Exposé for B.Sc.-Thesis
Multitouch Robot Control

Merlin Steuer
Matr. 6415125
2steuer@informatik.uni-hamburg.de

Supervised by: Dr. Norman Hendrich, Dennis Krupke
Department of Computer Science
University of Hamburg

February 25, 2018

Contents

1 Introduction

1.1 Motivation

Controlling dexterous robot hands is a big challenge of robotics, but using has a variety of obvious advantages: The similarity to a human hand gives enables it to grasp objects in nearly all positions and poses the real human hand could. Especially for complex manipulation tasks, where a simple robotic grasper with just a pair of pliers is not sufficient, the larger amount of degrees-of-freedom comes into action. Also, users might be able to better plan actions when they are controlling a device similar to their own hands, meaning the main task for them is to use a control interface to execute actions they would otherwise execute with their own hands.

Creating such a simple and easy-to-learn interface to a dexterous robotic hand is the goal of this bachelor thesis. The device used is a *Shadow Dexterous Hand C5/C6* by the Shadow Robot Company. It has five fingers controlled by electrical or pneumatic muscles using 20 degrees-of-freedom[Com08]. The hand is connected to a robotic arm allowing it to also be moved in space.

As a control device an off-the-shelf android tablet will be used, as these devices have become very widespread and - thanks to this - relatively affordable. With screen sizes of 10 inches and above and the capability to record more than 5 independent touch pointers and a number of additional sensors (gyroscope, orientation, ...) and feedback actuators (vibration, sound, ...) they make a good choice for a versatile control device.

1.2 Outline of this exposé

This exposé will first describe the two main approaches that will be used and evaluated within the thesis, going on to describing the planned implementation in Section 3. At last the possible methods for testing and evaluating the results of the project will be shown.

2 Basic approaches

2.1 Direct mapping of finger positions

The first approach to control the dexterous hand maps finger positions on the touch screen directly to positions of the robotic fingers. This makes use of the fact that most human grasping operations can be broken down to controlling fingertips in a plane[Toh+12, p. 1]. The biggest advantage of this approach is the intuitive and independent movement of single fingers, allowing the user to execute the grasping operation he thinks fits most to the given scenario. However, some drawbacks have to be considered: The workspace is quite limited to the size of the touch screen. The authors of [Toh+12] circumvent this by allowing some kind of scroll-operations when moving the hand to borders of the screen. Moving the grasper up or down is implemented by tilting the device.

Another solution to extend the limited workspace would be to scale up motion actions from the size of the touch screen to the desired size of the workspace. This would eliminate the need of unintuitive scroll operations within

the workspace, but induce new problems dealing with inaccuracy due to small errors being scaled up by (in the worst case) several orders of magnitude.

2.2 Parametrized grasp actions

2.2.1 General approach

Another approach is to simplify the control of each single joint within the robotic hand by executing a principle component analysis on recorded movement data of selected grasping operations[AB12]. This parametrises the actions, making it easier to control a grasp as not every joint position has to be recalculated manually, but using mathematical operations to derive the joint positions from the desired grasping pose that shall be executed. While this method allows grasping operations that are not restricted to controlling the fingertips in a planar environment, the user has less control over the distinct positions of each finger. It has to be evaluated, if this is a problem within the real use case.

Grasping operations would be simplified to actions like *open hand* or *close hand*, with the selected grasping pose being executed. This would give the user the ability to use the other fingers or the other hand to control other degrees-of-freedom like the position of the hand in 3D space simultaneously. With this approach, the user interactions would probably not be as intuitive as within the one described in section 2.1. This might lead to a more steep learning curve.

2.2.2 Absolute vs. relative positioning

When using the parametrized grasp actions, a design decision can be made between using absolute control of the robotic arm or positioning the hand and controlling the grasp operations relatively to each other. When using the absolute approach, the touch actions on the touch screen have to be made at the exact same spot to reproduce the same result. This also means that e.g. a distinct distance between the fingers in a two-finger pinch-gesture means a distinct position of the grasp pose.

Example: Let a grasping pose be controlled by numbers from 0 (hand fully open) to 1 (grasping/hand fully closed). The program now maps distances of two touch points to the space between 0 and 1 like in table 1.

finger distance	pose state
$5cm$	0.0
$2.5cm$	0.5
$0cm$	1.0

Table 1: Finger distance to pose state mapping (ex.)

When lifting a finger off the touch screen accidentally it would be mandatory to place it at the exact same spot on the touch screen again. Otherwise, a new distance of the fingers will be measured and a new pose state would be set. This could, if not handled by the program in any form, lead to failure and damage to the device or other equipment. These points could be applied to all actions done with the robotic arm including movement of the arm in 3D space and thus require a safety system to be implemented like a dead-man-switch or an easy-to-reach emergency shut down switch, bringing all devices into a safe state.

Using a relative position control would at least minimize the risk of uncontrolled behaviour by accidentally moving fingers. In this approach, all actions are done relatively, meaning that where the fingers are placed on the touch

screen has no effect on the actions executed. Fingers could be safely placed on and lifted from the screen at any time, as the robot will then remain still within the state it currently is. A two-finger pinch gesture could still be used to open or close the robotic hand, but the actual distance of the touch points would not directly map to the hand's state, but the change in the finger distance would map to a change in the hand's state. This could also be introduced to the arm's position in space, hand rotation etc.

Using this, the speed at which actions are executed could be controlled more easily, improving the precision and dexterity of the operations. Also, the possible workspace would be much greater, as e.g. moving the hand around in space could be done in multiple swipe operations over the full screen. After placing the arm, the desired grasping operations could then be executed relative to the new position. It would be possible and sensible to introduce multiple speed modes for letting the user choose between fast and not very precise operation to roughly place the hand where it shall be and then switch to a slower mode, allowing the user to very precisely execute all desired operations.

2.3 A combination of the above?

If a combination of the above approaches seems handy it might also be implemented and evaluated.

3 Implementation

3.1 ROS - Robot operating system

The robotic arm and hand are both controlled by the robot operating system. ROS is a meta-operating-system designed to control robots in a very modular way. This means that e.g. every sensor, every actuator and every controller is an own so-called *node*. Communication between nodes is done using TCP/IP-sockets. A ROS-master program coordinates all communication connections between nodes, as these are peer-to-peer and not relayed by the master program.

Communication mainly takes place by subscribing to so-called *topics*, where other nodes might then *publish* data to. Data published by one node is then sent to all nodes that subscribed a specific topic. This approach decouples the senders and receivers, allowing to easily replace or add more nodes as the robot system grows.

Introducing a new controller to a robot system does only require to add a new node to the ROS master program publishing data to a specific topic, telling the other actuators controlled by other nodes what to do. For example, when one ROS node has subscribed to a topic where it expects data to position all joints of a robotic hand, the only thing to be added is a new node, publishing to this topic.

Development for ROS mainly takes place in C/C++ and python, but third-party libraries for other languages are available, such as Java. The quality of these third-party-libraries differs as ROS develops quite fast, so it has to be evaluated if the libraries may be used for the given ROS setup.

3.1.1 rosjava

rosjava is the Java implementation of ROS. It is designed to be run within Android applications and a stable version for ROS *Kinetic*, the one used at the current installation of the robotic arm and hand, exists.

Thankfully, the libraries do not have to be compiled from source code to be integrated into an Android application, as a repository with pre-compiled binaries is available.

3.1.2 rosbridge

If *rosjava* turns out to be unusable for any reason, *rosbridge* would be the second choice to communicate with a ROS master. It is an interface designed to be run in web applications that offers a WebSocket server and a JSON protocol to connect to and exchange data with a ROS master. Java libraries for rosbridge do exist, however, quick research turned out that it can only be used on Android by replacing the WebSocket library used with one compatible to the Android operating system.

3.2 Development environment

As we are planning to develop an Android application to be run on an Android tablet, the software *Android Studio* is the current proposed integrated development environment (IDE) to choose. It has an integrated dependency-management, code-completion and debugging capabilities to ease and support the overall development process. As a source code version control system (VCS), git will be used.

For testing purposes, a simple ROS installation will be set up with the development environment to quickly test small portions of the code. Everything will be wrapped up within a virtual machine running Ubuntu 16.04 to meet the setup used within the TAMS lab.

3.3 Android application

The Android application developed within this thesis shall be able to execute on devices running Android 4.0.3 and above. This makes sure it runs on nearly all Android tablets published within the last 5 years. This requirement should not be much of a problem, since the user interface will be very limited and the basic APIs for touch interaction within the Android system did not change very much over the time.

3.3.1 Software architecture

A big goal is to create software that may easily be reused in other following projects. To achieve this, a layered software architecture will be introduced. By the time of writing, the exact software architecture is not fixed, but the following layers will probably exist within the software:

- **User interface:** This is the lowest layer, as it represents the *View* of the Android application. It recognizes the touch events and displays information about the touch points. It passes the information of the touch events

on to the touch processing layer. It will be the same for all described approaches.

- **Touch point processing:** This layer is the place where the real work takes place like parsing touch gestures, calculating joint angles of the robot's joints and storing state information that might be needed to control the robot. It takes the touch events of the user interface layer as an input and outputs the joint angles and positions that shall be sent to the robot.
- **ROS-Communication layer:** This is the layer taking care of communication with the robot. It takes the joint angle and position information calculated within the touch point processors and passes them to the robot using ROS. No calculation takes place within this layer.

Using this kind of architecture makes it easy to implement multiple approaches and different touch processors without having to change (or even rewrite) the user interface or the communication layers when applying changes to the touch processing. It is even possible to exchange the touch processor at runtime by just moving around some objects within the application, as the communication between the layers will probably be implemented using some sort of the *observer-pattern*. This implementation will most likely turn out as some kind of a Model-View-Controller pattern as the software architecture gets more elaborated.

3.4 Graphical representation of robot state

As published in the announcement for this thesis, a graphical representation of the state of the robotic arm shall be implemented as well. It is to be discussed whether this representation takes place within the same application that controls the robotic arm or within another application (built around Unity3D or similar frameworks) on another device. Separating the graphical representation from the controlling software would give more place - and thus control surface - on the controlling device. Including it within the same application would reduce the amount of devices to be carried.

As users are more accurate and faster when having the direct visual feedback of the real robot system when controlling it[Toh+12], the graphical representation could e.g. be used for training purposes. It would be implemented as another node within the ROS master subscribing to the same control topics as the real robot, meaning it receives and processes the same information sent by the controller node. Using ROS comes in handy here as it is no effort to choose whether the graphical simulator or the real robot system shall be controlled. Even controlling both at the same time is possible.

4 Tests and evaluation

4.1 Testing during development

As the development proceeds a set of standardized tasks should be set up to measure performance objectively. Using these tasks a process could be measured over time and an overview could be made showing the development of the

controlling abilities. Different approaches could then also be compared easily by executing the same task with multiple methods of controlling the robot and comparing the results.

Possible metrics would be the time needed to execute a task, the number of retries until the task was successfully executed and if the grasped object was damaged or not.

4.2 Evaluating usability with untrained test persons

When the development of the software gets to an end it could be sensible to evaluate the user experience of untrained users using the software to control the robot. After a short instruction, standardized tasks should be performed to see how the different approaches compare in terms of usability, learning speed and how intuitive they are. The metrics measured would be the same as in Section 4.2.

References

[AB12] N. Hendrich A. Bernardino C. Gang. *Parameterizing and creating new actions.* Tech. rep. 2012. URL: https://tams.informatik.uni-hamburg.de/projects/handle/HANDLE_D24.pdf.

[Com08] Shadow Robot Company. *Shadow Dexterous Hand C5 Technical Specification.* 2008. URL: http://www.shadowrobot.com/downloads/shadow_dextrous_hand_technical_specification_C5.pdf.

[Toh+12] Yue Peng Toh et al. "Dexterous telemanipulation with a multi-touch interface." In: *Humanoids.* IEEE, 2012, pp. 270–277. URL: http://dblp.uni-trier.de/db/conf/humanoids/humanoids2012.html#TohHLBZP12.