We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

1

Control of Cable Robots for Construction Applications

Alan Lytle, Fred Proctor and Kamel Saidi
National Institute of Standards and Technology
United States of America

1. Introduction

The Construction Metrology and Automation Group at the National Institute of Standards and Technology (NIST) is conducting research to provide standards, methodologies, and performance metrics that will assist the development of advanced systems to automate construction tasks. This research includes crane automation, advanced site metrology systems, laser-based 3D imaging, calibrated camera networks, construction object identification and tracking, and sensor integration and process control from Building Information Models. The NIST RoboCrane has factored into much of this research both as a robotics test platform and a sensor/target positioning apparatus. This chapter provides a brief review of the RoboCrane platform, an explanation of control algorithms including the NIST GoMotion controller, and a discussion of crane task decomposition using the Four Dimensional/Real-time Control System approach.

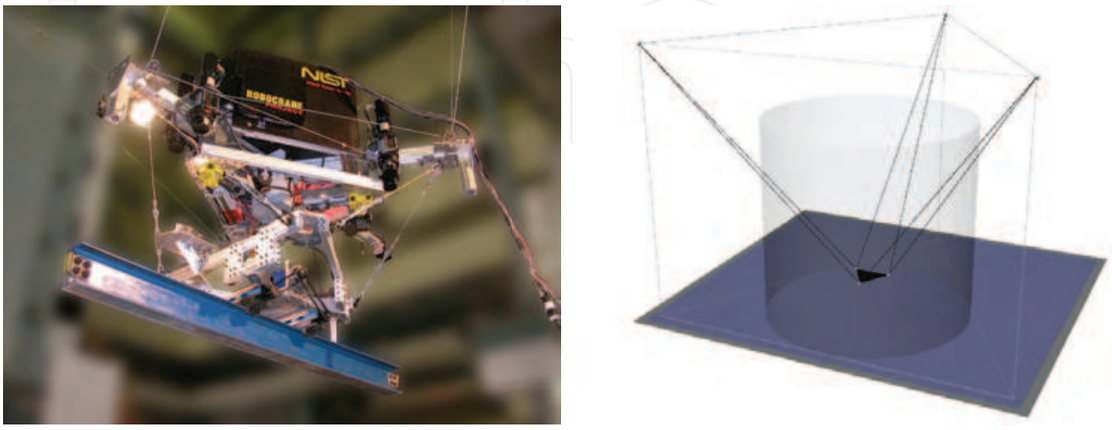
1.1 The NIST RoboCrane

RoboCrane was first developed by the NIST Manufacturing Engineering Laboratory's (MEL) Intelligent Systems Division (ISD) in the late 1980s as part of a Defense Advanced Research Project Agency (DARPA) contract to stabilize crane loads (Albus et al., 1992). The basic RoboCrane is a parallel kinematic machine actuated through a cable support system. The suspended moveable platform is kinematically constrained by maintaining tension due to gravity in all six support cables. The support cables terminate in pairs at three vertices attached to an overhead support. This arrangement provides enhanced load stability over beyond traditional lift systems and improved control of the position and orientation (pose) of the load. The suspended moveable platform and the overhead support typically form two opposing equilateral triangles, and are often referred to as the "lower triangle" and "upper triangle," respectively.

The version of RoboCrane used in this research is the Tetrahedral Robotic Apparatus (TETRA). In the TETRA configuration, all winches, amplifiers, and motor controllers are located on the moveable platform as opposed to the support structure. The upper triangle only provides the three tie points for the cables, allowing the device to be retrofitted to existing overhead lift mechanisms. Although the TETRA configuration is presented in this chapter, the control algorithms and the Four Dimensional/Real-time Control System (4D/RCS), for 3D + time/Real-time Control System, task decomposition are adaptable to

Source: Parallel Manipulators, Towards New Applications, Book edited by: Huapeng Wu, ISBN 978-3-902613-40-0, pp. 506, April 2008, I-Tech Education and Publishing, Vienna, Austria

many different crane configurations. The functional RoboCrane design can be extended and adapted for specialized applications including manufacturing, construction, hazardous waste remediation, aircraft paint stripping, and shipbuilding. Figure 1 depicts the RoboCrane TETRA configuration (a) and the representative work volume (b). Figure 2 shows additional retrofit configurations of the RoboCrane platform, and Figure 3 shows implementations for shipbuilding (Bostelman et al., 2002) and aircraft maintenance.



(a) (b)

Fig. 1. RoboCrane – TETRA configuration (a); Rendering of the RoboCrane environment. The shaded cylinder represents the nominal work volume (b).

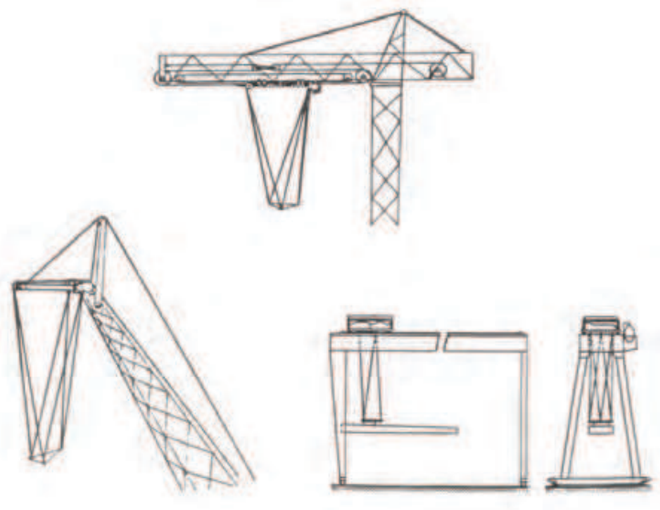


Fig. 2. Illustrations of RoboCrane in possible retrofitted configurations: Tower Crane (top), Boom Crane (lower left) and Gantry Bridge Crane (lower right).

1.2 Motivation for current research

Productivity gains in the U.S. construction sector have not kept pace with other industrial sectors such as manufacturing and transportation. These other industries have realized their productivity advances primarily through the integration of information, communication,

automation, and sensing technologies. The U.S. construction industry lags these other sectors in developing and adopting these critical, productivity-enhancing technologies. Leading industry groups, such as the Construction Industry Institute (CII), Construction Users Roundtable (CURT) and FIATECH, have identified the critical need for fully integrating and automating construction processes.
Robust field-automation on dynamic and cluttered construction sites will require advanced capabilities in construction equipment automation, site metrology, 3D imaging, construction object identification and tracking, data exchange, site status visualization, and design data integration for autonomous system behavior planning. The NIST Construction Metrology and Automation Group (CMAG) is conducting research to provide standards, methodologies, and performance metrics that will assist the development, integration, and evaluation of these technologies. Of particular interest are new technologies and capabilities for automated placement of construction components.

(a) (b)

Fig. 3. The NIST Flying Carpet – a platform for ship access in drydocks (a) and the NIST Aircraft Maintenance Project (AMP) – a platform for aircraft access in hangars (b).

2. RoboCrane kinematics

From (Albus et al., 1992), given an initial condition where the overhead support and the suspended platforms are represented by parallel, equilateral triangles with centers aligned along the vertical axis Z, (see Figure 4), the positions of the upper triangle with vertices A, B, and C and lower triangle with vertices D, E, and F are expressed as

$$\mathbf{A}=\begin{bmatrix} -b \\ -\frac{1}{3}b\sqrt{3} \\ -h \end{bmatrix} \quad \mathbf{B}=\begin{bmatrix} b \\ -\frac{1}{3}b\sqrt{3} \\ -h \end{bmatrix} \quad \mathbf{C}=\begin{bmatrix} 0 \\ \frac{2}{3}b\sqrt{3} \\ -h \end{bmatrix}$$
$$\mathbf{D}=\begin{bmatrix} 0 \\ -\frac{2}{3}a\sqrt{3} \\ 0 \end{bmatrix} \quad \mathbf{E}=\begin{bmatrix} a \\ \frac{1}{3}a\sqrt{3} \\ 0 \end{bmatrix} \quad \mathbf{F}=\begin{bmatrix} -a \\ \frac{1}{3}a\sqrt{3} \\ 0 \end{bmatrix} \tag{1}$$

With the positions of the vertices of triangles ABC and DEF as described in equations (1), when the lower platform is moved to a new position and orientation (D´E´F´) through a translation of

$$\mathbf{U} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{2}$$

and a rotation of

$$R_{yxz}(\gamma,\theta,\phi) = R_z(\phi)\cdot R_x(\theta)\cdot R_y(\gamma) \tag{3}$$

the cable lengths can be expressed as

$$\mathbf{L_1} = \begin{bmatrix} -b+\frac{2}{3}aQ_{12}\sqrt{3}-u_x \\ -\frac{1}{3}b\sqrt{3}+\frac{2}{3}aQ_{22}\sqrt{3}-u_y \\ -h+\frac{2}{3}aQ_{32}\sqrt{3}-u_z \end{bmatrix} \quad \mathbf{L_2} = \begin{bmatrix} -b+\frac{2}{3}aQ_{12}\sqrt{3}-u_x \\ -\frac{1}{3}b\sqrt{3}+\frac{2}{3}aQ_{22}\sqrt{3}-u_y \\ -h+\frac{2}{3}aQ_{32}\sqrt{3}-u_z \end{bmatrix}$$

$$\mathbf{L_3} = \begin{bmatrix} b-aQ_{11}-\frac{1}{3}aQ_{12}\sqrt{3}-u_x \\ -\frac{1}{3}b\sqrt{3}-aQ_{21}-\frac{1}{3}aQ_{22}\sqrt{3}-u_y \\ -h-aQ_{31}-\frac{1}{3}aQ_{32}\sqrt{3}-u_z \end{bmatrix} \quad \mathbf{L_4} = \begin{bmatrix} -aQ_{11}-\frac{1}{3}aQ_{12}\sqrt{3}-u_x \\ \frac{2}{3}b\sqrt{3}-aQ_{21}-\frac{1}{3}aQ_{22}\sqrt{3}-u_y \\ -h-aQ_{31}-\frac{1}{3}aQ_{32}\sqrt{3}-u_z \end{bmatrix} \tag{4}$$

$$\mathbf{L_5} = \begin{bmatrix} aQ_{11}-\frac{1}{3}aQ_{12}\sqrt{3}-u_x \\ \frac{2}{3}b\sqrt{3}+aQ_{21}-\frac{1}{3}aQ_{22}\sqrt{3}-u_y \\ -h+aQ_{31}-\frac{1}{3}aQ_{32}\sqrt{3}-u_z \end{bmatrix} \quad \mathbf{L_6} = \begin{bmatrix} -b+aQ_{11}-\frac{1}{3}aQ_{12}\sqrt{3}-u_x \\ -\frac{1}{3}b\sqrt{3}+aQ_{21}-\frac{1}{3}aQ_{22}\sqrt{3}-u_y \\ -h+aQ_{31}-\frac{1}{3}aQ_{32}\sqrt{3}-u_z \end{bmatrix}$$

where

$$\begin{array}{lll} \mathbf{L_1} = \mathbf{A}\text{ - }\mathbf{D'} & \mathbf{L_2} = \mathbf{B}\text{ - }\mathbf{D'} & \mathbf{L_3} = \mathbf{B}\text{ - }\mathbf{E'} \\ \mathbf{L_4} = \mathbf{C}\text{ - }\mathbf{E'} & \mathbf{L_5} = \mathbf{C}\text{ - }\mathbf{F'} & \mathbf{L_6} = \mathbf{A}\text{ - }\mathbf{F'} \end{array} \tag{5}$$

and $\mathbf{Q_{ij}}$ represents an element in the following rotation matrix:

$$\mathbf{Q} = \begin{bmatrix} \cos(\gamma)\cos(\phi)-\sin(\gamma)\sin(\theta)\sin(\phi) & -\cos(\theta)\sin(\phi) & \sin(\gamma)\cos(\phi)+\cos(\gamma)\sin(\theta)\sin(\phi) \\ \cos(\gamma)\sin(\phi)-\sin(\gamma)\sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) & \sin(\gamma)\sin(\phi)-\cos(\gamma)\sin(\theta)\cos(\phi) \\ -\sin(\gamma)\cos(\theta) & \sin(\theta) & \cos(\gamma)\cos(\theta) \end{bmatrix} \tag{6}$$

Therefore, for any new desired pose of the moving platform described by equations (2) and (3), the required cable lengths to achieve that pose can be calculated by the inverse kinematic equations shown in equations (4).

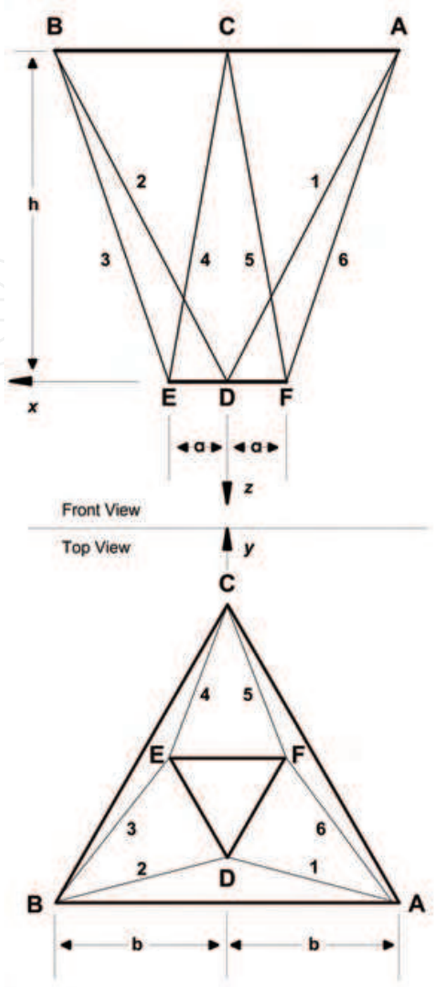


Fig. 4. Graphical representation of the RoboCrane cable support structure.

3. Measuring RoboCrane pose

The controller's estimate of the actual pose of RoboCrane differs from the actual pose due to several sources of error. Position feedback is provided through motor encoders that measure rotational position. Cable length is computed by multiplying the rotational position by the winch drum radius, with a suitable scale factor and offset.

However, the winch drum radius is not constant, but varies depending on the amount of cable that has already been wrapped around the drum, increasing its radius. It is possible to keep track of this and change the radius continually, by building a table that relates motor rotational position with effective radius.

Another source of error is that the cable length is affected by sag due to gravity. This sag depends on the pose of the platform and its load. Compensation can be achieved using an iterative process that begins with the nominal cable lengths, computes the platform pose using the forward kinematics equations, and determines the tensions on each of the cables using the transpose of the Jacobian matrix and the weight of the platform. The tensions can be used to generate the actual catenary curve of the cable, taking its nominal length as the

length of the hanging catenary curve. This process is repeated iteratively, with the nominal cable length as the fixed arc length of the catenary, and the chord between its endpoints as the continually revised length used by the forward kinematics.
Calibration errors in the mounting points of the ends of the cables further contribute to pose error. In practice these are not fixed points, but vary as the angles of the cables change the contact point to the pulleys or eye bolts that affix the ends. Even if these contact points were constant, their actual locations can be difficult to measure with precision, given their large displacement over a typical work volume.
Given these many sources of error, it is desirable to be able to measure the pose of the platform directly. There are many commercial systems for this purpose. An initial approach to external measurement implemented on RoboCrane uses a laser-based, large-scale, site measurement system (SMS).

3.1 The site measurement system (SMS)

A laser-based site measurement system (SMS) is used to track RoboCrane's pose and to measure object locations within the work volume. The SMS uses stationary, active-beacon laser transmitters and mobile receivers to provide millimeter-level position data at an update rate of approximately 20 Hz. This technology was chosen based upon a combination of factors including indoor/outdoor operation, accuracy, update-rate, and support for multiple receivers.
Each SMS transmitter emits two rotating, fanned laser beams and a timing pulse. Elevation is calculated from the time difference between fanned beam strikes. Azimuth is referenced from the timing pulse. The field of view of each transmitter is approximately 290° in azimuth and ± 30° in elevation/declination.
Similar to GPS, the SMS does not restrict the number of receivers. Line-of-sight to at least two transmitters must be maintained by each receiver in order to calculate that receiver's position. The optical receivers each track up to four transmitters and wirelessly transmit timing information to a base computer for position calculation.
For tracking RoboCrane's pose, four laser transmitters are positioned and calibrated on the work volume perimeter, and three SMS receivers are mounted on RoboCrane near the vertices of the lower triangle. The receiver locations are registered to the manipulator during an initial setup process in the local SMS coordinate frame. A transmitter and an optical receiver are shown in Figure 5. The SMS receivers mounted on RoboCrane are shown in Figure 6.

(a) (b)

Fig. 5. An SMS laser transmitter (a) and an SMS optical receiver (b).

The drawback of these systems is the added cost, and the need to maintain lines-of-sight between the platform and transmitters, potentially interfering with intended use. The benefits of accurate pose measurement are often significant enough to warrant their use.
In the first implementation of the SMS to track RoboCrane, position estimates were obtained at several stopping points during RoboCrane's trajectory, and these estimates were used as coarse correction factors for the encoder positions. Current work is focused on a dynamic tracking approach to eliminate the need for stopping points.

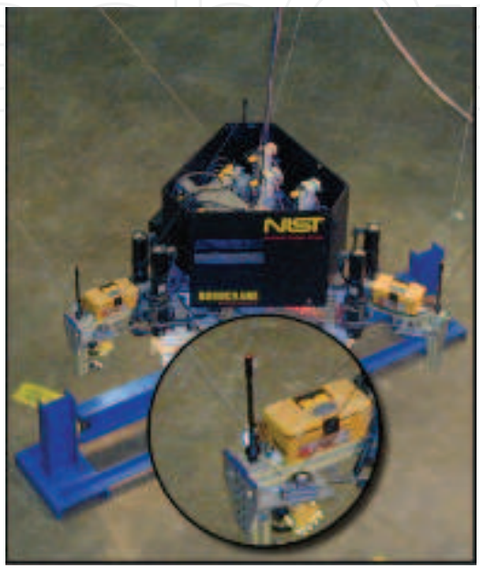


Fig. 6. The SMS on RoboCrane showing a close-up view of one of the three receivers.

3.2 Dynamic pose measurement

A commanded pose will generally result in a different actual pose due to various sources of system error such as those discussed previously. This relationship is depicted as

$$\mathbf{N} \rightarrow \mathbf{X} \rightarrow \mathbf{A} \tag{7}$$

or, in matrix form,

$$\mathbf{NX} = \mathbf{A} \tag{8}$$

where $\mathbf{N}$ is the commanded pose, $\mathbf{X}$ is the perturbation that includes all the sources of error, and $\mathbf{A}$ is the actual pose that results. The effects of $\mathbf{X}$ can be cancelled by commanding an adjusted pose, $\mathbf{N}^*$, where

$$\mathbf{N}^* = \mathbf{N}\mathbf{X}^{-1} \tag{9}$$

Using the adjusted pose allows us to achieve the original desired pose since

$$\mathbf{N}^*\mathbf{X} = \mathbf{N} \tag{10}$$

In general, most of the sources of error are unknown and variable, so computing $\mathbf{X}^{-1}$ apriori is not feasible. However, $\mathbf{X}^{-1}$ can be estimated by comparing a previously commanded

adjusted pose, $\mathbf{N}^*$, with the resulting actual pose, $\mathbf{A}^*$, as measured by the SMS. For time step, (i-1)

$$\mathbf{N}^*_{i-1}\mathbf{X}_{i-1} = \mathbf{A}^*_{i-1} \tag{11}$$

And the inverse of $\mathbf{X}_{i-1}$ can be calculated as

$$\left(\mathbf{X}_{i-1}\right)^{-1} = \left(\mathbf{A}^*_{i-1}\right)^{-1}\mathbf{N}^*_{i-1} \tag{12}$$

For the current time step, (i), the commanded adjusted pose can be calculated as

$$\mathbf{N}^*_i = \mathbf{N}_i\left(\mathbf{X}_{i-1}\right)^{-1} \tag{13}$$

where $\mathbf{N}_i$ is the desired pose for the current time step and $\mathbf{X}_{i-1}$ is the perturbation from the previous time step. Therefore,

$$\mathbf{N}^*_i\mathbf{X}_i \approx \mathbf{N}_i \tag{14}$$

If the platform is moving, then the cancellation is not perfect, since we are trying to cancel this time step's unknown perturbation transform with the inverse from the previous time step, which will be slightly different. If the platform is stationary, the two converge and the cancellation becomes perfect.

Platform motion has a more pronounced practical effect due to measurement latency for $\mathbf{A}$. When computing $\mathbf{X}^{-1}$, it is important that the $\mathbf{N}$ and $\mathbf{A}$ poses are synchronized. If the measured $\mathbf{A}$ pose lags the nominal $\mathbf{N}$ pose, then the compensation will have the effect of *leading* the motion. When speed slows, this leading will become an overshoot, and the platform will oscillate.

In the presence of measurement latency, one solution is to only compute the compensating transform $\mathbf{X}^{-1}$ when the platform is stationary. With this method, the platform is moved into an area of interest, held stationary for at least the latency period, and $\mathbf{X}^{-1}$ is computed. From that point, iteration is suppressed, and the compensating transform is constant. As the platform moves away from the compensation point, its accuracy diminishes.

If the latency is constant and can be measured, a solution is to keep a time history of nominal poses and their associated inverse transforms, and look back into this history by the amount of latency to associate a pair $\mathbf{N}$ and $\mathbf{X}^{-1}$ to the latent $\mathbf{A}$ measurement. If the measurements can be timestamped, then the same technique can be supplemented with timestamps to make the association. This technique can be used in the presence of variable measurement latency.

Controller latency also has an effect on the accuracy of the compensating transform. Figure 7 shows the magnitude of the translation portion of the compensating transform during tests with four different trajectory cycle times. In each test, the platform speed varied from 1 cm/s to 10 cm/s. These tests were done with a simulated measurement system that simulates actual position from the servo position run through the forward kinematics. In this case, the compensating transform should be small, and in fact it goes to zero as the

motion pauses between each speed setting. It is apparent from these figures that as the platform speed increases, the magnitude of the compensating transform increases, as is expected from servo lag. It is also apparent that as cycle time increases, so does the magnitude of the transform. This is due to the uncertainty between when nominal position is registered by the controller, and when it is read out some fraction of a period later.

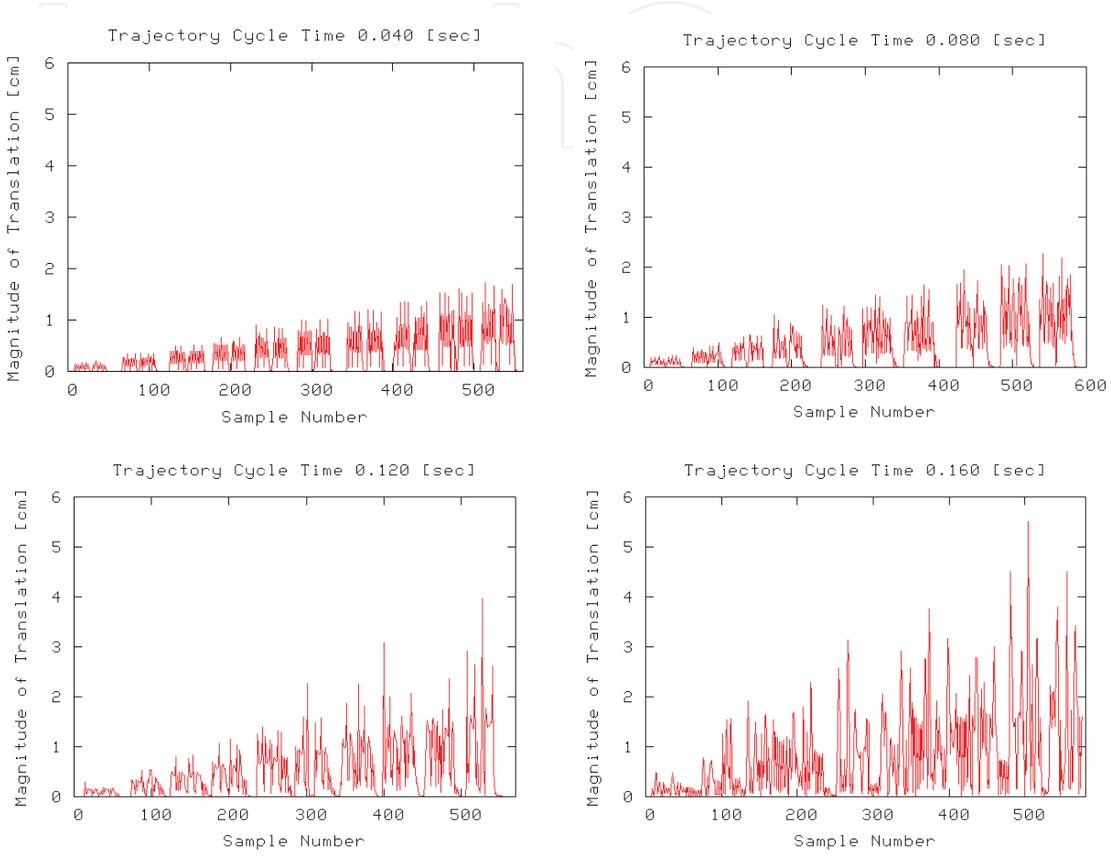


Fig. 7. Compensating transform magnitude (translation only) for four different trajectory cycle times. As the trajectory cycle time increases, the magnitude increases, and becomes more noisy as a result of the increased uncertainty in the latency between control output and compensation. (Note: Figures intended as qualitative examples of cycle time effects.)

Whenever a new $\mathbf{X}^{-1}$ transform is written to the controller, it has the potential to cause a jump in motion. To prevent this, transforms are "walked in" according to speed and acceleration limits. A large change in the transform will appear as a relatively quick but controlled move to the new, more accurate position. The effect of compensation is illustrated in Figure 8. The square path in the lower left of the figure is the uncompensated path, which is offset and slightly skewed from the ideal path due to kinematic miscalibration. Shortly after the second pass around the square path, compensation was turned on and its effects walked in over several seconds. This interval appears as the two line segments connecting the square paths. The square path in the upper right is the compensated path, whose adherence to the nominal edges at 0 cm and 10 cm is quite good.

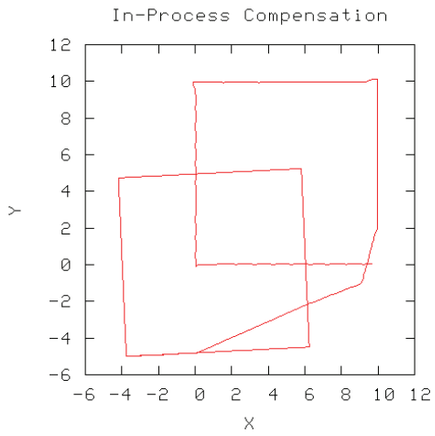


Fig. 8. Effect of in-process compensation The lower left square path is uncompensated and differs due to kinematic miscalibration. The upper right path is compensated. The connecting path results applying the compensation over time to avoid impulsive jumps.

When compensation is turned off, the last compensating transform remains in use. As the platform moves away from the point at which this transform was calculated, the compensation becomes less accurate. This is shown in Figure 9.

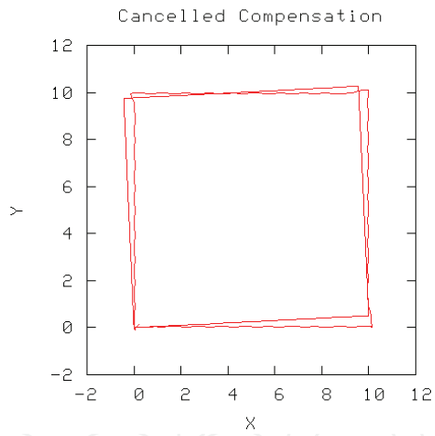


Fig. 9. Trajectory drift after cancelling in-process compensation. The correction was made at location (0,0), and no further updates were performed.

4. RoboCrane control

4.1 GoMotion controller description

The RoboCrane controller is a two-level hierarchy. The bottom level is servo control, which takes position setpoints for the cable lengths at a period of 1 millisecond, and runs a proportional-integral-derivative (PID) controller using feedback from encoders mounted on the motors to generate drive signals. The top level is trajectory planning, which takes desired goal poses and plans smooth Cartesian motion along a linear path, taking into account speed, acceleration and jerk constraints. The trajectory planner executes at a period of 10 milliseconds, calculating intermediate poses that are run through the inverse kinematic

equations to generate cable lengths sent to the servo controllers. Joint mode control is also possible, with goals specified in terms of desired cable lengths. The inverse kinematics are not needed in this case.

Servo control is divided among six similar modules, each running PID control with extensions that handle velocity and acceleration feedforward terms, output biasing, deadband and saturation detection for anti-windup of integral gain. A software application programming interface (API) localizes how the servo modules connect to specific hardware such as commercial input/output boards for encoder feedback and digital-to-analog conversion, open-loop stepper motors or distributed input/output. The servo modules run periodically at 10 times the period of the trajectory planner. Interpolation between setpoints sent by the trajectory planner is done using either linear, cubic or quintic polynomial interpolation of the setpoint over time, depending on application needs.

Trajectory planning is done following *S-curve* velocity profiling with specified velocity, acceleration and jerk. S-curve profiling has the advantage of bounding jerk, when compared with trapezoidal velocity profiling with abrupt changes in acceleration. S-curve profiling has seven motion phases, as shown in Figure 10.

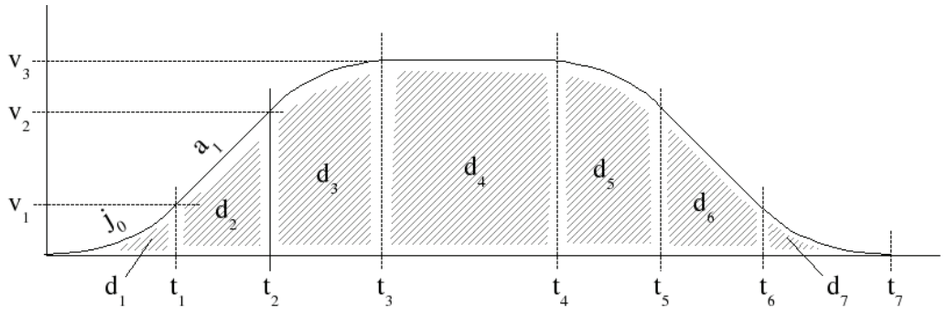


Fig. 10. S-curve velocity profile.

Here, v_3, a_1 and j_0 are the specific maximum velocity, acceleration and jerk, respectively. At each trajectory time step, the distance increment is computed as the area under the S-curve for that time interval.

In joint position control mode (individual cable actuation), trajectory planning is done for each cable independently. Given a desired target cable length, the S-curve profile is computed and distances are computed each trajectory period. These distances are sent to the servo module for that joint for interpolation and tracking. Coordinated joint position control is possible, in which a set of six target cable lengths comprises the goal. Six trajectory profiles are computed, and five of the six are scaled so that their final arrival time matches the time of the longest move.

In Cartesian position control mode, motion control is split into translation and rotation vectors. The translation vector is a three-element vector with X, Y and Z components pointing to the target location, with associated velocity, acceleration and jerk along the path. The rotation vector is a three-element vector about which the overall rotation from the current orientation to the target orientation takes place. The magnitude of this vector is the amount of rotation. Angular velocity, acceleration and jerk are used to generate a profile for this portion of the move. One of the two profiles is scaled to match the time of the longer of the two so that the translation and rotation arrive at the same time. At each trajectory cycle,

the translation and rotation are computed, run through the inverse kinematics equations, and sent as a set of target cable lengths for interpolation and tracking by the servo modules.
Motion along circular arcs is also supported. Rotational motion is planned as before. Translational motion is planned along the arc, where the distance under the S-curve profile is the distance along the arc. Aside from this geometric distinction, circular motion is the same as linear motion.

4.2 Initialization

When the controller begins executing, it assumes that the cable length measurements are uncalibrated. Cable length limits are invalid, as is any notion of the Cartesian pose of the platform or its limits. The controller allows individual cables to be moved independently, but inhibits Cartesian motion and cable length limit checking. Before any of these can take place, the platform must be "homed" to establish the offset between the initial arbitrary measurement of cable length (typically zero) and its true length.
In systems that lack a way to absolutely measure either cable lengths or Cartesian pose at startup, a homing procedure is used. There are several variations in this method. In one, fiducial marks are made on each cable, which when aligned with an associated mark on the platform denote that the cable is at a known length. The operator must manually jog each cable to align the marks, and indicate that the home condition has been met. The controller then computes an offset that is added to the raw feedback from the motor encoder to yield the known length value.
Another homing technique is to bring the platform to a known Cartesian location, such as level and oriented properly atop a mark on the floor. This requires manually moving the platform by adjusting cable lengths, which is unintuitive. In practice, the operator moves each cable so that the platform is relatively close to the home location, and falsely indicates that the cables are homed. Cartesian motion is then enabled, and the operator moves in Cartesian space for the final alignment. During this falsely-homed period, the platform motion will be skewed, but is usually close enough for intuitive positioning.
Homing is a time-consuming manual procedure. If the platform's Cartesian pose can be measured directly, such as with the SMS, then homing is not necessary. In this case, the controller is provided with the actual Cartesian position, which it runs through the inverse kinematics to get the cable lengths. The difference between these computed cable lengths and the uncalibrated lengths from the motor encoders is the offset used to calibrate the feedback.

4.3 Control modes

The RoboCrane controller supports various control modes. Teleoperation allows an operator to drive the platform directly, using a keyboard, mouse or joystick. Automatic control allows the execution of scripted trajectories.
Teleoperated Control: In teleoperated control, the operator uses a convenient input device, such as a keyboard, mouse or joystick, to move the platform directly. Typically a joystick is used, since it is most intuitive. This can be performed in either joint (i.e., cable lengths) or Cartesian space. With cable lengths, the operator selects a cable, and shortens or lengthens the cable according to the deflection of the joystick. If the controller has been homed, the Cartesian position is continually updated using the forward kinematics. Cable length motion is typically used only when homing the platform, since it results in unintuitive platform motion.

In Cartesian space, the operator uses the joystick to drive the platform in any of the X, Y and Z directions, or to rotate about these directions. The controller supports two reference frames: the world frame, with coordinates affixed to the unmoving ground; and the platform (or tool) frame, with coordinates affixed to the moving platform. World mode is typically used to position the platform near an area of interest, or to drive it along features in the world, such as the floor or walls. Tool mode is used to position the platform by driving it along axes aligned with grippers or tooling, so that approaches and departures can be made along arbitrary directions. The controller supports the definition of arbitrary tool coordinate systems, so that one tool can be dropped off, another picked up, and motion with respect to the new tool axes can be accomplished.

In world mode, Cartesian speeds from the joystick are converted into cable speeds using the inverse Jacobian. Given a desired Cartesian velocity of RoboCrane, $\mathbf{V}$, and using the inverse Jacobian[1] matrix, $\mathbf{J}^{-1}$, the cable speed vector, $\dot{\mathbf{L}}$, can be calculated as

$$\dot{\mathbf{L}} = \mathbf{J}^{-1}\mathbf{V} \tag{15}$$

where $\dot{\mathbf{L}}$ is the 6x1 cable speed matrix, $\mathbf{J}^{-1}$ is the 6x6 inverse Jacobian transform matrix, and $\mathbf{V}$ is the 6x1 Cartesian velocity vector (Tsai, 1999). The calculated cable speeds are transformed into winch motor rotation rates that are sent to the winches. Each motor encoder keeps track of the number of motor shaft revolutions and that number is directly related to cable length. The six cable lengths are then used to calculate a new Jacobian matrix, which is used the next time velocity commands are sent.

Since the inverse Jacobian matrix is calculated based on the instantaneous Cartesian pose of RoboCrane, the initial pose of RoboCrane must be known. This initial pose can be calculated by directly measuring the cable lengths and performing the forward kinematic calculations, or by placing RoboCrane in a predefined home pose at the beginning of each teleoperation session and initializing the cable lengths to preset values.

Speed changes are clamped to lie within acceleration limits, so that abrupt changes in joystick position do not impart abrupt changes in motor speed. Cartesian position and orientation limits are applied, so that attempts to drive the platform outside a limit will be inhibited.

Automatic Control: With automatic control, motions in either cable or Cartesian space can be scripted in programs. These programs can be written by hand, or generated by off-line programming systems that can automate the generation of complex tasks throughout a large work volume. This is accomplished through a third level in the hierarchy, the Job Cell level. This level interfaces to the motion controller using the same interface as the teleoperation application, but sending discrete moves instead of teleoperation speeds.

There are two basic modes of automatic control, either in cable space or in Cartesian space. Cable space motions are less common, and would be used to drive individual cables during maintenance activities. Cartesian space motions are primarily used in applications. As with

[1] The Jacobian transform (or simply Jacobian), $\mathbf{J}$ relates the velocities of the joints of a manipulator to the velocities (translational and rotational) of its end-effector, $\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}$, where $\dot{\mathbf{q}}$ and $\dot{\mathbf{x}}$ are the velocity vectors of the joints and end-effector, respectively (Tsai, 1999).

Cartesian teleoperation, programed Cartesian moves can be done either with respect to the world frame or the tool frame. A representative program is

```
# rotate to 30-degree yaw at 1, -2, 3
movew 1 -2.0 3.0 0 0 30.0
# move along the tool's Y axis 10 cm
movet 0 0.1 0 0 0 0
```

World motions are absolute (although they can be incremental), while tool motions are strictly incremental, since the tool origin moves along with the tool.

5. High level control

5.1 4D/RCS overview

The NIST RCS (Albus, 1992) methodology describes how to build control systems using a hierarchy of cyclically executing control modules. In (Bostelman et al., 1996), RCS was applied to a RoboCrane implementation.. At the lowest level of the hierarchy, each control module processes input from sensors, builds a world model, and generates outputs to actuators in response to commands from its supervisory control module. These functional components of a control module are termed sensory processing (SP), world modeling (WM) and behavior generation (BG), respectively. The servo control of a motor is a common example of a control module at the lowest level. Here, the sensor may be a motor shaft position encoder, the actuator is the motor shaft, the command is a desired setpoint for the shaft position, and the behavior may be the execution of a simple PID control algorithm. The SP function may simply be reading and scaling input from the encoder device, and the WM function may be maintaining a filtered estimate of the shaft position. Typical cycle times for such control modules are on the order of a millisecond.

One or more of these lowest-level control modules may be subordinate to a control module at the next level up in the hierarchy, termed the supervisor. In our example, the SP function at this level may simply provide each motor shaft position to the WM function, which would compute the overall position and orientation of the device's controlled point, perhaps the tool on a robot. The BG function may smoothly transform goal points to motor trajectories based on speed, acceleration and jerk. Here, goal points may arrive at variable intervals from the higher-level supervisor, one that may be reading them from a program file. Cycle times increase by about an order of magnitude for control modules that are one level higher in the hierarchy. For this trajectory planner, the cycle time would be about 10 ms.

A full RCS hierarchy would include additional lower-level control modules for individual tools, and control modules at higher levels of the hierarchy may coordinate the actions of many robots and auxiliary equipment. RCS has found its richest application in the area of mobile robotics. Here the SP functions include not just motors but cameras, 3D imaging systems (e.g. laser scanner), GPS and other navigation sensors. WM functions build maps of various resolutions and maintain symbolic representations of the world. BG functions reason on the symbolic representations, planning optimal paths around known features and reacting to sensed obstacles.

An RCS design differs from functional design or object-oriented design in that it begins with a task analysis of the system to be controlled. Here the designer identifies the tasks to be

performed at the top level, and then breaks each task down into subtasks that are performed by the subordinates. Usually the designer does not have complete freedom to determine the task breakdown, as some of the components that make up the system may have been reused from prior projects. In this case, the tasks must be expressed in terms of the available subtasks. Task analyses are helped enormously by considering scenarios that include system startup, shutdown, normal use and changes between various modes of operation. Often these scenarios bring to light the need for tasks that are not apparent from the original conception of the system.

An example of a comprehensive task analysis for the design of an automatic road vehicle controller can be found in (Barbera et al., 2004). The designers considered hundreds of scenarios listed in a manual of military driving, including lane changes, passing and intersection rules. What is made obvious by this analysis is that the top- and bottom-level tasks are relatively simple, while the tasks in the middle are the most complex. Other examples of task analyses for unmanned vehicle systems can be found in the latest version of RCS (known as 4D/RCS) (Albus et al., 2002).

Implementation of RCS control modules is done conceptually using state tables, which can then be programmed in any general-purpose computer language using conditionals or switch statements. The NIST RCS Library documents the software tools available for programming in C++ or Java. A detailed handbook (Gazi, 2001) covers the entire RCS analysis, design and programming using several examples and the RCS Library tools.

5.2 Crane task decomposition

Designing a new RCS-based controller for RoboCrane began by first identifying the requirements of the controller. The overall goal of the RoboCrane controller was defined as follows: to plan and execute tasks required for automated construction-material handling and/or building construction.

Controller Requirements: In order to accomplish its goal the RoboCrane controller needed to provide the following:

- Autonomous, semi-automated, and teleoperated modes of operation
- RoboCrane tool-point (i.e., platform) position and velocity control modes
- RoboCrane tool-point motion in joint, Cartesian, as well as other user-definable coordinate systems
- Cross-platform code portability (but still dependent on the real-time operating system)
- Adaptability to other robot/crane hardware
- Sensor-based collision avoidance

System Scope: Although the motivation for developing a controller was to be able to use it to control various cable-driven robots and to accomplish various tasks, the initial scope of the controller was limited to the following:

- Smooth and stable motion of the NIST RoboCrane
- Perform a steel beam pick and place task
- Construct a structure whose shape is limited by RoboCrane's current range of motion
- Connect the beam to the holder using drop-in connectors
- Carry beams whose size falls within RoboCrane's current load-carrying capabilities
- Communicate to RoboCrane using the current field bus architecture
- Operate under a real-time Linux operating system

- Use the built-in incremental winch motor encoders as well as the laser-based positioning system to determine RoboCrane's pose, but include the ability to add other sensors for pose determination in the future
- Acquire the steel beam and holder poses using the current laser-based positioning system

Task Decomposition: The next step in the RCS controller design process is to conduct a task decomposition of the controller's overall goal. RoboCrane's overall goal was divided into several subtasks, which were consequently also broken down into smaller tasks. This process continued until the lowest level tasks involved sending commands to the RoboCrane hardware (e.g., setting motor voltages). This is the lowest level of control that the controller can provide.

Figure 11 shows a sample task tree diagram resulting from the task decomposition process. In this figure the physical task of picking and placing a steel beam (as part of a steel erection sequence) is decomposed into 3 levels of subtasks. In keeping with the RCS architecture, each sublevel is responsible for planning and executing a smaller portion of the overall pick-and-place task. The lowest level is responsible for maintaining a commanded joint (or motor) velocity (or position). The next level up is responsible for generating and executing a series of n waypoints (i.e., positions and orientations in time) for the RoboCrane platform. The next higher level generates and executes the necessary commands to accomplish a segment of the pick-and-place operation. Finally, the highest level in Figure 11 is responsible for coordinating the execution of the segments that make up the overall pick-and-place task. This highest level also receives commands from higher levels (not shown in Figure 11) which coordinate the pick-and-place task with other tasks such as attaching a beam to a structure, picking and placing a column, and etc.

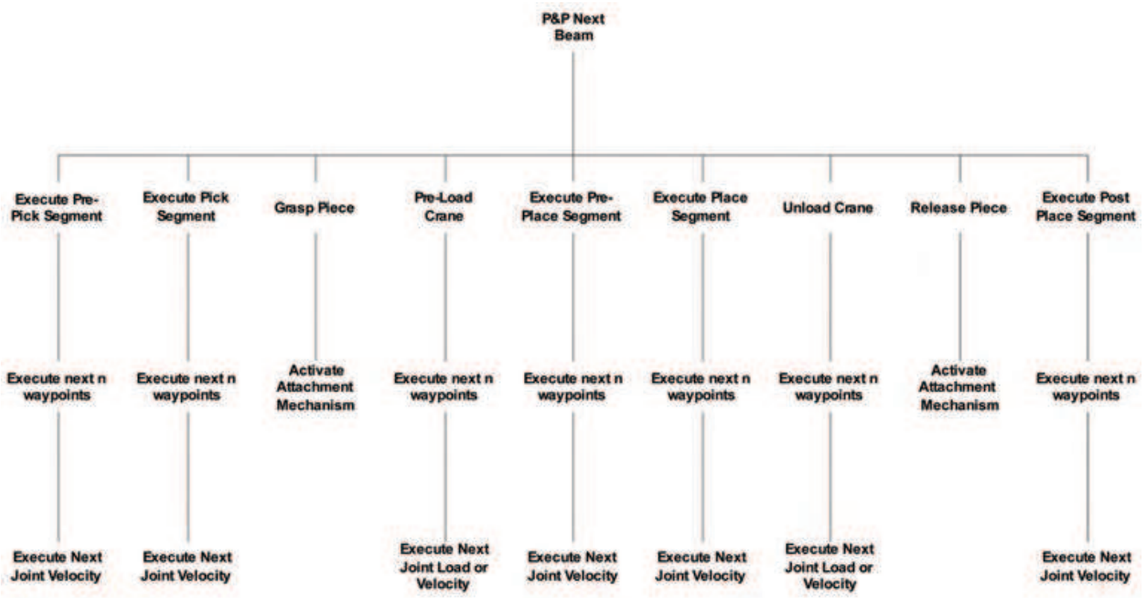


Fig. 11. Task tree diagram for the pick-and-place next beam task.

In addition to the physical tasks represented in the task tree diagram of Figure 11, other non-physical tasks are required in order to accomplish a pick-and-place operation. These

include tasks such as detecting obstacles, calculating collision free paths, etc. These tasks were also captured and broken down into 3 levels of subtasks, but are not included in Figure 11

State Tables: Following the task decomposition process the commands going into and out of each task, that are represented in the task tree diagram of Figure 11, are listed in a state table format. A state table (or state transition table) describes all possible input and output states (and actions) of a finite state machine. Table 1 shows a state table for the pick and place next beam task. The command that starts the execution of this task has the same name as the task itself and is also the title of the state table. The state table columns (from left to right) represent the input state numbers, the conditions that must be met to change the state, the output state numbers, and the output commands that are sent to lower level tasks, respectively.

When the pick and place next beam command is issued by a higher level task, the controller examines the state table shown in Table 1. The initial state of the pick and place next beam task is S0 and the first condition that is checked is whether the received command is new. If it is a new command, the state of the task is changed to S1 and the status of the task is changed to indicate that it is executing.

Pick and Place Next Beam			
S0	New Command	S1	Hold – Status=Executing
S1	Conditions Good to Move to Pre-Pick Pose	S2	Move to Pre-Pick Pose
S1	Timed out	S0	Hold – Status=Error
S2	Conditions Good to Move to Pick Pose	S3	Move to Pick Pose
S3	Conditions Good to Grasp	S4	Grasp Beam
S4	Conditions Good to Pre-Load Crane	S5	Pre-Load Crane
S5	Conditions Good to Move to Pre-Place Pose	S6	Move to Pre-Place Pose
S6	Conditions Good to Move to Place Pose	S7	Move to Place Pose
S7	Conditions Good to Unload Crane	S8	Unload Crane
S8	Conditions Good to Release	S9	Release Beam
S9	Conditions Good to Move to Post Place Pose	S10	Move to Post Place Pose
S10	At Post Place Pose	S0	Hold - Status=Done

Table 1. State table for the pick and place next beam task.

The next time the above state table is checked (i.e., during the next execution cycle of its corresponding control module) the new state of the task is S1, and the conditions that must be met are whether it is acceptable to move RoboCrane to the beam's pre-pick pose, or whether enough time has elapsed that something must be wrong. There may be one or more sub-conditions that must be satisfied in order to determine whether it is acceptable to proceed, but these can be aggregated into one description in the state table. If the conditions are met, the state of the task is changed to S2 and the command to move to the pre-pick pose is sent to a lower-level task. If time has expired, the state of the task is changed to S0 and an error is reported. Each lower level task that receives an output command reports its status back to the higher level task that issued the command until it finishes executing or

encounters an error. This process continues until all of the commands in the state table have been executed, at which point the pick and place next beam task is considered completed and the state of the table is reset to S0. For brevity, only a single timeout condition is shown in Table 1. In practice, numerous checks of this sort are made throughout the state table. Once the state tables for all of the tasks identified through the task decomposition process are completed they are organized into control modules as described next and implemented in software following the RCS guidelines.

Control Modules: As indicated in the prior RCS description, the commands in the task tree diagram of Figure 11 are organized into multiple levels. Each level's tasks may be grouped together into one or more modules responsible for coordinating and executing the tasks within it. Some of the critical modules (such as the servo algorithms) run as real-time processes within the operating system, while other less critical modules (such as long term path planning) run as non-deterministic processes.

Figure 12 shows the control architecture for the RoboCrane controller. The four levels above the software/hardware demarcation line in Figure 12 correspond to the four levels of Figure 11. The tasks have been grouped into the control modules shown. For example, the bottom level tasks of Figure 11 are grouped into the six "Servo" modules in Figure 12.

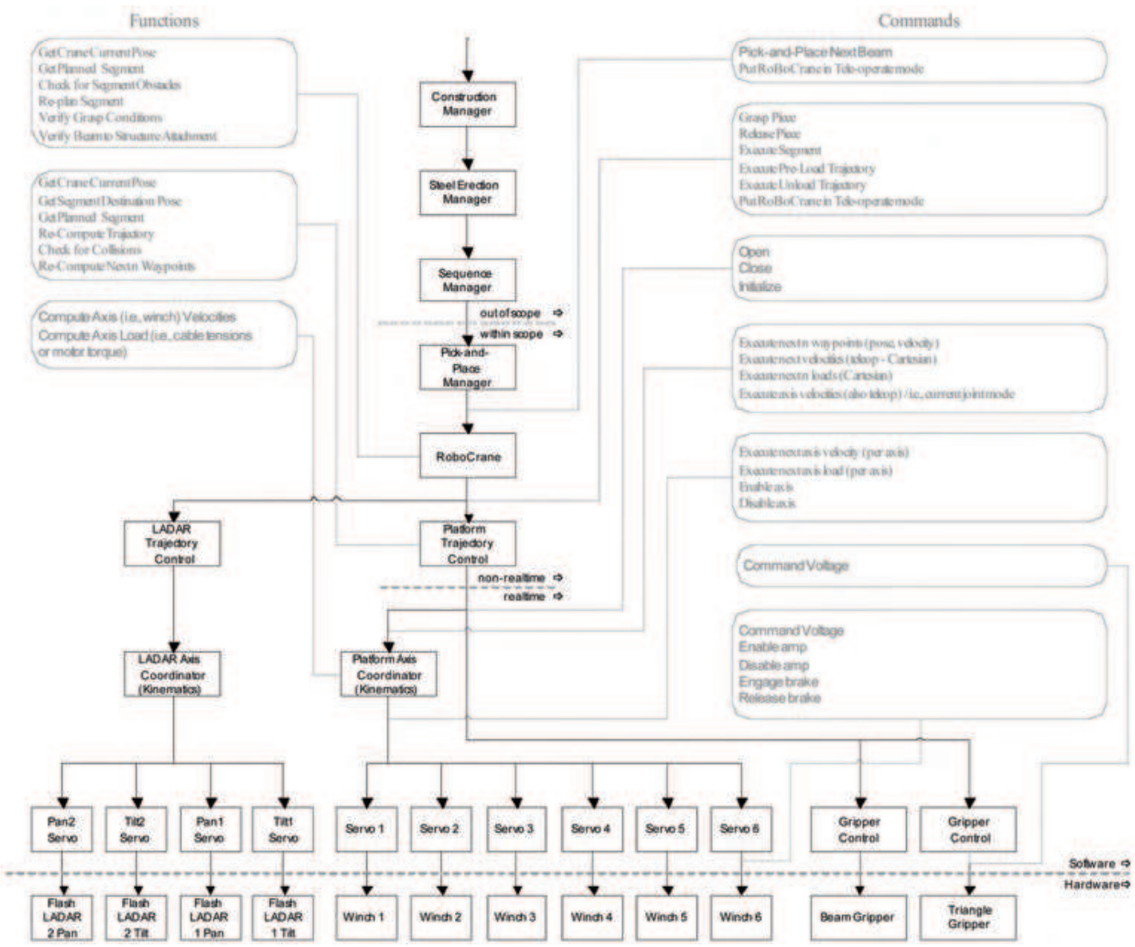


Fig. 12. RoboCrane controller architecture diagram.

Each of these modules are responsible for executing a servo algorithm which accepts the actual and desired positions (or velocity) of a winch motor as inputs and calculates a command voltage which maintains the desired position (or velocity). An alternate configuration would be to group the six servo modules into one.
Figure 12 also shows that the RoboCrane controller is part of a larger control architecture which includes four higher-level modules. For example, at the level above the RoboCrane controller would be a Pick-and-Place Manager that would actually command RoboCrane to perform the pick-and-place operation. The commands sent down by each module to a lower-level module are shown in the light gray boxes on the right. Some of the functions (or non-physical tasks) that each module performs are also shown in the light gray boxes on the left. The control modules above the Pick-and-Place Manager are also included in the figure. Finally, Figure 12 also includes modules for controlling the 3D imaging systems. These modules are responsible for coordinating the sensor orientations with the RoboCrane platform's motion in order to maintain a desired part of RoboCrane's environment within the combined sensors' field of view.

6. Conclusion

This chapter presented new research developments at NIST in control algorithms and controller design for parallel robots applied to Construction applications. In particular, this research focused on the NIST RoboCrane platform for automated placement of construction components. This work was the first to demonstrate the use of a laser-based site measurement system for 6 degree-of-freedom tracking of a robotic crane, and presented new methods for incorporating pose estimation errors in a compensation transform for the NIST GoMotion controller. Finally, this work presented task decomposition approaches for analyzing and automating construction crane operations based on a NIST 4D/RCS approach.

7. References

Albus, J., Huang, H., Messina, E., Murphy, K., Juberts, M., Lacaze, A., Balakirsky, S., Shneier, M., Hong, T., & Scott, H. (2002). 4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems. *National Institute of Standards and Technology, Gaithersburg, MD, NISTIR 6912*.

Albus, J.S., Bostelman, R.V., & Dagalakis, N.G. (1992). The NIST ROBOCRANE, A Robot Crane. *Journal of Robotic Systems, July*.

Barbera, T., Albus, J., Messina, E., Schlenoff, C., & Horst, J. (2004). How task analysis can be used to derive and organize the knowledge for the control of autonomous vehicles. *Robotics and Autonomous Systems 49(1-2)*, 67-78.

Bostelman, R., Jacoff, A., Dagalakis, N., & Albus, J. (1996). RCS-Based RoboCrane Integration. *Proc. Intelligent Systems: A Semiotic Perspective, Gaithersburg, MD, Oct*, 20-23.

Bostelman, R., Shackleford, W., Proctor, F., Albus, J., & Lytle, A. (2002). The Flying Carpet: A Tool to Improve Ship Repair Efficiency. *American Society of Naval Engineers Symposium, Bremerton, WA, Sept*, 10-12.

Gazi, V. (2001). The RCS Handbook: Tools for Real-time Control Systems Software Development. Wiley.

Tsai, L.W. (1999). Robot Analysis: The Mechanics of Serial and Parallel Manipulators. Wiley-Interscience.

Parallel Manipulators, towards New Applications

Edited by Huapeng Wu

ISBN 978-3-902613-40-0
Hard cover, 506 pages
Publisher I-Tech Education and Publishing
Published online 01, April, 2008
Published in print edition April, 2008

In recent years, parallel kinematics mechanisms have attracted a lot of attention from the academic and industrial communities due to potential applications not only as robot manipulators but also as machine tools. Generally, the criteria used to compare the performance of traditional serial robots and parallel robots are the workspace, the ratio between the payload and the robot mass, accuracy, and dynamic behaviour. In addition to the reduced coupling effect between joints, parallel robots bring the benefits of much higher payload-robot mass ratios, superior accuracy and greater stiffness; qualities which lead to better dynamic performance. The main drawback with parallel robots is the relatively small workspace. A great deal of research on parallel robots has been carried out worldwide, and a large number of parallel mechanism systems have been built for various applications, such as remote handling, machine tools, medical robots, simulators, micro-robots, and humanoid robots. This book opens a window to exceptional research and development work on parallel mechanisms contributed by authors from around the world. Through this window the reader can get a good view of current parallel robot research and applications.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen