AC 2011-1342: A MATLAB TOOLBOX FOR THE USB INTELLITEK SCORBOT

Joel Esposito, U.S. Naval Academy
Carl E. Wick, U.S. Naval Academy
Kenneth A. Knowles, U.S. Naval Academy

Professor Emeritus Weapons and Systems Engineering

The MATLAB Toolbox for the Intelitek Scorbot (MTIS): an open source educational robotics development library

Abstract - We present a MATLAB toolbox that interfaces directly with the Intelitek Scorbot – one of the most widely used educational articulated robots. The toolbox provides a user-friendly, open source method of accessing the robot's functionality from within MATLAB's powerful integrated development environment, which already includes numerical solvers, image processing routines, neural network libraries, and control system design tools. We describe the development process and the toolbox's features; and illustrate its capabilities with some projects from our own Introductory Robotics class where it was beta tested. A student opinion survey indicated that the toolbox was well received, but suggests its stability could be improved.

1. Introduction

It has been widely noted that engineering students benefit from a variety of teaching approaches, in particular visual and experiential learners prefer hands on laboratory experiences [1]. Teaching robotics is no exception [2, 3]. More recently there has been a movement toward developing and distributing free and open source software (FOSS) for robotics education and research [4, 5]. Inspired by these ideas, and the success of the MATLAB Toolbox for the iRobot Create [6] and Robotics Toolbox [7], we present a MATLAB toolbox that interfaces directly with the Intelitek Scorbot – one of the most widely used educational articulated robots.

For the past two decades the robotics educational scene has been dominated by the Intelitek Scorbot robot line. These devices have been among the most widely used tabletop articulated robot manipulators used for education.

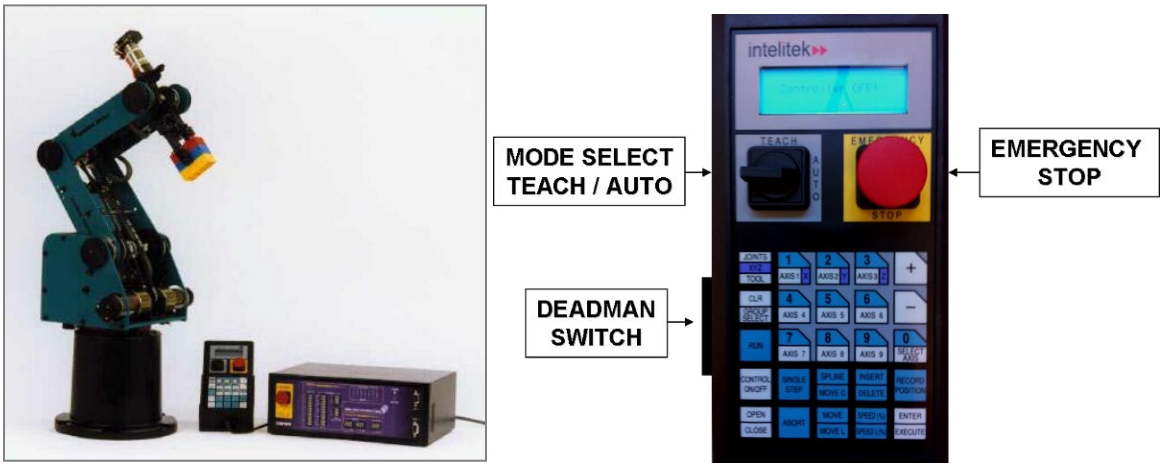


Figure 1: (left) The Intelitek Scorbot, with teach pendant and control box. (right) A close-up of the teach pendant.

Older versions of this robot were provided with a control box that could be controlled by either a "dumb" serial terminal or a personal computer (PC) via an RS232 serial port connection. A set of simple native ASCII commands were provided to control the robot's basic functions. While RS232 data transfer rates could be slow, it had two main advantages. First, the interface was driverless – making it platform independent. Therefore the robot could be controlled by any

terminal, PC, or microcomputer with a serial interface. Second, the relatively simple ASCII commands were completely transparent, making it relatively easy to develop open source libraries to control the robot from a variety of high level programming languages, such as C, C++, Python, or MATLAB. In particular, the authors have had many years of experience using MATLAB as the development environment of choice in their classes. MATLAB has been especially useful since robots are rarely stand alone systems. As such, they are frequently interfaced with existing image processing or path planning routines, which may likely have been written in one of these higher level languages.

In contrast, the latest Scorbot model, the ER 4u, is no longer controlled through an RS232 type serial link. The new interface requires connecting a PC to the control box via a Universal Serial Bus (USB). This provides superior data transfer rates, but requires a proprietary software environment (ScorBase) to control the robot.

This paper describes the development and use of a MATLAB Toolbox for the Intelitek Scorbot (MTIS). This Toolbox provides a series of seamless, high level MATLAB functions that can be used to control the Scorbot-ER 4u directly.

The remainder of the paper is organized as follows. Section 2 details our detective work for reverse engineering the required DLL files. Section 3 provides an over view of the MATLAB functions in the toolbox. Section 4 provides example code and projects from our undergraduate robotics laboratory at the U.S. Naval Academy. Section 5 presents some benchmark tests, to confirm the performance superiority of the USB interface to our previous RS232 interface. Section 6 concludes with the results from a usability survey we conducted from a test group of over 40 users. Appendix A provides installation instructions and compatibility information. Appendix B contains a Quick Reference of the toolbox's functions.

2. Reverse Engineering the Intelitek Dynamic Link Library Upon inspection we determined that the ScorBase Software provided with the Scorbot simply calls an Intelitek proprietary and undocumented dynamic link library (DLL) to communicate with the Scorbot's Control Box. Although a complete picture of the routines encapsulated in this DLL is not known outside of the manufacturer, there are web sites that share some incomplete knowledge about the DLL routines [8]. In addition, these primitive routines are very similar to those used with the old Scorbot.

However, our initial efforts to load the DLL into MATLAB failed. Unfortunately, it turned out that the new DLL was apparently written in C++ and was compiled in such a way that the library function names could not be successfully read by MATLAB.

The reason may be obscure to many non-computer scientists, and has to do with some of the flexibility afforded by programming languages like C++. One of these flexibilities is to be able to have several software routines with the same function name, but with different lists of parameters (similar to overloading). The compiler is left to decide which variant to use based on the parameters the programmer provides. This extension also allows for default parameters when incomplete lists are provided. For the compiler/linker to be able to decide upon which routine to use and how to handle parameters to the routines, each routine name must somehow have additional information available for the compiler/linker to use for proper selection. In C++

the term to describe this additional information is “name mangling”. Essentially, when C++ names are “mangled”, each routine name is changed in such a way that parameter information becomes part of the subroutine name. What is gained is that other C++ programs can potentially access variable class information directly and make correct decisions about which routine variant to use under any circumstance. What is lost in this process is unfortunately there is no universal way that name mangling is done. To be truly universal, a software package would currently have to be able to detect the method of mangling it sees from a large number of methods in use by different compiler manufacturers. The default taken by MATLAB (and probably many others) is to simply try to read names and give up if they are mangled. The end result is that we could load the DLL in MATLAB using the loadlibrary command, but no subroutine names could be found.

When the DLL was opened with a text editor (‘notepad’) a quick search for known DLL subroutine names quickly confirmed the use of name mangling. Consulting mangling techniques in use by several common compiler writers revealed that the technique was common to Microsoft compilers.

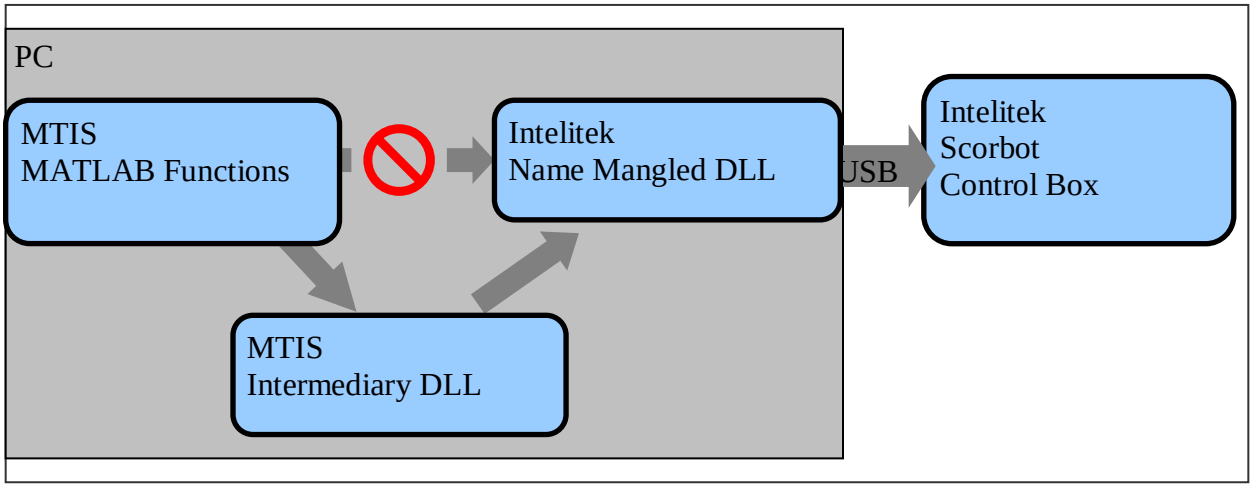


Figure 2: The calling structure of MTIS. MATLAB functions call an intermediate DLL, which calls the manufacturer’s name-mangled DLL to control the Scorbot via a USB connection.

After much thought, it was determined that it still could be possible to use the original Scorbot DLL routines if a second intermediate DLL was written and compiled without mangling the names (see Figure 2). This second DLL would, for the most part, simply pass parameters and call the mangled functions in the original, but by writing and compiling it with Microsoft C++, the second DLL would be also able to resolve the mangled names in the Scorbot DLL. If name mangling was ‘turned off’ when the second DLL was compiled, the routine names contained within the second DLL could be loaded into MATLAB.

Development of the new intermediate DLL proceeded by selecting from the known routines in the existing Scorbot DLL those routines normally used in our laboratory environment and for which sufficient knowledge was known to write an interface statement. As synchronization issues exist between robot motion and software (for example, robot motion must be completed before a new command is issued), several “Callback” functions from the original DLL were also used to help synchronize MATLAB calls with robot motion.

Parallel to this effort, it was discovered by examining software packages that came with the Scorbot, and with some trial and error, which directories the DLLs should reside in (see Appendix A for installation instructions).

3. Operating Principles

Once the DLL issue was resolved, we wrote a series of MATLAB "wrapper" functions that call the intermediary DLL (using MATLAB's callib function). Installation instructions for the DLL and wrapper function is given in Appendix A. A comprehensive list of toolbox commands and syntax is provided in Appendix B. This section provides an overview of some of the conventions and operating principles the toolbox employs.

Role of the Wrapper Functions: The wrapper functions are matlab files that call the DLLs. In some cases these functions simply replace particularly enigmatic or cumbersome library calls, with intuitively named functions. For example, setting the speed to 20% of maximum speed is much easier using the MTIC wrapper function, even though one could call the DLL directly without it.

Matlab's DLL Call: calllib('RobotDll' , 'RSetSpeed',int16(20))
Using MTIC: ScorSetSpeed(20)

In other cases, the wrapper functions perform more sophisticated roles, such as grouping together sets of library routines commonly called together, performing unit conversions, resolving sign inconsistencies, performing error checking or expanding the functionality of the original DLL.

Naming: Every function in the toolbox begins with the letters "Scor" to readily distinguish them from any built in MATLAB functions (ex. ScorInit, ScorHome, etc.). Commands that set properties of the robot have the word "Set" in them (ex. ScorSetSpeed); while ones that get sensor readings have the word "Get" (ex. ScorGetGripper).

Absolute vs. Relative Motion: All sensor/motion commands return/send absolute positions, unless otherwise stated. Relative, or incremental motions can be commanded using the version of the motion command with "Delta" in its name (ex. ScorCartMove *vs.* ScorDeltaCartMove)

Units: All commands use Centimeters and Degrees to indicate position and angles respectively.

Coordinates: The tool box uses two types of coordinates: *Cartesian* and *Joint*, as illustrated in Figure 3. *Cartesian* coordinates are specified as a 5 X 1 vector of the form [X Y Z Pitch Roll]. Commands that utilize Cartesian coordinates contain the characters "Cart" in the function name. (ex. ScorCartMove([X Y Z P R]). Joint coordinates are specified as a 5 X 1 vector of angles the form [Base Shoulder Elbow Pitch Roll]. Commands that utilize Joint coordinates contain the characters "Joint" in the function name (ex. ScorJointMove([X Y Z P R]) . Users should note that Intellitek's DLL is internally inconsistent in that the positive sense of the angles does not match the positive sense used by the teach pendant. The toolbox corrects for this mismatch – matching the labeling of the teach pendant.

Also, note that Pitch is defined differently in these two coordinate systems. Cartesian pitch is the orientation of the end effecter relative to the horizontal plane, whereas the joint angle version of pitch is defined as the relative angle between the forearm and the end effecter.

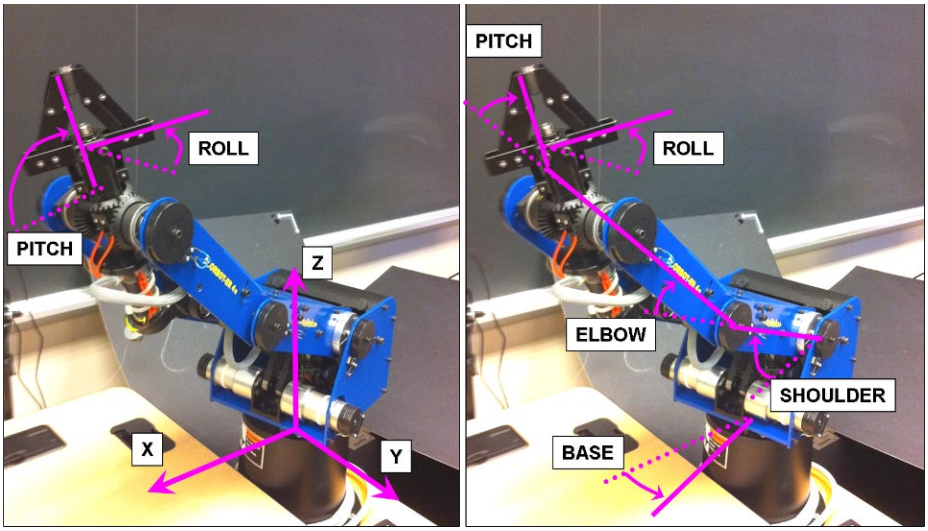


Figure 3: Illustration of the two coordinate systems used in the toolbox. (left) Cartesian coordinate system (XYZPR) and (right) Joint coordinate system (BSEPR). Note that pitch is defined differently in the two coordinate systems, but the roll angle is identical.

Teach Pendant Modes: There is a dial on the teach pendant to switch between "Teach Mode" (in which the teach pendant controls the robot) and "Auto" mode (in which the Control Box as authority over the robot). It turns out that the Scorbot cannot properly confirm that a motion is completed in Teach mode. The toolbox automatically detects which mode the pendant is in and asks the user to switch modes as appropriate.

Confirmation: Most commands in the tool box have an optional output argument called "confirmation", which indicates the command was successfully executed (confirmation =1) or not (confirmation = 0). For example a command to move the robot outside the boundaries of its workspace will result in failure

```
>> confirmation = ScorCartMove( [1000 0 0 0 0])
confirmation = 0
```

When in teach mode, the proper execution cannot be determined, and the value of confirmation is set to -1.

Command Blocking: It is possible to send commands to the robot much faster than it can physically execute them. This creates the possibility of missed commands if the buffer overflows. To remedy this, the toolbox blocks MATLAB from sending new commands to the Scorbot until two conditions are met. First, Intelitek's DLL contains a function called 'RmotionIsDone' which, despite its name, returns a Boolean indicating if the robot has *received* a command – not completed it. Second, we also block the sending of new commands until the Scorbot is within 5 millimeters of the commanded pose (this tolerance can be adjusted).

Kinematic Functions: Several functions are provided with the toolbox to solve typical kinematic problems in articulated robotics: forward kinematics, inverse kinematics and translational velocity Jacobian. These functions do not interact with the hardware in anyway, and are independent of the DLLs. Educators may wish to remove them from their installation if they want students to solve these problems themselves.

4. Example Program and Projects

An example of a very basic MATLAB program which uses the MTIS wrapper functions to pick up an object and determine its size is included below**.**

```
ScorInit;                          % Loads the DLL and initializes USB
ScorHome;                          % The Scorbot must be homed before beginning
ScorSetSpeed(80);                  % Set speed at 80 percent of max

ScorCartMove( [ 40 0 30 -90 0] );  % Moves to a position 40 cms in front of robot
%       with end effector pointing down
ScorSetGripper( -1 );              % Opens gripper fully

ScorDeltaCartMove( [0 0 -10 0 0]); % Moves down 10 cms  to pick up object
ScorSetGripper( 0 );               % Closes fully around object
ObjectWidth = ScorGetGripper();    % Get width of object in gripper in cm

fprintf('The object is %f centimeters wide. \n', ObjectWidth)
```

The following are some additional laboratory exercises, taken from our introductory robotics class, to illustrate other functionalities of the toolbox.

Example Project #1: Towers Of Hanoi

Students program the Scorbot to solve the classic puzzle (Figure 4 left) which involves moving a stack of rings/blocks from one peg to another according to specific rules. Typically students use the teach pendant and the ScorGetXYZPR command to record the location of the three pegs. ScorDeltaCartMove is used to complete the vertical motions, and ScorGetGripper can be used to sense which, if any, of the three blocks is currently in the gripper.

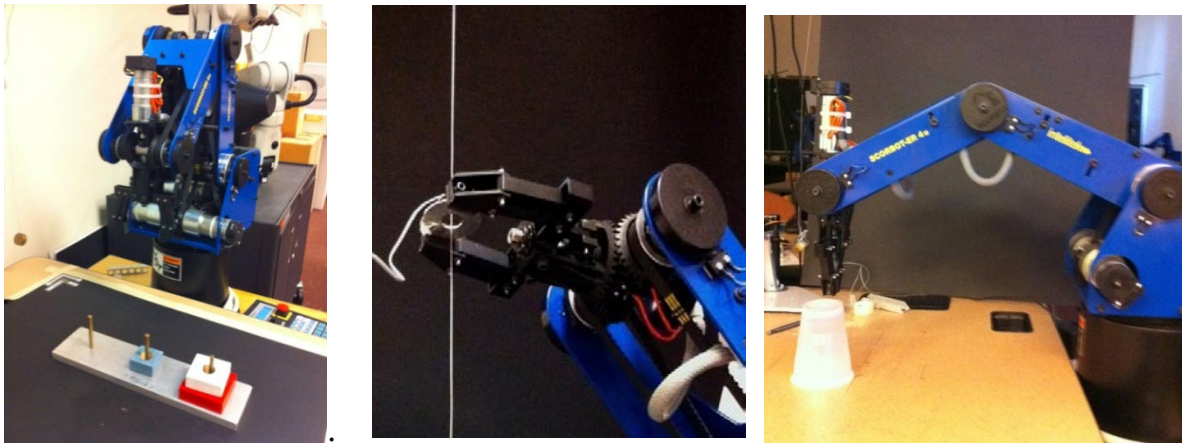


Figure 4: Three example projects: Towers of Hanoi Puzzle, Defusing an IED, and Cup Crushing.

Example Project #2: Defusing an IED (or Hasboro's "Operation" game)
Students attempt to thread a ½" inner diameter washer, along a taut vertical wire (Figure 4 center). A small pager motor and a 9 volt battery were rigged to buzz if the washer accidentally touches the wire. Students use ScorGetBSEPR to get the joint angles and ScorJacobian to compute the Jacobian matrix. They use the inverse of the Jacobian to determine how to move the end effector in a straight line using only ScorDeltaJointMove.

Example Project #3: Cup Crushing
Students use the Scorbot's end-effector to crush inverted plastic cups (Figure 4 right). While crushing a single cup is trivial, stacks of two or more cups can only be crushed in certain optimal locations in the workspace due to current limitations (thus maximum torque available) on the Scorbot's motor drivers. Students write a program which samples points in the robot's workspace, computes the joint angles using ScorInvKin, and the Jacobian with ScorJacobian. They find the location which maximizes the robot's mechanical advantage, and then crush plastic cups using ScorDeltaCartMove.

5. Benchmarking

We did three benchmark tests to compare the old RS232 interface with the USB interface:

1. *Encoder Reading:* Mean time to measure the joint angles (ScorGetBSEPR), across 1000 trials.

2. *Movement Time*: Mean time to execute a 10 cm vertical motion with a desired movement time of 1 second: ScorSetMoveTime(1); ScorDeltaCartMove([0 0 10 0 0]). A stopwatch was used to record the motion duration. Averaged across 60 moves.

3. *Sequences of Motions:* A test program sent the Scorbot a rapid sequence of 100 random motion commands. We manually recorded how many were missed.

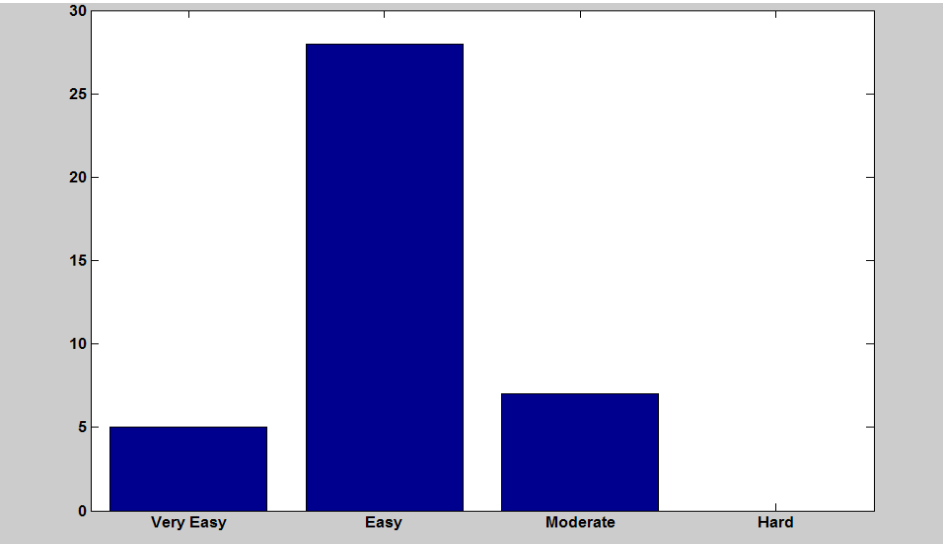
	RS-232 interface	MTIS
Encoder Reading (avg time in seconds)	0.12	0.02

Movement Time (target is 1 second**)**	1.24	1.01
Missed Motions (of 100)	12	0

As expected, improvement in performance is indicated in every test. In the case of the encoder, since the hardware is essentially identical, we conclude the difference in times is attributable to the time it takes to transmit the data over the RS232 (19200 baud) *vs.* the USB interface. In the case of the Movement time, since the hardware is identical, the difference is likely due to the data transfer rate, but may also depend on some of the inverse kinematic computation in the Scorbot's control box as well. However, we note that the additional time of 0.01 is within the margin of error of the stop watch. For us, one of the most significant improvements is that there were no missed motions by the Scorbot. Using the older RS232 interface, missed motions – or worse yet crashing the robot – was a relatively common occurrence. Students often inserted "pause" commands in their code, in an ad hoc manner, to avoid this. This improvement is most certainly due to the improved error handling of the toolbox.

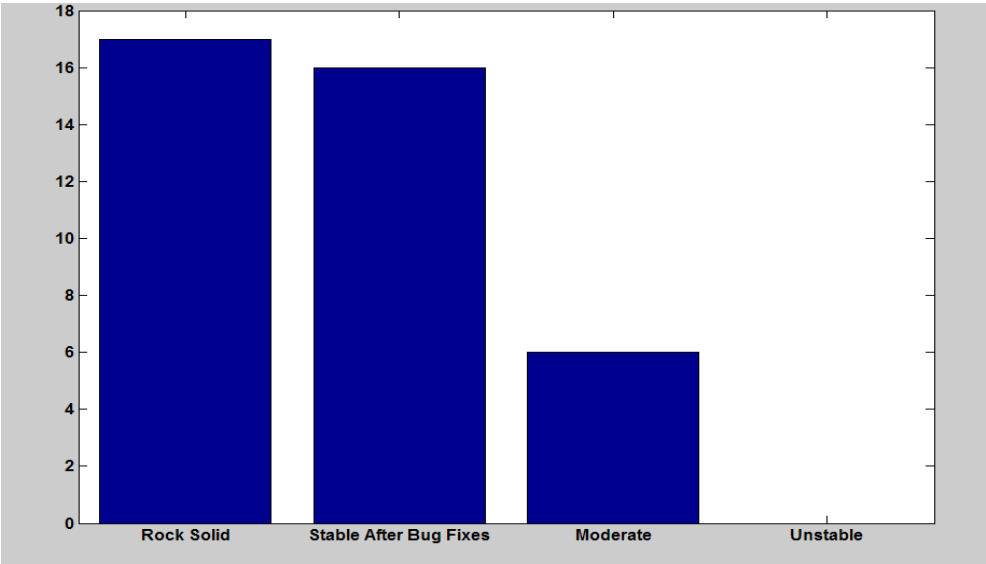
6. User Survey

The resulting MTIS package presents a "seamless" interface to the student and can be maintained within the MATLAB environment. As many software developers know, one of the best places to test software is in a student laboratory setting. The transition to these new robotic arms has been incredibly smooth. At the end of a semester-long course, which included eight 2-hour laboratory exercises involving the Scorbot, students (N=42) were asked to rate how easy it was to use the Toolbox (Figure 5) and how stable the Toolbox was (Figure 6), according to a predefined scale.



Very Easy	Easy	Moderate	Hard
The function names were self explanatory, no further documentation needed.	The help file resolved any confusion.	Some help files confusing or inconsistent.	After reading the help file, I still could not figure it out.

Figure 5: Student users of the toolbox (N=42) were asked: "How easy was it to learn how to use the functions in the toolbox?" According to the scale provided, the vast majority rated it as "easy".



Rock Solid	Issues in Early Labs	Moderately Unstable	Unstable
Rarely crashed.	Rarely crashed after bug fixes were installed.	Frequently crashed after Bug fixes installed.	Crashed so frequently I could not complete the assignments.

Figure 6: Student users of the toolbox (N=42) were asked: "How stable was the toolbox?" According to the scale provided, most rate it "Rock Solid" or "Stable After Bug Fixes".

Regarding ease of use, Figure 5 shows that the vast majority of student rated it "Easy". We are satisfied with this result, considering that the survey subjects are novice programmers / MATLAB users. Informally, we would guess that if the same students were asked to rate MATLAB's overall ease of use, they would likely rate it "moderate" on average.

Regarding stability, Figure 6 shows that all but 6 of the students were happy with the stability by the end of the semester. The initial deployment of the toolbox did not perform some of the error checking discussed in Section 3. In particular, it did not force the user to switch the teach pendant to Auto mode, nor did it check to see if the motion was complete before allowing new commands to be sent to the robot. These changes were added after the second of eight laboratory exercises, and had a notable impact on stability. However, even after the bug fixes, six of the 42 students still rated the stability as "Moderate – Frequently Crashed even after bug fixes were installed". Unfortunately, we were unable to replicate these issues and are currently investigating their cause.

7. Conclusion

In conclusion, we have provided a MATLAB interface consisting of a DLL and a set of documented wrapper functions that allow one to control the Intelitek Scorbot ER-4U from within MATLAB's integrated development environment.

There are several advantages to this.

1. The toolbox takes care of the low level hardware interfacing, allowing instructors who do not possess such programming expertise to offer laboratory based exercises in their classrooms.

2. The online distribution of this material frees researchers to focus their time on algorithm development instead of reinventing the low level interfaces.

The functions are freely available on the web. While the toolbox is free, of course MATLAB itself requires a license to use. In that sense it is not FOSS (Free and Open Source Software). However, we feel MATLAB is a viable alternative since:

1. many universities, the target market for the Scorbot, already possess a site license for MATLAB;

2. MATLAB, as compared to Python or Java, is frequently taught to non-computer science majors such as mechanical engineers; and finally

3. MTIS allows one to integrate the Scorbot with MATLAB's other powerful toolboxes such as Neural Networks, Control Systems, and Image Processing as well as third party toolboxes such as the Robotics Toolbox [7].

Regarding avenues of future work, one is to investigate the cause of occasional crashes. A second major thrust is to expand the capability of the toolbox to include velocity-based control commands. Finally, we plan to investigate the compatibility of the toolbox with Octave and other freeware MATLAB clones.

APPENDIX A: Getting Started

Version Compatibility

The toolbox is developed for the Intelitek ER-4U with USB connection. The Scorbase software version was 5.3.3.2. The control box software was version 14.

The toolbox was tested on MATLAB R2009B, but is likely to work on releases dating back to 2007. It is likely to work on any 32 bit version Windows based operating system. It is unlikely to work on 64 bit operating systems or Apple operating systems. It has not been tested on Octave or other freeware MATLAB clones.

Installation

1. Download and unzip the files from
   http://www.usna.edu/Users/weapsys/esposito/scorbot.matlab/

2. Copy the contents of the *ToMATLABBinWin32* folder and paste them into your MATLAB's *BinWin32* directory

3. Copy the contents of the *ScorbotMATLABFiles* in a convenient and accessible location.

4. In MATLAB add the folder from step 3 to the path. The path is the list of directories MATLAB looks at for function definitions. For example, if you placed that folder in C:\MATLAB\ScorBotToolBox
   >> addpath 'C:\MATLAB\ScorBotToolBox';

   You will need to do Step 4 each time you restart MATLAB. If you want the change to be permanent add:
   >> savepath;

Troubleshooting

All of these things have to be true in order to move the robot. If your robot is not working, find and check each of them.

Software: Open MATLAB, and type `>> ScorInit;` MATLAB will prompt you to put the teach pendant in Auto Mode;

Emergency Stop: The  button on teach pendant is raised;

Emergency Stop: The  button on control box is raised; and

Deadman Button: depressed *or* Teach pendant is in the magnetic holster.

TIP: Mnemonic SEED

APPENDIX B: Quick Command Reference

Command Syntax	Description
ScorInit	Loads DLLs, Enables Motors, Sets up USB Communications
ScorHome	Homes all joints
Confirmation = ScorControlEnable(OnOff)	OnOff=1 turn on all motors, 0 Off Useful to recover after impacted axis errors. Can also be done on Teach pendant. CONTROL ON/OFF <ENTER>
Confirmation = ScorSetMovetime(tsec)	Sets time limit to execute move in seconds -- effectively dictating speed. A very low time can cause moves to fail (Confirmation = 0 after motion commands).
Confirmation = **ScorSetSpeed** (PercentSpeed)	Set speed to an integer between 1 and 100. Units are percent of max. Optional output argument is 1 if command successful and 0 if it fails.
BSEPR = ScorGetBSEPR	Returns current position as BSEPR= [Base Shoulder Elbow Pitch Roll] degs 1X5 vector note .
XYZPR = ScorGetXYZPR	Returns current position as XYZPR = [X Y Z Pitch Roll] cms / degs 1X5 vector.
confirmation= ScorCartMove (XYZPR)	Moves the end effector in a straight line, from current position to XYZPR = [X Y Z Pitch Roll] in centimeters and degrees respectively. Will fail if move is outside of workspace. Note XYZPR must be 1 X 5.
confirmation= ScorDeltaCartMove(deltaXYZPR)	Incrementally moves the end effector in a straight line, by deltaXYZPR = [deltaX deltaY deltaZ deltaPitch deltaRoll] in centimeters and degrees respectively. Will fail if move is outside of workspace (confirmation =0). Note deltaXYZPR must be 1 X 5.
confirmation = **ScorJtMove**(BSEPR)	Moves the end effector, from current position to BSEPR= [Base Shoulder Eblow Pitch Roll] defined in degrees. Trajectory is a straight line in joint space (apparently arced in Cartesian space). Will fail if move is outside of workspace. Note BSEPR must be 1 X 5.
confirmation = **ScorDeltaJtMove**(deltaBSEPR)	Incrementally moves the end effector from current position by deltaBSEPR= [deltaBase deltaShoulder deltaElbow deltaPitch deltaRoll] defined in degrees. Generally deltas should be small (<<180). Note BSEPR must be 1 X 5.

confirmation=**ScorSetGripper**(cm)	cm=-1: Open gripper cm=0; Close gripper cm = 1-7 centiimeters, resolution of 1mm Does not account for width of pads (about 3 mm).
cm= ScorGetGripper	Returns distance in centimeters gripper is currently open.
J=ScorJacobian(BSEPR)	Linear Velocity Jacobian (3 X 5 Matrix). Given joint angles [Base Shoulder Elbow Pitch Roll] (deg) computes Jaconian. Units are centimeters
XYZPR=**ScorFwdKin**(BSEPR)	Forward Kinematic Solution. Given joint angles [Base Shoulder Elbow Pitch Roll] (deg) computes [X Y Z Pitch Roll] (cms and degs)
BSEPR=**ScorInvKin**(XYZPR)	Inverse Kinematic Solution. Given [X Y Z Pitch Roll] (cms and degs) compute joint angles [Base Shoulder Elbow Pitch Roll] (deg)
confirmation = **ScorAddToVec**(Pt,XYZPR)	Used by other commands. Save a pose as point # Pt (1 to 999). Pose specified in XYZPR = [X Y Z Pitch Roll] format (cms, degs). Must be a 1X5. See ScorGetXYZPR. Will fail if point I outside of workspace. Not recommended as a method to store points since points are erased if robot rebooted.
[XYZPR, confirmation] = **ScorCapturePose**(Pt)	Record current pose as point # Pt (1 to 999). For your reference, it returns X,Y,Z in cm and Roll, Pitch in degrees. .Not recommended as a method to store points since points are erased if robot rebooted. Note that teach pendant should be in Teach Mode.
confirmation= ScorMoveToPt(Pt,LorJ)	Moves to a previously recorded point Pt (integer from 1 – 999). See ScorCapturePose. 'L' produced a straight line motion. 'J' moves in a straight line in joint space which produces curved paths.
BSEPR=ScorCnts2Deg(cts)	Converts 1X5 vector of encoder counts to a 1X5 vector of joint angles [Base Shoulder Elbow Pitch Roll]
CNTS=**ScorDeg2Cnts**(BSEPR)	Converts a 1X5 vector of joint angles [Base Shoulder Elbow Pitch Roll] to a 1X5 vector of encoder counts to a

References

[1] R.M. Felder and R. Brent, "Understanding Student Differences" *Journal of Engineering Education, 94* (1), p. 57-72, 2005

[2] K. Nagai, "Learning While Doing: A Practical Robotics Education", IEEE Robotics and Automation Magazine, p 39 -43, June 2001

[3] J.A. Piepmeier, B.E.Bishop, and K. A. Knowles, "Modern Robotics Engineering Instruction", IEEE Robotics and Automation Magazine, p. 33-37, June 2003

[4] G.R. Bradski and A. Kaeller, *Learning OpenCV---Computer Vision with the OpenCV Library*, O'Reily Media, 2008

[5] S.Cousins, B.Gerkeym K.Conley, and Willow Garage, "Sharing Software with ROS", IEEE Robotics and Automation Magazine, June 2010, p 12-14.

[6] J.M. Esposito, J. Kohler, and O. Barton, "MATLAB Toolbox for the iRobot Create (MTIC)", www.usna.edu/Users/weapsys/esposito/roomba.MATLAB/ 2008

[7] P. Corke, "MATLAB Toolboxes: Robotics and Vision for Students and Teachers", IEEE Robotics and Automation Magazine, p 16-17, December 2007

[8] J.C. Mojebo, "The Scorbot ER4U function reference and notes for the usbc.dll", www.theoldrobots.com/book45/USBC-document.pdf