View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

N94-11577

SUPERVISED AUTONOMOUS CONTROL, SHARED CONTROL, AND TELEOPERATION FOR SPACE SERVICING

Paul G. Backes
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

ABSTRACT

A local-remote telerobot system for single and dual-arm supervised autonomy, shared control, and teleoperation has been demonstrated. The system is composed of two distinct parts: the local site, where the operator resides, and the remote site, where the robots reside. The system could be further separated into dual local sites communicating with a common remote site. This is valuable for potential space missions where a space based robotic system may be controlled either by a space based operator or by a ground based operator. Also, multiple modes of control integrated into a common system is valuable for satisfying different servicing scenarios. The remote site single arm control system is described and its parameterization for different supervised autonomous control, shared control, and teleoperation tasks are given. Experimental results are also given for selected tasks. The tasks include compliant grasping, orbital replacement unit changeout, bolt seating and turning, electronics card removal and insertion, and door opening.

I. INTRODUCTION

Supervised autonomous control, shared control, and teleoperation may be utilized for Space Station Freedom robotics applications. In teleoperation, trajectory points generated by an operator's motion of a hand controller are continuously sampled and communicated to a robot to track. In supervised autonomous control, autonomous commands are generated and then sent for execution on the robot. Trajectories are generated autonomously by specifying segment endpoints and trajectory parameters. The autonomous commands can be saved, simulated, and/or modified before sending them to the remote site for execution. Shared control is the merging of autonomous and teleoperation control. For example, the operator could specify the trajectory with the hand controller and the autonomous system could control the contact forces with the environment.

The planned baseline telerobotics capability for the Space Station is teleoperation with a Space Station based operator. Supervised autonomous control and shared control could provide valuable additional capability. Space Station based control, where the operator resides on the Space Station, could utilize supervised autonomy, shared control, or teleoperation. For ground (Earth) based control, there is expected to be an approximately 8 second round trip time delay for commands to the Space Station. Laboratory experiments indicate that time-delayed ground based control of Space Station robots can be safely achieved using supervised autonomous control. With such a system there would be dual local sites, one on Earth and one on the Space Station, communicating with a common remote site.

The basic architecture of the system provides a remote site capability with simultaneous multiple sensor based control and a local site capability which can generate commands and parameterization to send to the remote site. The Generalized Compliant Motion with Shared Control (GCMSC) [1] task primitive provides the remotes site system multi-sensor based control. The User Macro Interface (UMI) [2, 3] provides the local site task description and command sequencing. The utilization of sensors, both real and virtual, enhances task execution capability both by providing alternative approaches for executing a task and by making task execution more robust. A very simple

robotic system might have purely position control of a robot from a trajectory generator. Adding a hand controller allows the operator to perform position teleoperation. A force-torque sensor makes force/compliance control possible and therefore robust contact tasks. A virtual force field sensor can aid the operator during teleoperation to keep the robot away from joint limits and objects.

A task execution primitive is a function which controls a manipulator to perform the task described by its input parameter set. It generates the desired setpoints and performs the required control. The parameter list is the interface between a higher level task planning system and task execution. The planning system only needs to know how to describe the desired behavior of execution by setting the input parameters of the task primitive.

The paper will focus on the remote site GCMSC control and parameterization for specific tasks as well as give experimental results. The GCMSC primitive [1] and UMI [2, 3] have been described in previous publications. The paper is organized as follows. Section II discusses the input parameter set of the primitive and section III describes the control architecture. Motion control is described in section IV, monitoring and status reporting in section V, and command results in section VI. Section VII describes the implementation environment and section VIII discusses specific task parameterization and gives experimental results. Section IX describes new developments which extend the technology. Section X gives conclusions.

II. INPUT PARAMETER SET

The input parameter set is composed of five parameter types: system, trajectory, fusion, sensor, and monitor. Sensors generally have control and monitoring parameters. The addition of a sensor would normally require the addition of sensor and monitor input parameters for that sensor. The parameters are described throughout the remainder of the paper and are printed in bold letters.

III. CONTROL ARCHITECTURE OF THE PRIMITIVE

The GCMSC primitive provides six sources of robot motion which can be used individually or simultaneously. These sources of motion have two basic types: nominal motion trajectory generator and sensor based motion. The trajectory generator provides a feedforward Cartesian nominal position X_d of the NOM frame. Each of the sensors provides a perturbation to the nominal position of the NOM frame and these are all merged at the current NOM frame and the result is integrated with the past cumulative sensor based motion. The virtual restoration springs motion takes the integrated cumulative sensor based motion and tries to reduce it. The Generalized Compliant Motion control architecture is similar to position based impedance control [4, 5, 6, 7, 8].

The motion is programmed using the following kinematic ring equation.

$$trBase \cdot trTn \cdot trNom \cdot trDel \cdot trDrive = trBase \cdot trTnDest \cdot trNom \quad (1)$$

The WORLD frame is a fixed coordinate frame. **trBase** is the constant transform from the WORLD frame to a frame fixed in the manipulator's fixed first link, BASE. *trTn* is the variable transform from BASE to a frame fixed in the terminal link of the manipulator, TN. This transform changes each sample interval during control and is computed based on the results of all other inputs. **trNom** is the constant transform from the TN frame to the frame in which Cartesian interpolated motion will occur, NOM. *trDel* is the variable transform which has the integration of all sensor based motion. *trDrive* is the variable transform which provides Cartesian interpolated motion [9]. This transform is initially computed to satisfy the initial conditions of the transforms in the ring equation and is interpolated to the identity transform at the end of the nominal motion. **trTnDest** is the constant transform used to specify the nominal destination of the TN frame (is the expected value of the *trTn* transform at the end of nominal motion). At each sample interval, the trajectory generator calculates *trDrive*, sensor based motion calculates *trDel*, and then *trTn* is computed by solving equation 1. Inverse kinematics computes

the joint angles equivalent to $trTn$ and the robot controller servos the manipulator to these joint angles.

Most of the sources of input can specify their inputs in a coordinate frame specific to their functionality; nominal motion in NOM, teleoperation in TELEOP, force control in FORCE, etc. This is useful because the inputs may be most effectively specified in separate frames. For example, see the door opening task in Section VIII.

There are two time segments of motion during the execution of GCMSC: the nominal motion segment and the ending motion segment. When the primitive starts, it executes the nominal motion segment with the specified Cartesian interpolated motion and all other sensors. Motion stops if a monitor event is triggered or Cartesian interpolated motion completes. If the nominal motion segment completes normally, then the end motion segment begins. Exactly the same control occurs except there is no Cartesian interpolated motion; only the sensor based motion is active. But, whereas during the nominal motion segment the termination conditions were not being tested, they are tested during the ending motion and the motion can stop on a monitor event, time, or a termination condition. The ending motion is needed after the nominal motion segment to relax forces built up due to the nominal motion. Also, testing for ending conditions may not be desired until the nominal task is complete.

IV. MOTION CONTROL

The general architecture for control in GCMSC has been described above. The control for the individual inputs will be described in this section.

IV.A. Trajectory Generator

Trajectory generation is done utilizing the RCCL [10] trajectory generator. The $trDrive$ transform is initially given by

$$trDrive = (trTnInit \cdot trNom)^{-1} \cdot trTnDest \cdot trNom \quad (2)$$

where $trTnInit$ is the initial value of $trTn$. $trDrive$ is then linearly interpolated from this initial value to the identity transform at the end of the motion. The interpolation is controlled by the input parameters **timeVelSel**, **timeVelVal**, and **accTime**. **timeVelSel** selects whether to finish the motion in a specified time or with a specified velocity. **timeVelVal** is the time or velocity to execute the motion in. **accTime** is the time to ramp up to maximum velocity.

IV.B. Force Control

Force control is implemented independently in each degree of freedom of the Cartesian force control frame FORCE. The control modifies the position setpoint to control the forces [11, 12]. The result of force control each sample interval is the perturbation transform $trDelFc$. The first step of force control during a sample interval is the projection of forces[1] from the force-torque sensor frame to the SENSE frame (**trSense** is the transform from the NOM frame to the SENSE frame). A 6 DOF wrist force-torque sensor supplies forces and torques along and about the axes of the SENSOR frame centered in the force sensor. These are then projected to equivalent forces in the TN frame using rigid body force transformations. The load (the complete composite body beyond the force sensor) forces due to gravity are then computed. The mass and center of mass of the load with respect to the TN frame are given in the **massProp** input parameter. The current TN frame orientation with respect to the gravity vector is used with the load mass properties to determine the gravity load forces in the TN frame. These are then subtracted from the total sensed forces in the TN frame. The resulting forces and torques are those due only to contact and are then projected to the SENSE frame. The forces in the SENSE frame are then passed through a filter which reduces their magnitude by the values in the input vector parameter **deadZone** (if one of the force magnitudes is initially less than the **deadZone** value, then it is set to zero). The deadZone filter is useful to reduce drift due to inaccuracies in the mass properties of the load.

Force control is calculated in the FORCE frame using the forces projected into the SENSE

[1] in this paper the term forces generally implies a 6 vector of both forces and torques

frame. (**trForce** is the transform from the NOM frame to the FORCE frame). The FORCE and SENSE frames will usually coincide but there are cases where they may be different, such as leveling a plate on a surface where the SENSE frame is at the center of the plate and the FORCE frame is at the point of contact. If the SENSE and FORCE frames were both at the point of contact, then no moments would be felt and therefore no rotation due to force control would occur since the force line of action would be through the control frame.

The **selVectFc** selection vector selects which of the 6 DOF of the FORCE frame are to have force control. In these degrees of freedom, the contact forces which were projected from the TN frame to the SENSE frame are subtracted from the six setpoints in the **forceSetpoints** vector input parameter. The resulting force errors are then multiplied by the constants in the **forceGains** vector input parameter to produce a differential motion vector of six perturbations in the FORCE frame, three translations and three rotations given by

$$\vec{d}_f = (d_{fx}, d_{fy}, d_{fz}, \delta_{fx}, \delta_{fy}, \delta_{fz}) \quad (3)$$

The magnitudes of the elements of the $\vec{d}_f$ vector are then limited. The maximum magnitudes of the $\vec{d}_f$ perturbations per sample interval are the velocity limits given in the **maxForceVel** input parameter multiplied by the sample interval.

The ${}^{FORCE}trDelFc$ transform is a differential translation and rotation transform with elements given by $\vec{d}_f$ [9]. The $trDelFc$ transform is then transformed to the NOM frame. $trDelFc$ with respect to the FORCE and NOM frame are related by the following equation.

$${}^{NOM}trDelFc \cdot trForce = trForce \; {}^{FORCE}trDelFc \quad (4)$$

The $trDel$ transform of equation 1 is then updated with the perturbation due to force control with

$$trDel = {}^{NOM}trDelFc \cdot trDel \quad (5)$$

Premultiplication is required rather than postmultiplication because the motion is with respect to the NOM frame.

IV.C. Dither Sensor Control

Dither signals can be used to perturb the motion independently in each degree of freedom of the DITHER frame. Presently only a triangular waveform is available although other waveforms will be implemented such as sinusoidal and square. Dither is useful to overcome stiction, e.g., when pulling a pin out of a hole. The magnitude and period of the dither waveforms for each DOF of the DITHER frame are given in the input parameters **ditherMag** and **ditherPeriod**. As with force control, the inputs in each degree of freedom are elements of a differential translation and rotation transform, $trDelDt$ which is transformed to the NOM frame in the same manner as for $trDelFc$. The $trDel$ transform then updated with the perturbation due to the dither waveforms with

$$trDel = {}^{NOM}trDelDt \cdot trDel \quad (6)$$

IV.D. Teleoperation Sensor Control

The teleoperation sensor is actually a 6 DOF hand controller. Each sample interval the change in joint angles of the hand controller are read and put in a differential vector. This vector is multiplied by the hand controller Jacobian to get the input Cartesian motion perturbations. The appropriate Jacobian is used depending on the **teleMode** parameter to compute the Cartesian motion with respect the the hand controller grip which would be tool mode teleoperation, or with respect to a frame fixed with respect to the hand controller base, which would be used for world or camera mode teleoperation. These perturbations are then transformed to the TELEOP frame which is given with respect to the NOM frame by the input parameter **trTeleop**. Again, the mode determines how the perturbations are transformed to the TELEOP frame. **trCamera** is used for camera mode teleop to specify the present operator viewing orientation. The details of the various modes of teleoperation are explained in [13]. The **selVectTp** selection vector selects which degrees of freedom of teleoperation inputs to include and **teleGains** are weightings for the inputs. The **maxTelVel** limits the rate of teleoperation inputs.

Force reflection is also available in the system. The robot contact forces are sent to the hand controller where they are reflected to forces felt by the operator at the hand grip. Force reflection was not used during the tasks in Section VIII.

IV.E. Joint Sensor Control

The joint sensor control provides joint limiting. This prevents the arm from going into a joint limit or singularity. Joint angle perturbations for all the joints are computed and put into a differential vector. A joint angle perturbation is computed with

$$\Delta\theta = K_\theta(\theta_{actual} - \theta_{limit})^{-1} \quad (7)$$

where K_θ is the gain, θ_{actual} is the actual joint angle, and θ_{limit} is the limit that the joint is approaching, either as a joint limit or singularity. The differential vector is multiplied by the Jacobian to get the required Cartesian motion. This is transformed to the NOM frame and added to *trDel* as is the case with the other previous sensors.

IV.F. Virtual Restoration Springs Control

The virtual restoration springs act on the *trDel* transform to pull it towards the identity transform. This reduces the accumulated motion due to sensory inputs and causes the actual motion to approach the nominal motion. Virtual springs are applied in the DOFs specified by the **selVectSp** input parameter. Four virtual springs are used, one along each translational degree of freedom and one orientational spring. For the translational DOFs, the spring lengths are equal to the displacement vector, $\vec{p}$, elements of the *trDel* transform (*trDel* is a homogeneous transform with column vectors $\hat{n}$, $\hat{o}$, $\hat{a}$, and $\vec{p}$). The translational perturbations due to the virtual springs, $\vec{d}_s$, are then the spring lengths multiplied by the translational spring gains in the **springGains** vector, $\vec{k}_s$, input paramter, i.e., $d_{sx} = -k_{sx}p_x$, $d_{sy} = -k_{sy}p_y$, and $d_{sz} = -k_{sz}p_z$.

Virtual springs for orientation is applied about one axis with respect to the NOM frame. The selection of this axis depends upon the number of orientation degrees of freedom specified in **selVectSp**. The axis is $\hat{u}$ and the angular displacement about this axis is θ. If all orientation DOFs are selected, then $\hat{u}$ is the equivalent axis of rotation of the *trDel* transform and θ is the equivalent angle about the axis. If no orientation DOFs are selected, then no orientation perturbation is applied due to virtual springs. If only one orientation DOF is selected, then the corresponding axis $\hat{x}$, $\hat{y}$, or $\hat{z}$ is aligned by the orientation virtual spring. $\hat{u}$ and θ are selected such that a rotation about $\hat{u}$ by θ will align the selected axis. The virtual springs orientation perturbation is then $\delta_{s\theta} = -k_{s\theta}\theta$. The four virtual springs perturbation magnitudes are then limited to the magnitudes given in the **maxSpringVel** vector input parameter as the force control perturbations were limited by the **maxForceVel** values. The *trDel* transform is then updated with the perturbations due to virtual springs with

$$\begin{aligned} trDel \; &= trans(\hat{x}, d_{sx}) \cdot trans(\hat{y}, d_{sy}) \cdot \\ &\quad trans(\hat{z}, d_{sz}) \cdot rot(\hat{u}, \delta_{s\theta}) \cdot trDel \end{aligned} \quad (8)$$

where $trans(\hat{v}, d)$ is a translation of d along the $\hat{v}$ axis and $rot(\hat{v}, \delta)$ is a rotation of δ about the $\hat{v}$ axis.

V. MONITORS

Various parameters are continuously monitored during execution. The magnitudes of the translational part of *trDel* and the equivalent rotation of the orientational part of *trDel* are compared against the input parameters **posThreshold** and **orientThreshold**. If the values grow larger than the thresholds, then the motion stops. Also, the vector magnitudes of the contact forces and torques in the FORCE frame are compared against **forceThreshold** and **torqueThreshold** and motion stops if one of them is larger than the threshold. If the distance to a joint limit or singularity is less than the angles in the **jSafetyLimit** input vector, then motion stops.

Anther monitor is the termination condition monitor. It is used during the end motion (see section III). The end motion continues until all of the specified termination conditions are satisfied or until the time limit given by the **endTime** input parameter is passed. The **select** input parameter is a bit mask which selects which termination conditions to test for. Any combination of termination conditions can be tested. All termination conditions relate to forces and torques in the SENSE frame or sensor based motion specified by the *trDel* transform. Each termination condition is calculated as a moving average of data sampled each 200 ms over a window of **testTime** ms. Satisfaction of a termination condition means that its magnitude is less than its associated input parameter limit. The **endTransErr** condition is the mag-

nitude of the *trDel* transform $\vec{p}$ vector including only the position degree of freedom components. The **endAngErr** condition is the magnitude of the virtual restoration springs angular displacement, θ, described above. The **endTransVel** and **endAngVel** parameters are the rate of change of the **endTransErr** and **endAngErr** conditions. The **endForceErr** and **endTorqueErr** parameters are the magnitudes of the force and torque error vectors in the SENSE frame including only the force controlled degrees of freedom. The **endForceVel** and **endTorqueVel** parameters are the rate of change of the **endForceErr** and **endTorqueErr** conditions.

During execution of the primitive, the system executive reports the status of execution to the local site system. The report includes information such as contact forces and joint angles.

VI. COMMAND RESULTS

Various possible causes for the motion to stop have been described above. When the motion stops, the cause is returned to the local site system along with the system status. Each possible cause of motion termination has a unique command result code.

VII. IMPLEMENTATION ENVIRONMENT

The remote site with the GCMSC primitive and the local site with UMI are operational in the JPL Supervisory Telerobotics (STELER) Laboratory running PUMA 560 manipulators with six DOF wrist force-torque sensors and servoed grippers. The GCMSC primitive was written in the C programming language using utilities from the robot control C library (RCCL) [10]. The manipulator control is multiple rate with the Cartesian level control of the GCMSC primitive at a different rate from the joint level servo control. Presently the Cartesian level control (all control associated with the GCMSC primitive including trajectory generator and sensor based motion) runs with a 10 ms sample interval and the joint servo control has a 1 ms sample interval. Details on the hardware configuration of the system can be found in [14].

VIII. RESULTS

Various tasks have been executed in the JPL STELER lab utilizing the User Macro Interface for task description and sequencing and Generalized Compliant Motion with Shared Control for task execution. These tasks include compliant grasp, orbital replacement unit removal and insertion, bolt seating and turning, electronics card removal and insertion, and door opening. The different tasks utilized different combinations of the six sources of motion. For each task below, only the mentioned motion sources were used. All distance units used below are mm, forces are Newtons (N), and torques are N-mm. The **forceGains** input vector translation gains units are mm/N and orientation gains units are deg/N-mm. The **maxForceVel** vector has translation units of mm/sec and orientation units of deg/sec. The **springGains** input vector has three translational gains with units mm/mm and an orientation gain with units deg/deg.

The compliant grasp task utilized force control to both level the grippers on the grapple lug and to adjust the position of the robot as the fingers closed. The **trForce** transform was selected so the FORCE frame was between the robot fingers. The **forceSetpoints** input vector was all zeroes except for a force of -10 N along Z. The **forceGains** input vector was (0.02, 0.02, 0.02, 0.00003, 0.00003, 0.00003). The compliant ungrasp task opened the gripper while using force control to null out contact forces and virtual springs to make sure the gripper would not drift. The **forceSetpoints** input vector was all zeroes. The **forceGains** input vector was the same as for the compliant grasp task. The **springGains** input vector was (0.007, 0.007, 0.007, 0.015).

The orbital replacement unit (ORU) removal task utilized force control to pull the ORU and attached pin out of the passive connector. The arm carrying the ORU is shown in figure 1. The **massProp** inputs were 4.87 kg at position vector (in mm) (-90.3, -4.5, 336.6) relative to the T6 frame. The **trForce** transform was a translation of 400 mm along the T6 Z axis. The **forceSetpoints** input vector was all zeroes except for a force of 15 N along Z. The **forceGains** input vector was (0.02, 0.02, 0.02, 0.00001, 0.00001, 0.00001). The **maxForceVel** input vector was (30, 30, 30, 5,

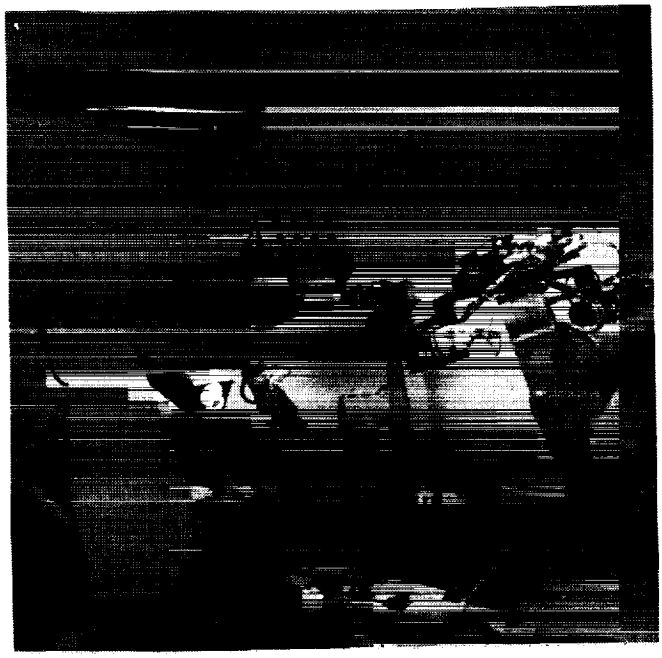

Figure 1: Manipulator carrying ORU

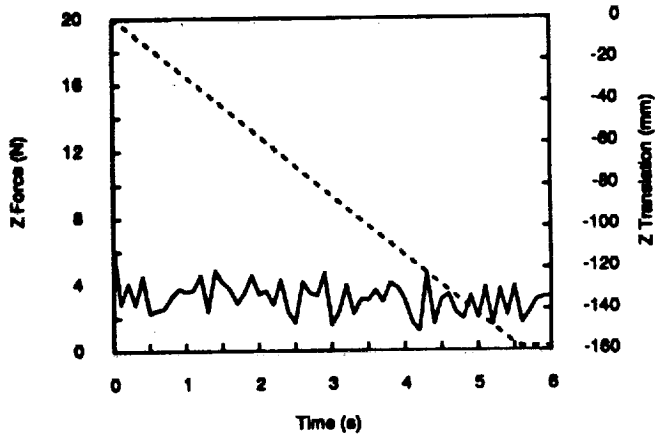

Figure 2: ORU removal task: solid is force along FORCE Z; dashed is translation along FORCE Z

5, 5). Figure 2 shows the force and displacement along the FORCE frame Z axis during the task. The figure shows that the **maxForceVel** of 30 mm/sec limited the velocity due to force control to 30 mm/sec so that the force could not reach its setpoint. The motion stopped on the position monitor with **posThreshold** input parameter of 160 mm.

The ORU insertion task used the same parameters as the oru removal task except that the task completed on the time monitor and the force setpoint along Z was -15 N. Figure 3 shows the force and displacement along the FORCE frame Z axis during the task.

The bolt seating task utilized Camera mode shared control teleopertion. The **teleMode** parameter specified Camera mode teleoperation. The **trTeleop** transform put the TELEOP frame on the socket shaft. The **forceGains** input vector was (0.03, 0.03, 0.03, 0.00003, 0.00003, 0.00003).

The bolt unscrew task used force control to cause the bolt to turn. The **trForce** transform was selected so that the FORCE frame was above the socket. The **forceSetpoints** input vector was (0, 0, -5, 0, 0, 6000). The **forceGains** input vector was (0.01, 0.01, 0.01, 0.00001, 0.00001, 0.00001).

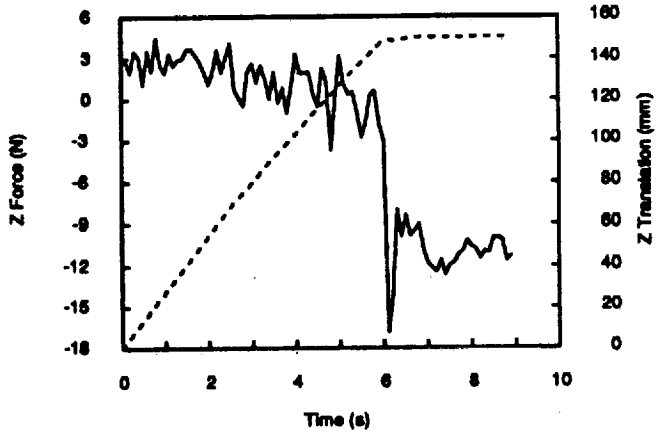

Figure 3: ORU insertion task: solid is force along FORCE Z; dashed is translation along FORCE Z

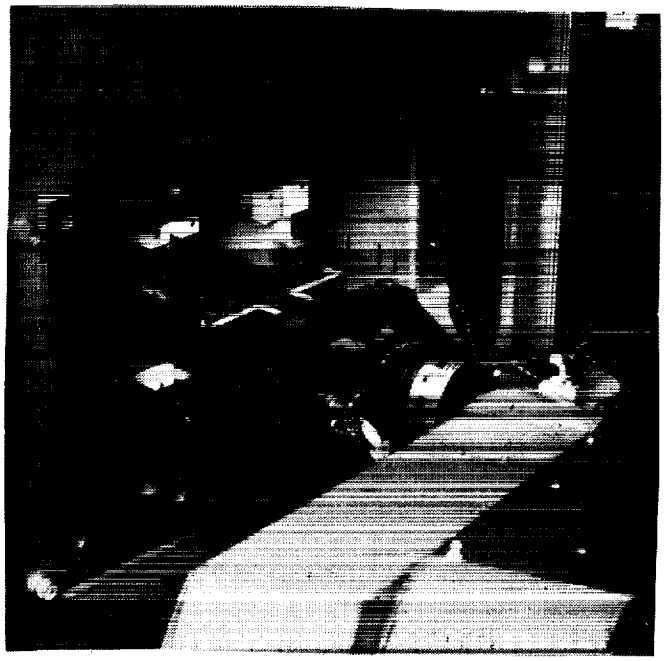

Figure 4: Electronics card insertion and removal

Figure 5: Operator at local site OCS

The -5 N force kept the socket on the bolt. The 6000 N-mm torque caused the bolt rotation. The **orientThreshold** input parameter of 90 degrees caused the task to terminate after the bolt rotated 90 degrees. The bolt screw task was the same except that a torque of -6000N-mm was used to screw the bolt on. The task terminate either on the **orientThreshold** of 90 degrees or on time if the bolt would not turn any more.

Four tasks were used for electronics card insertion and removal, as shown in figure 4. A real electronics card and chassis were used in the experiment. The first task was camera mode shared control teleoperation where the operator used the hand controller to partially insert the card into the card slot. The operator at the local site operator control station is shown in figure 5. Camera mode teleoperation caused the robot to move in the same direction relative to the cameras mounted on the camera arm (see figure 1) as the operator's hand moved relative to the stereo display monitor. Force control with zero setpoints was used to null out the contact forces between the card and the slot. The **teleMode** parameter specified Camera mode teleoperation. The **trTeleop** transform put the TELEOP frame on the electronics card. The **forceGains** input vector was (0.03, 0.03, 0.03, 0.00003, 0.00003, 0.00003).

Once the electronics card was successfully placed in the chassis slot, autonomous commands were used to slide the card to the backplane and seat it in the backplane. Sliding the card to the backplane was done using force control. The **forceSetpoints** input vector was (0, 0, -15, 0, 0, 0) and the **forceGains** input vector was (0.01, 0.01, 0.01, 0.00001, 0.00001, 0.00001). Figure 6 shows the translation and forces along the FORCE frame Z axis. A larger force is needed to seat the card in the backplane than was used to slide the card to the backplane. To seat the card in the backplane, the force along the FORCE Z axis was set to -60N and the same **forceGains** were used. The results are shown in figure 7. Unseating the electronics card from the backplane is achieved by applying a force of 60 N along the FORCE frame Z axis. After the card breaks free of the backplane, a velocity limiting filter limits the velocity using the **maxForceVel** input parameters (2.5, 2.5, 2.5, 3, 3, 3). The results are shown in figure 8.

Shared control was used for the dome cleaning task as shown in figure 9. A **trTeleop** transform of 290 mm along T6 Z was selected so that the TELEOP frame was in the middle of the pad. The operator was given three hand controller degrees of freedom of input - two tangential to the dome surface and one about the surface normal as specified

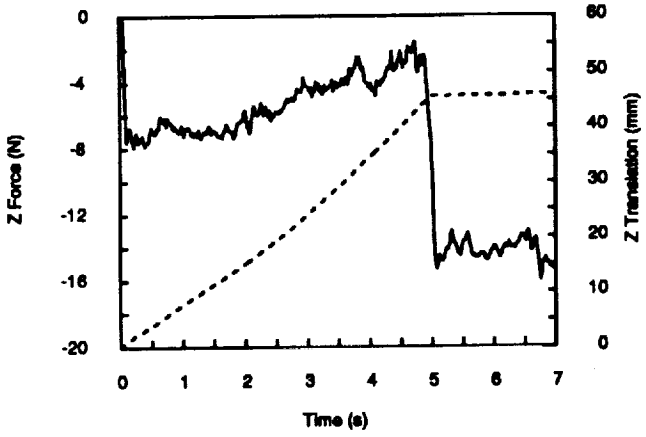

Figure 6: Electronics card sliding to the backplane: solid is force along FORCE Z; dashed is translation along FORCE Z

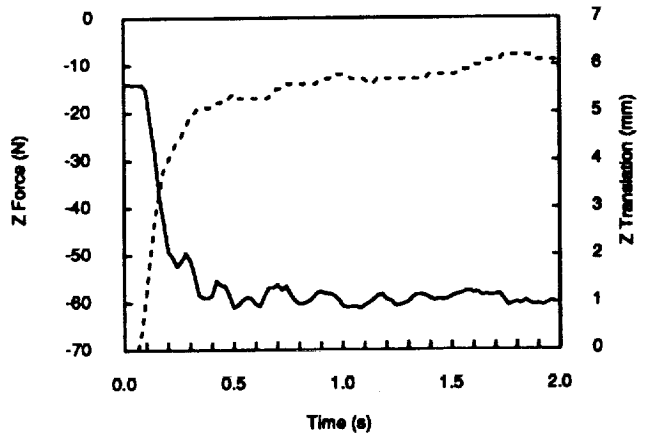

Figure 7: Electronics card seating in backplane: solid is force along FORCE Z; dashed is translation along FORCE Z

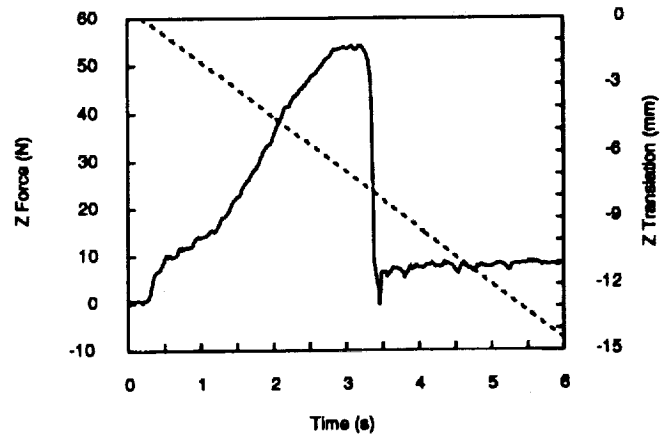

Figure 8: Electronics card unseating from backplane: solid is force along FORCE Z; dashed is translation along FORCE Z

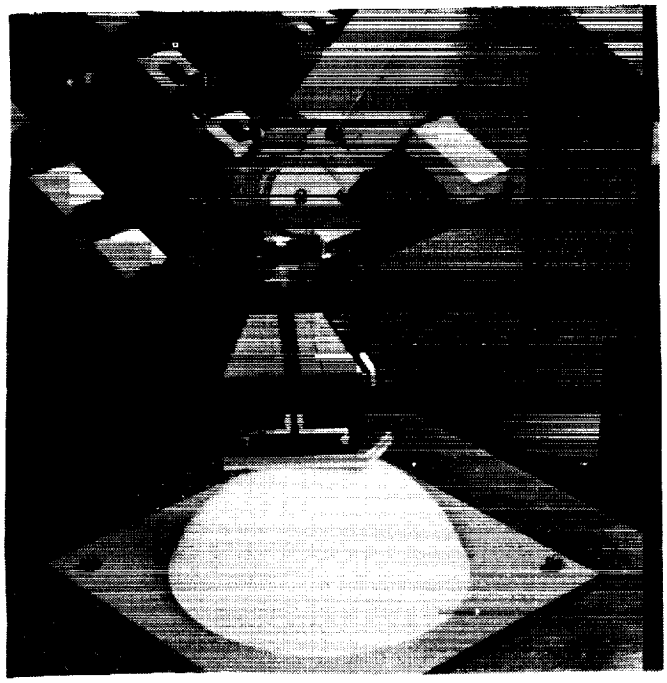

Figure 9: Dome cleaning task

in the **selVectTp** input vector (1, 1, 0, 0, 0, 1). The FORCE frame was the same as the TELEOP frame. The **forceSetpoints** input vector was (0, 0, -20, 0, 0, 0) and the **forceGains** input vector was (0.02, 0.02, 0.02, 0.00012, 0.00012, 0.00002). The 20 N force caused the pad to stay in contact with the curved surface. When the pad was moved so that the FORCE frame was not at the point of contact, then the 20 N would generate a moment and the pad would automatically rotate until the FORCE frame was again at the contact point. In this way, the operator could polish the dome surface but could not cause motion with the hand controller which would cause damage to the surface.

The last task is the door opening task which was done with both shared control teleoperation and autonomous control. The door task is shown in figure 10. For the door opening with teleoperation task, the **trTeleop** input transform was selected so that the TELEOP frame Z axis was along the hinge axis. The trForce input transform placed the FORCE frame at the knob where the robot was grasping the door. The **forceSetpoints** input vector was all zeros and the **forceGains** input vector was (0.015, 0.015, 0.015, 0.00002, 0.00002, 0.00002). The operator opened and closed the door simply by a one DOF rotation of the hand con-

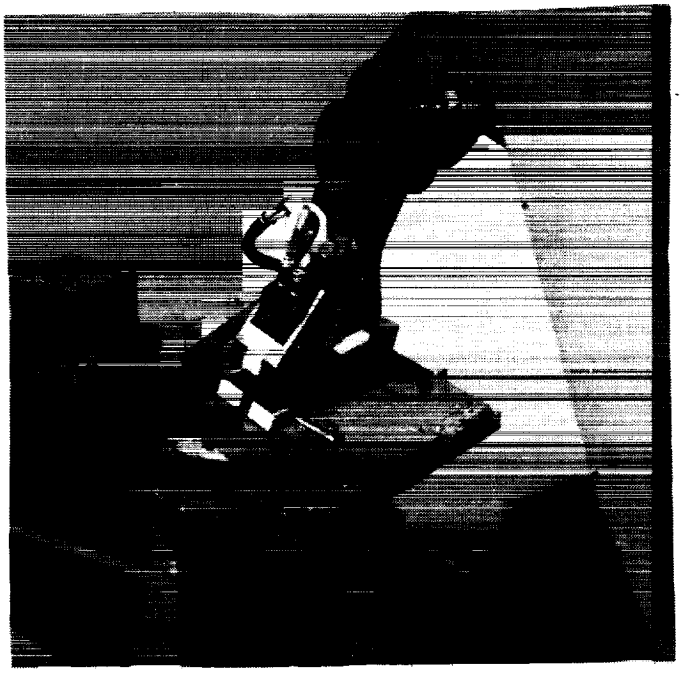

Figure 10: Door opening task

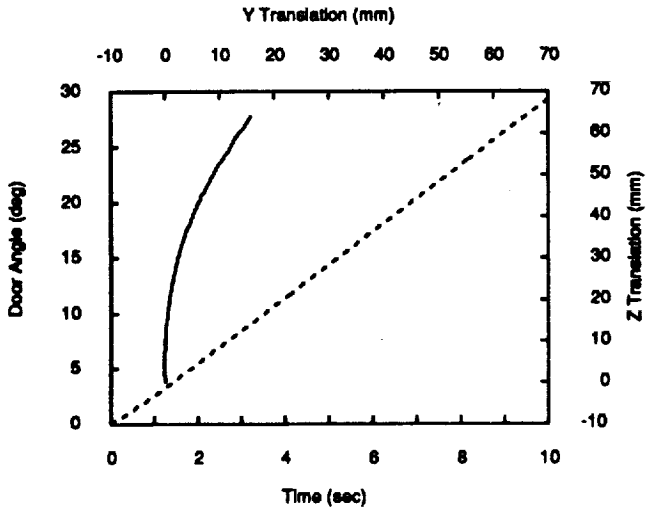

Figure 11: Autonomous door opening results: solid is motion of FORCE frame; dashed is rotation of NOM frame (hinge axis)

troller grip.

For the door opening with autonomous control task, the autonomous trajectory generator was used instead of teleoperation inputs to cause the nominal motion. The **trNom** input transform placed the NOM frame such that its Z axis was along the door hinge axis. The **forceSetpoints** and **forceGains** input vectors were the same as for the compliant teleoperation case above. Virtual springs were necessary so that the motion due to force control would not cause the actual motion to drift far from the reference nominal trajectory. The **springGains** input vector was set to (0.007 0.007 0.007 0.015). The **select** termination condition input was set to select the **endAngErr** as the termination condition to monitor; **endAngErr** was set to 0.1 deg. A relative autonomous motion was specified to rotate the NOM frame by 30 degrees. The results are shown in figures 11 and 12. The figures show that the door was successfully opened 30 degrees.

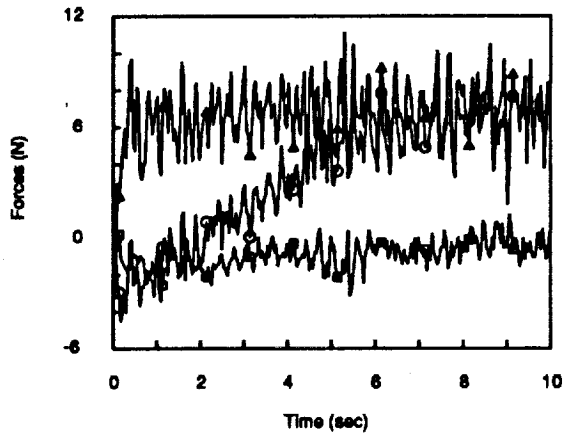

Figure 12: Autonomous door opening results: forces along FORCE frame X (□), Y(○), and Z(△) axes

The value of the virtual springs is shown by executing the same task but with the **springGains** input vector elements set to zero. The results are shown in figure 13. In this case the door opened a maximum of only 21.6 deg. The maximum rotation occurred when the trajectory gener-

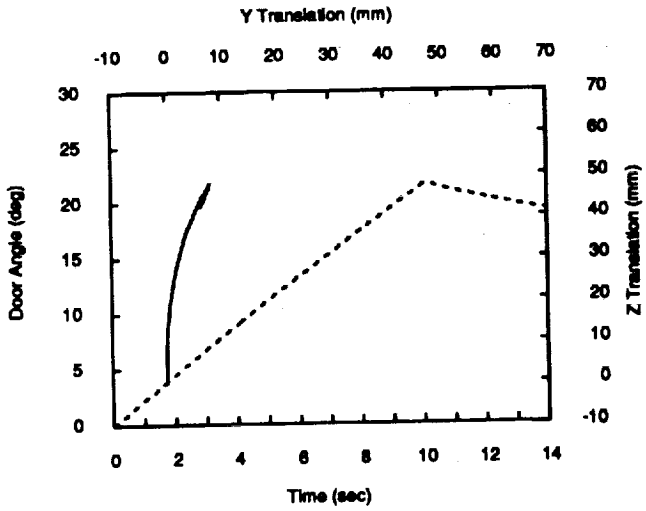

Figure 13: Autonomous door opening results (no virtual springs): solid is motion of FORCE frame; dashed is rotation of NOM frame (hinge axis)

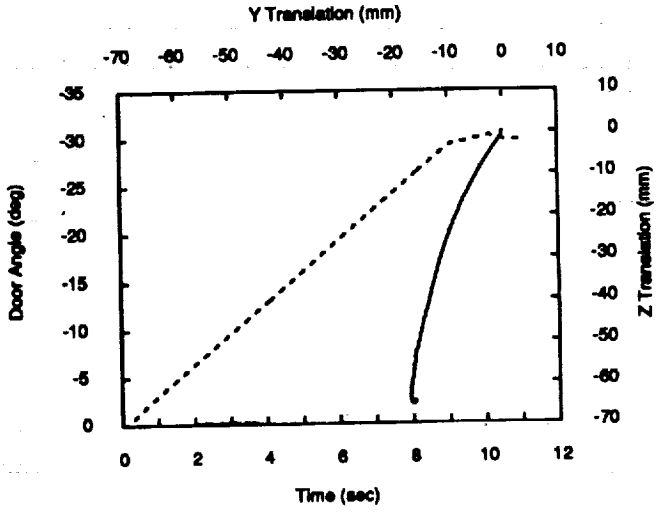

Figure 14: Autonomous door closing: solid is motion of FORCE frame; dashed is rotation of NOM frame (hinge axis)

ator finished. After that, the ending motion time segment began and the door slowly began closing due to its gravity weight. The ending condition of 0.1 deg. from the 30 deg. goal was never satisfied so it stopped on the **endTime** timeout. The reason that the door did not open all of the way is that the forces in the FORCE frame caused compliant motion to resist the nominal trajectory generator motion and there were no virtual springs to offset this motion.

The door opening task was followed by a door closing task. The same parameters as for the door opening task were used, including virtual springs, except that the nominal motion was negative 32 deg. and different termination conditions were used. A 32 deg. motion was used to be sure to have at least the 30 deg. of motion needed. The **select** termination condition input was set to select the **endTransVel** and **endAngVel** as the termination conditions to monitor; **endTransVel** was set to 1 mm/sec and **endAngVel** was set to 0.1 deg/sec. The results are shown in figures 14 and 15. The figures show that the door was successfully closed 30 degrees. The motion is nearly linear until the door makes contact and is closed at 30 deg. Then the rotation stops which triggers the termination condition.

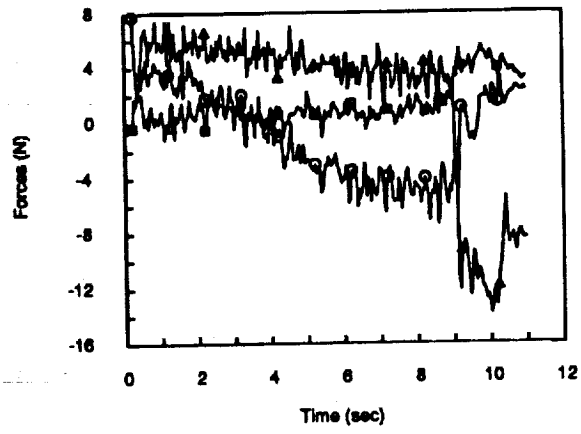

Figure 15: Autonomous door closing results: forces along FORCE frame X (□), Y(○), and Z(△) axes

IX. DUAL-ARM AND IMPEDANCE BASED REDUNDANT ARM CONTROL

The GCMSC primitive has been generalized for dual-arm cooperative control teleoperation, supervised autonomy, and shared control with the Dual-Arm Generalized Compliant Motion primitive [15]. It was then generalized for impedance based control of a six DOF manipulator [13] and then impedance based control of a redundant seven DOF manipulator [16].

X. CONCLUSIONS

A local-remote control system with unified autonomous control, shared control, and teleoperation has been described. The local site generates teleoperation and autonomous commands which are communicated to the remote site. The remote site uses a parameterized task primitive to execute tasks. The execution of various tasks in the laboratory demonstrates the capability of the system.

ACKNOWLEDGEMENTS

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

[1] Paul G. Backes. Generalized compliant motion with sensor fusion. In *Proceedings 1991 ICAR: Fifth International Conference on Advanced Robotics, Robots in Unstructured Environments*, pages 1281–1286, Pisa, Italy, June 19–22 1991.

[2] Paul G. Backes and Kam S. Tso. Umi: An interactive supervisory and shared control system for telerobotics. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1096–1101, Cincinnati, Ohio, May 1990.

[3] Paul G. Backes. Ground-remote control for space station telerobotics with time delay. In *Proceedings AAS Guidance and Control Conference*, Keystone, CO, February 8-12 1992. AAS paper No. 92-052.

[4] N. Hogan. Impedance control: An approach to manipulation: Part i — theory. *ASME Journal of Dynamic Systems, Measurement, and Control*, 107:1–7, March 1985.

[5] N. Hogan. Impedance control: An approach to manipulation: Part ii — implementation. *ASME Journal of Dynamic Systems, Measurement, and Control*, 107:8–16, March 1985.

[6] N. Hogan. Impedance control: An approach to manipulation: Part iii — applications. *ASME Journal of Dynamic Systems, Measurement, and Control*, 107:17–24, March 1985.

[7] Dale A. Lawrence and R. Michael Stoughton. Position-based impedance control: Achieving stability in practice. In *Proceedings AIAA Conference on Guidance, Navigation, and Control*, pages 221–226, Monterey, Ca, August 1987.

[8] Dale A. Lawrence. Impedance control stability properties in common implementation. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1185–1190, 1988.

[9] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* The MIT Press, 1981.

[10] J. Lloyd, M. Parker, and R. McClain. Extending the rccl programming environment to multiple robots and processors. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 465–474, 1988.

[11] Paul G. Backes and Kam S. Tso. Autonomous single arm oru changeout — strategies, control issues, and implementation. Jet Propulsion Laboratory Engineering Memorandum 347-88-258 (internal document). Published as BACK90F, November 22 1988.

[12] J. A. Maples and J. J. Becker. Experiments in force control of robotic manipulators. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 695–702, 1986.

[13] Paul G. Backes. Multi-sensor based impedance control for task execution. In *Proceedings IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[14] Samad Hayati, Thomas Lee, Kam Tso, and Paul G. Backes. A testbed for a unified teleoperated-autonomous dual-arm robotic system. In *Proceedings IEEE International Conference on Robotics and Automation*, 1990.

[15] Paul G. Backes. Dual-arm supervisory and shared control space servicing task experiments. In *Proceedings AIAA Space Programs and Technologies Conference*, Huntsville, AL, March 24-27 1992. AIAA paper No. 92-1677.

[16] Paul G. Backes and Mark K. Long. Redundant arm control in a supervisory and shared control system. In *Proceedings AIAA Space Programs and Technologies Conference*, Huntsville, AL, March 24-27 1992. AIAA paper No. 92-1578.