Toward Fieldable Human-Scale Mobile Manipulation Using RoMan

Chad C. Kessens[a], Jonathan Fink[a], Arnon Hurwitz[a], Matthew Kaplan[a], Philip R. Osteen[a], Trevor Rocks[a], John Rogers[a], Ethan Stump[a], Long Quang[b], Michael DiBlasi[b], Mark Gonzalez[b], Dilip Patel[b], Jaymit Patel[b], Shiyani Patel[b], Matthew Weiker[b], Joseph Bowkett[c], Renaud Detry[c], Sisir Karumanchi[c], Joel Burdick[c], Larry Mathies[c], Yash Oza[d], Aditya Agarwal[d], Andrew Dornbush[d], Maxim Likhachev[d], Karl Schmeckpeper[e], Kostas Daniilidis[e], Ajinkya Kamat[f], Sanjiban Choudhury[f], Aditya Mandalika[f], and Siddhartha S. Srinivasa[f]

[a]US Army Research Laboratory, 2800 Powder Mill Road, Adelphi, MD, USA 20783
[b]General Dynamics Land Systems, 1231 Tech Court, Westminster, MD, USA 21157
[c]NASA Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA, USA 91109
[d]Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA 15213
[e]University of Pennsylvania, 472 Levine Hall, 3330 Walnut St, Philadelphia, PA, USA 19104
[f]University of Washington, 185 E Stevens Way NE, Seattle, WA, USA 98195

ABSTRACT

Robots are ideal surrogates for performing tasks that are dull, dirty, and dangerous. To fully achieve this ideal, a robotic teammate should be able to autonomously perform human-level tasks in unstructured environments where we do not want humans to go. In this paper, we take a step toward realizing that vision by introducing the integration of state of the art advancements in intelligence, perception, and manipulation on the RoMan (Robotic Manipulation) platform. RoMan is comprised of two 7 degree of freedom (DoF) limbs connected to a 1 DoF torso and mounted on a tracked base. Multiple lidars are used for navigation, and a stereo depth camera visualizes point clouds for grasping. Each limb has a 6 DoF force-torque sensor at the wrist, with a dexterous 3-finger gripper on one limb and a stronger 4-finger claw-like hand on the other. Tasks begin with an operator specifying a mission type, a desired final destination for the robot, and a general region where the robot should look for grasps. All other portions of the task are completed autonomously. This includes navigation, object identification and pose estimation (if the object is known) via deep learning or perception through search, fine maneuvering, grasp planning via grasp library, arm motion planning, and manipulation planning (e.g. dragging if the object is deemed too heavy to freely lift). Finally, we present initial test results on two notional tasks: clearing a road of debris such as a heavy tree or a pile of unknown light debris, and opening a hinged container to retrieve a bag inside it.

Keywords: Autonomy, Mobile Manipulation, Grasping, Field Robotics, Unstructured, Debris, Robust, Full-stack

1. INTRODUCTION

While robots have proven extraordinarily useful for tasks that are dull, dirty, and even dangerous, real applications have remained limited for the most part to environments that are controlled. However, many of the most dangerous tasks we would like robots to perform require operation in uncontrolled environments, meaning that there is no ability to change factors (e.g. lighting, weather, terrain) to overcome potential deficiencies of the robot. The robot must meet the challenge where it is and cannot be "met halfway". Disasters have inspired several competitions toward that end, such as RoboCup Rescue[1] and the DARPA Robotics Challenge,[2] the latter of which focused on the need for robots to perform human-scale manipulation tasks to solve some of the challenges that the nuclear disaster at Fukushima presented. Beyond disaster response, robots with human-scale manipulation capabilities could also prove vital to increasing safety in military and law enforcement scenarios.[3]

Whereas challenge formats provide one avenue of progress, alternative forms of effort are useful due to their complementary strengths. In 2010, the US Army Research Laboratory (ARL) formed the Robotics Collaborative Technology Alliance (RCTA), a consortium of government, academic, and industry researchers, to advance the state of autonomous ground robot technologies. The program conducts leading edge research in basic and applied ground robotics technologies,

E-mail correspondence to: chad.c.kessens.civ@mail.mil

with an overarching goal of transitioning from tele-operation to more robust autonomy. The research addresses the US Army's manned-unmanned teaming (MUMT) requirement, particularly for the dismounted team. However, the research is also applicable to other MUMT scenarios including larger platforms and systems. The program's four research focus areas are: Perception, Intelligence, Human-Robot Interaction, and Dexterous Manipulation & Unique Mobility. Each area addresses critical technology gaps in achieving key autonomous robotic capabilities, such as high-speed perception and mobility in rough terrain, situational awareness in unstructured environments, collaborative human-robot mission planning and execution, multi-modal human-robot dialogue, and dexterous manipulation in cluttered environments. This paper focuses on this last gap, particularly when the manipulators are on a mobile platform. A robot capable of completing human-scale mobile tasks with little supervision would free up the dismounted soldier to focus on soldier tasks and serve as an important force multiplier.

Although the military has deployed thousands of field robots, a market survey early in the effort led us to conclude that current systems suffer from at least one of four problems. 1) Their mobility is insufficiently robust to the types of terrains on which the military may wish to deploy them.[4,5] 2) They are of insufficient size and/or strength to perform human-scale manipulation tasks.[6,7] 3) They are not commercially available.[8,9] 4) They are insufficiently open-source to easily integrate new hardware and test experimental autonomous research techniques, especially for a team with foreign national participants. Therefore, we chose to leverage work done on the Robosimian platform[9] for the DARPA Robotics Challenge due to its strength, but modifying its base to fit on a tracked, Talon-like base for improved speed and robustness. We call the system RoMan (Robotic Manipulator), and we produced 4 complete systems to facilitate the work.

This paper summarizes the essential features, development, and initial testing of RoMan. We begin with a general description of the system, including its hardware, sensors, computational abilities, overall software architecture, and calibration techniques. We then describe each of the component capabilities that must work together seamlessly to perform each entire task. These include base navigation, object and pose estimation (if object is known), grasp planning, and arm planning and manipulation of various objects. Next, we describe some initial experiments for our scenarios, which included clearing heavy and stacked light debris, as well as opening a container and retrieving a bag from inside. Finally, we offer our concluding thoughts and future directions.

2. ROBOT DESCRIPTION

We begin by describing the overall system design of RoMan, including the hardware components, software architecture, and system calibration techniques that enable the system to seamlessly function.

2.1 System Hardware

To carry out human-scale manipulation tasks (e.g. lifting objects, interacting with human-made infrastructure, etc.), a platform close to human scale and ability was necessary. Therefore, RoMan's foundation is comprised of a tracked base for mobility, two limbs for dual arm manipulation, a torso, end effectors for dexterous and powerful tasks, a sensor head and suite to provide situational awareness, and the necessary computational hardware. A mechanical drawing of the platform with basic dimensions can be found in Fig. 1.

2.1.1 Mobile Base

To provide better traction on unpaved roads, terrain involving soft material such as sand or loose gravel, and urban environments that involve partially destroyed infrastructure and debris, a tracked design was selected to take advantage of lower ground pressures throughout transport. QinetiQ NA provided the TL1 base platform, a non-ITAR variant of their TALON® platforms, which served as the foundation of RoMan's ground mobility. The drive-train consists of 2 independent electric motors for skid steering capability coupled with a 40:1 gearbox, capable of outputting a total 182.4 N-m of torque. Each motor is controlled via a Gold Cello 50/100 (ELMO Motion Control) motor controller located within the power distribution box centered on RoMan. The motor controllers maintain a master/slave configuration and utilize an Ethercat communication protocol with the primary computing system.

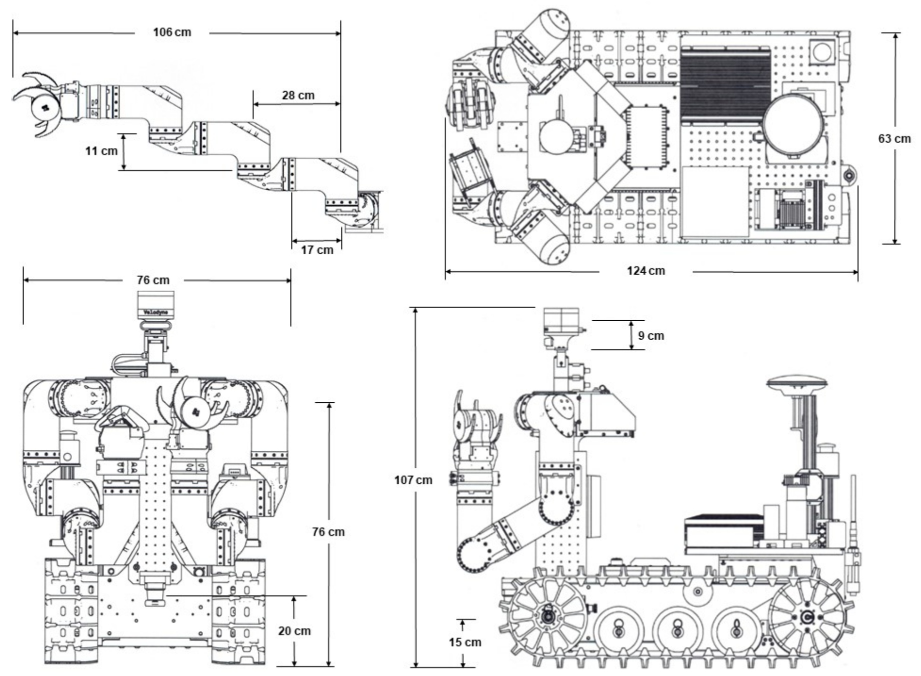


Figure 1. RoMan mechanical drawing with basic dimensions.

2.1.2 Manipulators

The RoMan limb design is derived from Jet Propulsion Laboratory's Robosimian,[10] with modifications made to the choice of end effectors. Each limb is offset by 45 degrees from the sagittal plane on a torso plate approximately 0.7 m from the ground, allowing the platform to reach objects that may rest on a shelf up to 1.5 m high. A previous configuration maintained a 90 degree offset, but this produced a lesser effective work area and required frequent base adjustments prior to executing object grasp attempts. The entire upper torso assembly rests upon a similar actuator found within the limbs to provide lateral access for manipulation and sensing and to minimize base reorientation, especially within narrow environments.

Two types of end-effectors were selected for the scope of objects the platform is expected to manipulate. RobotiQ's 3-Finger Gripper was selected for delicate to light-duty manipulation involving small and medium sized objects (e.g. bags, light debris, fuel containers, etc.). To fully utilize the limb's lift strength, the left arm was outfitted with Motiv Robotics' Camhand gripper for heavy duty manipulation tasks.

2.1.3 Sensors

Visual sensors are located on the anterior of the platform, primarily within the head assembly. A Velodyne Lidar® system (VLP-16) and an Intel® Realsense™ Depth and Tracking Camera (D435) were integrated together in a head assembly that provides perception from a single vantage point to maximize the short range and long range 3D- mapping capabilities. Both sensors are mounted on a 2-DoF actuation system developed by FLIR (D47), providing panning and tilting functionality to better view its surroundings and objects to be manipulated. For low ground obstacle avoidance, a Hokuyo single line LIDAR (UTM-30LX-EW) is installed at the height of to the drive wheel axles to detect obstacles larger than its radius, which it would have difficulty navigating over. Positioning sensors are located amongst the computing hardware near the caudal end of the platform. An Inertial Measurement Unit (IMU, Microstrain® 3DM-GX5-25) is located near the geometric center of the platform, and the Global Positioning System (GPS, Ublox EVK-M8T) antenna is mounted on a mast for reduced interference. Force/Torque sensors (ATI Mini-58) are equipped within the wrist of each limb to provide feedback for force-controlled manipulation.

2.1.4 Power

RoMan's track base contains the battery system and main power distribution system. The platform is powered by nine 9.9 Ah MIL-Spec Lithium-Ion batteries (Bren-tronics, BB-2590) arranged in a 3s3p configuration to provide 2200 Watt-hours of power. The modular design of the battery system allows the platform to remain operational while users remove and replace discharged batteries for un-tethered operations. For tethered operations, the platform is provided shore power from a single power supply (Keysight, N5769A); however only stationary tasks were recommended while tethered.

The platform's main operating voltage nominally runs between 72-83 V and provides bus voltages of 12 V and 16.5 V to support sensors and computational hardware, respectively. 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 1̃-2 hours of periodic operation. Nominal and peak power demands of the system can be found in Table 1.

Power Bus	Component	Distribution Location	Consumption (W)
12V	Netgear Switches	Computer Tray	24
	HRI E-Stop	Computer Tray	8
	Wifi Network	Computer Tray	7
	FLIR PTU	Sensor Head	70
	Velodyne	Sensor Head	8
	Hokuyo Lidar	Torso	8
16.5V	7 DoF Limb (Logic)	Chestplate	70
	Nuvo, AM1	Computer Tray	63.6
	Brix, AM2 & AM3	Computer Tray	33.6
	Robotiq Gripper	Limb	50
	Torso DoF (Logic)	Torso	10
19V	Zotac	Computer Tray	33.4 - 124.3
72-83V	Tracks	Base	54.2 - 346.1
	7 DoF Limb	Chestplate	778 - 2780
	Camhand	Limb	66
	Torso Actuator	Torso	111 - 397

Table 1. Power budgets for various components on RoMan.

2.1.5 Computing

RoMan's computational demands are typically met utilizing three computers but can be distributed among more depending on the application and available power. The main computer (AM1, NUVO-5002LP) performs all tasks involving actuating the track motors, manipulators, and Pan-Tilt unit. All of the actuation on the platform utilizes EtherCAT and USB communication protocols and are directly sent to AM1 for latency purposes. For developing the platform's world model, a second computer (AM2, GB-BXi78550) compiles sensor data from the on-board network and direct feed from the depth sensors. Perception development requiring GPU processing is performed on the third computer (AM3/AM4, ZBOX-QK7P5000) to execute neural network solutions. This perception computational hardware emphasizes raw sensor data processing towards classifying and detecting various objects, key-points, and other desirable attributes.

2.2 Software Architecture

The system's software stack is distributed across the aforementioned heterogeneous set of small form factor PCs, denoted AM$\{1, 2, 3, 4\}$. This is supplemented by a remote system connected via WiFi, typically a laptop termed roman-ops, from which an operator may issue behavior calls as well as inspect sensor data and plans from algorithms running on-board the platform. However, all computing required for RoMan to perform the given tasks is contained on the platform without connection to external computing resources. The architecture employs the common horizontal stratification of functional modules by level of hardware abstraction, as depicted in Fig. 5. This approach is taken, as opposed to the alternative vertical stratification,[11] due to the system's functional layers being furnished by different institutions within the program, necessitating distinct ownership of components within each level.

The majority of system modules are comprised of nodes within a network of the common robotics middleware ROS (Robot Operating System).[12] Each node is represented by one distinct rectangle within Fig. 5, with arrows depicting inter-process communication (IPC) between modules. The prime exception to this is at the lowest level of the stack, the drivers that interface with motor controllers and sensors, where the interface to hardware is provided by proprietary code from the Jet Propulsion Laboratory (JPL). A bridge between the ROS network and the underlying JPL drivers is provided in

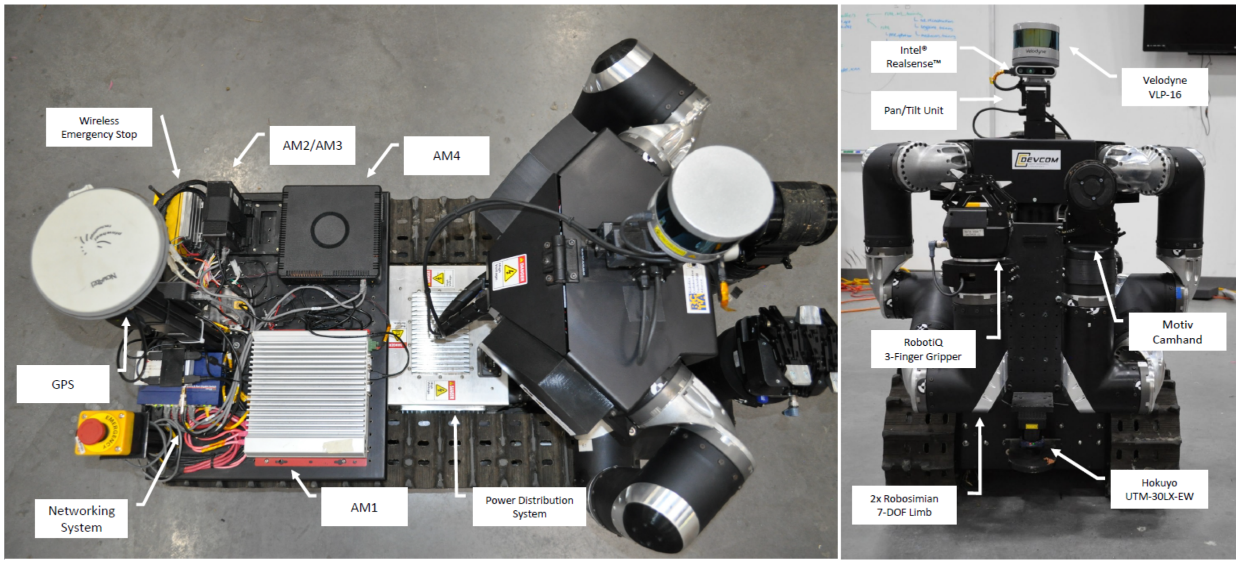


Figure 2. RoMan's major components labeled.

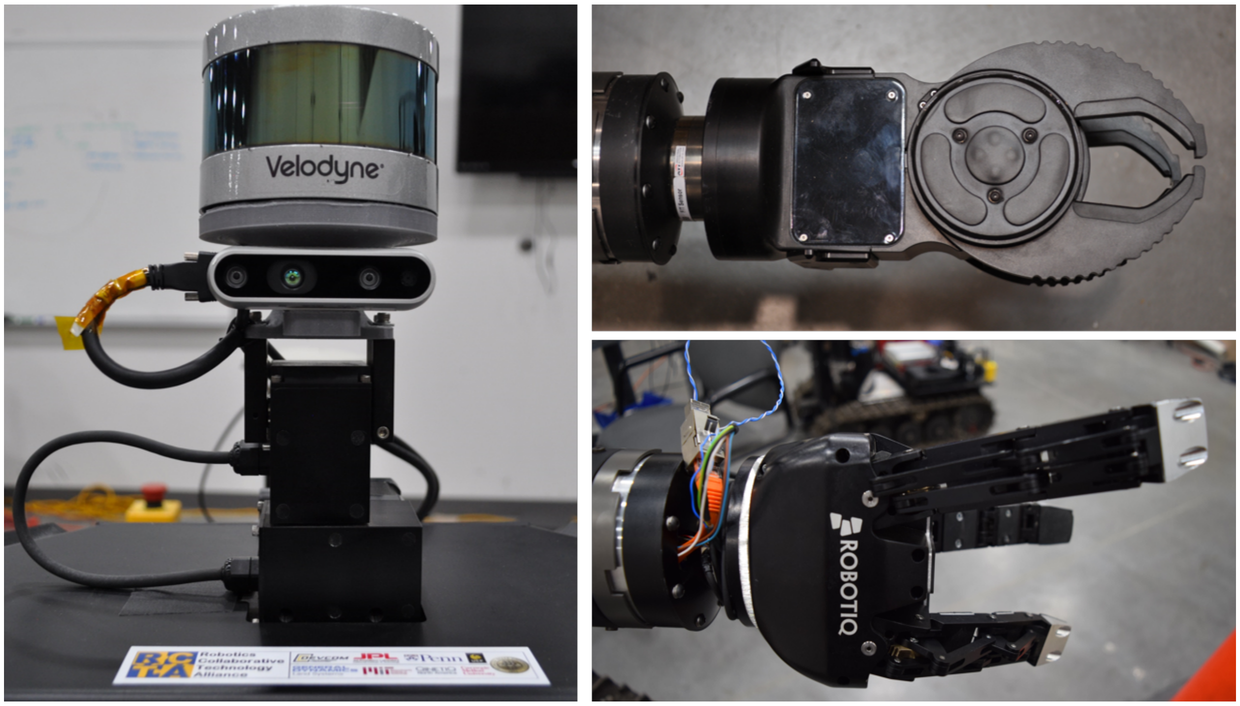


Figure 3. Sensors found within (left) head assembly and (right) selected end-effectors.

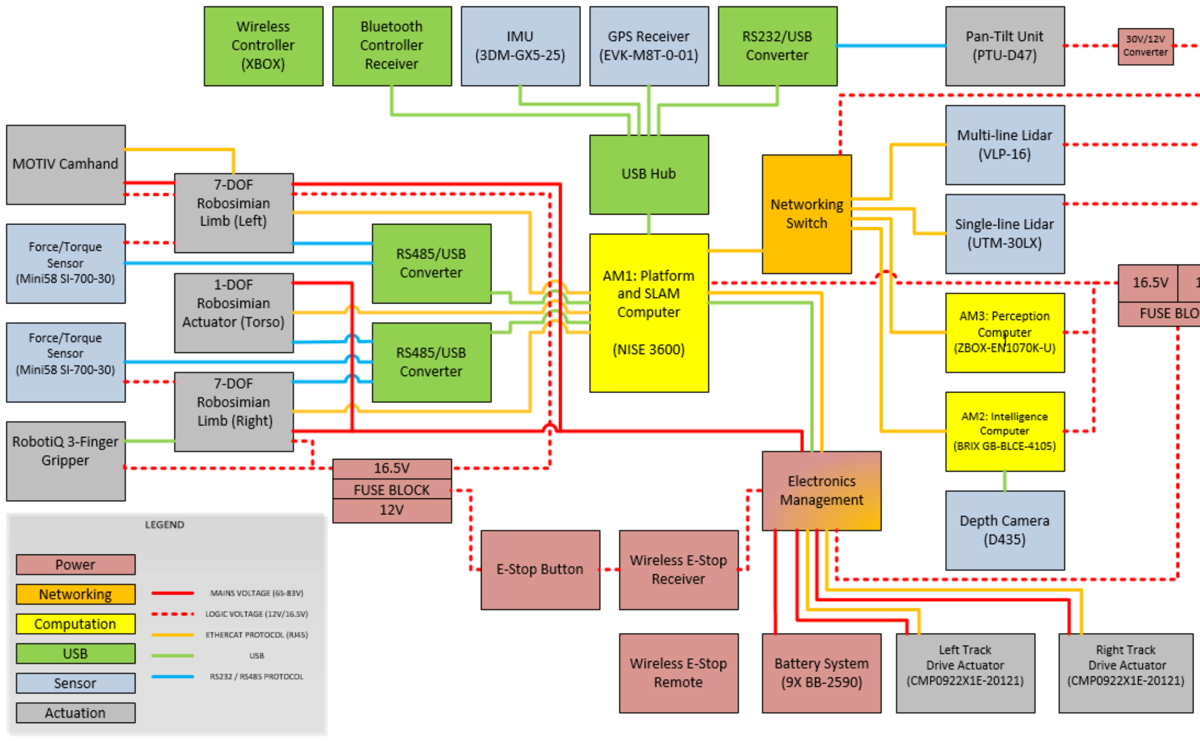

Wireless Controller (XBOX)
Bluetooth Controller Receiver
IMU (3DM-GX5-25)
GPS Receiver (EVK-M8T-0-01)
RS232/USB Converter
Pan-Tilt Unit (PTU-D47)
30V/12V Converter
MOTIV Camhand
7-DOF Robosimian Limb (Left)
USB Hub
Networking Switch
Multi-line Lidar (VLP-16)
Single-line Lidar (UTM-30LX)
Force/Torque Sensor (Mini58 SI-700-30)
RS485/USB Converter
AM1: Platform and SLAM Computer (NISE 3600)
16.5V
12V
FUSE BLOCK
1-DOF Robosimian Actuator (Torso)
AM3: Perception Computer (ZBOX-EN1070K-U)
Force/Torque Sensor (Mini58 SI-700-30)
RS485/USB Converter
7-DOF Robosimian Limb (Right)
AM2: Intelligence Computer (BRIX GB-BLCE-4105)
RobotiQ 3-Finger Gripper
16.5V
FUSE BLOCK
12V
Electronics Management
Depth Camera (D435)
E-Stop Button
Wireless E-Stop Receiver
Wireless E-Stop Remote
Battery System (9X BB-2590)
Left Track Drive Actuator (CMP0922X1E-20121)
Right Track Drive Actuator (CMP0922X1E-20121)
LEGEND
Power
Networking
Computation
USB
Sensor
Actuation
MAINS VOLTAGE (65-83V)
LOGIC VOLTAGE (12V/16.5V)
ETHERCAT PROTOCOL (RJ45)
USB
RS232 / RS485 PROTOCOL

Figure 4. RoMan system diagram.

the form of the RoMan Client module, seen at the far right of Fig. 5, which exposes motor controller primitives and status feedback. Successive layers of abstraction are built upon this driver interface within the ROS framework (pictured right to left), beginning with limb controllers and intrinsic coordination, local and global planning, and the highest module layer comprised of behaviors.

To consolidate the arm movement through various planners and movement profiles, all robot movements are passed through a centralized “Whole Body Planner” (WBP) ROS node, with the exception of coarse base navigation over long distances. Through the WBP, a behavior can control the arms, neck, and base of RoMan for various manipulation and perception tasks. This allows for interchangeability between planners and controllers without modifications to the top level behavior modules. This simplifies integration of behaviors from various institutions by ensuring that all high level hardware control uses the same message to communicate with the various low level planners. The WBP currently communicates with two arm planners, one designed at Carnegie Mellon University (CMU) and the other at the University of Washington (UW). The former uses the MoveIt! framework[13] and the latter uses the Aikido Framework. The WBP then arbitrates between the two planners and passes the chosen plan on to the Joint Trajectory Controller, which executes the motion.

Information about the objects RoMan encounters are stored in the World Model. For this experiment, the World Model is used to hold information about objects discerned by perception algorithms or entered by the user. All of this data is stored centrally in the World Model and is available to the various behaviors running on RoMan. This allows multiple behaviors to utilize data from a single source, as opposed to each behavior using a bespoke perception algorithm.

To synthesize compound tasks, such as clearing debris or opening containers, these top level modules are sequenced using a “Mission Executor” capable of parsing through and executing a behavior tree architecture. For the purposes of the tests in this paper, the Mission Executor is used to run a simple script of component behaviors in a serial fashion, affording binary outcome responses from each, and upon failure may either re-attempt the given behavior or fail the task in its entirety. These behaviors made use of the ROS Actionlib to report status updates to the Mission Executor, including failures or successes of behaviors. Later work will detail the extension of this structure to enable behavior branches, allowing greater deliberation to be captured within the task representation.

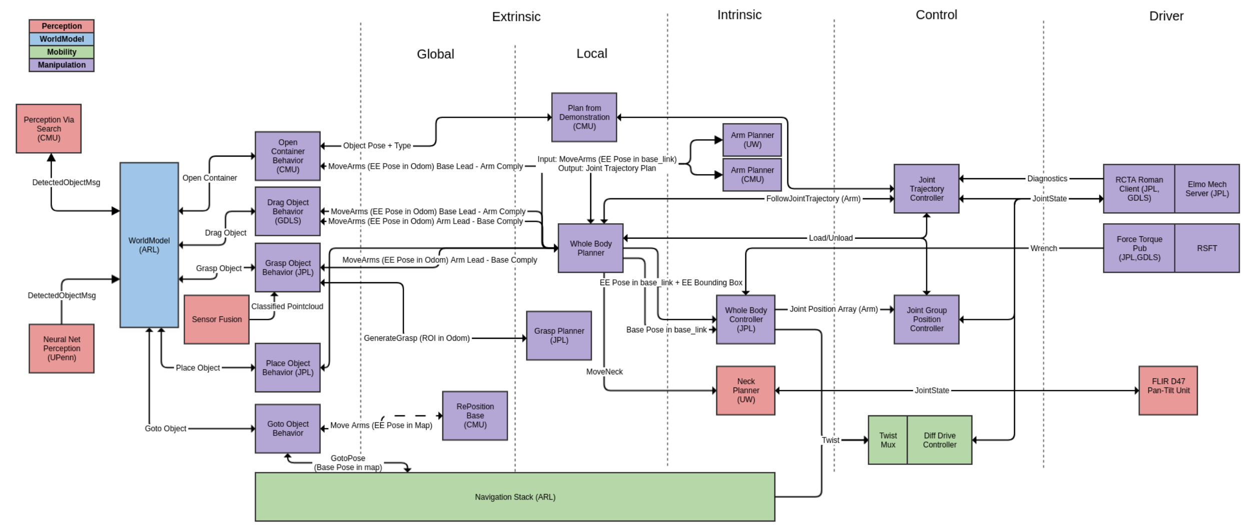

Perception
WorldModel
Mobility
Manipulation
Extrinsic
Global
Local
Intrinsic
Control
Driver
Perception Via Search (CMU)
DetectedObjectMsg
WorldModel (ARL)
DetectedObjectMsg
Neural Net Perception (UPenn)
Open Container
Drag Object
Grasp Object
Sensor Fusion
Classified Pointcloud
Place Object
Goto Object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

Figure 5. Block diagram of modules within the RoMan software architecture.

2.3 Calibration

To appropriately create a unified understanding of the world and RoMan's pose within it, we used two methods of calibration. Hand-eye calibration ensured data delivered from the joint encoders accurately localized the position of each end effector frame relative to the base frame. In addition, because some sensors were located on an articulated pan-tilt "head", calibration was critical to resolving the egomotion of the robot relative to the data perceived about the world.

2.3.1 Hand-Eye Calibration

Initial experiments indicated that vibrations from the mobile base could cause the transform between the pan-tilt unit (PTU) and the RealSense™ camera to drift over time. Therefore, we implemented an automated hand-eye calibration algorithm to correct for this drift when needed.

The hand-eye calibration involves finding a rigid transform between the pan-tilt unit and the RealSense™ camera, ${}^{camera}T_{PTU}$. We first attached an AprilTag to the gripper such that the rigid transform between the gripper and the AprilTag ${}^{AprilTag}T_{gripper}$ was accurately known. The position of the gripper in the camera frame could then be calculated using the perceived pose of the AprilTag. The transformation from the PTU to the gripper ${}^{gripper}T_{PTU}$ was calculated using the ROS TF tree using the URDF for the robot.

All of the joints in the kinematic chain from the PTU to the gripper were calibrated prior to running the hand-eye calibration routine to maximize the accuracy of ${}^{gripper}T_{PTU}$. To increase the robustness of our algorithm, we utilized a point cloud based approach. The arm was moved to locations throughout the workspace, and the transforms were captured at every location. They were then converted into point clouds. Finally, we used Iterative Closest Point (ICP) to find the appropriate transforms ${}^{camera}T_{PTU}$.

2.3.2 Articulated Sensor Calibration

To facilitate the calibration of the articulated sensors, we extended prior work in multi-sensor calibration.[14] While most sensor calibration systems consume raw input data and calibrate between sensors with no regard to where they fit within the larger robot definition, our calibration directly considers the information contained in the URDF model and ROS `tf` tree as part of the optimization, shown in Fig. 7 (left). The URDF definition is particularly important here, as it defines the non-fixed joints of the system, which the calibration framework uses to optimize both sensor and actuator positions. The resulting calibration graph is implemented as a tree of coordinate frames that are constrained by corresponding detections of a calibration object, which is shown in Fig. 7 (right). Specifically, the optimization graph contains the following entities:

- `Pose`: 6 Degree-of-Freedom (DoF) relative pose of a coordinate frame with respect to the parent frame. Sensor and Actuator types derive from Pose, and all existing `tf` coordinate frames are included as Poses.

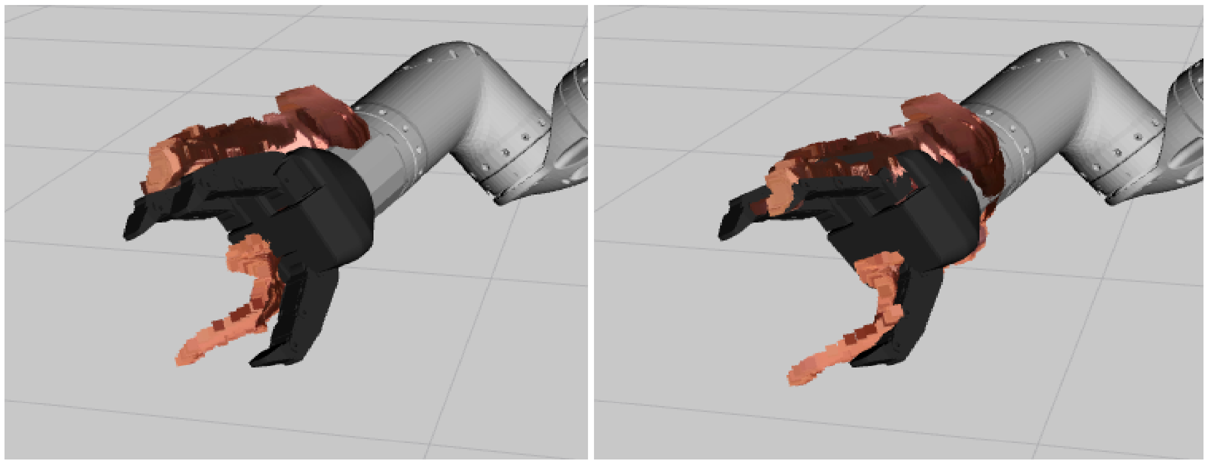


Figure 6. Hand-Eye Calibration Sample Result. Left: Before performing calibration, the point cloud (in orange) and the Robot Gripper are misaligned. Right: After hand-eye calibration, the alignment is significantly improved.

- `Actuator State`: 1 DoF state of the parent actuator at a given observation. This serves a similar function to the `joint_states` message in ROS, providing dynamic updates of the actuated poses.
- `Observation`: The geometric definition of the calibration object as seen from a given sensor at a given time. The three types of observations are `Line`, `Plane`, and `Tag`. `Line` represents an observation of a 2D lidar; `Plane` observations are derived from both camera and 3D point cloud sensors; and `Tag` observations are generated from the location of each individual fiducial tag pose, providing stronger optimization constraints for camera-camera sensor pairs.

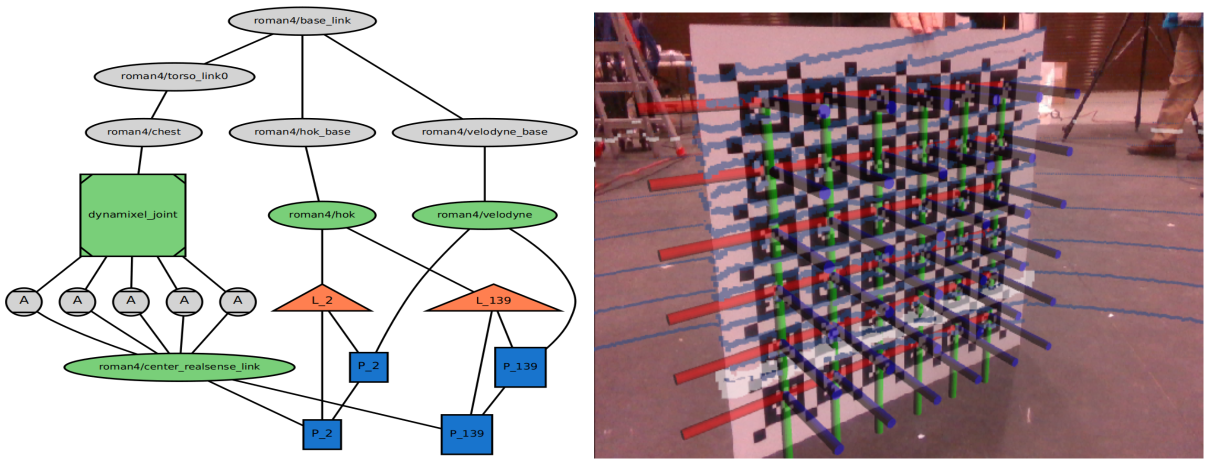


Figure 7. (Left) Optimization graph auto-generated from calibration performed on a RoMan. (Right) Corresponding data fusion visualization. The calibrated entities are a 2D lidar, 3D lidar, actuated camera and associated actuator (shown in green). Line observations (shown in orange) and plane observations (shown in blue) define the graph constraints for calibration.

3. COMPONENT ACTIONS

Having established the details of the system hardware and overarching software in section 2, we now provide an overview of the algorithms that plan and execute RoMan's component actions including navigation, object recognition and pose estimation, grasping, and manipulation under various conditions. While space limitations prevent in-depth descriptions of these algorithms, references in each subsection contain finer detailed descriptions for each component.

3.1 Coarse Base Navigation

RoMan is equipped with a set of navigational capabilities to enable safe traversal in a cluttered environment. This set of capabilities consists of three major components: 1) A metric simultaneous localization and mapping (SLAM) system keeps track of the location of obstacles relative to the robot's location. 2) A long range global planner generates coarse overall plans. 3) A local planner optimizes local control commands to faithfully follow the global plan while safely avoiding collision with fast moving obstacles.

The SLAM system is based upon the *OmniMapper*.[15] Briefly, this system localizes the robot and builds maps of obstacles by constructing a graph of measurements. These measurements consist of relative poses between adjacent *frames* along the robot's trajectory. A new frame is initialized with a LiDAR *point cloud* when the robot has traveled 0.5 meters from the last frame. Platform odometery and iterated closest point (ICP)[16] point cloud alignment each contribute measurements in the graph. Additionally, longer distance *loop-closure* measurements are added through ICP when revisiting a previously mapped area. The posterior trajectory estimate is given by optimizing this graph of measurements using the nonlinear least-squares optimization framework GTSAM.[17] Sensor frames are rendered along the robot's optimized trajectory into an occupancy grid using a negative log-odds updating rule. This occupancy grid is then sent to the global planner where it is used to identify free paths around obstacles.

Navigation goals are sent to the global planner in the form of a region specified by a goal location and a radius. The global planner samples locations within that region to identify a safe place where the robot's footprint will fit without coming into contact with obstacles. The global planner then identifies a kinematically feasible path by searching a sequence of motion primitives via the search-based planning library (SBPL).[18]

Finally, a local controller module converts the next segment of the global plan into a series of control commands in the form of left and right track velocities. In addition to the global plan, the latest LiDAR point cloud is used to identify moving obstacles not yet incorporated into the global map. The continuous trajectory segment that the local planner executes tries to match the global plan while avoiding obstacles and respecting velocity limits. Taken together, the SLAM system, global planner, and local controller allow RoMan to develop and execute coarse plans for navigating its environment to goals.

3.2 Object Recognition and Pose Estimation

While performing work in unstructured environments generally necessitates the ability to interact with unknown objects, some missions may involve interacting with predictable objects that can be known and trained on a priori. In these circumstances, performance improvements in terms of both speed and robustness are possible. Under varying circumstances, one approach to object recognition and pose estimation may outperform another. For RoMan, we integrated two complementary approaches to this perception problem: a neural network-based approach and a search-based approach.

3.2.1 Neural Network Approach

Although RoMan was equipped with numerous sensors including Lidar and RGB-D cameras, the Lidar data quickly became sparse at longer ranges, with few beams intersecting smaller or thinner objects of interest at longer ranges. Therefore, only RGB input from the RGB-D sensor was used for pose estimation by the neural network-based estimation pipeline, which is comprised of several components. First, object instances are detected in an image. For each object instance, we detect semantic keypoints and use them to estimate the 6-DOF pose of the object. The pose estimates and keypoint locations are then combined across timesteps using semantic SLAM. Both the object detection and pose estimation components of this pipeline are dependent on neural networks, making data collection and training important for the performance of the system.

For object detection, we used the maskrcnn-benchmark[19] implementation of the Faster-RCNN pipeline[20] to detect objects in RGB images. This pipeline was trained on a dataset[21] collected for objects that RoMan could reasonably be expected to know for its assigned tasks, namely a Czech hedgehog and a known toolbox. Our keypoint-based pose estimation pipeline is based upon work by Pavlakos.[22] We detect class-specific semantic keypoints for each object detection using a stacked-hourglass network. The detected keypoints are matched to a deformable model of the object class. The model's pose and deformations are then optimized using expectation maximization to match the detected keypoint locations.

To fuse object positions across multiple timesteps, we use a semantic SLAM algorithm.[23] This algorithm probabilistically assigns detected keypoints to object instances, and optimizes the pose of the robot and the detected keypoint locations across multiple timesteps. To improve keypoint annotation collection speeds for training our pose estimator, we use a pipeline that reconstructs a three-dimensional mesh of the object of interest, allowing the annotator to label the three-dimensional model and project the keypoint locations onto each image. This allows an annotator to label hundreds of images with a single click. More details of this pipeline are available in a companion paper.[21]

3.2.2 Perception Via Search Approach

PErception Via SeaRCH or PERCH refers to an approach that searches for the best possible explanation of the observed scene in a space of rendered scenes. On RoMan, PERCH was deployed for the task of object articulation wherein RoMaN is required to open a known container with a hinged lid. Specifically, PERCH estimates the 3-DoF pose of the crate (x, y, yaw) to be opened, which then serves as an input for the manipulation planner.

The task of opening a hinged lid imposes stringent accuracy requirements on estimated pose due to workspace limitations and the need to locate the handle, which occupies a limited area with respect to the entire container. In addition, the size of the container and its close proximity to the robot for the task of opening occludes large regions of the space from the camera, thus making accurate pose estimation considerably more difficult. To counter scenarios such as these, PERCH formulates the pose estimation task as a search over a specially constructed tree of rendered scenes in different 3-DoF poses and uses the input depth data to converge to a global optimum solution corresponding to the accurate pose.

Given the input point cloud I, PERCH estimates poses of $O_{1:K}$ objects in the scene by seeking to find a rendered point cloud R_j having $j(\leq K)$ objects, such that every point p in I has an associated point in R_j and vice-versa. In other words, PERCH seeks to minimize the following objective :

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

in which OUTLIER($p|C$) for a point cloud C and point p 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. To counter the intractability of this joint global optimization problem owing to a large search comprising of all possible joint poses of all objects, PERCH decomposes the cost function over individual objects added to the rendered scene. The decomposition is subject to the constraint that the newly added object does not occlude those already present. This allows the optimization to be formulated as a tree search problem where a successor state is added to the tree whenever a new object is added to the rendered scene. An important implementation detail in PERCH is that every time a successor state is generated with a new object pose, local-ICP[24] is used to refine the pose of that object to account for discretization artifacts. Then, the successor state is re-rendered with the refined pose O_j to obtain $\Delta\tilde{R}_j$ (the ICP adjusted point cloud containing points of R_j that belong exclusively to object O_j), which is then used to compute the edge cost.

3.3 Local Base Control

The intrinsic controller executes short movements of the platform. While the SLAM solution described in Section 3.1 allows for accurate long distance travel including obstacle avoidance, manipulation requires exact short movements to approach graspable objects. The intrinsic controller utilizes track odometry and IMU sensor data to achieve localization. It is not exposed to the map frame of the transformation tree, nor is it exposed to the occupancy grid generated by the SLAM solution. The intrinsic controller also reads the force and torque data from the wrists of the robot to determine the current forces applied to the end effector.

Goals are sent to the intrinsic controller in the form of a limb, a desired end effector location, and a maximum wrench threshold that can be applied to the end effector. The intrinsic controller then determines the twist to reach that goal and iterates over the current location until it has reached the goal location. If the end effector exceeds the force or torque thresholds during the movement, the base will stop and return an error indicating the wrench reading. This ensures the end effector and arm will not be damaged during the movement of large heavy objects, and informs higher level planners of the applied wrench.

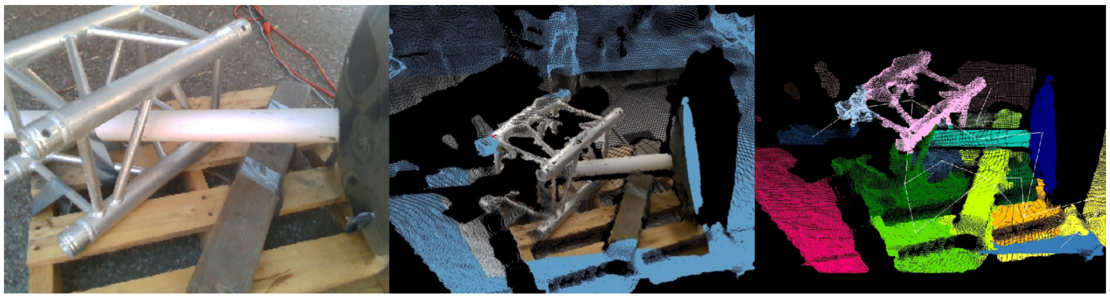


Figure 8. Left: RGB image of example debris pile containing aluminum truss segment, safety barrier, 4x4 wood section, and pallet. Center: Point cloud of workspace captured with RealSense D435. Right: Singulated object candidates with geometric adjacency from Locally Convex Connected Patches algorithm.[25]

3.4 Grasp Planning

To affect prehensile manipulation of objects within the task scene, the platform must be capable of both identifying grasp regions compliant with the specified task, and synthesizing end-effector poses within that region that are geometrically suited to the gripper. These are divided into two distinct processes, termed region of interest selection, and grasp pose synthesis, as described below.

3.4.1 Region of Interest Selection

As manipulation focuses on objects within the reachable workspace of the platform, the grasp planner solely utilizes the colored pointcloud from the near-sighted PTU RGB-D camera to both infer task context and plan grasp poses in geometric space. This provides both an RGB image (Fig. 8, left), and a pointcloud representation from IR stereo (Fig. 8, center).

When operating in unstructured environments with little to no *a priori* information on the objects that may comprise it, this geometric and color information may be all that is available to reason about the suitability of regions within the scene. The computer vision community has proposed a wide variety of algorithms to address this particular problem, and one such algorithm was employed to infer geometric structure within the pointclouds, namely Locally Convex Connected Patches.[25] This abstracts points within the scene into “super-voxels”, aggregated point clusters across regions with similar normals, and then seeks to group them into convex regions by comparing the mean normals of adjacent regions. Points may then be labelled as belonging to particular assumed convex regions, as depicted with a label to color mapping (see Fig. 8, right).

These segmented regions may then have meta-parameters calculated, such as center of mass (mean point value), volume (from convex hull), or edge points, that can be formulated into a region selection criteria based on the manipulation task at hand. In the example of removing objects from a pile, for instance, a naïve approach that proved effective for simple piles was to designate a “priority point” at an $\mathcal{R}^3$ point above the workspace, then select the object centroid closest to this point, which was therefore the topmost with respect to gravity. The points belonging to the select region are then provided to the geometric planner as the target object, along with the greater pointcloud, which is used to ensure grasp candidates will not collide with surrounding objects.

3.4.2 Grasp Pose Synthesis

Once a task suitable grasping region within the workspace has been determined, end-effector positions that will allow the selected gripper to achieve a mechanically stable grasp (termed *candidate grasp points*) must be found for the points comprising the region. This is achieved through application of a library of known suitable grasp positions for the given gripper, entries which are comprised of geometric object primitives and their associated hand positions, as in Fig. 9. These pairs form a set of *grasp prototypes* that may be fit to a given task scene to produce candidate grasp poses. Points within the region of interest are sampled, and geometric object primitives randomly chosen from the library (or a subset) fitted to the surrounding points to produce a “grasp score” via kernel methods,[26] which represents the quality of match between the prototype and the candidate pose.

After each candidate grasp is sampled, it is checked against a range of additional criteria for viability, each of which may be a function of task specific parameters that are specified in the call to the grasp planner. These include allowed approach angle of the gripper (such as to prevent grasping from underneath), which is defined as an $\mathcal{R}^3$ approach vector and $\mathcal{R}$ deviation angle tuple; and $\mathcal{R}^3$ position within the workspace. Grasp candidates that pass these less computationally intensive checks are subject to collision checking with the gripper model (depicted as grey regions in Figure 9) against pointcloud elements surrounding the region of interest. Finally, candidates that will not result in collisions are checked for reachability via a call to an external Inverse Kinematics solver, such as MoveIt!.

All grasps passing the full set of checks are then returned by the grasp planner to the module that requested them, in a vector ordered by *grasp score*, such that the grasps with closest match to a prototype may be prioritized in higher level planning. In the context of removing items from a pile, or grasping anti-vehicle barriers, this processing takes the form of attempting to plan joint-space trajectories from the present configuration to the desired end-effector pose, offset along the grasp direction from the candidate pose by 13 cm so as to prevent the target object being considered a collision. Upon finding a plan and moving the arm to this *pregrasp* pose, a Jacobian-based Cartesian motion controller may then be employed to linearly move the gripper towards the object until contact is detected, at which time the gripper is closed.

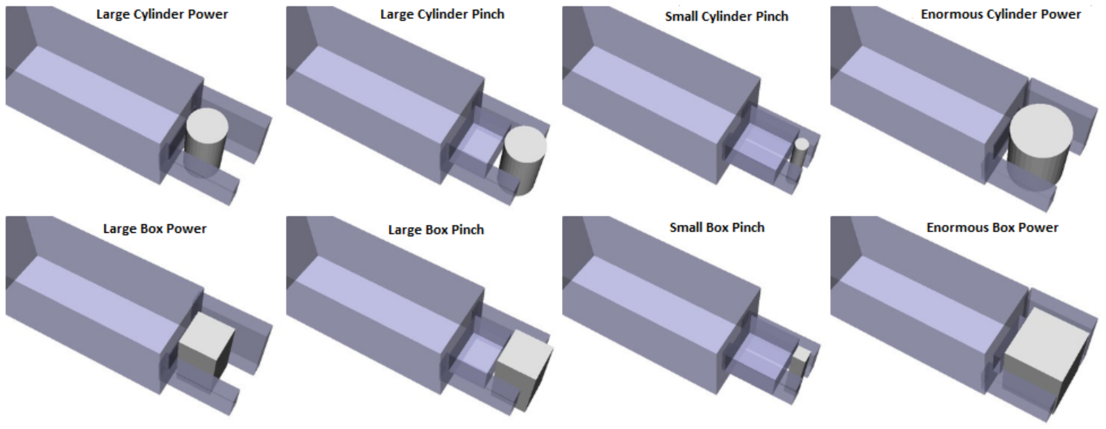


Figure 9. Geometric object primitives (in white) and their associated hand positions (in grey). These pairs form a set of *grasp prototypes* that may be fitted to a given task scene to produce candidate grasp poses.

3.5 Arm Planning and Manipulation

3.5.1 Free Trajectory Planning

RoMan's motion planning framework provides it the capability of fast and feasible manipulation planning for both arms in cluttered environments. Given a set of goal poses by the grasp planner, RoMan invokes multiple motion planning algorithms in parallel threads until a solution is successfully computed: Anytime Repairing A* (ARA*)[27] and Generalized Lazy Search (GLS)[28] search for a feasible solution on a graph in the 8-dimensional configuration space of the arm.

ARA*[27] is an anytime heuristic search algorithm which runs A* with inflated heuristics in succession while search time is available. ARA* efficiently reuses search efforts from previous iterations and continuously reduces the inflation factor to lower the sub-optimality bound over the solution. Our implementation is based on the Search-based Motion Planning Library (SMPL)* and MoveIt![13] framework.

GLS[28] is a computationally efficient, lazy shortest path planning algorithmic framework. Lazy search algorithms delay collision checking until absolutely necessary. GLS operates with the key insight that *interleaving* lazy search and collision checking reduces the total computational time stemming from search efforts as well as collision checks before terminating with the shortest path. We use the open source implementation of GLS† with the AIKIDO‡ framework.

The planning algorithms check for path feasibility by collision checking against a voxelized representation of the debris and obstacles around RoMan. An arbiter receives feasible plans from the planners and returns the first available plan for execution.

*https://github.com/aurone/smpl
†https://github.com/personalrobotics/gls
‡https://github.com/personalrobotics/aikido

3.5.2 Articulated Object Manipulation

RoMan also has the ability to manipulate articulated objects. The underlying motion planner used to generate a joint space trajectory for the arm is an Experience-graph (E-graph) based algorithm.[29] We embed a user-generated demonstration in the E-graph, and compute a joint-space trajectory based on this graph that can be used for manipulating articulated objects.

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 to incorporate user-generated demonstrations[30] in G_e. Since the demonstration shows how to manipulate objects, we increase the state-space by one, where the new dimension tracks the state of the object being manipulated. As a result of this, both G and G_e are updated to account for this added dimensionality in the state space. In addition, the heuristic, H_e is updated to account for the added dimenstionality. The planToManipulate algorithm shows the high-level framework.

Algorithm 1: planToManipulate

```
planToManipulate($G, D, s_{start}, z_{goal}, obj$);
1: $T = T_{obj} \in D$ ;
2: $G_{manip} = buildGraph(G,T)$ ;
3: $G^E = createEGraph(T)$;
4: $\pi = findPath(G_{manip}, G^E, T, s_{start}, z_{goal})$ ;
5: $return\ \pi$ ;
```

First, the demonstrations corresponding to the object that is to be manipulated are obtained. Here, $D = \langle T_1 ... T_m\rangle$, where each T_i is a set of discretized trajectories corresponding to the i^{th} object in the environment. Using these demonstrations, the graph G_{manip} is constructed, which consists of the updated state-space and heuristic definition, based on the task specified. The *createEGraph* function uses the demonstration to create the Experience Graph G^E. Finally, a planner is run on the two graphs as described in the original E-graph paper.[29]

3.5.3 Heavy Object Dragging

When manipulating large or heavy objects (e.g "czech hedgehog" anti-vehicle barrier), moving the end-effector with an object in-hand can cause large reaction forces to be exerted on the digits, potentially exceeding strength limits and damaging the system. To account for this, local Cartesian motions of the end-effector may utilize live measurements from the wrist-mounted force-torque sensor to reorient the gripper to reduce these reaction forces, in essence creating "software compliance". This is achieved by deflecting the goal pose of the Cartesian motion as a linear function of the measured wrist torque and a pre-specified compliance parameter.

An illustrative example is when one support of the anti-vehicle barrier is lifted prior to executing a drag behavior. When the leg of the barrier closest to the platform base is grasped and raised, the barrier's center of mass is distal to the grasp point, causing a torque to be exerted upon the end-effector as it begins to support the barrier's weight (in the positive Y direction of the Forward Left Up coordinate base frame). By actively rotating the end-effector about the Y axis during the motion, this torque can be reduced, allowing the rear legs of the barrier to support more of the barrier's weight.

To provide an intuitive means of specifying compliance, deflection is calculated in a hybrid frame (denoted hyb), located at the origin of the end-effector frame (ee), but with an orientation matching that of the platform base frame. By specifying non-zero torsional compliance about the Y-axis, for example, the gripper will only allow rotation about the Y-axis in the base frame (*robot*) during a motion, rather than in a frame attached to the instantaneous orientation of the end-effector. This is achieved by transforming the *wrench* W (force torque pair) measured at the wrist into the hybrid frame through equation 4, where Ad_{b2a}^{T} is the transposed *adjoint transformation* of a wrench from frame a to frame b,[31] and $R_{robot2ee}$ is the rotation between the instantaneous end-effector frame and the robot base frame.

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

Compliance may then be specified anisotropically via a 6x6 diagonal matrix, C_{hyb}, which converts the hybrid frame wrench into a deflection in *twist* coordinates (the dual space of wrenches).[31] This is then transformed back into the end-effector frame to produce $G_{ee2ee_{defl}}$, which is the homogeneous transform representing the deflection of the end-effector from its goal pose due to the software compliance:

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

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

The net result is an allowance for deviations from the goal in directions that are not critical to achieving the task, which limits internal forces and reduces the risk of damage or task failure.

4. TESTING AND RESULTS

To determine RoMan's effectiveness at autonomously performing human-scale tasks, two notional scenarios were used for testing. First, RoMan can be used to clear debris in the form of multiple stacked objects that RoMan can move freely, or a heavy object that RoMan must drag, such as a downed tree or Czech hedgehog anti-vehicle obstruction. Second, RoMan can be deployed to retrieve an object, including one inside of a hinged container. The sections below provide results on initial testing of each of these scenarios.

4.1 Debris Clearing

The scenario motivating the Debris Clearing Task is as follows:

> A robotic vehicle was deployed on a scouting mission. While performing route reconnaissance, it encountered debris, both light and heavy, blocking its path. With no obvious way around the debris, the vehicle deployed RoMan to clear the path so that the larger vehicle could safely continue its mission. RoMan is given a desired goal position. It proceeds along a path to that goal until it encounters an obstacle. It determines that there is no way around the obstacle, so it grasps the obstacle and attempts to move it. The result of the attempt is either: a) RoMan determines that it cannot move the obstacle, b) RoMan determines the obstacle is too heavy to lift freely, so it attempts to drag the obstacle out of the way, or c) RoMan is able to lift the obstacle and carry it. After removing the first debris object, RoMan continues to clear debris until the path is clear, and it can get to the desired goal position. The cleared space is wide enough for the larger vehicle to move through.

Debris was composed of one or more objects considered to be relevant to the task, either vertically stacked or adjacently placed, but not interlocking or unstable. Individual objects were chosen to represent a range of masses and geometries, both primitive (e.g. prisms, cylinders) and complex (e.g. tree branch), rigid (e.g. metal truss) and soft (e.g. cone, bag). Table 2 provides a list of sample objects and their specifications.

4.1.1 Debris Clearing Integration

Our progression, shown in Table 3, was planned for increasing complexity, from removal of a single, low-mass object to dismantling a stack containing multiple objects. As we integrated the required behaviors (component actions), we further improved them with new features as needed in each subsequent sprint. Behaviors were tested individually, in parts of a sequence, and end-to-end, where the full workflow is shown in Fig. 10. Note that strict inter-dependencies among behaviors required parallel testing with development and integration.

Item	Mass	Dimensions
Short Aluminum Truss Segment	6 kg	0.5 m x 0.3 m x 0.3 m
Safety Barrier	7.5 kg	1 m x 0.45 m diameter
PVC Pipe	2.2 kg/m	0.1 m diameter x various lengths
Small Traffic Cone	3.2 kg	0.36 m x 0.36 m x 0.71 m
Wooden Blank	2 kg	0.05 m x 0.05 m x 0.61 m
Czech Hedgehog	18 kg	1.3 m x 1.3 m x 1.3 m
Tree Branch	unknown	unknown

Table 2. Debris pile item candidates with their estimated masses and dimensions.

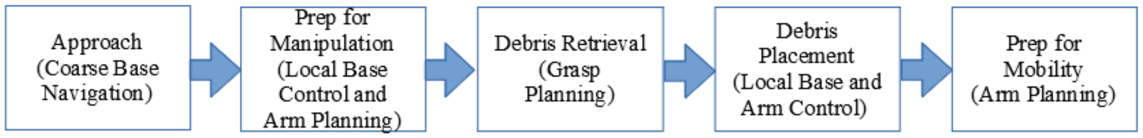

Approach (Coarse Base Navigation)
Prep for Manipulation (Local Base Control and Arm Planning)
Debris Retrieval (Grasp Planning)
Debris Placement (Local Base and Arm Control)
Prep for Mobility (Arm Planning)

Figure 10. Debris Clearing Workflow.

Beginning with sprint 2, we endeavored to reduce the grasp behavior execution times, which included optimizations in grasp planning such as reducing number of selected grasps to filter via inverse kinematics (IK) and reducing candidate grasps given to the arm planner. Multiple debris objects also presented challenges in prioritizing the regions of the pile to pursue (see section 3.4). Arm planning with obstacle avoidance, which considerably increased planning times, was also desired to prevent collisions with non-targeted debris. This emphasized the need for planning optimizations (see section 3.5).

For the final sprint, the pipeline was tested for heavy objects that RoMan was incapable of lifting entirely off the ground. This sprint relied on changing grasp behavior parameters to prioritize grasps in lower regions. We also incorporated force and torque feedback in the local controllers to enforce safety stops and “software compliance” when manipulating heavy objects (see section 3.5.3). Several attempts were made to remove a single heavy object, but no complete end-to-end successful run was achieved in the testing for this paper. That said, RoMan succeeded in all portions of the task in pieces, but required retuning the grasp behavior and adding new prototypes. Still, grasping the hedgehog proved difficult with the Robotiq hand due to the lack of accuracy in the tool frame placement (as compared to the Camhand). Releasing the large tree branch during placement was also an occasional issue, as the hand would sometimes fail to open due to internal forces resulting from the drag process. Simple placement using straight backward base movement was sufficient to remove the heavy objects from the path without base planning; though, more end-to-end testing is required to test navigation after dragging.

4.1.2 Debris Clearing Task Results

We started with 8 single test runs for Sprint 1. Unfortunately, we later discovered wheel encoder errors caused unreliable state estimation, thus invalidating our Sprint 1 test results.

Sprint 2 was successful overall, especially with certain objects, such as the truss. The component action sequences were streamlined, and several new behaviors were added to both ease transitions between runs and remove wrapped executors into more standalone behaviors. This helped restructure the mission logic through quicker configuration changes and made the debris clearing mission easier to repeat. There were 110 single object clearing runs, mostly consisting of portions of the full run, with an overall test success rate of 30%.

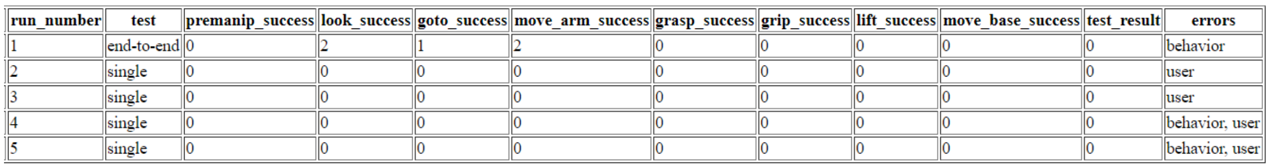
Several runs were attempted for Sprint 3; though, while grasping stacked objects in a small pile (2-3 items) was successful in steps, a full run ending with navigation to goal was not achieved. There were 11 multiple debris object runs with 64% success on grabbing at least 1 object.

Finally, we had 15 heavy object runs for Sprint 4, including both the Czech hedgehog and tree branch, with 60% success; although, we manually stepped through the behaviors instead of using the scripted sequences. As shown in Fig. 12, our success rates increased as we progressed through the sprints, indicating that our changes succeeded in improving

#	Sprint	Component Actions	Additional Features	Complexity
1	Go to the end of the street and stop at an immovable obstacle.	Coarse Base Navigation	Goto object	Low
2	Go to the end of the street and remove a single light obstacle from the path.	Coarse Base Navigation, Local Base Control, Local Arm Control, Arm Planning, Grasp Planning	Neck controller, Gripper Control, Force/Torque triggers	Moderate
3	Go to the end of the street and remove multiple light obstacles from the path.	Coarse Base Navigation, Local Base Control, Local Arm Control, Arm Planning, Grasp Planning	ROI generation, OA in Arm Planning, Neck planning	Moderate
4	Go to the end of the street and remove a single heavy obstacle from the path.	Coarse Base Navigation, Local Base Control, Local Arm Control, Arm Planning, Grasp Planning	Wrench reactive control	High

Table 3. Debris Clearing Sprints

robustness. Fig. 13 shows the distribution of our errors, which helped identify the subsystems with the highest frequency of failures. In sprints 2, 3, and 4, the local maxima were either around 0 (no errors) or 3-3.5 (actions and behaviors). We also noticed a slight decrease from 0.5 to 0.3 as we progressed through the sprints, which again is an indication of the steady improvements.



run_number	test	premanip_success	look_success	goto_success	move_arm_success	grasp_success	grip_success	lift_success	move_base_success	test_result	errors
1	end-to-end	0	2	1	2	0	0	0	0	0	behavior
2	single	0	0	0	0	0	0	0	0	0	user
3	single	0	0	0	0	0	0	0	0	0	user
4	single	0	0	0	0	0	0	0	0	0	behavior, user
5	single	0	0	0	0	0	0	0	0	0	behavior, user

Figure 11. Results of the first 5 debris clearing experiments of Sprint 2. Each was for a single truss object.

Description of Variables:

- date and time: the recorded date and time of each test run.
- test_type: these are categorized as single behavior, multiple behaviors, or end-to-end runs.
- obj_count: the number of items in the debris pile.
- obj_types: available object types are from Table 2.
- <behavior>_success: the number of successful attempts. The criterion for success is suggested in the aforementioned sections for the respective behavior.
- test_result: the overall success or fail of the run.
- errors: these are categorized based on where the error resides in the successive layers of abstraction (see Section 2.2). An error score of 0 to 5 is given as follows: 0 means no errors observed; 1 indicates error due to hardware failure or damage; 2 indicates error due to controller logic; 3 indicates error due to logic in action planner; 4 indicates error in behavior (component action); 5 indicates configuration and/or operational error.

4.2 Container Opening and Object Retrieval

This task was based on a notional mission to retrieve a valued item from inside a known container in an unsafe area. Thus, RoMan needed to be able to locate the container, navigate to it, open it, and retrieve the item from within. This workflow, illustrated in Fig. 14, was implemented as a behavior tree using the RCTA intelligence architecture's Mission Executor.

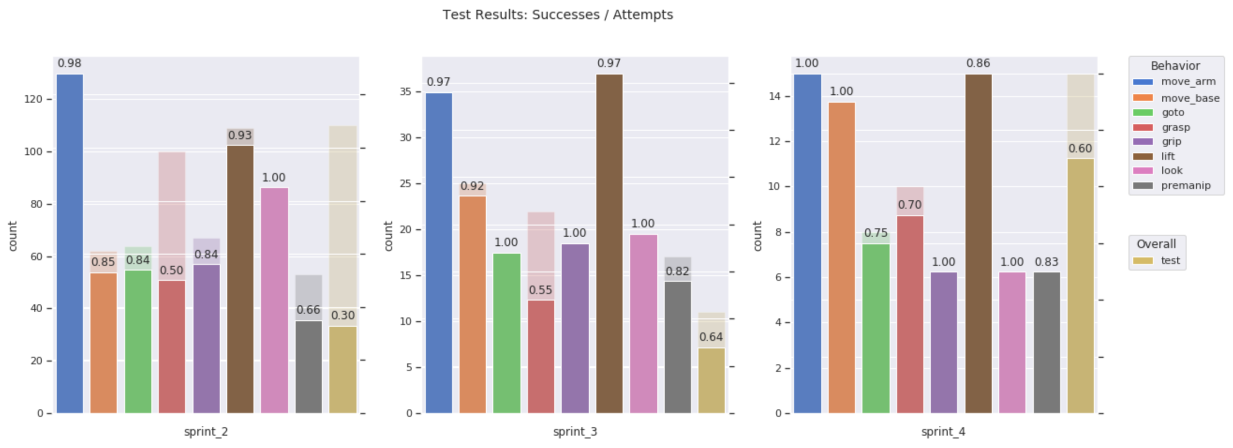


Figure 12. Debris Clearing Sprint Results. Behavior success rates are displayed above their respective bar, and the overall test success rate is the rightmost bar in each plot. Transparent bars represent number of attempts. Left: Navigate to the end of the street, removing a single light obstacle. Center: Navigate to the end of the street, removing multiple obstacles. Right: Navigate to the end of the street, removing a single heavy obstacle.

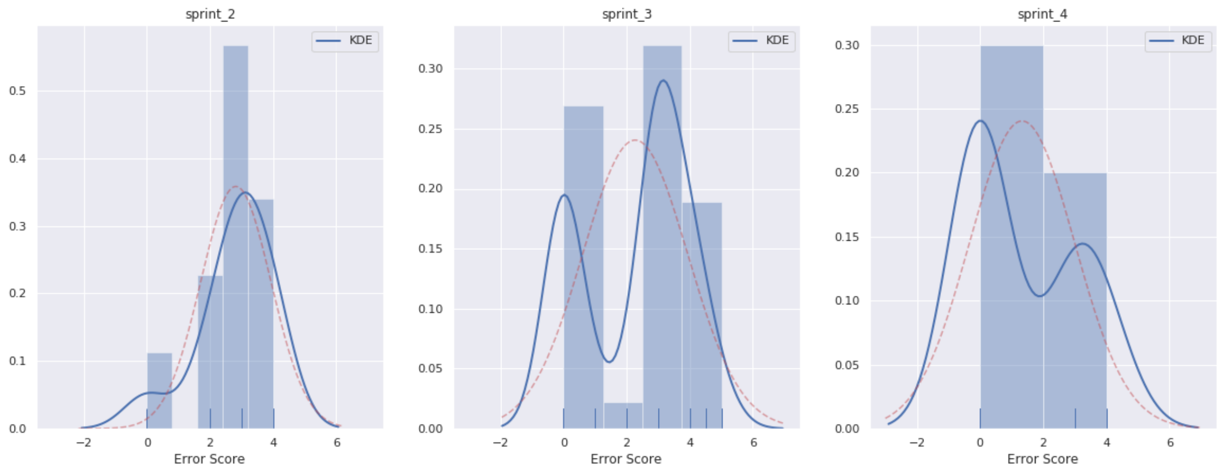


Figure 13. Debris Clearing Error Results. Distribution of error scores showing the Kernel Density Estimation (blue line) and the normal distribution (red line), one sprint in each plot. Left: Navigate to the end of the street, removing a single light obstacle. Center: Navigate to the end of the street, removing multiple obstacles. Right: Navigate to the end of the street, removing a single heavy obstacle.

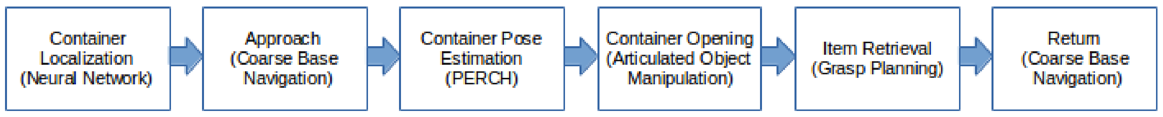

Container Localization (Neural Network)
Approach (Coarse Base Navigation)
Container Pose Estimation (PERCH)
Container Opening (Articulated Object Manipulation)
Item Retrieval (Grasp Planning)
Return (Coarse Base Navigation)

Figure 14. Container Opening and Item Retrieval Workflow.

4.2.1 Container Task Manipulation Integration

Work began integrating the Container Task component behaviors without a usable perception solution. Thus, the pipeline was initially tested with only known container poses. The container pose was taken in 2D, as it was assumed that container and robot were on flat ground and that the height of the container was constant. The object manipulation planner (container opener) was determined to be fairly sensitive to errors in container pose estimation. Initial testing at the University of Pennsylvania indicated a 5% success rate for the object manipulation planner when the actual pose deviated from the estimated pose by more than 2 cm.

Preset container poses were directly input into the object manipulation planner via ROS messages during initial integration. When the container could be opened this way with some degree of reliability, three fiducial markers (AprilTags) were added to the container in known locations in the container frame. Due to the unreliability of AprilTag orientation, the AprilTag container detector relied solely on tag position. As long as the torso-mounted camera had a view of two unoccluded tags, it was able to determine container pose yaw with sufficient accuracy to regularly open the container.

With the system able to open the container, the next task was to retrieve the bag from within. The grasp planner module was fairly mature at this time, and integrated easily into the behavior. However, initial attempts to use the same limb to both open the container and remove the item were rarely successful, as the open container lid became a significant obstacle for that limb to plan around without knocking the lid closed. Fortunately, the dual-arm RoMan configuration became available around this time, and using the second limb to grab the item out of the bag resulted in a higher success rate, speed, and improved visual appeal.

4.2.2 Container Task Perception Integration

Once the available perception modules were more mature, work began on integrating a close-up (fine) container pose estimation solution to eliminate the need for AprilTags. Two candidate algorithms were considered – the neural network semantic keypoint detector (see section 3.2.1, and the perception through search (PERCH) approach (see section 3.2.2). The keypoint detector, while faster than PERCH, proved to be highly susceptible to minor occlusions, and rarely provided a container pose accurate enough at close range to allow the container opening to succeed. With this realization, work focused on improving the performance of PERCH. By moving the CPU-based PERCH algorithm onto the fastest machine available on-board RoMan, and severely limiting the container search space based on task invariants and prior pose information, integrators achieved a 10X speed-up of PERCH, resulting in an acceptable solution.

4.2.3 Container Task Results

Unfortunately, container pose errors from the far-field neural network object detection and pose estimation combined with the inaccuracy of the coarse base navigation resulted in RoMan rarely being able to drive up to the container within the tight tolerances required by the container opening algorithm. However, the remainder of the pipeline, including the near-field search-based perception algorithm, succeeded in opening a container placed directly in front of the platform and removing the bag with a success rate near 80%.

5. CONCLUSIONS AND FUTURE WORK

RoMan is a robotic mobile manipulation platform capable of autonomously performing human-scale work in unstructured environments. In this paper, we provided a detailed account of the overall hardware system and outlined the software architecture and the integration of its component actions including coarse and local base navigation, two approaches to object recognition and pose estimation, grasping, planning, and manipulation of free, heavy, and constrained objects. We also tested and provided initial results for RoMan's performance on two notional tasks: debris clearing and object retrieval from a hinged container. The advances we have made in each of these areas bring us one step closer to realizing a fieldable mobile manipulation surrogate that can protect humans by performing human-scale tasks in potentially dangerous environments.

Despite this, significant gaps in robustness remain. We intend to continue bridging these gaps by transforming our currently scripted mission executor into a behavior tree approach with condition checking. This will enable the system to more adequately determine when and how it has failed to perform a sub-task, and be more likely to correct the fault without relying on human intervention. Time continues to be an issue as well, with RoMan under-performing human speeds for most tasks by at least an order of magnitude. Algorithms will be assessed for bottlenecks and further parallelized where possible. Finally, we intend to explore dynamic whole-body actions using multi-body dynamics analysis, providing the potential to achieve larger forces and more varied tasks without requiring an increase in system mass.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. 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.

REFERENCES

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

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

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

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

[5] Fitzgerald, C., "Developing baxter," *2013 IEEE Conf on Technologies for Practical Robot Applications (TePRA)* , 1–6 (2013).

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

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

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

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

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

[11] Brooks, R. A., "A Robust Layered Control System For A Mobile Robot," *IEEE Journal on Robotics and Automation* **2**(1), 14–23 (1986).

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

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

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

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

[16] Segal, A., Haehnel, D., and Thrun, S., "Generalized-ICP," (2009).

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

[18] Likhachev, M. and Stentz, A., "R * Search," *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (2008).

[19] Massa, F. and Girshick, R., “maskrcnn-benchmark: Fast, modular reference implementation of instance segmentation and object detection algorithms in pytorch,” (2018).

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

[21] Narayanan, P., Yeh, B., Holmes, E., Martucci, S., Schmeckpeper, K., Mertz, C., Osteen, P., and Wigness, M., “An integrated perception pipeline for robot mission execution in unstructured environments,” *SPIE Defense + Commercial Sensing* (2020).

[22] Pavlakos, G., Zhou, X., Chan, A., Derpanis, K. G., and Daniilidis, K., “6-dof object pose from semantic keypoints,” *Proceedings of 2017 IEEE international conference on robotics and automation (ICRA)* , 2011–2018, IEEE (2017).

[23] Bowman, S. L., Atanasov, N., Daniilidis, K., and Pappas, G. J., “Probabilistic data association for semantic SLAM,” *Proceedings of 2017 IEEE International Conference on Robotics and Automation (ICRA)* , 1722–1729, IEEE (2017).

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

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

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

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

[28] Mandalika, A., Choudhury, S., Salzman, O., and Srinivasa, S., “Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles,” *Proceedings of the International Conference on Automated Planning and Scheduling* **29**(1), 745–753 (2019).

[29] Phillips, M., Cohen, B. J., Chitta, S., and Likhachev, M., “E-graphs: Bootstrapping planning with experience graphs.,” (2012).

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

[31] Murray, R. M., Li, Z., and Sastry, S. S., [*A Mathematical Introduction to Robotic Manipulation*], CRC Press (1994).