Model-based autonomous system for performing dexterous, human-level manipulation tasks

Citation: Hudson, Nicolas et al. "Model-Based Autonomous System for Performing Dexterous, Human-Level Manipulation Tasks." Autonomous Robots 36.1–2 (2014): 31–49.

As Published: http://dx.doi.org/10.1007/s10514-013-9371-y

Publisher: Springer US

Persistent URL: http://hdl.handle.net/1721.1/105451

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Auton Robot (2014) 36:31–49
DOI 10.1007/s10514-013-9371-y

Model-based autonomous system for performing dexterous, human-level manipulation tasks

Nicolas Hudson · Jeremy Ma · Paul Hebert · Abhinandan Jain · Max Bajracharya · Thomas Allen · Rangoli Sharan · Matanya Horowitz · Calvin Kuo · Thomas Howard · Larry Matthies · Paul Backes · Joel Burdick

Received: 1 March 2013 / Accepted: 11 October 2013 / Published online: 28 November 2013

Abstract This article presents a model based approach to autonomous dexterous manipulation, developed as part of the DARPA Autonomous Robotic Manipulation Software (ARM-S) program. Performing human-level manipulation tasks is achieved through a novel combination of perception in uncertain environments, precise tool use, forceful dual-arm planning and control, persistent environmental tracking, and task level verification. Deliberate interaction with the environment is incorporated into planning and control strategies, which, when coupled with world estimation, allows for refinement of models and precise manipulation. The system takes advantage of sensory feedback immediately with little open-loop execution, attempting true autonomous reasoning and multi-step sequencing that adapts in the face of changing and uncertain environments. A tire change scenario utilizing human tools, discussed throughout the article, is used to described the system approach. A second scenario of cutting a wire is also presented, and is used to illustrate system component reuse and generality.

Keywords Autonomous · Manipulation · Estimation · Dual arm · Tool use · Task sequencing

1 Introduction

Autonomous robotic manipulation in uncertain environments is poised to become a beneficial technology in manufacturing, in-home care, and in projecting human-like capabilities into hazardous situations or environments. The presented work is part of the DARPA Autonomous Robotic Manipulation Software (ARM-S) program which emphasizes the completion of complicated tasks with only high level human guidance or supervision while adapting to dynamic unstructured environments.

The ARM-S program currently consists of three teams, each developing autonomy software on a local copy of the ARM robot (Fig. 1). Each team delivers software to DARPA for evaluation on the remote robot. This independent testing finds the breaking point of each algorithm, and enforces robustness to environmental factors such as lighting, room and floor background, and differences between the test and development robot.

The end-to-end system described in this paper is an extension of our DARPA ARM Phase 1 system (Hudson et al. 2012). This Phase 1 work focused on short independent tasks, which are discussed briefly here. The important conclusion of this work was that ARM teams were able to produce a general manipulation system capable of high reliability (>90 %) for short tasks (Table 1).

This article describes the ongoing Phase 2 manipulation system capable of executing sequences of single and dual

Electronic supplementary material The online version of this article (doi:10.1007/s10514-013-9371-y) contains supplementary material, which is available to authorized users.

N. Hudson (✉) · J. Ma · P. Hebert · A. Jain · M. Bajracharya · C. Kuo · T. Howard · L. Matthies · P. Backes
Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena 91109-8099, CA, USA
e-mail: nhudson@jpl.nasa.gov

T. Allen · R. Sharan · M. Horowitz · J. Burdick
California Institute of Technology, 1200 E California Blvd, Pasadena 91125, CA, USA
e-mail: jwb@robotics.caltech.edu

Present Address:
C. Kuo
Stanford University, Stanford, CA, USA

Present Address:
T. Howard
Massachusetts Institute of Technology, Cambridge, MA, USA

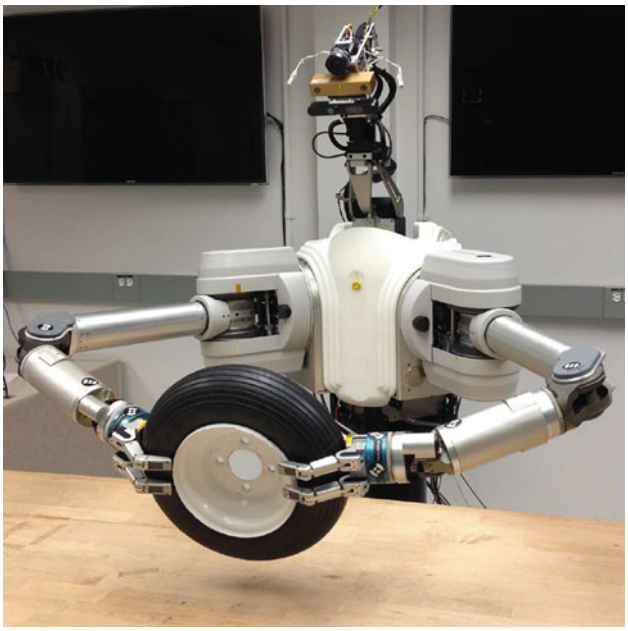


Fig. 1 The DARPA ARM robot used in the program. The sensing head consists of a Point Grey Research Bumblebee2 color stereo camera, PrimeSense ASUS Xtion-Pro depth camera, Prosilica Gig-E color camera and two microphones on a 4-DOF neck. The arms are Barrett Technology 7-DOF WAM arms with a 6-DOF force sensor at each wrist. The Barrett BH8-280 hands have a strain gauge in each finger, and tactile sensing pads on the palm and distal finger surfaces

Table 1 DARPA ARM-S Phase 1 test results

Team	Successes (of 72)	Grasping (of 48)	Manipulation (of 24)	Average time (s)
JPL	67	47	20	75.4
B	67	47	20	80.6
C	64	46	18	77.5
D	58	47	11	125.7
E	58	41	17	170
F	49	42	7	151.8

arm tasks. However, the developed system has evolved from the demands, assumptions and procedures involved in Phase 1 testing.

In Phase 1, six teams were given a one-arm version of the ARM robot shown in Fig. 1, and tasked with performing tests focused on grasping, placing, and short manipulation sequences. The tests and the perturbations created by the DARPA test team forced each development team to create a system capable of executing robust behaviors. Grasping tasks were considered successful if the robot could pick an object clear off the table, and then place it at a specific target location. The evaluated grasping objects included: a ball, shovel, maglite, floodlight, radio, pelican case, rocks, screwdrivers, and hammers. Some of these objects had exact prior models, and had been previously seen by the robot and development team, but novel rocks, screwdrivers and hammers were tested with no previous observations. For these later objects only a canonical geometric model was given to give context to the autonomy system. The manipulation tasks were: turn on a flashlight, staple paper, hang up a phone, open a door, unlock a door with a key, and drill a block of wood at a specific target location. These results, a list of teams and a disussion of the ARM program is discussed in Hackett et al. (2013) by the DARPA test team.

In both Phase 1 and 2, all manipulation tasks (specifically manipulated tools) had exact prior models, and the developed autonomy system required prior labeling of feature points (triggers etc.). For all tasks, Creating effective closed loop behaviors was critical as the system has low open loop accuracy, with the end-to-end error between a location in the camera frame and manipulator end-effector position on the order of 5 cm.

The Jet Propulsion Laboratory (JPL), the University of Southern California (USC) (Pastor et al. 2011; Kalakrishnan et al. 2013), and the National Robotics Engineering Center (NREC) (Bagnell et al. 2012) teams have continued manipulation development in Phase 2 of the ARM program. All teams have demonstrated various strengths in their respective approaches.

The Phase 2 work described in this paper includes dual handed manipulation, and complicated interdependent task sequences. The following Sects. 1.1 and 1.2 describe these tasks, and are used to give an overview of system components.

The fundamental approach we have developed for autonomous manipulation relies on modeling the environment, augmenting apriori geometric object data with semantic labels, and associating libraries of behaviors with each object. While most objects can be 'grasped', robotic understanding of when to pull a drill's trigger requires additional and novel modeling of the world. We choose to make this model explicit. By conditioning on prior models, we interact with the environment, and refine object properties or poses, after deliberate contact is made (Hebert 2013). Understanding the uncertainty associated with an object allows rationalization over when or how to contact the object (Hebert et al. 2013).

1.1 Wheel change scenario

A wheel change-out task will be used to illustrate the technical advancements presented in this paper and is referenced in the sections which follow. Generic task steps include finding and assembling a battery operated impact driver, removing lug nuts from the wheel using the impact driver, removing the wheel from the axle, and replacing the wheel. The lug nuts must then be grasped, replaced and tightened with the impact driver.

Apart from containing difficult individual steps, this task sequence couples individual behaviors. For example, the initial grasp of the impact driver will affect the kinematic feasibility of lug nut removal.

With each task plan, the sequence initiates by visually scanning the world to detect, classify and to provide initial pose estimates of objects on the table that is positioned in front of the robot (Sect. 3.1).

The high level task directive is split into specific individual behaviors. A task level planner deliberates over all steps ensuring feasibility with the detected object poses (Sect. 3.5). The task planner then calls behaviors and monitors their completion.

The task level planner utilizes a manipulation planner (Sect. 3.3.2) which generates a manipulation set (Sect. 3.3.1) encoding possible behaviors and admissible arm positions from which these behaviors can be executed, given current object poses. For example, the manipulation planner will verify if the current impact driver pose allows a specific *tool_use_grasp* behavior, or if the driver must be relocated using a *power_grasp* behavior. Subsequent motions are between the current arm state and a point in the manipulation set is accomplished using the arm planner (Sect. 3.3.3).

Specification and execution of manipulation behaviors are governed by the manipulation controller (Sect. 3.4). Control behaviors are sequences of control actions defined as a set of task frame feedback controllers and end-conditions. These controllers allow delicate contact to be made with objects and exploration of their surfaces using feedback control.

As soon as contact with an object is made, the extra kinesthetic information is integrated by the world estimator (Sect. 3.2). Even before contact, visual servoing (Sect. 3.2.2) is utilized.

After the impact driver has been suitably grasped, it is used to remove each nut. Again the visual servoing, force control, and estimation of object positions are utilized to achieve high precision localization of the driver over the nut. Each nut is then unscrewed using the impact driver and placed on the table by positioning the nut socket vertically over the table. The impact driver is released and the wheel is bi-manually grasped. A dual-arm coordinated plan is generated to remove the wheel off the hub and to place it on the table (Sect. 3.3.3). A second visual scan of the scene is conducted to localize a second wheel, the hub and the unscrewed nuts. The wheel is again bi-manually grasped and a similar plan is generated to place the wheel over the hub. Force control is utilized to slide the wheel onto the hub. The nuts are then replaced and tightened.

1.2 Wire cut scenario

A wire-cutting task was also performed by the system and will be discussed in depth in the results and discussion sections of this paper (Sects. 4, 5) to provide further evaluation with experimental results of the overall system as a whole.

As a brief overview, the general task involved removing a burlap sack from the table to reveal a hidden junction box with a green wire running through it. A toolbag, placed also on the table, was then repositioned and unzipped to allow the robot to search the bag for the appropriate tool (a pair of trimmers). The tool was removed from the bag, re-positioned and re-grasped to properly cut the green wire. Overall success was achieved if the wire was cut.

Much like the wheel change-out task, our system was able to complete this wire-cutting task through the use of task plans to call individual behaviors that utilize manipulation planners to eventually perform each subtask.

2 Related work

There are currently several integrated manipulation platforms with autonomous end-to-end capabilities. These include HERB (Srinivasa et al. 2010) which uses a generalized approach to tracking movable objects (GATMO) (Gallagher et al. 2009). Grasping is performed using force closure and caging grasps, and manipulation of these objects are executed open-loop. The STAIR project (Quigley et al. 2007) uses a pan-tilt-zoom camera to obtain high resolution images of environmental objects for image feature detection. A learned classifier selects a nominal grasp location and optical proximity sensors are used to localize the grasper further before a grasp is executed open-loop. El-E (Jain and Kemp 2010) similarly does not use a physical model of its graspable objects, and also eschews the use of an environment map. A desired object is illuminated by a laser pointer before being visually segmented and then grasped with a parallel jaw gripper.

More recently, the (Knepper et al. 2010) investigate "focusing planning" in which high level tasks such as navigation are solved and begun before a complete solution is calculated. Solutions at different scales of a problem are stored for later use. Grasps are pre-computed for given objects and then culled at execution due to potential collisions, while objects are identified using SIFT-features from camera data.

Xue et al. (2012) use a time-of-flight camera to gather depth information which is segmented into known and unknown components of the environment, which are considered as obstacles. Grasping is done using sampling and simulated collision checking, with impedance control used to serve ice-cream at a trade fair.

Willow Garage has successfully completed a number of tasks, including door opening, inserting a plug into a socket (Bohren 2011), towel folding (Maitin-Shepard et al. 2010), and even cooking pancakes (Beetz et al. 2011). The first three required algorithms specific to detection of objects. The latter task also involved reasoning about probable locations for required objects, which were then manipulated through sim-

ple pick and place operations. The spatula was regrasped when a generalized distance to known good grasps was sufficiently low, then treated as an end effector. Further localization was performed by pressing the spatula against the pan. In this task, and others that used the Willow Garage PR2, feedback from tactile sensors were used to react to perceived poor grasps (Chitta et al. 2012), resetting the grasp planning if the grasp was poorer than expected. The division of more sophisticated tasks for the PR2 are handled through a hierarchical state machine SMACH (Bohren 2011) that are constructed a-priori.

These previous works are similar in that each uses the *sense-plan-act* paradigm wherein each component is performed in sequence with a return to *sense* only in the case of failure at a particular stage. The difficulty is that additional information that could be gathered during the execution of an act cannot be incorporated and is ignored. Furthermore, the necessity of interactive manipulation to further localize objects beyond visual sensor accuracy has only been recognized by these works in an ad-hoc way specific to individual tasks. For tasks that require finer manipulation (e.g. key insertion), such localizing motions may be necessary.

3 Technical approach

The following section will describe the technical approach of this paper. Figure 2 illustrates the system architecture, the main software modules/processes, the flow of data between the modules, and provides a guide to the following technical subsections.

3.1 Perception

Perception for the DARPA ARM robotic manipulator is comprised of segmentation, classification, and localization (6-DOF pose) of objects in the environment by using all the available visual sensors on the robot. At the beginning of any manipulation task, the perception subsystem performs an initial scene extraction to identify objects of interest in the world. This process begins by populating a voxel map which is then followed by segmentation and clustering processes that generate bounding boxes around objects and obstacles. A combination of iterative-closest-point (ICP) fitting of known 3D CAD models and contour matching is applied to specific task objects to attain a more accurate 6-DOF pose. Once the world is populated with known objects and obstacles, the perception subsystem continues to update its voxel map throughout the task and provides additional support for tracking and re-detection of objects and features when needed.

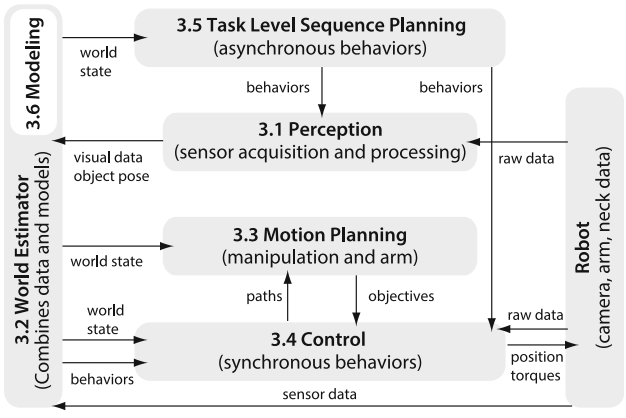


Fig. 2 The ARM-S system architecture. Each *block* represents a separate software module/process and each *arrow* indicates the flow of data between each module. The *block number* corresponds to a section in this paper

3.1.1 Mapping

The 3D voxel mapping and segmentation framework described in Bajracharya et al. (2013) and tested on a dynamic platform as described in Ma et al. (2012) is used in our framework. All points from the stereo camera and depth camera (Fig. 3a) are projected into the robot frame and associated voxels store calculated statistics: point averages, red-green-blue color averages, surface normals based on neighboring voxel information, and point variances (Fig. 3b). The accumulated map (Fig. 3c) is then segmented into regions of similar statistics, i.e. normal edges and faces, and optionally color (Fig. 3d). Bounding boxes are assigned to the extracted segments (Fig. 3e) and a subsequent clustering process groups overlapping segments into larger bounding boxes which are then later used for finer pose fitting.

3.1.2 Object detection

Because the DARPA ARM program is designed such that developed software is shipped and tested at a remote site, our developed object detection algorithms have to be robust to environmental change, particularly lighting. As such, our approach relies more heavily on geometric-based features and properties of objects rather than appearance-based features, though both are used.

3D CAD models of task objects are provided for all manipulation tasks in the DARPA ARM program which we use to our advantage to improve the overall 6-DOF pose estimate of task objects in the scene. An iterative-closest-point (ICP) algorithm is used in most cases to fit point clouds of segmented regions in the world to the provided models of known objects (Fig. 3f). As is often the case with ICP algorithms, incorrect 6-DOF pose solutions can occur at local minima and may result in poorly determined poses. To circumvent this issue, we seed the ICP algorithm with a best estimate of the object's pose from the world estima-

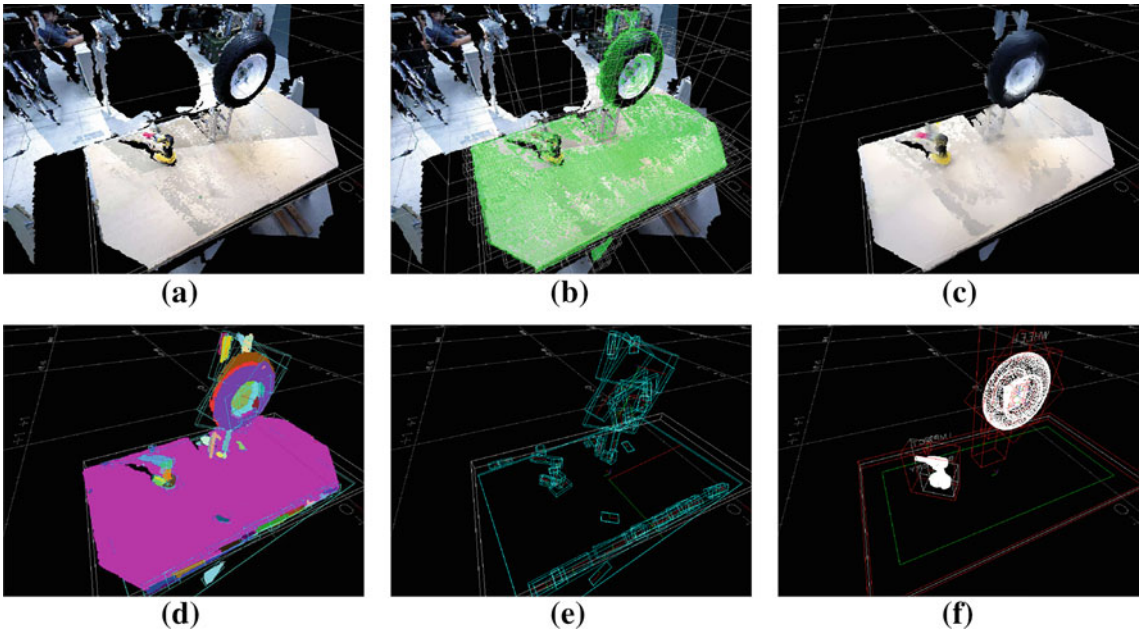


Fig. 3 A scan sequence of the environment prior to a manipulation task consists of the following steps: **a** point cloud data is collected from the depth camera and stereo camera at various neck angles, **b** a wireframe of the 3D-voxel map is used to accumulate statistics from point cloud data, **c** the smoothed 3D map includes average point locations, RGB/intensity, surface normals, and 3D variance, **d** segmentation is applied to the 3D map to group regions with similar statistics (i.e. normal vectors), **e** principal component analysis is applied to segmented regions to form statistically accurate bounding boxes, **f** segmented bounding boxes are clustered together and point fitting of model data is applied to determine accurate 6-DOF pose of task objects

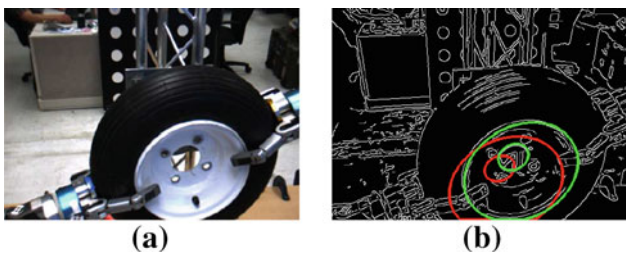


Fig. 4 **a** The rectified left stereo image of the wheel. **b** The resultant binary image from a Canny edge detection. The predicted model location from the world estimator is shown in *red*, and the contour fitted result against the detected edges is shown in *green*

tor if available (as described in Sect. 3.2); otherwise various different initial poses of the task object are used and the solution with the lowest score is chosen. Additionally, to ensure an even distribution of data points from segmented objects, we limit and store only 3 data points per voxel cell in the map.

For some task objects, 3D data may not exist or is unreliable due to specular surfaces or rounded edges on some objects. In these cases we apply a contour matching object detector that matches contours of a reprojected model of the object against current edge lines detected in the scene via a simple Canny edge detector (see Fig. 4). Priority is given to ICP-fitting for object detection, unless the threshold of necessary 3D points is not met, in which case we resort to using contour fitting.

3.1.3 Object tracking

In many test scenarios, visual feedback and tracking is utilized to achieve high precision requirements (Sect. 3.2.2). This requires in-step communications between the perception and world estimator subsystems to detect objects, or features of objects, and use those measurements to update the object's estimated pose at a relatively high rate.

Most task objects have distinguishing features that make the tracking process more robust. In the wheel changing task, the impact driver tool tip is marked red which allows for reliable detection of the tip. For tracking of this object feature (and similarly for others), we create a region of interest (ROI) in the camera image centered about an expected location provided by the world estimator (Fig. 5a). The ROI image is then thresholded on a pixel color (e.g. red pixels) and clustered into segments with the largest segment selected to give the x, y, z position of the tip (Fig. 5b). To get the orientation of the tip (an essential step to correctly place the tip onto the lug nut), we perform a single ICP process against a simplified model of the impact driver tip (a half cylinder, representing the observable portion of the red tip). To ensure a good fit, we only perform the last

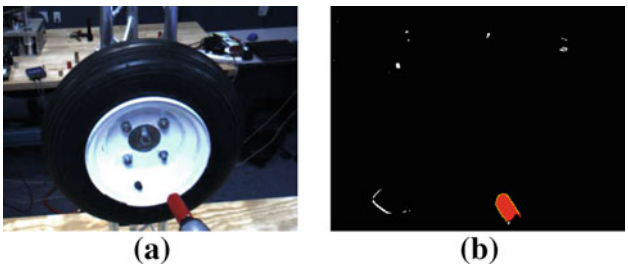


Fig. 5 **a** The ROI image of the impact driver tip. **b** The resultant binary image thresholded on *red*, with the largest red segment being the actual tip as indicated in *red*

ICP fitting when enough 3D points have been extracted from the previous step and return a failed status otherwise. The final detected tip is sent to the world estimator where additional filtering uses the tip pose to update the entire impact driver pose.

3.2 World estimation

A robust autonomous system must deal with uncertainty. An estimation framework providing continuous state information for closed-loop feedback is especially critical for successful precise manipulation. More importantly, in sequences of tasks composed of multiple steps, estimation is used in part for determining when key events have successfully completed. In particular, a world estimator continually maintains the state of the world W which includes the state of all objects $\{O\}$, and the state of the robot, R. The state of the objects includes a linkage list L that describes current geometric constraints between objects. Furthermore, the state of the robot includes a grasp state $\mathcal{G} = \{\mathcal{G}_l, \mathcal{G}_r\}$ that describes which object each manipulator is grasping.

3.2.1 Kalman filter framework

Our approach to estimation is entirely model-based. Recent work (Hebert et al. 2013) is incorporated for estimating coordinated bi-manual manipulation. An unscented Kalman filter (UKF) is implemented that continuously estimates both the robot state and the state of the object being manipulated, $X = \{X_R, X_O\}$. The robot state is composed of both the manipulator and neck joint bias, $X_R = \{\theta, \phi\}$. In the single arm case, the object state X_O, represents the object frame ($\mathcal{F}_O$) relative to the end effector frame ($\mathcal{F}_P$) and consists of the full 6-DOF pose (position and axis/angle) represented by the homogeneous transform G_{PO} (as shown in Fig. 6). However, in the dual arm case ($\mathcal{G}_l = \mathcal{G}_r = O$), an additional estimate of the object state, now relative to the opposite end effector, is included. As a result, a constrained UKF is implemented to ensure that the estimates of the object state in the bi-manually grasped case are consistent.

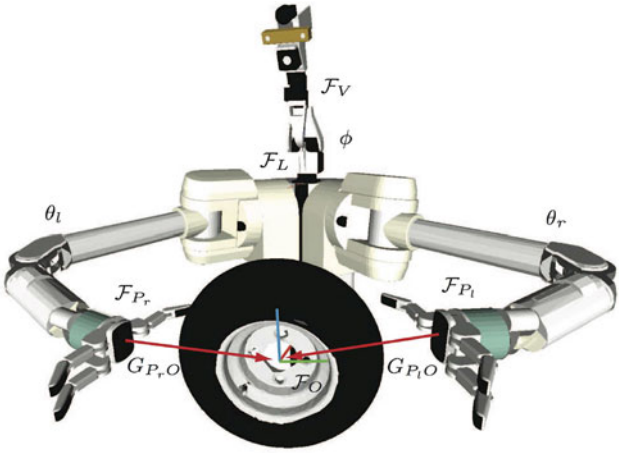


Fig. 6 Robot, object state and relevant frames

Measurements are naturally broken down into two kinds. The first kind is used to estimate the internal robot state while the second kind is used to estimate the object state. Estimating the robot state can be done in one of two ways. The first utilizes a multi-cue approach that combines shape, silhouette and appearance measurements from multiple modalities (Hebert et al. 2012). The second uses fiducuials placed on the manipulator and a visual detector provides 6 DOF measurements. Multiple fiducial measurements are then fused in the UKF. Figure 7 shows two detected fiducials and the estimated robot arm overlaid. These two approaches provides visual measurements in the visual frame ($\mathcal{F}_V$) and allows the filter to infer the biases in the joints of both the manipulator and neck.

Measurements used to estimate the object state may include kinesthetic and tactile measurements (which provide a point of contact measurement on the object surface) and force-torque measurements (which provide torques caused

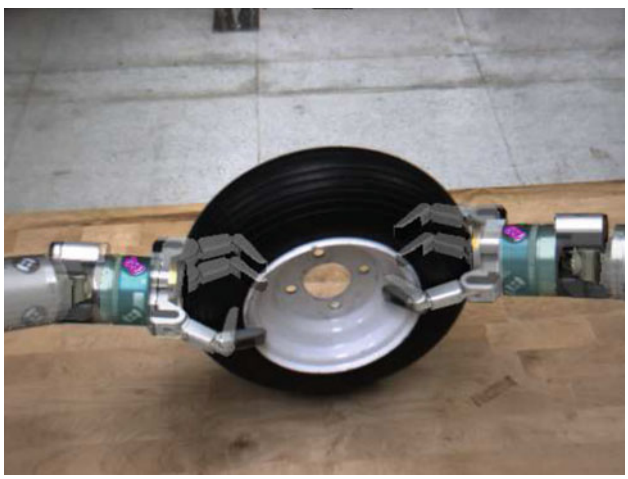


Fig. 7 Two fiducials detected shown in *red* and the overlaid estimated robot location

by the object's mass) (Hebert et al. 2011). Visual object measurements may include 3D features, such as SIFT or SURF or more generally 3D surface point clouds. In the case of the wheel change-out, the visual pose measurement of the red-drill tip is included into the measurement from which the object pose is inferred from the UKF.

3.2.2 Visual servoing

In some instances, vision-aided positioning of the robot end effector or the object is needed to successfully complete a task. The end effector *visual-servoing* behavior uses the fused robot state (i.e. joint biases) to compute a end effector correction used in grasping scenarios. Similarly, from the fused visual, kinesthetic, and tactile object measurements, an object-to-goal correction may be computed and used for manipulation scenarios such as the nut removal using the impact driver. These corrections are continuously estimated and used by the system, specifically the controller. In addition, the estimated in-hand object pose is used to compute expected joint torques and inverse dynamics controllers.

3.3 Motion planning

The motion planning system is composed of three components: an arm planner, a manipulation set generator, and a manipulation planner. The arm planner synthesizes collision free trajectories between two specified arm positions. The manipulation set is a collection of poses, either of the hand relative to an object (i.e. grasps), or poses of one object relative to another. The manipulation planner acts as an intermediary between the controller behaviors, which typically involve contact, and the arm planner, which requires collision free configurations.

3.3.1 Manipulation sets

A *manipulation set* is a set of relative poses associated with a specific controller behavior, a primary object and an optional secondary object. For example, the manipulation set for the behavior *power grasp* and the object *impact driver* is a set of poses of the hand relative to the impact driver. This manipulation set also stores the four parameters that specify the finger positions. For the behavior *attach* and the objects *wheel* and *hub*, the manipulation set is a set of the four poses the wheel takes when mounted on the hub (assuming a fixed hub, there are four possible configurations with which the wheel can be attached). Because CAD models of the objects are available, manipulation sets are generated offline. Some manipulation sets are generated manually, using a GUI to visualize the relative positions of the objects or hand. Others, such as *power grasps* are generated automatically, as described below.

Generation of *power grasps* involves producing a large number of potential starting poses, simulating the controller behavior (moving until the hand touches the object, then closing the fingers), and then evaluating each resulting grasp to produce a much smaller number of high scoring grasps. The starting poses are generated by placing the palm of the had against a randomly selected triangular element of the object model. A random rotation around the hand's z-axis and a random value of the finger spread between 0° and 90° are used as an initial pose. This pose is then backed away from the object by a fixed amount and checked for collisions. If this configuration is collision free, a grasp is generated by simulating pressing the hand into the object and closing the fingers, as described below.

The hand is moved towards the object in small increments, with the fingers in an open position, until contact is detected, and penetration of the hand into the object reaches a threshold, usually 1 mm. Then each finger is closed in small increments until contact is detected. For each contact, a position and a normal are calculated. Each grasp is then given a score based on these contacts, and the best 128 grasps are saved. The set is also pruned by hand to improve performance.

3.3.2 Manipulation planner

The manipulation planner acts as an intermediary between the arm planner, which only plans collision free paths, and the controller behaviors, which typically involve contact. The manipulation planner considers the position of objects in the world, the manipulation set, and the expected behavior of the controller to return a set of *pre-manipulation poses*. These poses are starting points for the controller behavior, and are collision free, to satisfy the arm planner's requirements.

A behavior is specified by an action type, a primary object, and an optional secondary object. For example, a grasp behavior is specified with a grasp type (e.g. *power grasp*) and an object (e.g. *impact driver*). Mounting the wheel on the hub is specified with the *attach* action, with *wheel* as the primary object, and *hub* as the secondary object. (Note that the manipulation planner only considers a single action. For example, the fact that the nuts must be removed before removing the wheel is not considered by the manipulation planner; this is done by the task level planner.)

Each controller behavior has an associated *expected motion* and a set of *potential motions*. The expected motion is the predicted nominal motion executed absent uncertainty when executing a given behavior. Potential motions are motions that are likely to be executed while running, and are updated based on the outcome of previous behavior executions. For example, the expected motion for a power grasp is a 5 cm move in the hand's +z-axis and one of the potential motions is a 10 cm move in the hand's +z-axis.

The pre-manipulation pose is computed to be the pose of the hand(s) such that the expected motion of the controller ends in the desired position. For example, the task of placing the wheel on the hub involves the controller behavior *attach*, the primary object *wheel*, and the secondary object *hub*. The expected behavior is a 5 cm move in the frame of the +x-axis of the primary object. To find the set of pre-manipulation poses, the manipulation planner starts from the pose of the hub. Each element of the manipulation set gives the position of the wheel relative to the hub, allowing calculation of the desired pose of the wheel. The inverse of the behavior's expected motion is applied, giving the pre-manipulation pose of the wheel.

For each element of the manipulation set, the pre-manipulation pose is calculated. Each of these is checked to ensure that valid inverse kinematics exist, it is collision free, that both the expected motion and all potential motions can be executed from this position, and any other constraints associated with the behavior are satisfied. Any that do not meet these criteria are eliminated. The redundant degree of freedom (shoulder yaw) is chosen to maximize the distance from joint stops. Remaining poses are ranked based on manipulability and score of the corresponding element of the manipulation set. The manipulability metric rewards large distances to joint stops at both the start and end of the expected motion, as well as small changes in angles during the expected motion, thus penalizing configuration changes during the expected motion. This ranked set is passed to the arm planner.

3.3.3 Arm planner

The output of the manipulation planner is a ranked set of hand poses (and corresponding joint angles) for all given behaviors, which the arm planner uses as goal states to generate a collision free trajectory. A best first policy is used and returns the first feasible trajectory.

We have transitioned to a Rapidly-exploring Random Tree (RRT) Connect planner for for the ARM-S Phase 2 work. In Phase 1, we planned using searches in a parametric representation of the velocity space of both the arm and neck angles, at the cost of additional computational complexity (Hudson et al. 2012; Howard et al. 2008). The new planner runs faster and more easily handles constrained motions by sampling in the joint space of the two arms, but post-process smoothing of the motion is required.

The single arm motion planner is a standard RRT planner in the 7-DOF joint space, based on Kuffner and LaValle (2000). Smooth paths are produced with a post-processing step, which performs a greedy search for collision free paths between nodes in the original path, and eliminates the unnecessary nodes. The robot and objects in the environment are represented as collections of primitive shapes, which are used for fast collision checking using standard techniques. Objects are grown a small fixed amount to account for sensing and execution uncertainty.

The RRT-based planner can only operate in collision-free conditions; however, when performing manipulation tasks, the robot's arms will many times appear to be or actually be in collision with objects in the environment. Consequently, a pre-processing step is performed to decollide the arm from the environment. The planner performs a random search of small perturbations to the arm with increasing amounts of perturbation until a collision-free position is achieved. If the motion required is too large, it fails. While not guaranteed to find the shortest path motion, or a motion in the proper direction to achieve the desired goal, we observed this fast and simple technique to be effective. This is likely because with small errors, the smallest perturbation motion is generally the correct one. If localization errors of the hand relative to the object are too large, a path that moves the arm through the object it is manipulating could result.

In order to efficiently plan kinematically constrained dual arm tasks (such as holding an object with both hands), a single RRT is used for one dominant arm, but admissible nodes are restricted to those feasible by both arms. For each new node selected, the hand pose of the dominant hand is used to compute the hand pose of the non-dominant hand through their initial rigid transform. The inverse kinematics for the non-dominant hand is then used to test the feasibility of the pose. While this approach could lead to infeasible plans since there is no guarantee that the entire motion is feasible (due to approaching joint limits, which would require a configuration change), we found that for motions in relatively open environments, the approach was effective.

When a straight line in joint space is not impeded by obstacles, generation and smoothing of paths each took approximately 5 ms when limited to one core of a 2.4 GHz processor. Circumnavigating a large obstacle in the middle of the workspace took on the order of 200 ms to generate the path, and 50 ms to smooth it. Because the cost of computing the IK and additional collision checking is relatively small, the time to plan a dual arm task is comparable to the single arm case.

3.4 Control

The control system can be decomposed into several complementary components. First *behaviors* are specified in a hierarchical set of task frame controllers. These task frame controllers produce a smooth end-effector desired motion, which is then inverted into smooth joint space desired trajectories. Inverse dynamics, which include grasped objects and desired control forces, are used to provide feed-forward torques to the joint level controllers, which are combined

with low-gain PD joint feedback around the desired position trajectories. System models and online state estimation are used in each control component.

3.4.1 Task frame behaviors

The capabilities of the robotic manipulator system are encoded as a set of *behaviors*. These behaviors are defined in task (e.g. end effector) frames, and are thus specified independently of arm configuration. Here we define a behavior as a hybrid automaton which is composed of a set of discrete states or *actions*. Each action is a guarded motion, composed of a set of parametrized *controllers*, which describe the dynamics of the system while the action is being executed, and a set of *end-conditions* which dictate when an action is complete or when an error has occurred.

For example, a *contact-grasp* behavior, used when grasping the tire, is composed of several discrete actions. First the arm tracks the free-space trajectory bringing the hand into proximity with the tire. All free-space motions end in a constraint offset from the object, so inadvertent contact is not made. During this motion a set of safety guards will abort the action with any excessive sensed force at the arm joints or at the wrist force sensor. Successful completion of this motion is defined as the task-frame moving to within a small specified distance of the planned constraint. After a successful initial motion, a second action which combines visual servoing and a motion toward the object is initiated. Given that the hand is now near the object, the visual tracking (end-effector error with respect to the object pose) is used as a control input. Contact with the object is achieved by specifying a task frame velocity input towards the object grasp point. After contact occurs, subsequent actions control the contact force, finger closure and finger strain control.

Efficient creation and specification of controllers is achieved through the decomposition of the system into a set of hierarchical control frames (Fig. 8). The composition of controllers in each control frame and merging controllers from different frames is done using the Generalized Compliant Motion (GCM) framework (Backes 1994).

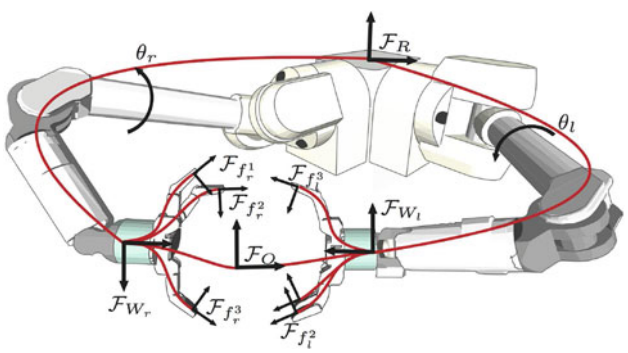


Fig. 8 Dual Arm Control Network: Controllers can be specified for each finger, object and wrist. These controllers are specified independently, with the wrist inheriting motion from connected controllers

The GCM framework is used to specify and merge control objectives. GCM is a task space controller which enables abstracted task definitions independent of system dynamics. In essence, GCM trades off control objectives by modeling each as a second order system. This creates a system of spring-like responses centered at each control set-point, which 'pull' on the end effector. All manipulation actions for the ARM project are comprised of only five control primitives: force control, visual servoing, Cartesian tracking, dithering, and inheritance. The output of GCM specifies the desired end-effector pose trajectory based on sensory feedback.

The following equations represent the GCM modification of the task frame motion of a single arm at the wrist. Here $G_*^*(k)$ is a homogeneous transform at discrete time-step k:

$$G_{RP}(k) = G_{RP}^{\text{drive}}(k) G_{Pm} G^{\text{inh}}(k) G_m(k) G_{Pm}^{-1} \tag{1}$$

$$G_m(k) = G_m(k-1) \Delta G_m(k) \tag{2}$$

$$\Delta G_m(k) = \Delta G_F(k) \Delta G_V(k) \Delta G_D(k) \Delta G_C(k), \tag{3}$$

where G_{RP} is the current transform from the robot $\mathcal{F}_R$ frame to the end-effector palm frame $\mathcal{F}_P$, $G_{RP}^{\text{drive}}(k)$ is the drive transform (or the open loop motion from any plan), G_{Pm} is a quasi-static transform from the palm to a merge frame $\mathcal{F}_m$, and $G_m(k)$ is the combined motion of any task frame feedback controllers in the merge frame. The merge frame can be any frame which is static (for the current behavior) with respect to the palm frame. Typically this is set to identity as the palm frame provides an intuitive place to consider motions of the arm.

$G^{\text{inh}}(k)$ is an inherited motion (or a projection of motion) from any child controller in the network. (i.e. a wrist controller may inherit motion from a finger or object controller). This is very useful for coordinating arm motion. For instance, in dual handed manipulation of the wheel, feedback controllers are specified in the object frame, forces are resolved in the object frame, and the produced motion is repeated by each arm through inheritance. Note however that the internal forces (i.e. in a move-squeeze decomposition) are accomplished by specifying motion in the wrist controller and not the object controller. Also in dual handed motion, desired control forces are computed through optimal load balancing (Sect. 3.6).

The total feedback driven motion in the task frame $G_m(k)$ is the integrated output of all task frame controllers. These controllers consist of force feedback, ΔG_F, visual servoing, ΔG_V (which closes the loop around the error from the kinematic wrist position to the wrist position seen the camera fame), dither motions, ΔG_D (used to inject 1–10 Hz motion into the end-effector, much like a human wiggling a key into a

lock), and Cartesian motion, $\Delta G_C(k)$ (which force the wrist towards a desired set-point or location on an object). The individual controllers, such as the force controllers, close the loop around respective error in a second order PD controller:

$$\Delta G_F(k) = K_p(e(k)) + K_d((e(k) - e(k-1)) \quad (4)$$

$$e(k) = F_m(k) - F_d(k), \quad (5)$$

where K_p and K_d are proportional and derivative gains, and in the case of a force controller, $F_m(k)$ is the measured force and $F_d(k)$ is the desired force.

Note that the visual servoing controller and Cartesian motion controller are significantly simplified from traditional methods due to persistent estimation (Sect. 3.2) and system modeling. As all errors are realized in $SO(3)$, tracking is now simply updating the desired position of the manipulator.

Action reuse is also maximized in this framework. There are no specialized behaviors for moving into contact for an object, or visual servoing with an object in hand. The same controller specification (and even the end conditions) are reused for freespace motions into contact, either with one hand independently or holding an object with one or two hands. The frame in which the controller is run will need to be specified (palm, wrist, object). Recorded sensor forces are projected into the correct frame using the system models and are used for feedback (5) and end condition monitoring.

3.4.2 Inverse kinematics and redundancy resolution

Arm redundancy during control is only considered in inverse kinematics calculations from the task frame to the joint space. In general, motions generated from feedback and tracking are feasible because they are checked in Manipulation Planning (Sect. 3.3.2). Each control behavior contains a history of expected resultant motion, as well as potential motions (due to object uncertainty). If kinematic infeasibility results during a behavior run, the task frame motion at which this infeasibility occurred is added to the potential motion set. Rerunning the Manipulation Planner would now account for this possibility of motion. Arm redundant degrees of freedom (θ_i, in Fig. 8 are searched over during each control step to minimize a cost function balancing total joint motion and distance from joint limits. Searches are limited to 100 evaluations of inverse kinematics at each step to maintain a fixed control loop execution time.

3.4.3 Joint space control

The task frame motion $x = G_{RP}(k)$ is converted to joint space motions $(\ddot{q}, \dot{q}, q)$ through inverse kinematics. A combination of low-gain joint level PD feedback and feed-forward inverse dynamics are used to produce joint torques:

$$\tau = M_{a,o}(q)\ddot{q} + C_{a,o}(q)\dot{q} + g_a(q) + g_o(q, O) + J^T F_d + K_d(\dot{q}_d - \dot{q}) + K_p(q_d - q) \ , \quad (6)$$

where $M_{a,o}$ and $C_{a,o}$ are the mass and gyroscopic terms from the world model which include both the arm a and grasped object o properties. The gravitational terms from the arm $g_a(q)$ is a learned torque map provided by Barrett Technologies, while object terms $g_o(q, O)$ are computed from the world model for grasped object 'O'. Desired task frame control forces F_d are also added to the feed-forward terms using the manipulator Jacobian (J). Use of low PD feedback gain terms (K_p, K_d) and inverse dynamics terms was adopted over the high gain PD feedback used in Phase 1 (Hudson et al. 2012) after demonstration by and discussion with the USC team (Pastor et al. 2011; Kalakrishnan et al. 2013).

Compensation of mass from objects such as the impact driver or wheel consist of a significant fraction of arm output torque, so accurate estimation of object location in the grasp, and mass properties is required for precision system tracking. Our approach uses less offline learning of arm properties than Kalakrishnan et al. (2013), but more online estimation of object parameters and location (Hebert 2013).

3.5 Task level planning

The dexterous manipulation tasks in DARPA ARM-S program require tasks to be executed in a particular order for successful completion. The task level planner maintains communication with other components of the system such as the estimator, planner and control.

The task level planner maintains the state of the physical world W^{est} as estimated by sensors, and a kinematic model of the world W^{model}. Both W^{est} and W^{model} are of type W, introduced in Sec. 3.2. W^{est} is kept updated using the estimator at all times. W^{model} is used to predict or simulate the changes in the world even before the robot starts execution.

The task planner additionally maintains a library of atomic tasks $T^{atom} = \{T_1^{atom}, T_2^{atom}, \dots\}$ which correspond to single behaviors that some component (including itself) of the system can carry out. Each component when asked to execute an atomic task (behavior), may in turn carry out a sequence of actions, which the task level planner does not track. The task level planner concerns itself with the overall success or failure of such tasks. Some examples of tasks are ESTIMATE_WORLD_STATE (estimation), GENERATE_ MANIPULATION_SET (planner), PLAN_ARM_TO_POSE (planner) and MOVE_USING_PLAN (control). These atomic tasks can take arguments α_i^{atom} such as the object on which the action must be performed, or the current world state. An atomic task with an instantiated argument can be thought of as a map

$T_i^{atom}\left(.|\alpha_i^{atom}\right) : W \rightarrow W.$

The application of these atomic tasks can be on W^{est} or W^{model}. In the former case, the tasks are carried out by the actuators and sensors, in the presence of noise and imperfections. However, in the latter case, application of a task implies simulating the expected change in the world state assuming perfect observation and control.

Atomic tasks that involve kinematic changes correspond to executing control behaviors (Sec. 3.4). When such tasks are simulated, W^{model} is propagated using the same *expected motion* that the controller uses to complete the corresponding behavior. Recall that the inverse of this *expected motion* is used by the manipulation planner to generate the starting pose in the first place. This makes the simulation consistent with idealized execution by the controller. The resulting W^{model} kinematically satisfies the *end-conditions* for the control behavior. Changes in grasp state $\mathcal{G}$ and linkage list L are also updated by appropriately translating the *end-conditions* of the control behavior. Thus, simulated grasping of an object correctly updates $\mathcal{G}$ with the object grasped, while a parts assembly behavior correctly updates L in W^{model}.

The library also contains some basic sequences (seq.) of atomic tasks, called compound tasks, $T^{comp} = \{T_1^{comp}, T_2^{comp}, \ldots\}$, where T_k^{comp} are finite sequences of elements of T^{atom}. The parameters for each atomic task are stacked or sequentially generated to make the argument for the sequence. A common compound task is given by Seq. 1.

Seq. 1 A common compound task

EXECUTE_BEH_SEQ(grasp_type, $O_{primary}$, $O_{secondary}$)

1: (Estimator) w = ESTIMATE_WORLD_STATE
2: (Planner) M = GENERATE_MANIPULATION_SET(w, grasp_type, $O_{primary}$, $O_{secondary}$)
3: (Task Planner) **if** $M = \emptyset$, **return** Failure, **exit**
4: **repeat**
5: (Task Planner) pick new $m_i \in M$
6: (Planner) p = PLAN_ARM_TO_POSE(w, m_i)
7: (Task Planner) if p is not valid, **goto step 11**
8: (Control) MOVE_ARM_USING_PLAN(p)
9: (Control) MOVE_FOR_BEHAVIOR(grasp_type)
10: **return** Success, **exit**
11: **until** M is exhaustively sampled.
12: **return** Failure, **exit**

In the ARM-S program, manipulation tasks are specified as an ordered list of T_j^{comp}. This list is internally translated to a directional graph where each node represents an atomic task. Note that there is no need to restrict ourselves to two levels of encoding task hierarchy. We can make compound tasks from other completely defined compound tasks as well.

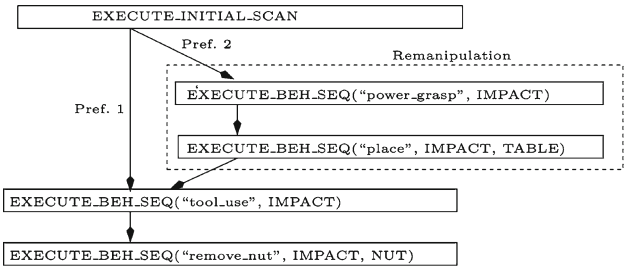


Fig. 9 Abstracted digraph for removing nuts with impact driver, Example 2. Two alternate routes exist with remanipulation preferred less

3.5.1 Remanipulation

On many occasions, due to kinematic constraints of the robot, there may be no feasible plans to accomplish a behavior. For example, for EXECUTE_BEH_SEQ("tool_use_grasp", IMPACT, Ø) the grasp planner may find no feasible solutions in the current pose of the impact driver (IMAPCT) on the table. To solve this we equip the task sequence to optionally execute a subsequence for remanipulating the impact driver on the table so that subsequently a tool use grasp is feasible.

Adding alternative optional subsequences involves allowing the nodes of the task sequence digraph to have multiple children and parent nodes and can be represented as in Fig. 9. Whenever there are multiple children to a node, then the child nodes are ordered by preference.

3.5.2 Kinematically dependent tasks

In many situations, some (compound) tasks in the sequence are kinematically dependent on each other. For the tool use example, this can arise in multiple ways.

Example 1 In this case, we are able to pick up the impact driver with a hand pose that is amenable to tool use (press trigger). However, after we have picked the impact driver off the table with a certain pose, the RRT planner may fail to find a path to orient the impact driver over the nut that we want to remove from a fixture. This happens usually when the initial plan for picking up the impact driver is near the physical joint limit of the wrist, even though other wrist solutions may exist.

Example 2 This example is in the need for re-manipulation of the impact driver in order that the next task (tool use grasp) is feasible.

Kinematically dependent tasks are usually linked by manipulation sets. They are therefore solved by linking them together inside nested iterative tasks and exhaustively searching until a feasible solution is found for all tasks in the linked subset. The task sequence for solving Example 1 is shown in Seq. 2.

Seq. 2 Task sequence for Example 1

EXECUTE_LINKED_SEQ("tool_use_grasp", IMPACT, Ø)

(Estimator) w_1 = ESTIMATE_WORLD_STATE
(Planner) M_1 = GENERATE_MANIPULATION_SET(w_1, "tool_use_grasp", IMPACT, Ø)
(Task Planner) **if** $M_1 = \emptyset$, **return** Failure, **exit**
repeat
  (Task Planner) pick new $m_{1,i} \in M_1$
  (Planner) p_1 = PLAN_ARM_TO_POSE(w_1, $m_{1,i}$)
  (Task Planner) if p_1 is not valid, **goto step 21**
  (Control) MOVE_ARM_USING_PLAN(p_1)
  (Control) MOVE_FOR_BEHAVIOR(tool_use_grasp)
  (Estimator) w_2 = ESTIMATE_WORLD_STATE
  (Planner) M_2 = GENERATE_MANIPULATION_SET(w_2, "remove_nut", IMPACT, NUT)
  (Task Planner) **if** $M_2 = \emptyset$, **goto step 21**
  repeat
    (Task Planner) pick new $m_{2,j} \in M_2$
    (Planner) p_2 = PLAN_ARM_TO_POSE(w_2, $m_{2,j}$)
    (Task Planner) if p_2 is not valid, **goto step 20**
    (Control) MOVE_ARM_USING_PLAN(p_2)
    (Control) MOVE_FOR_BEHAVIOR("remove_nut")
    return Success, **exit**
  until M_2 is exhaustively sampled.
until M_1 is exhaustively sampled.
return Failure, **exit**

Similarly, the sequence for Example 2, in which we must re-manipulate before executing the sequence from Example 1 consists of searching over four manipulation sets, corresponding respectively to:

1. EXECUTE_BEH_SEQ("power_grasp", IMPACT, Ø)
2. EXECUTE_BEH_SEQ("place", IMPACT, TABLE)
3. EXECUTE_BEH_SEQ("tool_use_grasp", IMPACT, Ø)
4. EXECUTE_BEH_SEQ("remove_nut", IMPACT, NUT)

Figure 9 showed an abstracted digraph of the task level plan for this example. Images of the robot resorting to remanipulation are shown in Fig. 10. All four compound tasks are kinematically linked.

3.5.3 Kinematic verification based execution

Once the task sequence directed graph is populated from the task specification, a kinematics-only verification of the entire sequence is carried out. First a sensor based estimate of the world is made and is used to initialize W^{model}. In our current implementation, we use depth-first traversal of the sequence. This preferentially chooses highly ranked child nodes first so that we are likely to find the most user preferred sequence for execution. As each node, n is encountered its effect is *simulated* on the output of the parent node corresponding to its initial condition, $W^{model}(parent(n))$. Planning tasks can produce additional quantities such as the manipulation set M or a plan p, which can be considered to augment the world state. Each node can flag *success* in which case a new world state, $W^{model}(n)$ is generated and search proceeds or *failure* in which case we backtrack until a parent with unexplored child or an unfinished iteration task is encountered.The verification ends either in success (a node with no children is reached with no failure) or fails if exhaustive search produces no successful path.

Once the task sequence execution starts, it is natural to observe discrepancy between W^{est} and W^{model} from the

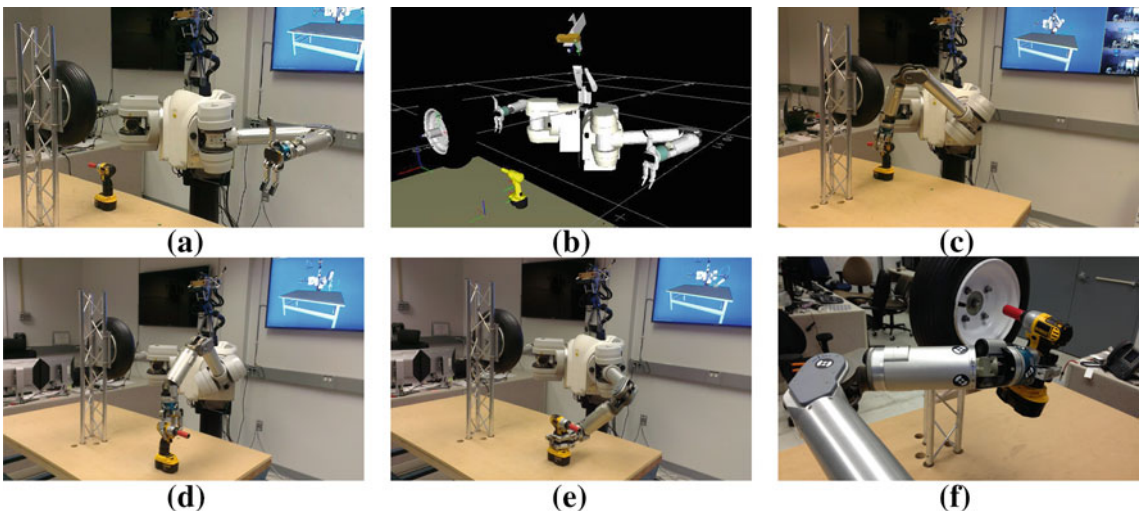


Fig. 10 Execution for Example 2, Fig. 9. **a** Initial world state. **b** Initial scan used to populate W^{model}. Kinematic verification reveals that the robot cannot grasp the impact driver for tool use in this configuration (the Pref. 1 path). **c** Robot picks up impact driver using power grasp (Pref. 2 path). **d** Impact driver is placed in a location which will allow feasible behavior for next step. **e** Robot is able to pick up impact driver with a tool use grasp. **f** Robot is able to plan to correct position for nut removal

kinematic verification at any node. In our current implementation, we assume that this discrepancy is not large enough to change the remanipulation decisions that have already been been made in the verification step. However, if kinematically linked tasks are encountered we recompute manipulation sets and the associated RRT plans, since they are more sensitive to the actual world state.

3.6 Modeling

The mechanical, geometric, kinematic, dynamics, sensor and actuator characteristics of the robot fundamentally effect the operation of the robot and its interactions with the task environment. Our ARM-S model-based autonomy modules use a foundational modeling layer to generate information and required data. While relying extensively on embedded models, we do not have the unrealistic expectation that the models be perfect, but instead that they help reduce the demands on the autonomy software. A-priori knowledge and online estimators are used to continuously update and improve the model data to reduce uncertainty and handle changes during task execution.

The ARM-S embedding modeling layer is an adaptation of the *RoboDarts* embedded modeling architecture (Jain 2013) for robotics applications. RoboDarts in turn is built upon the fast *DARTS* computational dynamics software (Dynamics and Real-Time Simulation (DARTS) Lab 2013) for articulated linkages based on the *spatial operator algebra (SOA)* mathematical framework and algorithms (Jain 2010). While a significant application of the DARTS software has been for the closed-loop simulator development for space and robotic platforms, RoboDarts is tailored to embedded use for the fast computation of the broad variety of model-based information needed by the autonomy modules. Furthermore, SOA's structure-based algorithms are able to adapt to handle run-time changes to the robot, its tasks and the environment.

A generic *frames layer* in RoboDarts supports arbitrary queries for relative poses between the links, sensors, task objects, feature frames, etc. in the system. Lazy evaluation and caching is used for efficiency to compute transforms only on demand, and to avoid recomputing unchanged values. The frames layer automatically handles system topology changes from run-time body attachments and detachments, and the addition and deletion of bodies. The frames layer also supports forward kinematics functions for the robot arms, neck, hands and the dependency of various poses on the joint coordinates. These layers are used by all modules for computing needed pose transforms as well as for driving the 3D visualization graphics. Additionally specialized inverse kinematics code for the WAM arms (generated using the OpenRave IKFast module Diankov and Kuffner (2008)) is used to compute the complete set of joint angle solutions for any end-effector pose. The inverse kinematics is used for motion planning, as well as for redundancy management during real-time control. Jacobians and manipulability measures are available for all linkages in the system. These are used for planning and motion control, as well as for projecting end-effector forces into joint torques.

A collision detection module (built upon the Bullet collision detection library Coumans et al. (2013)) supports the checking of collisions between bodies. This is used by the motion planner to generate collision free paths as well as by the grasp planner to generate grasp sets. This module allows one to select the coarseness of the collision shape geometries, to selectively hide objects, as well as to add padding to the shapes. The selective collision filtering feature allows users to disable collision checking between specific pairs of bodies (eg. connected bodies) as needed. Collision filters are automatically updated when bodies are attached and detached from each other during run-time (eg. impact driver and its battery). This also extends to the run-time grasping and ungrasping of objects by the hands. Such grasping is not limited to rigid body objects and can involve *articulated task object linkages* such as trimmers. Pose estimates generated by the estimator are used to continually update the attachment poses within the modeling layer.

When heavy objects such as the tire or the impact driver are grasped by a hand, appropriate gravity compensation torques need to be applied to avoid degrading end-effector positioning accuracy from arm sag. Such feed-forward compensating torques are computed by the modeling layer for the control module. Dual-arm manipulation introduces loop constraints within the system topology. While reducing the available number of motion degrees of freedom, these constraints lead to internal forces that build up within the robot. A move/squeeze decomposition approach (Jain 2010) is used to compute the feed-forward terms that optimally load balances the torques across the arms.

The modeling layer also has provisions for full dynamics based time simulation of selected, or all, the bodies within the system. The smooth dynamics solver uses the minimal coordinates, O(N), articulated body forward dynamics algorithms for solving the smooth dynamics, and a minimal coordinate complementarity based solver to handle non-smooth contact/collision and constrained dynamics (Jain et al. 2012). Such dynamics simulations are used by the task planner to plan execution sequences, and by the estimator to generate dynamics based predictions.

4 Test results

4.1 Wheel change

Testing results of the wheel change scenario include formal testing by the DARPA test team at a remote site, and internal JPL testing.

The complete wheel change scenario has not been completed yet, as the current Barrett™BH8-280 hand cannot kinematically pickup the lug nuts and re-thread them on the axle assembly. Full completion and testing of the tire change sequence will be enabled with the addition of new robot hands from the DARPA Autonomous Robotic Manipulation low cost Hand (ARM-H) track (Hackett et al. 2013). These new hands have been demonstrated to re-thread the lug nuts to the axle assembly bolts using tele-operation, but have not yet been fully incorporated into the software algorithms in this paper.

The removal of the wheel and all four nuts, including the impact tool grasp and it's use was demonstrated by the JPL team at a remote DARPA test site. The success rate for this task sequence was 3/5. To the best of our knowledge, this is the best that any team achieved during remote testing. We had two failures to remove a single nut, which was not understood by the autonomy system, causing sequence failure. Both nut-removal failures were caused by incorrect data association during object tracking; we utilize the distinctive red tip of the impact drill (Fig. 5 as discussed in Sect. 3.1.3). Classification of the red tip failed during testing due to shadows and lighting conditions at the remote site that differed from the development environment. This event occurred during the verification that the impact driver was on the nut, causing the autonomy system to move incorrectly to a recovery position. As a result, the impact driver unfortunately jammed on the nut and the subsequent motion caused a safety fault end-condition to trigger (Sect. 3.4.1). This required a human reset and the stoppage of the test. Further autonomy and robotic deliberation could in the future be used to rationalize this occurrence.

Table 2 Internal wheel change testing: times for sub-task completion (seconds)

Task	1	2	3	4	5	Avg.
Scan scene	45.3	44.3	46.8	41.3	46.8	44.9
Task plan	7.6	8.3	7.1	8.7	7.6	7.8
Re-grasp	0	27.2	0	25.2	0	26.2
Impact grasp	33.6	33.3	33.0	32.9	33.0	33.2
Remove nut1	40.3	52	51.8	72.9	69.1	57.2
Remove wheel	36.3	36.9	36.8	36.6	37.7	36.9
Re-scan	24.0	21.4	20.4	21.7	21.1	21.7
Attach wheel	68.0	70.6	71.6	70.3	70.9	70.3

To provide a more detailed breakdown of testing results, the same tests have been run at JPL, with the estimation and perception system being updated with more rigorous outlier rejection on data association results. The wheel change task was run 5 times. We have tabulated run times of each task component independently in Table 2. Two of the tests positioned the impact driver such that it required re-manipulation. The task planner, through verification of the task sequence, detected that re-grasping was required and updated the task sequence.

We have provided a video recorded at JPL showing the wheel change task as supplementary material available from Autonomous Robotics.

4.2 Wire cutting

Testing results of the wire-cutting scenario (illustrated by Fig. 11) presented here are only from internal JPL test-

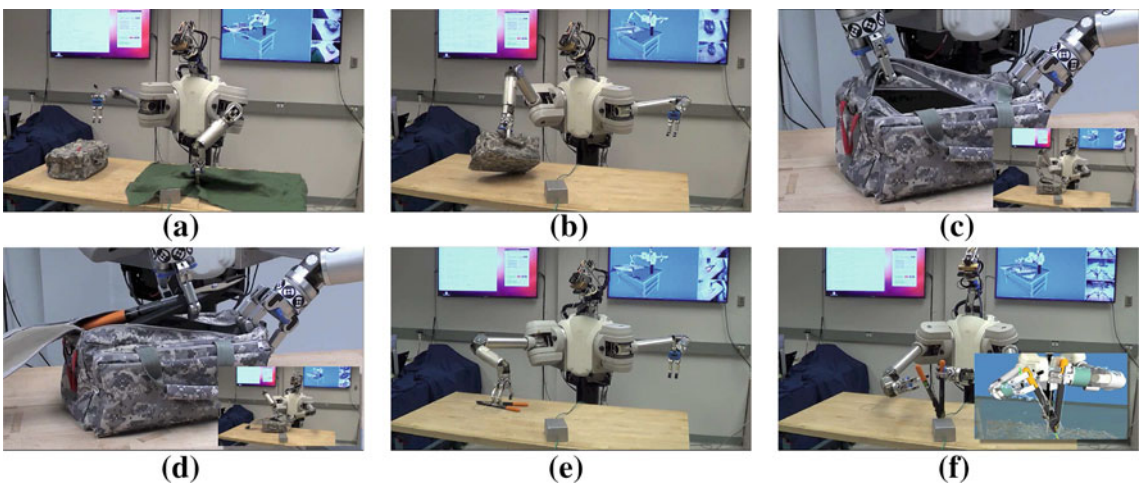


Fig. 11 Wire-Cutting Scenario overview. **a** The robot first removes the burlap sack to uncover a hidden junction box with a *green wire*. **b** The toolbag is then repositioned in front of the robot for further manipulation. **c** The robot unzips the bag and removes the lid/flap obstructing the opening of the bag. **d** A search behavior is commenced to remove the necessary cutting tool. **e** The bag is removed from the scene and the trimmers replaced on the table for a better re-grasp. **f** The trimmers are re-positioned with both hands above the wire for cutting

Table 3 Internal wire cut testing: times for sub-task completion (seconds)

Task	1	2	3	4	5	Avg.
Uncover	57.1	56.7	57.0	56.9	57.8	57.1
Move bag	48.1	50.4	47.3	47.4	47.5	48.1
Open bag	127.4	127.7	125.3	117.1	125.6	124.6
Search	87.7	75.9	164.8	85.0	90.5	100.8
Grasp	43.4	42.7	95.7	46.0	93.9	64.3
Extract	14.2	12.8	15.9	11.5	11.7	13.2
Position	47.3	35.4	109.4	51.5	46.7	58.1
Cut wire	61.2	59.2	62.3	55.1	64.2	60.4

ing. While the full scenario was tested with the DARPA test team, only certain subtasks of the scenario were completed successfully at the DARPA test site due to varying environmental conditions (lighting, calibrations, etc.) that identified over-tuning of parameters in our system and prevented a continuous end-to-end task completion, though each individual subtask was completed successfully on their system.

The complete end-to-end wire-cutting scenario has been completed successfully during internal JPL testing. This internal testing had improved trimmer detection/localization, manipulation planning and grasp verification. Table 3 shows the timing of each completed subtask during 5 trials of internal testing. Our testing showed that we could reliably complete the scenario up-to and including the grasp subtask of the trimmers. This was due to correct grasp and bag verification, allowing automatic retrying of the subtask. In the bag-search subtask the robot experienced 4 failures in the 5 trials and continued to re-attempt the task until the trial was a success. Similarly with the grasp subtask, 3 failures were encountered and the robot continued the subtask until success was achieved. The entire scenario, which included detecting and localizing (extracting) the object in the hand, positioning the trimmers, and lastly cutting of the wire was found to be less reliable. In our testing the robot managed to cut the wire 3/5 times during fully automated sequence runs. (One trial did not cleanly cut the wire, leaving some of the sheath connected, but was counted as success). The two failures resulted from two separate issues. The first failure occurred from the inability to verify a successful cut. The robot did manage to position the trimmers over the wire and attempt a cut but missed. The second failure initiated from a bad initial grasp of the trimmers, leading the motion planner incapable of finding feasible kinematic solutions to position the trimmers over the wire.

We have provided a video recorded at JPL demonstrating a successful end-to-end wire-cutting sequence as supplementary material available from Autonomous Robotics.

5 Discussion

The presented system utilizes a novel combination of ideas and has required the development and refinement of many system components. In creating the system, care was taken to generalize components, enable algorithmic reuse, and require as little tuning or special cases as possible. This system has been tested with many one-off tasks and multiple sequences of tasks, both on a local development robot, and on a distinct independently run DARPA system. However the system falls short of being truly deployable, or quickly adaptable to new task and sequences; the system requires an interface or learning algorithms to enable general users to specify or demonstrate new behaviors and sequences, and a more complete understanding of the discrete system state. The following discussion attempts to highlight aspects of the system which worked well, and those which would benefit from new approaches.

The estimation module (Sect. 3.2), which persistently updates manipulated objects, significantly increased the reuse of behaviors, and reduced the need to parametrize behaviors (Sect. 3.4.1). For instance a *move into contact* control behavior was completely generic for aligning and touching the wheel to the axle hub (dual handed), aligning the trimmers before cutting (dual handed), or aligning and contacting the impact driver and the nut on the wheel (single handed). The use of models, understanding the connections between the hands and the object, accurate estimation, and the ability to defined control frames in the object, made behavior definition very easy and consistent across widely varying objects. Note that there are still specialized behaviors in the system, for instance grasping the impact driver before tool use required a specific approach angle and extra fixturing steps to robustly align the finger and the trigger. These specific behaviors were only ever associated with specific manipulation tasks (unzipping, cutting, etc).

Grasping tasks (where the only requirement was a secure grasp) tended to be immediately achievable by the system for novel objects once grasp sets were generated from the geometric object (Sect. 3.3.1), and the object could be classified and detected by the perception system. Surprisingly the same table-grasp behavior (where the open hand was pressed against the object and or table, and the fingers scraped closed over the table surface) immediately worked for picking up the trimmers from within the bag, picking up the burlap sack, and picking up the tool bag itself. Other ARM teams, using a similar behavior, have experienced comparable results (Kazemi et al. 2012).

The ability to verify task completion (and detect or estimate the discrete state of the environment) was not rigorously implemented in the developed system. The continuous state estimator (object location, mass, etc. in Sect. 3.2), performed

well conditioned on the correct discrete state (*zipper grasped*, *bag open*, *tire attached*, *nut removed*), but the detection or classification of the discrete state was usually a specialized ad-hoc algorithm. This in a large part explains why the probability of executing an entire sequence of tasks was similar to the product of each sub-task's probability of success. For instance, we have a very high success rate of picking up the impact driver, and a very high rate of removing each nut, but overall the sequence did not perform as well as expected in independent testing. Most discrete state classifications were based on simple thresholds which were tuned at the development site, and occasionally failed at the DARPA test site. For instance, we used visual distance classifiers for the red impact driver tip with respect to the nut position to check for attachment of the impact driver and nut. A false negative from this classifier was enough to ruin an entire tire-change sequence. While some classifiers worked well (was a heavy object grasped), many of the most difficult manipulation tasks have similarly difficult verification problems (knowing that we have grasped the zipper, and not a fold of the bag, is as difficult as actually grasping the zipper). There certainly more sophisticated methods of learning to classify task outcomes (Pastor et al. 2011), but it is unclear how observable all of these criteria are in difficult manipulation tasks. To realistically approach human-like performance in manipulation, it seems likely that human-like performance in understanding the system state, or the task outcome is required.

The DARPA ARM program did not seek to emphasize the role of perception in manipulation systems. The developed system provided very high accuracy in both classification and localization of objects within the confines of the table top. However creating algorithms for object recognition was time consuming and often required special case detectors or classifiers in addition to the reusable ICP and contour-based algorithms. For instance, finding a thin, flexible green wire, and localizing it in 3D space, or detecting lug-nuts on a plywood surface, or localizing the zipper on the camouflage bag all required distinct algorithms, and often imposed constraints on where the objects were located to succeed. Generic scene understanding, object classification and localization is an open topic of research. While imprecise visual object localization could be updated through kinesthetic feedback mechanisms, and not affect task outcome, no task succeeded with incorrect object classification.

A significant difficultly, and in general the most time consuming aspect of developing sequences to achieve a complicated task (such as changing a tire), was understanding the intersection of kinematic constraints from each subtask. The task planner (Sect. 3.5), was developed to select valid elements from a manipulation set (Sect. 3.3.1) enabling the entire sequence, but often the manipulation sets (or correspondingly the admissible poses before a behavior is executing) were too limited to achieve the whole sequence without significant constraints in the location of objects in the environment. For instance, the position of the tire in the workspace of the robot was critical to task completion, and was required to be between 40 and 50 cm away from the robot. If the tire was too far away, dual handed manipulation would fail, and it was hard to see the impact driver tip; if the tire was too close, there was inadequate room to maneuver the impact driver. The choice of how to pick up the impact driver was reduced to a picking up the driver sideways as the distance of the wrist to palm in the Barrett system is 15cm, causing severe workspace issues. This problem would likely be ameliorated with a mobile base, but was a challenge in this project. However, it is important to note that often in development we apparently had sufficiently expressive manipulation sets so that each subtask would on it's own succeed, but once integrated into the whole sequence we found the set to be insufficient.

The importance of testing with independent evaluators and different copies of the same system or locations, is often neglected in manipulation research (if only for cost reasons). The ARM program did a rigorous job of enforcing this testing on participants. A significant result of this is that all teams, despite their best efforts to create a general system, have anecdotally reported lower testing performance on the alternate systems. [1] While it is perhaps expected that developing on one system and testing on another may result in a performance drop, it is unclear if this could be corrected by developing on multiple systems, or if there is requirement to modify the developed approach.

6 Conclusions

The paper puts forth a complete model-based system that is able to autonomously complete human-level manipulation tasks through decomposition and deliberation over subtasks. Through novel model-based perception and estimation, efficient planning and reactive control, the system can perform robustly in semi-structured and uncertain environments. Unlike most other systems, remote independent testing of the software was performed to ensure robustness and test for generality.

Future work will incorporate new robotic hands, developed under the DARPA ARM program which are more dexterous than the existing hardware.

Further research into verification of task completion and understanding the discrete system state is required to improve system reliability and adaptability to new scenarios. Future work should also include the ability to synthesize

[1] Personal communication at PI meetings.

sequences of behaviors and or learn from demonstration of sequences.

Acknowledgments The research described in this publication was carried out at the Jet Propulsion Laboratory, California Institute of Technology, with funding from the DARPA Autonomous Robotic Manipulation Software Track (ARM-S) program through an agreement with NASA.

References

Backes, P. G. (1994). Dual-arm supervisory and shared control task description and execution. *Robotics and Autonomous Systems*, *12*, 29–54.

Bagnell, J. A. D., Cavalcanti, F., Cui, L., Galluzzo, T., Hebert, M., Kazemi, M., Klingensmith, M., Libby, J., Liu, T.Y., Pollard, N., Pivtoraiko, M., Valois, J. S., & Zhu, R. (2012). An integrated system for autonomous robotics manipulation. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2955–2962).

Bajracharya, M., Ma, J., Howard, A., & Matthies, L. (2013). Real-time 3d stereo mapping in complex dynamic environments. In: *IEEE International Conference on Robotics and Automation Workshop: Semantic Perception Mapping and Exploration*.

Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mosenlechner, L., Pangercic, D., Ruhr, T., & Tenorth, M. (2011). Robotic roommates making pancakes. In: *IEEE-RAS International Conference on Humanoid Robots* (pp. 529–536).

Bohren, J., Rusu, R., Jones, E., Marder-Eppstein, E., Pantofaru, C., Wise, M., Mosenlechner, L., Meeussen, W., & Holzer, S. (2011). Towards autonomous robotic butlers: Lessons learned with the pr2. In: *IEEE International Conference on Robotics and Automation* (pp. 5568–5575).

Chitta, S., Jones, E., Ciocarlie, M., & Hsiao, K. (2012). Mobile manipulation in unstructured environments: Perception, planning, and execution. *IEEE Robotics Automation Magazine*, *19*(2), 58–71.

Coumans, E., et al. (2013). Bullet physics library. http://bulletphysics.org.

Diankov, R., & Kuffner, J. (2008). *OpenRAVE: a planning architecture for autonomous robotics* (Technical Report CMU-RI-TR-08-34). Pittsburgh, PA: Robotics Institute. Accessed 13 Nov 2013.

Dynamics and Real-Time Simulation (DARTS) Lab (2013). http://dartslab.jpl.nasa.gov. Accessed 13 Nov 2013.

Gallagher, G., Srinivasa, S. S., Bagnell, J. A., & Ferguson, D. (2009). GATMO: A generalized approach to tracking movable objects. In: *IEEE International Conference on Robotics and Automation*.

Hackett, D., Pippine, J., Watson, A., Sullivan, C., & Pratt, G. (2013). An overview of the DARPA autonomous robotic manipulation (ARM) program. *Journal of the Robotics Society of Japan* (to appear).

Hebert, P. (2013). *Estimation and inference for grasping and manipulation tasks using vision and kinesthetic sensors*. Ph.D. thesis, California Institute of Technology.

Hebert, P., Hudson, N., Ma, J., & Burdick, J. (2011). Fusion of stereo vision, force-torque, and joint sensors for estimation of in-hand object location. In: *IEEE International Conference on Robotics and Automation*.

Hebert, P., Hudson, N., Ma, J., & Burdick, J. (2013). Dual arm estimation for coordinated bi-manual manipulation. In: *IEEE International Conference on Robotics and Automation*.

Hebert, P., Hudson, N., Ma, J., Howard, T., & Burdick, J. (2013). Action inference: The next best touch for model-based object localization, parametrization, and identification. In: *IEEE International Conference on Robotics and Automation*.

Hebert, P., Hudson, N., Ma, J., Howard, T., Fuchs, T., & Burdick, J. (2012). Combined shape, appearance and silhouette for object manipulation. In: *IEEE International Conference on Robotics and Automation*.

Howard, T., Green, C., Kelly, A., & Ferguson, D. (2008). State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. *Journal of Field Robotics*, *25*(6–7), 325–345.

Hudson, N., Howard, T., Ma, J., Jain, A., Bajracharya, M., Kuo, C., et al. (2012). End-to-end dexterous manipulation with deliberate interactive estimation. In: *IEEE International Conference on Robotics and Automation*.

Jain, A. (2010). *Robot and multibody dynamics: Analysis and algorithms*. New York: Springer.

Jain, A. (2013). Structure based modeling and computational architecture for robotic systems. In: *IEEE International Conference on Robotics and Automation*.

Jain, A., Crean, C., Kuo, C., von Bremen, H., & Myint, S. (2012). Minimal coordinate formulation of contact dynamics in operational space. In: *Robotics Science and Systems, Sydney, Australia*.

Jain, A., & Kemp, C. C. (2010). EL-E: An assistive mobile manipulator that autonomously fetches objects from flat surfaces. *Autonomous Robots*, *28*, 45–64.

Kalakrishnan, M., Pastor, P., Righetti, L., & Schaal, S. (2013). Learning objective functions for manipulation. In: *IEEE International Conference on Robotics and Automation*.

Kazemi, M., Valois, J. S., Bagnell, J. A., & Pollard, N. (2012). Robust object grasping using force compliant motion primitives. In: *Proceedings of Robotics: Science and Systems, Sydney, Australia*.

Knepper, R., Srinivasa, S., & Mason, M. (2010). Hierarchical planning architectures for mobile manipulation tasks in indoor environments. In: *IEEE International Conference on Robotics and Automation* (pp. 1985–1990).

Kuffner, J. J., & LaValle, S. (2000). Rrt-connect: An efficient approach to single-query path planning. In: *IEEE International Conference on Robotics and Automation* (Vol. 2, pp. 995–1001).

Ma, J., Susca, S., Bajracharya, M., Matthies, L., Malchano, M., & Wooden, D. (2012). Robust multi-sensor, day/night 6-dof pose estimation for a dynamic legged vehicle in gps-denied environments. In: *IEEE International Conference on Robotics and Automation*.

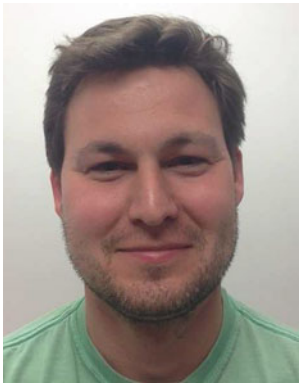
Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., & Abbeel, P. (2010). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In: *IEEE International Conference on Robotics and Automation* (pp. 2308–2315).

Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E., & Schaal, S. (2011). skill learning and task outcome prediction for manipulation. In: *IEEE International Conference on Robotics and Automation*.

Quigley, M., Berger, E., & Ng, A. (2007). Stair: Hardware and software architecture. In: *AAAI Robotics Workshop*.

Srinivasa, S., Ferguson, D., Helfrich, C., Berenson, D., Romea, A. C., Diankov, R., et al. (2010). HERB: A home exploring robotic butler. *Autonomous Robots*, *28*, 520.

Xue, Z., Ruehl, S. W., Hermann, A., Kerscher, T., & Dillmann, R. (2012). Autonomous grasp and manipulation planning using a tof camera. *Robotics and Autonomous Systems*, *60*(3), 387–395.

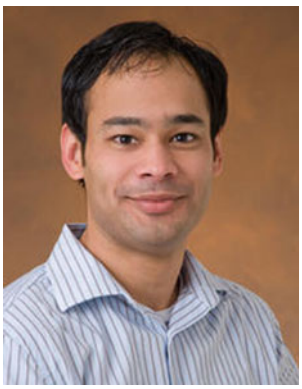


Nicolas Hudson PhD is currently a member of technical staff in the Mobility and Manipulation Group at the Jet Propulsion Laboratory in Pasadena, California. He is currently the task manager for the DARPA ARM-S task.

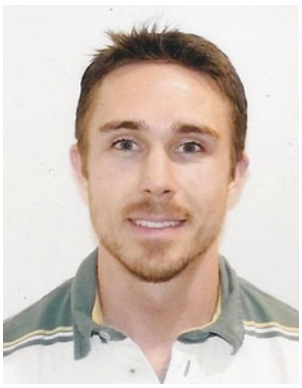
Jeremy Ma PhD is currently a member of technical staff in the Computer Vision Group at the Jet Propulsion Laboratory. He leads the perception team effort for the DARPA ARM-S task.

Paul Hebert PhD is currently a member of technical staff in the Manipulation and Sampling Group at the Jet Propulsion Laboratory in Pasadena, California. He is the estimation lead of the DARPA ARM-S task.

Abhinandan Jain PhD is a Senior Research Scientist at the Jet Propulsion Laboratory. His reasearch interests are in the areas of robot and multibody dynamics, control applications, and the development of computational modeling algorithms and architectures.



Max Bajracharya is a senior member of the Computer Vision Group at the Jet Propulsion Laboratory, Pasadena, CA. His research focuses on applications of computer vision to autonomous mobile robots in unstructured environments. His current work includes vision-guided manipulation, learning terrain classification, autonomous instrument/tool placement, coordinated arm/vehicle tool control, and person detection. Max received his Bachelors and Masters degrees in computer science and electrical engineering from MIT in 2001.



Thomas Allen received his BS degree in mechanical engineering from the University of California, Berkeley in 2005. After three years in the aerospace industry he is now working towards his PhD in mechanical engineering at the California Institute of Technology, advised by Joel Burdick. His research focuses on robotic grasping and caging.

Rangoli Sharan received her BTech degree in Electrical Engineering from the Indian Institute of Technology in 2007. She is currently working towards a PhD in Control and Dynamical Systems at California Institute of Technology in Pasadena, CA. Her research interests include robot task planning, partially observable discrete systems and formal verification techniques in control.

Matanya Horowitz is a doctoral student in the Control and Dynamical Systems department at Caltech. Before coming to Caltech, Matanya obtained four technical undergraduate degrees and an MS from the University of Colorado at Boulder. His research spans topics in computer vision, control theory, and robotic manipulation.

Calvin Kuo is currently a graduate student at Stanford University. Formerly a software engineer in the Robotic Simulation and Modelling Group at the Jet Propulsion Laboratory providing modelling assistance for the DARPA ARM-S task.

Thomas Howard PhD currently a Post-doctoral Fellow in the Roboust Robotics Group at the Massachusetts Institute of Technology. He was previously a Research Technologist with the Robotics Software Systems Group at the JPL. Dr. Howard earned a PhD in Robotics from Carnegie Mellon University in 2009.

Larry Matthies is a Senior Research Scientist at JPL and is the Supervisor of the Computer Vision Group (3474) in the Mobility and Robotic Systems Section. His also an Adjunct Professor in Computer Science at the University of Southern California and is a member of the editorial boards for the Autonomous Robots journal and the Journal of Field Robotics.

Paul Backes PhD is the Group Supervisor of the Robotic Manipulation and Sampling group at Jet Propulsion Laboratory, California Institute of Technology, where he has been since 1987. He received the BSME degree from U.C. Berkeley in 1982, MSME degree from Purdue University in 1984, and PhD in Mechanical Engineering from Purdue University in 1987. Dr. Backes received the 1993 NASA Exceptional Engineering Achievement Medal for his contributions to space telerobotics, 1998 JPL Award for Excellence, 1998 NASA Software of the Year Award Sole Runner-up, 2004 NASA Software of the Year Award, and 2008 IEEE Robotics and Automation Award. He has served as an Associate Editor of the IEEE Robotics and Automation Society Magazine. He is the Principal Investigator for the DARPA ARM-S task.

Joel Burdick PhD is currently the Richard L. and Dorothy M. Hayman Professor of Mechanical Engineering and Bioengineering at the California Institute of Technology. He leads the Caltech team in the DARPA ARM-S effort.