Chapter 3

Manipulation primitives

As detailed in Chapter 2, neuroscience studies concluded that humans likely perform tasks as a set of different action-phase controllers that attain task subgoals. Experimental results from Chapter 2, suggest that humans learn a set of corrective reflexes that are triggered automatically during the execution of action-phase controllers in order to adapt to unexpected events.

This chapter, inspired by the action-phase controllers, proposes the paradigm of manipulation primitives, a tool for modelling and execution of reactive manipulation actions. Manipulation primitives constitute a vocabulary of atomic sensor-based actions, which can be coordinated using graphical methods to describe complex tasks.

We define a *manipulation primitive* as a single reactive controller, designed to perform a specific primitive action on a particular embodiment. Each primitive is parametrized 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.

The strength of the manipulation primitives paradigm is demonstrated by developing a set of primitives for object transport and manipulation. After providing the implementation and testing of several basic primitives, two examples of a complex task composed by those primitives are shown.

The embodiment independence that this paradigm enables, is detailed and discussed in Chapter 6 together with the main system architecture.

3.1 Related Work

3.1.1 Manipulation primitives

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 [Speeter, 1991], to define object movements [Michelman and Allen, 1994] and its relations [Morrow and Khosla, 1997] and to control a manipulator [Hasegawa et al., 2003]. 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 [Krüger et al., 2011] and the physical interaction framework of [Prats et al., 2010].

An apparently similar concept are Dynamic Movement Primitives (DMPs) introduced by Schaal et al. [Schaal, 2006]. 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 [Schaal et al., 2005], and can even be adapted reactively as the scene is observed to change. However, they are basically different idea of primitives presented in this chapter. DMPs are focused on motion description in a lower level, while our primitives describe robot manipulation skills.

3.1.2 Sensor based control

As suggested in Chapter 2, corrective reflexes are a mechanism used by humans to adapt to unexpected events while manipulating objects. An approach to address the problem of unknown environments is to use sensors, for example vision, to build the necessary models. Vision has been used to obtain the shape of unknown target objects [Morales et al., 2006, Aarno et al., 2008] and to determine the location and pose of objects [Azad et al., 2006]. In both cases, visual input was used to plan feasible grasps. On the other hand, visual feedback can also be used on-line during reaching for an object. [Murphy et al., 1993] 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 [Coelho Jr. and Grupen, 1997, Platt et al., 2002, Mouri et al., 2007, Bekiroglu et al., 2011] or the shape of an object [Allen and Roberts, 1989] 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 [Coelho Jr. and Grupen, 1997, Huber and Grupen, 2002]. Several works have combined the use of several sensors to complete the process of grasp planning and execution [Allen et al., 1997, Grzyb et al., 2008].

3.2 Manipulation primitives framework

A *manipulation primitive* is a single reactive controller, designed to perform a specific primitive action on a particular embodiment. However, there are primitive actions that do not involve physical interaction but perception. Hence, we define *perceptual primitives* as sensor based processes focused to obtain information from the environment or

detect events. *Events* represent the detection of a specific perceptual or internal condition. Primitives are together with events the elementary symbols of a vocabulary that is used to describe tasks. *Tasks* are defined as a cyclic, directed, connected and labelled multi-graph where the nodes correspond to primitives (or other tasks) and the edges to events. It is important to highlight that a task, can also have as nodes, not only primitives but other tasks. A *task* is usually a semantically meaningful goal, such as emptying a grocery bag or clearing a table. An example of task definition for clearing a table is depicted in Fig. 3.1.

The main concepts of the manipulation primitives paradigm are summarized in the following list:

Manipulation primitive A reactive controller, designed to perform a specific primitive action on a particular embodiment.

Perceptual primitive A sensor based process focused to obtain information from the environment or detect events.

Events Represent the detection of a specific perceptual or internal condition and are triggered by manipulation or perceptual primitives.

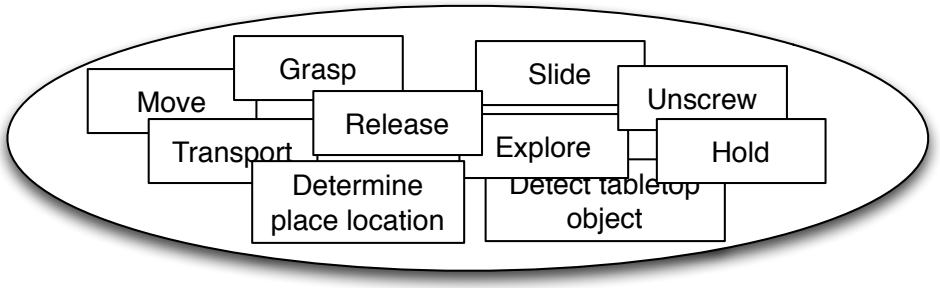
Task A cyclic, directed, connected and labelled multi-graph where the nodes correspond to *manipulation primitives* or *tasks* and the edges to *events*.

Plan An instance of a *task* with the parameters set for a specific execution in a determined scenario and context.

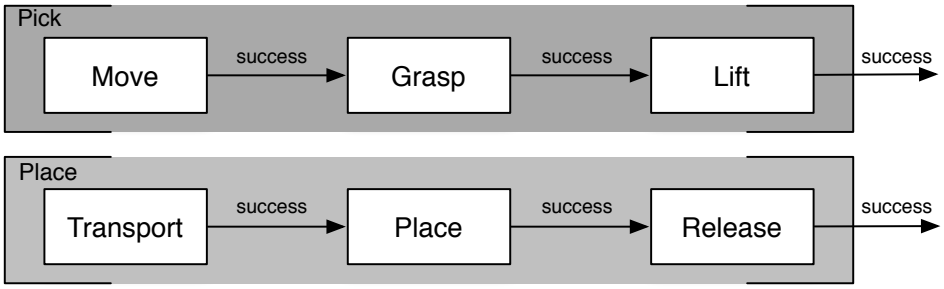
Manipulation primitives are parametrizable, thus, a task planner requires some information (i.e. parameters) to tune and instantiate tasks for a specific scenario. The description of such a planner is out of the scope of this thesis but its outcome must be a composition of parametrized primitives to address the target scenario (See Fig. 3.1). Finally, primitives can finish with several degrees of accomplishment, which in its more simple expression would be an event from the pair *Success/Failure*.

Although some of the parameters are set by the task definition itself, there are still some others that must be defined for a specific scenario and environment state. Moreover, the parameters can be defined and changed on-line according to the result of other primitives or internal conditions. An instantiation of a task with defined parameters according to the current scenario is called a *plan*.

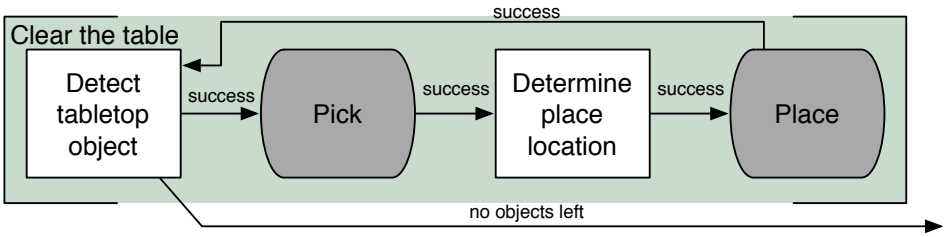
In this thesis, the focus is not in pure perception but in sensor based manipulation. However, a set of *perceptual primitives* is necessary for high level task definitions. For example, an important role of the perceptual primitives is the object detection, recognition and scene understanding that can lead the parameter tuning for other subsequent primitives during the execution of a plan.



(a) Primitives are the elementary symbols of a vocabulary used to compose tasks.



(b) Simple tasks can be composed combining manipulation primitives connected by events.



(c) Definition of the clear the table task using pick and place sub-tasks and perceptual primitives.

Figure 3.1: Composition of the clear the table task. Starting with the vocabulary of primitives, the primitives are connected by events to form simple pick and place tasks. Then the pick and place tasks are used in combination with more primitives to define the clear the table task.

Primitives are embodiment specific. However, embodiments with similar capabilities allow the definition of primitives with similar behaviour 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*. The abstraction mechanisms are described in Chapter 7.

3.3 Sensor-based primitives for manipulation

Table 3.1 shows the implemented vocabulary of primitives, their description and some examples of tasks where the primitives could be used. This set of primitives is sufficient to deal with many of the manipulation tasks a robot may encounter in household, real-life scenarios. However, there are potentially many other manipulation primitives that could be implemented, to enable a robot to perform a wider variety of tasks, or improve the performance of some of them. Some examples to extend the presented primitive vocabulary are push, pull and button pushing. Moreover, focused primitives for specific tool use, would further extend the tasks that the robot is able to perform.

Primitive	*Short description*	*Task examples*
Grasp	Secure a stable grasp that rigidly attaches an object the hand	Clear the table, empty a box
Transport	Move a grasped object from the a starting to a target position	Clear the table, bring water
Place	Move the grasped object towards a supporting surface to place it	Set the table, load the dishwasher
Release	Release a grasped object opening the hand and moving the arm if necessary	Pour water, load the dishwasher
Slide	Slide an object over a surface towards the target position	Wipe the table
Explore	Move the hand towards a specified direction and stop when a contact is detected	Empty a grocery bag
Unscrew	Perform a series of grasp and twist movements on an object cap to unscrew it	Pour water

Table 3.1: Description of the implemented vocabulary of manipulation primitives.

Name	*Parameters*	*Control and sensor requirements*
Robust grasp	Pregrasp size, grasp preshape	Arm control, FT and tactile sensors
Transport	Obstacles, trajectory, constraints	Arm control
Place	Contact threshold	Arm control, Force-torque sensor
Release	Hand position	Arm control
Slide	Slide force threshold	Arm control, Force-torque sensor
Explore	Direction, Contact threshold	Arm control, Force-torque sensor
Unscrew	Grasp force, pull force threshold, unscrew angle	Arm control, FT and tactile sensors

Table 3.2: Implemented manipulation primitives, parameters and requirements.

The primitives are described independent of a particular hardware platform but a set of control and sensor requirements are needed to implement each one of them (See Table 3.2).

All the primitives are parametrizable, requiring one common parameter: an approach vector (6D pose). It will be used conveniently depending on the primitive purpose. All the implemented primitives, together with some of their optional parameters are listed in Table 3.2. The primitives detailed in this section were implemented for either the right (Barrett Hand) or left (Schunk SDH2 Hand) hands of the UJI Humanoid torso: Tombatossals. See more details about the robotic platform in Appendix A.1.

3.3.1 Grasp primitive

Object manipulation requires direct or indirect interaction with objects. Although objects can be manipulated using tools, non-prehensile manipulation or caging grasps, it usually requires to establish a rigid relation between the target object and the robot end effector. In that scenario, the manipulated object is usually attached to the robot end effector after a grasp execution.

Regardless of the grasping taxonomy considered (e.g. [Feix et al., 2015], [Cutkosky, 1989]), there is a wide variety of grasp types. However, this primitive provides a generic grasping ability that can be used for the main grasping requirements of a robot. For other types of non-prehensile manipulation, specific grasps for tool use or control panel interaction, other focused primitives should be used.

The simplest implementation of a 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 [Felip and Morales, 2009]. We propose a novel sensor based controller that performs several corrective movements in order to achieve a stable grasp. The purpose of the corrective movements is to place the hand in a position that is more likely to produce a stable grasp than the initial position. For the implementation of corrections, inspiration was taken from the human experiments presented in Sec. 2.3.6.

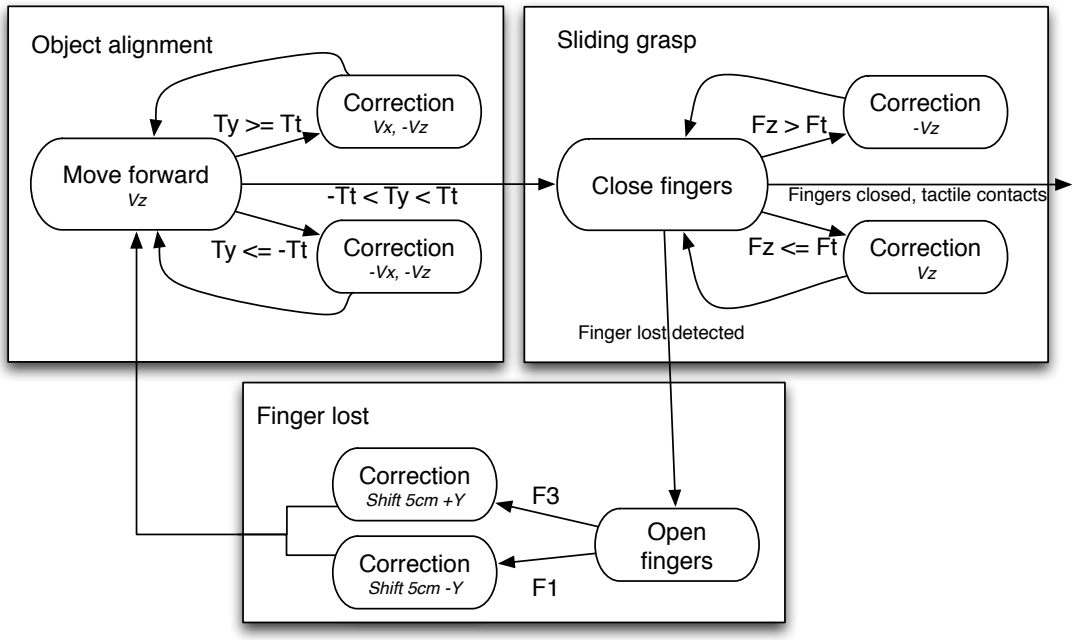


Figure 3.2: Robust grasp primitive. Fz is the force in the Z hand axis, Ty is the torque on the Y hand axis, Tt is the Y torque threshold, Ft is the Z Force threshold, Vz is a velocity in the Z hand frame. Vx is a velocity in the X hand frame. Important reference frames are depicted in Fig. 3.3.

Primitive description

The initial assumption of the grasp primitive is that the hand is near the object, at a close distance. If the starting position of the hand has been carefully planned and the positioning in the vicinity of the object executed accurately, the hand should only move towards the specified grasp direction and close to obtain a stable grasp of the object. Unfortunately, this is not often the case, a series of corrective movements are thus performed in order to obtain a robust grasp. These corrections are executed sequentially and divide the primitive execution into three main phases: alignment, sliding grasp and force adaptation (See Fig. 3.2).

A previous version of this primitive was published in [Felip and Morales, 2009]. A similar strategy is used by [Kazemi et al., 2012] also showing robustness and better grasp performance under uncertainty than non-reactive approaches.

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

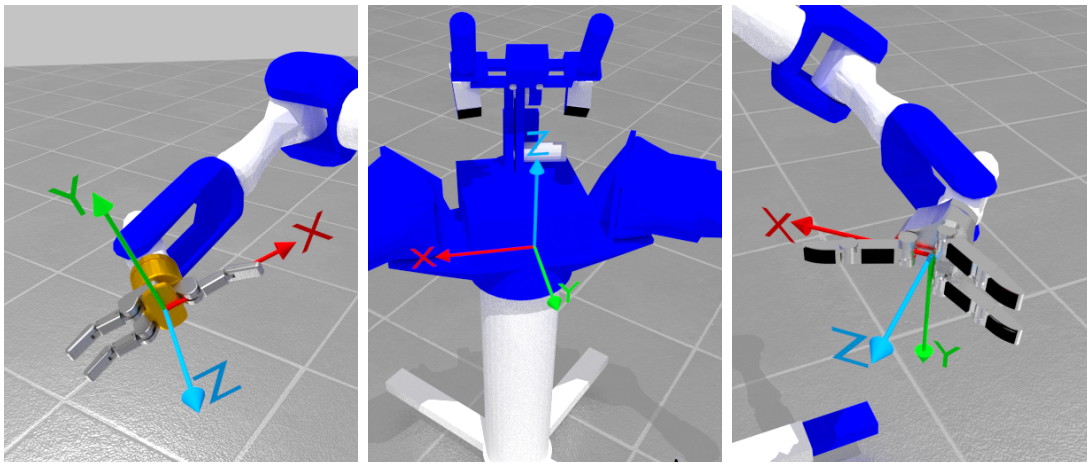


Figure 3.3: Robot important reference frames. Left: Right hand reference frame. Center: Robot base reference frame. Right: Left hand reference frame.

grasp type used (if not set by the PREGRASP_TYPE parameter) will depend on the hand configuration in the instant when the primitive takes control.

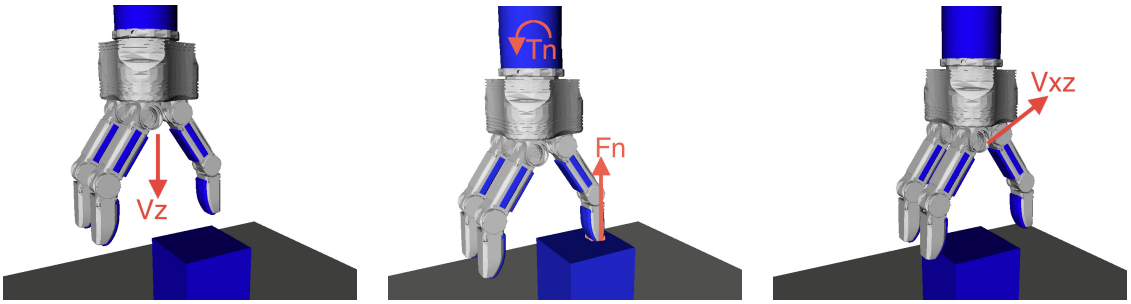
Alignment

In some situations, the initial starting position and the grasp direction result in the hand not correctly facing the center of the object (Fig. 3.4a), and thus there is a premature collision during the approaching (Fig. 3.4b). This contact is detected using a wrist mounted force-torque sensor. Using the torque, the contact point is estimated and a correction is performed to center the object (Fig. 3.4c).

$$\vec{v} = \begin{cases} \vec{v_x} - \vec{v_z} & \text{if } T_y > T_{treshold} \\ -\vec{v_x} - \vec{v_z} & \text{if } T_y < -T_{treshold} \\ \vec{v_z} & \text{otherwise} \end{cases} \tag{3.1}$$

Eq. (3.1) 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 (w.r.t. hand frame). The alignment is finished when a contact is detected and $-T_{treshold} \leq T_y \leq T_{treshold}$

An example of an execution of this phase is depicted in Fig. 3.4. The contact can be also detected using the tactile sensors available. Alignment correction improves grasping of objects with location uncertainty by allowing the hand to align its center with the



(a) (b) (c)

Figure 3.4: Grasp primitive: Alignment phase. (a)Arm moving towards the object. (b)Contact generates torque in the wrist. (c)Correction movement is performed.

object. The effects of this alignment phase are analysed in Sec. 3.4 where a thorough evaluation and comparison is conducted and the results are discussed.

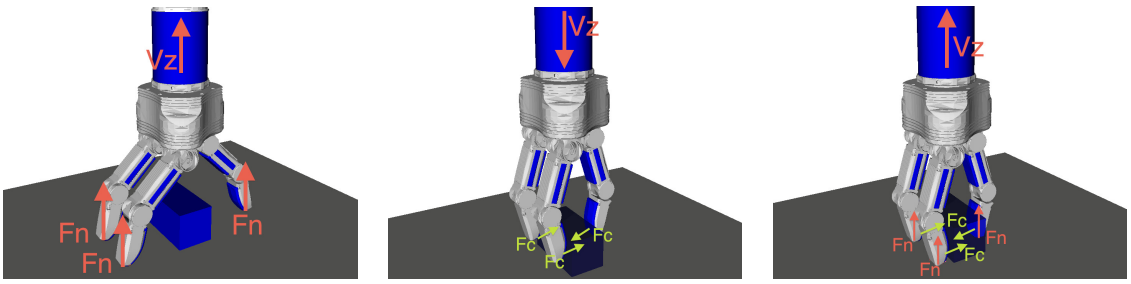
Sliding grasp

When approaching, the hand makes contact in occasions with the supporting surface instead of the object (See Fig. 3.5(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 forward or backward depending on the force sensed along its Z axis. Concurrently the fingers are closing (see Fig. 3.5) 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.

It is important to highlight that this phase is also suitable to grasp objects laid out on other surfaces different from tables or workbenches, such as handles on doors, drawers, dishwashers and so on.

Equation 3.2 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.

$$\vec{v} = \begin{cases} -\vec{v_z} & \text{if } Fz_{sensor} <= -Fz_{threshold} \\ 0 & \text{if } -Fz_{threshold} < Fz_{sensor} < Fz_{threshold} \\ \vec{v_z} & \text{if } Fz_{sensor} >= Fz_{threshold} \end{cases} \tag{3.2}$$



(a) (b) (c)

Figure 3.5: Grasp primitive: Sliding grasp phase. (a)The fingers contact the table while closing and the normal force Fn is detected by the force sensor. Thus the controller sets the velocity to $-\vec{v_z}$ in order to move the hand back. (b)The fingers are closing and although the fingers touch the object exerting the contact force Fc, the contact with the table is lost. Therefore, $\vec{v_z}$ is set forwards. (c)The hand contacts the table again but the object is already grasped and the fingers can no longer keep closing, hence the sliding grasp phase ends.

The behavior of this correction phase is shown in Fig. 3.5. 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. 3.5(a)). The fingers continue closing and because no contact force is detected, the arm moves forward (Fig. 3.5(b)). In Fig. 3.5(c) the fingers are not able to close anymore and the sliding grasp ends.

Finger lost correction

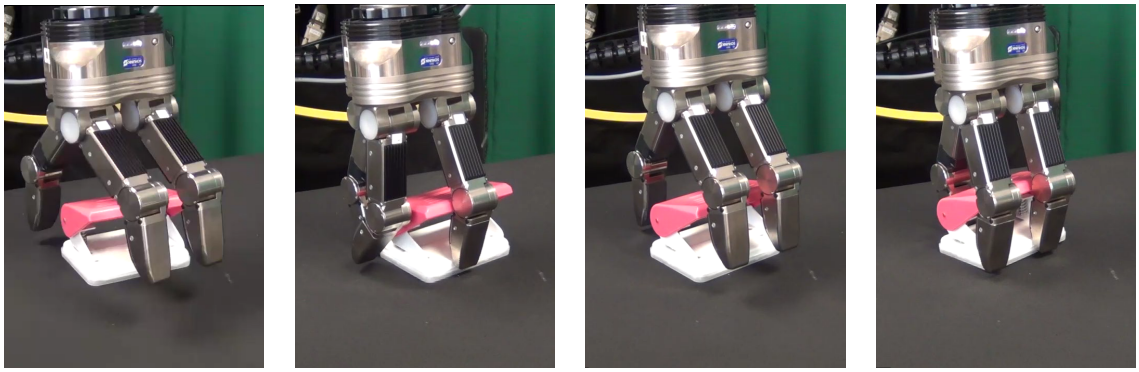
It may happen, if the error is big enough, that the hand closes and one of the external fingers lose contact with the object. In this case, the *finger lost* corrective movement is triggered, the hand opens and shifts towards the detected contacts in order to place all the fingers on the object. Fig. 3.6 shows the execution of such a corrective movement in a real experiment.

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 by the joint configuration or contact information.

Primitive parameters

- PREGRASP_TYPE (default:none): Encodes the starting configuration of the hand, for example: cylindrical, spherical or hook. If no value is specified the current hand configuration is used.



(a) (b) (c) (d)

Figure 3.6: Grasp primitive: Finger lost correction. (a) Grasping position after the alignment phase. (b) The hand closes and one finger does not contact the object. (c) The hand opens and performs a corrective movement. (d) Finally a good grasp is achieved.

- PREGRASP_SIZE (default:none): Can take a value from 0 to 1 and encodes the hand opening prior to the grasp motion.
- FORCE_THRESHOLD (default:2N): Force threshold for the contact detection using the force-torque sensor. This value has to be set depending on the force-torque sensor sensitivity.
- TORQUE_THRESHOLD (default:5Nm): Torque threshold for the contact detection using the force-torque sensor. This value has to be set depending on the force-torque sensor sensitivity.
- GRASP_DIRECTION (default:[0,0,1]): Direction to move the hand to grasp the object using the hand frame. The important frames of the robot used for the implementation are depicted in Fig. 3.3.
- MAX_DISTANCE (default:0.2m): Max distance that the hand will move towards GRASP_DIRECTION looking for a contact. When this distance has been covered, the primitive will switch to the sliding grasp step. This parameter has to be set depending on the grasp planner and the hand size.
- CORRECTION_VELOCITY (default: 0.005m/s): Velocity used to perform the corrective movements during the alignment and sliding grasp phases of the grasping primitive.

3.3.2 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.

The problem of controlling redundant manipulators has been widely studied in robotics [Waldron and Schmiedeler, 2008, Chiaverini et al., 2008] and nowadays there are open source tools that can be tuned to provide advanced control for custom manipulators [Smits, 2015, Corke, 2011, Sucan and Chitta, 2015]. Controllers can be classified depending on the space they are controlling (e.g. joint, Cartesian) and the control type (position, velocity or effort). In order to ease the programmer's task, it is quite common that commercial robots provide and API with different joint control modes (position, velocity, effort). However, all the controllers are finally built on top of an effort controller that is converted to a voltage sent to the motors.

Moreover, on top of the controllers used, motion planning algorithms can be used to plan collision free trajectories and perform optimal movements from point A to point B.

Primitive description

This primitive implements a Cartesian velocity controller that allows the trajectory to be constrained by specifying optional parameters. A trajectory can be specified in joint space as a list of joint positions that define the state of each joint during all the transport primitive execution. However, a less restrictive definition can be used by specifying the trajectory as a list of end-effector Cartesian positions. Instead of defining the exact trajectory that the robot must follow, it is also possible to specify position, velocity or acceleration limits in the Cartesian space.

Equation 3.3 shows how the force-torque threshold parameter 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. n is a normalization term used to keep the resulting velocity inside the velocity limits. The resulting velocity is sent to the Cartesian velocity controller that converts the desired Cartesian velocity into joint velocities using the Jacobian pseudo-inverse approach. We assume that the robot provides a joint velocity control and the final conversion to joint efforts is done by the robot's internal controller.

$$\vec{v} = \begin{cases} n(x_{target} - x_{current}) & \text{if } ||F_{sensor}|| < F_{threshold} \\ 0 & \text{if } ||F_{sensor}|| >= F_{threshold} \end{cases} \tag{3.3}$$

Optional parameters can also be used to describe environment obstacles as an obstacle point cloud, in which case a force-field [Khatib, 1985] 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. 3.7 an example of a position constrained trajectory is shown. The convex hull of the box is defined as forbidden space to define position constraints.

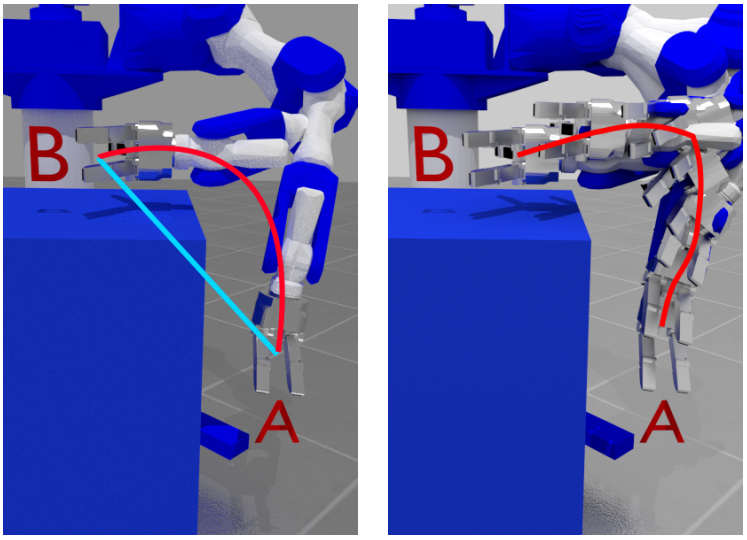
Jacobian pseudo-inverse approaches for Cartesian velocity control, are fast and can work real-time, hence they are very useful for closed loop control. However, they are gradient descent approaches and the trajectories generated may fall into local minima quite easily if the start and end positions are separated enough or the trajectories travel outside the workspace of the robot. Therefore, this control approaches are suitable for close distance movements, fine manipulation and closed loop control. For long distance movements inside the robot workspace, other approaches should be used.

As an alternative method, we have implemented an interface to well known path planning libraries such as MoveIt! [Sucan and Chitta, 2015]. In this case, instead of using an online gradient descent strategy, the trajectory is calculated in advance and a set of waypoints is provided. The plan is executed taking into account the force-torque threshold to satisfy collision security constraints. Figure 3.7(b) shows the result of calculating a motion plan with MoveIt! path planning. There is a wide variety of planning algorithms that can be used, the plugin architecture of the planning library used by MoveIt! makes it simple to switch between planners and use the best for each application. For this primitive implementation we have used the KPIECE planner [Sucan and Kavraki, 2012], it is a tree-based planner that uses a discretization to guide the exploration of the continuous space. Their authors claim that this planner has been shown to work well consistently across many real-world motion planning problems.

The selection of the method used to move the robot, can be automatically selected depending on the distance to the target. The PLANNING_DISTANCE_LIMIT parameter is used to decide which method to use, targets further that the configured distance will trigger the planning algorithm. Moreover, if the target is close but a local minima is detected while executing the movement, the control is switched to the motion planning in order to recover, if the planning fails the primitive throws an error event.

Primitive parameters

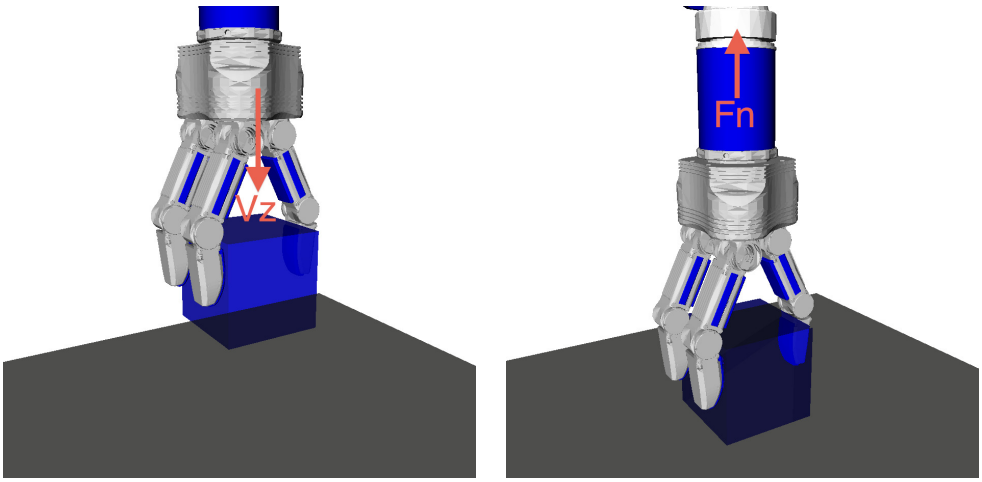
- FORCE_THRESHOLD (default: 15N): Force threshold to consider an unexpected contact.



(a) (b)

Figure 3.7: Transport primitive. (a) Example of execution of the constrained transport primitive from the starting point A to the target point B. Blue line: Standard trajectory. Red line: Position constrained trajectory. (b) Example of a calculated plan using MoveIt! planning library.

- ACCELERATION_LIMITS (default: [0,0,0,0,0,0]): End effector linear and angular acceleration limits in m/s^2 and rad/s^2. If all the values are zero, no constraints are taken into account.
- VELOCITY_LIMITS (default: [0,0,0,0,0,0]): End effector linear and angular velocity limits in m/s and rad/s. If all the values are zero, no constraints are taken into account.
- OBSTACLES (default: none): Point cloud representing obstacles. Obstacles can also be specified by geometric primitives.
- WAYPOINTS (default: none): End effector 6D pose waypoint list. The primitive will move from the starting position to the target position through the specified waypoints.
- PLANNING_DISTANCE_LIMIT (default:0.03m): Minimum distance to the target that will trigger the use of a planning approach to move the arm instead of the on-line Cartesian velocity control.



(a) (b)

Figure 3.8: Place primitive. (a)Arm moving the object towards the surface with velocity $\vec{v_z}$. (b)The object contacts the supporting surface and the normal force Fn is detected by the wrist force-torque sensor.

3.3.3 Place primitive

Placing an object involves a supporting surface that usually is a horizontal plane. However, that is not always the case, the plane can be slanted or be vertical if the task is, for example, to stick the object to the wall. Moreover, the supporting surface could not be a plane (e.g. bowl, dish). The place primitive is used to gently place an object on a supporting surface asserting the success using on-line sensor feedback.

Primitive description

The arm moves towards the supporting surface until a contact is detected. Equation 3.4 describes the control action taken depending on the force sensor readings where $v_{\vec{place}}$ represents a constant velocity defined by the VELOCITY parameter and the SURFACE_NORMAL parameter, $v_{\vec{place}} = \vec{n} \cdot v$. 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. 3.8.

$$\vec{v} = \begin{cases} -\vec{v_{place}} & \text{if } ||F_{sensor}|| < F_{threshold} \\ 0 & \text{if } ||F_{sensor}|| >= F_{threshold} \end{cases} \tag{3.4}$$

Primitive parameters

- FORCE_THRESHOLD (default: 5N): Force threshold to consider that the supporting plane has been contacted.
- SURFACE_NORMAL (default: [0,0,1]): Normal of the point of the supporting surface where the object will be placed. This will be used to determine the motion direction.
- VELOCITY (default: [0.005 m/s]): Linear velocity towards the supporting surface.

3.3.4 Release primitive

The purpose of this primitive is to release the object from a previous grasp. Releasing an object can be difficult if the fingers, while opening, collide with the supporting plane or other parts of the object (see Fig. 3.9(a)). To handle this problem, the release primitive opens the hand smoothly while the arm moves back if a contact with the environment is detected.

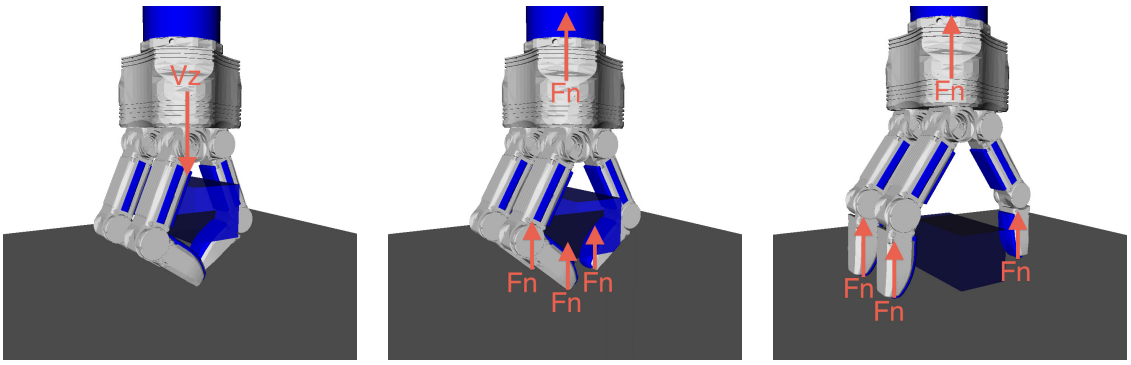
Primitive description

The movement of the arm is force-controlled and the arm only moves back if there is a contact detected between the opening fingers and the surface (see Equation 3.5). The sequence of movements that this primitive performs is shown in Fig. 3.9. This primitive can be configured by setting the target hand position after release and the $F_{threshold}$ parameter.

$$\vec{v} = \begin{cases} -\vec{v}_z & \text{if } Fz_{sensor} < -F_{threshold} \\ 0 & otherwise \end{cases} \tag{3.5}$$

Primitive parameters

- HAND_TARGET_POSITION(default: 1.0): Defines the target hand opening to consider the release primitive finished. 0.0 = fully closed, 1.0 = fully open.
- FORCE_THRESHOLD (default: 5N): Force threshold to consider that the fingers contact the supporting plane while opening.
- SURFACE_NORMAL (default: [0,0,1]): Normal of the supporting plane. This will determine the motion direction if the fingers collide with the surface in order to move the hand away from the contact.
- VELOCITY (default: [0.001 m/s]): Linear velocity used to move the hand away from the supporting surface.



(a) (b) (c)

Figure 3.9: Release primitive. (a)Hand before opening the fingers. (b)The hand cannot release the object, the fingers are blocked by the surface and cannot continue opening. 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.

3.3.5 Slide primitive

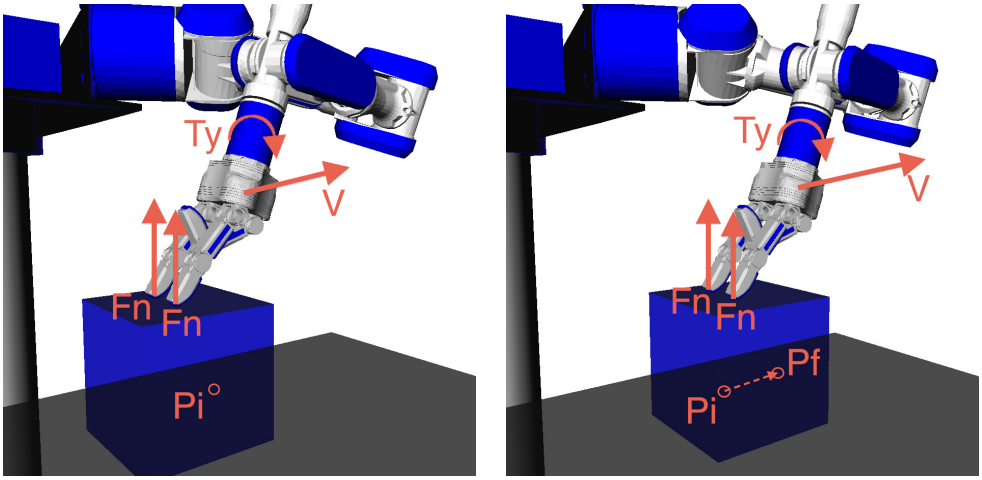
A very frequent type of non-prehensile manipulation is sliding objects on a supporting plane. The purpose of the slide primitive is to provide the robot with such an ability by exerting a constant force on an object against its supporting plane and sliding it to a target position.

Primitive description

Using force control the arm applies a force (F_n in the desired range $F_{min} < F_n < F_{max}$) to the object, then moves towards the target, keeping the applied force constant (Fig. 3.10(a)). The constant force F_n the arm and object allowing the robot to slide the object on the surface from the starting to the target position (Fig. 3.10(b)).

Equation 3.6 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}(LOWER_FORCE_THRESHOLD parameter) and F_{max} (UPPER_FORCE_THRESHOLD parameter). This primitive uses a predefined hand preshape (see Fig. 3.10).

$$\vec{v} = \begin{cases} x_{target} - x_{current} & \text{if } F_{min} < ||F_n|| < F_{max} \\ -\vec{x_z} & \text{if } ||F_n|| <= F_{min} \\ \vec{x_z} & \text{if } ||F_n|| >= F_{max} \\ 0 & otherwise \end{cases} \tag{3.6}$$



(a) (b)

Figure 3.10: 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 with velocity V. The primitive keeps the applied force stable between the lower and upper thresholds.

Primitive parameters

- UPPER_FORCE_THRESHOLD (default: 9N): Upper force threshold to consider that the hand is exerting enough pressure on the object. If the value is higher the controller will loosen the pressure moving slightly away from the object.
- LOWER_FORCE_THRESHOLD (default: 5N): Lower force threshold to consider that the hand is exerting enough pressure on the object. If the value becomes smaller, the controller will increase the pressure moving towards the object.
- SURFACE_NORMAL (default: [0,0,1]): Normal of the supporting plane. This will determine the direction used to push the object against the surface.
- TARGET_VELOCITY (default: [0.005 m/s]): Linear velocity used to move the hand from the starting position to the target position.
- CONTACT_VELOCITY (default: [0.005 m/s]): Linear velocity used to push the object against the surface and to loose the contact in case the force applied becomes to high.

3.3.6 Unscrew primitive

To have a robot able operate autonomously in household scenarios, the ability to open containers is necessary. For example, in order to pour liquids from a container, the lid

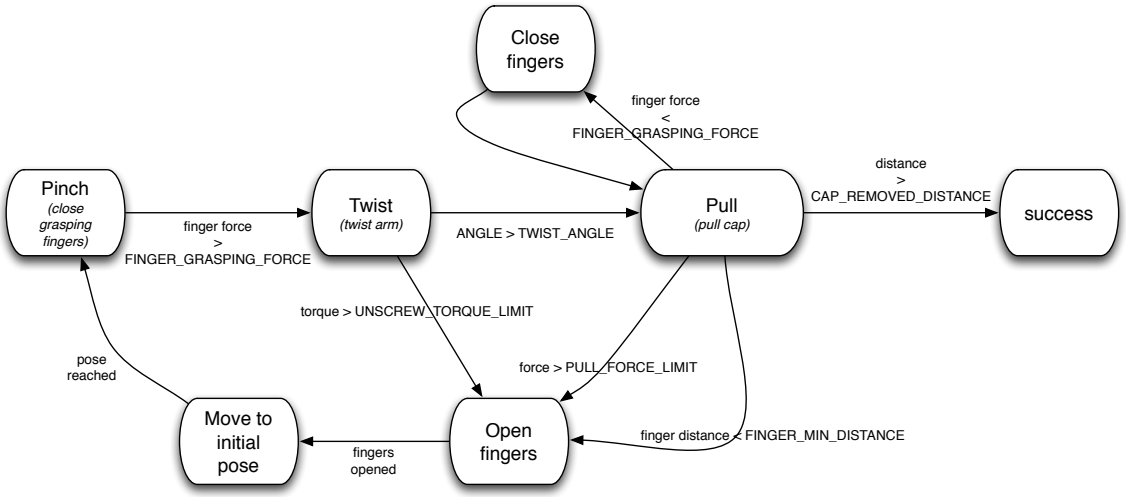


Figure 3.11: Unscrew primitive execution diagram. The execution begins with the pinch phase. This diagram shows the operation of the unscrew primitive, it does not depict a task formed of several basic primitives.

needs to be opened first. Although not all the containers have a screwed lid it is one of the most common cases.

The purpose of the unscrew primitive is to provide the robot the ability to open containers that have a screwed lid. Unlike the other primitives presented in this chapter, the unscrew primitive focuses on solving a specific manipulation interaction that would be very difficult to define as a task composed by other primitives.

This manipulation primitive assumes that the hand is over the cap ready to grasp it. Moreover, it uses several parameters to adapt its behaviour to the target object.

Primitive description

The unscrew primitive is divided into three phases: pinch, twist and pull. The phases and the transitions of the primitive are depicted in Fig.3.11. Note that the diagram shows the internal states of the unscrew primitive in order to explain its operation and does not depict a task formed of several basic primitives.

Pinch

Closes the selected fingers until the desired FINGER_GRASPING_FORCE is achieved on each fingertip. The force applied is determined using the tactile sensors on each finger.

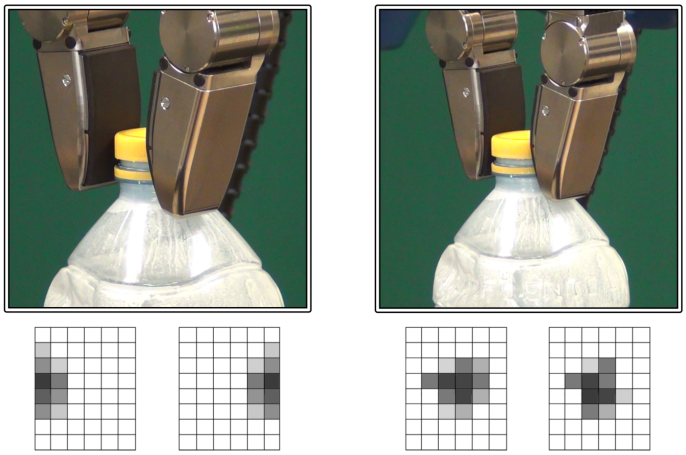


Figure 3.12: Cap grasping and tactile sensor readings. Left: Initial cap grasping, grasping center is not aligned with the cap. Right: Cap grasping after correcting the pose.

Twist

Twists the cap for the specified TWIST_ANGLE and transitions to the pull stage. The hand fingers are still controlled and in the case that the contact with the cap is lost, the fingers keep closing until the contact is detected again. This stage terminates prematurely if the torque detected is over the UNSCREW_TORQUE_LIMIT parameter or the fingers are closed without grasping the cap (inter-finger distance below FINGER_MIN_DISTANCE).

Pull

On this stage the arm pulls back to test if the cap is free. While pulling, the primitive is monitoring the tactile contacts and the force sensor. As in the previous stage, the fingers keep closing if the contact with the cap is lost. If the detected force is over the PULL_FORCE_LIMIT threshold or the fingers get closer than the FINGER_MIN_DISTANCE: the fingers open, the arm moves back to the initial position and the primitive starts again from the grasp stage. On the other hand, if the arm pulls the cap further than the CAP_REMOVED_DISTANCE the primitive terminates successfully.

Reactive adaptation

Due to the intrinsic uncertainty of real environments and systems, it is not possible to guarantee that the position of the fingers when pinching the cap is exactly aligned with the center of the cap. Moreover it is also not possible to produce approach vectors that are perfectly aligned with the cap normal vector (see Fig.3.12). Thus the cap will

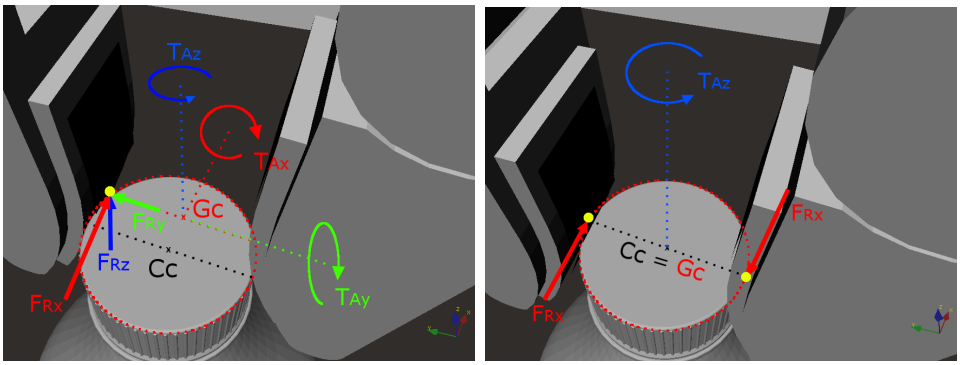


Figure 3.13: Left: Forces and torques produced when twisting the cap with an unaligned grasp point, F_{Ry} and F_{Rz} are undesired components that do not contribute to unscrew the cap but produce undesired torques. Right: Forces and torques produced with an aligned cap. C_C: Cap center. G_C: Grasp center. F_R: Reaction Force. T_A: Torque applied.

hardly be pinched by its center and the rotation axis of the hand will not be aligned with that of the cap.

The twist movement performed to unscrew the cap has the rotation center in the middle of the grasping fingers. It means that the fingers rotate around their middle point assuming that the cap axis is centred. If the cap axis does not match with the twist movement center, the unscrew movement will apply undesired forces on the object (see Fig.3.13). That fact can cause the unscrew motion to move the cap or the object away from the fingers. As the cap position is not tracked by the primitive, the object displacement can cause the subsequent grasps to fail. Thus, the cap misalignment can decrease the robustness of the primitive.

In order to solve this issue, a sensor based strategy is used. It dynamically aligns the fingers center with the cap center to improve the robustness of the primitive. During the grasp and unscrew phases, the contacts on the fingertips are recorded and averaged over time. Before moving the arm back to the starting position, if the recorded contacts are not aligned with the grasping fingers center frame, the distance to that frame is used to correct the position, see Fig.3.12. As for the robust grasp primitive, we have conducted experiments to evaluate the improvement of the reactive approaches. The experiments and results are discussed in the next section of this chapter.

Primitive parameters

- UNSCREW_TORQUE_LIMIT: Default 10 N/m. Defines the torque limit that will be applied by the robot to unscrew the cap. If higher torque is detected, a collision is assumed and the unscrew movement stops and moves back to the starting position to grasp the cap again.

- PULL_FORCE_LIMIT: Default 7N. This parameter defines the force limit when pulling from the cap of the object. If the robot applies more than the defined force when trying to remove the cap, we assume that it is not completely unscrewed, hence it stops pulling and moves back to the starting position to grasp the cap again and perform another unscrew movement.
- GRASPING_FINGERS: Default: 2. Defines the number of fingers to be used for pinching and unscrewing the cap. This parameter is used to automatically configure the preshape.
- FINGER_GRASPING_FORCE: Default: 5N. Defines the target force for each finger. Fingers will keep closing until target force is reached.
- TWIST_ANGLE: Default: 30 degrees. Defines the angle that the primitive will twist the cap on each twist stage.
- FINGER_MIN_DISTANCE: Default: 12mm. Minimum inter-finger distance that will allow an object to be in hand. If the distance is shorter, the controller assumes that the hand is closed but empty.
- CAP_REMOVED_DISTANCE: Default: 40mm. Distance that the cap has to be separated from the container to consider it free to move away.

3.4 Experiments

Three experiments have been implemented to validate and illustrate the usefulness of the manipulation primitives paradigm: validation of the robust grasp primitive, validation of the unscrew primitive and completion of a manipulation task using a set of the primitives described. The two first experiments are focused in illustrating the design of a manipulation primitive and the importance of reactive strategies. The last case is focused on showing how primitives can be combined to solve more complex manipulation tasks.

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. 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. Further information about this robotic platform is given in Appendix A.1.

3.4.1 Validation of robust grasp primitive

As explained in Chapter 2, humans perform reactive movements to adapt to unexpected sensor input. Intuitively, reactive movements should increase the grasp success, especially in real and uncertain environments. This experiment is carried out to validate the intuition and quantitatively provide information about the importance of such a reactive strategy when grasping objects.

Experimental setup

In order to compare the robust grasp primitive with a non-adaptive grasp controller, we have designed a naive grasping primitive. The experimental procedure, for each primitive, consisted on grasping 10 different household objects (see Figure 3.14) 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 of the evaluated primitives has tried 200 grasps.

A trial is considered successful if the grasp obtained after the execution is considered stable by a human observer. If the object, when lifted, does not move in the hand, the grasp is considered stable by the observer.

Naive grasping primitive

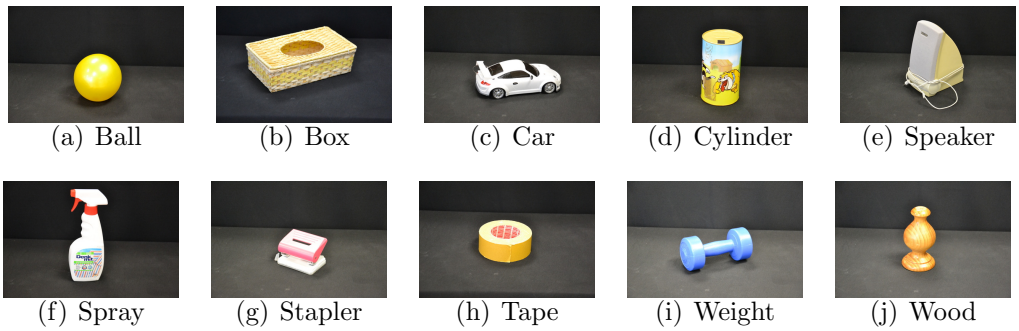
We have designed a grasping strategy that does not use tactile and force sensors to perform corrections. The naive grasping primitive behaves as follows: The arm moves forward 10cm or until a contact is detected. Then the hand closes stopping each finger that detects contact. Finally force is slightly increased on the distal phalanxes to establish the final grasp.

Environment

The experimental environment consists of the robot in front of a table. On the table there is a single test object in any position that can be reached using a top grasp by the left hand.

Test objects

In order to represent the different objects in household scenarios, an heterogeneous set of objects has been selected. The object test set is composed of ten different household objects (see Figure 3.14). There are objects with primitive shapes (cylinder, ball) and symmetries (tape, wood) but there are also more complex objects (spray, stapler). The objects are completely unknown to the robot, the primitives have no prior information about them.



(a) Ball (b) Box (c) Car (d) Cylinder (e) Speaker

(f) Spray (g) Stapler (h) Tape (i) Weight (j) Wood

Figure 3.14: Set of 10 objects used for testing the robust grasp primitive.

Assumptions

The objects have to be placed on the table inside the workspace of the arm used to perform the experiments. Moreover, the objects used, have to have lambertian properties in order to be visible by the visual system used. No other assumptions about shape, weight or other physical properties are made.

Parameter settings

There are no specific parameter settings for the primitives used in this experiments. Both the naive and robust grasp primitives are configured with their default values as presented in Section 3.3.1.

Object detection

In order to grasp the objects, the robot needs to determine 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. The methods used in this experiment to detect objects are detailed in Chapter 6.

To determine the object position, Kinect RGBD images in combination with algorithms from the Point Cloud library [Rusu and Cousins, 2011] 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. A detailed description of the visual pipeline is provided in Section 6.3.3.

Approach vector calculation

As stated in Section 3.2, any primitive requires at least an approach vector as an input parameter. Usually, the approach vector used as the starting point for a grasping procedure, is determined by a grasp planner. In this experiment, we have used a grasp planner based on the object point cloud eigen vectors to calculate the approach vector.

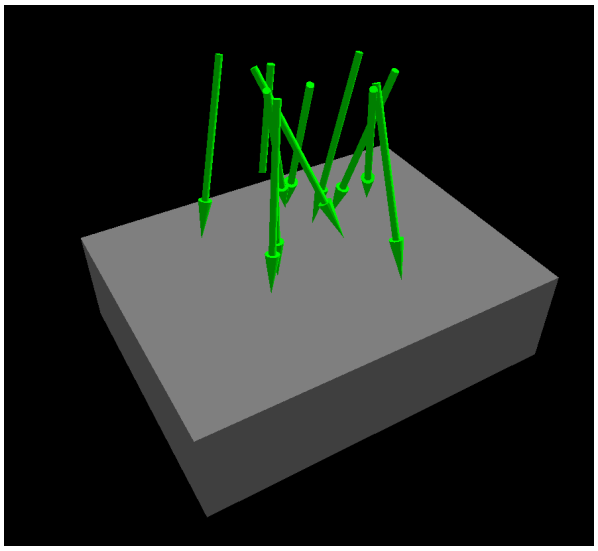


Figure 3.15: Set of 10 approach vectors after adding the random uniform error.

The input of the grasp planner is the segmented point cloud containing the object to grasp. The approach vector is calculated using the *simple grasp generator* described in Sec. 6.3.5 and the result is filtered by orientation to keep only vectors that approach from the top.

In order to test the controller under rough error conditions, we have performed tests adding a random uniform error to the approach vector calculated by the system.

The error that we have introduced to the approach vectors follows an uniform distribution, $\mathcal{U}(0, 5)$cm in each axis and $\mathcal{U}(0, 15)$degrees on each axis. Figure 3.15 shows a set of 10 approach vectors for an object after adding the uniform error.

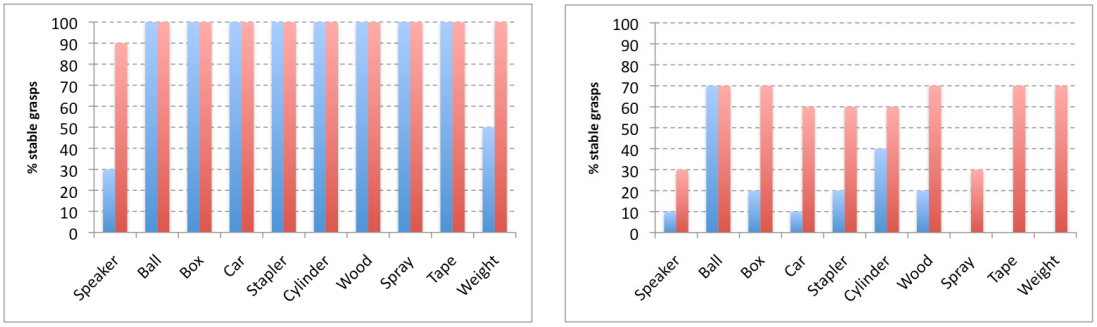
Results and discussion

Table 3.3 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 3.16 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

	stable grasps	failures	%success
naive	88	12	88%
robust	99	1	99%
naive + error	19	81	19%
robust + error	59	41	59%

Table 3.3: Grasping experiments overall results after 100 attempts.



(a) Grasping performance without error.

(b) Grasping performance with error.

Figure 3.16: Grasping performance after 10 trials per object. Blue: Naive controller, Red: Robust controller

have some properties, asymmetry 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 3.3 under error conditions the performance of the naive controller drops to 19% while the robust primitive holds a 59% of robust grasps achieved.

3.4.2 Validation of unscrew primitive

The purpose of this experiment is to validate the implementation of the unscrew primitive, testing it on different objects. Furthermore, this experiment also validates that the sensor-based reactive behaviour improves the performance of the primitive by comparing the results of tests with and without corrective movements.

Experimental setup

To validate the unscrew primitive and the impact of the reactive adaptation on its performance, an object test bench composed of eight objects has been set up and the task has been executed 10 times for each object with and without the reactive adaptation enabled. Overall the unscrew task has been executed 160 times, 80 with tactile corrections and 80 without them.

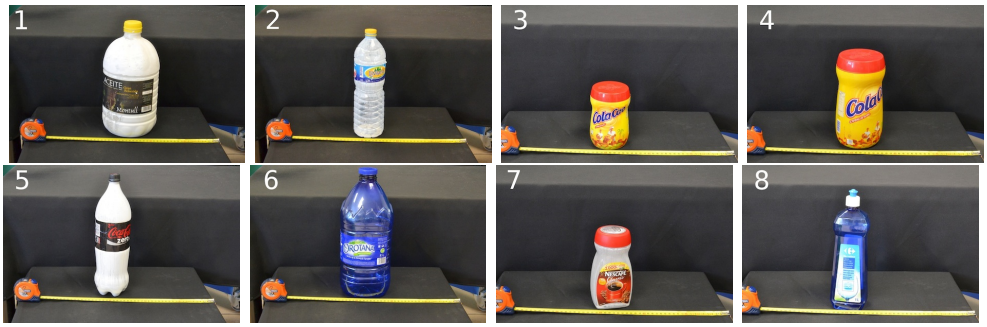


Figure 3.17: Object test bench for the cap unscrew experiments. Object id on the top right of each picture. The lenght of the measuring tape in the pictures is 40cm.

Environment

The experimental environment consists of the robot in front of a table. On the table there is a single test object in any position that can be reached by the right hand. The object is standing upright.

Test objects

The object test set is composed of eight different household objects (see Fig. 3.17). In order to represent the different types of bottles and containers, an heterogeneous set of objects with different shapes, textures, sizes, weight and with different caps was chosen. Moreover, it is important to note that the cap thread is also different from one object to another, hence they require a different number of turns to fully detach the cap from the container.

Assumptions

Although the visual system is able to deal with more than one object, to simplify the experimental process, only one object is standing in front of the robot. The objects are unknown to the robot, the only assumption is that each object has a cap that can be unscrewed and they are laid out upright on the table and inside the workspace of the left arm. Objects' surface has to have lambertian reflectance properties in order for the visual system to detect them. No other assumptions about the objects are made.

Task definition and metrics

The same visual system and pipeline used for the robust grasp validation is used to detect the objects (see Section 6.3.3.). Moreover, as the objects are standing on the table, it needs to be grasped with one hand to fix it before the other hand can perform the unscrew attempt. The grasping procedure is the same as specified in Sec. 3.4.1 but using a side approach vector instead of a top one. Once the object is grasped it is moved to a manually determined vantage position. Then the cap detection starts. It consists

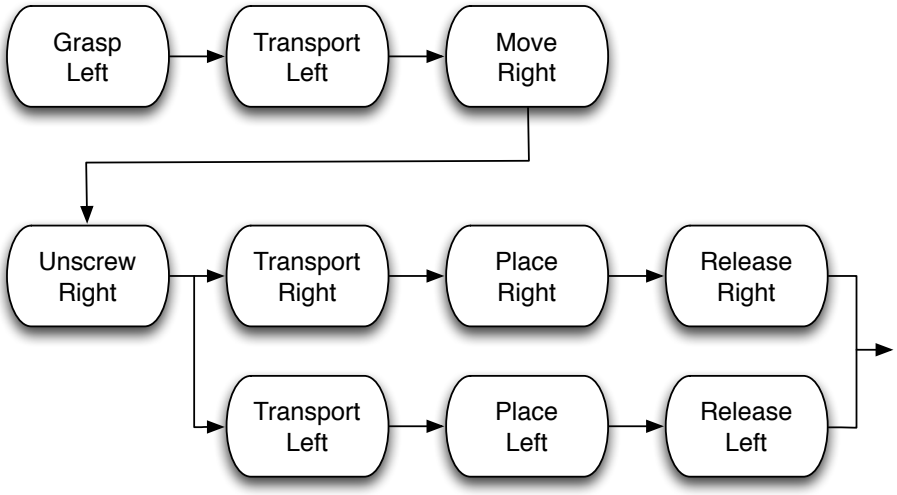


Figure 3.18: Unscrew task definition using the manipulation primitives framework. All the depicted transitions are triggered by a success event. For the sake of clarity, the error transitions are not depicted in the figure; all the primitives have a transition to an error state.

basically on determining the centroid of the upper part of the object which is assumed to be the cap. That centroid is used to generate an approach vector from the top that will be used by the unscrew primitive as the starting position. The whole grasp and unscrew task is described using the manipulation primitives framework as depicted in Fig 3.18.

For each task execution, the following values were measured:

- Time: The time it takes to complete the unscrew primitive execution.
- Twist moves: The number of twist movements required to open the container.
- Cap unscrewed: It will be considered that the robot has succeeded unscrewing the cap if at the end of the task execution the cap is separated from the recipient or can be separated from the object without turning it.
- Success: It will be considered that the whole task has succeeded if the robot removes the cap from the object and places it on the table by itself. This does not include situations where the cap falls while lifting it or as the result of a twist movement.

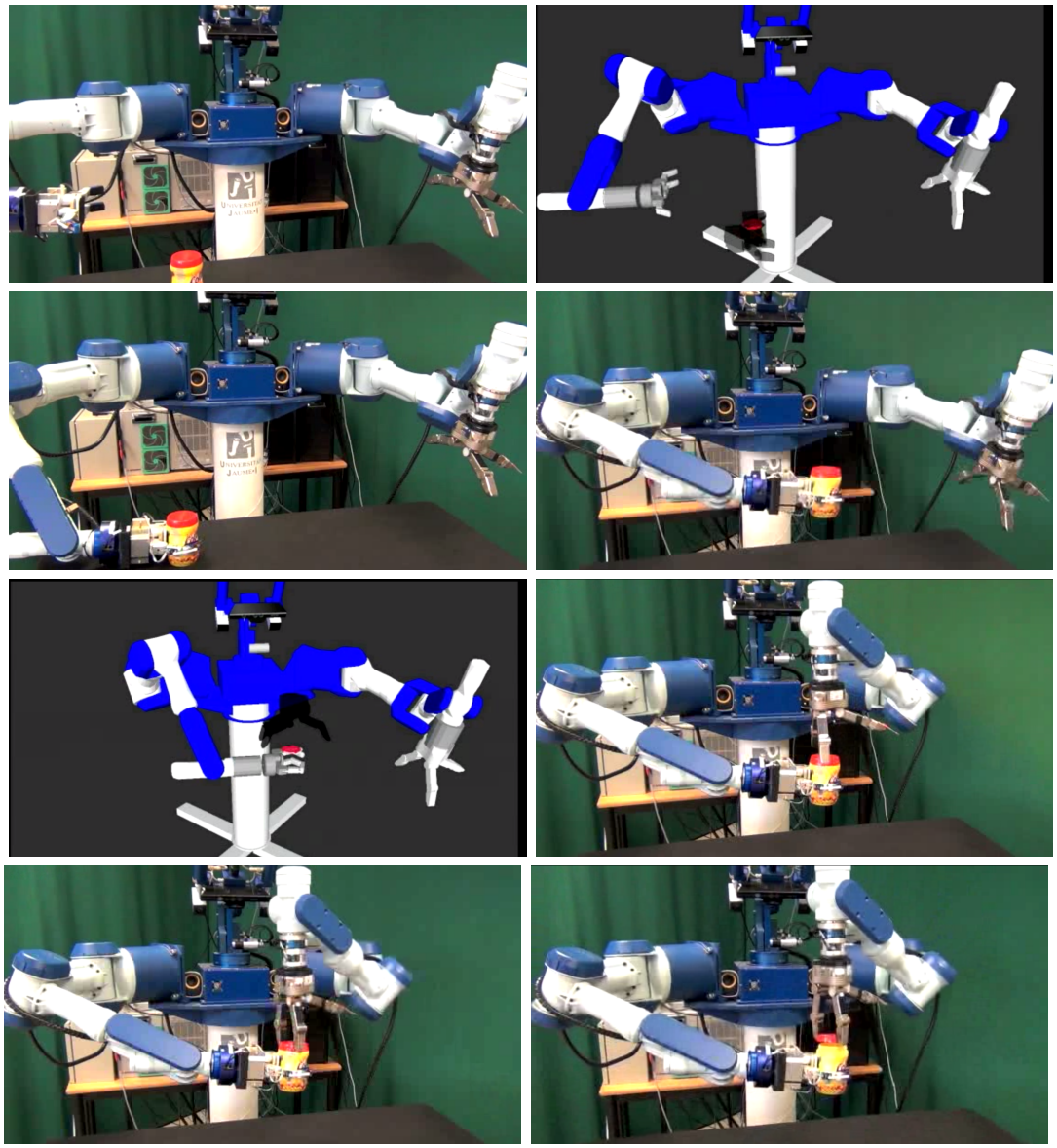


Figure 3.19: Unscrew task execution sequence.

Parameter settings

Like for the other primitives, it is recommended to set the parameters of the primitive depending on the object and the environment, nevertheless the default parameters have been used over the object test set with good results. Adjusting the parameters to each object may improve the performance of the primitive.

Force-torque limit parameters were empirically obtained for our robotic platform. The sensor readings were monitored during several executions and the parameters were set manually.

Grasp preshape and grasping fingers were determined by the task and robot embodiment. These parameters are designed for future improvements, for example to allow selecting the grasp preshape and the grasping fingers according to the cap geometry.

Distance parameters were selected regarding the smallest cap diameter possible (FINGER_MIN_DISTANCE) and the maximum cap height (CAP_REMOVED_DISTANCE).

Results and discussion

The overall results are shown in Table 3.4. After 80 task executions for each primitive configuration (with and without corrections enabled), the success rate of both primitives is very high, almost 90%. Although the primitive with the reactive adaptation enabled has slightly better success rate, it cannot be considered meaningful from this results. The main reason seems to be the low error and noise in the estimation of the bottle cap center in our setup. To test other scenarios with less precision and more noise, 15 more grasps were performed on Object 2 adding some uniform random noise ($\pm 1.5cm$) to the approach vectors calculated to grasp the cap. Object 2 was selected for this extra tests because, as can be seen in Fig. 3.20, it is has been the most problematic object for the strategy. The results obtained after 15 task executions with both primitive configurations is shown in Table 3.5. In this case, when the approach vectors are less precise, is where the robustness of the reactive strategy arises.

Fig. 3.20 depicts the success rate of the whole task for each object in the test bench. Meanwhile Fig. 3.21 shows the cap unscrewed rate. In some cases the cap unscrewed rate is higher than the task success rate, this is because although the cap was unscrewed, the robot failed to grasp it correctly and move it away from the object. Regarding the elapsed time required to perform the task, Fig. 3.22 shows that, for almost all the objects, the reactive strategy requires a bit more time. As expected, performing corrections takes more time than moving back always to the same position.

On the other hand the same twist movements by both primitives could be expected because both of them turn the cap the same TWIST_ANGLE degrees, but as shown in Tables 3.4 and 3.5 the reactive primitive requires more twist movements to open the object.

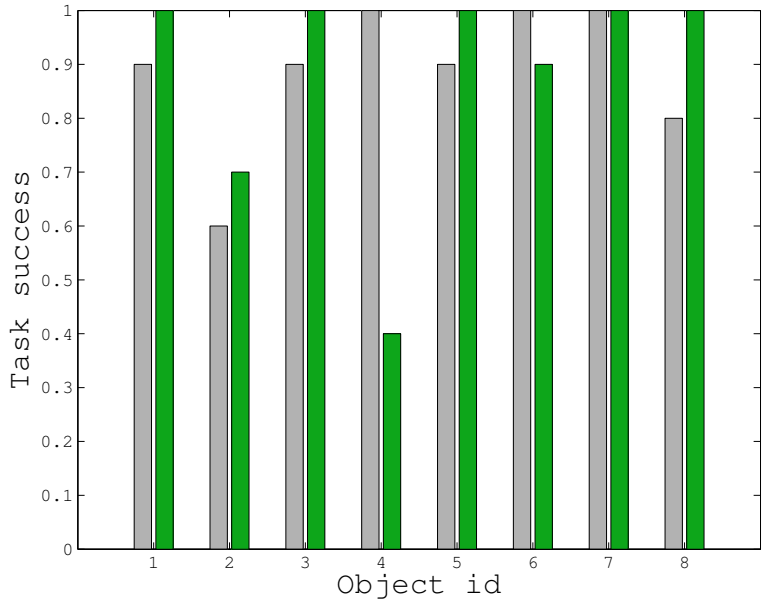


Figure 3.20: Grey: Task success rate for each test object after performing the unscrew task 10 times per object with the reactive adaptation enabled. Green: Reactive adaptation disabled.

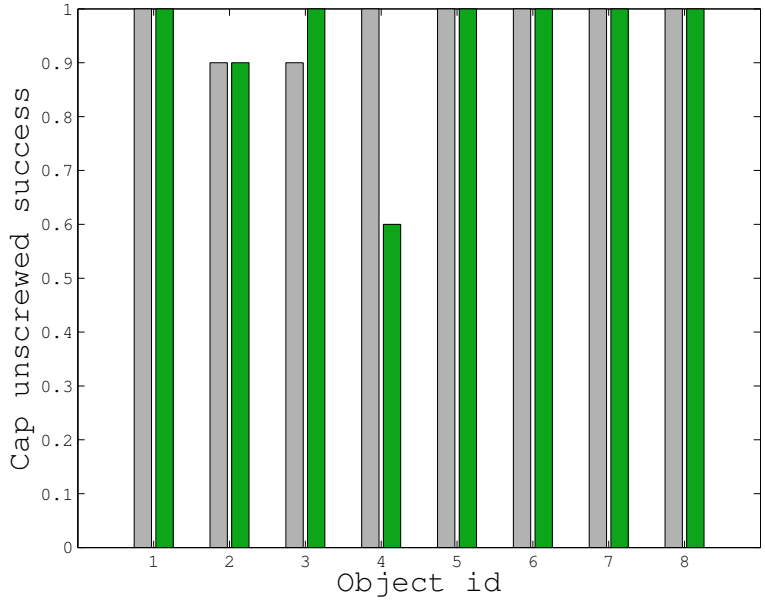


Figure 3.21: Grey: Cap unscrewed rate for each test object after performing the unscrew task 10 times per object with the reactive adaptation enabled. Green: Reactive adaptation disabled.

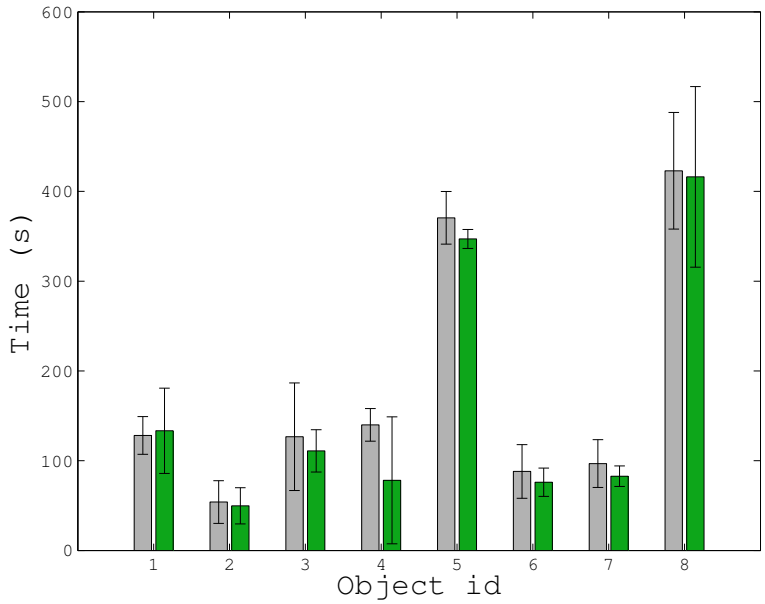


Figure 3.22: Grey: Unscrew task elapsed time for each test object after performing the unscrew task 10 times per object with reactive adaptation enabled. Green: Reactive adaptation disabled.

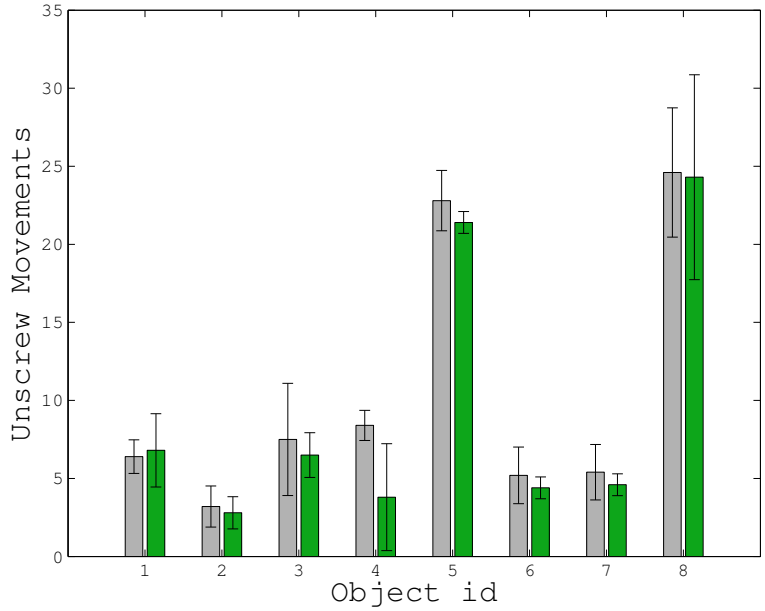


Figure 3.23: Grey: Twist movements performed for each test object after performing the unscrew task 10 times per object with reactive adaptation enabled. Green: Reactive adaptation disabled.

Reactive	Success rate	Cap unscrewed rate	Time	Moves
No	70/80 (87.5%)	75/80 (93.8%)	156.47	8.98
Yes	71/80 (88.6%)	78/80 (97.5%)	172.42	10.06

Table 3.4: Averaged results after 80 executions of each primitive

Reactive	Success rate	Cap unscrewed rate	Time	Moves
No	9/15 (60.0%)	10/15 (66.7%)	57.5	3.4
Yes	12/15 (80.0%)	13/15 (86.7%)	66.9	3.77

Table 3.5: Averaged results after 15 executions of each primitive over Object 2 with extra noise

Thus the difference in the elapsed time is caused by the highest amount of unscrew movements performed by the reactive primitive. The time taken (Fig. 3.22) and the number of movements performed (Fig. 3.23) are highly correlated. This would be expected if each unscrew movement took the same time, but as depicted in Fig. 3.11 the twist movement can be finished by other conditions, however the high correlation of time and twist movements suggests that the twist movement always ends with the TWIST_ANGLE condition instead of UNSCREW_TORQUE_LIMIT or FINGER_MIN_DISTANCE.

3.4.3 Emptying a box: Execution of a complex task

The purpose of this experiment is to demonstrate that the manipulation primitives paradigm is valid for describing and executing complex sensor-based tasks. To do so, we choose the task of emptying a box with no previous information about the number, identity, location and pose of the objects inside.

The task, depicted in Figure 3.24, is described using the manipulation primitives presented in Section 3.2 and a task specific perceptual primitive. The primitives are laid out in a loop that consists of pick and place sub-tasks that are repeated until there are no objects left in the box.

Three different methods were implemented for the approach vector generation and their impact in the task performance was measured.

Experimental setup

In order to validate the task implementation we carried out a total of 30 experiments of emptying a box filled with five unknown objects (see Fig. 3.26(b)). As can be seen in Figure 3.24, a key part of the task execution is the generation of the initial approach vectors. Three strategies were implemented: *random blind*, *blind exploration* and a *vision-based* method. 10 experiments were performed for each approach vector generation method.

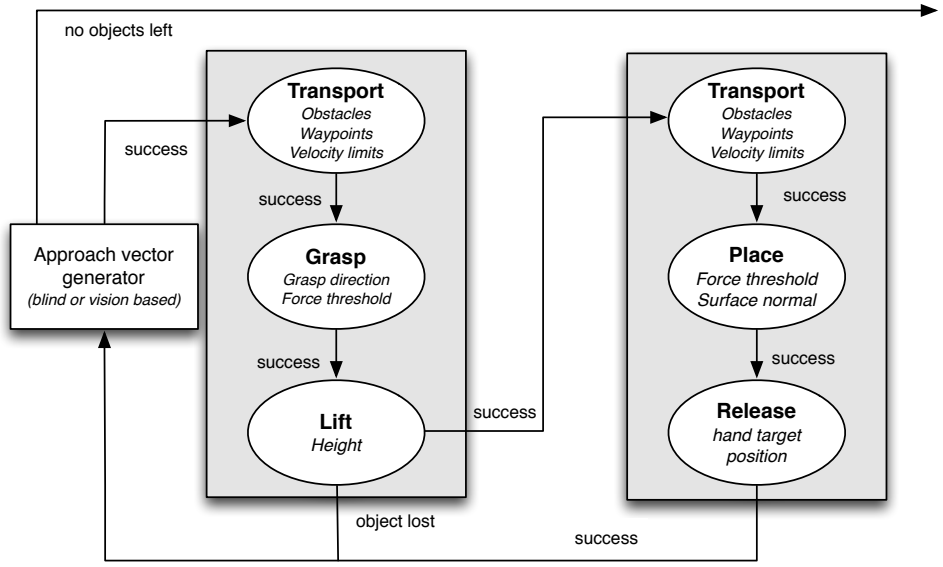


Figure 3.24: Task definition for a pick and place task. Primitives are represented by circles. Perceptual primitives are depicted using white boxes. Grey boxes group primitives into sub-tasks. Inside each primitive, some examples of parameters are written in italics.

To compare the performance of using different approach vector generators, different metrics were used:

- Task success: If all the objects from the box are removed, the task execution is considered successful.
- Grasp attempts: Every time the robot tries to grasp an object from the box, the grasp might not be successful and the robot can fail to grasp the object. In that case several attempts may be needed to grasp an object.
- Time: Time taken to empty the box.

Environment

The experimental environment consists of the robot in front of a table. On the table there is a box full of objects that can be in any position, even stacked (see Fig. 3.26(b)).

Test objects

There are no assumptions about the shape, material or texture of the objects. However, the objects play an important role on the performance of the grasping strategies, hence it is important to have the same objects for all the tests even if they are not exactly in

the same position for all the tests. For the experiment, we have selected five household objects with different primitive and compound shapes, see Fig. 3.26(b).

Assumptions

Objects' pose and geometry are unknown. The pose and size of the box that contains the objects are known. 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 limits are 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.

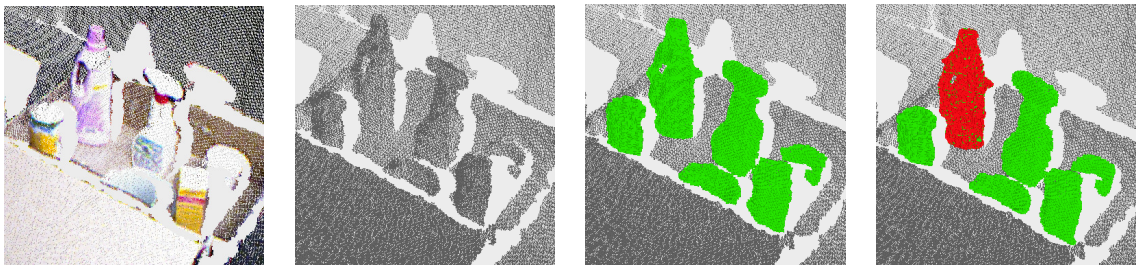
No further assumptions on the objects are made, except for the visual based approach vector generator. It requires the material of the objects to be visible by the active RGBD camera.

Parameter settings

The required parameters are: the starting approach vector to a target object and the target position to place it. The approach vector for the grasp primitive is generated by the approach vector generator, on the other hand the target position to perform the place sub-task is manually set to a pose outside the box. Due to the reactive implementation of the place primitive, objects can be placed on top of each other, the primitive will detect the contact and place them smoothly.

Approach vector generation

- *Random blind* strategy: top-grasp approach vectors are generated uniformly at random inside the known location of the box. In this case, there are no means to determine the number of objects left in the box, instead of using a large timeout, to save time, the end of the whole process is determined by a human supervisor.
- *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 box 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. 3.26(a). The task ends after completing an exploration trajectory without finding an object.
- *Vision-based* strategy: the Kinect sensor is used in the same fashion as in Sec 3.4.1 and 3.4.2. Objects are segmented from the environment by a pass-trough filter using the known box boundaries and clustered as shown in Fig. 3.25. 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.



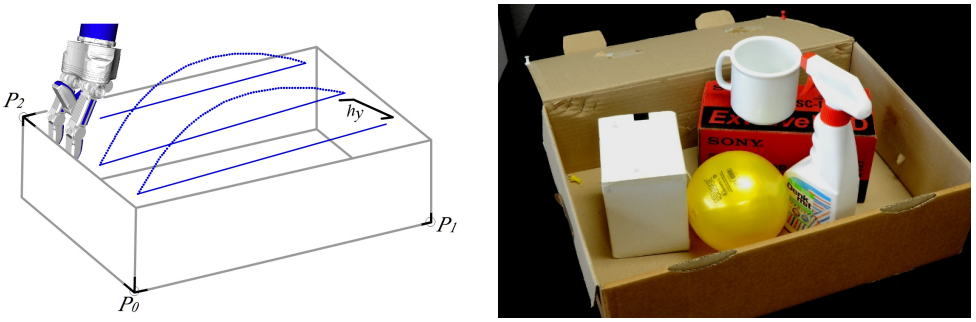
(a) Original 3D image.

(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 3.25: 3D point cloud segmentation phases for the visual-based approach vector generation.



(a) Hand preshape for exploration and exploration trajectory.

(b) A possible object layout

Figure 3.26: Exploration trajectory and object layout.

Method	Attempts	Time	Task success
Random	30.5	284.1 s	100%
Exploration	32.8	334.8 s	100%
Vision	7.1	101.4 s	100%

Table 3.6: Average results for each method after 10 task executions.

Results and discussion

Table 3.6 shows the averaged results for each approach vector generation method. All the methods were able to empty the box successfully 10 times out of 10. Regarding the number of attempts and time consumed, the vision based method outperforms the other two methods. However, the interesting result is that the blind methods were also able to complete the task successfully every time. In addition, blind methods are more general as they do not require any assumption about object material reflectivity properties.

Surprisingly, the exploration method requires almost the same attempts than the random method, this result is related with the density of objects. The impact of object density is depicted in Fig. 3.27 where the average number of required attempts to grasp one object, depending on the number of objects remaining in the box is shown. When there is only one object left (low object density) the random method requires way more attempts. As expected, the exploration method takes more time (for a similar number of attempts) than the random method because it has to perform the exploration trajectory for every attempt.

It is important to note that the reactive grasping primitive plays a crucial role because the generated approach vectors are quite inaccurate, especially in the blind approaches.

3.5 Conclusion

This chapter presented the manipulation primitives framework. Starting from the idea of action-phase controllers, provided from neuroscience studies, we have implemented several basic action-phase controllers to form a vocabulary of basic actions that has been used to define more complex tasks.

As the studies detailed in Chapter 2, corrective actions are also present in the human grasping process, following this idea, we have extended the manipulation primitives paradigm with reactive capabilities, implemented them and validated their importance.

From a practical point of view, the main contributions are threefold. 1) A robust reactive grasp primitive was presented. Experiments verified that the reactive control was able to recover successfully from significant planning errors. 2) An unscrew primitive was also implemented and verified as well supporting the reactive control approach already shown with the grasp primitive. 3) It was shown that the combination of sev-

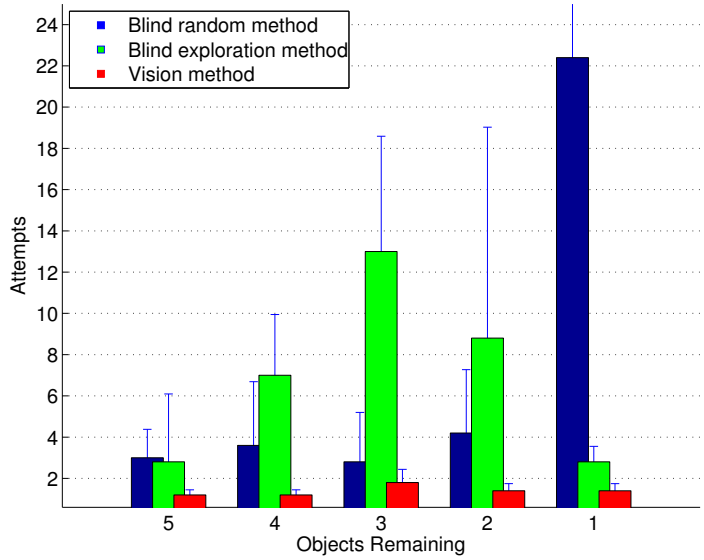


Figure 3.27: Average and standard deviation of required attempts depending on the number of objects remaining. The standard deviation for the blind random method when there is only one object left is truncated in the picture, its value is 39.32

eral 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 improve 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 results support the paradigm based on reactive manipulation primitives as a good way to generate and execute plans in unstructured and uncertain scenarios. The manipulation primitive approach is also suitable as an abstraction layer to provide a way to share plans, and more generally, knowledge, between different embodiments. 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. In Chapter 5 the available simulation tools and the implementation of dynamic simulation for robotic manipulation is presented and discussed.

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 author firm belief that the inclusion of reactive capabilities is essential in coping with the whole scope of complexity present in the real world.

The research and experiments presented in this chapter have been previously published. The design of the robust grasp primitive was published in [Felip and Morales, 2009] and it was improved and presented together with the manipulation primitives paradigm in [Felip et al., 2012]. The empty the box experiments were presented in [Felip et al., 2013]. Finally the unscrew primitive and the related experiments were published in [Felip and Morales, 2014].