Special Issue: Robotics Collaborative Technology Alliance (RCTA) Program

Systems Article

Human-Scale Mobile Manipulation Using RoMan

Chad C. Kessens[1], **Matthew Kaplan**[1], **Trevor Rocks**[1], **Philip R. Osteen**[1], **John Rogers**[1], **Ethan Stump**[1], **Arnon Hurwitz**[1], **Jonathan Fink**[1], **Long Quang**[2], **Mark Gonzalez**[2], **Jaymit Patel**[2], **Michael DiBlasi**[2], **Shiyani Patel**[2], **Matthew Weiker**[2], **Dilip Patel**[2], **Joseph Bowkett**[3], **Renaud Detry**[3], **Sisir Karumanchi**[3], **Larry Matthies**[3], **Joel Burdick**[3], **Yash Oza**[4], **Aditya Agarwal**[4], **Andrew Dornbush**[4], **Dhruv Mauria Saxena**[4], **Maxim Likhachev**[4], **Karl Schmeckpeper**[5], **Kostas Daniilidis**[5], **Ajinkya Kamat**[6], **Aditya Mandalika**[6], **Sanjiban Choudhury**[6] **and Siddhartha S. Srinivasa**[6]

[1]DEVCOM Army Research Lab, 2800 Powder Mill Road, Adelphi, MD, USA 20783
[2]General Dynamics Land Systems, 1231 Tech Court, Westminster, MD, USA 21157
[3]NASA Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA, USA 91109
[4]Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA 15213
[5]University of Pennsylvania, 472 Levine Hall, 3330 Walnut Street, Philadelphia, PA, USA 19104
[6]University of Washington, 185 E Stevens Way NE, Seattle, WA, USA 98195

Abstract: We present the design, integration, and evaluation of a full-stack robotic system called RoMan, which can conduct autonomous field operations involving physical interaction with its environment. RoMan offers autonomous behaviors that can be triggered from succinct, high-level human input such as "open this box and retrieve the bag inside." The robot's behaviors are driven by a set of planners and controllers grounded in perceptual reconstructions of the environment. These behaviors are articulated by a behavior tree that translates high-level operator input into programs of increasing sensorimotor expressiveness, ultimately driving the lowest-level controllers. The software system is implemented in ROS as a set of independent processes connected by synchronous and asynchronous communication, and distributed across two on-board planning/control computers. The behavior stack drives a novel platform consisting of a pair of custom, 500 Nm/axis manipulators mounted on a rotatable torso aboard a tracked platform. The robot's head is equipped with forward-looking depth cameras, and the arms carry wrist-mounted force-torque sensors and a mix of three- and four-finger grippers. We discuss design and implementation trade-offs affecting the entire hardware-software stack and high-level manipulation behaviors. We also demonstrate the applicability of the system for solving two manipulation tasks: 1) removing heavy debris from a

Received: 1 October 2020; revised: 29 May 2021; accepted: 3 August 2021; published: 15 June 2022.

Correspondence: Chad C. Kessens, DEVCOM Army Research Lab, 2800 Powder Mill Road, Adelphi, MD, USA 20783, Email: chad.c.kessens.civ@army.mil

DOI: https://doi.org/10.55417/fr.2022041

roadway, where 64% of end-to-end autonomous runs required at most one human intervention; and 2) retrieving an item from a closed container, with a fully autonomous success rate of 56%. Finally, we indicate lessons learned and suggest outstanding research problems.

Keywords: mobile manipulation, military applications, autonomy

1. Introduction

1.1. Motivation

Whereas the dream of robotics has been to assist humans by performing tasks that are dull, dirty, and dangerous, real-world applications to date have largely been confined to the dull. This shortcoming derives from significant challenges that remain in developing sufficiently robust capability for operating in uncontrolled environments, which are typical of the dangerous situations for which we would like to use robots as our proxies. Until those challenges are further resolved, strategies that shape the environment to "meet the machine halfway" can overcome some of the current deficiencies, but the dangerous tasks rarely afford that latitude. One path forward is using competitions such as RoboCup Rescue (Jacoff et al., 2003) and the DARPA Robotics Challenge (Krotkov et al., 2017) to drive researchers toward more robust solutions for challenging problems in disaster scenarios. This continued advancement also impacts military and law enforcement scenarios (Nguyen and Bott, 2001).

While the competition model energizes organic development of cross-functional teams to address these issues, the cooperative model brings its own complementary strengths to problem-solving. Starting in 2010, the US DEVCOM Army Research Laboratory (ARL) sponsored the Robotics Collaborative Technology Alliance (RCTA), a broad consortium of academic, industry, and government partners, to advance autonomous ground robotics by conducting leading-edge research to transform field robots from largely teleoperated tools into more autonomous teammates. Over its ten-year program, the RCTA nurtured numerous advances in the areas of perception, intelligence, human-robot interaction, dexterous manipulation, and unique mobility. These contributions include high-speed perception in rough terrain, situational awareness in unstructured environments, multimodal human-robot dialogue, and dexterous manipulation in the presence of clutter, among others. For this paper, we integrated those capabilities to execute missions requiring mobile manipulation in unstructured environments. By advancing autonomous capabilities, we intend to reduce the operator's cognitive burden, thus increasing the force-multiplication effects of these systems.

1.2. Related Work

In this section we discuss related work at the systems level. Mobile manipulation tasks require an extraordinary breadth and depth of expertise across many disciplines, each of which continues to advance actively. Because of this complexity, there is a great deal of related work associated with each of the individual research thrusts. Therefore, we have included some related work as appropriate to specific research components in Section 3, and encourage the reader to seek our more detailed papers on each of those components for a more comprehensive literature review in those areas. This section focuses on systems-level related work.

To identify an appropriate platform for integrating and testing our research advancements, we conducted a market survey early in the program. In particular, we wanted a system that could provide mobility on at least mildly rugged terrain with reasonable speed, deliver human-scale forces, be flexible enough to allow integrating multiple research technologies, and be robust enough for us to continue development over multiple years. We chose to exclude bipeds from our search to reduce the level of effort required to manage system stability. The results of our survey indicated that available systems suffered from at least one of four deficiencies, which would limit their usefulness for our purposes.

1. First, many platforms had insufficient speed and/or stability for operating in appropriate terrains. The military desires robots that can operate "at the speed of the fight" and generally

on highly rugged terrain. While the field is still working toward those goals, some consideration must be given to them when demonstrating technologies to military leaders. However, much of the cutting-edge work in mobile manipulation at the time was being done for home service activities. Perhaps the most notable robot used for such research was the PR2 (Bohren et al., 2011). Great work was also done using alternatives such as Herb 2.0 (Srinivasa et al., 2012), STAIR (Quigley et al., 2007), UBR-1 (Li and Fritz, 2015), TIAGo (Pages et al., 2016), and Fetch (Wise et al., 2016), among others. Because the home service environment is relatively flat, the mobile bases of these systems were wheeled and not designed for operating on rugged outdoor terrain. Thus, they were ill-suited for our aims.

2. Second, many platforms were too small or weak to perform at the human scale, even if they could handle the terrain. Two of the standard platforms used for military applications, the Packbot (Yamauchi, 2004) and Talon (Wells and Deguire, 2005), fell into this category. It was also difficult to find manipulators that could deliver human-scale forces with a reasonable mass and reach. Many standard manipulator packages at the time such as Baxter (Fitzgerald, 2013) could not deliver the required force, and industrial arms tended to be too massive. Even today's state-of-the-art commercial robots designed for less structured environments, such as Vision 60 (Akbari Hamed et al., 2020) and Spot (Bouman et al., 2020), are less forceful than humans.
3. Third, many platforms commonly used for research are not commercially available, but are "one-offs," developed by a particular institution for a specific need. This situation often means that systems either have not been through enough design cycles to provide the desired level of ruggedness, cannot easily be reproduced, and/or would have little support staff able to assist with troubleshooting problems. Examples include Robosimian (Karumanchi et al., 2017), Chimp (Stentz et al., 2015), Momaro (Schwarz et al., 2017), Centauro (Klamt et al., 2020), and Robo-Sally (Katyal et al., 2014). In these cases, true partnerships would need to be formed with the development teams to effectively utilize these platforms, and "out-of-the-box" performance could not be guaranteed without commercial quality control.
4. Finally, even commercially-available platforms could not provide guaranteed solutions. While one can reasonably expect a higher degree of performance robustness, many commercial products restrict access to low-level controls and are not as flexible or adaptive, which can make it challenging to develop and integrate new technologies, both hardware and software. Examples include the Packbot (Yamauchi, 2004), Vision 60 (Akbari Hamed et al., 2020), and Spot (Bouman et al., 2020).

To accommodate these shortcomings, we sought to take advantage of the scale, strength, and openness of the Jet Propulsion Laboratory's (JPL) Robosimian (Karumanchi et al., 2017), but overcome its speed limitations by adapting its limbs for use on a larger, modified Talon chassis. Both JPL and Qinetiq were RCTA partners, thus addressing the support concerns previously indicated. The solution also balanced the openness and robustness requirements, though the required integration effort remained substantial. We call the resulting system RoMan (derived from Robotic Manipulator). The program produced four RoMan systems for distribution across team sites to facilitate co-development and limit down-time for any necessary repairs.

1.3. Paper Outline

This paper describes the development, integration, and testing of the RoMan system. Section 2 provides details on the hardware and high-level software integration, including calibration. Section 3 details the many lower-level software component actions, including navigation, perception, object pose estimation, grasp planning, simulation, and manipulation planning. Section 4 then discusses testing around two notional missions: 1) clearing a roadway of a large, unknown debris object, such as a fallen tree, and 2) opening a known, hinged container to retrieve an unknown object of interest (e.g. laptop bag) from inside. Section 5 summarizes important lessons learned, as well as what we

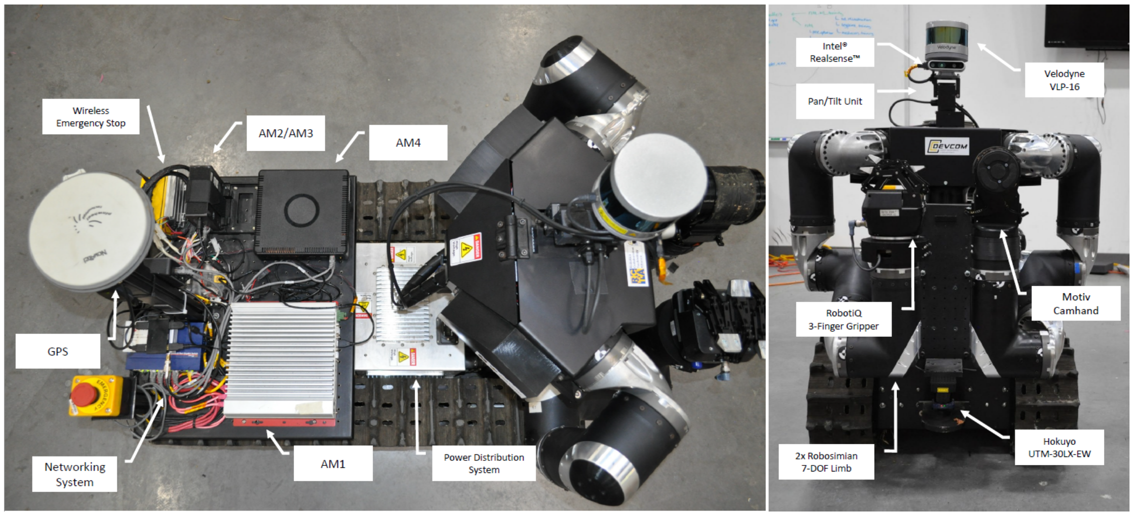


Figure 1. RoMan's major hardware components labeled. ©SPIE 2020.

see as critical open research problems. Finally, Section 6 concludes with a summary of contributions and a description of intended future directions.

2. Robot Description

We begin by describing RoMan's hardware, high level software architecture, and methods used for calibration.

2.1. Hardware System

To perform human-scale manipulation activities, a platform with near human size, strength, and dexterity was necessary. As such, RoMan has a tracked mobile base, two limbs with seven degrees of freedom (DoF) each, a one DoF torso, end effectors for dexterous grasping and power grasping, a multi-modal sensor head, and the computational hardware required for support. The following subsections describe those subsystems. Figure 1 shows the main platform hardware subsystems. For additional details, see (Kessens et al., 2020).

2.1.1. Mobile Base

A tracked mobile base was selected for improved traction and navigation on unpaved roads and soft ground such as sand, gravel, and soils by reducing ground pressure. QinetiQ NA provided a non-ITAR variant of their TALON® platforms to serve as RoMan's mobile base. The skid-steer drive train is powered by two independent electric motors, each connected to a 40:1 gearbox, for a total output of 182.4 N-m. A Gold Cello 50/100 (ELMO Motion Control) motor controller controls each motor in a master/slave configuration.

2.1.2. Manipulators

The design of RoMan's manipulators is based upon the Jet Propulsion Laboratory's Robosimian (Hebert et al., 2015) entry into the DARPA Robotics Challenge. These limbs provided both the required strength and dexterity for performing the desired tasks. The bases of the limbs are oriented 90 deg apart on a torso plate approximately 0.7 m from the ground, allowing the platform to reach objects that may rest on a shelf 1.5 m high. An additional actuator is used within the one DoF torso, allowing for manipulation and sensing on either side of the platform without reorientation. For manipulation efforts requiring precision grasps, the right manipulator is outfitted with RobotiQ's

3-Finger Gripper for small and medium sized objects (bags, light debris, fuel containers, etc.). For heavier objects and those requiring full body movement, the left manipulator uses Motiv Robotics' Camhand gripper, which is capable of utilizing the full strength of the manipulator system.

2.1.3. Sensors

A Velodyne Puck (VLP-16) and an Intel®RealSense™(D435) provide the primary visual information for the platform, including long and short range mapping, and information on objects to be manipulated and their surroundings. These are affixed to a FLIR (D47) Pan/Tilt Unit (PTU) mounted above the limbs to provide a higher perspective for long distance navigation and object handling purposes. An additional Lidar sensor (single line, Hokuyo UTM-30LX) is aligned with the front drive wheels' axis for low ground obstacle avoidance (i.e curbs, rocks). Platform orientation information is available using an Inertial Measurement Unit (IMU, Microstrain®3DM-GX5-25) located near the platform's geometric center, as well as a Global Positioning System (GPS) antenna (UBlox EVK-M8T). Finally, each manipulator is mounted to a 6-DoF Force/Torque sensor (ATI Mini-58) within the wrist to provide feedback for force-controlled activities.

2.1.4. Computing

The computational demands of the system can be categorized into 3 areas: Platform and SLAM (AM1, NUVO-5002LP), Intelligence (AM2, GB-BXi78550), and Perception (AM3/AM4, ZBOXQK7P5000). The Platform and SLAM system contains the controllers for the track base, limbs, manipulators, and pan/tilt actuation. For latency purposes, EtherCAT and USB communications are directly sent to AM1. AM2 processes the world model, including the various levels of panning and behaviors based on the collected sensor data from the sensor suite. The perception computational hardware (AM3/AM4) emphasizes raw sensor data processing towards classifying and detecting various objects, key-points, and other desirable attributes using neural network solutions.

2.1.5. Power System

RoMan's base contains the main power distribution and battery systems. The robot is powered by nine 9.9 Ah MIL-Spec Lithium-Ion batteries (Bren-tronics, BB2590) arranged in a 3s3p configuration, providing a total of 2200 Watt-hours of power. As each battery is self-contained, the platform can remain powered while each battery is removed for charging and replaced. During operation, the system's power consumption is dependent on the activity performed. Typical operation consisting of base platform movement and object manipulation allows for one to two hours of periodic untethered operation. However, the platform can also operate on shore power provided by a single power supply (Keysight, N5769A). The platform's main operating voltage is nominally 72-83 V, with bus voltages of 12 V and 16.5 V to support sensors and computational hardware, respectively. Additional hardware details can be found in (Kessens et al., 2020).

2.2. Software System

2.2.1. Computing Architecture

The RoMan platform utilizes a heterogeneous set of small form factor computers as mentioned in Section 2.1.4. This includes CPU and GPU resources allowing for all necessary computation to take place on-board the platform. An external laptop, connected via WiFi is used to send commands, inspect sensor data and analyze feedback from on-board algorithms. While this computer allows the operator to send goals to the robot and monitor actions, no computation necessary for RoMan to complete an action takes place off-board the platform. The architecture is organized using horizontal stratification, as different modules have been developed by individual institutions, allowing many modules to operate independently of other behaviors within the same level.

With the exception of low-level motor controllers and sensor drivers, RoMan uses modules communicating through the Robot Operating System (ROS) framework (Quigley et al., 2009). Each module is run as a ROS node, as represented in Figure 2. Communication between the higher

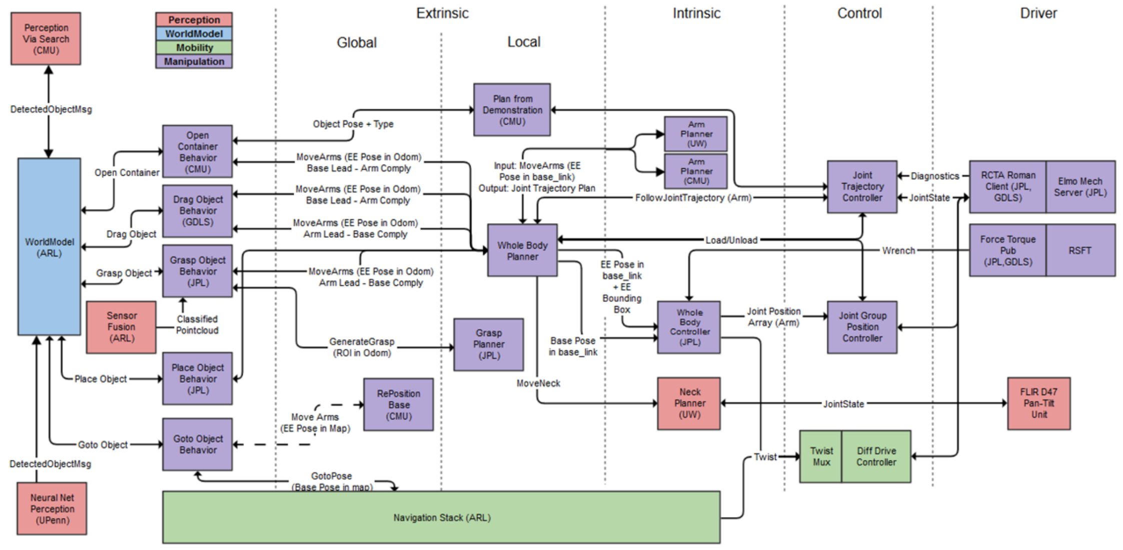

Perception Via Search (CMU)
Perception
WorldModel
Mobility
Manipulation
Extrinsic
Global
Local
Intrinsic
Control
Driver
DetectedObjectMsg
Plan from Demonstration (CMU)
Object Pose + Type
Open Container Behavior (CMU)
Open Container
MoveArms (EE Pose in Odom) Base Lead - Arm Comply
Arm Planner (UW)
Arm Planner (CMU)
Input: MoveArms (EE Pose in base_link) Output: Joint Trajectory Plan
Joint Trajectory Controller
Diagnostics
JointState
RCTA Roman Client (JPL, GDLS)
Elmo Mech Server (JPL)
Drag Object Behavior (GDLS)
Drag Object
MoveArms (EE Pose in Odom) Base Lead - Arm Comply
FollowJointTrajectory (Arm)
WorldModel (ARL)
MoveArms (EE Pose in Odom) Arm Lead - Base Comply
Whole Body Planner
Load/Unload
Wrench
Force Torque Pub (JPL,GDLS)
RSFT
Grasp Object
Grasp Object Behavior (JPL)
MoveArms (EE Pose in Odom) Arm Lead - Base Comply
EE Pose in base_link + EE Bounding Box
Sensor Fusion (ARL)
Classified Pointcloud
Whole Body Controller (JPL)
Joint Position Array (Arm)
Joint Group Position Controller
GenerateGrasp (ROI in Odom)
Grasp Planner (JPL)
Base Pose in base_link
Place Object
Place Object Behavior (JPL)
MoveNeck
RePosition Base (CMU)
Neck Planner (UW)
JointState
FLIR D47 Pan-Tilt Unit
Move Arms (EE Pose in Map)
Goto Object
Goto Object Behavior
Twist
Twist Mux
Diff Drive Controller
DetectedObjectMsg
GotoPose (Base Pose in map)
Neural Net Perception (UPenn)
Navigation Stack (ARL)

Figure 2. Software Block Diagram

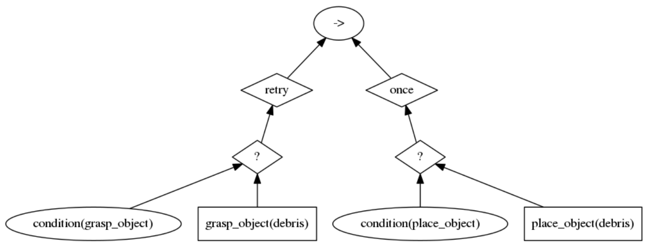
level nodes is conducted using ROS messages, while the lower level motor controllers use proprietary code developed by the Jet Propulsion Laboratory (JPL). The bridge between these two is provided by the RoMan client node, which exposes the motor controllers to the ROS system. Through this organizational scheme, the higher level behavior ROS nodes send goals to lower level planner ROS nodes, which send velocity commands through the RoMan Client to the limb and track motor controllers.

The highest level nodes consists of *Behaviors*. These behavior nodes interact with the World Model to detect relevant objects, collect information about those objects, and perform complex, high level manipulation planning to interact with the world. These behaviors utilize the ROS ActionLib stack to provide feedback to the World Model about object attributes learned by the behavior. These behaviors are run via behavior trees, which will be expanded upon in Section 2.2.2. To consolidate motion commands, all behaviors utilize the Whole Body Planner (WBP) for actions resulting in physical movement of the platform. Through a message to the WBP, a node can command the sensor head, arms, and/or tracks of the platform for manipulation and perception behaviors. The exception is for coarse base navigation (Section 3.1), which uses a separate planner. The WBP simplifies the integration of behaviors developed by various collaborators by providing a unified motion framework, and allows for interchangeability of motion planners. The WBP utilizes two limb motion planners. The first was developed at Carnegie Mellon University and uses the MoveIt! Motion Planning framework (Chitta et al., 2012). The second was developed at the University of Washington and uses the AIKIDO Motion Planning library[1]. The WBP is capable of issuing a goal to both of these planning frameworks, and arbitrating between the planned trajectories before executing the chosen path.

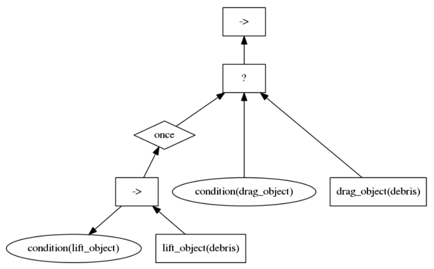
2.2.2. Behavior Tree

To compound multiple modular behaviors into a single task, the behaviors are sequenced through a "Mission Executor." The Mission Executor parses through a behavior tree representing the steps necessary to complete the overarching task, with appropriate failure modes. Each behavior is largely modular and does not depend on the completion of a single behavior before it can be run. A simple example of a behavior tree is shown in Figure 3a. This tree has two behavior nodes: "Grasp Object"

[1] https://github.com/personalrobotics/aikido



(a) Simple



(b) More Complex

Figure 3. Behavior tree examples.

and "Place Object." The Mission Executor reads the tree, and first checks the grasp object condition. Finding it to be false, the Mission Executor begins the behavior to Grasp Object called debris. Once the object is grasped, the executor moves on to the place behavior. Similarly, upon finding the place object condition to be false, it begins the place object behavior. This behavior has a decorator "once" telling the mission executor that if this behavior fails it should not be repeated, while other behaviors (e.g. grasp object) would default to repeating until they are successful. This "once" decorator is used for behaviors that should not be repeated, for example if the behavior may cause damage to the platform. Figure 3b shows a slightly more complex behavior tree example. This is an example of simple logic for moving an object of unknown weight. First the robot attempts a "Lift Object" behavior, and if that fails it is assumed that the object is too heavy to lift off the ground to move. The next option for moving that object is to drag it across the ground, and so the robot will continue to the "Drag Object" behavior (Colledanchise and Ögren, 2017).

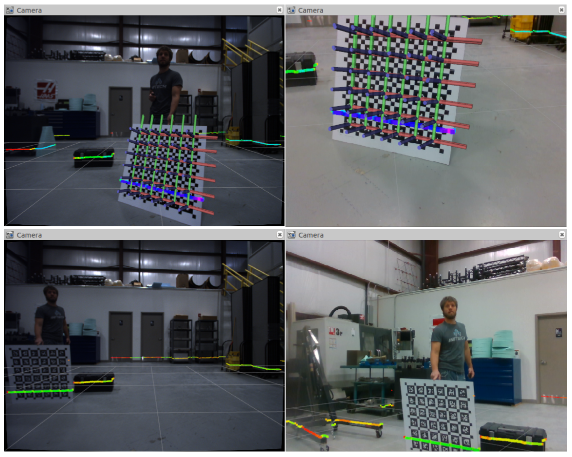
For the manipulation task, the Mission Executor synthesizes a number of individual modular behaviors into a single, robust mission. The use of the Mission Executor allows for easy integration of a wide variety of behaviors, which are either functions written specifically to interact with the Mission Executor, or using a wrapper to communicate the necessary information. This system allows for a single manipulation experiment to utilize eight distinct behaviors, without explicitly depending on past behaviors and with different error responses for each behavior. While certain behaviors may be set to repeat in the event of a failure, other failures are allowed to be fatal to avoid damaging hardware. The design modularity simplified testing, enabling behaviors to be run individually, or in smaller sub-mission sets. Additionally, behaviors can be easily replaced in order to test new methods and algorithms without affecting the rest of the behavior tree. The use of the RCTA Mission Executor simplified integration, testing and execution of experiments, while also increasing the robustness of mission execution.

2.3. Calibration

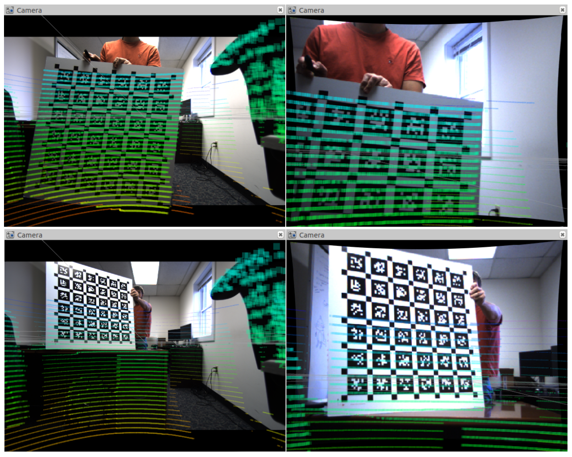
For a complex electromechanical system to perform well, it is critical that the system understands the relationships among its components. Therefore, we detail our calibration methods relating our sensory components to one another, as well as relating sensor head to the end effector.

2.3.1. Multi-Sensor Calibration

Multi-sensor calibration is an enabler for various downstream autonomy applications, including robust state estimation, depth-vision object detection, mobile manipulation, and directed perception. While online calibration is the ultimate goal of many researchers, the community still lacks generic approaches for performing multi-modal calibration across diverse sensor configurations. Multi-calibration systems typically support sets of one or two sensing modalities (Maddern et al., 2012), (Kümmerle et al., 2018), (Zhou et al., 2018), (Levinson and Thrun, 2013), (Pandey et al., 2014), (Scaramuzza et al., 2007), (Le and Ng, 2009), often with separate processes for each modality.



(a) RoMan



(b) Benchtop

Figure 4. Visualization of actuated calibration accuracy. For each sub-figure, the left column shows data collected by a static camera; the right column shows corresponding actuated camera data. Lidar scans, point clouds, and fiducial coordinate frames from both cameras are visualized to demonstrate effective data fusion.

Here, we present a unified graph-based representation that jointly calibrates any number of 2D laserscan, 3D point cloud, or color image sensors, with a particular emphasis on rapidly recalibrating *embodied* systems. We extend prior work (Owens et al., 2015) to support actuated cameras. Our approach is a target-based approach, which helps ensure accurate data association across sensors at the cost of requiring a planar target with visual fiducials. Similar to our work, (Pradeep et al., 2014) use calibration targets to calibrate a series of cameras and manipulator joints. Here, we also calibrate both the pose of an actuated sensor with respect to a kinematic chain, but also the pose of the root of the kinematic chain itself. Specifically, we define the following node types in the optimization:

- *Pose*: 6 Degree-of-Freedom (DoF) pose of a coordinate frame relative to the parent node frame. Fixed transforms, Sensor poses and Actuator poses derive from this type.
- *Actuator State*: 1 DoF joint configuration of a kinematic chain at a given time.
- *Observation*: The simple geometric primitive used to describe a calibration object detection. *Line* segments are formed by 2D laser scanner detections, *Plane* segments correspond to 3D point cloud detections, and *Point* detections represent the origin location of a detected fiducial tag.

We demonstrate the actuated sensor calibration on two mobile manipulator sensor configurations as well as a benchtop setup, and compare results with a benchmark calibration tool (Rehder et al., 2016). Visualizations of the data projection after calibration for a robot platform and the benchtop setup are shown in Figure 4. We use the benchtop configuration to compare the estimated transform between an actuated camera at a given joint configuration with a separate static multi-sensor calibration taken for the same joint configuration. Our calibration framework and a widely used camera calibration toolbox were used for static calibration.

Table 1 shows the results of comparing a static calibration at three different joint configurations with the expected calibration from a single actuated sensor calibration. The results show that for all poses, the pose difference between the static and actuated versions of our framework is within 1-3 cm in translation and 1-4 degrees rotation. For the second collection, the Kalibr toolbox produced a large difference between both the static and actuated versions of our software, which can be explained by the fact that Kalibr also optimizes intrinsic camera calibration parameters, while our framework uses pre-calculated intrinsic parameters. Table 1 also shows the difference between the estimated intrinsics for each trial, demonstrating the variation in intrinsic parameters for a single algorithm, suggesting more data are needed to properly optimize these parameters.

Table 1. Actuated versus static comparison. For three distinct actuation states, datasets for static calibration were collected and compared to an actuated calibration. The first two columns are rotation and translation differences from a static baseline calibration, MSG-Cal, (Owens et al., 2015) to another static calibration, Kalibr, (Rehder et al., 2016), and an actuated calibration. (Rehder et al., 2016) also estimates camera intrinsic parameters, which are shown in remaining columns, showing wide estimate variation for these critical parameters.

Experiment	Δr (deg)	Δx (cm)	f_x	f_y	c_x	c_y
Kalibr static	0.155	0.61	964.6	1010.3	977.4	607.9
MSG-Cal actuated	1.536	1.58	1052.0	1049.4	976.9	605.1
Kalibr static	10.892	6.76	1012.0	1055.9	977.9	594.1
MSG-Cal actuated	3.896	2.44	1052.0	1049.4	976.9	605.1
Kalibr static	0.103	0.52	963.8	1010.2	977.1	609.6
MSG-Cal actuated	2.590	1.27	1052.0	1049.4	976.9	605.1

2.3.2. Hand-Eye Calibration

In addition to calibrating the multiple sensor systems, we also needed to maintain calibration between the sensors and the actuators. For example, initial testing revealed that vibrations during base mobility could cause drift in the transform between the pan-tilt unit (PTU) holding the RealSense™. To correct for this, we implemented an automated hand-eye calibration (HEC) routine. Prior to the implementation of this module, we would have to manually ensure that the pointcould of the manipulator generated by the Realsense™would match up with the 3D model generated using the robot kinematics.

First, all joints from the pan-tilt unit to the end effector were individually calibrated in an effort to maximize the accuracy of the raw transform. We then attached an AprilTag in a known pose on the end effector so that the camera could accurately compute the transform independently from the linkage, based on the perceived pose of the AprilTag. The ROS TF tree with the robot's URDF was then used to compute the transform based on the linkage, and the two results were then compared.

Finally, we moved the end effector through a sequence of locations throughout the workspace, capturing effects not included in the kinematic model, such as imperfect manipulator rigidity. Transforms at each location were then converted into point clouds, and we used the Iterative Closest Point (ICP) algorithm to find the most appropriate transforms for operational use.

Though this approach helped us correct for the aforementioned drifts in transforms, it has a two major short comings. Firstly, we have no way to identify when the transforms have drifted. Hence, we have to perform the HEC routine at regular intervals, which leads to wasted time. Secondly, since the HEC routine requires the end effector to move through a sequence of locations stored in advance. This means that the robot workspace has to be free, when performing HEC.

3. Component Actions

3.1. Coarse Base Navigation and Localization

To enable reliable traversal of an unstructured environment, RoMan uses a hierarchical navigation system. This system generates trajectories to goal positions based on obstacle locations and platform localization provided by a metric simultaneous localization and mapping (SLAM) system. A local planner extracts trajectory segments within a fixed horizon and deviates as needed based on fast moving or unmapped obstacles. A local controller then sequences velocity commands to follow the prescribed trajectory segment. Together, these modules enable the platform to complete navigation sub-goals required to achieve its mission.

The SLAM system is used by RoMan to determine the pose of the robot in an absolute or relative reference frame, as well as to track the locations of obstacles in the environment. The system is based on the *OmniMapper* (Trevor et al., 2014), which collects *relative* measurements between subsequent locations along the robot's trajectory, as well as *loop-closure* measurements to prior locations when they are revisited. The measurements between subsequent locations consist of platform track odometry and iterated closest point (ICP) (Segal et al., 2009) alignment of 3D LiDAR sensor data at these locations for mapping. Loop closure measurements are made from ICP point cloud alignments using the current estimated trajectory as an initialization point. This set of measurements is organized in a nonlinear factor graph and optimized via GTSAM (Dellaert and Kaess, 2006), which computes the maximum-likelihood trajectory. LiDAR data is then rendered into a negative log-odds occupancy grid along this trajectory to accumulate a global costmap.

Navigation goals are generated by the behavior tree (see Section 2.2.2) to the navigation system in the form of *goto-region* goals. These goals specify a desired goal pose together with an arrival radius. The arrival radius allows the navigation system to make adjustments to the final pose to avoid coming into contact with any obstacles, which improves the reliability of the navigation system. The navigation system generates global plans from the robot's current location to a safe pose within the goal region, which are kinematically feasible via the search-based planning library (SBPL) (Likhachev and Ferguson, 2009). The global plan is re-computed during traversal as new map information is discovered.

Local segments of the global plan are evaluated by the local planner along with the most recent point cloud to prevent collision with fast moving obstacles not incorporated into the global map. This local segment is then sequenced to the motor controller as a series of track velocity commands. These components enable RoMan to navigate its environment to achieve its mission goals.

3.2. Object Recognition and Pose Estimation

In some instances, RoMan will be expected to interact with objects known to the system. Such interactions can be more efficient if the known information is used. Thus, we took two approaches to recognizing objects and estimating their poses: one based on neural networks, and the other based on search.

3.2.1. Neural Network Approach

Our neural network based approach to object recognition and pose estimation allows us to detect and localize objects from single RGB frames. Our pipeline first detects all instances of the desired object classes in the image. We estimate the pose of each object instance by detecting class-specific semantic keypoints.

Object Detection is performed using the maskrcnn-benchmark (Massa and Girshick, 2018) implementation of Faster-RCNN (Ren et al., 2015), trained with the RCTA dataset (Narayanan et al., 2020).

Keypoint-based Pose Estimation is performed by detecting semantic keypoints using a stacked-hourglass neural network. The error between the keypoints' rays and a deformable model of that object class is minimized to estimate the position and orientation of the object. This form of pose estimation was first proposed in (Pavlakos et al., 2017).

Training and Data Collection One of the primary limitations of our pipeline is the large number of keypoint annotations required to train our pose estimator for each new object class. This problem is especially challenging because the RCTA program had a large number of unique object classes that are not present in existing datasets. We mitigated this problem by generating an automatic labeling pipeline, which used an RGB-D scan of an object to create a three-dimensional model that could be labeled with keypoints. The labels could be projected from the three-dimensional model to the original two-dimensional images, drastically speeding up labeling. This pipeline is further described in (Narayanan et al., 2020).

3.3. Perception via Search Approach

Our PErception via SeaRCH approach or PERCH (Narayanan and Likhachev, 2016) estimates the 3-DoF pose (x, y, yaw) of an object of interest through a search for the best possible explanation of the observed scene in a space of rendered scenes. We used PERCH to estimate the 3-DoF pose of a known container with a hinged lid, which was fed to a manipulation planner that RoMaN used to open the container.

The task of accurately estimating the container pose is particularly difficult in the presence of high occlusions. The large size of container and its close proximity to the robot during manipulation, necessitated by planner and workspace constraints, occluded large parts of the container from the onboard Realsense camera. However, by rendering a set of candidate 3-DoF poses, PERCH is able to recreate such scenarios and find a rendered scene that corresponds closely to the observed scene. Specifically, PERCH minimizes the following objective function:

$$J(O_{1:k}) = \underbrace{\sum_{p \in I} \text{OUTLIER}(p|R_k)}_{J_{observed}(O_{1:K}) \text{ or } J_o} + \underbrace{\sum_{p \in R_k} \text{OUTLIER}(p|I)}_{J_{rendered}(O_{1:K}) \text{ or } J_r} \tag{1}$$

in which OUTLIER$(p|C)$ for a point p and a point cloud C is defined as follows:

$$\text{OUTLIER}(p|C) = \begin{cases} 1 & \text{if } \min_{p' \in C} ||p' - p|| > \delta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where δ is the sensor noise resolution, I is the input point cloud, $O_{1:K}$ are objects in the scene, R_k is a rendered point cloud with k objects, and J_r is the cost associated with a rendered point cloud corresponding to a given candidate pose. The set of candidate poses are generated by uniformly discretizing the (x, y, yaw) space and local ICP (Chen and Medioni, 1992) is used to refine every rendered pose to account for discretization artifacts.

3.4. Local Base Control

Base movement for manipulation was achieved using a technique separate from that described in Section 3.1 to perform short, precise movements to bring RoMan within grasping distance. While the SLAM based navigation provides robust movement over long distances with obstacle avoidance and localization, manipulation requires the ability to move to precise locations, close to objects which the platform cannot maneuver over. This Intrinsic Controller (IC) uses track odometry and IMU data for localization, and is not exposed to the map frame of the transformation tree, or the occupancy grid generated by the SLAM solution. As such, it does not avoid obstacles and is unsuitable for long distance navigation. This controller also has access to the wrench data being obtained by force-torque sensors in the wrist of both arms. Beyond approaching graspable objects, the IC is used to move when holding an object, whether it is a heavy object being dragged or a light object being lifted and moved to a different location.

Goals are sent to the IC in terms of the desired end effector location and a wrench threshold for the movement. The IC then calculates the necessary base location to achieve the end effector goal, and executes a base trajectory to achieve that goal. The wrenches on the end effectors are monitored throughout the movement, and any wrench exceeding the set threshold will result in a halt of the base movement and return a specific error noting the wrench threshold has been exceeded. This stops RoMan from damaging the end effector or arm, and can inform higher level planners of the attributes of an object being manipulated, based on how difficult it is to move.

3.5. Grasp Planning

3.5.1. Task Compliant Grasp Region Selection

When inspecting a manipulation workspace, the platform must be capable of inferring structure from the complex array of visual signatures with which it may be presented. This information must be intelligently utilized to plan grasps that are not only geometrically compliant with the task at hand, but are within the force capabilities of the system. An example of a manipulation scene is depicted as an RGB image in Figure 5 (left), where an aluminum truss, safety barrier, and 4x4 wood block rest atop a wooden pallet. In this example, while the yellow and black safety barrier (tipped

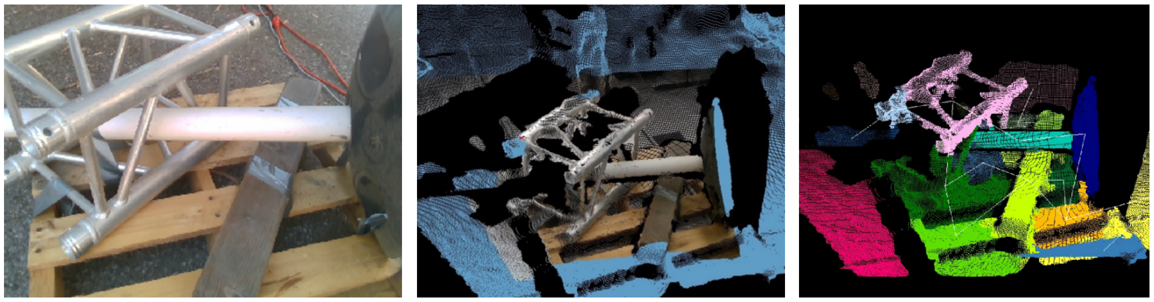


Figure 5. Left: An example of a debris pile RGB image containing a safety barrier, aluminum truss segment, 4x4 wood section, and pallet. Center: A point cloud of the same workspace captured with RealSense D435. Right: Object candidates for grasping singulated with geometric adjacency from the Locally Convex Connected Patches algorithm (Stein et al., 2014).

on its side) may present an obvious target to a naive grasp selection methodology, a human observer (possessing advance geometric reasoning) may identify that the safety barrier is wedged into the aluminum truss, rendering it more difficult to extract than the wood piece beneath it. Our aim, therefore, is to imbue the platform with the ability to similarly identify force interactions between objects within the scene, in order to choose suitable grasps that expedite disassembly of the pile.

As RoMan is intended to operate within complex unstructured environments, we begin by assuming that it possesses little to no *a priori* knowledge of either intrinsic or extrinsic properties of possible manipulands within its vicinity. This assumption necessitates developing an understanding of the geometric context of the scene utilizing only the sensory inputs available, being that of color imaging and 2.5D depth, as depicted in Figure 5 (center). The first step in this process of scene interpretation is to cluster individual depth elements (points within the pointcloud) into distinct objects, a problem often termed *singulation*, which in this implementation was afforded by the Locally Convex Connected Patches algorithm (Stein et al., 2014), with a singulation of the example workspace shown in Figure 5 (right).

Once a reasonable attempt to discriminate between objects within the scene has been made, the geometric regions representing grasp candidates that are furnished as a result must be selected between based on suitable criteria. Parallel work is presently investigating means of inferring the latent support structure within the pile through vision and iterative interaction, but for the purposes of this paper, a naive approach of selecting the topmost object along the line of gravity, and within reach of the selected arm, proved sufficient. This was achieved by designating a *priority point*, nominally at the origin of the shoulder coordinate frame for the arm that would be grasping objects, then finding the object candidates with the centroid closest to that priority point. Grasps were then synthesized using the pointcloud of that selected object, while treating points surrounding it as collisions.

3.5.2. Grasp Pose Synthesis

The second stage of the grasp planning process involves synthesizing mechanically stable hand configurations about the geometry of the target object. Despite accommodating the ability to grasp previously unseen objects, this is where we recruit *a priori* knowledge of the gripper to inform suitable placement locations on the object. This understanding is distilled into a library of 8 geometric object primitives and associated relative hand positions known to afford viable grasps, each pair of which forms a *grasp prototype*, such as those depicted in Figure 6. Points within the cloud representing the target object are sampled, and grasp prototypes randomly chosen from the library (or a subset) are fit to the surrounding points producing a “grasp score” developed in (Bowkett et al., 2021) via the kernel methods described in (Detry et al., 2017). This score represents the quality of the match between a given prototype and the object pointcloud for the sampled grasp pose.

Once a sufficient number of potential grasps have been sampled and their respective matching score calculated, they are checked for a range of failure conditions, including a viable approach

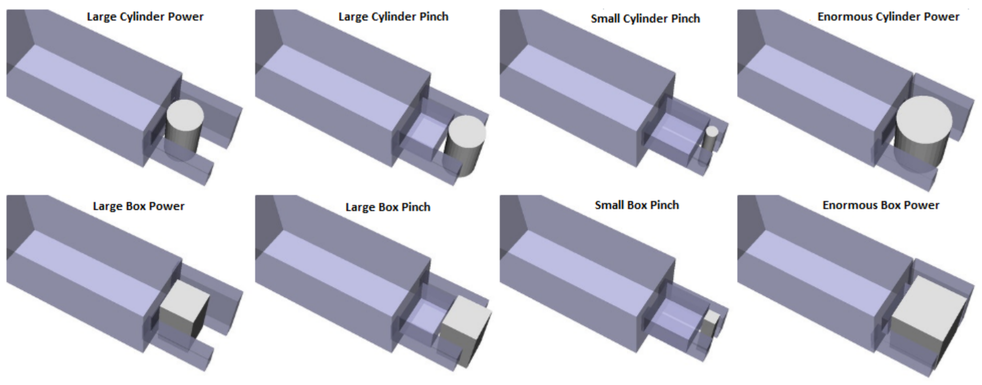


Figure 6. Hand positions (in gray) associated with representative geometric object primitives (in white). These pairs form sets of *grasp prototypes* that may then be fit to a given task scene to produce a set of candidate grasp poses. This range of shapes serves as an object agnostic base set, intended to accommodate common shapes, which can then be extended with affordance specific primitives such as a crate handle.

angle for the gripper (to prevent grasping from underneath), defined as an $\mathcal{R}^3$ approach vector and $\mathcal{R}$ deviation angle tuple; $\mathcal{R}^3$ position within the workspace; and collision with surrounding points in the cloud. If a candidate passes all of these conditions (which are in increasing order of computational complexity), it is checked for reachability with the platform specific arm kinematics, using an external library such as MoveIt! or AIKIDO. A vector of all grasps that pass this final check is then returned by the grasp planner, ordered via the grasp score that represents the match quality between the geometric primitive and the target object pointcloud.

3.6. Arm Planning and Simulation

3.6.1. Arm Planning

The manipulation planning component of our software pipeline receives goal arm configurations or grasp poses (generated as detailed in Section 3.5) as input for manipulating the environment. Our planning component is a *metaplanner* composed of two independent and parallel frameworks - one implementing Anytime Repairing A* (Likhachev et al., 2004) and another implementing Generalized Lazy Search (Mandalika et al., 2019). The two frameworks independently receive the goal position or pose and invoke motion planning algorithms to generate efficient and safe plans for the arms. Since ROMAN's environment is highly unstructured, this redundancy allows the robot to more robustly generate efficient plans in any environment. An Arbiter implemented as a ROS node receives plans from both the frameworks and returns to the joint controllers the first plan received.

Motion planning in continuous spaces is computationally hard. Therefore we represent the 8-dimensional space of the arm as a graph where the vertices represent joint configurations of the arm, and the edges connecting nearby vertices represent motions between joint configurations. We run `ARA*` and `GLS` to search the graph for the shortest feasible path in the available planning time. Path feasibility is evaluated by collision checking against a voxelized representation of the debris and obstacles around RoMan, while path optimality is determined by the length of the path.

`ARA*` is a fast anytime heuristic search algorithm that runs several iterations of `A*` on the graph with decreasing inflation in the heuristic. The algorithm terminates when the available search time is exhausted or when the optimal path on the graph has been determined. `ARA*` efficiently reuses search efforts from previous iterations and continuously lowers the sub-optimality bound over the solutions it returns by decreasing the inflation in the heuristic. Our implementation is based on the Search-based Motion Planning Library (SMPL)[2] and MoveIt! (see Section 3.6.2).

[2] https://github.com/aurone/smpl

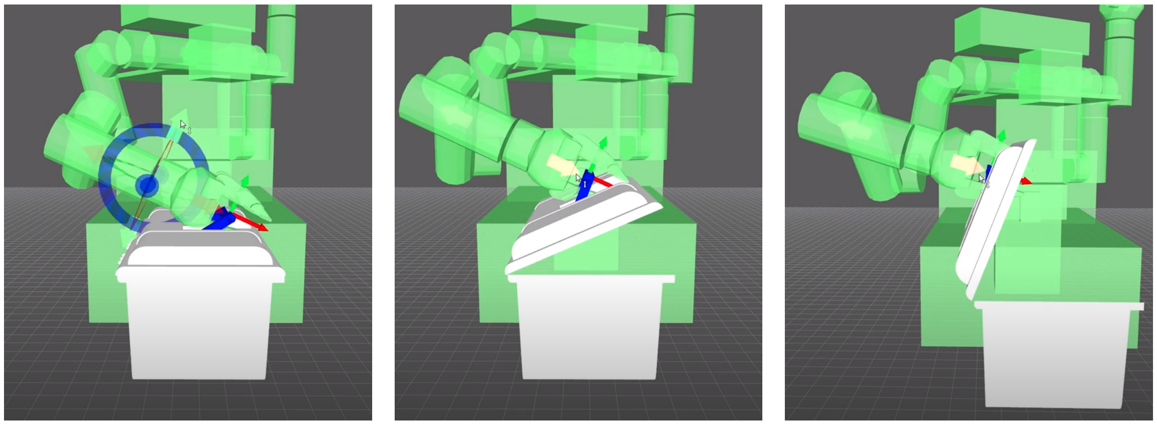


Figure 7. The RoMan robot performing a container opening task in simulation.

`GLS` is a computationally efficient, lazy shortest path planning algorithm. Evaluating the collision status of edges on the graph is a major computational bottleneck in robot motion planning. `GLS` delays collision checking until absolutely necessary. `GLS` operates with the key insight that *interleaving* lazy search and collision checking reduces the total planning time stemming from search efforts (such as vertex expansions) as well as collision checks. We use the open source implementation of `GLS`[3] and AIKIDO (see Section 3.6.3).

The amount of clutter in the environment and the position of the obstacles relative to the initial manipulator state affect the planners' relative performance. Since both planners forward propagate the search tree, GLS is affected by the obstacles in the number of edge evaluations and graph operations it needs to perform before terminating. ARA* is affected by how close the weighting heuristic is to the true cost. In general, we observed that as the amount of clutter in the environment increased, saving on edge evaluations allowed GLS to outperform ARA*, while ARA* outperformed GLS in sparser environments since edge evaluation did not involve as many complex geometric tests (i.e. edge evaluation was cheaper and GLS was optimizing for it unnecessarily).

3.6.2. Simulation: MoveIt!

In our pipeline, we used simulation tools for the following two problems -

1. Setting up the system in simulation to exhaustively test the planner before employing it on the real robot.
2. Generating demonstrations in simulation, and further using these demonstrations for manipulating articulated objects.

Using simulation for system testing - For setting a reliable test pipeline, we developed software support that enabled us to run the motion planning algorithms used on RoMan in simulation. Using the ROS + MoveIt! (Coleman et al., 2014) framework, we set up a simulation of the robot performing the container opening task. This simulation was visualised using the RViz framework (Kam et al., 2015).

Generating demonstrations from simulation - The Experience-graph based planner has the ability to use demonstrations for executing a given task. Because RoMan's 7 DoF arm is not compliant, it was not possible to use kinesthetic demonstration. Therefore, the demonstration that we used for motion planning was generated in simulation. As shown in Figure 7, the end-effector is moved using a computer mouse in the simulator to demonstrate a container opening maneuver. This demonstration is then embedded into the Experience-graph, as explained in Section 3.7.1.

[3] https://github.com/personalrobotics/gls

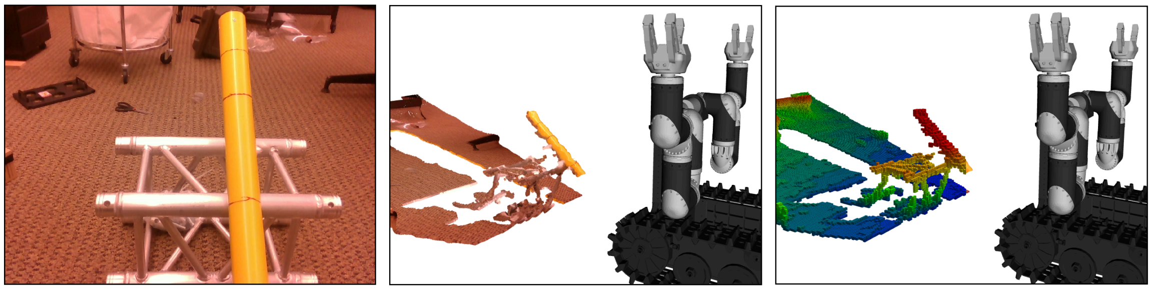


Figure 8. An illustration of ROMAN's view of its environment (left), the camera input visualized in simulation (middle), and the voxelized representation necessary for obstacle avoidance (right).

3.6.3. Simulation: AIKIDO

Manipulation planning in unstructured environments requires a tight integration of real-time perception data, motion planning algorithms to generate safe and efficient motions, and controllers that command the robot actuators. We use the C++ library AIKIDO (Srinivasa et al., 2020) for integration.

Kinematics, Dynamics and Safety. AIKIDO (Artifical Intelligence for Kinematics, Dynamics and Optimization) is tightly coupled with DART (Dynamic Animation and Robotic Toolkit) (Lee et al., 2018b), a physics simulator used to perform kinematics and dynamics computations for motion planning. The simulation platform is also used to perform collision tests to generate safe motions of the arms. Since ROMAN operates in an unstructured environment, we use DART to interpret dense point cloud data obtained from the sensors as a voxelized representation, as seen in Figure 8, to perform fast collision tests during motion planning. AIKIDO uses the ROS framework to visualize simulations in RViz (Kam et al., 2015).

Motion Planning. AIKIDO provides an interface to the Open Motion Planning Library (OMPL) (Şucan et al., 2012) planners. In our experiments for manipulation planning, we use Generalized Lazy Search (GLS) (Mandalika et al., 2019), an optimal and efficient motion planning algorithm implemented as an OMPL planner. The planner searches for the shortest path on an explicit graph in the 8-dimensional configuration space of the arms. The vertices in the graph represent possible configurations of the arms, and the edges represent motions between configurations. We use AIKIDO and DART to generate a graph free of self-collisions with the robot, and AIKIDO's OMPL interface to GLS to search for collision-free paths online. Further details on the planning algorithm are provided in Section 3.6.1.

3.7. Manipulation

3.7.1. Manipulation of Articulated Objects

To manipulate articulated objects, RoMan utilizes a graph-search based motion planning algorithm known as Experience-graphs (E-graphs) (Phillips et al., 2012). This algorithm can make use of past experiences or demonstrations to speed-up the search. We provide RoMan with user generated demonstrations that it utilizes to manipulate articulated objects.

On RoMan, the E-graph algorithm is demonstrated on a container opening task. The object manipulation module coordinates the operation of the limb and gripper through the container opening task space via a motion planner equipped with the E-graph algorithm. The container, which is placed in front of RoMan, needs to be opened to retrieve a bag located inside this container. To do this, the state of the container lid (i.e. its angle relative to the container body) is initialized to zero and added to the motion planner state space. The object manipulation module takes as input the 6D pose of the container and a user generated demonstration that takes the lid of the container from a "closed" state (where the angle between the lid of the contained and the ground plane is 0 degrees) to an "open" state (where the angle between the lid of the contained and the ground plane is greater than 90 degrees). The object manipulation module then uses the E-graph motion planner to generate a series of limb trajectories interleaved with gripper commands. These trajectories and commands are executed open loop, without remapping the low-level object perception to the open state.

Experience graphs - An experience graph, G_e, contains a set of previously planned paths. The original planning problem is represented using a graph G. The heuristic function H_e is computed such that the planner is biased towards picking edges from G_e, than from G. It does so by explicitly penalizing the planner when an edge from G is chosen, rather than from G_e. Formally, this is stated as

$$h^E(s_o) = \min_{\pi} \sum_{i=1}^{\lambda-2} \min\{\varepsilon^E h^G(s_i, s_{i+1}), c^E(s_i, s_{i+1})\} \tag{3}$$

where $h^E(s_o)$ is the E-graph heuristic for a state s_o, h^G is the heuristic function for G, π is a path $\langle s_0 ... s_{\lambda-1}\rangle$ and $s_{\lambda-1} = s_{goal}$, c^E is the cost of an edge in G^E, and ε^E is a scalar ≥ 1. The benefit of using edges from G_e is that it reduces the amount of exploration of G, which is typically a much bigger graph then G_e.

Demonstration-based Experiences - The E-graph framework can be used for incorporating user-generated demonstrations in G_e. Since the demonstration shows how to manipulate a given object, another variable is added to the state-space that tracks the state of the object being manipulated. As a result, both G and G_e are updated to account for this added dimensionality in the state-space. In addition, the heuristic, H_e also has to be updated to account for the added dimensionality. In the rest of this section, we briefly describe how the user-generated demonstration is embedded into the E-graph, and how resulting updates are made to the state-space and the heuristic function.

The original planning problem is represented using a graph $G = \langle V, E\rangle$, where each vertex $v \in V$ represents a robot state, and edges $e \in E$ connect neighboring robot states. The user-generated demonstrations are denoted as $D = \langle T_1 ... T_m\rangle$, where each T_i is a set of discretized trajectories corresponding to the i^{th} object in the environment. $T_b = \{\langle a^b_{11} ... a^b_{1k}\rangle ... \langle a^b_{l1} ... a^b_{lk}\rangle\}$, where $a^b_{ij} \in T_b$ is the j^{th} point in the i^{th} trajectory for object b, and $a^b_{ij} \in \mathbb{R}^{\mathrm{n+1}}$. The extra dimension corresponds to the state of the object being manipulated, which we term z. We represent the full state of the robot as $coord(v) \in \mathbb{R}^{\mathrm{n}}$, and the state of the object b at a^b_{ij} as $zcoord(a^b_{ij}) \in \mathbb{R}^{\mathrm{n+1}}$. There is no prior model for any of the objects the robot interacts with.

To account for the new dimension z, we create a new set consisting of the states the robot can possibly be in. This set contains the vertices in the original graph, but repeated for each possible value of the new variable z. We call this set V_{orig}. The states in the demonstration may not necessarily fall on any state in the original graph. Hence, we construct a new vertex set V_{manip}, which contains states from V_{orig} and all the states from the demonstration.

Due to the additional dimensions, the connectivity of the graph also changes. The new edge set E_{manip} is a combination of edges from the original graph E_{orig} (replicated for each value of z), edges that come from demonstrations E_{demo}, "bridge edges" E_{bridge}, and "Z-edges" E_z. Bridge edges connect demonstration states to states in the discretized original graph. To make such a connection, we check if (1) the two states fall into the same discretized bin, and (2) if they share the same z-value (i.e. the manipulated object must be in the same state). Z-edges generalize the demonstrations to create edges on the object's constraint manifold that may not have existed in the demonstrations. Based on this new graph, we introduce a heuristic function that guides the planner to modify the object towards the goal configuration. The heuristic function estimates the cost of getting the robot in contact with the object plus the cost of manipulating the object so that the variable z moves through all the required values to become z_{goal}. For further details about the planner, see (Phillips et al., 2016).

3.8. Manipulation of Heavy Objects

One potential mission concept of interest is to support larger vehicle mobility by removing obstructions from roadways. Many of the objects that may be encountered, such as the "czech hedgehog" anti-vehicle barrier, are of sufficient mass that to lift them directly with the robotic gripper would exceed the force or torque they are designed to sustain, potentially damaging the system. Such objects may be maneuvered by only partially supporting their weight, allowing portions to remain resting upon the ground, while guiding their motion by pushing or dragging. Even with such accommodations, the simple task of raising one section of an object to affect a dragging motion imparts significant torque on the gripper, if its orientation is kept rigid during an upward Cartesian motion.

To reduce the torque experienced by the gripper during the lifting motion, we can allow the orientation of the gripper to deflect forward as the centroid moves upward, twisting the grasp in unison with the object as it rotates about the remaining ground support(s). This is termed *wrench-reactive compliance*, and amounts to applying software compliance to the mechanically rigid system, specified in anisotropic fashion so as to permit select axes to comply while others remain fixed along the specified motion path. The compliance is defined in a hybrid frame (denoted hyb), which is located at the origin of the end-effector frame (ee), but whose orientation is set to that of the platform base ($robot$), so as to prove intuitive to a human operator. For example, by specifying a non-zero torsional compliance about the Y-axis (in FLU coordinates), the motion controller will only allow rotational deflection about the Y axis during a Cartesian motion. The deflection is calculated by transforming the force torque pair, or *wrench* W, measured at the wrist into the aforementioned hybrid frame through equation 4, where Ad_{b2a}^T is the transposed *adjoint transformation* of a wrench from frame a to frame b (Murray et al., 1994), and $R_{robot2ee}$ is the rotation between the instantaneous end-effector frame and the robot base frame.

$$Ad_{hyb2ee} = \begin{bmatrix} R_{robot2ee} & 0 \\ 0 & R_{robot2ee} \end{bmatrix}, \quad W_{hyb} = Ad_{hyb2ee}^T Ad_{ee2ft}^T W_{ft} \ . \tag{4}$$

The anisotropic compliance is then specified via a 6x6 diagonal matrix, C_{hyb}, which converts the hybrid frame wrench into a deflection in *twist* coordinates, being the dual space of wrenches (Murray et al., 1994). This may then be transformed back into the frame of the end-effector so as to produce the deflection introduced by the compliance $G_{ee2ee_{defl}}$, as is calculated via

$$C_{hyb} := diag(c_{f_x}, c_{f_y}, c_{f_z}, c_{\tau_x}, c_{\tau_y}, c_{\tau_z}) \ , \quad T_{hyb}^{defl} = C_{hyb} W_{hyb} \tag{5a}$$

$$T_{ee}^{defl} = Ad_{ee2hyb} T_{hyb}^{defl} \ , \quad G_{ee2ee_{defl}} = e^{\hat{T}_{ee}^{defl}} \tag{5b}$$

Example compliance factors used during lift of an anti-vehicle barrier are 0.2 rad/Nm in Y rotation, 0.005m/N in X translation; greater detail on the approach is given in (Bowkett et al., 2021). The ATI wrist force-torque sensor is used to affect the above compliance, in addition to triggering safety stops to prevent mechanically damaging wrench, and detecting contact when a "move to grasp" behavior is invoked.

4. Integrated System Testing

Having fully integrated the hardware and software systems onto RoMan, we now demonstrate and evaluate the platform's ability to perform two different tasks autonomously. First, we attempt to remove a heavy object blocking a notional road to facilitate the continued movement of a vehicle convoy. Second, we attempt to retrieve an item from a closed container. Each scenario relies on the successful execution of the full stack of services. When failures occur, we assess the root of the failure, which can be in the interplay between components and not just the individual components themselves.

Note that the experiments in this project were not generally set up for formal statistical analysis as their purpose was largely to establish baseline primitives for autonomy. The graphs and tables

Table 2. Baseline tele-operable actions

Actions	Tele-Operable (Y/N)	Avg. Execution Time	Description
Goto Object	Y	1min 22s*	Approach/coarse navigation
Pre-manip Object	N	10s	Prep for manipulation
Look	Y	$\leq$1s	Sensor head control
Grasp	N	26s	Grasp planning and execution
Drag	Y	12s	Local base control
Move Arm	N	5s	Arm planning

*Distance from start to target was roughly 20 meters.

given are therefore to be taken as descriptive guides to performance. In two cases, namely Table 3 and Table 5, we can state statistically significant findings, given the reasonable assumption of independent trial runs.

4.1. Heavy Debris Removal

A debris clearing task within a vehicle convoy scenario demonstrates many of RoMan's strengths and facilitates a better understanding of human-scale manipulation. In this scenario, a soldier instructs RoMan to advance ahead of the team to unidentified debris. RoMan receives an *a priori* location of a general area containing debris and proceeds to navigate toward this location. Once RoMan is within the *debris region*, various aforementioned detection techniques are used in an attempt to identify objects (see Section 3.2). An identifiable object may include items such as a *Czech hedgehog*, which is a standard anti-mobility device. Identification of such objects can speed up the grasping process due to its known properties. Other debris objects, such as a fallen tree, are considered unknown. Subsequently, RoMan performs a *pre-manip* routine, grasps the object, determines if it is light enough to move freely or too heavy and must be dragged, and manipulates the object accordingly.

Even for a human, removing heavy debris is challenging. RoMan's execution requires minimizing adverse effects on the mission and potential human teammates; therefore, robustness and speed are critical. Robustness involves both software and hardware considerations, such as providing subsystem loop-closures and enforcing limits to prevent overtaxing actuators. Mobile manipulators are notoriously slow, even when tele-operated.

To provide a baseline with which to compare RoMan's autonomous functions, we ran a series of experiments under human tele-operation. Table 2 shows which capabilities were able to be tele-operated. During our baseline trials, autonomous-only capabilities had dynamically configurable parameters which were set on-the-fly as needed. These parameters fine-tune the behaviors and required discernment based on past experience.

Currently, several of the decisions RoMan makes are predetermined, including limb selection, grasp priority location, grasp prototypes, lift distance, and drag distance. We chose reasonable configurations based on the baseline trials. Future work may rely on the Mission Planner to arbitrate these decisions. The next section describes our baseline setup and debris pile arrangement in more detail.

4.1.1. Experimental Setup

The heavy debris consisted of a single tree branch or Czech hedgehog blocking a path. The Czech hedgehog was approximately 18 kg and 1.3 cubic meters in volume. The tree branch spanned a minimum of 3 meters, with multiple branches protruding from the trunk (see Figure 9). The characteristics of either object's graspable point only needed to conform with the Camhand gripper specifications, such as friction grip payload and stroke range. Other objects, such as traffic barriers/walls and barrels were used as immovable structures to constrain the path.

The key metrics from this experiment were the number of human interventions and the emergency stop (E-stop) count. Human interventions are performed by an experienced RoMan Operator, and are limited to tele-op maneuvers to skip over non-recoverable behaviors, on-the-fly behavior

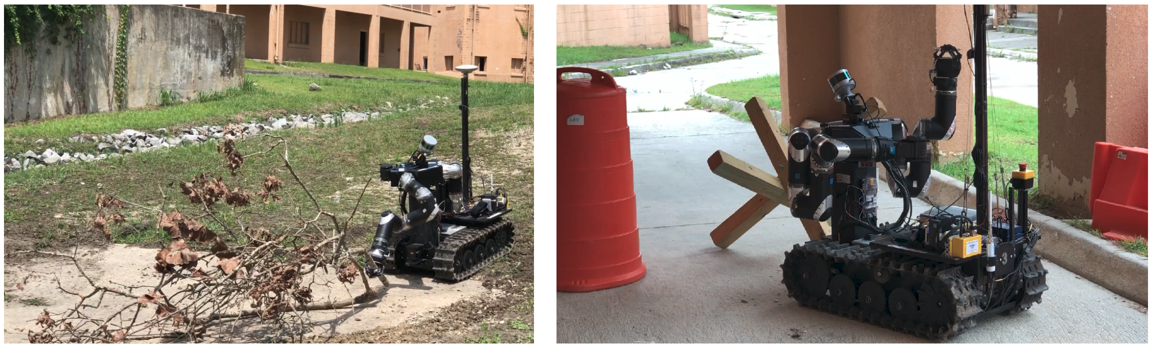


Figure 9. Left: RoMan dragging a tree branch. Right: RoMan in front of a Czech Hedgehog.

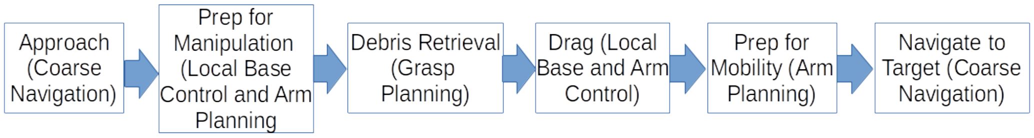

Approach (Coarse Navigation)
Prep for Manipulation (Local Base Control and Arm Planning
Debris Retrieval (Grasp Planning)
Drag (Local Base and Arm Control)
Prep for Mobility (Arm Planning)
Navigate to Target (Coarse Navigation)

Figure 10. Behavior pipeline for the Debris Clearing Sequence.

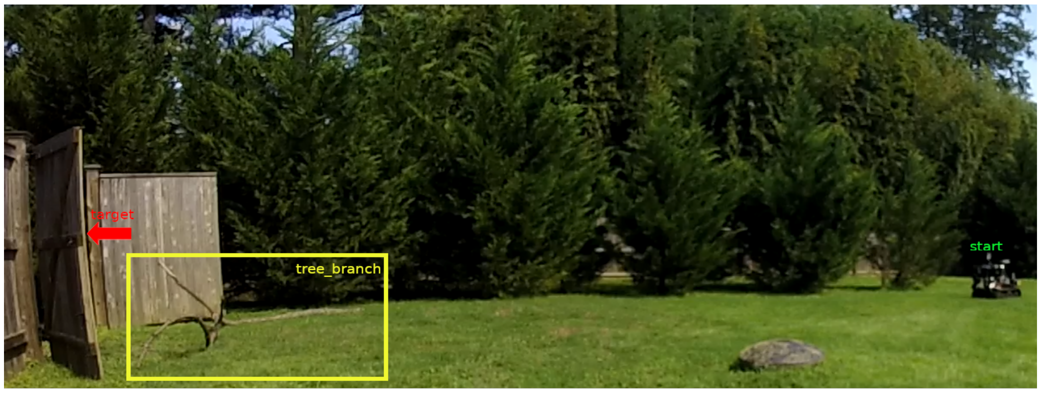


Figure 11. Experimental setup showing start and target location.

parameter tuning/calibration, and/or resetting hardware driver nodes (e.g. Camhand drivers, RS-Limb servers, etc.). Tele-op maneuvers were performed using an XBox wireless controller. Parameter tuning and calibration changes were made using RQT, a ROS GUI with a number of plugins. The Linux terminal interface was used to monitor the status of the system, and to reset any hardware drivers or other issues. E-stop was engaged for any unexpected behaviors and *limb elmo mech server* failures. By design, an E-stop also powered down RoMan's grippers.

For the experiments, RoMan started at least 30 meters away from the debris pile. The test sequence then follows: (1) navigate to *debris region*, (2) move base within RoMan's task space using an inverse reachability lookup for the closest object in front, (3) move limbs from travel posture to a preset stow posture, (4-5) adjust sensor head and base, (6) move limbs from stow posture to preset pre-grasp, (7) grasp planning and execution, (8) lift using *wrench-reactive compliance*, (9) drag object a predetermined distance, (10) lower, again using *wrench-reactive compliance*, (11) move base back a predetermined distance to accommodate subsequent limb motions, (12-14) open gripper, sequence through preset limb postures to reset to travel posture, (15) navigate past previously blocked path. Steps (1-5), (9), (11), and (15) were tele-operated for the baseline trials. A condensed version of the test sequence is shown in Figure 10. We also wish to note anecdotally that, while we did not actively control lighting conditions, successful experiments were performed indoors and outdoors at varying times of day.

Special considerations were taken for the baseline trials since the tele-operator has certain advantages in terms of situational awareness. For example, having a third person point-of-view of the test run. Thus, we limited the tele-operator to only PointCloud data from the Velodyne, Hokuyo laser scans and RealSense imagery since these are used by the navigation stack. Furthermore, autonomous control speeds for the base and PTU sensor head were set to the maximum tele-operable speeds of 0.3 meters per second and 0.3 radians per second, respectively.

4.1.2. Results

A total of 25 end-to-end autonomous runs and 10 baseline tele-operated runs were conducted. Table 3 shows the percentages of the runs in which human interventions were needed to complete the mission. The number of interventions ranged from zero to at most three. We expected potential hardware failures requiring reset, which contributed to a majority of interventions in both our baseline and autonomous trials. Precarious gripping and inherent platform shaking attributed to objects slipping out or breaking during dragging. This source of intervention could be minimized with *Mission Planning*, which was not available during these experiments. We were checking for failure conditions using various sensors, and *Mission Planning* would have used this feedback to determine the best recovery method. Secondly, hardware robustness over time has affected the intervention count. Actuators such as limb and track motors potentially see more failures as more energy is used up. At lower power budgets, brownouts have occurred. Loss of calibration also affected behaviors such as navigation and grasping. For example, the PTU on RoMan regularly changed its zero crossing due to hardware slipping. During the experiments, we discovered about a 20% offset in the wheel odometry. This contributed to poor state estimation and loop closures during navigation. Lastly, errors at the behavior level, such as incorrect ROI selection and infeasible grasping approach axes, required manual correction via ROS dynamic reconfigure.

From a statistical significance standpoint, Table 3 shows no interventions in 50% of 10 Baseline runs (i.e. 5 'successes' in 10 runs), and no interventions in 16% of 25 Autonomous runs (i.e. 4 'successes' in 25 runs). If we take the probability of such 'success' to be 0.5 for non-autonomous runs as the rule, it is of interest to test whether autonomous runs have a significantly different success rate given the data. Here, our one-side null hypothesis is that autonomous runs have a *geq* 0.5 probability of success of non-intervention versus the alternative hypothesis of < 0.5. We apply the *binom.test* of R (Team et al., 2013) (an 'exact' test based on binomial probabilities), and find that the probability of this null hypothesis being true is 0.00004, in which case we clearly reject the null hypothesis and accept the alternate: Autonomous runs need more interventions. This seems self-evident even without a test, but it's worth considering the formal results which also gives a useful 95% confidence interval for success in autonomy of (0, 0.33), or 0% to 33%.

While our results show the success rate with 0 human interventions well below the baseline, performing the same comparison for all missions allowing at most one intervention indicates that the autonomous operation outperformed the baseline 64% to 60%. That trend continued for higher numbers of interventions. Note that most of the interventions were due to hardware failures, which required a simple driver node reset. A breakdown of intervention types is shown in Table 4.

Figure 12 shows the success rates for each of the behaviors in the debris clearing sequence. The behavior success is determined by whether or not the robot meets the criteria for the underlying actions to return success. Typically, if the behavior fails, a human intervention is required to advance

Table 3. Percentage of Human Interventions

Mode	**Number of Interventions**			
	0	≤ 1	≤ 2	≤ 3
Baseline	50%	60%	90%	100%
Autonomous	16%	64%	92%	100%

Table 4. Percentage of Intervention Types

Mode	Types of Interventions		
	Reset	Parameter	Teleop
Baseline	33%	66%	-
Autonomous	53%	7%	40%

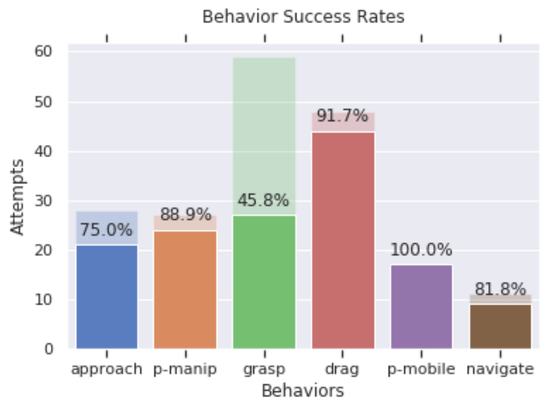


Figure 12. Success Rates by Behaviors

the mission. The re-execution of the behavior is recorded as additional attempts. Non-recoverable failures will result in cease of attempts of subsequent behaviors and mission failure. On average, the elapsed mission time for baseline and autonomous runs are 3m 26s and 2m 43s, respectively. This is a 22% increase in execution speed for autonomous missions which is attributed to about a 4 second delay for each action needing human initiation. However, navigation in particular was always faster through teleop. Recovery through human interventions increased cycle times on average of 15 seconds for resets, 10 seconds for parameter changes, and 4 seconds for teleop. A breakdown of average execution times for each action is shown in Table 2.

Grasping proved to be the least robust behavior. Much of this was due to the gripper overheating, causing grasping execution to fail. A few less obvious reasons include the gradual drift in the PTU calibration, which mounts the RealSense camera, and non-ideal preparation for manipulation poses, which set RoMan up for unreachable grasps. Although grasp had the highest failure rate, it was exercised the most, followed by the drag behavior. This made sense because during drag, the branch would sometimes slip out, and another grasping attempt would follow. In certain cases, we declare a complete mission failure, and many behaviors such as prep for mobility and/or navigate to target would not be attempted. Examples of non-recoverable failure cases include prolonged debugging and/or replacing damaged hardware.

4.2. Container Opening and Bag Retrieval

A second manipulation scenario was created to highlight the more delicate capabilities of RoMan and the RCTA software stack. In this scenario, RoMan locates a known container, opens it, retrieves an unknown item from within, and delivers the item to a specified location. However, the manipulation task requirements and workspace constraints of the manipulator limited admissible base goal poses such that the non-holonomic navigation was unable to reliably position the robot for the task. Therefore, for the purposes of these experiments, RoMan started with the container adequately within its workspace.

The size, shape, and operation of the container is the only prior information available to the robot. The container pose estimate relies on a search for the known size and shape, and the container opening algorithm must understand how the lid is hinged to be able to open it properly - this is captured as the object state, the lid angle relative to the container base, in the articulated object manipulation planner's e-graph, see Section 3.7. However, the placement of the container within the workspace is not known, and the robot has no a priori information about the item it retrieves from within the container.

This scenario was implemented as a manipulation pipeline within the RCTA software stack. Operation of the pipeline was governed by the RCTA Mission Executor following a container opening and item retrieval behavior tree.

4.2.1. Experimental Setup

The first pipeline module, Container Pose Estimation, was handled by a search-based perception algorithm called PERCH (see Section 3.3). A ROS interface was created between PERCH and the RCTA Mission Executor packages, allowing for simpler parameter adjustment and consistent control and data messaging. Additionally, the source code for PERCH was optimized to run on the computing resources available on the RoMan platform by filtering the input point cloud and reducing the point density. These modifications result in a version of PERCH capable of rapidly determining the pose of the container within the accuracy required for the object manipulation planner.

Early setup runs revealed that from position to position, there was a varying offset in the container pose along the handle axis, possibly due to point cloud inconsistency at the back of the container. A ROS parameter was added to adjust the reported container center and account for this inconsistency, which sometimes resulted in damage to hardware. The PERCH pose was also checked visually via rviz during every run to prevent this. However, unless the pose was obviously catastrophic, RoMan was allowed to attempt to open the container.

During testing human interventions are used if the RoMan was in a potentially hazardous position, or if a failed step can be recovered to complete the run. When a step has failed and been recovered by human intervention the entire run is marked as a failure, and the step which caused the failure is recorded. Interventions are performed by an experienced RoMan operator using a terminal interface and RViz tools.

Additionally, the grasp executor parameters were modified to facilitate container opening. First, trial and error identified a set of grasp primitives that favored consistent item retrievals while balancing processing time. Then, the allowable limb approach cone was modified to better suit the item retrieval task, as RoMan must be able to grab the item from inside the container while avoiding the open lid and sides. Fortunately, these modifications did not affect performance, as the grasp planner was consistently able to find over 60 grasp candidates within the container.

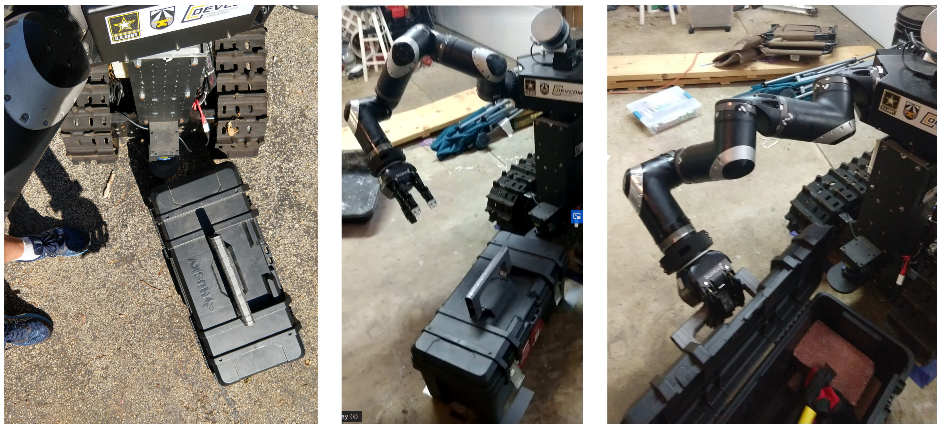


Figure 13. Left: RoMan positioned in front of the container, acquiring a pose estimate. Middle: RoMan at the pre-grasp configuration prior to moving to grasp the handle. Right: RoMan after opening the container, prior to grasp release.

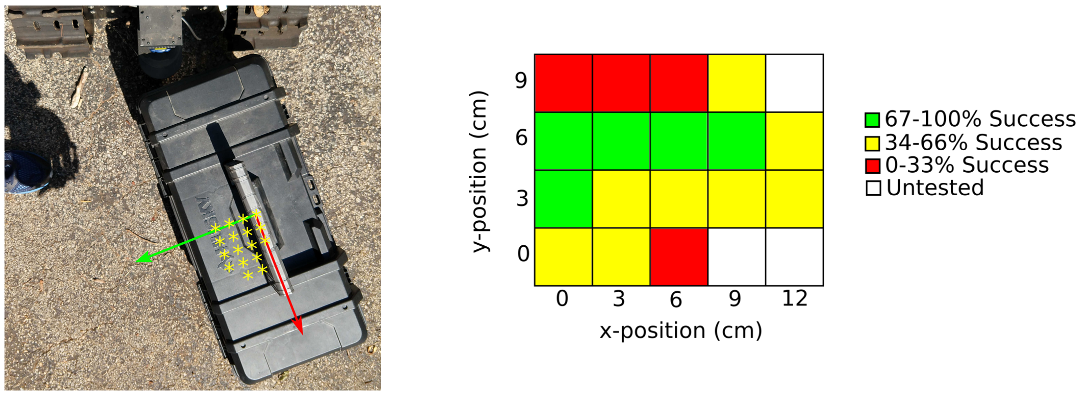


Figure 14. Container opening experiment. Left: range of positions attempted, shown in yellow. Right: success rates for each position.

The behavior tree was also modified to add intermediate limb positions. Observations during development indicated that using the basic limb planner to move the limbs into "pre-grasp" and "pre-manipulation" poses before attempting the more complex lid opening and item grabbing movements was helpful. Adding these steps increased success rates of the more complex tasks by reducing the chances of large inverse joint winds, avoiding manipulation singularities, and reducing more complex planning spaces.

Finally, breaking up some of these large, repeated limb motions into small chunks reduced mechanical faults, avoided dangerous near-collisions, and allowed RoMan to keep its limbs close enough to its center of gravity to avoid shaking that caused the PTU to lose calibration.

4.2.2. Results

The complete test sequence involved (1) a preset limb move, (2) the opening of the container via OMP, (3-5) a sequence of three preset limb movements to transition from a post container manipulation pose to the preset pre-grasp pose, (6) grasp planning and execution, and (7) a preset movement to remove the item from the container.

The experiments were setup indoors, with no base movement. The container was placed in front of RoMan inside the workspace of both limbs, and the behavior tree was executed. During initial setup, a successful position that was consistent with prior testing and demonstration was chosen as the control. 25 control trials were run, with a completely autonomous end-to-end success rate of 56%.

After the control trials, the container was moved to test the robustness of the behavior to positional accuracy. First, the container was moved away from the robot, along the container handle axis. These trials proved relatively unsuccessful. The required movement was "cramped" and the arm had trouble maneuvering around the lid. The container was then moved laterally, in the negative y direction of robot frame. Trials were then performed by moving the container progressively further away from the robot. During these first 74 non-control trials, the container handle yaw did not change. The results are shown in Figure 14 as starting locations colored by how often the container opening task completed successfully. Because the range of the motion was relatively large in comparison to the manipulator's constrained task space to include handle release and arm retraction without closing the lid, there were only a small set of positions from which the task could be completed. Generally, the task was robust (represented in green) when the manipulator was able to stay toward the interior of its workspace. As the task required motions closer to the edges of the workspace, probability of success fell (yellow and red) due to increased sensitivity to error.

The second grouping of trials involved taking three non-adjacent, relatively successful container positions and varying their yaw. After trials at 5 degrees counter-clockwise (CCW), it was apparent that CCW was not only relatively unsuccessful, but also potentially harmful to the robot. Therefore,

Table 5. Effect of Yaw Angle

Yaw	-5	0	5	10
Success	5	13	12	3
Failure	8	2	3	5
Rate	38%	87%	75%	38%

Table 6. Success Rates by Module

Module	**Success**
PERCH	99%
OMP	73%
Grasp Planner	95%

additional trials were performed at only 5 and 10 degrees clockwise (CW). The results are shown in Table 5.

According to Table 5, a yaw angle that stays within the [0, 5] interval has greater container-opening success than a yaw angle set outside those limits. Here we test this as a two-sided hypothesis, with the null hypothesis being that there is no difference in the proportion of success rates for yaw angle inside or outside the [0, 5] interval. We use the *prop.test* of R (Team et al., 2013), which is a two-sample test for equality of proportions with Yates continuity correction. The test result is to give the null hypothesis a probability of 0.00245, and we thus accept the alternative hypothesis that indeed a yaw angle in the [0, 5] range is advantageous in container opening.

Table 6 shows a breakdown of the success rates by module. The single greatest failure point was the container opener. The most likely explanation is that the system constraints throughout the motion require paths that get too close to workspace singularities. The grasp planner performed well. Most grasp module failures could be attributed to execution or motion planning errors, particularly when the arm failed to plan around the sides and lid of the container (probably due to common difficulty in resolving the point cloud data at the back of the container). PERCH failure was also rare.

Overall, RoMan successfully demonstrated autonomous container opening and bag retrieval, and proved moderately tolerant to variation in container position and orientation.

5. Discussion

5.1. Lessons Learned

Constructing a highly complex, autonomous, electromechanical system involves seemingly endless decisions throughout the program's entire life cycle, often requiring compromises due to resource scarcity. In this section, we provide some lessons we learned through both success and failure in the hopes that others might improve their ability to make wise decisions. While many are well known, it is easy to forget or discount, and reminders are prudent.

1) Plan and devote adequate resources to maintaining equipment. Proper maintenance is often undervalued and considered a lower, "nice to have" priority, because it is easy to take an optimistic view that relatively few problems will arise. This view results in resources being allocated to other areas considered higher priorities. However for us, inadequate staffing—resulting in only the bare-essential maintenance activities—contributed to repeated hardware problems, hampering both development and testing. In particular, because the program shared a small number of personnel for maintenance across all tasks, we found ourselves waiting on repairs on more than one occasion, while needs of other tasks were deemed higher priority. Our maintenance staff had to perform at a Herculean level at times to make up for the shortfall. Do not give maintenance resources short shrift.

2) When you do not have access to source code, ensure adequate access to source developers. In particular, we had many problems with our Ethercat interfaces dropping packets, slowing down communications, and outright crashing servers. We did not have access to the source code to resolve the issue ourselves, and we also did not plan sufficient time with or access to the developer's resources to adequately fix the problem. Faults resulting from this kind of constraint continually plagued the effort through its conclusion.

3) Be sure to identify hardware that is as robust as possible to your specific operating situation. While this sounds obvious, it is easy to underestimate the harshness of the conditions the robot will actually operate in. Similarly, it is easy to overestimate our ability to avoid problem conditions by programming intelligently, or by "being careful." This oversight places an unrealistic burden on operators at best, and at worst results in losing time and replacing expensive parts. As one example, we often experienced robot fingers becoming jammed in wooden features, or being sheared off by unfortunate movements in close proximity to objects being manipulated. If available, grippers designed to be more robust to the loads we were capable of delivering would have been prudent. Alternatively, low-level control software to take reflexive action to avoid such situations, such as applying our wrench-reactive controller more broadly, might have reduced the number of incidents.

4) Build in fault tolerance whenever possible. Related to the previous point, when designing your own hardware, ensure fail-safes are in place. Do not rely on operator procedure. As one example, our actuators did not use a slip ring. They were wired through the joint, but relative encoders were used, not absolute encoders. In some situations, brown-outs or full power loss resulted in damage to joints because they were inadvertently over-rotated.

5) Avoid feature creep; keep it simple. At times, we added features/capabilities that simply over-complicated the system and resulted in more problems than solutions. As one example, we added a pan/tilt unit (PTU) to acquire a broader range of sensory perception, especially during manipulation tasks. However, this resulted in one issue after another, including significant calibration difficulties, occlusion problems, collision concerns, and more hardware failures. Think carefully about the simplest way to address issues.

6) Clearly define experimental parameters and limits as early as possible. Take care not to be overly ambitious in the dimension/size of your experimental space. It is particularly easy to over-extend when aiming at unstructured environments. We continued to identify more and more parameters to which we wanted to demonstrate robustness, including a variety of paths, obstacle geometries and number, and lighting conditions. The mission creep and scope ended up becoming intractable, and some time was lost trying to meet ever-moving goal posts.

7) Plan regular, in-person "sprints" or experiments. There is no substitute for physical presence. Although the team worked together well remotely throughout the program, our progress significantly accelerated when we all met in one location for a week-long (sometimes two) sprint. Team focus increased as the sprint approached, and the event better prepared us for continued, remote collaboration afterward. While we did not run a controlled experiment, we suggest considering such events at least quarterly.

8) Stay disciplined – formulate and adhere to a consistent, principled pipeline for documenting and pushing code updates. Occasionally, the desire for rapid development resulted in code changes being made directly on the robot, subverting the standard commit pipeline. These actions meant version control was lost, and we spent valuable time tracing hard-to-find changes that ended up wreaking havoc due to unintended consequences on other subsystems. Development teams must stay disciplined and true to their processes to avoid trading (potential) short-term gain for long-term pain.

9) Minimize abstraction layers in code wherever possible and reasonable. When using a parameter, it is important to know where it is being set and where or how it might be modified. On more than one occasion, we had to look in five or six files (including the install space) to understand how a particular parameter value was evolving in order to debug a function. This is rarely ideal. Whenever possible, collect all parameters relevant to a particular module or task in a central location to ease the debugging task.

5.2. Open Research Problems

While our efforts—and those of the broader community at large—have shown great strides toward realizing human-scale mobile manipulation in recent years, we see several challenges remaining. In particular, we view the following areas as critical for the community to continue advancing human-scale robotic manipulation:

1) Improved sampling distributions in manipulation planning. As the environment gets more cluttered, the planners adopted in this project struggled to adapt their behavior to the distribution of the obstacles around them. GLS and ARA* are constrained by the initial choice of explicit graph or motion primitives. An *a priori* choice of the graph can either result in an unnecessarily dense graph, leading to large computation times, or a sparse graph that contains, at best, a low-quality solution, if it contains one at all. While asymptotic, sampling-based algorithms iteratively densify the configuration space, a large part of their computation time is expended in sampling narrow or constrained passages and bottlenecks to generate a high-quality solution (Amato et al., 1998; Wilmarth et al., 1999; Hsu et al., 2003). This issue is further exacerbated in higher dimensions, as in manipulation planning (Hauser, 2015). This challenge motivates learning sampling distributions conditioned on the environment and the planning query (start and goal locations). Recent work has shown promising results in this area (Ichter et al., 2018; Kumar et al., 2019; Qureshi et al., 2019; Ichter et al., 2020; Chamzas et al., 2020). However, those studies focus on relatively structured environments, such as manipulating near indoor furniture. In unstructured environments, such as those addressed in this project, there is a need to capture and learn more fundamental representations of obstacles to model arbitrary obstacle distributions.

2) Reasoning about the consequences of object manipulation. In this project we do not explicitly reason about the interactions between objects as we manipulate them, so long as the arm and the object it has grabbed do not collide with the environment as the arm moves. However, choosing which object to pick first is itself a discrete task-planning problem, requiring reasoning about how the adjacent debris would react to picking a particular object off the pile. While humans do this seamlessly, sometimes even stabilizing the pile with one arm and removing items with another, RoMan was limited in such capabilities. This situation motivates problems that go beyond pure geometric planning to reasoning about the dynamic interactions between objects in the scene. Unstructured environments addressed in this project exacerbate the problem in that they lend uncertainty to the physical parameters associated with the obstacles.

3) Manipulation strategies conditioned on objects' physics parameters. This project was primarily concerned with picking up an object and placing it out of the way, without explicitly reasoning about the object's geometry and physics characteristics. In scenarios like this, a human might regrasp objects to lessen the mechanical load on their arm. For example, when picking up a heavy log, even if the human initially pulls the log out of the debris by its end, one might reorient and regrasp the object closer to its center of mass for subsequent (mechanically easier) manipulation. Such a consideration is not addressed in the current scope of this paper. Thus one of the most common failure modes we observed was the lifting of heavy objects incurring a large wrench on the hand triggering a safety stop or, worse, hardware failure. In the environments described in this paper, the high potential for objects to impose reaction forces beyond the safe limits of actuators necessitates a greater level of reasoning about robot-environment interactions. This goal requires intersecting research ideas that aim to generate policies and plans robust to variations in physical parameters (Burkhardt et al., 2018; Lee et al., 2018a), as well as manipulation strategies that reorient and regrasp objects appropriately.

6. Conclusions and Future Work

In this paper, we presented the design, integration, and evaluation of a full-stack, autonomous, robotic system called RoMan, which is capable of exerting human-scale forces on the environment to perform meaningful work. We demonstrated this capacity through the removal of large, heavy barriers, such as a fallen tree in a roadway, and through the opening of a container to retrieve an object of interest, such as a laptop bag. These were achieved in both indoor and outdoor settings under a variety of lighting conditions. The resulting capabilities demonstrate the autonomous capacity to perform human-scale work in environments where humans cannot or should not go, without introducing onerous cognitive burdens on operating units, foreshadowing the possibility of future teaming functions.

Having shown the ability to deliver human-scale forces quasi-statically, our future work aims to extend this capability to dynamic forcing and tool-use paradigms. Additional behaviors will further coordinate the actions of the manipulators and base in the dynamic realm for increased force delivery. We also intend to improve speed, as these autonomous activities are still likely too slow for realistic adoption.

Acknowledgments

This research was sponsored by the U.S. DEVCOM Army Research Laboratory (ARL) under Cooperative Agreement W911NF-10-2-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARL or the U.S. Government.

ORCID

Chad C. Kessens https://orcid.org/0000-0001-6366-0880
Matthew Kaplan https://orcid.org/0000-0002-9625-0682
Trevor Rocks https://orcid.org/0000-0003-1723-2791
Philip R. Osteen https://orcid.org/0000-0001-8266-9848
John Rogers https://orcid.org/0000-0002-6074-0823
Ethan Stump https://orcid.org/0000-0002-0119-2169
Arnon Hurwitz https://orcid.org/0000-0002-6392-6228
Jonathan Fink https://orcid.org/0000-0002-1834-2442
Long Quang https://orcid.org/0000-0001-8780-7592
Mark Gonzalez https://orcid.org/0000-0002-7531-5774
Jaymit Patel https://orcid.org/0000-0003-0783-9397
Michael DiBlasi https://orcid.org/0000-0003-3846-4196
Shiyani Patel https://orcid.org/0000-0002-8770-7281
Matthew Weiker https://orcid.org/0000-0003-0233-0882
Dilip Patel https://orcid.org/0000-0002-2719-3936
Joseph Bowkett https://orcid.org/0000-0002-3101-489X
Renaud Detry https://orcid.org/0000-0003-0597-1167
Sisir Karumanchi https://orcid.org/0000-0002-0685-4125
Larry Matthies https://orcid.org/0000-0002-9045-0181
Joel Burdick https://orcid.org/0000-0002-3091-540X
Yash Oza https://orcid.org/0000-0003-0816-5494
Aditya Agarwal https://orcid.org/0000-0002-3535-3258
Andrew Dornbush https://orcid.org/0000-0001-8398-1294
Dhruv Mauria Saxena https://orcid.org/0000-0003-0878-0534
Maxim Likhachev https://orcid.org/0000-0002-9539-2398
Karl Schmeckpeper https://orcid.org/0000-0003-4989-2022
Kostas Daniilidis https://orcid.org/0000-0003-0498-0758
Ajinkya Kamat https://orcid.org/0000-0002-1675-1138
Aditya Mandalika https://orcid.org/0000-0003-1114-5435
Sanjiban Choudhury https://orcid.org/0000-0003-2762-8888
Siddhartha S. Srinivasa https://orcid.org/0000-0002-5091-106X

References

Akbari Hamed, K., Kamidi, V. R., Pandala, A., Ma, W.-L., and Ames, A. D. (2020). Distributed feedback controllers for stable cooperative locomotion of quadrupedal robots: A virtual constraint approach. In *Proceedings of the American Control Conference.*

Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D. (1998). Obprm: An obstacle-based prm for 3d workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168.

Bohren, J., Rusu, R. B., Jones, E. G., Marder-Eppstein, E., Pantofaru, C., Wise, M., Mösenlechner, L., Meeussen, W., and Holzer, S. (2011). Towards autonomous robotic butlers: Lessons learned with the PR2. In *2011 IEEE International Conference on Robotics and Automation*, pages 5568–5575. IEEE.

Bouman, A., Ginting, M. F., Alatur, N., Palieri, M., Fan, D. D., Touma, T., Pailevanian, T., Kim, S.-K., Otsu, K., Burdick, J., et al. (2020). Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. *arXiv preprint arXiv:2010.09259.*

Bowkett, J., Karumanchi, S., and Detry, R. (2021). Grasping and transport of unstructured collections of massive objects. *To appear: Field Robotics.*

Burkhardt, M., Karumanchi, S., Edelberg, K., Burdick, J. W., and Backes, P. (2018). Proprioceptive inference for dual-arm grasping of bulky objects using robosimian. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4049–4056. IEEE.

Chamzas, C., Kingston, Z., Quintero-Peña, C., Shrivastava, A., and Kavraki, L. E. (2020). Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions. *arXiv preprint arXiv:2010.15335.*

Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155.

Chitta, S., Sucan, I., and Cousins, S. (2012). MoveIt![ROS topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19.

Coleman, D., Sucan, I., Chitta, S., and Correll, N. (2014). Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785.*

Colledanchise, M. and Ögren, P. (2017). Behavior trees in robotics and ai: An introduction.

Dellaert, F. and Kaess, M. (2006). Square root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203.

Detry, R., Papon, J., and Matthies, L. (2017). Task-oriented grasping with semantic and geometric scene understanding. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Fitzgerald, C. (2013). Developing baxter. In *2013 IEEE Conf on Technologies for Practical Robot Applications (TePRA)*, pages 1–6.

Hauser, K. (2015). Lazy collision checking in asymptotically-optimal motion planning. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2951–2957. IEEE.

Hebert, P., Bajracharya, M., Ma, J., Hudson, N., Aydemir, A., Reid, J., Bergh, C., Borders, J., Frost, M., Hagman, M., et al. (2015). Mobile manipulation and mobility as manipulation-design and algorithms of robosimian. *Journal of Field Robotics*, 32(2):255–274.

Hsu, D., Jiang, T., Reif, J., and Sun, Z. (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *2003 IEEE international conference on robotics and automation (cat. no. 03CH37422)*, volume 3, pages 4420–4426. IEEE.

Ichter, B., Harrison, J., and Pavone, M. (2018). Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE.

Ichter, B., Schmerling, E., Lee, T.-W. E., and Faust, A. (2020). Learned critical probabilistic roadmaps for robotic motion planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9535–9541. IEEE.

Jacoff, A., Messina, E., Weiss, B. A., Tadokoro, S., and Nakagawa, Y. (2003). Test arenas and performance metrics for urban search and rescue robots. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 4, pages 3396–3403. IEEE.

Kam, H. R., Lee, S.-H., Park, T., and Kim, C.-H. (2015). Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60(2):337–345.

Karumanchi, S., Edelberg, K., Baldwin, I., Nash, J., Reid, J., Bergh, C., Leichty, J., Carpenter, K., Shekels, M., Gildner, M., et al. (2017). Team RoboSimian: semi-autonomous mobile manipulation at the 2015 DARPA robotics challenge finals. *Journal of Field Robotics*, 34(2):305–332.

Katyal, K. D., Brown, C. Y., Hechtman, S. A., Para, M. P., McGee, T. G., Wolfe, K. C., Murphy, R. J., Kutzer, M. D., Tunstel, E. W., McLoughlin, M. P., et al. (2014). Approaches to robotic teleoperation in a disaster scenario: From supervised autonomy to direct control. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1874–1881. IEEE.

Kessens, C. C., Fink, J., Hurwitz, A., Kaplan, M., Osteen, P. R., Rocks, T., Rogers, J., Stump, E., Quang, L., DiBlasi, M., et al. (2020). Toward fieldable human-scale mobile manipulation using roman. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II*, volume 11413, page 1141316. International Society for Optics and Photonics.

Klamt, T., Schwarz, M., Lenz, C., Baccelliere, L., Buongiorno, D., Cichon, T., DiGuardo, A., Droeschel, D., Gabardi, M., Kamedula, M., et al. (2020). Remote mobile manipulation with the centauro robot: Full-body telepresence and autonomous operator assistance. *Journal of Field Robotics*, 37(5):889–919.

Krotkov, E., Hackett, D., Jackel, L., Perschbacher, M., Pippine, J., Strauss, J., Pratt, G., and Orlowski, C. (2017). The DARPA robotics challenge finals: Results and perspectives. *Journal of Field Robotics*, 34(2):229–240.

Kumar, R., Mandalika, A., Choudhury, S., and Srinivasa, S. S. (2019). LEGO: Leveraging experience in roadmap generation for sampling-based planning. In *2019 IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Kümmerle, J., Kühner, T., and Lauer, M. (2018). Automatic calibration of multiple cameras and depth sensors with a spherical target. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8.

Le, Q. V. and Ng, A. Y. (2009). Joint calibration of multiple sensors. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3651–3658.

Lee, G., Hou, B., Mandalika, A., Lee, J., Choudhury, S., and Srinivasa, S. S. (2018a). Bayesian policy optimization for model uncertainty. *arXiv preprint arXiv:1810.01014*.

Lee, J., Grey, M., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S., Stilman, M., and Liu, C. (2018b). Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500.

Levinson, J. and Thrun, S. (2013). Automatic online calibration of cameras and lasers. In *Proceedings of Robotics: Science and Systems*.

Li, W. and Fritz, M. (2015). Teaching robots the use of human tools from demonstration with non-dexterous end-effectors. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 547–553. IEEE.

Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945.

Likhachev, M., Gordon, G. J., and Thrun, S. (2004). ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pages 767–774.

Maddern, W., Harrison, A., and Newman, P. (2012). Lost in translation (and rotation): Fast extrinsic calibration for 2D and 3D lidars. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Minnesota, USA.

Mandalika, A., Choudhury, S., Salzman, O., and Srinivasa, S. (2019). Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 745–753.

Massa, F. and Girshick, R. (2018). maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark.

Murray, R. M., Li, Z., Sastry, S. S., and Sastry, S. S. (1994). *A mathematical introduction to robotic manipulation*. CRC press.

Narayanan, P., Yeh, B., Holmes, E., Martucci, S., Schmeckpeper, K., Mertz, C., Osteen, P., and Wigness, M. (2020). An integrated perception pipeline for robot mission execution in unstructured environments. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II*, volume 11413, page 1141318. International Society for Optics and Photonics.

Narayanan, V. and Likhachev, M. (2016). PERCH: Perception via search for multi-object recognition and localization. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2016-June.

Nguyen, H. G. and Bott, J. P. (2001). Robotics for law enforcement: Applications beyond explosive ordnance disposal. In *Enabling Technologies for Law Enforcement and Security*, volume 4232, pages 433–454. International Society for Optics and Photonics.

Owens, J. L., Osteen, P. R., and Daniilidis, K. (2015). Msg-cal: Multi-sensor graph-based calibration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3660–3667.

Pages, J., Marchionni, L., and Ferro, F. (2016). Tiago: the modular robot that adapts to different research needs. In *International workshop on robot modularity, IROS*.

Pandey, G., McBride, J., Savarese, S., and Eustice, R. (2014). Automatic extrinsic calibration of vision and lidar by maximizing mutual information. *Journal of Field Robotics*, 32.

Pavlakos, G., Zhou, X., Chan, A., Derpanis, K. G., and Daniilidis, K. (2017). 6-DoF Object Pose from Semantic Keypoints. *IEEE International Conference on Robotics and Automation (ICRA)*.

Phillips, M., Cohen, B. J., Chitta, S., and Likhachev, M. (2012). E-graphs: Bootstrapping planning with experience graphs. In *Proceedings of Robotics: Science and Systems*.

Phillips, M., Hwang, V., Chitta, S., and Likhachev, M. (2016). Learning to plan for constrained manipulation from demonstrations. *Autonomous Robots*, 40(1):109–124.

Pradeep, V., Konolige, K., and Berger, E. (2014). Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *Experimental robotics*, pages 211–225. Springer.

Quigley, M., Berger, E., Ng, A. Y., et al. (2007). Stair: Hardware and software architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, pages 31–37.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. *ICRA workshop on open source software*.

Qureshi, A. H., Simeonov, A., Bency, M. J., and Yip, M. C. (2019). Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE.

Rehder, J., Nikolic, J., Schneider, T., Hinzmann, T., and Siegwart, R. (2016). Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Scaramuzza, D., Harati, A., and Siegwart, R. (2007). Extrinsic self calibration of a camera and a 3D laser range finder from natural scenes. In *IEEE International Conference on Intelligent Robots and Systems*, pages 4164–4169.

Schwarz, M., Rodehutskors, T., Droeschel, D., Beul, M., Schreiber, M., Araslanov, N., Ivanov, I., Lenz, C., Razlaw, J., Schüller, S., et al. (2017). Nimbro rescue: Solving disaster-response tasks with the mobile manipulation robot momaro. *Journal of Field Robotics*, 34(2):400–425.

Segal, A., Haehnel, D., and Thrun, S. (2009). Generalized-ICP. In *Proceedings of Robotics: Science and Systems (RSS)*, volume 2, page 435.

Srinivasa, S., Koval, M., Velagapudi, P., King, J., Lee, J., Lee, G., Hou, B., Liddick, C., Yi, D., Mandalika, A., Niyaz, S., Gordon, E., Kim, Y., Jin, P., and Choudhury, S. (2020). personalrobotics/aikido: AIKIDO v0.3.1. https://doi.org/10.5281/zenodo.3871156.

Srinivasa, S. S., Berenson, D., Cakmak, M., Collet, A., Dogar, M. R., Dragan, A. D., Knepper, R. A., Niemueller, T., Strabala, K., Weghe, M. V., et al. (2012). Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, 100(8):2410–2428.

Stein, S. C., Schoeler, M., Papon, J., and Worgotter, F. (2014). Object partitioning using local convexity. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 304–311.

Stentz, A., Herman, H., Kelly, A., Meyhofer, E., Haynes, G. C., Stager, D., Zajac, B., Bagnell, J. A., Brindza, J., et al. (2015). Chimp, the CMU highly intelligent mobile platform. *Journal of Field Robotics*, 32(2):209–228.

Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82. http://ompl.kavrakilab.org.

Team, R. C. et al. (2013). R: A language and environment for statistical computing.

Trevor, A. J., Rogers, J. G., and Christensen, H. I. (2014). Omnimapper: A modular multimodal mapping framework. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1983–1990.

Wells, P. and Deguire, D. (2005). TALON: a universal unmanned ground vehicle platform, enabling the mission to be the focus. In *Unmanned Ground Vehicle Technology VII*, volume 5804, pages 747–757. International Society for Optics and Photonics.

Wilmarth, S. A., Amato, N. M., and Stiller, P. F. (1999). Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1024–1031. IEEE.

Wise, M., Ferguson, M., King, D., Diehr, E., and Dymesich, D. (2016). Fetch and freight: Standard platforms for service robot applications. In *Workshop on autonomous mobile service robots*.

Yamauchi, B. M. (2004). Packbot: a versatile platform for military robotics. In *Unmanned ground vehicle technology VI*, volume 5422, pages 228–237. International Society for Optics and Photonics.

Zhou, L., Li, Z., and Kaess, M. (2018). Automatic extrinsic calibration of a camera and a 3D lidar using line and plane correspondences. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5562–5569.

How to cite this article: Kessens, C. C., Kaplan, M., Rocks, T., Osteen, P. R., Rogers, J., Stump, E., Hurwitz, A., Fink, J., Quang, L., Gonzalez, M., Patel, J., DiBlasi, M., Patel, S., Weiker, M., Patel, D., Bowkett, J., Detry, R., Karumanchi, S., Matthies, L., Burdick, J., Oza, Y., Agarwal, A., Dornbush, A., Saxena, D., M. Likhachev, M., Schmeckpeper, K., Daniilidis, K., Kamat, A., Mandalika, A., Choudhury, S., & Srinivasa, S. S. (2022). Human-scale mobile manipulation using RoMan. *Field Robotics, 2*, 1232–1262.