

# **A Simulink Based Robotic Toolkit for Simulation and Control of the Puma 560 Robot Manipulator\***

W. E. Dixon<sup>1</sup>, D. Moses<sup>2</sup>, I. D. Walker<sup>2</sup>, and D. M. Dawson<sup>2</sup>

<sup>1</sup>Robotics and Process Systems Division, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831-6305

<sup>2</sup>Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634-0915

E-mail: dixonwe@ornl.gov, Telephone: (865) 574-9025

Keywords: Robotic Toolkit, Real-Time Control, Simulink, Graphical User Interface

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-96OR22464. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

Submitted to the IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 29-Nov. 3, 2001, Maui, Hawaii

---

\* This research was performed in part by a Eugene P. Wigner Fellow and staff member at the Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725 and is supported in part by the U.S. NSF Grants DMI-9457967, DMI-9813213, EPS-9630167, ONR Grant N00014-99-1-0589, a DOC Grant, and an ARO Automotive Center Grant.

# A Simulink-Based Robotic Toolkit for Simulation and Control of the PUMA 560 Robot Manipulator\*

W. E. Dixon<sup>†</sup>, D. Moses<sup>‡</sup>, I. D. Walker<sup>‡</sup>, and D. M. Dawson<sup>‡</sup>

<sup>†</sup>Robotics and Process Systems Division, Oak Ridge National Laboratory, P.O. Box 2008-6305, Oak Ridge, TN 37831

<sup>‡</sup>Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634

dixonwe@ornl.gov, dmoses, ianw, ddawson@ece.clemson.edu

## Abstract

In this paper, a Simulink Robotic Toolkit (SRTK) for the Puma 560 robot manipulator is developed on the MATLAB/SIMULINK-based platform. Through the use of the Real-Time Linux Target and the Real-Time Windows Target, the SRTK can be executed on the Linux or Win32-based operating systems (e.g., Windows 95/98/NT) in real-time. Moreover, the graphical user-friendly nature of Simulink allows the SRTK to be a flexible tool that can easily be customized to fit the specific needs of the user. That is, based on the layered approach of the SRTK, the user can perform operations such as calibration, joint control, Cartesian control, Cartesian PD control, impedance control, some trajectory generation tasks, and real-time simulation of the Puma 560 through a user-friendly MATLAB-based graphical user interface (GUI) without writing any code. One of the key features of the toolkit is that users can easily incorporate additional functionality and hardware through the simple block diagram interface that Simulink provides. Moreover, the MATLAB-based GUI can also easily be modified to allow the user to exploit the added functionality with the GUI by using straightforward MATLAB script files. Hence, the SRTK allows a researcher to use the Puma 560 without the burden of the external issues related to the control, interface, and software issues, while providing for the flexibility for easily modifying components for increased functionality.

## 1 Introduction

The field of robotics is highly diversified and multidisciplinary. For example, a few of the present robotics research efforts are directed at trajectory generation, redundancy resolution, actuator-level control, sensor development and application (e.g., visual servoing), and biologically inspired kinematic design. The diversified nature of robotics presents a roadblock to many researchers, because to utilize a robotic system, the researcher must first address a variety of issues such as: hardware interfacing, actuator-level control implementation, software development, and user-interface development before the issues that are of interest to the researcher can be addressed. One avenue that robotics researchers have pursued to overcome this obstacle is the development of a software platform that provides the user with basic operating capabilities of the robotic system. To this end, software developers have utilized: 1) robot control languages, 2) high-level programming languages like C/C++, 3) graphical control environments, and 4) robotic libraries for common programming languages.

Robot control languages provide a set of commands for implementing control applications on the robotic system; how-

ever, they are typically provided by the vendors of the robotic system and exploit proprietary hardware such as special purpose processors. Due to the fact that these languages are provided by the vendor and exploit propriety hardware, the users ability to incorporate additional functionality and/or hardware components is very limited. Due to the limited scope of robot control languages, some researchers have been motivated to develop a new software platform from scratch using high-level programming languages such as C. Clearly, this approach allows a developer to implement a custom system that fits specific needs, however as described earlier, the developer must address issues that are not related to the specific research task. In addition, the development of the custom software platform is very time consuming and error-prone and requires a high level of skill. Since the development of a custom software platform is such an extensive task, some software libraries have been developed to facilitate robotic applications (e.g., RCCL [1] and the robot control library ARCL [2]). However, the size and complexity of these libraries make them very difficult to modify (see [3] for a discussion related to the obstacles that were faced using RCCL and ARCL to develop a robotic manipulator system for decommissioning tasks).

One of the limiting factors of the robotic platform concepts described above is that to incorporate new functionality or new hardware, a user must modify the source code. To overcome this problem, object-oriented approaches and robot control libraries have been developed (see [4], [5], [6], and the references within). One of the most recent object-oriented concepts is given in [3]. In [3], Loffler et al. describe the structure and benefits of the QMotor Robotic Toolkit (QRTK), which is “a set of C++ libraries and programs that follow object-oriented concepts to ensure code reuse, modularity, scalability, and an intuitive code structure” that executes on the QNX4 and QNX6 operating systems. However, one of the potential drawbacks of the QRTK is that the user is required to write and debug code written in C or C++ to implement control algorithms, and if the user wants to incorporate additional functionality, the user is required to write additional object-oriented code. In [7], Ge et al. describe the open-architecture platform OpenRob for model building, control design, and simulations for robot manipulators on Win32 operating systems (e.g., Windows 95/98/NT); however, OpenRob users are required to write custom dynamic link libraries (DLLs) to develop customized robot modules, the construction of the GUI is hidden and would require an experienced user to modify the GUI to add more flexibility, and although the authors of [7] claim real-time control under Win32 operating systems, no description is provided regarding how real-time performance is achieved.

In contrast to the previous platforms, several researchers have elected to develop robotic applications based on the MATLAB-based platform. Specifically, in [8], Chen and Naughton proposed a MATLAB/Simulink-based control system platform for the Linux operating system as a means for students to design, simulate, and implement various control laws on a DC motor without requiring the students to: 1) program the controller using languages such as C or C++, 2) switch back and forth between platforms for design, simulation, and implementation tasks, 3) and spend time debugging

---

\*This research was performed in part by a Eugene P. Wigner Fellow and staff member at the Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725 and is supported in part by the U.S. NSF Grants DMI-9457967, DMI-9813213, EPS-9630167, ONR Grant N00014-99-1-0589, a DOC Grant, and an ARO Automotive Center Grant.

the control code. Unfortunately, as stated in [8], the control is not implemented in real-time due to the fact that MATLAB-MEX functions were utilized. In [9], Pires describes the development of a MATLAB-based toolbox called MATROBCOM that can be utilized to control some industrial robotics and automation equipment in Win32 operating systems. Unfortunately, since MATROBCOM also relies on the use of MATLAB MEX functions, the toolbox cannot be executed in real-time. In contrast, the Simulink Robotic Toolkit (SRTK) that is described in this paper is developed on the MATLAB and Simulink platforms in conjunction with the Real-Time Linux Target (RTLTL) [10] or the Real-Time Windows Target (RTWT) [11], and hence, the SRTK can be executed in real-time<sup>1</sup> in Linux and Win32 operating systems. Specifically, the SRTK for the Puma 560 arm is set of MATLAB script files (m-files), MATLAB-toolboxes, and Simulink block-diagrams that can be used for real-time simulation, control, and data-logging for post-runtime analysis of the Puma 560 robot manipulator. The driving philosophy behind the SRTK is that a user-friendly MATLAB/Simulink-based environment enabled by a MATLAB-based Graphical User Interface (GUI) allows users with various backgrounds to exploit the functionality of MATLAB and Simulink to perform robotics-based research on the Puma 560. That is, the efforts of a broad class of researchers are enabled by the fact that the SRTK exploits the modular nature of Simulink block-diagrams and the functionality that the wide range of MATLAB toolboxes provides. For example, a researcher with expertise in control theory can simply remove the control block from the SRTK Simulink block-diagram and substitute a user-developed control block-diagram for the new control algorithm. Likewise, researchers investigating problems related to robot vision or trajectory generation could easily modify the Simulink block-diagram to test the specific research concepts without having to devote time and energy to address issues that are not related to the research effort (e.g., interfacing with the Puma 560 hardware, developing an actuator-level control algorithm). Although there are many elements of the SRTK that are generic to any robot manipulator (e.g., trajectory generation), the SRTK is robot specific due to the requirement for robot specific elements (e.g., calibration routines, hardware interface elements). Extending the SRTK to other robots is not a trivial task since either the robot would have to be truly open architecture (i.e., open access to all the hardware components as in the Barrett Whole Arm Manipulator (WAM)) or considerable back engineering efforts of the robot would be required. In fact, the current development of the SRTK for the PUMA 560 was greatly facilitated by the extensive back engineering efforts by Voyle [17]. Since the WAM is a truly open architecture robot, collaborative efforts with Barrett are in progress regarding extending the SRTK to support the WAM.

The structure of the SRTK is based on a layered approach, in which the top-most layer is the GUI. The GUI enables a user to exploit the functionality of the SRTK without requiring the user to work with the Simulink block-diagram; however, based on the layered structure of the SRTK, a user can easily integrate additional hardware, incorporate additional functionality, or modify the algorithm of an existing function (e.g., replace the SRTK trajectory generator with a user-developed trajectory generator) by simply modifying the various layers of the Simulink block-diagram. In some instances, the SRTK GUI may need to be modified to reflect the additional user defined functionality. Since the GUI is constructed using simple MATLAB script files, the user can create their own GUI or simply modify the SRTK GUI as desired. To provide an overview of the SRTK, this paper is organized as follows. In Section 2, a description of the underlying Simulink-block diagram is provided. In Section 3, the different operating modes of the SRTK are described. In Section 4, the real-time Puma simulator is described. In Section 5, a brief overview of the

<sup>1</sup>Hard real-time (i.e., predictable latency) is guaranteed using RTLTL, however, it is not clear that it is guaranteed using MATLAB's Real-Time Workshop (RTW) and the RTWT; although, RTW is advertised as providing real-time performance, to the best of our knowledge, no measures of latency have been published that would indicate hard real-time is achieved.

functionality of the SRTK GUI is given. Concluding remarks are presented in Section 6.

## 2 Description of the Simulink Block-Diagram

The underlying Simulink block-diagram of the SRTK is organized to facilitate the addition or removal of different operating modes and/or auxiliary hardware components. As described below, the Simulink block-diagram can be divided into three main sections including: the Input Section, the Computation Section, and the Output Section. This structure allows the user to either target the basic Input/Output (I/O) operations with the Puma 560 through the Input and Output sections, or target the modification of the functionality and execution of the different operating modes that the SRTK provides through the Computation section.

**Input Section:** The Input Section of the Simulink block-diagram provides the input interface between the Puma 560 and the SRTK. That is, every signal that is read into the SRTK from the Puma 560 and the GUI such as the encoder values, the potentiometer values, the desired joint angles, and the desired Cartesian positions are located in this section along with input signals from the GUI, such as the desired end-effector position. If a user does not incorporate any additional hardware with the Puma 560, the Input Section will not need to be modified unless the user wants to place additional scopes on the diagram to observe the input signals. If a user adds additional hardware to the Puma 560, then the user would be required to include new inputs in this section. In addition to the input signals from the Puma 560, the Input Section has also incorporated selector switches, which link the GUI to the Simulink block-diagram. That is, the selector switches provide a means for a user to execute the different operating modes of the SRTK from the GUI.

**Computation Section:** The Computation Section incorporates the mathematical details for the various modes of operation for the Puma as separately labeled subsets (masked blocks). Each of the operating modes is encapsulated in a separate subsystem that is either enabled or disabled as desired using the selector switches that are accessed through the GUI. One of the advantages of the SRTK is that the user-friendly GUI, which is utilized to control all the functions of the SRTK, is constructed using standard MATLAB functions, and hence, if a user desires to incorporate additional functionality in the SRTK, the user can simply modify the existing Simulink block-diagram and then either create a new GUI or modify the existing GUI to incorporate the additional functionality.

**Output Section:** The Output Section of the diagram provides the output interface between the Puma 560 and the SRTK. If a user does not incorporate any additional hardware with the Puma 560, the Output Section will not need to be modified unless the user wants to place additional scopes on the diagram to observe the output signals. If a user adds additional hardware to the Puma 560, then the user would be required to include new outputs in this section. In addition to the signals output to the Puma 560, the Output Section also incorporates the Puma 560 dynamic model to facilitate real-time simulation of the Puma without activating the hardware. The Output Section also has embedded safety features that enable the SRTK to avoid commanding excessively high voltage or torque values to the Puma 560. A termination block is also included in this section for Windows users to allow the SRTK to disable the arm-power at the end of the real-time simulation mode.

## 3 Operating Modes

### 3.1 Calibration Mode

The calibration mode is the most vital mode of the SRTK because it calibrates the Puma 560 hardware. If the Puma 560 is not calibrated correctly, the other modes of the SRTK will

not function correctly because the information received from the manipulator sensors will be invalid. In fact, the other modes of the SRTK GUI are not enabled until the Puma has been successfully calibrated. To calibrate the Puma 560, the user simply presses the Calibrate Puma button of the GUI. When the Calibrate Puma button is pressed, the SRTK acquires voltage readings (that relate to the joint angle) from the potentiometers that are incorporated with each joint of the manipulator. To filter the noise that is inherent to the potentiometer voltage signals, the average of 100 readings is taken, and hence, a 100 millisecond delay is incurred since the SRTK is operated at 1kHz. Based on the filtered voltage readings from the potentiometers, denoted by  $PV \in R$ , the encoder values of each joint, denoted by  $EV_{estimated} \in R$ , are set using the following expression:

$$EV_{estimated} = PS * PV + PI \quad (1)$$

where  $PS \in R$  and  $PI \in R$  are parameters that represent the “potentiometer slope” and the “potentiometer intercept” that must be determined by the user for the specific Puma 560 (see [12] for information regarding these parameters). Based on the rough estimates of the joint positions, the SRTK enables a proportional derivative (PD) joint controller to move each joint of the Puma to an index pulse of the joint encoder. The motion of the manipulator is always towards the “Ready Position” because the “Ready Position” locates each joint at its midpoint, and hence, the possibility of reaching a joint limit during calibration is eliminated. When the encoder index pulse is detected, the encoder readings at that point are recorded as  $EV_{atIndex} \in R$ . The correct encoder count at the nearest index pulse, denoted by  $EV_{atNIndex} \in R$ , is then determined by the SRTK as follows:

$$EV_{atNIndex} = \text{floor} \left( \left[ \frac{\text{double}(EV_{atIndex} - eFirst)}{eDelta} \right] + 0.5 \right) eDelta + eFirst \quad (2)$$

where  $\text{floor}()$  is a function that rounds the argument down to the nearest integer,  $\text{double}()$  is a function that converts the argument to a real value, and  $eFirst, eDelta \in R$  are constant parameters that depend on the user’s specific Puma 560 and denote the encoder count measured at the first index pulse from the joint limit and the number of encoder counts between two index pulses, respectively (see [12] for specific information regarding these parameters). After the SRTK determines the correct encoder count at the index pulse, the manipulator moves back to the initial position. To determine the calibrated position of the encoder (and hence the joint angle) at the present position, denoted by  $EV_{calibrated} \in R$ , the following equation is utilized:

$$EV_{calibrated} = EV_{atNIndex} + EV_{present} - EV_{atIndex} \quad (3)$$

where  $EV_{present} \in R$  is utilized to denote the current position instead of  $EV_{estimated}$  since the manipulator may not have moved back to exactly the same encoder position. The correct encoder values are set in the SRTK when the Calibration Done button of the GUI is pressed.

**Remark 1** Due to the similarity in the construction, the required calibration parameters given in [12] can be used to calibrate Puma 560 manipulators. That is, given the information in [12], a user is not required to use parameter identification techniques.

### 3.2 Zero Gravity Mode

Often times, a user may want to teach specific joint positions to the manipulator by simply positioning the joints by hand. This task is difficult due to the weight of the Puma 560 links. That is, once the mechanical joint brakes are released, the user has to support the weight of the entire manipulator while attempting to accurately position each of the joints. To facilitate this operation, the zero gravity mode of the SRTK applies a gravity compensation torque to each joint allowing the controller to absorb the burden of supporting the manipulator links and allowing the user to simply move each joint to the

desired location. This goal is achieved through the SRTK by applying a gravity compensation torque to each joint of the manipulator as shown below

$$\tau = G(q) \quad (4)$$

where  $\tau \in R^6$  denotes the control torque input,  $G(q) \in R^6$  denotes the gravity effects acting on the manipulator, and  $q(t) \in R^6$  denotes the joint angles. Once the user positions the Puma in a desired configuration, the joint positions can be stored for later use by other operating modes.

### 3.3 Joint Control Mode

This mode is the one of the most basic modes for controlling the Puma 560. The user-defined inputs for this mode are the final desired joint positions and a movement scale factor. Given the initial position and the user-defined final position of the manipulator, the maximum of all the joint errors is calculated as follows:

$$\begin{aligned} \text{delta}_{\max} &= \max(|q_{final}(i) - q_{initial}(i)|) \\ \forall i &= 1, 2, \dots, 6 \end{aligned} \quad (5)$$

where  $\text{delta}_{\max} \in R$  denotes the maximum joint error, and  $q_{final}, q_{initial} \in R^6$  denote the desired joint position and the initial joint position, respectively. Based on the maximum joint error and the user-defined movement scale parameter, denoted by  $MS \in R$ , the time scale for the desired trajectory is calculated from the following equation:

$$\bar{t} = \frac{t}{MS(\text{delta}_{\max})} \quad (6)$$

where  $\bar{t} \in R$  represents a scaled time signal. Once the time scale has been determined, the position, velocity, and acceleration trajectories are developed from the following equations:

$$s\text{delta} = q_{final} - q_{initial} \quad (7)$$

$$q_d = s\text{delta} (6\bar{t}^5 - 15\bar{t}^4 + 10\bar{t}^3) + q_{initial} \quad (8)$$

$$\dot{q}_d = s\text{delta} (30\bar{t}^4 - 60\bar{t}^3 + 30\bar{t}^2) \quad (9)$$

$$\ddot{q}_d = s\text{delta} (120\bar{t}^3 - 180\bar{t}^2 + 60\bar{t}) \quad (10)$$

where  $q_d(\bar{t}), \dot{q}_d(\bar{t}), \ddot{q}_d(\bar{t}) \in R^6$  denote the desired position, velocity, and acceleration trajectories that are generated by the SRTK.

After the desired trajectory has been formulated by the SRTK, the difference between the actual and desired position and velocity of the manipulator joints are utilized in conjunction with a gravity compensation term and the desired acceleration trajectory to construct the following torque control input:

$$\tau = K_p (q_d - q) + K_d (\dot{q}_d - \dot{q}) + \ddot{q}_d + G(q) \quad (11)$$

where  $K_p, K_d \in R$  are constant control gains and  $\tau(q) \in R^6$  represents the control torque input. One of the advantages of this mode is that the PD control law does not contain singularities since the control is performed at the joint level. However, the safety measures will disable the arm power and the mechanical locks will hold the manipulator at the terminated position if the controller attempts to force the Puma 560 in a configuration that violates a joint limit. At the end of the motion, the joints are maintained at the desired positions.

### 3.4 Cartesian Control Mode

The Cartesian control mode enables the user to select desired final Cartesian coordinates for the manipulator rather than selecting the desired final joint configurations as in the joint control mode; however, for simplicity, in this mode and the subsequent Cartesian PD and impedance control modes, the roll, pitch, and yaw of the end-effector are held fixed and only the position of the end-effector in the XYZ Cartesian plane is controlled. Based on the initial and final Cartesian position of the manipulator, the SRTK generates a desired trajectory

(in a similar manner as given in (5)-(10)) in the Cartesian Space that is scaled by the user defined movement scale factor, and a PD control law is applied to make the Puma follow the trajectory. Specifically, to force the Puma manipulator to follow the desired trajectory generated by the SRTK, a PD control law is applied to the first three joints of the Puma 560 as follows [13]:

$$\tau(XYZ) = K_p(XYZ_d - XYZ) + K_d(\dot{X}\dot{Y}\dot{Z}_d - \dot{X}\dot{Y}\dot{Z}) \quad (12)$$

$$\tau(q) = J^T \tau(XYZ) + G(q) \quad (13)$$

where  $K_p, K_d \in R$  are constant control gains,  $\tau(XYZ) \in R^3$  denotes the torque control input in the Cartesian space,  $\tau(q) \in R^6$  denotes the torque control input in the joint space,  $J^T(t) \in R^{6 \times 3}$  represents the transpose of the Jacobian for the first three joints,  $G(q) \in R^6$  represents the gravity effects,  $XYZ_d(t) \in R^3$  denotes the desired Cartesian position of the end-effector, and  $XYZ(t) \in R^3$  denotes the actual Cartesian position of the end-effector that is calculated on-line from the forward kinematics of the robot as shown below:

$$XYZ = f(q) \quad (14)$$

where  $f(q) \in R^3$  represents the forward kinematics for the first three joints of the Puma 560 calculated using the Denavit-Hartenberg parameters according to Lee [14].

When the Puma manipulator is controlled in this mode, potential singularities may arise from the Jacobian. When the manipulator approaches these singularities, the generated torques will be very high and undesirable and unpredictable motion of the Puma may result. Safety measures implemented in the Output Section of the Simulink block-diagram will stop the motion of the manipulator, unless the unpredictable torque values remain within the joint torque saturation limits. In addition to singularities, extremely high torque values may result due to excessively aggressive trajectories. That is, to follow a desired Cartesian trajectory that requires fast movements, excessive joint torques may result.

### 3.5 Cartesian PD Control Mode

The Cartesian PD control mode is similar to the Cartesian control mode since both modes allow the user to select the final position of the end-effector in the Cartesian space. In both modes, the SRTK generates a desired trajectory from the present position to the user defined final Cartesian position where the trajectory is generated in a similar manner as in (5)-(10). One of the differences in the Cartesian PD control mode is that the inverse kinematics of the Puma 560 are calculated on-line to relate the Cartesian space desired trajectory to the joint space. Specifically, once the Cartesian trajectory has been calculated, the inverse kinematics of the Puma 560 are calculated on-line to relate  $XYZ_d(t)$  to the desired trajectory in the joint space; however, since there are 8 possible solutions to the inverse kinematics problem, the SRTK computes the inverse kinematics for each of the possible configurations as shown below

$$q_{config(i)} = f^{-1}(XYZ_d) \quad \forall i = 1, 2, \dots, 8 \quad (15)$$

where  $q_{config(i)} \in R^6$  denotes the  $i$ -th inverse kinematics solution of the 8 configurations. The SRTK then selects the joint angles for the configuration that results in the smallest change from the current position as follows:

$$\begin{aligned} q_{Paul} &= q_{configuration} \ni \left\{ |q_{present} - q_{config(i)}| \right\} \quad (16) \\ &= \min \left( \sum_{j=1}^6 |q_{present}(j) - q_{config(i)}(j)| \right) \\ \forall i &= 1, 2, \dots, 8. \end{aligned}$$

That is, the inverse kinematics for each of the 8 different configurations of the manipulator is determined, and the configuration that results in the smallest change from the present position (measured by the sum of the absolute value of the difference in the  $j$ -th element of the vectors) is selected as the current configuration. To calculate the inverse kinematics,

Denavit-Hartenberg parameters based on the angle definitions according to Paul given in [14] are used; however, the joint angles required by the control are defined according to the angle definitions by Armstrong given in [14]. Hence, the SRTK Simulink block-diagram converts the angles from the convention used by Paul in [14] to the convention used by Armstrong in [14].

Once the desired trajectory has been related to the joint space, the following PD controller is used to force the manipulator to follow the desired trajectory:

$$\tau = K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}) + \dot{q}_d + G(q) \quad (17)$$

where  $K_p, K_d \in R$  are constant control gains,  $\tau(q) \in R^6$  represents the joint control torque input,  $G(q) \in R^6$  represents the gravity effects, and  $q_d(\bar{t}), \dot{q}_d(\bar{t}), \ddot{q}_d(\bar{t}) \in R^6$  denote the desired position, velocity, and acceleration trajectories (defined in Armstrong's notation [14]) that are generated by the SRTK. When the manipulator is controlled in this mode, potential singularities may arise due to singularities in the desired trajectory that is obtained by the inverse kinematics of the Puma 560. However a marked advantage of this control mode over the Cartesian control mode is that since the control is not actually calculated in the Cartesian space, high torques will not be generated when the Puma is close to a singularity.

### 3.6 Impedance Control Mode

The impedance control mode is very similar to the Cartesian control mode, however, this mode provides for the additional capability of force feedback provided an auxiliary force/torque sensor is incorporated with the Puma 560. In the absence of force feedback (e.g., if no forces are present or if there is no auxiliary force/torque sensor present), a desired trajectory is generated by the SRTK in a similar manner as given in (5)-(10) that provides a smooth transition from the initial Cartesian position of the end-effector to the user-defined final Cartesian position. In the absence of force feedback, the same controller given in the Cartesian control mode is utilized to move the manipulator along the desired trajectory. The advantage of the impedance mode is that if forces are exerted on the end-effector, the desired trajectory of the manipulator is modified to reduce the forces.

Since the desired trajectory is generated in the base coordinate system of the manipulator, a rotation matrix is utilized to translate the forces measured at the end-effector to the base coordinate system. The rotation matrix is calculated online from the following equation:

$$f^* = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \begin{bmatrix} Cf_x \\ Cf_y \\ Cf_z \end{bmatrix} \quad (18)$$

where  $Cf_x(t), Cf_y(t), Cf_z(t) \in R$  represent the forces acting on the manipulator along the  $X, Y,$  and  $Z$  Cartesian coordinate axis. Note that it is assumed that the force-torque sensor is mounted so that it aligns with the robot  $X, Y,$  and  $Z$  directions when the robot is in the "ready" position so that the desired trajectory will change appropriately based on the force feedback information. Once the forces have been related to the base coordinate system of the manipulator, a filtering technique [13] is employed to offset the original desired trajectory as shown below

$$XYZ'_d = XYZ_d - \mathcal{L}^{-1} \left\{ \frac{f^*}{As^2 + Bs + K} \right\} \quad (19)$$

where  $XYZ'_d(t) \in R$  denotes the modified desired trajectory,  $XYZ_d(t) \in R$  denotes the desired trajectory in the absence of external forces,  $f^*(t)$  is defined in equation (18),  $\mathcal{L}^{-1}\{\cdot\}$  denotes the inverse Laplace transform operator,  $s$  denotes the Laplacian variable, and  $A, B, K \in R$  are positive constant filter parameters. The velocity profile of the desired trajectory is generated using the following equation:

$$X\dot{Y}Z'_d = X\dot{Y}Z_d - \mathcal{L}^{-1} \left\{ \frac{s f^*}{As^2 + Bs + K} \right\} \quad (20)$$

To force the Puma manipulator to follow the desired trajectory, a PD control law is applied to the first three joints of the Puma 560 as follows [13]:

$$\begin{aligned} \tau(XYZ) &= K_p (XYZ' - XYZ) + K_d (X\dot{Y}Z' - X\dot{Y}Z) \\ \tau(q) &= J^T \tau(XYZ) + G(q) \end{aligned} \quad (21)$$

where  $K_p, K_d \in R$  are constant control gains,  $\tau(XYZ) \in R^3$  denotes the torque control input in the Cartesian space,  $\tau(q) \in R^6$  represents the joint control torque input,  $J^T(t) \in R^{6 \times 3}$  represents the transpose of the Jacobian for the first three joints, and  $G(q) \in R^6$  represents the gravity effects. The control law given in (21) and (22) is subject to excessive torque commands due to potential singularities and aggressive trajectories in a similar manner as the controller given in (12) and (13); however, in this mode, potential high torques can also result from excessive force values.

### 3.7 Blender Mode

For each of the previously described modes the user provides the SRTK with the desired final configuration of the manipulator either in the Cartesian space or the joint space. Based on the desired final configuration, the SRTK generates a desired trajectory for the manipulator to follow. In contrast to the other modes, the blender mode provides the user with the ability to enter a list of (up to 7) desired positions for the manipulator, rather than requiring the user to enter the points one at a time. Based on the list of desired configurations, a desired trajectory is computed by the SRTK in the joint space to allow for a smooth transition between the points.

The desired trajectory, denoted by  $q_d(t) \in \mathbb{R}^6$ , can be divided in two separate stages: the transition phase and the constant velocity phase. The transition phase represents the trajectory when the robot is close to one of the desired positions and begins to transition towards another desired position. To characterize the transition points between the desired manipulator configurations, the following error functions are determined:

$$\Delta B = A - B \quad \Delta C = C - B \quad (23)$$

where  $A \in \mathbb{R}^6$  denotes the constant joint angle configuration that is present when the transition phase is initiated,  $B \in \mathbb{R}^6$  denotes the next desired joint configuration, and  $C \in \mathbb{R}^6$  denotes the joint configuration after position  $B$ . Based on the characterization of the transition points, the desired trajectory during the transition phase is determined as follows [15]:

$$q_d = \left[ \left( \Delta C \frac{t_{acc}}{T_1} + \Delta B \right) (2 - h) h^2 - 2\Delta B \right] h + B + \Delta B \quad (24)$$

$$\dot{q}_d = \left[ \left( \Delta C \frac{t_{acc}}{T_1} + \Delta B \right) (3 - 2h) h^2 - \Delta B \right] \frac{1}{t_{acc}} \quad (25)$$

$$\ddot{q}_d = \left( \Delta C \frac{t_{acc}}{T_1} + \Delta B \right) \frac{3h(1-h)}{t_{acc}^2} \quad (26)$$

where

$$h = \frac{t_{acc} + t}{2t_{acc}} \quad (27)$$

$t_{acc} \in \mathbb{R}$  is a positive constant user-defined acceleration factor that determines the amount of blending (or smoothing) between the points,  $T_1$  is the nonzero time required to travel from point  $B$  to point  $C$ , and  $q_d(t), \dot{q}_d(t), \ddot{q}_d(t) \in \mathbb{R}^6$  denote the desired position, velocity, and acceleration trajectories. A high value for relates to a smooth transition between the points with decreased accuracy (how close the manipulator comes to the actual point), whereas a lower value of relates to improved accuracy with a rougher transition between the points.

The most important parameter for the constant velocity phase is the user-defined maximum velocity. Given the maximum velocity and the maximum distance for a joint to move, the time required by the constant velocity phase can be determined. The equations used by the SRTK to generate the

desired trajectory for the constant velocity phase are shown below [15]:

$$q_a = \Delta C h + B \quad q_a = \frac{\Delta C}{T_1} \quad \ddot{q}_a = 0 \quad (28)$$

$$h = \frac{t}{T_1} \quad (29)$$

Since the trajectory generator requires three points (i.e.,  $A, B,$  and  $C$ ) potential problems may occur at the beginning and end of the trajectory generation. To overcome this potential problem, the first and last desired positions are repeated twice. To force the Puma manipulator to follow the desired trajectory, the same proportional-derivative control given in (17) is applied to each joint of the Puma 560.

### 3.8 External Mode

In each of the other modes (except the blender mode), a desired trajectory is generated by the SRTK to move the Puma manipulator from the current position to a user-defined final desired position. In contrast, the external mode provides the user with the ability to generate a desired position trajectory using MATLAB script files. When the user executes the MATLAB script file to generate the trajectory, the SRTK receives the information through the input section of the Simulink block-diagram. Since the desired trajectory is sent to the Simulink block-diagram at approximately 100 Hz and the SRTK operates at 1kHz, the desired trajectory must be interpolated so that intermediate points are generated for smoother operation. To this end, the desired trajectory is over-filtered by three, cascaded  $2^{nd}$  order low pass Butterworth filters with the following transfer function:

$$\frac{Y(s)}{U(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \quad (30)$$

where  $s$  denotes the Laplacian variable,  $\frac{Y(s)}{U(s)}$  denotes the filter transfer function,  $\omega_n \in R$  is the desired cut-off frequency and  $\zeta \in R$  denotes the damping coefficient. In conjunction with the filters, a differentiator is present to generate the velocity and acceleration profiles. These profiles are passed through saturation functions to limit the velocity and acceleration values to safe levels.

The control that is executed in the external mode is the joint space PD control law given below:

$$\tau = K_p (q_a - q) + K_d (filter(\dot{q}_a) - \dot{q}) + filter(\ddot{q}_a) + G(q) \quad (31)$$

where  $K_p, K_d \in R$  are constant control gains,  $filter()$  indicates a filtered signal,  $\tau(q) \in R^6$  denotes the joint torque control input,  $G(q) \in R^6$  represents the gravity effects.

## 4 Puma Simulator

The Puma simulator provides the user with a means to view a real-time graphical representation of the Puma movement based on the control input to the nonlinear dynamic model. The Puma simulator is a useful tool to determine if the torque control input contains singularities, results in a joint motion that exceeds the mechanical joint limit of the manipulator, or for detecting other unpredictable motions that may occur as a result of the applied strategy. The Puma simulator requires the user to specify the desired angle for each joint or the desired Cartesian position of the manipulator end-effector and a movement scale parameter that relates to the speed of the manipulator motion. When executed, the SRTK reads in the current position of each joint, and a trajectory is generated by the SRTK to enable each joint to reach the desired end position at the same time. The trajectory is then scaled by the user-defined movement scale parameter. A control law that is selected by the user from one of the previous control modes is then applied to the nonlinear dynamic model of the Puma 560 that is incorporated into the structure of the Puma simulator.

The nonlinear dynamic model used in the Puma simulator is given below [16]:

$$q = M^{-1}(q) [\tau - G(q) - C(q, \dot{q})\dot{q}] \quad (32)$$

where  $M^{-1}(q) \in R^{6 \times 6}$  denotes the inverse of the inertia matrix,  $C(q, \dot{q}) \in R^{6 \times 6}$  represents the centripetal-Coriolis matrix,  $G(q) \in R^6$  represents the gravity effects, and  $\tau(q) \in R^6$  denotes the joint torque control input. Once the control is applied, the joint acceleration, denoted by  $\ddot{q}(t) \in R^6$  is computed based on (32). The computed acceleration is then integrated to obtain the joint velocity and joint position terms where each joint is initialized to zero; hence, the Puma will always begin from the "Zero Position". The resulting motion of the Puma is then displayed in the GUI. Since the nonlinear dynamic model of the Puma is incorporated in the simulator, the displayed motion of the Puma should be very representative of the actual motion of the manipulator. In addition, the simulator display page can also be utilized to view the operation of the Puma during the actual implementation of the controller. This option may be an especially important feature for remote operation of the Puma. The display provided by the GUI can be set to show a stick-figure view or a 3-dimensional view of the manipulator motion, or simply the joint angles of the manipulator. The Puma simulator with the 3-dimensional view enabled is shown in Figure 1.

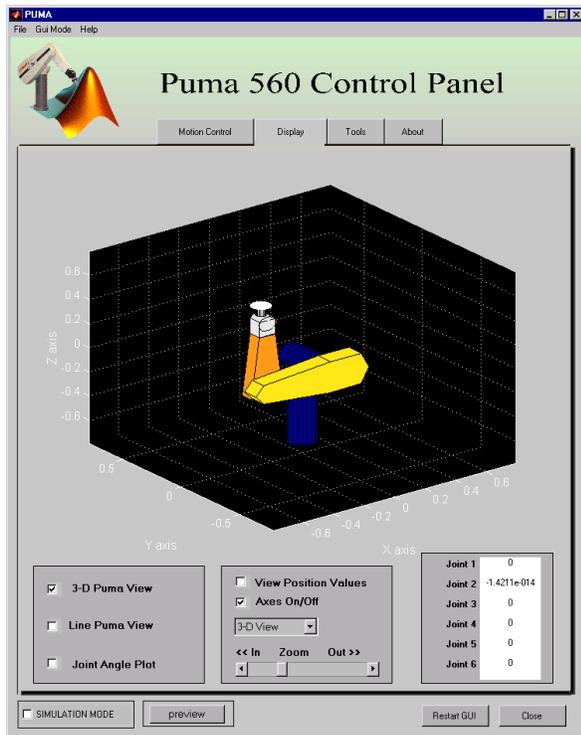


Figure 1: 3-Dimensional View of the Puma 560 from the Puma Simulator

## 5 Conclusion

In this paper a Robotic Toolkit for the Puma 560 robot manipulator was described. The advantages of the SRTK were described as: 1) real-time execution, 2) user-friendly GUI, 3) a basic framework that allows operation of the Puma and the freedom to target a specific application or research interest, and 4) ease of modification through the underlying Simulink block diagram. The hardware requirements were not described in this paper due to the fact that the Puma 560 has

recently become a turnkey system. That is, if a user has a Puma 560, a PUMA controller retrofit can now be purchased (see <http://pages.prodigy.net/tridentrobotics> for more information) that allows a user to simply connect the Puma and the controller and a PC together, and then run the supplied DOS programs to verify the hardware is properly operating. Once the user verifies that the hardware is properly operating, the SRTK can be installed on the PC and the system is complete. The complete SRTK system has been utilized by the Clemson University robotics and control program for several research and student projects. Future directions of the SRTK will include integrating additional robotic manipulators such as the Barrett Whole Arm Manipulator (WAM).

## References

- [1] J. Lloyd, M. Parker and R. McClain, "Extending the RCCL Programming Environment to Multiple Robots and Processors", *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 465 - 469, 1998.
- [2] P. Corke and R. Kirkham, "The ARCL Robot Programming System", *Proc. Int. Conf. Robots for Competitive Industries*, Brisbane, Australia, pp. 484-493.
- [3] M. S. Loffler, D. M. Dawson, E. Zergeroglu, and N. P. Costescu, "Object-Oriented Techniques in Robot Manipulator Control Software Development", *Proceedings of the American Control Conference*, June 2001, to appear.
- [4] D. J. Miller and R. C. Lennox, "An Object-Oriented Environment for Robot System Architectures", *IEEE Control Systems Magazine*, pp. 14-23, February 1991.
- [5] C. Zieliński, "Object-oriented robot programming", *Robotica*, Vol. 15, pp. 41-48, 1997.
- [6] C. Kapoor, "A Reusable Operational Software Architecture for Advanced Robotics", *Ph.D. thesis*, University of Texas at Austin, December 1996.
- [7] S. S. Ge, T.H. Lee, D.L. Gu, and L.C. Woon, "OpenRob: An Open-Architecture Platform for Model Building, Controller Design, and Numerical Simulation", *IEEE Robotics and Automation Magazine*, Vol. 7, No. 3, pp. 42-54, 2000.
- [8] Y.-C. Chen and J. M. Naughton, "An Undergraduate Laboratory Platform for Control System Design, Simulation, and Implementation", *IEEE Control Systems Magazine*, Vol. 20, No. 3, pp. 12-20, 2000.
- [9] J. N. Pires, "Interfacing Industrial R&A Equipment Using MATLAB", *IEEE Robotics and Automation Magazine*, Vol. 7, No. 3, pp. 32-41, 2000.
- [10] Z. Yao, N. P. Costescu, S. P. Nagarkatti, and D. M. Dawson, "Real-Time Linux Target: A MATLAB-Based Graphical Control Environment", *Proc. of the IEEE Conference on Control Applications*, pp. 173-178, 2000.
- [11] The MathWorks, 3 Apple Hill Drive, Natick, MA 01760-2098, <http://www.mathworks.com>.
- [12] Unimation Inc., *Breaking Away from VAL*, Danbury, 1982.
- [13] D. M. Dawson, M. M. Bridges, and Z. Qu, *Nonlinear Control of Robotics Systems for Environmental Waste and Restoration*, Prentice Hall, 1995, ISBN 0-13-124629-1.
- [14] P. Corke and B. Armstrong-H'elouvy, "A meta-study of PUMA 560 dynamics: A critical appraisal of literature data", *Robotica*, Vol. 13, No. 3, pp. 253-258, 1995.
- [15] R. Paul, *Robotic Manipulators: Mathematics, Control and Programming*, The MIT Press, 1981.
- [16] B. Armstrong, O. Khatib, J. Burdick, "The Explicit Dynamic Model and Inertial Parameters of the Puma 560 Arm", *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, pg. 510, 1986.
- [17] R. Voyle, *Mark V Automation TRC User's Manuals*, <http://pages.prodigy.net/tridentrobotics/manuals.html>, 2000.