Mobile Manipulation and Mobility as Manipulation – Design and Algorithms of RoboSimian

Paul Hebert, Max Bajracharya, Jeremy Ma, Nicolas Hudson, Alper Aydemir, Jason Reid, Charles Bergh, James Borders, Matthew Frost, Michael Hagman, John Leichty, Paul Backes, Brett Kennedy
Jet Propulsion Laboratory
4800 Oak Grove Dr. Pasadena, CA 91109
Paul.Hebert@jpl.nasa.gov

Paul Karplus
Stanford University
Palo Alto, CA

Katie Byl, Brian Satzinger
University of California, Santa Barbara
Santa Barbara, CA 93106

Krishna Shankar, Joel Burdick
1200 E California Blvd
Pasadena, CA 91125

Abstract

This article presents the hardware design and software algorithms of RoboSimian, a statically stable quadrupedal robot capable of both dexterous manipulation and versatile mobility in difficult terrain. The robot has generalized limbs and hands capable of mobility and manipulation, along with almost fully hemispherical 3D sensing with passive stereo cameras. The system is semi-autonomous, enabling low-bandwidth, high latency control operated from a standard laptop. Because limbs are used for mobility and manipulation, a single unified mobile manipulation planner is used to generate autonomous behaviors, including walking, sitting, climbing, grasping, and manipulating. The remote operator interface is optimized to designate, parameterize, sequence, and preview behaviors, which are then executed by the robot. RoboSimian placed fifth in the DARPA Robotics Challenge (DRC) Trials, demonstrating its ability to perform disaster recovery tasks in degraded human environments.

1 Introduction

One goal of the DARPA Robotics Challenge (DRC) was to develop robots that can operate in a degraded human environment and perform tasks normally performed by a human. While most teams developed robots with a bipedal human form-factor, JPL's RoboSimian robot was designed with the philosophy that a passively stable, deliberate robot would be safer, more robust, and sufficiently fast for performing disaster response tasks. RoboSimian has four general purpose limbs and hands, capable of both mobility and manipulation, two active wheels on its body, and two passive caster wheels on its limbs, all of which are used to achieve passively stable postures, while remaining highly mobile and dexterous. The robot uses multiple stereo cameras to achieve nearly full 360 degree passive, low-power 3D sensing. It has four identical limbs, with each joint within the limb using an identical electric rotary drivetrain, minimizing cost and complexity, and simplifying maintenance. The under-actuated hands are significantly less complex than a human hand with fewer digits and active degrees of freedom, but can accomplish the set of grasps needed for effective operation in disaster scenarios. They are robust to damage and also serve as the robot's feet. An operator controls the robot by issuing high level commands based on images and 3D data sent back by the robot. The operator uses standard input devices (mouse and keyboard) to designate parameterized behaviors. Given a desired

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **MAY 2014**	2. REPORT TYPE	3. DATES COVERED **00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE **Mobile Manipulation and Mobility as Manipulation - Design and Algorithms of RoboSimian**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California, Santa Barbara,Department of Electrical and Computer Engineering,Santa Barbara,CA,93106**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Submitted to Journal of Field Robotics (JFR), Special Issue on the DRC, 2014.

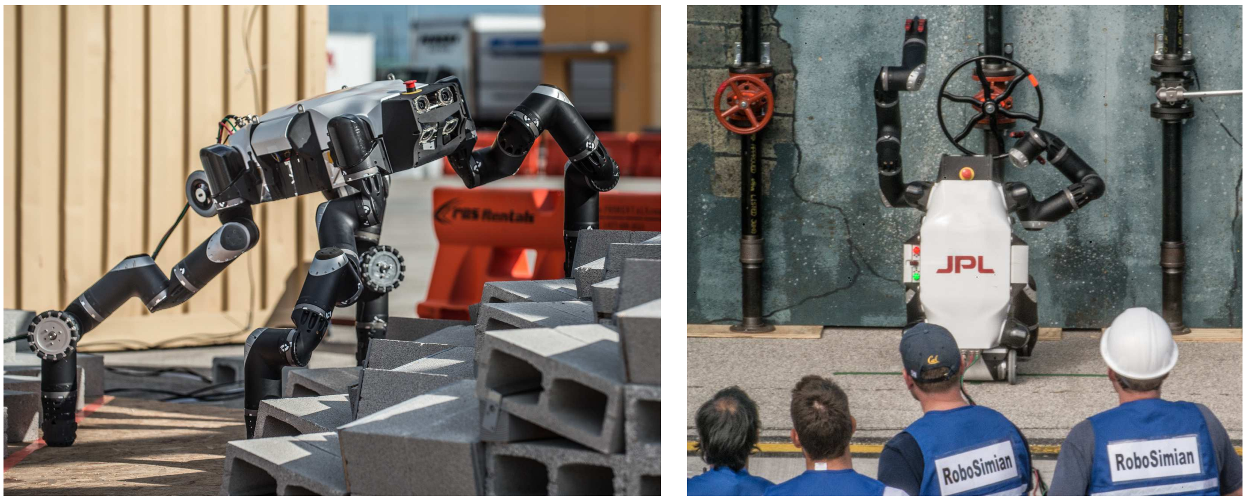
14. ABSTRACT
This article presents the hardware design and software algorithms of RoboSimian, a statically stable quadrupedal robot capable of both dexterous manipulation and versatile mobility in difficult terrain. The robot has generalized limbs and hands capable of mobility and manip- ulation, along with almost fully hemispherical 3D sensing with passive stereo cameras. The system is semi-autonomous, enabling low-bandwidth, high latency control operated from a standard laptop. Because limbs are used for mobility and manipulation, a single unified mobile manipulation planner is used to generate autonomous behaviors, including walking sitting, climbing, grasping, and manipulating. The remote operator interface is optimized to designate, parameterize, sequence, and preview behaviors, which are then executed by the robot. RoboSimian placed fifth in the DARPA Robotics Challenge (DRC) Trials, demon- strating its ability to perform disaster recovery tasks in degraded human environments.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**25**	

behavior, the robot plans safe motions, and after the operator reviews and approves the plan, executes the behavior.

Enabling practical, safe, and robust highly capable mobility and dexterous manipulation with low-bandwidth operator interactions required several innovations. RoboSimian's unique common joint, common limb design results in an extremely versatile but high performance mechanism. It merges 360 degree passive sensing to maintain its pose and create 3D maps and situational awareness, even when some cameras are occluded. It uses a single, unified mobile manipulation planner for mobility and manipulation, enabling it to walk, climb, sit, and manipulate. To enable an operator to control the robot with only high level commands, a framework for developing, paramaterizing, and executing behavior sequences was developed. Finally, a remote operator interface designed around efficiently commanding these behaviors based on objects detected by the robot was created.



(a) RoboSimian traversing the Terrain task.

(b) RoboSimian turning the second valve at the Valve task.

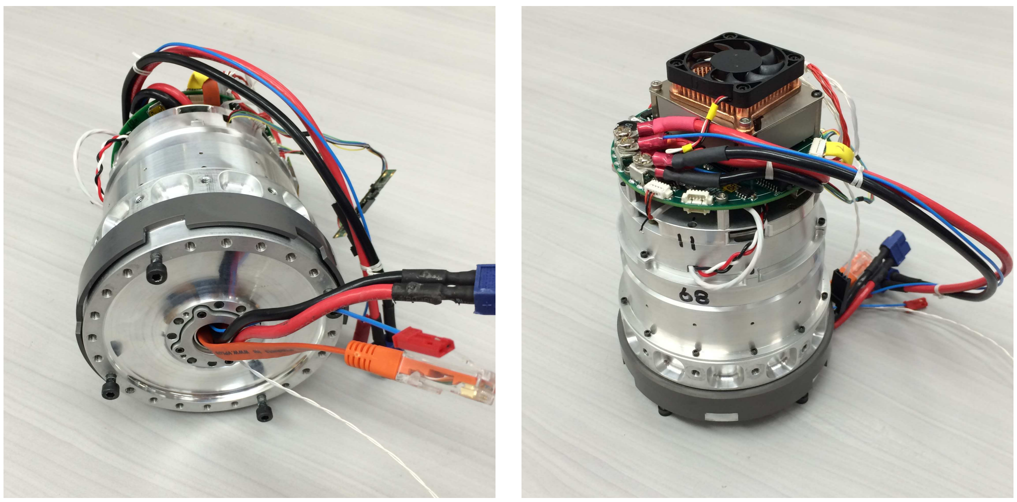
Figure 1: RoboSimian competing at the DARPA Robotics Challenge (DRC) in the Terrain task (a) and the Valve task (b).

This paper describes RoboSimian's first robotic platform (named Clyde), its hardware design, and its software algorithms. RoboSimian was designed, built and programmed by engineers and researchers at the Jet Propulsion Laboratory, the University of California at Santa Barbara, Stanford University, and the California Institute of Technology. Section 2 describes the design and build of RoboSimian, including the joints, limbs, body, and hand, along with the system architecture. Section 3 describes the software architecture and algorithms. Section 3.8 describes ongoing research, and Section 4 describes and discusses RoboSimian's performance at the DRC trials.

2 Hardware Design

In order to conserve mass, volume, complexity, and operational efficiency while retaining broad operational flexibility, RoboSimian has generalized limbs capable of both mobility and manipulation arranged in an axi-symmetric fashion. In order to achieve the stability and robustness that drives the RoboSimian design, the robot must be capable of at least three points of contact while still being able to manipulate an object, thus requiring a minimum of four limbs. Traditional anthropomorphic robotic design dictates that robotic joints be larger in the proximal location and decrease in size and torque capacity towards the distal end. This is reasonable because moment loads generated by an outstretched limb or iron cross are greatest at the proximal joints and decrease towards the distal end. Once a robot grasps a rigid object (such as a ladder or vehicle roll cage) with multiple limbs, the forces imparted to the robots limbs become symmetric. Loads at the distal joints and proximal joints can be equally large and damaging. Thus RoboSimian uses the same lightweight, high-torque actuator design for each of its 28 joints. There are several advantages to this design: 1) engineering resources can be focused on designing a single high performance actuator; 2) economies of scale resulting from larger quantities of machined parts and commercial off-the-shelf (COTS) components lead to lower production costs as items are ordered in bulk quantities; 3) Field support becomes easier as there is only a single actuator unit to swap out. Repairs are also simplified as spare components only need to be stocked for one design.

2.1 Actuator Design



(a) RoboSimian actuator showing the output side with power and communications cables running through the actuator for 'daisy-chaining'.

(b) RoboSimian actuator showing the custom motherboard hosting the Elmo Whistle Gold and microcontroller.

Figure 2: RoboSimian actuators illustrating the motherboard, Elmo Whistle Gold, the microcontroller and the output side of the actuator with power and communication cables.

The RoboSimian actuator consists of COTS drivetrain components mounted and housed by custom machined aluminum parts. A frameless DC brushless motor directly drives a 160:1 harmonic drive, with the output supported by a crossed-roller bearing. A power-on-to-disengage magnetic safety brake is included on the motor rotor and is able to hold a torque at the output side, shown in Figure 2a with all power removed. This is critical to the operations strategy of RoboSimian, where the robot may have to hold limb poses for long periods of time. Two position sensors are included: an optical incremental encoder on the motor rotor and a capacitive absolute position sensor on the actuator output.

Actuator electronics are mounted on the back side of the actuator and together with the mechanical components form the actuator assembly (Figure 2b), the building block of the RoboSimian limb. The electronics consist of a custom motherboard which hosts an Elmo Whistle Gold servo drive for the drive electronics and a microcontroller to handle the brakes and actuator health monitoring. The servo drive and microcontroller communicate with the upstream electronics on EtherCAT and RS-485 networks respectively. The motherboard also has connectors to allow daisy-chaining of the power and communications harnessing.

2.2 Limb Design

Each RoboSimian limb has 7 degrees of freedom and can be broken down into simple subcomponents. A limb consists of 3 elbow assemblies and an azimuth actuator connected to the body. An elbow contains

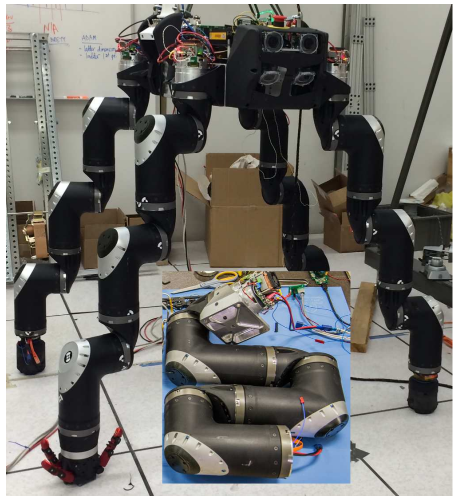


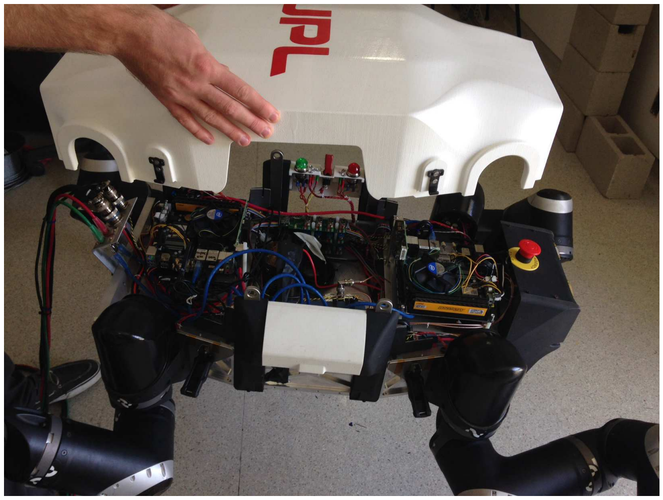
Figure 3: RoboSimian's limbs extended with inset showing a compact folded limb.

two actuators paired orthogonally with connecting structure. Elbows are linked together with additional structure. Non-structural elbow caps cover the actuator electronics and allow easy access to the motor controllers. The caps have molded rubber inserts (elbow pads) to protect the structure from scrapes and scratches. Harnessing is run through the centerbore of the each actuator and down the limb, breaking off at each distributed motor controller. The limb design terminates in a 6-axis force/torque sensor which serves as an interface to end-effectors. The mechanical configuration of the limb has the actuators offset in a manner that allows greater than 360° rotation, limited only by harness twist. This affords the limb greater flexibility and increases the solution set for a given motion. As pictured in Figure 3, the offset actuators also allow for compact limb storage.

2.3 Body Design

Much like the other RoboSimian hardware, the body design was centered around symmetry and simplicity while leaving ample room for ease of access to necessary internal components. Following this approach, mass and volumetric savings were achieved by using a monolithic aluminum structure. This structure was able to serve as the component deck to which all internal components and limbs could be mounted with no additional structural elements.

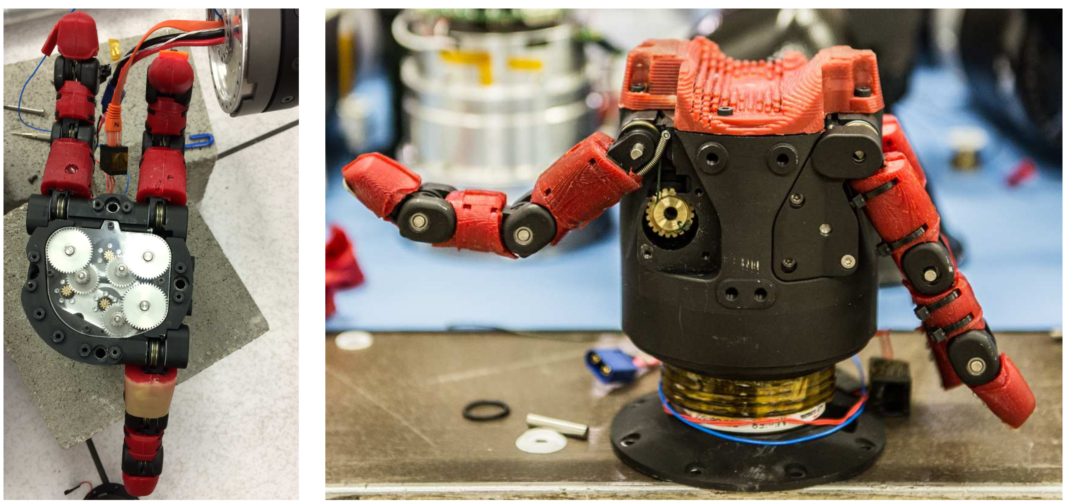
Following the spirit of the DRC challenge and urban disaster relief, the body shape and size was designed to fit into and through confined spaces. This drove the body aspect ratio to be slightly rectangular in order to fit the required internal components. At the center of the chassis, a custom battery was designed to keep mass low, centralized, and to fit within the body shell. This battery is swappable and can be easily accessed by releasing four over-center latches which hold the upper body shell on. Upon removing the upper body shell, virtually all internal components can be accessed, diagnosed or swapped with the removal of a few small fasteners at most. The body shell itself was rapid prototyped for cost savings, then coated with high strength enamel to give it structural rigidity and tolerance to high impact events without fracturing. Figure 4 shows the removal of the body shell and quick access to internal components. In addition to the body shell, a few smaller carbon filled rapid prototyped pieces were added to give protection to the array of cameras which are mounted just outside of the aluminum chassis. Note that the RoboSimian chassis and internal components can all be assembled and tested on a table top. Limbs, cameras, and any other external components can also be easily accessed or swapped with the removal of the body shell. This design allows RoboSimian to be a highly versatile multiple sub-system robot which can be quickly changed, re-configured or added to at any time.



(a)

Figure 4: RoboSimian's body shell removed showing the quick latches and easy access to internal components.

2.4 Hand Design



(a) RoboSimian hand with drive gears exposed. Note the staggered opposing fingers.

(b) RoboSimian hand with tendon spool exposed. Note the shape of the palm to maximize contact with a tool such as the drill.

Figure 5: RoboSimian hand with tendon spool exposed and the drive gear system exposed.

Similar to its namesakes, RoboSimian uses all of its extremities for both mobility and manipulation. The RoboSimian hand was designed to be walked on in addition to being the main interface to the DRC tasks. The body of the hand is a monolithic aluminum part that houses the mechanisms and drive electronics for the fingers, as well as providing structural rigidity for the full weight of the robot while walking. The palm is constructed from a high-durometer molded polyurethane that is both robust for walking and conformable and tacky for manipulation. The palm also accommodates a USB web camera to assist with teleoperation, but this feature was left out for the competition hands.

The hand has three under-actuated fingers, each with a braided Dyneema® tendon wrapped around pulleys at each of the three joints. The tendons are wrapped on a cable spool (Figure 5b), which is driven by a DC brushed motor through multiple gear reductions (Figure 5a). The closing order of the fingers is tuned via pulley diameter and return springs, the latter of which serve double-duty to pull the fingers back along the body of the hand for walking on the palm. This hyperextension toward the wrist protects the fingers and exposes the palm for making solid contact with the ground, similar to the hoof of an animal. The fixed layout of the fingers (two opposed with the third to the side) allows for wrap grasps with holding forces in excess of 500N, pinch grasps with forces in excess of 10N, and trigger grasps capable of power tool actuation. The tendon actuators are non-backdriveable such that all grips can be maintained with no power.

The electronics for the fingers reside on a single custom PCB which hosts a DC/DC power converter, three H-bridge drivers, and a single microcontroller with custom motor control software. Analog to digital converters are also included for position sensors on each of the finger joints and tension sensors for each of the tendons, but due to schedule and integration constraints, these sensors were not included in the competition hands.

A set of stumps with the same form as the hands but without fingers was also fabricated. For the competition, which allowed changing end-effectors between events, RoboSimian switched between hands and stumps, with the set not in use stored inside the body shell.

2.5 System Design

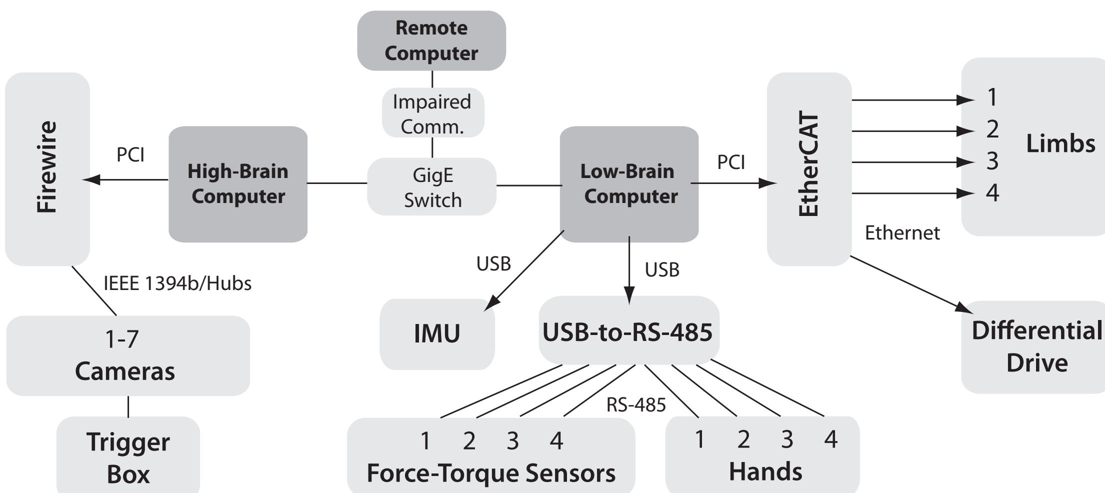


Figure 6: Communication pathways to different hardware interfaces on RoboSimian.

The communication pathways in RoboSimian are described and laid out in Figure 6. The low-brain and high-brain computers are networked via a GigE Ethernet switch located in RoboSimian. The remote computer (i.e. the operator machine) is on the same network but connected to a low-bandwidth/high-latency filter (provided by DARPA to simulate impaired communications) external to the robot which is then connected to the internal robot switch.

The low-brain connects and communicates to each limb through an Ethernet physical layer link. Control and communication to each limb as well as the differential drive is achieved through the EtherCAT® protocol. Using a tree topology of the EtherCAT® network, each limb is a segment of the network and each joint in the limb is a slave along the network segment. The low-brain also communicates to four ATI Industrial Automation force-torque sensors that are placed at the end of each limb just before each hand. These sensors communicate to the low-brain via RS-485 as the physical layer of a half-duplex serial protocol using a twisted pair down each limb. The hand controllers also communicate via RS-485 to the low-brain. These sensors communicate to the low-brain via RS-485 as the physical layer of the Modbus protocol using a twisted pair down each limb. The Internal Measurement Unit (IMU), a VectorNav VN-200 Rugged inertial navigation system (INS), is also connected via USB to the low-brain.

The high-brain computer's main connectivity is with the cameras. The high-brain uses a PCI express splitter which connects multiple Point Grey Research IEEE 1394b PCI express cards, with each card having two IEEE 1394b ports. Each port is then connected to a Point Grey Research 5 port IEEE 1394b hub connecting the Point Grey Research Flea2 cameras. Each stereo pair is externally triggered via a signal generator that emits a 10Hz square wave signal.

Table 1: Specifications of the RoboSimian power and battery system.

	Trials Implementation	Intended Finals Configuration
Configuration	24S4P	46S2P
Voltage	88.8 V	170.2 V
Capacity	1.95 kWh	1.87 kWh
Power (continuous)	37.4 kW	18.7 kW
Power (peak)	74.8 kW	37.4 kW
Charge Time	2 hours	2 hours
Mass	12.5 kg	12.5 kg

As with the other elements of the robot, the as-designed power system is predicated on an assumption of operations of much greater duration than a run in the DRC Finals. To this end, the robot is designed to carry a lithium-ion battery that enables an estimated operation time of many hours. At a continuous average power of around 200W, the robot could operate for about 10 hours. However, in practice we expect that the average power draw to be lower due to the percentage of the time waiting on instructions from the operators. While waiting, the power draw could drop into the tens of watts, significantly extending the estimated operation time.

In order to slow the robot down and simplify the electronics, an initial implementation of the battery was chosen for the DRC Trials. This version has a voltage of approximately 90V depending on state of charge. Despite the availability of the DRC Trials battery, the decision was made to further simplify the power system to a remote tethered supply because the mobility penalty was already being paid due to the requirement for tethered communications and emergency stop functions. If the rules for the DRC Finals dispense with tethers, a new battery design may be implemented that will push the voltage up to approximately 170V, which will roughly double the upper end joint velocity. Table 1 shows the specifications of the power system for the DRC Trials implementation and the intended DRC Finals configuration.

3 Software Algorithms

RoboSimian's software architecture and algorithms are designed to enable low-bandwidth, high latency control of a highly capable robot. The robot is capable of performing basic behaviors, such as walking to a designated location, grasping an object, or performing manipulation tasks autonomously. The operator is responsible for designating and sequencing the behaviors to perform more complex tasks, such as turning a door handle, opening the door, and traversing through the doorway. We expect an operator to be at a standoff location, without line-of-sight to the robot, and without a high bandwidth connection to either the robot or to a central command center. Therefore, we limited the amount of processing on the operator control unit (OCU) to a single standard laptop computer, and had the robot provide sufficient situational awareness to the operator to perform complex tasks.

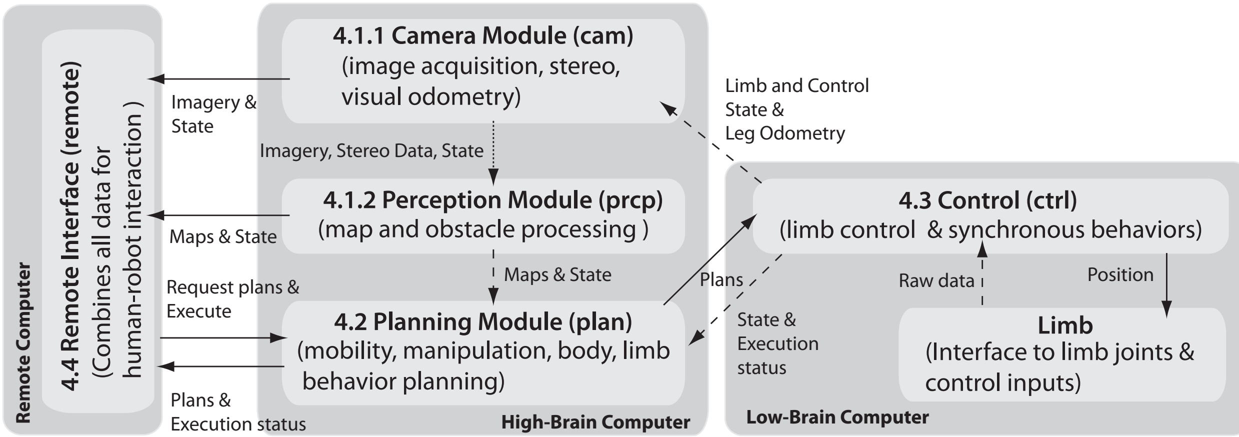


Figure 7: The RoboSimian software architecture. Each light-gray colored block represents a separate software module/process and each arrow indicates the flow of data between each module. Solid, dashed and dotted lines represent TCP, UDP and shared-memory communications, respectively. Each dark-gray colored block represent which machine these processes run. The block number corresponds to a section in this paper.

3.1 Software Architecture

The software of RoboSimian, like other JPL autonomous systems, is model-based and data driven. The software consists of multiple processes running simultaneously across two computers inside RoboSimian as well as one remote operator machine. The two RoboSimian computers commmunicate over a Gigabit Ethernet link. Each RoboSimian computer, the high-brain and low-brain machines, runs 12.04 Ubuntu LTS on an Intel Quad-Core i7 with 16GB of memory. The low-brain machine runs a low-latency (soft real-time) kernel and the EtherLab® open-source real-time kernel module, which runs low level processes such as limb and control processes. The high-brain machine is responsible for higher level processes not concerned with real-time execution but rather higher throughput. Table 2 describes the processes and which RoboSimian machine hosts that process.

Process	Computer	Description
launch-server	remote	Starts/Stops individual processes. Initializes limbs.
remote-server	remote	Remote robot operation and interface to processes on robot.
cam-server	high-brain	Captures imagery. Computes stereo, visual odometry and pose.
prcp-server	high-brain	Produces manipulation and walking maps from multiple stereo cameras.
plan-server	high-brain	Generalized mobility and manipulation planner.
ctrl-server	low-brain	Communicates with the limbs, executes behaviors and motion specifications.
limb-server(1-4 and dd[1])	low-brain	Responsible for low-level motion commands and monitors limb health.

Table 2: Description and location of the processes running on RoboSimian computers used for the DRC.

Processes communicate with each other via shared-memory and/or inter-process communication (IPC) either through TCP or UDP. With IPC communication, each module subscribes to other modules' messages, which are sent asynchronously. Messages and data that can be sent as a constant stream are sent via UDP while one-off messages or messages requiring receipt confirmation are sent via TCP. Data products that are both large in size and sent as a stream are sent via shared-memory. Imagery and stereo data are sent via shared-memory.

The modules consist of the following and are described in further detail in the following sections. The mechanics modeling (**model**) module provides the infrastructure to create, modify, and query a model of the world and robot, and is used in almost all of RoboSimian's modules. The camera (**cam**) module acquires imagery from the stereo cameras, computes stereo range images, and performs visual odometry. The perception (**prcp**) modules takes the image data and produces map and scene information. The **plan** module produces feasible mobility and manipulation plans. The control (**ctrl**) module executes the generated plans and behaviors by commanding the limbs through the **limb** modules. Lastly, the **remote** module is the remote user-interace that views robot data and commands the robot. Figure 7 graphically illustrates the processes and where they run, as well as the data they share and how they share it.

3.2 Mechanics Modeling

The modeling modules (also known as the "model manager") maintains the kinematic, geometric, and other physical properties of the robot, objects, and environment. The model data structure allows for fast (constant time) insertion, removal, and querying of simple (open chain, tree topology) kinematic models made up of physical bodies (links) and virtual joints. The models are structured so that individual properties of a body can be passed between different software modules, allowing one module to update certain aspects of the model and then communicate only this information to another module, which can then update its model. Using the model data structure and current state of the system, generic functions can be easily formed to compute forward kinematics, Jacobians, center-of-mass, and collisions. The model can also be drawn with OpenGL for visualization in operator interfaces or for debugging algorithms.

For collision detection, bodies can be padded to account for uncertainty, and collisions between any two bodies can be filtered in or out. Filtering of collisions is important when the robot is expected to make contact with the environment, an object, or itself. The robot is modeled using primitive shapes, such as boxes and cylinders, to keep collision detection very fast. Objects, such as a tool that the robot might use, are also modeled using primitive shapes, and can be attached to the robot end-effector to allow the system to account for the object's mass and geometry. Similarly, the environment is segmented into oriented bounding boxes, each of which are added to the model.

Specialized algorithms are used for closed form inverse kinematics of the RoboSimian limb, with multiple approaches for redundancy resolution. The most common approach is to search over a free joint and select the solution that minimizes some cost function. Because of RoboSimian's limb design, we allow one of two joints to be the free joint, resulting in solutions that are biased to minimizing pitch or roll joint motions. To reduce computation time, we typically only search a limited angle range from the current joint angle. As a faster but less flexible alternative for IK, we use an iterative Jacobian-based solver seeded with the limb's current joint angles, or a predefined set of joint angles.

3.3 Cameras

The vision system of RoboSimian is built using only low-cost passive cameras. The system is comprised of only stereo cameras and are used to provide both context from imagery and 3-D data from stereoscopy. Figure 8 shows the designed layout of the stereo cameras. RoboSimian is designed to provide complete 360° situational awareness (SA) by having $10cm$ baseline stereo cameras all around the robot perimeter for hazard avoidance (also referred to as hazard cameras or HAZ cameras). 360° coverage is provided through the use of fish-eye lenses. These lenses provide a horizontal field of view (FOV) of 105° and a vertical FOV of 73°. Each camera in RoboSimian is the same Point Grey Reseach Flea2 color camera with 1024×768 pixels which as a result allowed for simple replacement.

The HAZ cameras alone cannot provide the scene context and map resolution that would be needed for the finer manipulation tasks that RoboSimian would face. To resolve this, two additional pairs are also added to the front face of the robot, stacked vertically around the front facing HAZ cams. In the walking stance, one of the additional pairs ($10cm$ baseline) would face straight forward to aid in navigation (NAV).

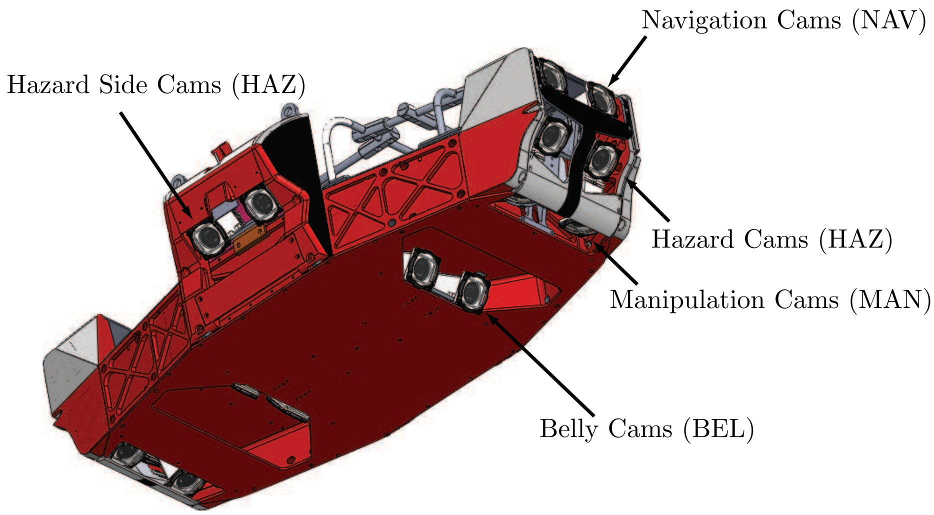


Figure 8: RoboSimian chassis with the designed stereo camera layout.

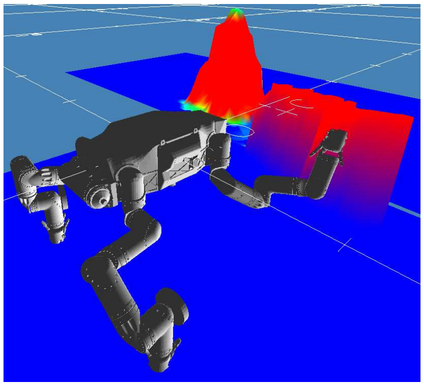
In the sit posture, the second additional pair (12cm baseline) would now face straight forward to aid in manipulation and driving (MAN). Note that whether forward-facing or rear-facing, RoboSimian is designed to be symmetric, so any additional stereo cameras added to the front end have also been added to the rear end. In addition to the HAZ, NAV, and MAN camera pairs, RoboSimian is designed to include a set of stereo cameras mounted in the "belly" (BEL) of the robot that would be primarily used when RoboSimian is in an upright posture. Note that due to time constraints, at the DRC Trials the three stereo pairs on the front face of the robot were the only cameras wired and in use.

Each camera's intrinsics are calibrated using a CAHVORE model (Gennery, 2006), which handles fish-eye lenses. The extrinsics calibration is broken up into two steps. First, each pair is calibrated with respect to each other using a dot-patterned calibration board. If the two pairs are able to fully view the calibration board, the relative pose between the pairs can be resolved. There are cases on RoboSimian in which two pairs cannot simultaneously view the board (e.g. MAN and HAZ). In this case, an intermediate camera is used to act as a link between the two pairs to resolve two relative poses. These poses are then chained to generate a complete relative pose between the camera pairs. The second step in the extrinsics calibration is to compute the relative pose between a stereo pair and the robot base frame (the frame used for the robot pose). This step is done by detecting fiducials on RoboSimian's front limbs and computing the relative pose between the kinematic fiducial pose in the robot base frame and the detected fiducial pose in the camera frame. This is discussed more in Section 3.4 and illustrated in Figure 11a.

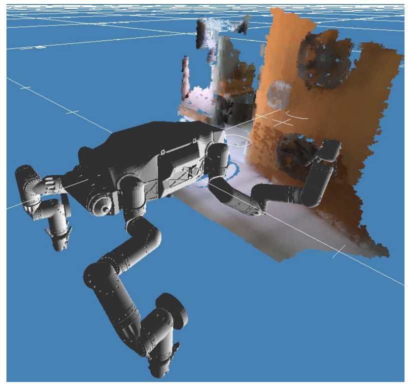
The **cam** module is responsible for capturing all cameras simultaneously. Each camera is triggered externally via a signal generator. Each stereo pair's acquired imagery is rectified and processed in the stereo processing pipeline. Stereo is continually processed on all pairs on RoboSimian. Likewise, visual odometry (VO) is also processed for each pair which provide VO pose updates. However, at any one time, only one camera pair's VO updates are used to compute the pose of the robot. However, if there is camera occlusion or if VO fails, the camera pair used for VO is switched to the next available and valid pair. Stereo data and robot state are then passed to the perception (**prcp**) module, described in the following subsection, and are used to compute environment maps and obstacle bounding boxes. This data pipeline is shown and described in Figure 7.

3.4 Perception

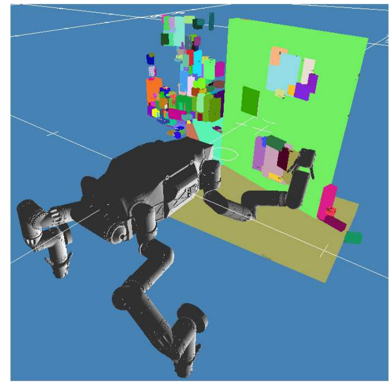
The perception system is responsible for building, maintaining, and processing 3D maps based on the stereo range images and the visual odometry poses that are computed by the camera module. The output of



(a) The walk map.



(b) The manipulation map.



(c) The bounding boxes of map segments.

Figure 9: Maps built by the perception module with range images obtained from RoboSimian's stereo cameras a) walk map b) manipulation map c) Surface normals based map segmentation produces object bounding boxes.

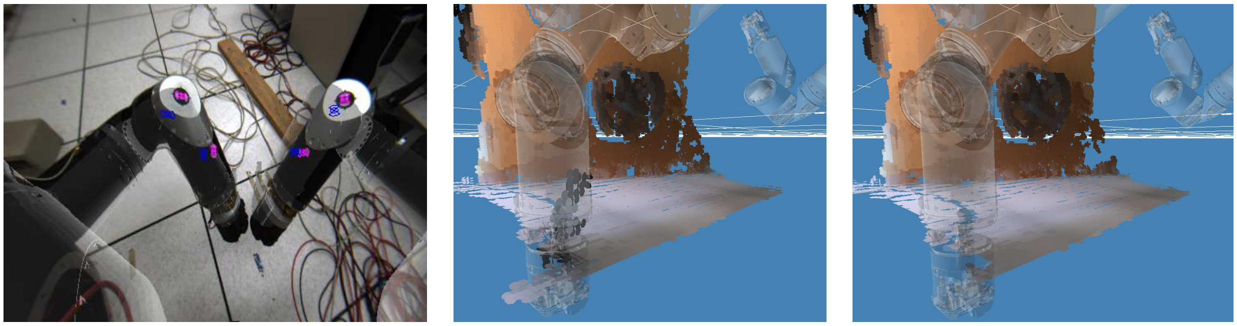
the perception system is used by the planning module and the operator for collision-free motion planning, insertion of virtual objects into the map, and visual inspection of the environment as perceived by the robot.

The tasks in the DRC Trials impose a variety of features and minimum requirements that a mapping system should provide. To this end, the perception module maintains two specialized voxel-based maps. The first one, the manipulation map, is a high resolution (typical values ranges in between $0.02m - 0.05m$), n-tree 3D map structure based on previous work (Bajracharya et al., 2012). This map is used when moving the end-effector to manipulate an object in the world and is sent to the remote module for visualization upon request. The second is the walk map, which is a coarse elevation map for walk planning. The walk map is lower resolution and provides the basis for foot-step planning over rough terrain.

Due to its high resolution and full 3D structure, the manipulation map requires higher memory and processing compared to the the walk map. The real-time requirements of the system are strained if the manipulation map is built at large scales. High fidelity 3D structure matters most when manipulating objects. The manipulation map's computational footprint is kept bounded by expiring voxels in the map that are not seen in a set number of images and keeping the map size small enough to cover the workspace of RoboSimian's end effectors. In contrast, the walk map is maintained for the entire extent of a run. The 2.5D nature of the walk map also ensures faster planning algorithms.

RoboSimian's maps are used for collision free motion planning. Since RoboSimian's sensor coverage includes the limbs' workspace, often 3D points from RoboSimian's own body appear in the map. This in turn affects collision checking as the robot appears to be in collision with itself. The solution chosen for this problem is to simulate a given camera's acquired image in a 3D virtual world (OpenGL context) tuned to that camera's model parameters and solely populated by the robot's CAD model. With up-to-date joint angles and given that the actual robot is built similar to the CAD model, this virtual view acts as a mask image to the stereo images that return range points. Thus using this mask image, 3D points that belong to RoboSimian's own body are deleted from acquired range images before they are inserted into the map.

In order to accurately project RoboSimian's CAD model into a camera frame, the camera coordinate frame needs to be known with respect to the robot's coordinate frame. To this end we have used a fiducial detector from previous work (Hudson et al., 2014). By placing fiducials at known positions on the limbs, the 6-DOF fiducial pose in the robot frame ($robot2fiducial$) can be computed by forward kinematics. Further, by detecting the fiducials in acquired images, their pose in the camera frame ($cam2fiducial$) can be obtained. The camera's position in the robot frame is then obtained by running a least-squares optimization method that minimizes the Euclidean distance between the two fiducial poses. Figure 10 shows an example of this



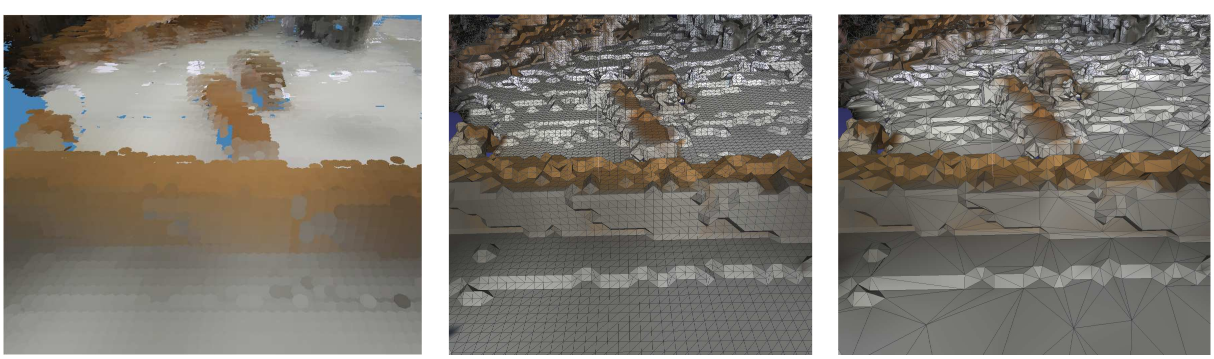
(a) (b) (c)

Figure 10: Removing 3D points that belong to RoboSimian's body is a necessary step for collision checking. a) Fiducials on the arms and initial estimate of robot frame overlaid with transparency. Blue marks correspond to the initial estimate of fiducials and pink marks correspond to actual found fiducials in the image. b) 3D points from RoboSimian's limbs appear in the map. c) Robot points are cleared from the map.

process.

Another type of processing performed on the 3D maps is surface normals based segmentation of the map into objects. As a result of segmentation, parts of the map that generally constitute a single entity (such as the ground plane or a brick that sits on the floor) can be abstracted to a bounding box. These bounding boxes are then used in collision checking since collision checking solely based on map voxels would be prohibitively slow, especially when a given configuration needs to be evaluated for collisions during motion planning.

Generating meshes from voxel maps has several benefits. In the context of the DRC where bandwidth is limited, transmitting each voxel in the map can incur long latencies which wastes precious operator time. Furthermore, collision checking with the object bounding boxes explained previously can result in boxes that contain significant amount of free space, unnecessarily limiting the motion capabilities of the robot. In contrast, a mesh of the map that consists of triangles will preserve the true surface boundaries of objects. The mesh structure can also be decimated so as to reduce the amount of triangles. This reduction removes detail that is redundant as in planar parts of the map or unneeded as in parts that are far away from the robot whose fine structure is not relevant to the task at hand. For mesh generation, the marching cubes algorithm is performed at each voxel (Lorensen and Cline, 1987). By looking at the surface crossing at each voxel, this algorithm generates triangles at a per voxel basis, hence being highly parallelizable. Other meshing algorithms such as dual contouring and adaptive skeleton climbing that have certain advantages



(a) (b) (c)

Figure 11: Maps showing the mesh representation of a manipulation voxel map. a) The fine voxel map used for manipulation purposes. b) Initial mesh generation from the voxel map. c) The subsequent decimated mesh of the voxel map.

over marching cubes were not considered because of the computational penalty that comes with them (Ju et al., 2002) (Poston et al., 1998). The decimation of the mesh is done by determining the cost of removing each vertex in the mesh and greedily removing those vertices until the resulting mesh has a certain number of vertices.

3.5 Remote Operation

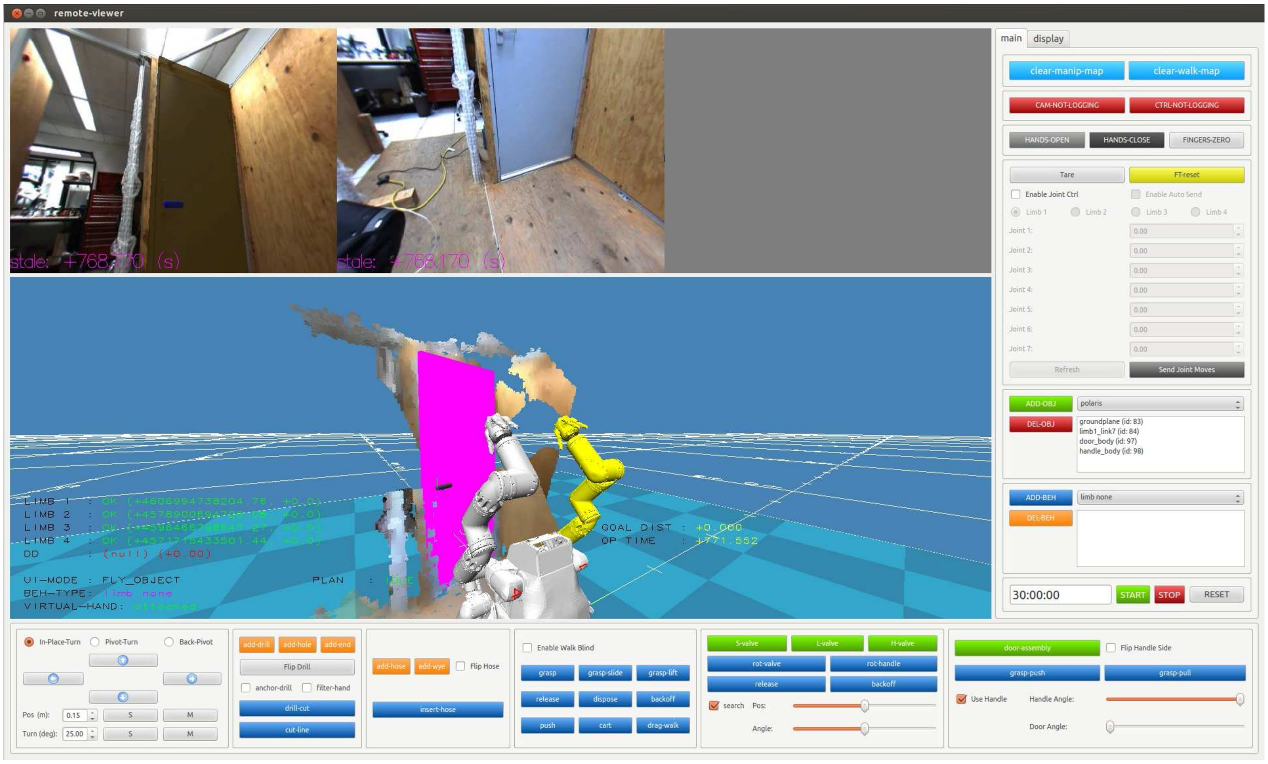
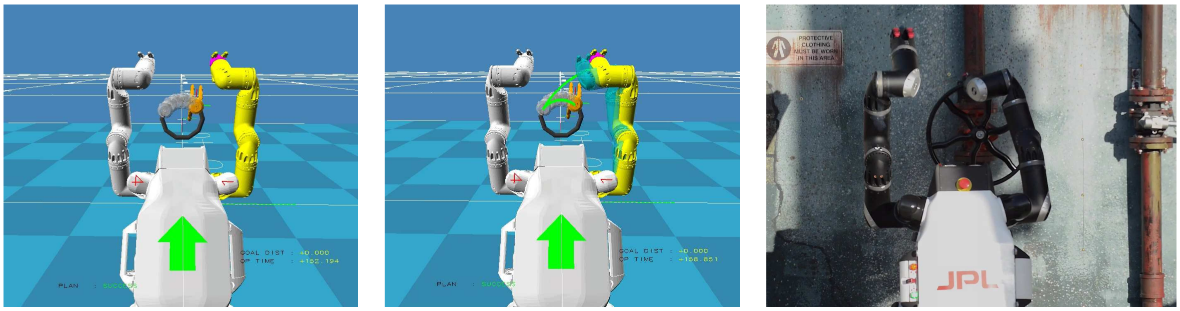


Figure 12: The remote operator interface for RoboSimian is shown above. A number of widgets were added to simplify user interaction and optimize speed to achieve certain tasks. The top window is designed to display the left camera image from each available stereo-camera pair. The main center window is designed to display a 3D rendering of the world for situational awareness. This window plots the current robot pose, the intended robot action, and the sensed environment from perception maps. It also serves as a window for operator interaction to move and manipulate objects and the robot.

The primary purpose of the remote module was to control the robot and model the world. The remote-interface allowed for configuring the robot "end-state" and requesting a plan solution to move all the limbs and joints to properly end in that configuration via simple key-presses and mouse clicks. It also allowed for inserting known models of objects (e.g. valves, ladders, hoses, etc.) into the world manually so that RoboSimian could interact with objects for manipulation.

The remote interface peripherals to RoboSimian proved essential to controlling the robot and were designed to be easy to use for an operator with minimal training and limited information / options displayed to avoid clutter and overload of information. Several different user input devices were considered and implemented for controlling RoboSimian: the Razer Naga Hex gaming mouse (which afforded additional key presses on the side of a traditional mouse), the Omega series haptic device from Force Dimensions, 3D monitors, and the SpaceNavigator by 3D Connexion. It became readily clear in our testing with these devices that aside from the traditional mouse and keyboard, any additional non-native computer peripheral required a significant amount of tuning at the driver level and user practice to become quick and effective. Consequently, a simpler approach of a color-coded keyboard and mouse with user-defined hotkeys was decided upon. The graphical user-interface itself (shown in Figure 12) was written with the Qt library for widgets (buttons, sliders, checkboxes, etc.) and OpenGL extensions for plotting of 3D data and stereo-camera images.



(a) (b) (c)

Figure 13: (a) A gray trail of hand locations offers the operator a preliminary preview of the action the robot will take. (b) A successful motion plan is shown in green, with the previewed robot motion shown in transparent blue. (c) RoboSimian is shown completing the valve turn during the DRC Trials.

Due to bandwidth limitations imposed by the competition rules and guidelines, real-time teleoperation of the robot was not possible. As such, semi-autonomous behaviors were developed that allowed the user to specify any chained sequence of behaviors to follow. Figure 13a shows one such sequence that was used for the valve task. It consists of a “grasp” behavior to grasp the valve, followed by a large “rotate” behavior to rotate the valve a specified amount about the rotation axis, and finally a “release” behavior to open the hand. If a collision-free plan existed to execute the entire sequence, RoboSimian would execute the motion from the beginning behavior to the end. Complicated sequences of behaviors were eventually scripted into single actions that could be called by the operator with a single button press (e.g. “rotate-valve”, “insert-hose”, “push-open-door”), depending on the task.

Note that since the **plan** module was run on the robot high-brain computing stack and not the same machine as the remote module, only plan “previews” were sent back to the user to inspect and verify the intended plan motion. Only once the user verified a given “preview” plan would the robot then proceed to execute the entire plan. Figure 13b shows the current robot pose in gray and the previewed robot motion in transparent blue. Figure 13c is a live image taken during the actual competition of RoboSimian executing the valve turn.

To establish a sense of scene-understanding and situational awareness, the 3D voxel map used in the perception process along with the stereo images can be sent to the remote module interface. To optimize the amount of data sent over the limited bandwidth, only voxel deltas are sent when delivering map data and compressed images are sent when delivering imagery. Figure 12 shows the situational awareness provided by the voxel map representation of the world along with the left-stereo camera images from the available stereo camera pairs. Additionally, because the objects with which we intend to interact with are well known and modeled, a graphical object model is overlaid in the stereo image (top left image of Figure 12). This feature combined with the 3D voxel map afford the user the capability to accurately fit objects into the world before interacting with them.

3.6 Planning

Planning of behaviors was performed onboard the robot using a combination of a mobile manipulation motion planner and a behavior planner. Given a paramterized behavior, such as grasp an object, and the world state from perception data, the behavior planner determines the goal state for the robots body and limb end-effectors. The motion planner is then used to determine the motions of the limbs to maintain stability and achieve the desired body and end-effector poses.

3.6.1 Mobile Manipulation Motion Planning

RoboSimian uses a unified mobile manipulation motion planner to generate all the motions that it executes. The planner ensures that the robot stays statically stable at all times, and that all motions are kinematically feasible, kinodynamically smooth, and collision free (except when contact is expected). Motions are continuously replanned to account for sensing or execution uncertainty. Planning time scales with the complexity of the problem, but typical plans are generated in tens of milliseconds.

With four 7-DOF limbs, four 3-fingered hands, and its differential drive wheels, RoboSimian is an extremely capable and flexible mechanism. However, because of the number of degrees of freedom, achieving a desired motion by manually moving the actuators is difficult and dangerous. Furthermore, manually coding efficient and safe motions, even for relatively simple sequences such as statically stable walking, is difficult due to the amount of coordination required and extremely flexible, but sometimes non-intuitive kinematic configuration of the robot. Consequently, to keep the robot safe and stable, and to exploit the robot's capabilities, we require that all motions be generated by a motion planner.

In order to exploit RoboSimian's ability to walk, climb, sit, manipulate, roll, and smoothly transition between behaviors, we developed a single, unified mobile manipulation planner capable of planning all motions. In parallel, we also developed specialized planners that were optimized for certain behaviors, such as walking and driving. While these planners could perform specialized tasks better, transitioning between the planners was a non-trivial integration problem. Ultimately, due to severe time and resource constraints, we did not use the specialized planners for any tasks.

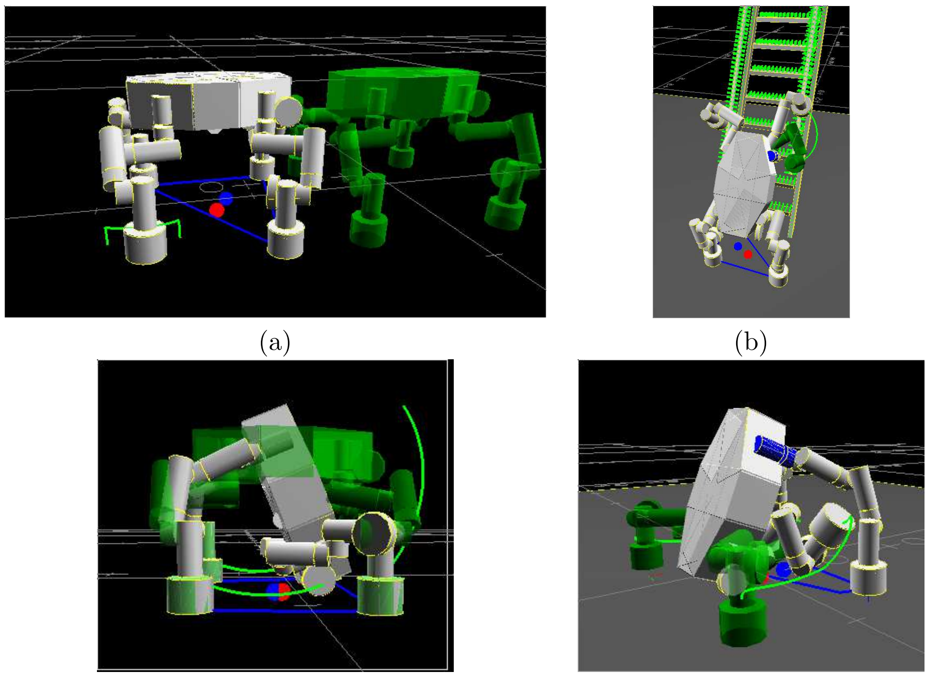
We achieve generality and speed by decomposing the mobile manipulation planning problem into a combination of open-chain and closed-chain motion planning problems. The open-chain planner plans the motions of an unconstrained serial manipulator system in joint space. The closed-chain planner plans the motion of a parallel manipulator system in Cartesian space of the manipulator base frame. For RoboSimian, this can typically be thought of as moving a limb end-effector, or moving the body while keeping the robot's points of contact stationary. In combination, the two planners can perform almost any task, including walking, climbing, transitioning to sitting, adjusting the body posture, single-limb manipulation, dual-arm manipulation, and driving.

The open-chain planner is simply an RRT-Connect planner (Kuffner and LaValle, 2000) with greedy path smoothing (Hudson et al., 2014). It can be used on any number of degrees-of-freedom of a serial manipulator, but we typically only use it for a single 7-DOF limb, attached at the chassis shoulder. This planner assumes the end state is specified as joint angles; because there are typically redundancies in the manipulator, the goal angles are chosen by a behavior planner, which optimizes the use of the redundant degrees-of-freedom when perfoming the limb inverse kinematics (see Section 3.2).

The closed-chain planner is implemented by simply interpolating the base frame from its current pose to the goal pose, with the possibility of a single randomly selected intermediate pose. During each step of interpolation, the joint angles are computed from the limb IK for limbs in contact. The intermediate pose is only used if the initial motion is not feasible, and is limited to only several iterations of random sampling. The search could be implemented as an RRT, but we found that it was not necessary, as the intermediate poses were very rarely necessary.

Due to the planner decomposition, the natural way to specify tasks is by specifying a desired body pose (position and orientation) and the desired limb end-effector poses. Given these inputs, a high level planner searches for a global plan using the two base planners to achieve the desired end state. A simple finite-state-machine (FSM) is used to transition between the planners and some basic behaviors. The high level global planner is a brute force search over body and limb motions to achieve the goal. For example, for walking, the high level planner tries to achieve the desired body pose and end-effector poses by moving the end-effectors in a fixed order, moving the body to remain statically stable on the remaining limbs or points of contact. If the goal cannot be achieved, it then computes an intermediate goal, and tries to achieve it in the same way. If no global plan can be found with the limb order being used, it uses a new limb order. In practice, we use seven limb orders, which are selected by regression testing all possible limb orderings in simulation, and choosing the ones that most commonly lead to a feasible solution.

For each high level command provided, the planner first computes a global plan for the entire motion sequence to ensure it is feasible. However, when the system actually starts executing the sequence, each motion is replanned before it is executed to account for inaccuracies in execution. For computation efficiency, only the single next motion is planned, rather than replanning the entire sequence to the goal. This assumes that there are no large deviations during execution, which is reasonable since the system is designed to fault and stop if there is a large unexpected deviation. The replanning architecture also allows the planner to react to unexpected end-effector poses. If the motion hits the terrain early or late, the planner reacts by adjusting the body posture during the next move. In general, the planner tries to keep the body roll and pitch aligned to the average pitch and roll of the end-effectors. This naturally makes the robot climb up steep objects, such as a wall or ladder.



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

Figure 14: (a) A visualization of the planner performing a limb motion for taking a step; the robot is shown with the representation used for collision checking (made up of primitive shapes) and the padding around those shapes; the robot in gray is the current planned state of the robot, and the transparent green is the starting state; the blue lines show the polygon of support; the blue sphere shows the center of the polygon of support; the red line shows the robot's planned center of mass projected onto the polygon of support; the green line shows planned limb wrist trajectory. (b) A plan for climbing a ladder, with holds automatically generated from a simple model of the ladder; the small green lines show the normal for possible grasp or step locations. (c) and (d) A plan for transitioning from an upright posture to a sitting posture; (c) shows the four limb motion required to transition (but the limbs wrists will stay stationary, and the body will move), and (d) shows the plan for placing the caster wheels on the ground in front of the robot.

3.6.2 Behavior Planning

In many cases, generating the task specification (the body pose and limb end-effector poses) for the motion planner is non-trivial. For example, when turning a valve, the robot needs to know where to grasp the valve so that it can also perform the turning motion. For walking, the robot needs to know where it can or cannot step, while still allowing the body to move to balance for the next step. When climbing, the robot needs to know what it can grasp, step on, or support against. To generate the motion specifications, we use a behavior planner.

The input to the behavior planner is a behavior specification, which consists of a behavior type, and the input parameters specific to that behavior. For example, a "grasp-and-rotate" behavior is specified by a position and orientation in which to grasp an object, an axis of rotation, and the amount to rotate. This can then be used for tasks such as opening a door handle, opening the door, or turning a valve. The output of the behavior planner is a ranked set of fully specified behaviors, which include the desired body pose of the robot, the joint angles of the limbs required for the behavior, and other parameters, such as the expected force on the end-effector, and what controllers and checks to run (e.g. force control in the end-effector z-axis during a certain motion).

Behavior planning is broken up into manipulation behaviors and mobility behaviors. For manipulation planning, the goal is to determine the best end-effector pose and starting joint angles of a limb to perform a certain behavior. For grasping, the operator can specify a desired end-effector orientation, an orientation with mirroring along an axis, or a palm normal. The behavior planner then searches over the possible orientations, discretizing the search when necessary (e.g. over yaw around the palm normal). The behavior planner can optionally also perturb the pose if no solutions are found at the specified poses.

For each pose, the behavior planner first determines if there is an IK solution for the desired pose. Then for each IK solution for the pose, it forward propagates the expected behavior motion and verifies that a continuous IK solution (i.e. no configuration changes) exists for the entire behavior. The cost of each solution is the length of completion, or for motions that complete enough of the behavior, the cumulative joint angle distance from the current joint angles to the solution. The motion planner is then called to test feasibility of achieving the start position of the behavior, in order of lowest cost solutions, exiting after the first feasible solution.

For mobility behaviors, such as walking, climbing, or shifting the body, the behavior planner is responsible for producing a set of body and end-effector poses, and a IK reference posture, which is then fed to the motion planner. While a complete system would use perception data to autonomously compute a path of body poses through the terrain, due to time constraints we required the operator to navigate the terrain by providing a delta goal position and orientation to the robot. Typically when walking, these delta motions were 1-2 meters long. Using the desired goal pose, the behavior planner then interpolated from the current robot pose to the desired pose, placing nominal limb holds for each limb in succession over the body motions. The effective step length could be regulated by changing the body delta (in position and orientation) used to place a new limb hold. In practice, the planner adaptively selected the step length by starting with the longest distance and then reducing it if no solutions were found by the motion planner.

When walking or climbing, the behavior planner first selects a nominal limb hold based on the expected body pose. This is independent of any terrain knowledge, and so can be used for completely blind, reactive motions. However, when terrain knowledge is available, either generated from vision data, or supplied by a user model, the planner automatically selects holds by searching around the nominal hold. When a grid map is available, the planner searches a local neighborhood in the map for feasible holds, and returns the best hold. When a model is provided by the operator, or when holds are specified directly by the operator, holds are indexed into a KD-tree and nearest neighbor search is used to find feasible holds within a fixed distance of the nominal hold. Holds are classified as weight holds, which are flat z-normal weight bearing holds (e.g. the ground), balance holds, which are flat but not z-normal weight bearing (e.g. a wall), or grasp holds, which can be grasped (e.g. a railing).

The IK reference posture biases the motion planner to choose solutions that minimize the distance to a set of reference joint angles. This allows the robot to "walk" in different ways for the same set of limb holds. For example, with a reference posture of the limbs pointing down under the chassis, and a free pitch joint, the robot will tend to walk using its pitch joints, similar to a horse or dog. But for a reference posture with the limbs pointing out from the chassis, and a free yaw joint, the robot will tend to walk using its yaw joints, similar to a spider. The behavior planner can change the reference posture at any time, and the planner will naturally migrate to that solution as it moves. This is useful for transitioning between a low sprawled walk, which is more stable and kinematically feasible on very rough terrain, and walking tall, narrow, and upright, which is useful for walking through narrow openings or up stairs.

3.7 Control & Behaviors

The control and behavior process is the on-board robotic motion execution system. Behaviors are represented by a asynchronous hierarchical state machine, which defines task frame control specifications and monitors execution of the synchronous fixed-rate control loop. The control loop is responsible for achieving specified task frame goals, and inverting the desired motions into joint space trajectories which are executed on each limb's distributed motor controllers.

3.7.1 Behaviors and Sequences

The asynchronous behavior system defines desired task frame motions and feedback set-points for each limb. The behavior state machine translates high level requirements (such as moving a limb into contact) into motions (task frame positions and motions) and set-points (desired end effector force). The behavior system is executed as a finite state machine, for instance first moving along a velocity vector, then detecting contact, and then achieving a desired force set-point. Each behavior also provides abstraction for the planning engine, translating the state machine into an efficiently searched kinematic representation for verification of reachability, balance, and force limit checking.

The behavior state machine architecture was previously developed in the DARPA Autonomous Manipulation Software (ARM-S) Program (Hudson et al., 2014). The following definitions and concepts follow from this program.

An action is defined as a single finite state of a behavior. An action defines the current control goals, for instance moving the end effector around a valve axis, and tracking a desired torque about the axis. Specifications are not required to be in orthogonal axes, and may compete for desired end effector motions. An action also has a set of end conditions that dictate when an action is complete or when an error has occurred. An action will monitor both extrinsic signals, such as increases in valve rotation torque, indicating reaching the end of a valve handle's travel, or internal fault state, such as reaching kinematic constraints.

A behavior is implicitly a hybrid automaton, where the control goals define the continuous motion of the system, and the guarded transitions between states are defined by the end-conditions and transitions of the finite state machine. A behavior encapsulates the idea of automatic fault recovery: the behavior state machine is programmed to differentiate between conditions that are recoverable, with a predefined recovery sequence, or conditions that require halting the execution sequence.

To exploit operator involvement in the system, and to simplify the development and testing of behaviors, RoboSimian utilized the sending of behavior sequences to the robot. A behavior sequence is simply a sequential script of behaviors, with transitions between behaviors constrained to the sequence, or exiting on a single behavior failure. The on-board execution system is agnostic to the use of a behavior compared to a behavior sequence; for instance, a complicated grasp-rotate-release behavior could be created, which would execute exactly as a sequence of a grasp behavior, a rotate behavior, and a release behavior sequence. The use of sophisticated hierarchical behaviors was important in the DARPA ARM-S program, where completely autonomous execution was required. Behaviors can have well defined and logical transitions between states to facilitate automated fault recovery. However, given the ability and desire to have a human user monitor the robotic execution in the DRC, it was safer and easier to test a set of simple behaviors with simple fault conditions, and to rely on the operator to create and reorganize a behavior sequence. This also has the advantage that the human can step though each behavior or send novel sequences to the robot during trials to cope with unexpected situations, but does lack the automatic fault recovery present in DARPA ARM-S.

The most complicated behavior tested on RoboSimian was an “open-door” sequence. This sequence was composed of a grasp, rotate (handle), rotate (door), release behaviors, all completed while in a walking stance. This sequence allowed automatic determination of where to place the robotic body to open the door, however this was not utilized in the DRC Trials. Instead, RoboSimian opened the door while on it’s differential drive, using a grasp, rotate (handle) sequence, and then having the user manually put the robot’s free leg (limbs) into the open door gap, and rotating on the differential drive. Utilizing behaviors sequences allowed for verification of many subtle occurrences due to the limited kinematic range of the limb manipulators. When turning the valve, a grasp-rotate-release sequence was scripted, and used by the planner to search for an admissible starting grasp location. The behavior-planning system was able to down select initial grasp poses as the required release motion from the valve would cause intersection with the robot.

3.7.2 Control

Actions are translated into Generalized Compliant Motion (GCM) controllers (Backes, 1994) further developed under the DARPA ARM-S program (Hudson et al., 2014). GCM translates competing control objectives into end effector positions trajectories or velocity set-points by representing control goals as linear second order systems in the task frame and superimposing them. GCM has the significant advantage of making specifying behaviors simple, allowing arbitrary frames and competing control objectives to be specified, with relative importance being represented by the stiffness of the representative second order system. In both DARPA ARM-S and the DRC, a small set of task frame control objectives: force, Cartesian trajectory tracking, visual servoing (minimizing visual tracking goal errors), dithering (random searching motions, such as wiggling a key), kinematic constraint satisfaction were sufficient to execute all tasks.

The GCM fusion of control objectives in the task frame produces a continuous desired end-effector motion. This is then translated into desired joint space motions through numerical inverse kinematics. GCM controllers and numerical IK were run at 250 Hz in a soft real time loop to produce desired joint position and velocity set-points. A separate process for each limb interpolated these set-points at 1kHz and sent position goals to each motor controller over EtherCAT. The distributed Elmo motor controllers track joint positions at 10kHz in hard-real time.

The 10kHz PID controllers produced high bandwidth joint tracking, and the 100:1 motor gear ratios create a stiff and precise limb, which requires all task frame controllers to produce smooth joint trajectories. No feed-forward control or intrinsic limb compliance were utilized to achieve control objectives. Both of these later approaches have been shown to be advantageous (Hudson et al., 2014), (Pastor et al., 2011) (Kalakrishnan et al., 2013) in the ARM-S program, but the high control bandwidth in RoboSimian enabled achieving adequate task frame compliance using sensed force feedback.

This end effector compliance was required for robust walking over uncertain terrain. By simply specifying desired contact forces for behaviors, executed every step touchdown, the open loop planners were able to walk blindly over the entire DRC walking course.

3.8 Ongoing Mobility Research

Traditionally, kinematic redundancies have been avoided in robot leg designs because of their inherent increased mechanical and planning complexity without clear benefits for walking. In contrast, redundant kinematics are favored for manipulation because of increased dexterity. Because RoboSimian uses the same kinematically redundant limbs for mobility and manipulation, effective strategies for choosing inverse kinematics (IK) solutions and planning walking motions are key to improving RoboSimian’s ability to traverse rough terrain.

Ongoing research activities toward this goal have focused on the use of an IK lookup table designed to give unique solutions with smooth joint trajectories for any end-effector trajectory in the (potentially truncated) workspace. This IK table could be used on all limbs at all times to create effectively non-redundant limbs, but at the cost of a great deal of capability. Instead, we use the IK table only at limited times, and at other times allow for arbitrary motions in joint space on certain limbs.

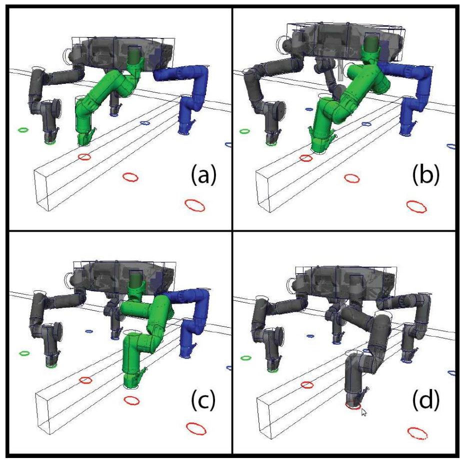


Figure 15: Animation stills showing RoboSimian stepping over an obstacle. The blue limb is the dominant limb (which determines the body pose), the green limb is the swing limb, and the grey limbs are dependent. a) The step motion begins from a small standoff distance above the foothold. b) The swing leg clears the obstacle. c) The step motion ends a small standoff distance above the destination foothold. d) A small cartesian move is used to make contact with the foothold.

We use the IK table to determine the positions of all four limbs at the beginning and end of a step or body shift motion. Uniquely determining the end pose allows the use of planning algorithms such as RRT Connect, and the proper design of the IK table ensures that the choice is appropriate. During these motions, one limb (known as the dominant limb) is allowed to move arbitrarily in jointspace. The forward kinematics of the dominant limb determines the position of the body relative to the world. When there is a swing limb, it is also allowed to move arbitrarily in joint space during the motion (the dominant and swing limbs can be considered to be one serial mechanism). Achieving kinematic closure of the remaining non-dominant and non-swing (dependent) limbs is accomplished by using IK table solutions for those limbs between the body pose (which is determined by the dominant limb) and the footholds for the dependent limbs. Figure 15 shows an example of this process.

It is our hope (supported by tentative simulation and experimental results to date) that this approach will substantially improve RoboSimian's mobility performance in terms of speed and tolerance for difficult terrain.

4 Field Testing

The RoboSimian robot has been tested on a variety of mobile manipulation tasks in lab environments, outdoor in controlled but unstructured tests, and as part of the 2014 DARPA Robotics Challenge (DRC) Trials. In order to understand the limitiations of the system, when preparing for the DRC Trials, we tested the robot in conditions, terrains, and tasks that were harder and less structured than the tasks at the Trials.

4.1 Trial Results

The DRC challenge was broken into 8 tasks that focused on mobility and manipulation. The tasks were broken up over two days and each task had a time limit of 30 minutes. Human interventions were allowed so that the task and robot could be reset, but with each intervention there was a 5 minute penalty. Each task consisted of 3 subtasks for 1 point each. If the task was completed succesfully without any interventions, a bonus point was awarded for a total of 4 points. Each robot was required by DARPA to be tethered for communications and power. In addition, each robot had the option to be belayed for safety; RoboSimian was the only robot to not use a belay during the competition.

Two of the tasks focused primarily on mobility. These tasks were the *Ladder* and *Terrain* task. The Ladder task required the robot to climb an industrial step ladder which was 8 feet tall and could either be at a 60° or 75° angle with or without hand rails. Stepping on the first rung was 1 point, the fourth rung was 1 point and reaching the clearing was another point. The *Terrain* task was a 40 feet obstacle course primarily made of cinder blocks. It was broken up into three distinct sections of increasing difficulty each worth 1 point. The *Vehicle* task consisted of 2 subtasks. The first subtask required the robot to drive a Polaris Ranger XP900 through a 150 feet course worth 1 point. The second subtask required the robot to dismount the vehicle and was worth 2 points. The robot started in the vehicle with the vehicle running and in gear. To dismount, placing the vehicle in park was not required.

Manipulation tasks were *Valve*, *Hose*, *Door*, *Wall*, *Debris*. The *Valve* task consisted of turning 3 valves (small and large circular valves and a handle valve) that were placed side by side on a wall. Closing each valve, which required a full turn for the circular valves or a 90° turn for the handle valve, was worth 1 point each. The *Hose* task consisted of 3 subtasks each worth 1 point. The first subtask was to grasp the hose from a reel and carry it 4 feet, the second was to place the hose in contact with a wye and the third subtask required the hose to be screwed onto the wye. The *Door* task consisted of opening three doors (a push door, a pull door and a weighted pull door) in succession worth 1 point each. The *Wall* task consisted of grasping a cutting tool and cutting a triangle in a wall composed of drywall. Each complete cut of each side of the triangle was worth 1 point with the last point requiring that the triangular piece be removed. The *Debris* task consisted of removing wood pieces from an entrance of a doorway. Removal of 10 pieces of wood from the pathway was worth 2 points. Once the pathway is clear, the robot was required to cross the doorway for the final point.

The DRC Trials took place over two days on December 19^{th} and 20^{th}, 2013. RoboSimian's schedule for the first day consisted of the *Terrain*, *Ladder* and *Driving* tasks. RoboSimian attempted the *Terrain* task first and was able to achieve 2 points in the 30 minutes with 0 interventions. The first point was achieved relatively quickly in the run. The first cinder block hill obstacle presented the most difficulty which caused several hyper-extensions of RoboSimian's limbs. These hyper-extensions often prevent RoboSimian from taking another step. Undoing these extensions are time consuming and as a result, forced JPL to reach the maximum time limit. Given the short project timeline and preparation for the DRC trials, the *Ladder* and *Driving* tasks presented significant safety risk for the robot. JPL chose not to attempt those tasks primarily insure the robot for the second day of tasks, consequently scoring 0 points for those two tasks.

Successes in the second day represented the bulk of RoboSimian's points. RoboSimian used the upright driving posture as shown in Figure 1b in the following manipulation tasks. The *valve* task was the first event on the second day. RoboSimian scored 4 points on this task with 0 interventions within 29 minutes. One difficulty that manifested during this task was that RoboSimian falsely stopped during the first two valve attempts. It was only on the second valve did the operators realize that the valve model that they were placing into the virtual world, from which the rotate behaviors are generated, did not precisely match the visual data of the valve. The valve model was misaligned causing the rotate behavior to rotate around an incorrect axis. This misalignment triggered RoboSimian's force safety system forcing the operator to continue re-attempting the valve turn. This delay caused RoboSimian to only complete the *valve* task in the last minute.

Team	Score (pts)
Schaft	27
IHMC	20
Tartan Rescue	18
MIT	16
RoboSimian	14
Traclabs	11
WRECS	11
Trooper	9

(a) Ranking by points of top 8 teams that competed in the Trials.

Task	Score (pts)
Obstacles	2
Ladder	0
Driving	0
Valves	4
Debris	4
Door	2
Hose	2
Wall	0
Total	14

(b) RoboSimian's points by task.

Figure 16: DARPA Robotics Challenge Results - breakdown by team and by task

The *hose* challenge was the next task to complete. RoboSimian achieved only 2 points in the 30 minutes with 0 interventions. This task was one of the least tested tasks in our development. RoboSimian was able to successfully grasp the hose hanging from the reel and carry it past the yellow line for the first point. RoboSimian was able to position itself in front of the wye and place the hose directly on the wye inlet. Due to the project timeline, JPL did not have a behavior in place to rotate the hose onto the wye. As a result, the operators had to teleoperate the robot by joint moves in an effort to secure the hose onto the wye. It was difficult to view the wye and the hose through the cameras as the hand obstructed the view of the of wye. RoboSimian operators let go of the hose thinking it was securely fastened on the wye. The hose dropped to the ground and RoboSimian reached the maximum time limit.

JPL approached the *debris* task differently than the other teams. RoboSimian managed to remove three pieces of wood manually from the initial setup and placed them at the side of the doorway. In placing the pieces at the side, one piece unfortunately did not fully clear the doorway. RoboSimian reattempted to clear that piece of wood and succeeded. In order to pass through the doorway, RoboSimian had to remove the truss. RoboSimian did so by grasping a member of the truss while driving and turning backwards so that the truss was no longer in the pathway to the door. As the truss was removed so were additional pieces of wood. The remaining pieces of wood on the ground were then grasped and/or dragged forward past the threshold of the door. RoboSimian managed to complete the task with 4 points and 0 interventions.

The *door* task proved to be an interesting challenge. The first door (the push door) provided RoboSimian with the most difficulty. The door handles were lever style and required a full 90° of rotation. As a result, this caused a finger in our hand to break. While our behaviors are parametrized and use force control, our handle placement in the virtual world was slightly off causing RoboSimian to not rotate the full 90°. We attempted to further rotate, but with the door handle still misaligned, which caused the finger tendon that had the most load to snap. From our imagery, the operators thought the hand was non-functional. Knowing that we only had one working hand left and two doors left to open, the operators used the end effector to rotate the handle by simply contacting the door and sliding the end effector down. As the end effector maintained contact with the door, the door naturally opened when the door lock tongue cleared.

On the next door (a pull door), RoboSimian was able to open it just enough to put its lower limb in the opening to pry the door fully open. While the weather, especially the wind, proved fatal to other teams, the wind aided us by opening the door even more so that we could use our lower limbs. RoboSimian was able to only open 2 doors within the 30 minutes and with 0 interventions.

The *wall* task was immediately after the *door* task. As a result, the team had only 15 minutes to repair the broken hand and to verify that the other hand was functioning normally. As a result, when we attempted to pickup the tool for cutting, our grasp was less than solid and unfortunately we dropped the tool. Our team took its only intervention to replace the tool and to investigate the finger. One finger needed a slight repair and RoboSimian continued the task. RoboSimian was able to grasp the tool. The tool was turned on by using an appendage on RoboSimian that made contact with the trigger button on the tool. The operators made end-effector Cartesian moves so that the appendage made contact with this button. The operators were able to know when the tool was turned by analyzing the power spectral density of the FFT of the force-torque sensor readings. RoboSimian was then able to place the tip of the cutting tool on the top-most corner of the triangle. As it tried to insert the drill into the wall, the tool slipped out of RoboSimian's hand. Time had elapsed and RoboSimian did not receive any points on the *wall* task.

4.2 Other Results

We have used RoboSimian's unified mobile manipulation planner to walk in very different postures, over flat and rough terrain, climb up on a wall, walk while dragging an object, balance while manipulating, manipulate a variety of objects, and switch from standing to sitting and back to standing.

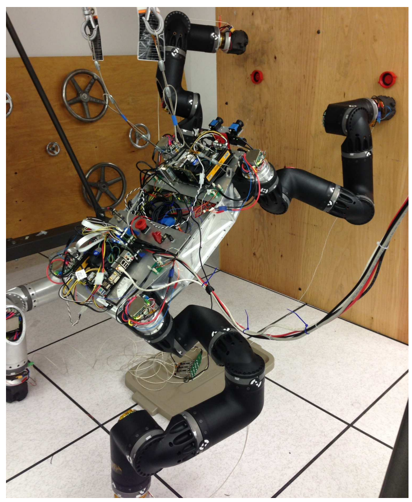


Figure 17: RoboSimian attempting to climb onto a wall.

The planner computes motions for manipulation behavior sequences, such as turning a valve or opening a door, automatically shifting the body to balance when necessary. The robot can walk in a narrow upright posture, like a horse, or a low sprawled posture, like a spider, or any other posture, by simply changing the reference posture. It can walk blindly over rough terrain when necessary, and will automatically adjust its posture and react to unexpected obstacles. It can climb up a wall by simply changing the desired body pose to be upright, shown in Figure 17. The planner was used with manual hold selection to demonstrate climbing up a ladder, but was not fielded due to time constraints. The flexibility of the planner allowed adding useful capabilities, such as taking a step forward with a back foot to create a tripod stance when lifting a front limb for manipulation, or not lifting one limb when walking while holding an object so it would be dragged on the ground.

Transitioning from a standing posture to a sitting posture is a sequence of desired body poses and limb positions. First, the behavior planner is used to find a body posture that is pitched up with the wheels in contact with the groundplane. Then, it computes the lower limb positions so that the caster wheels are in contact with the groundplane in front of the robot. And finally, the upper limbs are raised into a user specified posture. Transitioning to standing is the opposite procedure, with the front limbs placed into contact with the ground plane in front of the robot, the back limbs placed in contact behind the robot, and a then the body placed parallel to the groundplane with an adjustable height. Because each motion is automatically computed and collision free motion planning used for each motion, the sequence is robust to different starting conditions, terrains, and objects in the environment.

4.3 Discussion & Lessons Learned

The trials allowed the teams to experiment with different approaches both in hardware and in software. The JPL RoboSimian team learned many important lessons for fielding a fully functioning, built-from-scratch robot.

Other than the inevitable list of features that would have been useful but for which there was not sufficient time for implementation, there were a few elements of the RoboSimian hardware design that did not function as expected. We expect to make hardware improvements in the hand design. In particular, the hand, while capable of relatively high ultimate grasp strength, was challenged by any applied moments. This weakness was particularly evident in the wall cutting task, in which the side loads during cutting quickly worked the tool out of the hand's grasp. Another area of interest is the layout of the many stereo camera pairs. While the protected location of the cameras makes for a physically robust system, the limbs tended to obscure the cameras while performing manipulation operations. The solution to this issue may come from the integration of the remaining stereo pairs (only 3 pairs were integrated out of the possible 10 pairs), but we may also work to reintroduce a camera system into the hand design.

In terms of mobility, four-legged motion planning and mobility was not a trivial task, especially with the number of degrees of freedom of RoboSimian. While our approach was to remain statically stable, the complexity of walking still proved to be initially difficult. There were considerable intricacies to enable walking robustly for the terrain course of the DRC. Walking faster and planning to avoid configurations which prevent RoboSimian from taking another step are two tasks that still require more work and effort.

While we always kept three limbs in contact with the ground in order to be statically stable, this approach introduced the problem of walking while manipulating or grasping an object in our fourth hand. The original intention was to either walk bipedally or dynamically with three limbs. Given the amount of time to the Trials, those two approaches were not explored fully. Two other approaches were also considered that involved being statically stable with four limbs. The first was to walk on the manipulating limb but on the second to last link (on a elbow cap) as opposed to the end-effector. The other approach was to simply walk on the knuckles of the fingers. Unfortunately the hand design was not robust enough to support such actions. The final solution for the Trials ended up being wheeled mobility via the differential drive for mobile manipulation. In fact from an algorithmic perspective, this solution ended up being the most robust and easiest solution for mobile manipulation. As a result, RoboSimian started in this upright-wheeled drive configuration for all manipulation tasks at the Trials . This allowed the team to focus on the manipulation algorithms rather than on mobility algorithms for manipulating. The switch to the differential drive accelerated the development of RoboSimian dramatically and allowed us to perform and be competitive at the Trials.

The **plan** process ran on the high-brain computer of the robot. This was originally decided as the planner was continually replanning, which required fast access to state data and communication with the **ctrl** process. While this was primarily useful for limbed mobility, manipulation plans often did not require replanning. Due to the communication limits imposed, manipulation plans took time to be sent to the remote station. In cases when produced plans were not trusted by the operator, additional plans were requested, and the operation time increased unnecessarily. In the future, we intend to have two instances of the **plan** process – one running on the robot and one running on the remote side. This allows for requested plans that require replanning to be run on the robot and those that do not require replanning to be processes on the remote side. This is particularly beneficial for behavior and manipulation plans that can be computed quickly reducing operation time. The **plan** instance on the robot side could solely be a mobility planner and the remote side planner could be a manipulation planner.

The RoboSimian software had some notion of objects but it was primarily used for the operators to assign objects into the virtual world displayed in the remote interface. These objects aided in composing behaviors. Objects were not part of the autonomy, i.e. the state of objects were not persistent, grasp points were not selected based on the object, and the object location was not estimated by the autonomy software. In the next revision, the notion and use of object models will be much more pervasive throughout the autonomy software similar to our other work (Hudson et al., 2012), (Hudson et al., 2014) and not just used as a remote operator tool. The use of object models will reduce the operator time and make plans safer by checking collisions between the world and the object model.

Acknowledgments

The research described in this publication was carried out at the Jet Propulsion Laboratory, California Institute of Technology, with funding from the DARPA Robotics Challenge Track A 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.

Bajracharya, M., Ma, J., Howard, A., and Matthies, L. (2012). Real-time 3d stereo mapping in complex dynamic environments. In *International Conference on Robotics and Automation - Semantic Mapping, Perception, and Exploration (SPME) Workshop, 2012*, Shanghai, China.

Gennery, D. (2006). Generalized camera calibration including fish-eye lenses. *International Journal of Computer Vision*.

Hudson, N., Howard, T., Ma, J., Jain, A., Bajracharya, M., Myint, S., Kuo, C., Matthies, L., Backes, P., Hebert, P., Fuchs, T., and Burdick, J. (2012). End-to-end dexterous manipulation with deliberate interactive estimation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2371 –2378.

Hudson, N., Ma, J., Hebert, P., Jain, A., Bajracharya, M., Allen, T., Sharan, R., Horowitz, M., Kuo, C., Howard, T., et al. (2014). Model-based autonomous system for performing dexterous, human-level manipulation tasks. *Autonomous Robots*, 36(1-2):31–49.

Ju, T., Losasso, F., Schaefer, S., and Warren, J. (2002). Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346.

Kalakrishnan, M., Pastor, P., Righetti, L., and Schaal, S. (2013). learning objective functions for manipulation. In *ieee international conference on robotics and automation*.

Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *COMPUTER GRAPHICS*, 21(4):163–169.

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

Poston, T., Wong, T.-T., and Heng, P.-A. (1998). Multiresolution isosurface extraction with adaptive skeleton climbing. *Comput. Graph. Forum*, 17(3):137–148.