The Use of MATLAB for Robotic Control in an Undergraduate Robotics Laboratory

Jenelle Armstrong Piepmeier, Kenneth A. Knowles, Bradley E. Bishop
U.S. Naval Academy
105 Maryland Ave (Stop 14A)
Annapolis, MD 21402

ABSTRACT

An effective undergraduate robotics course will have strongly coupled laboratory and classroom components. It is important that the students experience the application of classroom theory. Often, this application is transparent when using the vendor supplied programming environments. For example, Cartesian move commands will move the robot to a desired point in the workspace without a need for explicit solution of the inverse kinematics problem by the student. Programming environments such as MATLAB, Maple, or C\C++ have long provided an ideal simulation environment for studying kinematic or dynamic robotic models. Environments such as MATLAB are especially ideal for engineering students with limited programming expertise. By taking advantage of the serial port capabilities in MATLAB's Release 12 and later versions, along with the ability to compile existing C\C++ code under the MATLAB shell, the instructor can devise assignments that allow the student to easily model and control robotic systems in the MATLAB environment. This paper discusses two approaches and representative laboratory assignments.

I. INTRODUCTION

Robotic textbooks such as those by Niku [5], Spong [6], and Craig [1] present common topics such as transformations, inverse and forward kinematics, Jacobians, manipulator dynamics, and trajectory generation. Additional topics include control, sensors, vision, and artificial intelligence. The degree to which each of these topics is covered in a course depends on the level of the students and the departmental emphasis (electrical, mechanical, or computer science). Problem sets at the end of each chapter typically assign problems that are worked by hand. Craig [1] includes programming assignments such as software generation of forward or inverse kinematic functions. Few suggestions are given for laboratory exercises in any of these standard textbooks, primarily due to the wide variance in laboratory equipment and protocols typical in robotics education. Common assignments such as the *Towers of Hanoi* focus on algorithm flow, while other standard labs focus on behavior or path planning. These are all easily implemented using vendor supplied move commands.

A shortcoming in these common laboratory strategies is that they do not build on the introductory material that is emphasized in the classroom. This paper presents methods for utilizing the capabilities of MATLAB to quickly engage undergraduate students in an introductory robotics course. In order for us to adequately discuss these methods, Section II addresses the curriculum into

which they are integrated. Section III discusses a forward and inverse kinematics assignment using precompiled mex function for robot control. Section IV presents a Jacobian-based control assignment using the serial port capabilities of MATLAB.

II. BACKGROUND

Midshipmen in the Systems Engineering Major at the USNA take an interdisciplinary curriculum with an emphasis on control systems and dynamics. During their first-class (senior) year, they must select five technical electives demonstrating a concentration in at least two areas. The robotics track is one of the most popular options, comprising three courses. The first course emphasizes manipulators and machine vision, including coordinate transformations, forward and inverse kinematics, Jacobians, and simple image processing. The second course covers camera-robot calibration, visual servoing, and pattern recognition. These courses are three credit classes, including two hours of lecture and two hours of lab per week, with an open enrollment. The laboratory consists of ten robotic workstations outfitted with machine vision systems. We use both the SCORBOT ER-V and the ROBIX™ RCS-6 kits (See Figure 1). The third course, on mobile robotics, covers the design and implementation of various locomotive methodologies, closed-loop control systems, sensor suites, novel actuators and path planning techniques for mobile robots using the Parallax Basic Stamp II™ and the RCX microcontroller from the LEGO® MINDSTORM™ robotics development kit. This course is limited to one or two sections of about 18 students, providing three credit hours with one hour of lecture and four hours of lab. Programming environments for all classes include MATLAB™, Borland C/C++, PBASIC and NotQuiteC. A one semester programming course is a prerequisite for all of the robotics courses.

The curriculum that we utilize focuses on open-ended problems with more than one plausible solution. The use of reconfigurable kits (ROBIX and LEGO) allows rapid prototyping of solutions to challenging problems in a reasonable time frame while still maintaining technical rigor and appropriate level of intellectual challenge. In our framework, there is a strong coupling between lectures and laboratory exercises, allowing us to put to use in the laboratory all of the mathematical material presented in lecture. We have recently switched over to a classroom/laboratory hybrid structure, allowing us to move from lecture to experiment without undue interruption in the flow of the course.

III. SYSTEM MODELLING USING MEX FUNCTIONS

In commonly used robotic texts, the introductory chapter defines robotics and explores the role of robotics in industry and society. This is followed by transformation matrices and forward and inverse kinematics. Homework, quizzes, and exams are used to reinforce lectures on the topics. For some students, these three-dimensional topics are obvious, but for others it is impossible to relate the two-dimensional pictures to a three-dimensional reality. Furthermore, besides a poor grade, there is no immediate feedback on the consequences of incorrectly assigning coordinate frames or improperly deriving the inverse kinematic problems.

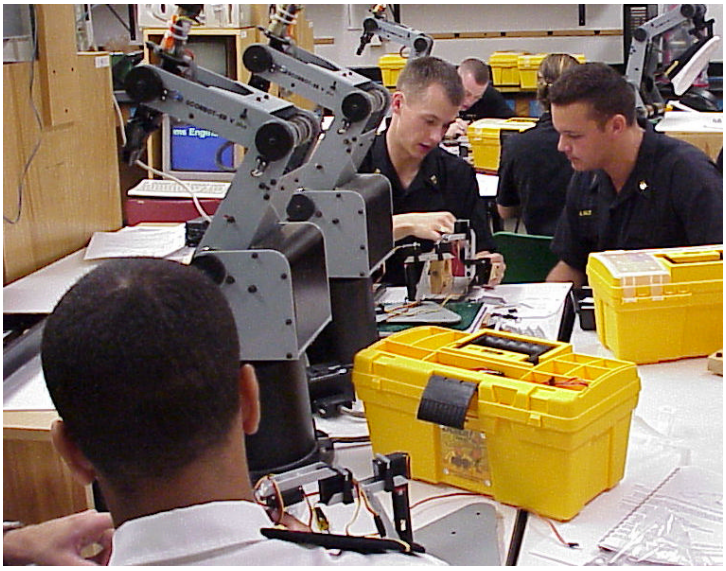

Figure 1 Students design and analyze a robotic manipulator for solving a test-bed problem on manipulable workspaces and repeatability, using magnets and metallic washers to simulate a spot welding process.

We have implemented two complementary laboratory experiments focusing on forward and inverse kinematics using inexpensive ROBIX RCS-6 kits. The manufacturer-supplied controller requires parallel port communication. While MATLAB does not support parallel port communications, the instructor can create mex-functions that implement the Robix supplied C-functions that can communicate with the robot.

In the assigned laboratory exercises, students built a simple 2-link manipulator. Using rulers and straightforward experiments to characterize the resolution of the servomotors, students developed a kinematic model of their robot. Coordinate frames were assigned, and link parameters were measured. Then, using MATLAB, students implemented robot control using the models that they had derived on paper, using a precompiled mex-function to control the robot from the MATLAB environment.

In the first kinematic laboratory assignment, students wrote a forward kinematics function. The function accepted joint values and returned position values. Using a precompiled mex-function, the students then commanded the robot to go to the same joint values. They were then required to visually verify that the robot's end effector was located at the Cartesian position that their forward kinematic function returned. Students quickly learned the consequence of assigning rotation axes that did not correspond with the positive direction of rotation for the motor they were using. The also learned that some link parameters needed to be negative, depending once again on the direction of their assigned coordinate frames.

For the next two-hour assignment, the students wrote an inverse kinematic function that accepted Cartesian endpoint positions and returned joint positions. They then used this function to calculate the joint values needed to draw as small shape on a piece of paper located in the robot's workspace. The precompiled move function was used to command the robot to draw the shape. During this laboratory exercise, the impact of mathematical errors is immediate and obvious. They

also saw that giving their inverse kinematic function a position outside of the robot's workspace resulted in complex joint angles.

The MATLAB environment allows students to focus on the kinematics rather than variable declarations or the details of matrix multiplication. Students can easily see the values assigned to variables, and quickly debug. Moving the robot by using the precompiled mex-functions links the theory to the physical system represented by their kinematic functions.

Mex functions allow the instructor to use vendor supplied C functions in the MATLAB environment. These can be used to access camera information or communicate with robotic hardware. The reader is referred to Mathwork's documentation [4] for detailed information, but the code shown in Figure 3 demonstrates a simple implementation. For brevity, variable declaration and robot initialization are omitted.

```
void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
// declare variables (omitted)

if(nrhs != 3)
mexErrMsgTxt("Three arguments required.");
ServoNum =  mxGetN(prhs[0]) ;
ServoList = mxGetPr(prhs[0]);
MoveList = mxGetPr(prhs[1]);
NumMoves =  mxGetM(prhs[1]);
delay= mxGetScalar(prhs[2]);

// initialize robot using Robix Rascal C functions (omitted)

// loop through desired move sequence passed from MATLAB (omitted)

int i,j;
for (i=0;i<NumMoves; i++)
 {
   for (j=0;j<ServoNum;j++)
   {
    // call the C function
    iRetCode = rbxServoJump(iRobotHandle,(int)*(ServoList+j), (int)*(MoveList+j*NumMoves+i)) ;
    Sleep(20);
   }
   Sleep(delay);
 }
// uninitialize robot using Robix Rascal C functions (omitted)

}
```

Figure 2 Sample mex function written in C. For brevity, variable declaration and robot initialization are omitted. When compiled as a rbxmove.dll, the student can call this function in MATLAB. Three arguments are passed to the function: a vector listing the servomotors to be moved, the absolute positions the respective servos should be moved to, and a time delay.

This example illustrates using both arrays and scalar values in a mex function. There are three arguments that the student passes to the function. The first is a vector containing a list of the servo motors that are commanded. The second is an array with each row representing the servo positions for a given move. Thus, the mex function moves the robot through a series of positions. The third

value is a time delay between moves. This function is included in a C/C++ file (with the appropriate headers included) and compiled in MATLAB using the *mex* command.

While our students have a background in programming in C, their programming skills are not necessarily up to the sophistication required to use the ROBIX-supplied functions. By using a mex function, we remove the programming hurdle and focus the student's attention on the fundamentals of robotics.

IV. ROBOT JACOBIAN CONTROL USING SERIAL PORT CONTROL

After studying forward and inverse kinematics, the students experimented with Jacobian control. This time, the five-degree of freedom SCORBOT-ERV Plus was controlled via MATLAB's Release 12 serial port capabilities.

In a two-week laboratory assignment, students first wrote a function to calculate and return the Jacobian matrix and then experimented with task-space control using the Jacobian. They were given an error tolerance, and had to experiment with how to use the Jacobian to move the robot in a straight line from point A to point B in the robot's workspace without exceeding the error tolerance.

Students used instructor provided functions that initialized serial port communications, sent move commands to the robot, and read in the current position of the robot. MATLAB's *plot3* command was used to plot the three-dimensional desired and actual positions of the robot's end effector. Figure 2 shows the results of robot control without updating the Jacobian. Students modified the control strategy to achieve straight-line motion within a given error tolerance.

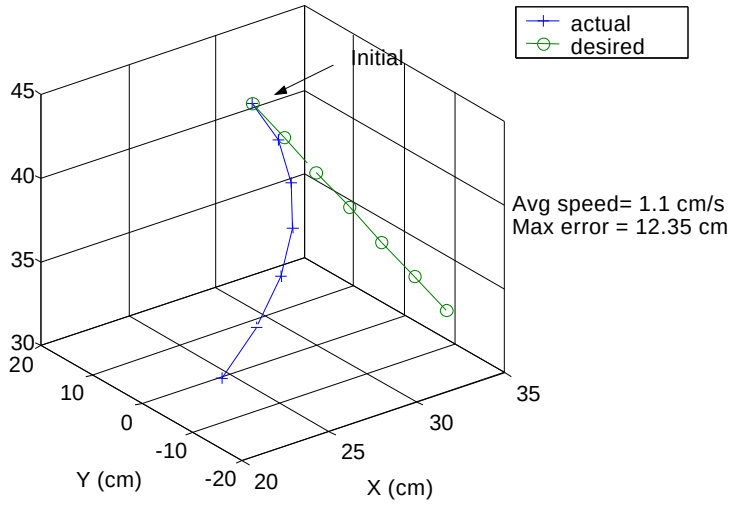

Figure 3 Sample plot showing actual and desired trajectories for the Scorbot end effector. The Jacobian was not updated at each step, so the error accumulated. Students modified the control strategy to reduce the error to within a given tolerance.

This lab helped students understand that the Jacobian changes throughout the workspace, and gave them experience in closed-loop control. Plotting the actual and desired robot positions provided visual feedback to the students on the effectiveness of their strategy. A downside is that the serial port communications are slow, and can require substantial error checking in the instructor provided functions.

Serial port control of robotics using MATLAB is not a recent development. Khepera robots by K-Team can be controlled serially in MATLAB using a K-Team developed toolbox [3]. Third-party software for serial port communication has been available for earlier versions of MATLAB as well. However, the advent of serial port objects in MATLAB's Release 12 provides a Mathworks supported method for communicating with peripheral devices such as robots. For both the student and the instructor, this is a significant improvement.

MATLAB's serial port interface provides direct access to devices that you connect to your computer's serial port. This interface is established through a serial port object. The serial port object supports functions and properties that easily allow the user to configure the communication protocol and read and write data to the serial port.

The reader is referred to the External Interfaces/API documentation for an introduction to serial port objects and communication [3]. Three main MATLAB functions are used: *serial*, *fprintf*, and *fscanf*. The *serial* function creates the serial object, and *fscanf,* and *fprintf* read and write strings to the serial port. Depending on the complexity of interfacing with the robot's controller and the students' MATLAB programming expertise, the instructor may want to provide additional functions that interface with the robot. Figure 4 shows sample MATLAB code for interfacing with the SCORBOT.

```
% Lab #6

% Create the serial object handle 's' and initialize the Scorbot.
s=initscor('Com2');

% Open the serial port.
fopen(s)

% To send simple commands such as open, close, or home,
% use fprintf.

fprintf(s,'open')
pause(1)
fprintf(s,'close')

% For more complicated commands sequences, use the instructor-written functions
% The function readpos returns two variables. It takes the serial handle 's'
% as an argument.

[q,x]=readpos(s)

% q is a 5x1 vector with the five encoder values for each joint.
% x is a 5x1 vector = [X; Y; Z; Pitch; Roll]
%  - where (X,Y,Z) are in cm
%  - where pitch & roll are in degrees

% To move the robot's joints, use the jointmove function.
% For example:

q=q-1000; % subtract 1000 from each element of q
movetime=2 %duration of move in seconds
```

```
jointmove(s,q,movetime)

% The first argument is the serial handle
% The second argument is the joint variables in encoder counts
% The third argument is the duration of the move (this affects speed)

% When you're done, close the serial port
fclose(s)
```

Figure 4 Sample MATLAB script file. The instructor supplied the functions *initscor, readpos,* and *jointmove*.

In addition to complete robotic workstations, the availability of inexpensive serial port controllers for remote control (RC) servomotors suggest some interesting possibilities for undergraduate design projects. For example, Ebert-Uphoff at the Georgia Institute of Technology has integrated servos and links from the Robix kits with Mondo-tronic's SSC2 servo control board for undergraduate study of parallel robots [2]. The control boards allow the students to control the robot via MATLAB with simple commands. Pontech's SV203 boards (with serial interface) combine both servo control and I/O capabilities with a *very* simple command set.

V. CONCLUSION

MATLAB has long been used for algorithm development, system simulation, and data analysis. This paper has examined interfacing with the manufacturer supplied robotic controllers using either mex functions or serial port objects. The undergraduate roboticist can experimentally verify simulations by controlling robotic devices within the MATLAB environment. While real-time control is not achieved, a stronger coupling between theory and application can be achieved.

VI. REFERENCES

[1] Craig, J. J., *Introduction to Robotics: Mechanics and Control*, 2nd edition, Addison-Wesley Publishing Company, New York, 1989.

[2] Robotics Center for Teaching, available at http://robot.me.gatech.edu/teaching_center.html

[3] K-Team Website, available at http://www.k-team.com/support/faqs.html#serial

[4] Mathworks On-Line documentation, available at http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml.

[5] Niku, S. B., *Introduction to Robotics: Analysis, Systems and Applications*, Prentice Hall, Upper Saddle River, NJ, 2001.

[6] Spong, M. W. and Vidyasagar, M., *Robot Dynamics and Control*, John Wiley & Sons, New York, 1989.

AUTHORS

JENELLE ARMSTRONG PIEPMEIER
Jenelle Piepmeier is an Assistant Professor in the Weapons and Systems Engineering Department at the United States Naval Academy. She received the B.S. degree in Engineering from LeTourneau University in 1993, and the M.S. and Ph.D. degrees in Mechanical Engineering from the Georgia Institute of Technology in 1995, and 1999, respectively. Her research interests include robotics, computer vision, and quasi-Newton optimization methods.

KENNETH KNOWLES
Kenneth Knowles is a Professor of Systems Engineering with the Weapons and Systems Engineering Department at the U.S. Naval Academy. Dr. Knowles received his BME, MME and Ph.D. degrees from the University of Virginia. His research interests are in machine vision and robotic system applications. He supports the NASA Hubble Space Telescope periodic servicing missions as Goddard Space Flight lead for the EVA support team.

BRADLEY E. BISHOP
Bradley E. Bishop is an Assistant Professor in the Weapons and Systems Engineering Department at the United States Naval Academy. He received the B.S. degree in Electrical Engineering from Michigan State University in 1991, and the M.S. and Ph.D. degrees in ECE from the University of Illinois at Urbana-Champaign in 1994 and 1997, respectively. His research interests include robotics, nonlinear and hybrid control, and computer vision.