereo vision and tactile feedback, in: International

Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), Coimbra, Portugal, 2008, pp. 851 – 858.

[30] M. Kazemi, J.-S. Valois, J. A. D. Bagnell, N. Pollard, Robust object grasping using force compliant motion primitives, Tech. Rep. CMU-RI-TR-12-04, Robotics Institute, Pittsburgh, PA (January 2012).

[31] L. Petersson, M. Egerstedtt, H. Christensen, A hybrid control architecture for mobile manipulation, in: Proc. IEEE/RSJ IROS'99, 1999, pp. 1285–1291.

[32] J. Laaksonen, J. Felip, A. Morales, V. Kyrki, Embodiment independent manipulation through action abstraction, in: Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, USA, 2010, pp. 2113 –2118.

[33] R. Ellenberg, R. Sherbert, P. Oh, A. Alspach, R. Gross, J. Oh, A common interface for humanoid simulation and hardware, in: Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on, 2010, pp. 587 –592. `doi:10.1109/ICHR.2010.5686325`.

[34] M. Tenorth, A. Perzylo, R. Lafrenz, M. Beetz, The roboearth language: Representing and exchanging knowledge about actions, objects, and environments, in: IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 2012.

[35] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, in: Robotics and Automation. Proceedings. 1985 IEEE International Conference on, Vol. 2, 1985, pp. 500 – 505. `doi:10.1109/ROBOT.1985.1087247`.