Grasping Robot Integration and Prototyping: The GRIP Software Framework

Brice Denoun, Beatriz León, Miles Hansard, and Lorenzo Jamone.

***Abstract*—Robotic manipulation is fundamental to many real-world applications; however, it is an unsolved problem, which remains a very active research area. New algorithms for robot perception and control are frequently proposed by the research community. These methods must be thoroughly evaluated in realistic conditions, before they can be adopted by industry. This process can be extremely time consuming, mainly due to the complexity of integrating different hardware and software components. Hence, we propose the Grasping Robot Integration and Prototyping (GRIP) system, a robot-agnostic software framework that enables visual programming and fast prototyping of robotic grasping and manipulation tasks. We present several applications that have been programmed with GRIP, and report a user study which indicates that the framework enables naive users to implement robotic tasks correctly and efficiently.**

I. Introduction

ADVANCES in robust autonomous grasping and manipulation would enable new degrees of autonomy in industrial production facilities [1]. This prospect has driven research into both the hardware [2] and software [3] aspects of the problem, leading to increasingly capable robotic systems [4], [5]. Although new algorithms are usually evaluated on physical robots, constraints in industry are often quite different from those in the research laboratory. For this reason, it is crucial for companies to test and compare newly developed methods, in the context of their own setup and constraints. For example, a new robotic arm controller, which proved effective in a human-robot handover experiment, could be of interest for a company in a pick-and-place task, subject to performance evaluation. However, this presents several problems, the most challenging of which is *integration*. For example, the integration of a new planner or grasping algorithm, into an existing production pipeline, requires substantial effort and resources, owing to the divergence of tools that are used in research and industry. For this reason, companies may be reluctant to explore new systems that have been developed in the wider research community.

To partially bridge this gap between academia and industry, we believe that a framework for efficient evaluation of new systems is required. We argue that such a tool should meet three conditions. First, it should be easy for users to interface their own hardware with the components that needs testing, in order to reproduce the essential features of their working environment. Second, integrating a new software component (e.g. kinematics library or motion planner) should be possible without having to understand the internal details. Third, it should be possible to design and evaluate tasks related to grasping and manipulation, with minimal investment of time. We believe that a tool with these characteristics would support the interaction of research and industry, so that companies could maintain a competitive advantage, by implementing the best performing methods from the research community. On the other hand, academia would see the application of novel ideas to real-world use cases, which could reveal new research problems. Furthermore, the community at large would benefit from the increased capabilities of robots.

B. Denoun, M. Hansard, and L. Jamone are with the ARQ Centre for Advanced Robotics at QMUL, School of Electronic Engineering and Computer Science, Queen Mary University of London, UK {b.d.denoun, m.hansard, l.jamone}@qmul.ac.uk.

B. Denoun and B. León are with The Shadow Robot Company, London, United Kingdom.

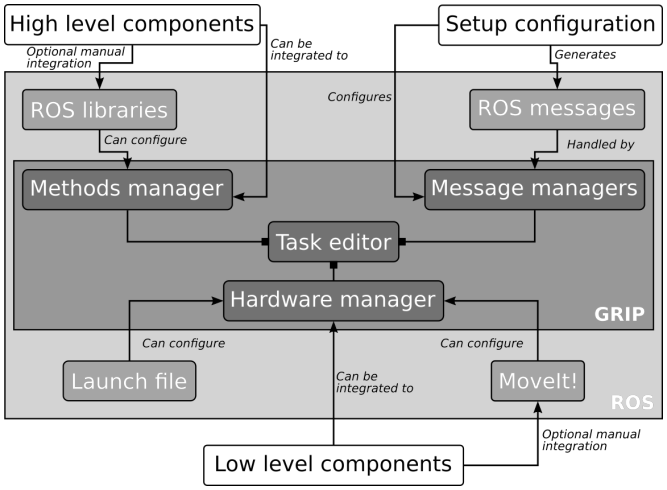


Fig. 1. Overview of the GRIP framework. The architecture allows users to operate robots with integrated components (hardware and/or software) for grasping and manipulation tasks. The task editor provides an intuitive interface for designing and tuning the robot's behaviour. Arrows indicate the different (mutually compatible) ways to interface external components.

Hence we propose the Grasping Robot Integration & Prototyping (GRIP) framework, a system for rapid and intuitive prototyping of grasping and manipulation tasks, involving multiple integrated components. This hardware-agnostic software framework is designed to reduce the time and effort required to explore, test and evaluate newly published robotics algorithms. In particular, the Graphical User Interface (GUI) guides and supports the user in all the necessary steps, including: (1) hardware configuration, (2) software integration, (3) design and execution of complex manipulation tasks.

II. Related work

Working with robots has become substantially easier in recent years, because robot-agnostic mid-level software has provided the communication tools that are required to operate

complex platforms [6], [7]. In particular, the Robot Operating System (ROS) [7] furnishes the main tools to manage low-level tasks such as communication and control, as well as high-level tasks such as package management and algorithm integration. These features have made ROS one of the most commonly used systems in a wide range of robotic research, such as navigation [8] or planning [9]. For robotic grasping and manipulation, MoveIt! [10] is a widely used ROS-compatible solution. However, making the most of software stacks such as ROS and MoveIt! requires an extensive knowledge of hardware, motion planners, kinematic libraries, sensors, and controllers – and how they interact with each other. On the other hand, companies tend to use more domain specific solutions [11], which are usually faster [12], and which satisfy particular constraints. However, the deployment of ROS-based algorithms, on a given platform, may be challenging for inexperienced users. This problem has been partially addressed, in other software systems.

For example, PyRobot [13] provides a common Python-based interface to operate supported robots. This software, which wraps around ROS, enables Machine Learning and Computer Vision researchers to focus on developing high level components (such as grasp pose determination or vision-based navigation), without having to deal with low-level components (such as controllers or kinematics). However, only components implemented in Python are supported, and integrating new robots may require advanced ROS, MoveIt! or Python knowledge, which can be an obstacle to the naive user.

Other works have been presented to enable visual programming of robots through state machines [14], [15]. One example, which is comparable to our proposed framework, is RAFCON (RMC Advanced Flow Control) [16]. This hardware and middleware independent framework allows the user to design complex robotics tasks, through an intuitive GUI. It uses a system of hierarchical state machines to determine and coordinate the robot behaviour. RAFCON is a very general framework for programming robots, ranging from simple indoor navigation to complete space exploration missions. To handle such complex sequences of operations, the task description is high-level (e.g. 'explore', or 'pick object') rather than low-level (e.g. use a specific inverse kinematic solver to move to a computed position). Given the variety of tasks and robots that can be programmed, the practical use of this tool becomes very complex, and may be dependent on previously developed examples. In particular, the integration of low-level components may not be straightforward.

Some industry-specific frameworks are also commercially available and allow for rapid and intuitive design of automated and/or autonomous tasks, such as Artiminds[1], Mech-Viz[2] and drag&bot[3]. These systems mainly support industrial robots and can be programmed through a drag and drop interface. They come with a set of available operations (e.g. move or search). However, these systems are not always able to accommodate external components (e.g. from academia), as companies usually offer extensions for autonomous sensory-based systems. We have therefore identified the need for a framework that can incorporate both low-level and high-level components, in an intuitive way.

This is the aim of GRIP: a hardware agnostic ROS-based software framework, which facilitates prototyping and robot integration, specifically for grasping and manipulation tasks. Both *low-level* and *high-level* components (kinematic libraries, controllers, motion planners, sensors and learning methods) can be integrated, without imposing constraints on their implementation (see Figure 1). This greatly improves the reusability of existing software components, in the industrial setting. Unlike the existing solutions, that mainly focus on easing the programming part, GRIP guides users from robot and software integration to task execution thanks to a reactive GUI. The architecture of our framework minimises the effort required to interface existing software, even on complex systems (e.g. bimanual robots). We therefore believe that GRIP is a useful tool, which makes it possible to evaluate grasping and manipulation software/hardware, with minimal overhead. The optimal combination of components, for a given task, can then be integrated into an existing and optimised pipeline.

III. Overview of GRIP

The main purpose of this framework is to provide an environment in which robots can be operated with ROS-compatible software components, coming from academia, as well as external software components. Although GRIP provides a wide range of integration and interfacing options, we made sure to keep its use straightforward and as programming-free as possible, as follows:

- Robot interfacing; from material widely available online
- Software and sensor integration; regardless of their implementation details
- Variables definition; to be used during the robot execution
- Task design and execution with integrated components; through a visual drag and drop interface

Typical use-cases implemented using this procedure are described in section VI and videos are available online[4].

Section IV explains how we ensure that a maximum of existing components can be integrated to GRIP with a minimum overhead while maintaining a good communication between components. Section V presents the different mechanisms implemented to graphically guide naive users through all the steps from robot integration to task execution. Section VI demonstrates the use of GRIP in five use cases, and briefly discuss the differences with comparable works. The intuitiveness and contributions of our framework are evaluated in section VII, based on a practical user study, involving participants with different degrees of expertise in robotics.

IV. Integration

This section explains how GRIP is able to run a wide range of hardware and software, in order to make good use of existing components. This entails two challenges: how to

[1]https://www.artiminds.com/
[2]https://en.mech-mind.net/
[3]https://www.dragandbot.com/
[4]https://sr-grip.readthedocs.io/en/latest/

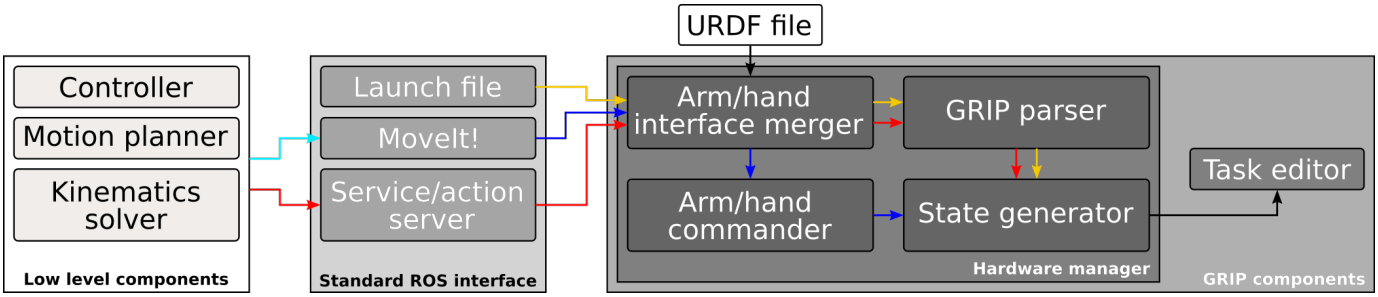


Fig. 2. Diagram showing the different ways of interfacing a robot with GRIP. Colours indicate the integration modalities, as described in section IV of the text. Robots can be configured through MoveIt! (blue: IV-A1) or using an existing *launch file* that gathers all components to be run (orange: IV-A2). External low-level components can also be integrated into our framework by wrapping them into ROS actions or services (red: IV-B). Black arrows indicate consistent operations across the integration modalities.

avoid the need for GRIP-specific configuration files, and how to integrate software components for which a standardised format is yet to be defined.

A. Interfacing hardware

We assume that the robot has ROS-compatible drivers, as commonly provided by the manufacturer. In addition, like most ROS-compatible software, we rely on the standard Unified Robot Description Format (URDF) file, which typically accompanies the drivers. This standard file encapsulates hardware-specific information, which downstream services can access. Although GRIP provides a way to integrate several components separately (see section IV-B), it also offers two ways to interface robots (i.e. running several components at once) from materials widely available online.

1) Using MoveIt!: As previously mentioned, MoveIt! [10] is a ROS-compatible framework for manipulation tasks, which is commonly used in the research community. It embeds kinematics libraries, controllers, and motion planners for robotic hands/arms. Once a MoveIt! configuration package is defined for a robot, it can be controlled through a C++ or Python API. This stack allows the user to carry out basic operations such as motion planning or collision avoidance, without having to implement them. Due to an active and large community, a wide range of configurations are available for standard robot arms (e.g. Franka Panda, Universal Robots, Kuka, Kinova) and hands (e.g. Allegro Hand, Shadow Dexterous Hand). In order to use such resources, robots can be interfaced to GRIP using only a MoveIt! configuration package. After graphically specifying some options (e.g. available planners, robot speed), this configuration is used to initialise an arm and/or hand commander (see Figure 2). These commanders are objects which, in addition to include most of MoveIt! functionalities, embed useful functions e.g accessing to upstream information to ensure a full support of components previously integrated to the MoveIt! stack (light blue arrow in Figure 2). We believe this compatibility with MoveIt! is essential, in order to embrace as much existing work as possible.

2) Using a launch file: GRIP also offers the possibility to interface a robot through a launch file (i.e. ROS file gathering all the components to run, potentially including MoveIt!). However, the controllers and components to be used (even if already run in the launch file) must be registered in our framework, in order to create the appropriate states in the task editor (see Figure 2). This interfacing mode allows users to operate complex hardware, without a detailed understanding of it. We demonstrate this facility with both the Allegro Hand and the Shadow Dexterous Hand, in section VI.

B. Integrating software components

For GRIP to be able to run software written in different programming languages, and with different interfaces, users need to:

- Create a ROS package
- Wrap their code inside a ROS service or action server (see Figure 2)

These operations are straightforward, do not need an in-depth knowledge of the ROS stack and are widely documented[5]. We provide examples of this process for components written in different programming languages[6]. Since an external component can expect input and can output meaningful information to be used by other components, the server needs to follow these rules:

- The request (i.e. input) of the server needs to be named *input* and does not have any constraint about its type
- The response (i.e. output) of the server must contain two fields; an integer named *outcome* and *returned object*, without constraint about its type

The *outcome* field needs to be returned according to the component's behaviour. For instance a grasping method can find one grasp (outcome 0), several (outcome 1) or none (outcome 2). The only constraint is that the total number of outcomes needs to be known when adding this component to the GUI. We believe that this interfacing modality is generic and modular enough to handle most of the different types of components that are encountered.

C. State generator

As illustrated in Figure 2, interfacing hardware and software results in a set of states available in the task editor (see section V-B) which enables users to graphically design and execute complex manipulation tasks. When interfaced through

[5]http://wiki.ros.org/actionlib/Tutorials
[6]https://sr-grip.readthedocs.io/en/latest/user_guide/resources.html

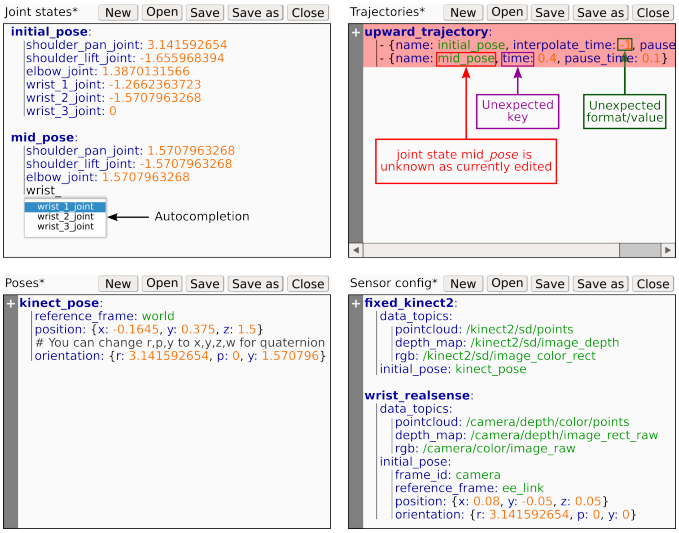


Fig. 3. Example of the configuration required to run two RGBD-sensors: one fixed, and one mounted on the wrist of a robot. This figure also shows some of the mechanisms implemented to provide the user feedback about the current configuration. It includes auto-completion, live parsing and live update of already registered configuration. The '+' symbol in the margin marks a template, which must be completed in order to add a new component (e.g. pose, trajectory or sensor).

MoveIt!, robots can be operated via states that internally call one or more methods of the configured commander. For each external component integrated to GRIP, its configuration is parsed to generate a new state using the Jinja2 templating library[7]. This results in states that can be graphically configured (e.g. input of the component) and that contain the user-specified number of outcomes. When executed, these states call the server and trigger the transition set by the user, according to the *outcome* field of the response.

D. Message managers

One important factor when designing robotic tasks is controlling the flow of information between components. It becomes even more challenging when dealing with components coming from different sources. ROS introduced the notion of messages that are sent through channels called topics. These topics can be listened to by any ROS-node to catch the information sent by others. However, storing and/or sending specific messages across components is not straightforward, especially for naive users. In order to make the communication between integrated components more intuitive, GRIP features message managers. These objects allow the messages sent by others nodes or previously defined in the GUI to be stored or retrieved at any level of the task. Users can either interact with the managers through the GUI (e.g. add/remove a new entry, display the current messages that are stored) or through Python and C++ APIs that can be included in external scripts.

GRIP provides managers for messages containing generic information, including joint states, poses, plans, and trajectories. As illustrated in Figure 3, users can initialise these managers through the GUI. Although these messages are generic and widely used in low and high-level components, we provide users with the possibility to add managers for custom messages through a well documented procedure. The purpose of the message managers is to give researchers better control of the flow of messages, which can be confusing when working with unfamiliar software components.

It is important to note that the GRIP interface modalities are not mutually exclusive, and that a user can operate a robot in the task editor regardless of how components are interfaced to our framework. This makes it possible to run a custom low-level controller while also making use of MoveIt!. We will demonstrate the use of multiple interface modalities in section VI. We believe that this approach will greatly facilitate the incorporation of new software components. Our framework is unique in offering this freedom of interfaces, while also providing a task editor to carry out experiments.

V. Graphical User Interface

Unlike other available software tools, GRIP provides an intuitive and reactive GUI that guides users through all of the steps from robot integration to task execution. In this section, we highlight some of the mechanisms implemented to help the user.

A. Integration stage

Integrating unfamiliar components can be challenging, for instance due to unfamiliar syntax. In order to improve its usability, our GUI includes a wide range of features, which help the user to configure its robot. For example, we have implemented real-time checks on all of the different entries the user can interact with. If the input is not valid, then the user is directly notified by red signals. These measures range from simple filename conventions, to full configuration syntax checks. As illustrated in Figure 3, we have implemented a real-time parser, which will indicate whether or not the current input is valid. In addition, we have implemented different levels of help, depending on the information provided by the user to the GUI, such as autocompletion or display of appropriate templates. These features are designed to help users familiarise themselves with GRIP, and to help them cope with complex robotic tasks.

B. Task editor

In common with other robot programming frameworks [14], [16], we opt for a graphical programming paradigm through state machines. This approach is more intuitive than textual programming and does not require specific prior knowledge. Unlike FlexBE [14] or RAFCON [16] that implemented their own state machine system, we decided to base our task editor on the well established SMACH [17] library. This ROS-compatible software provides a wide range of state classes and containers (e.g. hierarchical, concurrent, sequential) allowing users to create complex and adaptive tasks through textual programming. While SMACH containers include a *userdata* object (i.e. dictionary of variables accessible across the states),

[7]https://jinja.palletsprojects.com

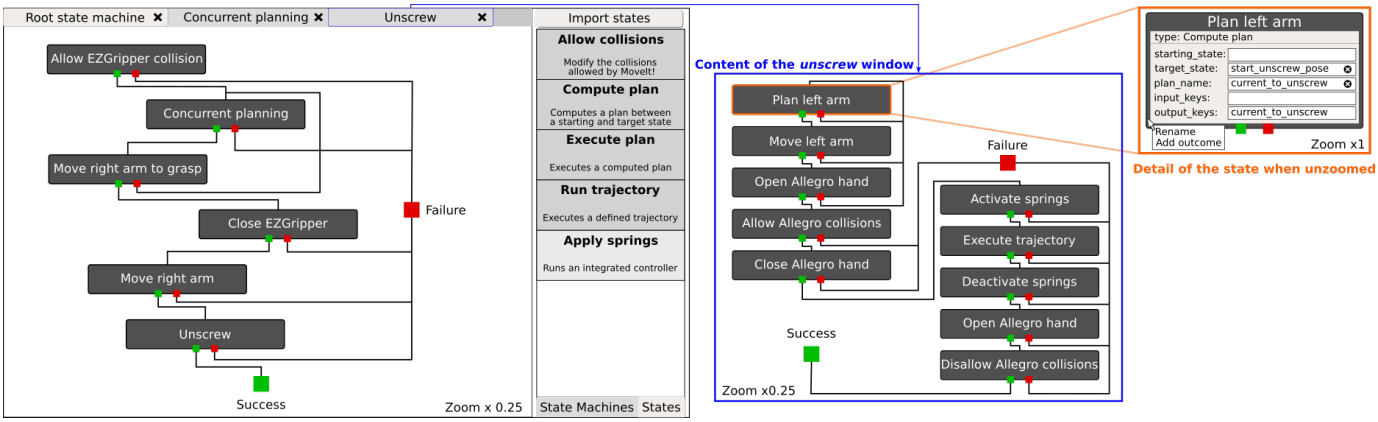


Fig. 4. Appearance of the task editor when designing a bimanual operation (see section VI-**Bimanual robots**). The user can navigate both between and within hierarchies, via the sub-windows that are created when new containers are added. Different levels of zoom will show or hide the configuration data of each state, to ensure an appropriate visualisation.

a certain degree of expertise is required to implement coherent and deep state machines with a correct flow of information. For this reason, all of the generated states and containers proposed in our framework are derived from SMACH components, and include our modifications to enable intuitive visual programming. This will be demonstrated in section VII.

As illustrated in Figure 4, creating a task consists of:

- Drag and dropping states or state machines in the window
- Configuring each state
- Connecting the outcomes of the different elements to define the behaviour of the robot

This intuitive way of programming allows the user to understand the logic of the task at a glance. Unlike RAFCON or FlexBE, for which the user needs to define global variables, our state machines can dynamically store/retrieve information from both its userdata and the different message managers. In addition, each state can quickly be further configured directly in the task editor. These features allow for rapid configuration of complex robotic tasks, unlike textual programming.

One of the main challenges related to such a GUI is navigating inside substantial state machines with deep hierarchies. An appropriate visual interface can ease the design and configuration of complex tasks. In particular, we have implemented zooming and multi-windowing mechanisms (see Figure 4). Zooming-in on a specific state displays the different options that can be configured. When a new state machine is added to an existing one, a compact and box-like representation will appear, and a sub-window corresponding to the new container will be created. This window will be dedicated to the definition and configuration of the given hierarchy, while keeping a clear view in the first sub-window. The task editor also features consistency checks, which can assist novice users. At execution time, our containers allow the user to keep track of which state is currently being run, which provides a useful tool for debugging.

We believe that GRIP's task editor enables intuitive visual programming of grasping and manipulation tasks. It can be used by experienced users, while providing a simple and straightforward interface to novice users. We argue that the interactive mechanisms, implemented in the GUI, make it easy to define and edit complex robotic behaviours (see section VII-B).

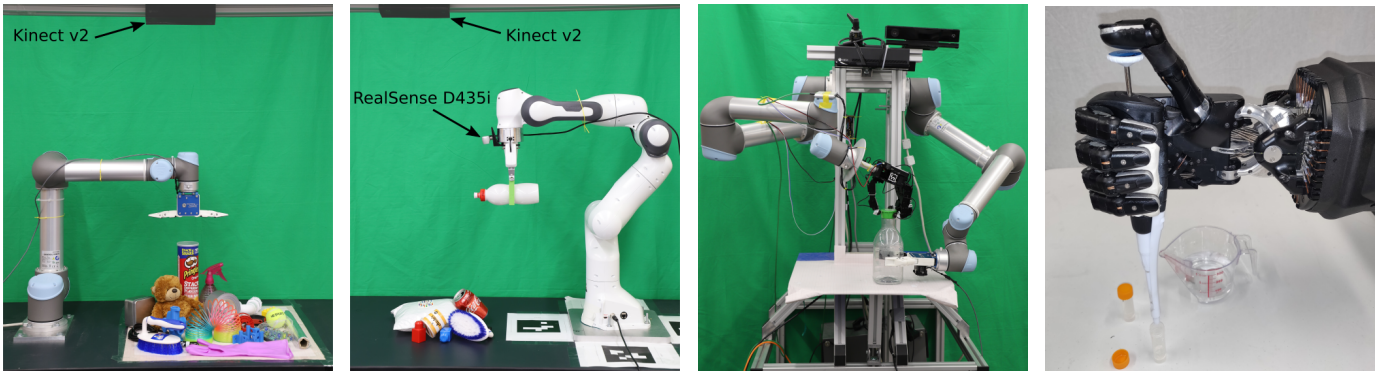
VI. Case Studies

This section outlines five examples, from the wide range of scenarios in which GRIP can been used.

Benchmarking (Figure 5-a): GRIP can be used to evaluate and analyse several state-of-the-art components of a grasping or manipulation pipeline. In the present work, we benchmarked four grasping algorithms [18], [19], [20], [21] on a set of 20 objects, over a total of 6000 grasps. Each method has been deployed on a EZGripper mounted on an Universal Robot UR5 arm. Since these different methods have not been demonstrated using similar platforms, their associated code usually includes robot-specific components for control or motion planning. However, some of these components are not available for our platform. In addition, none of the methods we benchmarked describe a grasp using the same convention, which leads to four different messages. The architecture of GRIP allowed us to configure a common motion planning and control pipeline using MoveIt!, which ensures that we evaluate only the performance of the grasp generation. We quickly integrated the core methods by wrapping them into services and using the C++ and Python API to make their output compatible with the controller provided with the EZGripper.

Visual-exploration (Figure 5-b): Multiple sensors can easily be integrated with GRIP, thereby facilitating the exploration of cluttered scenes. In this case, we used a RealSense D435i mounted directly on the end effector of a Franka Panda arm, in conjunction with an externally fixed Kinect2. In this work, the robot performed a pre-defined spiral trajectory above the objects, and periodically captured point clouds from the RealSense device. The collected point clouds were registered through a custom method and the combined point cloud was used as input to an existing grasping algorithm [20]. Here, GRIP allowed us to easily run and coordinate a recently published method with an external component, not specifically designed for ROS.

Bimanual robots (Figure 5-c): Two robot arms are controlled



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

Fig. 5. Examples of robotic tasks that can be executed with GRIP. These include (a) benchmarking of grasping algorithms, (b) dual-camera based exploration of clutter, (c) bimanual dexterous manipulation, and (d) repetitive industrial tasks.

in a coordinated way to unscrew the lid of a plastic bottle. One arm is equipped with an EZGripper and is used to hold the bottle using a MoveIt! controller. The other arm is equipped with an Allegro Hand which is controlled using an external ROS package [22] to unscrew the lid. For this use case, we used the launch file provided in the ROS package of the spring framework that directly runs gravity compensation for the Allegro hand's fingers. Once two MoveIt! arm controllers were registered in our GUI, we were able to easily coordinate the two arms to unscrew the lid. Here, we integrated a custom-made controller for the EZGripper, which adapts the force applied to keep the bottle stable. The task editor corresponding to this use case is illustrated in Figure 4.

Industry-related tasks (Figure 5-d): A complex dexterous manipulation task was carried out using the Shadow Dexterous Hand mounted on a UR10 robot arm. The task consisted of picking a pipette, drawing some liquid contained in a glass container to release it in a small plastic tube, which is subsequently grasped and moved to a pre-determined pose. This use case reflects the potential of the GRIP framework to cope with automated real-world tasks. Similarly to many commercial robots, both the Shadow Hand and the UR10 are ROS compatible and are provided with launch files that run all the necessary ROS components. Thanks to the hardware manager, both robots could be operated by GRIP by simply interfacing a launch file provided by the manufacturer. The necessary states were then generated automatically, by GRIP. The task editor was subsequently used to choose the sequence of way-points and motion trajectories for the arm and finger joints required to automate the task. The entire process took approximately 30 minutes to complete.

Design of reactive behaviours: The GRIP framework can also be used to easily generate new and reactive behaviours from existing components at the task level. In fact, due to the task editor interface and the ease with each new states can be added, reactive behaviours can be implemented graphically, without having to modify an existing method. For instance, we upgraded a plain pick-and-place system, so that placement into a crate is only attempted if the picking stage succeeded. After integrating a grasping method [19], we have integrated a high level method that performs an image difference to check if an object has been grasped, and a new state that generates a new end-effector pose within a given radius.

These different scenarios show the variety of tasks and robots that can be targeted using GRIP, by combining different integration modalities. It is important to note that some of these tasks could not have been implemented using commercial solutions (e.g. drag&bot) because the set of supported robots is fixed. On the other hand, GRIP allows users to interface their own robot, using widely available materials. Designing complex tasks such as the bimanual operation through textual programming (e.g. with SMACH) would be tedious due to the substantial number of states and containers involved. While RAFCON and FlexBE provide advanced GUIs to design similar tasks, integrating new robots or new software components is less flexible. For example, each component to be run by such frameworks would need to be rewritten in a specific language following a very specific format or template.

VII. User Study

GRIP's practical usability has been evaluated in a user study. Its purpose is to assess the intuitiveness of (1) interfacing different robot hardware, (2) integrating different software components and (3) designing and executing manipulation tasks, in typical robotic scenarios.

A. Experimental protocol

In order to evaluate GRIP in realistic conditions, we asked participants to perform a set of four tasks, in the following sequence:

1) Integrate a UR5 robot arm using MoveIt! and make it move between two pre-recorded joint states.
2) Change the kinematics library and motion planner configured by default to TRAC-IK[8] and BiTRRT[9]. Repeat the previous motion to make sure the integration was successful.
3) Using a launch file, run the Shadow Dexterous Hand mounted on a UR10 arm, and add the necessary states

[8]http://wiki.ros.org/trac_ik_kinematics_plugin
[9]https://ompl.kavrakilab.org/classompl_1_1geometric_1_1BiTRRT.html

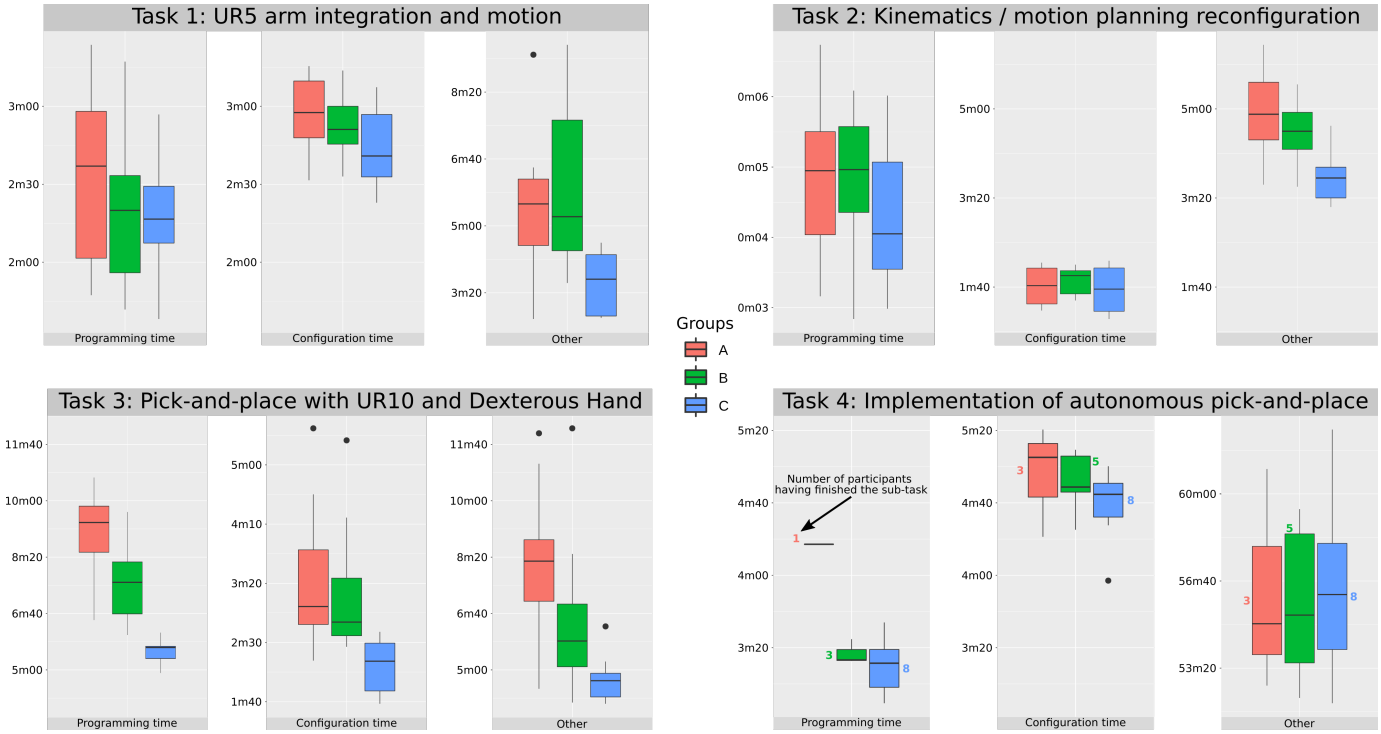


Fig. 6. Experimental time distributions, with respect to different tasks (four panels), and levels of expertise (coloured boxes, with quartiles above and below the median, in each case). A total of 12 participants managed to complete all tasks within 1 hour 45 minutes, and 4 more participants completed all tasks except programming the robot. All participants managed to complete the three first tasks within 1 hour. For simple tasks (top row), robot configuration and programming time is similar regardless of the expertise of users; this can be attributed to the intuitive GRIP GUI. However, the level of expertise does affect the time required to complete more complex tasks (bottom row). The general trend is that the time spent to read and assimilate the documentation (denoted 'other') is inversely related to the level of expertise.

to the previous task to create a pick and place scenario from pre-recorded poses.

4) Add a Kinect2 and integrate an external grasping algorithm [19], to run an autonomous pick and place task.

In order to avoid unexpected behaviours, participants were asked to select the joint states and poses among a list of safe ones. Participants were provided with the links to the official repositories of the grasping algorithm[10] and of the Shadow Hand[11]. For convenience, the names of the topics generated by the Kinect2 were also provided in the instructions.

In order to reflect realistic conditions, participants did not have any training time with our software, but were provided with a documentation. The latter only contained generic principles (e.g. wrapping code inside a ROS server) and examples for robots and software components that were not involved in the experiment (e.g. Franka Panda robot), limiting the documentation bias.

The cohort was composed of 25 participants; 15 males and 10 females between 22 and 54 years old. All participants had a Computer Science background and were first-time users of GRIP. Among them, only three had experience in programming Universal Robot arms, and none had worked with the Shadow Dexterous Hand. Before the study, each participant was asked to complete a questionnaire about their background. No personal information was asked (apart from age and gender) and participants gave their consent that the collected data was going to be anonymously used in a scientific study. This questionnaire consists of three 0–4 rating scales corresponding to the following questions:

- How familiar are you with robotics? Was it part of your education, or do you have other experience in the field?
- How familiar are you with programming robot hardware, for grasping and manipulation tasks?
- How familiar are you with ROS (Robot Operating System)?

For these three questions, answering '0' means having no knowledge whereas '4' means having in-depth knowledge and thus being proficient in the task, tool or concept. Based on the replies from the participants, we were able to define three groups:

(A) Those with no knowledge of robotics (9 participants).
(B) Those with basic robotics knowledge, but who have never used ROS (8 participants).
(C) Those who have previously used ROS, or who have previous experience in robot programming for manipulation tasks (8 participants).

This classification allows us to estimate the value of GRIP, for each type of user. A useful metric for this is the time spent by participants on each task and sub-task (i.e. configuring hardware and software components, design and execution of the task). We also measured how much time participants spent consulting documentation and external resources, rather

[10]https://github.com/dougsm/ggcnn
[11]https://github.com/shadow-robot/sr_interface/tree/kinetic-devel

than using the framework itself. In order to collect this data, participants worked with a version of GRIP that embeds several timers. Every time the user changed the window focus, time stamps were stored. In addition, participants were asked to press a button when they finish a task. The resulting time stamps allowed us to determine the time spent on windows within and outside our GUI.

At the end of the allotted time (1 hour 45 minutes), participants were asked to complete another questionnaire about their experience. It consists of one binary question and three 0–4 rating scales:

- How intuitive was the configuration of a robot?
- How intuitive was programming the robot (i.e. using the task editor)?
- How frustrating was your experience with the software?
- Have you completed all of the tasks? If not, how much more time do you estimate would have been required?

When rating frustration, we asked participants to consider the overall usability of GRIP and whether they thought the GUI was overly constraining. For this particular scale, '0' means no frustration, whereas '4' means very high frustration (i.e. they were not able to perform anything). For the last question, if participants answered 'no', then they had to choose between the following possibilities: less than 15 minutes, between 15 and 30 minutes, between 30 and 45 minutes, between 45 minutes and 1 hour, more than an hour.

B. Results

The breakdown of the time spent on each task for each group of participants is shown in Figure 6. Similarly, the ratings from the post-experiment questionnaire are reported in Figure 7 for each group of participants. Both sets of data are used to evaluate GRIP, based on three aspects:

Intuitiveness of configuration: The total amount of time spent on configuring hardware and software components is consistent among the three groups and remains low (less than 6 minutes) across the tasks. We attribute this effect to our intuitive GUI, which helps users, across all groups, to integrate hardware and software components.

Intuitiveness of programming: The total amount of time spent on designing and executing the two first tasks is not statistically different across the three groups. For the third task, the difference becomes significant, as experienced participants spent less time than more naive users. We attribute it to the difference in participant's background. In fact, users that are unfamiliar with robotics would need more time to understand and coordinate all the components required in a pick-and-place scenario. However, the task editor was reported to be quite intuitive by all users, regardless of their group.

Completion time: All participants were able to complete the three first tasks in a short amount of time. This means that even naive users were able to integrate and configure two robots, and to execute an automated pick-and-place operation within an hour. The reported time also consists of external factors, including reading the documentation and assimilating novel concepts. This demonstrates that our GUI and interface modalities are intuitive and straightforward, even for users with no previous knowledge in ROS or robotics. Within the allotted time, task 4 was completed by 12 participants (1 from group A, 3 from group B and all participants from group C), with an average completion time of 1 hour 38 minutes. As indicated in Figure 6, four naive participants managed to finish configuring the robot and to integrate the grasping algorithm, but did not have enough time to program it. This is because users needed to understand the inputs and outputs of the grasping algorithm to successfully port it, which can be a long process without any background in robotics and computer vision.

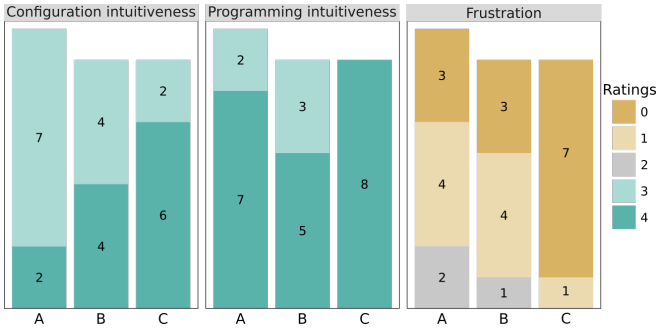


Fig. 7. Distribution of ratings given by participants in each expertise group, in the second questionnaire. Each participant was asked to evaluate their experience regarding the robot configuration, programming intuitiveness, and general frustration.

The results of this user study show that even without previous knowledge of robotics, users are able to configure different robots and program them to perform simple tasks, in a limited amount of time, owing to the intuitive GUI. On the other hand, our framework allows users with more knowledge to easily carry out typical grasping tasks in less than 1 hour 45 minutes, with integrated software and/or hardware. These results highlight the contribution of our framework: an intuitive and modular interface to (1) interface new robots, (2) integrate new components and (3) design and execute grasping and manipulation tasks.

VIII. Conclusion

We have presented and evaluated GRIP, a hardware agnostic ROS-based and standalone software, which facilitates robot programming with integrated components for grasping and manipulation tasks. Its intuitive GUI allows users to configure a wide range of robots and software components. In order to decrease porting and integration effort, our software also supports a wide range of hardware and software integration methods. It features a dedicated interface, which can be used to intuitively design and execute a wide range of tasks, through hierarchical state machines. This was demonstrated in five case studies, involving different robots and tasks. It was shown that the modular GRIP framework can cope with tasks such as pick and place, and in-hand manipulation, as well as exploration with several sensors, and complex tasks involving a bimanual system. A realistic experiment showed that GRIP can be used by naive users to program robotic manipulation tasks successfully in a very short amount of time. Notably, users

have reported that the system is very intuitive, due to its well organised structure and easy to use GUI. GRIP is not limited to a specific application or to a specific robotic setup; rather, it is a general framework that can help any user to evaluate a new component on their own setup. We believe that this will facilitate the exchange of knowledge between academia and industry, and the deployment of robotic solutions in real-world applications.

ACKNOWLEDGMENT

This work was supported by the EPSRC UK (NCNR, EP/R02572X/1 and MAN^3, EP/S00453X/1) and by the Shadow Robot Company.

REFERENCES

[1] C. Canali, F. Cannella, F. Chen, G. Sofia, A. Eytan, and D. G. Caldwell, "An automatic assembly parts detection and grasping system for industrial manufacturing," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 215–220, IEEE, 2014.

[2] M. Honarpardaz, M. Tarkian, J. Ölvander, and X. Feng, "Finger design automation for industrial robot grippers: A review," *Robotics and Autonomous Systems*, vol. 87, pp. 104–119, 2017.

[3] Y. Litvak, A. Biess, and A. Bar-Hillel, "Learning pose estimation for high-precision robotic assembly using simulated depth images," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3521–3527, IEEE, 2019.

[4] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[5] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, "The ingredients of real world robotic reinforcement learning," in *International Conference on Learning Representations*, 2020.

[6] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings 2001 ICRA. IEEE international conference on robotics and automation*, vol. 3, pp. 2523–2528, IEEE, 2001.

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[8] R. L. Guimarães, A. S. de Oliveira, J. A. Fabro, T. Becker, and V. A. Brenner, "Ros navigation: Concepts and tutorial," in *Robot Operating System (ROS)*, pp. 121–160, Springer, 2016.

[9] M. Moll, I. A. Sucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[10] S. Chitta, I. Sucan, and S. Cousins, "Moveit!," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[11] A. S. Huang, E. Olson, and D. C. Moore, "Lcm: Lightweight communications and marshalling," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4057–4062, IEEE, 2010.

[12] A. Shakhimardanov, N. Hochgeschwender, M. Reckhaus, and G. K. Kraetzschmar, "Analysis of software connectors in robotics," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1030–1035, IEEE, 2011.

[13] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "Pyrobot: An open-source robotics framework for research and benchmarking," *arXiv preprint arXiv:1906.08236*, 2019.

[14] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with application to rescue robotics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2796–2802, IEEE, 2016.

[15] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp, "Ros commander (rosco): Behavior creation for home robots," in *2013 IEEE International Conference on Robotics and Automation*, pp. 467–474, IEEE, 2013.

[16] S. G. Brunner, F. Steinmetz, R. Belder, and A. Dömel, "Rafcon: A graphical tool for engineering complex, robotic tasks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3283–3290, IEEE, 2016.

[17] J. Bohren and S. Cousins, "The smach high-level executive," *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, 2010.

[18] A. Makhal, F. Thomas, and A. P. Gracia, "Grasping unknown objects in clutter by superquadric representation," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pp. 292–299, IEEE, 2018.

[19] D. Morrison, P. Corke, and J. Leitner, "Learning robust, real-time, reactive robotic grasping," *The International Journal of Robotics Research*, 2019.

[20] H. Liang, X. Ma, S. Li, M. Görner, S. Tang, B. Fang, F. Sun, and J. Zhang, "Pointnetgpd: Detecting grasp configurations from point sets," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3629–3635, IEEE, 2019.

[21] T. Suzuki and T. Oka, "Grasping of unknown objects on a planar surface using a single depth image," in *Advanced Intelligent Mechatronics (AIM), 2016 IEEE International Conference on*, pp. 572–577, IEEE, 2016.

[22] G. Solak and L. Jamone, "Learning by demonstration and robust control of dexterous in-hand robotic manipulation skills," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8246–8251, 2019.