Simulation of the Underactuated Sake Robotics Gripper in V-REP and ROS

Bachelor-Thesis von Simon-Konstantin Thiem aus Bayreuth
Tag der Einreichung:

1. Gutachten: Dr. techn. Elmar Rueckert
2. Gutachten: Prof. Jan Peters, PhD
3. Gutachten:

TECHNISCHE UNIVERSITÄT DARMSTADT

Simulation of the Underactuated Sake Robotics Gripper in V-REP and ROS

Vorgelegte Bachelor-Thesis von Simon-Konstantin Thiem aus Bayreuth

1. Gutachten: Dr. techn. Elmar Rueckert
2. Gutachten: Prof. Jan Peters, PhD
3. Gutachten:

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 28. September 2017

(Simon-Konstantin Thiem)

Abstract

The Sake Robotics Gripper is a cheap, robust and versatile gripper that has not been simulated yet. It is an underactuated gripper and therefore needs a workaround to achieve realistic behavior in simulation. The model has to be able to interpret the exact same ROS messages the gripper receives. This paper proposes a reproduction of the Sake Robotics Gripper in V-REP. We analyze the tools provided by V-REP to develop an algorithm to achieve underactuation in simulation. This contributes to the kitchen cleaning task of the GOAL-Robots project with a platform to do machine learning on. All models are available open source. Our model can be used as a foundation for research in complex grasping and manipulation tasks with the Sake Robotics Gripper.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 713010 (GOAL-Robots) and No 640554 (SKILLS4ROBOTS).

Contents

1 Introduction

In the last years there has been a lot of progress in the creation of domestic robots. Even though they are able to perform certain tasks such as cleaning floors or mowing lawn, they still struggle with more complex manipulation and grasping tasks [1]. Various grippers are built based on the example of the human hand [2] [3]. In machine learning this results in complicated models due to a hand's complexity, limiting the learning results. [4].

To gain versatile grasping capabilities a simple gripper model is implemented. It is used for work on the kitchen cleaning task in the GOAL-Robots project autonomous robots [5]. We use the SAKE Robotics gripper (SRG) [6]. The gripper is attached to a Kuka LBR iiwa 14 R820 [7]. Compared to other state of the art grippers, the SRG is relatively cheap. This two-finger gripper is versatile since it encompasses objects and thus can grip thin as well as bulky items. Additionally it does not have fragile finger tips that can break. It is an underactuated gripper and has only a simple one-dimensional input. This enables the application of control policies with only few parameters.

In this work we provide an open source model of the SRG for the virtual robot experimentation platform V-REP (Coppelia Robotics GmbH, Zürich Switzerland) [8]. The model enables realistic simulations with physical contacts. We provide an interface for our simulation that requires the same inputs as the robot. This allows the use of the same code for learning algorithms on both the robot and the simulation. Even though the control signal of the actuator is one-dimensional, the gripper has four degrees of freedom. We propose an algorithm that tries to mimic the behavior of underactuation in V-REP. Also we show a workaround to handle V-REP's lack of touch sensors. The code to handle the underactuation is implemented in a child Script in V-REP. This way we can maintain the advantage of having a low dimensional input. Thus the only information that needs to be learned is how far the hand has to close. We then show how we compensate the lack of touch sensors with proximity sensors and how we handle their differences.

1.1 Related Work

There has been some research about the proper design of grippers. Asfour et al. present a domestic robot for research purposes that uses a five finger gripper. They propose to do offline grasp analysis for each object that is supposed to be grasped [9]. This results in a huge database. Using online learning the system is able to manipulate objects that have not been previously seen.
Prior to the study of Asfour et al. grippers have mainly been used and designed for industrial purposes. In most cases they were constructed for one specific task only. Lanni and Ceccarelli give a broad overview of two-handed grippers and then propose an objective function for the constrained optimization problem to develop grippers for specific tasks [10]. With machine learning tools grippers can be taught to grasp objects that are similar in size and shape without special hardware required. There is also research on gripping materials which are difficult to clench. Drigalski et al. introduce a 3D printable gripper that is able to hold textiles [11], whereas Menciassi et al. focus on grasping objects of size of 1 mm and smaller [12].
In the late 1970s, Hirose and Umetani proposed a soft gripper which has an adjustable amount of finger links that each result in a degree of freedom. Equally to the SRG an underactuated pull on a wire causes the two fingers/chains to move inwards. Due to the chains being able to have a arbitrary length, objects can be completely enveloped when grasped [13].
Being able to simulate a project ahead of its physical implementation can be used to proof a concept and collect data for machine learning tasks. For our work, we used V-REP. It has a kinematic, a dynamics and a path planning module. While the implementation of the kinematic and the path planing module execute movement exactly as programmed, they are not able to correctly detect objects and forces. The dynamics module on the other side makes use of one of the physics engines. Ivaldi, Peters, Padois and Nori did an analysis about the tools that are capable of dynamics simulations. While they point out the diversity of the tools, most of them rely on physics engines like Bullet or ODE that were originally designed for video games [14]. While games try to achieve a smooth user experience, their prior concern is on speed and less on accuracy. In robotics however, the main goal is representing the reality as close as possible.

This often results in a problem as modern physics engines can still be insufficient for manipulation tasks with physical contacts.

We chose to use V-REP as development environment because it is a user friendly, intuitive program, that has many features implemented, while maintaining an even performance with other state of the art robotics simulation tools like Gazebo [15].

1.2 Structure

In this paper we give an overview about the Data-flow of the Simulation. We discuss the problems that arose and propose a way to solve them. We explain the underactuation of the SRG and discuss how we used it in simulation. We present our contributions towards the GOAL-Robots project and conclude with some future work ideas.

2 The Underactuated Gripper

The SRG has a good size for gripping objects like bottles, packages and other everyday household items that are mostly used in the kitchen. It has a grasp width of 145 mm and can hold up to 5 kg when it can grasp while wrapping around an object or 2.5 kg when holding it with finger tips.

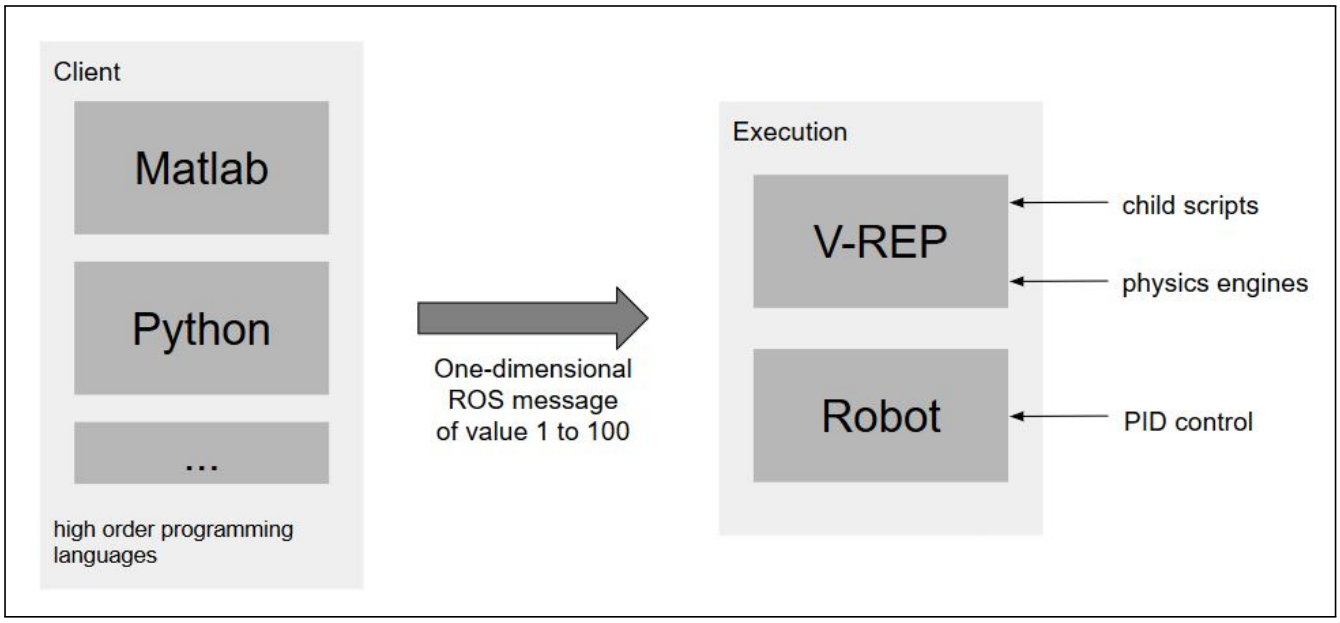

Figure 2.1: The Client is used to send Commands to the Robot or to the V-REP Simulation using ROS. For this the exact same Code in the Client can be used.

2.1 Data-flow

The task of learning how to grasp an object can be separated into three parts (Fig. 2.1). The learning algorithm, the Robot Operating System (ROS) and lastly the execution on the hardware or in simulations. For that purpose we use the Kuka LBR iiwa 14 R820 [7] with the SRG [6] and alternatively V-REP [8] for simulation. Any desired learning algorithm can be implemented in a high order programming language with a ROS interface. We decided to use Matlab for the purpose of rapid prototyping.

The learning algorithm has to send ROS messages containing information about the desired motion

of the robot. ROS is in charge of communication between Matlab and V-REP or the robot. Another series of messages deliver information for the motor which is in charge of moving the fingers of the SRG. These take a value between 1 and 100 that determines how far the SRG closes. The SRG is underactuated due to the signal for the grasp being only one-dimensional even though it has 4 degrees of freedom. This will be discussed in the section (2C). The interpretation of these messages take place in the Simulation Software V-REP by Coppelia Robotics [8].
A child script, written in Lua, can be attached to each simulated object. These control their actions and can be used to receive and interpret ROS messages. We wrote them in a way, that we are able to use the exact same ROS messages for the robot and the SRG. This is of use because it allows the use of the same interface for simulation and hardware respectively. Hence it is possible to generate learning data by simulations which then can be used for hardware applications.

2.2 Problems of Current Physics Engines

Several physics engines (Bullet 2.78, Bullet 2.83, ODE, Vortex, Newton) can be chosen in V-REP to simulate the movement of the SRG. We used the Open Dynamics Engine (ODE) [16] since it performs best for our task. There are three issues that determine the behavior in the simulation which are outlined below.
(i) The mesh files of the SRG were taken from the official Github page of SakeRobotics [17]. Due to the SRG being a non-convex shape, the mesh files for it are non-convex as well, which is the reason for a low cohesion of every two separate parts of the gripper. This results in the possibility of joints being unable to apply force towards the right directions, items being pushed away instead of being grasped and fingers of the gripper jumping back and forth. We address the first two problems in the Section (2C). Fortunately, the ODE physics engine is able to avoid fluctuation between non-convex shapes reasonably well.
(ii) V-REP does not have sensors implemented that recognize touch or contacts. A potential solution is the use of a built-in force sensor in V-REP, which however requires a rigid link between the SRG and the grasped object but this cannot slip at all. Alternatively proximity sensors can be used. These are able to detect if an object is close to the fingers. The distance threshold at

which they should signal an object contact can be tuned. However these values need to be greater than zero and therefore, the sensors will fire slightly before an actual touch will happen This will be discussed in Section (2D).
(iii) Current physics engines are not able to handle soft items. These would require a different behavior [18] [19]. Thus we are not able to simulate the grasp of items that can be squeezed.

2.3 Controlling the Gripper

The hardware link between the SRG and the robot has only a one-dimensional actuator. By default torsion springs hold the fingers in a 180 degrees straight position. To close the fingers the robot pulls on a wire exerting an opposing force against the springs, resulting in a contraction of the gripper. This command can be sent via a ROS message with a value between 0 and 100. The value determines how far the wire will be pulled. First the finger bases start moving until they hit a resistance. The wire is also connected to the finger tips. When the force cannot be emitted onto the base of the finger anymore, it will be transfered to an area with lower resistance. Here this is the finger tip. Once the tips of the finger touch each other or the object, force is applied towards the object. This way a firm grip surrounding the grasped object can be established (Fig. 2.2).

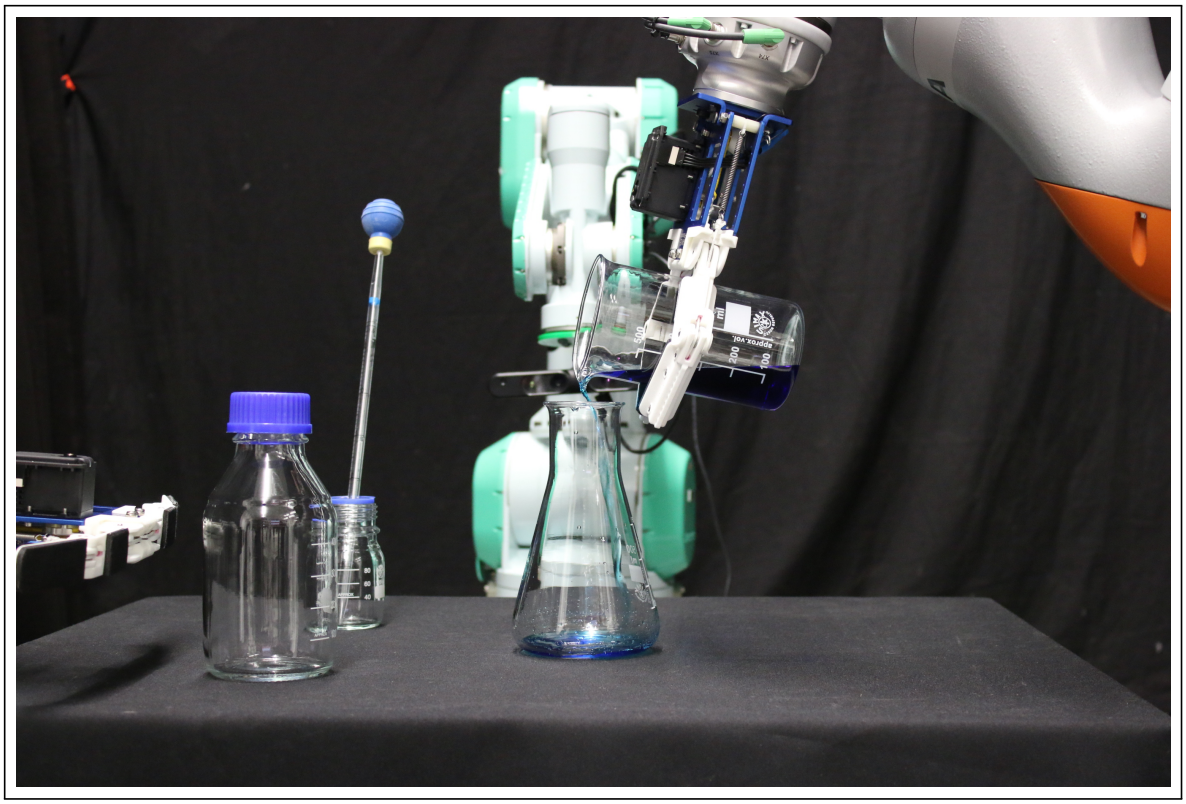

Figure 2.2: Picture of the SRG pouring a liquid.

The SRG uses a PID controller to administer the pulse width modulation that controls the motor. This is denoted by τ and calculated by a feed forward term $u_{FF}(t)$ and a feedback term $u_{FB}(t)$ of the torque u at the time t. The κ is the pulse width modulation limit. Let $q(t)$ be the minimum jerk, $\dot{q}(t)$ its velocity and $\ddot{q}(t)$ its acceleration. With $q^*(t) \in [0, 100]$ denoting the desired jerk, the pulse width modulation is defined by:

$$\tau(t) = \lfloor u_{FF}(t) + u_{FB}(t) \rfloor_\kappa \quad \text{with} \tag{2.1}$$

$$u_{FF}(t) = K_{FF_1}\dot{q}(t) + K_{FF_2}\ddot{q}(t) \quad \text{and} \tag{2.2}$$

$$u_{FB} = K_P(q^*(t) - q(t)) + K_I(q^*(t) - q(t))\dot{q}(t) + \frac{K_D(q^*(t) - q(t))}{\dot{q}(t)} \quad \text{resulting in} \tag{2.3}$$

$$\tau(t) = \lfloor K_{FF_1}\dot{q}(t) + K_{FF_2}\ddot{q}(t) + K_P(q^*(t) - q(t)) + K_I(q^*(t) + \frac{K_D(q^*(t) - q(t))}{\dot{q}(t)} \rfloor_\kappa \quad . \tag{2.4}$$

```
Input: posOfBase, posOfTip, proxSensorBase, proxSensorTip, newPos, currentPos
opening = false
if currentPos < newPos then
    opening == true
else if newPos < currentPos then
    opening == false
if opening then
    Once tip is completly opened:
    if posOfTip < 0 then
        open base:
        posOfBase ← newPos
        keep tip at 0:
        posOfTip ← 0
    else open tip
        posOfTip ← (newPos − posOfBase)
closing gripper while the base of the finger touches an object:
else if proxSensorBase then
    posOfTip ← (newPos − posOfBase)
    if proxSensorTip then
        applying pressure:
        posOfBase ← 0.5(newPos − posOfBase − posOfTip) + posOfBase
        posOfTip ← 0.5(newPos − posOfBase − posOfTip) + posOfTip
else
    posOfBase ← newPos
```

Algorithm 1: Simulated Underactuation for one finger in a single timestep.

2.4 Modeling

Our goal is to implement this underactuation with V-REP (after the ROS messages have been received). We receive a value between 0 and 100 and map it to all possible radiants of the finger, where 0 is open and 100 is closed. Our algorithm (Alg. 1) closes the finger until either the desired value is reached or one of the base proximity sensors fire. This happens slightly before they touch an object. Thus no force is yet applied towards the item and therefore it does not move. This way we vastly reduce the possibility that the item is pushed away by the fingers. Once the proximity sensor of the finger tip fires, the base finger starts grasping again, too. This happens when they either reach the object that is to be gripped or the other finger respectively. Now they apply force

to the object to establish a secure grip. This is done by closing the base fingers and the finger tips at the same time (Fig. 2.3).

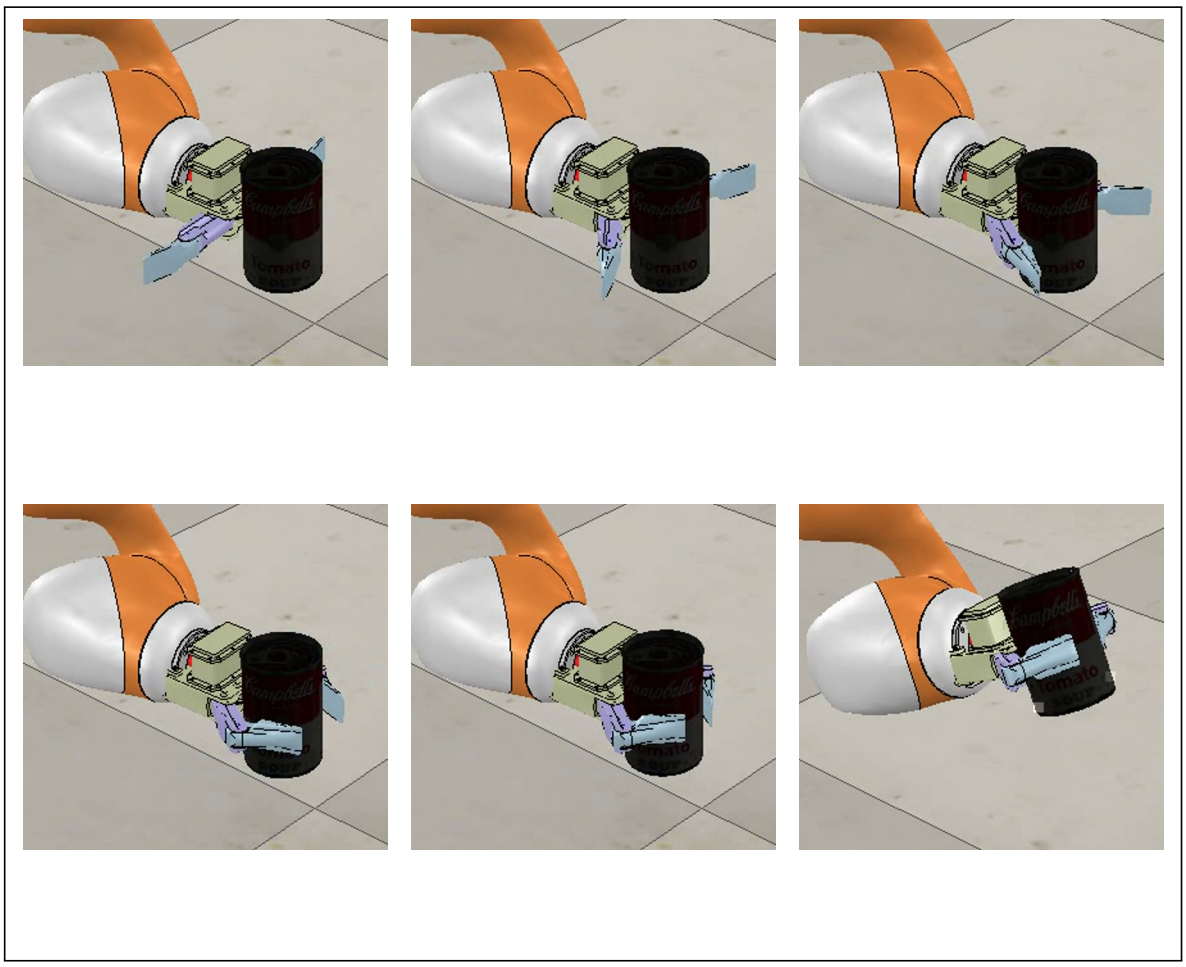

Figure 2.3: Sequence of the simulated gripper gripping the tomato soup can from the ycb object model set [20] [21].

2.5 Contribution to the Kitchen Cleaning Scenario

Being able to simulate the SRG gives us the opportunity to do reinforcement learning for the kitchen cleaning scenario in the third main objective: autonomous robots of the GOAL-Robots project. Apart from being cheap and robust, with his underactuation the SRG has the advantage of having a low dimensional grasp controller compared to other state of the art grippers.

The mesh models of the objects that are used in the kitchen cleaning scenario can be downloaded from the ycb-Website [20] [21]. These can be loaded into V-REP so that graping tasks can be performed on them (Fig. 2.3).

2.6 Future Work

Being able to simulate the SRG provides us with a chance to apply reinforcement learning to grasps in simulation. This however requires a reward function of how well a grasp has been performed. A number of such reward functions have been proposed [22] [23] [24] [25]. In future work we want to evaluate different reward functions for grasping using our SRG model.

3 Conclusion

The Sake Robotics Gripper (SRG) is a powerful gripper that can grasp numerous types of objects and supports loads up to 5 kg. To the best of my knowledge, no simulation of the SRG has been produced yet. Hereby we present an open source model in V-REP. We circumvent V-REP's lack of touch sensors and display a way to use proximity sensors for this purpose. To mimic the Sake Grippers behavior, we send a one dimensional ROS message to the simulation. This is the same message the actual robot takes. This way we provide an interface between a high level programming language like Matlab or Python to V-REP. In V-REP we are able to interpret the message and let the model perform grasps. We present an algorithm that uses the one dimensional message and creates an imitation of the SRG's underactuation. The code is available here: https://git.ias.informatik.tu-darmstadt.de/thiem/EZGripper_vrep_model.

4 Code

Child script used to control the EZ Gripper:

```
function setPosition(msg)
        -- can be used if a static position of the fingers is desired
        simSetJointTargetPosition(jl, msg.data[1])
        simSetJointTargetPosition(jr, msg.data[2])
        simSetJointTargetPosition(jlt, msg.data[3])
        simSetJointTargetPosition(jrt, msg.data[4])
end

function grasping_switch(msg)
        if (msg.data) then
                grasping = 1
        else
                grasping = 0
        end
end

function grasp(msg)
        max_grip = 3
        grip_strength = msg.data
        new_position = max_grip*grip_strength/100
        if(grip_strength < current_grasp)then
                opening = true
        elseif (grip_strength > current_grasp) then
                opening = false
        end
        current_grasp = grip_strength
```

```
        single_side_grip_control(jl, jlt, plb, plt, new_position, opening)
        single_side_grip_control(jr, jrt, prb, prt, new_position, opening)
end

function single_side_grip_control(handle_joint_base, handle_joint_tip,
 handle_prox_base, handle_prox_tip, newposition, opening)

        -- closing the gripper while the base touches an object
        if((not opening) and
         simCheckProximitySensor(handle_prox_base,sim_handle_all) == 1) then
                simSetJointTargetPosition(handle_joint_tip,
                 (newposition - simGetJointPosition(handle_joint_base)))
        if(simCheckProximitySensor(handle_prox_tip, sim_handle_all) == 1) then
                -- applying pressure
                simSetJointTargetPosition(handle_joint_base,
                 0.5*(newposition- simGetJointPosition(handle_joint_base) -
                 simGetJointPosition(handle_joint_tip)) +
                 simGetJointPosition(handle_joint_base))
                simSetJointTargetPosition(handle_joint_tip,
                 0.5*(newposition- simGetJointPosition(handle_joint_base) -
                 simGetJointPosition(handle_joint_tip)) +
                 simGetJointPosition(handle_joint_tip))
                end
        -- opening the gripper while the base touches.
        elseif (opening) then
                -- open base if tip is open
                if (simGetJointPosition(handle_joint_tip) <= 0) then
                       simSetJointTargetPosition(handle_joint_base, newposition)
                        --make sure tip stays at 0.
                        simSetJointTargetPosition(handle_joint_tip, 0)
                -- otherwise open tip
                else
                        simSetJointTargetPosition(handle_joint_tip,
                         (newposition - simGetJointPosition(handle_joint_base)))
```

```
                end
        else
                simSetJointTargetPosition(handle_joint_base, newposition)
        end
end

 if (sim_call_type==sim_childscriptcall_initialization) then
        current_grasp = 0
        opening = false
        grasping = 0

        jl = simGetObjectHandle('Left_Base')
        jr = simGetObjectHandle('Right_Base')
        jlt = simGetObjectHandle('Left_Tip')
        jrt = simGetObjectHandle('Right_Tip')

        plb = simGetObjectHandle('ProxS_LB')
        plt = simGetObjectHandle('ProxS_LT')
        prb = simGetObjectHandle('ProxS_RB')
        prt = simGetObjectHandle('ProxS_RT')

        simSetJointTargetVelocity(jl,0)
        simSetJointTargetVelocity(jr, 0)
        simSetJointTargetVelocity(jlt, 0)
        simSetJointTargetVelocity(jrt, 0)

        simSetJointForce(jl, 100)
        simSetJointForce(jr, 100)
        simSetJointForce(jlt, 100)
        simSetJointForce(jrt, 100)

        --grasp_handle = simExtRosInterface_subscribe('/startgrasp',
         'std_msgs/Bool', 'grasping_switch')
        gripper_state_handle = simExtRosInterface_subscribe('/position_grasp',
```

```
        'std_msgs/Float64MultiArray', 'setPosition')
        gripper_one_dim_signal = simExtRosInterface_subscribe('/grasp',
        'std_msgs/Int32','grasp')
end

if (sim_call_type==sim_childscriptcall_actuation) then
end

if (sim_call_type==sim_childscriptcall_sensing) then
end

if (sim_call_type==sim_childscriptcall_cleanup) then
        --simExtRosInterface_shutdownSubscriber(grasp_handle)
        simExtRosInterface_shutdownSubscriber(gripper_state_handle)
        simExtRosInterface_shutdownSubscriber(gripper_one_dim_signal)
end
```

Child script used to control the Kuka LBR iiwa 14 R820:

```
function setKukaPosition(msg)
        -- Set-up some of the RML vectors
         --(from the original threaded child script in V-REP)
        vel=110
        accel=40
        jerk=80
        currentVel={0,0,0,0,0,0,0}
        currentAccel={0,0,0,0,0,0,0}
        maxVel={vel*math.pi/180,
               vel*math.pi/180,
               vel*math.pi/180,
               vel*math.pi/180,
               vel*math.pi/180,
               vel*math.pi/180,
               vel*math.pi/180}
        maxAccel={accel*math.pi/180,
               accel*math.pi/180,
               accel*math.pi/180,
               accel*math.pi/180,
               accel*math.pi/180,
               accel*math.pi/180,
               accel*math.pi/180}
        maxJerk={jerk*math.pi/180,
               jerk*math.pi/180,
               jerk*math.pi/180,
               jerk*math.pi/180,
               jerk*math.pi/180,
               jerk*math.pi/180,
               jerk*math.pi/180}
        targetVel={0,0,0,0,0,0,0}

        target_position = msg.data
```

```
        for i= 1, 7 do
                simSetJointTargetPosition(jointHandles[i], msg.data[i])
        end
end

if (sim_call_type==sim_childscriptcall_initialization) then
        jointHandles={-1,-1,-1,-1,-1,-1,-1}
        for i=1,7,1 do
                jointHandles[i]=simGetObjectHandle('LBR_iiwa_14_R820_joint'..i)
                simSetJointForce(jointHandles[i], 500)
        end
        gripper_state_handle = simExtRosInterface_subscribe('/kuka', 'std_msgs/Float64MultiA
end


if (sim_call_type==sim_childscriptcall_actuation) then

end

if (sim_call_type==sim_childscriptcall_sensing) then

end

if (sim_call_type==sim_childscriptcall_cleanup) then

end
```

Matlab Code used to sent the messages through ROS to V-REP:

```
while(true)
    startpos()
    movetocup()
    grip()
    lift()
    placeposition
    startpos()
    movetocup()
    open_full()
    secure_start_position()
end



function startpos()

kukamove = rospublisher('/kuka', 'std_msgs/Float64MultiArray')
    for i=1:500
        msg = rosmessage(kukamove)
        msg.Data =[0 0 0 0 0 0 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
    end
end

function movetocup()
kukamove = rospublisher('/kuka', 'std_msgs/Float64MultiArray')
```

```
    for i=1:300
        msg = rosmessage(kukamove)
        msg.Data = [5*pi/180 -71*pi/180 0 110*pi/180 0 90*pi/180 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
    end
        for i=1:100
        msg = rosmessage(kukamove)
        msg.Data = [5*pi/180 -75*pi/180 0 85*pi/180 0 70*pi/180 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
        end
        for i=1:200
        msg = rosmessage(kukamove)
        msg.Data = [5*pi/180 -78*pi/180 0 80*pi/180 0 66*pi/180 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
        end

end
function secure_start_position()
kukamove = rospublisher('/kuka', 'std_msgs/Float64MultiArray')

    for i=1:100
        msg = rosmessage(kukamove)
```

```
        msg.Data = [5*pi/180 -75*pi/180 0 85*pi/180 0 70*pi/180 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
        end
    for i=1:300
        msg = rosmessage(kukamove)
        msg.Data = [5*pi/180 -71*pi/180 0 110*pi/180 0 90*pi/180 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
    end
        for i=1:50
        msg = rosmessage(kukamove)
        msg.Data = [0 0 0 0 0 0 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
    end



end


function placeposition()
kukamove = rospublisher('/kuka', 'std_msgs/Float64MultiArray')
     for i=1:1000
        msg = rosmessage(kukamove)
        msg.Data = [70*pi/180 -55*pi/180 0 70*pi/180 0 -55*pi/180 0];
```

```
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
    end

end

function lift()
kukamove = rospublisher('/kuka', 'std_msgs/Float64MultiArray')
    for i=1:100
        msg = rosmessage(kukamove)
        msg.Data = [3*pi/180 -40*pi/180 0 70*pi/180 0 -20*pi/180 0];
        msg.showdetails
        send(kukamove, msg)
        disp(msg.showdetails)
    end
end

function closehalf()
pub_position_grasp = rospublisher('/grasp', 'std_msgs/Int32')

    for i=0:500
        msg = rosmessage(pub_position_grasp)
        msg.Data = floor(i/10)
        msg.showdetails
        send(pub_position_grasp, msg)
        disp(msg.showdetails)
    end
```

```
end
function openhalf()
pub_position_grasp = rospublisher('/grasp', 'std_msgs/Int32')

    for i=1000:-1:500
        msg = rosmessage(pub_position_grasp)
        msg.Data = floor(i/10)
        msg.showdetails
        send(pub_position_grasp, msg)
        disp(msg.showdetails)
    end
end
function open_full()
pub_position_grasp = rospublisher('/grasp', 'std_msgs/Int32')

    for i=1000:-1:0
        msg = rosmessage(pub_position_grasp)
        msg.Data = floor(i/10)
        msg.showdetails
        send(pub_position_grasp, msg)
        disp(msg.showdetails)
    end
end

function grip()
pub_position_grasp = rospublisher('/grasp', 'std_msgs/Int32')

    for i=0:1000
```

```
        msg = rosmessage(pub_position_grasp)
        msg.Data = floor(i/10)
        msg.showdetails
        send(pub_position_grasp, msg)
        disp(msg.showdetails)
    end
end
```

Bibliography

[1] H. Van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," in Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on, pp. 121–127, IEEE, 2015.

[2] H. Liu, K. Wu, P. Meusel, N. Seitz, G. Hirzinger, M. Jin, Y. Liu, S. Fan, T. Lan, and Z. Chen, "Multisensory five-finger dexterous hand: The dlr/hit hand ii," in Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pp. 3692–3697, IEEE, 2008.

[3] S. Ekvall and D. Kragic, "Interactive grasp learning based on human demonstration," in Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, vol. 4, pp. 3519–3524, IEEE, 2004.

[4] T. Osa, J. Peters, and G. Neumann, "Experiments with hierarchical reinforcement learning of multiple grasping policies," in International Symposium on Experimental Robotics, pp. 160–172, Springer, 2016.

[5] "Goal robots." http://www.goal-robots.eu/project/third-main-objective-project". Accessed: 2017-09-19.

[6] "Sake robotics." http://sakerobotics.com/products2/. Accessed: 2017-09-18.

[7] "Kuka lbr iiwa 14 r820." https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/industrieroboter/lbr-iiwa. Accessed: 2017-09-19.

[8] "Coppelia robotics." http://www.coppeliarobotics.com. Accessed: 2017-09-19.

[9] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schroeder, and R. Dillmann, "Toward humanoid manipulation in human-centred environments," Robotics and Autonomous Systems, vol. 56, no. 1, pp. 54–65, 2008.

[10] C. Lanni and M. Ceccarelli, “An optimization problem algorithm for kinematic design of mechanisms for two-finger grippers,” Open Mechanical Engineering Journal, vol. 3, pp. 49–62, 2009.

[11] F. von Drigalski, D. Yoshioka, W. Yamazaki, S.-G. Cho, M. Gall, P. M. U. Eljuri, V. Hoerig, J. Beltran, M. Ding, J. Takamatsu, et al., “A versatile, open-source two-finger gripper for textile manipulation,”

[12] A. Menciassi, A. Eisinberg, I. Izzo, and P. Dario, “From" macro" to" micro" manipulation: models and experiments,” IEEE/ASME Transactions on mechatronics, vol. 9, no. 2, pp. 311–320, 2004.

[13] S. Hirose and Y. Umetani, “The development of soft gripper for the versatile robot hand,” Mechanism and machine theory, vol. 13, no. 3, pp. 351–359, 1978.

[14] S. Ivaldi, J. Peters, V. Padois, and F. Nori, “Tools for simulating humanoid robot dynamics: a survey based on user feedback,” in Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on, pp. 842–849, IEEE, 2014.

[15] L. Nogueira, “Comparative analysis between gazebo and v-rep robotic simulators,” Seminario Interno de Cognicao Artificial-SICA 2014, p. 5, 2014.

[16] “Open dynamics engine.” http://www.ode.org. Accessed: 2017-09-19.

[17] “Github repository of sakerobotics containing the second generation of the ezgripper.” https://github.com/SAKErobotics/EZGripper. Accessed: 2017-09-19.

[18] D. J. Montana, “The kinematics of contact with compliance,” in Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on, pp. 770–774, IEEE, 1989.

[19] D. J. Montana, “Contact stability for two-fingered grasps,” IEEE Transactions on Robotics and Automation, vol. 8, no. 4, pp. 421–430, 1992.

[20] “Ycb benchmarks.” http://www.ycbbenchmarks.org. Accessed: 2017-09-19.

[21] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in Advanced Robotics (ICAR), 2015 International Conference on, pp. 510–517, IEEE, 2015.

[22] C. Ferrari and J. Canny, "Planning optimal grasps," in Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on, pp. 2290–2295, IEEE, 1992.

[23] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in AAAI, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[24] T. Baier-Lowenstein and J. Zhang, "Learning to grasp everyday objects using reinforcement-learning with automatic value cut-off," in Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, pp. 1551–1556, IEEE, 2007.

[25] T. Osa, A. M. G. Esfahani, R. Stolkin, R. Lioutikov, J. Peters, and G. Neumann, "Guiding trajectory optimization by demonstrated distributions," IEEE Robotics and Automation Letters, vol. 2, no. 2, pp. 819–826, 2017.