Software design of an assistive robotic manipulator for versatile control authority in multi-action manipulation tasks

Breelyn Styler[1,2], Cheng-Shiu Chung [1,2], Alexander Houriet[1,2], Dan Ding [1,2]
[1]*HERL, VA Pittsburgh Healthcare System,* [2]*Department of Rehabilitation Science and Technology, University of Pittsburgh*

INTRODUCTION

Electric power wheelchair (EPW) users with upper limb impairments require assistance to participate in everyday activities. For those with arm impairments, even basic tasks like eating with a spoon or drinking from a cup are nearly impossible. The ability to reach and manipulate objects is consistently rated as one of the most important challenges [1]. There are few assistive manipulation options that allow for tasks requiring finer hand movements. In some cases, family or hired caregivers help with activities of daily living [2,3], but, ideally, users would be able to access tools and technology to empower them to engage independently in self-care tasks. Assistive Robotic Manipulators (ARMs) provide an option to allow EPW users to engage in such tasks [4], and provide a variety of functions compared to specialized low-tech manipulation tools [5]. However, currently available commercial wheelchair-mounted ARMs are cumbersome to control. In the same way EPWs require a joystick to navigate a wheelchair through a room, an ARM requires a joystick to move the end effector through the environment. Unlike an EPW, the ARM requires the actuation of six joints through a two axes joystick (forward/back, left/right) with many mode switches that toggle between orienting the wrist, extending/retracting the arm, and opening/closing the gripper to interact with the environment. This interaction quickly becomes cognitively difficult and increases operation time. Bhattacharjee et al interviewed an expert Kinova ARM user that stated manually controlling the arm "(fully non-autonomous) to pick one piece of fruit and bring it to his mouth would take him approximately 45 minutes" [6]. Work by Herlant et al describes how a low-dimensional joystick interface causes a high number of mode switches which consumes 17.4% of task time and increases Kinova ARM user's cognitive load [7]. Despite these difficulties, many users perceive the control challenges as worthwhile given that these devices also increase independence [3].

ARM control challenges can be alleviated, while maintaining independence benefits, through the addition of autonomous robotic software functions [8]. Where autonomy is best applied, and how the user switches their control with the autonomy system, is currently an active area of research [6,8]. This paper focuses on describing the software architecture that allows the designer to assign each task action to user control or robot control for multi-action manipulation tasks. We define this as an assignment of control authority where an agent (user teleoperation or robot software) is responsible for executing the current action. The ability to change the control authority assignment allows future investigation into a user's controllability preferences and creates a software technology that can adapt to a user's changing abilities, preferences, and contexts. This architecture facilitates the investigation of the appropriate discretization of control assignment between a user and assistive robot within a manipulation task.

SYSTEM OVERVIEW

The hardware system consists of a Kinova Gen3 ARM with wrist mounted camera, joystick, Jetson Xavier NVIDIA computer, and a touchscreen showing a graphical user interface (GUI) for transitioning between user and robot control. The software architecture uses the Robot Operating System (ROS) framework with the publish-subscribe messaging pattern. Each process in the system, referred to as a node, can publish and receive messages asynchronously. Our current system consists of six nodes (rectangle boxes in Figure 1). For this paper, we will focus on describing the *manipulation node* (Figure 1). The manipulation node executes the sequence of states and actions defined by the task's internal state machine.

HIERARCHICAL SOFTWARE DESIGN

Manipulation tasks are inherently hierarchical where precise manipulation movements combine to produce more functional movements (e.g., push, pull, reach) which then form sub-tasks that combine to complete a task. These sub-tasks are often completed in phases which gives them a sequential temporal element where postconditions are satisfied before transitioning to the next sub-task. For example, drinking from a glass first requires that a glass is retrieved and filled. We will refer to a sub-task as a *task sequence* (T_s) which consists of the state and action steps within a functional task. Multiple task sequences define a *main task* (T_m), where ($T_s \subseteq T_m$).

State and actions

Upon start of the manipulation node, the current *main task* is parsed from the configuration file (e.g., make popcorn, drink water) and instantiated. The executive layer loops through the *system states*. TELEOP-FREE is the starting system state outside the sequence of any task. This state allows the user to freely move the ARM and displays interaction objects seen by the ARM camera to the user through a touchscreen interface. If the user selects a highlighted interaction object, the executive transitions to AUTONOMY, TELEOP-IN-TASK, or USER-FEEDBACK which are the three system state types assigned to any state within a T_s. These three system states delineate control authority where AUTONOMY is robot software control, TELEOP-IN-TASK transitions control to the user and waits for touchscreen feedback for transitioning control back to the robot, and USER-FEEDBACK prompts for user information. All three of these states can transition between each other. At the completion of a task (or task sequence) the system state transitions to END which transitions back to TELEOP-FREE, a user-controlled state.

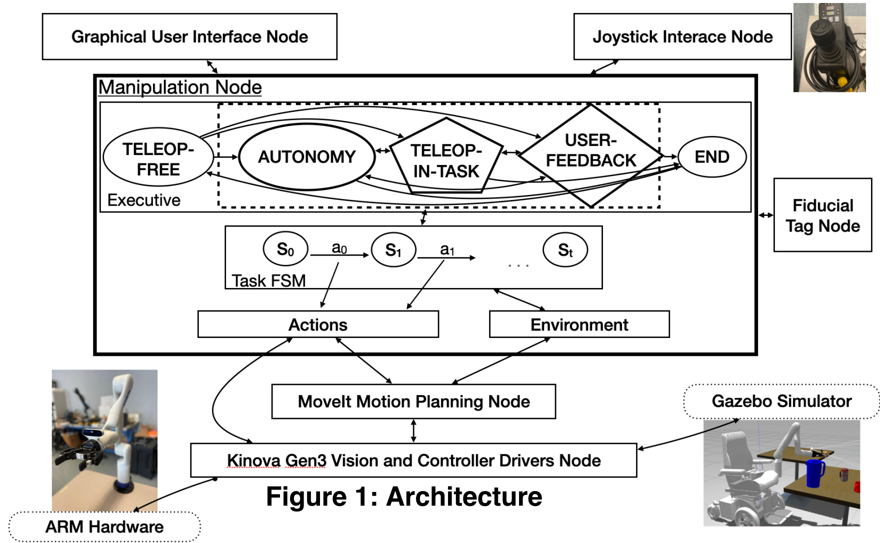

Figure 1: Architecture

A finite set of *task states* ($s \in S$) are within a *task sequence* T_s which is defined as $T_s: s_0 \overset{a_0}{\rightarrow} s_1 \ldots \overset{a_t}{\rightarrow} s_t$, such that s_0 is an initial state. Task states can generalize across tasks (e.g., start, to_grasp, has_object), or be task specific (e.g., start_drink, finish_drink). Every task state within a *task sequence* is assigned a system state type. Our goal is to continuously refine the states to discover commonalities which further reduce code complexity.

Actions ($a \in A$) are defined in an action library that are re-used modularly. The most primitive actions have four commands: 1) opening the gripper fully; 2) closing the gripper for some amount; 3) moving the ARM to a goal configuration that is collision free (via motion planning), and 4) moving the ARM to a goal configuration where there is no collision checking (direct cartesian movement). These four primitive actions create more general object interactions that combine to form task sequences. Figure 2 shows a legend with three possible T_s state types and four primitive actions that describe state-action sequences. For example, dispensing popcorn requires closing the gripper, motion planning to a collision free offset above the dispensing handle, and executing a cartesian action that moves the gripper down to dispense the popcorn kernels. This sequence can be labelled as a PUSH (Figure 4).

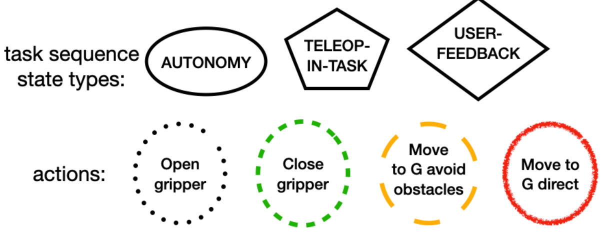

Figure 2: Legend for task sequences

For state transitions, we construct a finite state machine (FSM) with the state pattern software design. An abstract context task class defines environment obstacles, associated object interaction positions, and stores a reference to the starting task state. The task class uses polymorphism to transition between states. An abstract state class encapsulates state-specific actions and defines the next state for each of the current state's possible actions.

Object properties

Before the system starts, a designer sets each task sequence's associated environment and interaction objects in a configuration file. Environment object properties define obstacles in the world that are sent to the motion planner. Interaction object properties define positions and grasps which are parsed and commanded when performing actions. For example, picking up a cup is configured in a PICK sequence that defines a radius and height for the cup obstacle, and an offset position and grasp type for grasping the cup. Each object can be tied to a fiducial marker tag (QR-like code) placed in the environment that when identified by the ARM's camera can be selected by the user.

REPRESENTATIVE TASKS

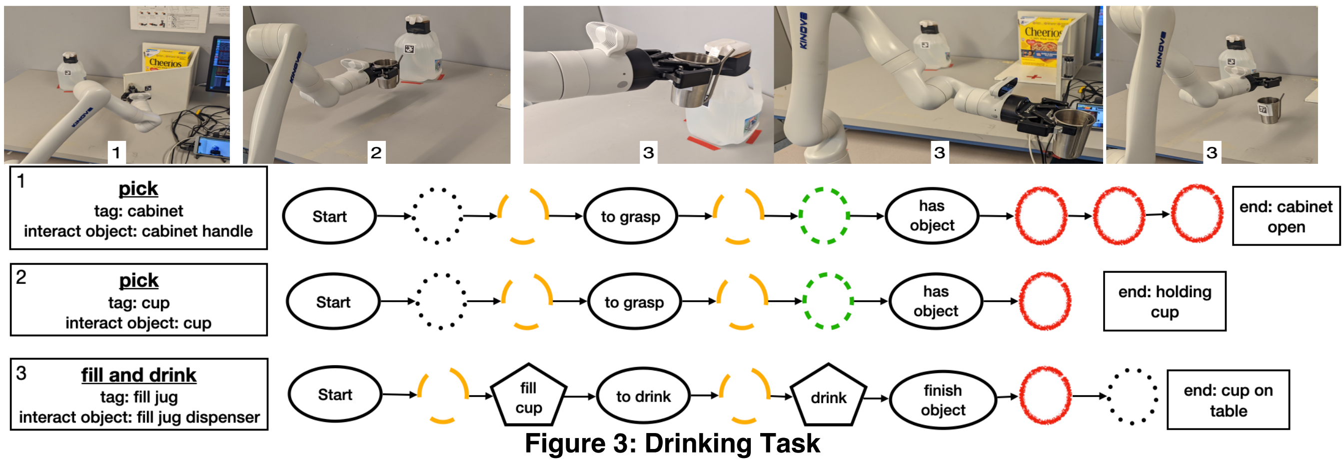

Figure 3: Drinking Task

Our first software iteration assigns user control to gross actions that move the gripper towards interaction objects (during TELEOP-FREE), and actions that require an unknown time duration (e.g., drinking water, filling water). Robot control is assigned to the finer manipulation actions of the task (e.g., aligning gripper orientation, grasping). The system was implemented for two main tasks: a drinking task and a popcorn making task. Generalizations emerge through these two tasks across task actions, states, and task sequences which reduce software code. For example, both tasks contain pick sequences which share the same state and action state machine.

In the drinking task, Figure 3, first a cabinet is opened (open gripper, move to cabinet, align with handle, grasp, pull back in a circle arc), and then a cup is retrieved (open gripper, reach to cup, grasp cup, retract) from the cabinet. The cup is then filled and brought to a drink position before being returned to the table. Each phase has an internal state machine with multiple states and actions which can be assigned to the user or robot. For example, filling and drinking are two user-assigned actions (Figure 3 sequence 3), since the completion time of these action can vary based on user preference.

The popcorn making task increases the complexity. This task contains similar components as the drinking task but now interaction objects can be used in more than one sequence and different actions can be performed with a single object. This is where the sequential order of the task is now tracked by the system for knowing the next action and prompting action choices

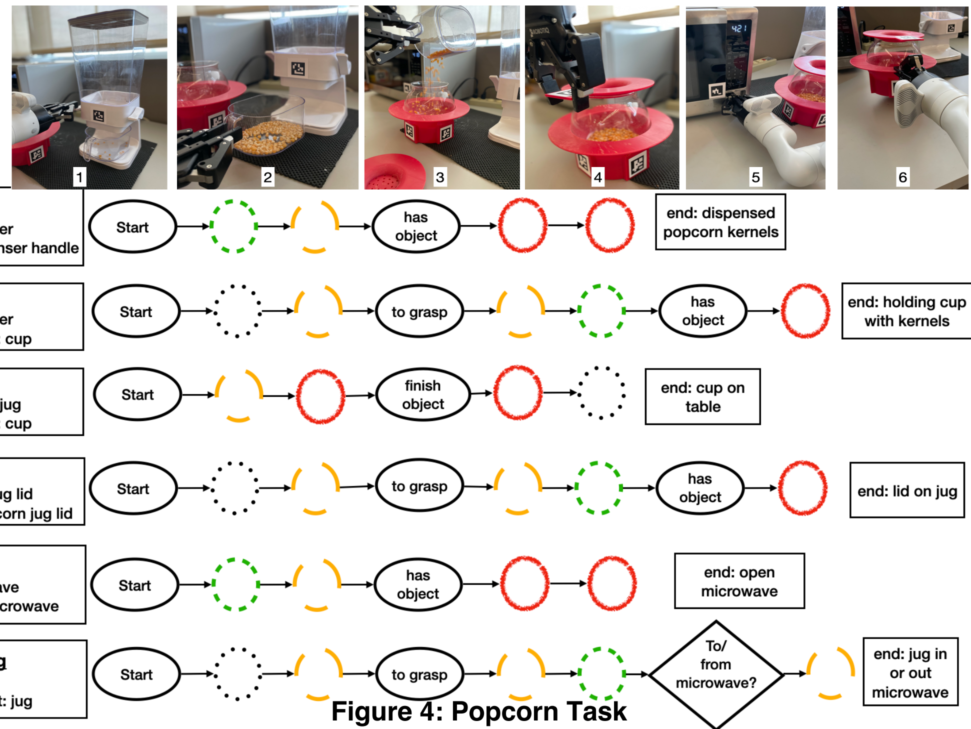

Figure 4: Popcorn Task

to the user for robot control. Note that if the user is in control of the current action, the sequential order is unnecessary, since the user is free to manipulate the interaction object however they wish. Figure 4 shows the full popcorn task. The complexity of the task was further eased using adaptations to the popcorn jug and a voice-controlled smart microwave. First popcorn kernels are dispensed into a cup, retrieved, and poured into an adapted jug. After the cup is placed on the table, a lid with perforated wholes is placed on the jug to prevent

popcorn from escaping. Then the microwave is opened, and jug is placed into the microwave. The last sequence (Figure 4 sequence 6) also shows an example of the user feedback state. The *move jug* sequence can be generalized by asking the user information, through the touchscreen, to know if the jug should be placed in the microwave or removed from the microwave.

DISCUSSION

Our initial testing will compare manually operating the ARM joystick in the drinking and popcorn tasks with switching the control authority assignment of the user and robot autonomous software. This will evaluate if assigning a fixed control authority, where the user controls larger gross arm movements and the robot software controls finer manipulation movements, is preferred.

As we refine our code to support more tasks, commonalities will further categorize states and actions at an appropriate resolution. We hope to collaborate with occupational therapists for matching their task analysis techniques with our hierarchical assumptions. These collaborations will better inform the design increasing the usability and effectiveness of assistive robotic systems. Another continuation is to gather information as the user is controlling the system for more seamless control authority transitions. For example, if we could classify that the user is struggling, we could have the robotic system more seamlessly offer help and transition. Additionally, understanding the user's progression within an action could allow the robot software to anticipate next actions for better supporting the user.

For future testing, the ARM will be mounted to the participant's wheelchair. As the user controls the ARM, they often shift their weight by adjusting their field of view, or for personal comfort. These movements affect the robotic arm base's reference frame, creating shifts in both the z-axis (like a bounce) and translationally necessitating sensing of the ARM's base position. This tracking is also necessary for tasks that include the motion of the EPW. For example, opening a refrigerator includes the strategy of moving the power wheelchair backwards while having the arm gripped to the door handle. This example also expands the number of control authority agents. The EPW is another user-controlled input (already supported by the system if assigned to user control), or it could be modeled as an additional control authority agent (if the wheelchair software independently controls the system).

CONCLUSION

This paper presents a hierarchical software architecture for facilitating assignment of control authority within multi-action manipulation tasks. By allowing system designers to easily change which parts of the task are controlled, we can run studies that explore participant preferences and perceptions of autonomous robot software assisted platforms. This is important for driving user-centered design and understanding where assistive robotic technologies can adapt within the user's routine.

REFERENCES

[1] Holliday PJ, Mihailidis A, Rolfson R, Fernie G. Understanding and measuring powered wheelchair mobility and manoeuvrability. Part I. Reach in confined spaces. Disabil Rehabil 2005;27:939–49.

[2] Susan Charlifue P. Family Caregivers for Veterans with SCI: Exploring the Stresses and Benefits. 2016.

[3] Allin S, Eckel E, Markham H, Brewer BR. Recent Trends in the Development and Evaluation of Assistive Robotic Manipulation Devices. Phys Med Rehabil Clin N Am 2010;21:59–77.

[4] King C-H, Chen TL, Fan Z, Glass JD, Kemp CC. Disability and Rehab: Assistive Technology Dusty: an assistive mobile manipulator that retrieves dropped objects for people with motor impairments. Assist Technol 2012;7:168–79.

[5] Pascher M, Baumeister A, Klein B, Gerken J. Recommendations for the Development of a Robotic Drinking and Eating Aid - An Ethnographic Study. INTERACT 2021

[6] Bhattacharjee T, Gordon EK, et al. Is More Autonomy Always Better? Exploring Preferences of Users with Mobility Impairments in Robot-assisted Feeding 2020. HRI March 2020; pp. 181–190

[7] Herlant L V, Holladay RM, Srinivasa SS. Assistive Teleoperation of Robot Arms via Automatic Time-Optimal Mode Switching. HRI 2016

[8] Argall BD. Autonomy in Rehabilitation Robotics: An Intersection. Annu Rev Control Robot Auton Syst 2018;1:441–63.