Genetic Algorithms with Lego Mindstorms and Matlab

Frank Klassner[1] James C Peyton-Jones[2] Kurt Lehmer[1]

[1]Center of Excellence in Enterprise Technology, Computing Sciences Department, Villanova University
[2]Center for Nonlinear Dynamics and Control, Villanova University
{frank.klassner,james.peyton-jones,kurt.lehmer}@villanova.edu

Abstract

This paper presents a case study in combining Lego Mindstorms™ NXT with Matlab/Simulink to help students in an undergraduate Machine Learning course study genetic algorithm design and testing. The project uses the VU-LRT toolbox to enable students to access the hardware capabilities of the Mindstorms platform from within Matlab. The course's enrollment was comprised of students from several majors with a variety of programming backgrounds. The course is part of an interdisciplinary cognitive science concentration. We report on the VU-LRT toolbox, the considerations imposed by the diversity of the student population on the design of the laboratory module and student evaluations of the laboratory module.

Introduction

Robotic systems today are enjoying an increased consideration in education as an "electronic tangible" that can provide students in computing-related fields with active-learning experiences. For some time, researchers have found that students' motivation to learn can be increased significantly with hands-on robotics-based projects, [1,2,3]. Others have successfully used robotics as a unifying theme in introductory courses, [4,5], and still others have used robotics as way to attract women into Computer Science [6]. These findings, as well as improvements in economies of hardware design, have led to the design of educational robotics platforms with increased computing capabilities.

The increase in embedded computational power, however, is also strongly correlated to programming complexity. In undergraduate artificial intelligence courses, particularly in liberal arts institutions, there is a balance that must be maintained between presenting all the programming details behind the low-level source code for controlling a robot and presenting the general techniques being taught with the robot. Especially in courses with students from several disciplines (a not uncommon occurrence in AI courses!), the instructor who chooses to use embedded programming experiences on robots should benefit from a programming tool that is flexible enough to provide both text-based programming experience for students who seek to understand the details of programming a robot platform with all of its foibles and graphical programming for those seeking a more high-level understanding of AI techniques.

We present a case study combining Lego Mindstorms™ NXT [13] with Matlab/Simulink to help students in an undergraduate Machine Learning course study genetic algorithm design. The course is part of a Cognitive Science concentration program as well as a Computer Science elective. Often students in fields such as psychology, biology, and philosophy take the course. The course requires that students have a semester of programming experience, but in practice this experience varies. Some students have Java training, while others have C training. Still others have graphical programming experience. This project uses the VU-LRT toolbox [8] that was developed to enable students to access the hardware capabilities of the Mindstorms platform from within Matlab at two levels: the hardware-oriented textual programming level and the abstraction-oriented Simulink level.

The paper presents a summary of the VU-LRT Toolbox, a description of a genetic algorithm lab that uses the VU-LRT, and student evaluations of the project.

The Matlab/Simulink VU-LRT Toolbox

The design of the VU-LRT toolbox for supporting embedded programming projects on the Lego Mindstorms NXT platform was informed by a number of high level design tools currently available for supporting embedded programming on robotic platforms. To varying degrees, these tools allow users first to simulate their designs, then implement them on target hardware, and finally to tune system parameters while the code is actually running on

the target. This development cycle is both practical and educational and is widely used in industry. Specifically, these tools include Microsoft Robotics Studio (MSRS), LabView from National Instruments, and Matlab / Simulink from the Mathworks. The Matlab / Simulink environment is very pervasive in the STEM community, and was already tightly integrated into the research activities and educational curriculum at Villanova University where VU-LRT was developed. Simulink was therefore chosen as the design environment for VU-LRT.

The use of Matlab and Simulink for educational robotics applications is not new. Dr. Behrens, from the Institute of Imaging and Computer Vision, in Aachen, Germany developed the RWTH toolbox for wirelessly sending commands and receiving data to/from a LEGO NXT platform [7]. This 'remote control' approach has the control algorithm running on the host PC with the robot acting primarily as a dumb sensor / actuator. Though simple, the approach is limited to low bandwidth control applications due to the time varying delays which inevitably occur in the host-target communication channel. A truly embedded / real-time solution, very similar to that advocated in this paper, has also been developed by T. Chikamasa in the form of the Embedded Robot (ECRobot) coder [12]. The ECRobot toolbox represents a significant advance from hand-coded algorithm implementation and has been used by the authors and others in earlier projects [13,14]. However its function-call based architecture does not conform to the normal Simulink Real-Time Workshop (RTW) design process, and it imposes significant constraints on user designs. The VU-LRT toolbox aims to provide a more user-friendly blockset and to integrate more seamlessly with the standard RTW design process. However, it still builds on the same real-time operating system used by ECRobot and has benefitted considerably from this earlier work.

The Villanova University LEGO Real Time target (VU-LRT) provides a blockset and toolchain to enable target implementation of high-level Simulink designs on the NXT brick. The blockset defines a high-level interface to NXT hardware for users, as well as the target-specific low-level code and cross-compilation details necessary to implement this in the final executable. The non-target-specific parts of the code generation and the overall 'build' process are handled by the MathWorks Real Time Workshop (RTW) toolbox. The process is illustrated in Figure 1. The start point is a user design in the form of a Simulink model. When the user initiates a 'build' command, the Real Time Workshop automatically generates the corresponding C code, as well as a 'makefile' which defines how to cross-compile this code into a real-time executable. The VU-LRT toolbox provides the NXT-specific template that is used by the Real Time Workshop to generate the makefile, as well as the target specific code needed to access the various NXT input / output devices. The combined code is then automatically cross-compiled and linked into an executable which is downloaded to the target and run.

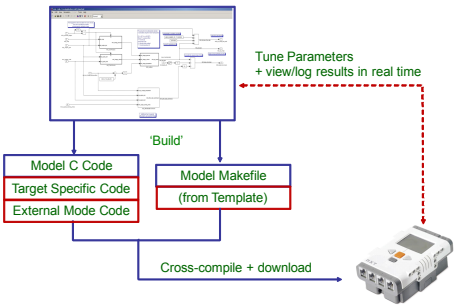

Figure 1. The Rapid Prototyping Process

A variety of third party, public domain tools are required to perform the cross-compilation and download operations outlined above, and the executable runs under a real time OSEK operating system that has been developed for the NXT [19]. The installation of these tools can be complex, so the VU-LRT toolbox ships with an automatic installer for the entire tool set. The VU-LRT toolbox (and installer) is currently available for download from the MathWorks fileshare site [16].

From the user perspective, the build and compilation details shown in Figure 1 are largely transparent, and target specific features encapsulated within the VU-LRT blockset shown in Figure 2.

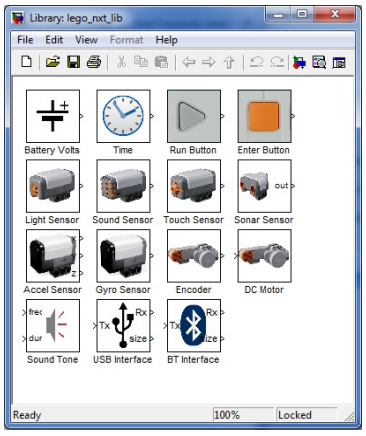

Figure 2. The VU-LRT Blockset

The user can drag and drop any of these blocks into their design in order to access any of the NXT's in-built features or attached sensors and actuators. A summary of these blocks and their function follows:

- Battery Volts – outputs the current battery voltage
- Time – outputs the time in ms since the model execution began

- Run Button – outputs a 1 if the 'Run' button is pressed, else 0
- Enter Button – outputs a 1 if the 'Enter' button is pressed, else 0
- Light Sensor – outputs a measure of the light received by the light sensor
- Sound Sensor – outputs a measure of the sound intensity received by the sound sensor
- Touch Sensor – outputs a 1 if pressed, else 0
- Sonar Sensor – outputs the distance to the closest object in view by the sonar sensor
- Acceleration Sensor – Outputs acceleration data in three axes (x, y and z)
- Gyro Sensor – outputs the rotation rate of the sensor
- Encoder – outputs the number of encoder pulses received as the motor rotates
- DC Motor – sets the applied motor voltage as a percentage of battery volts
- Sound Tone – sets the frequency and duration of tones driving the internal loudspeaker
- USB Interface – enables the model to communicate with a host PC over USB
- BT Interface – enables the model to communicate with a host PC using Bluetooth

Note that the USB and BT interface blocks provide a means for transmitting and receiving data between the host PC and the NXT during runtime. This is a very useful feature for communicating specific data values or vectors between host and target, although it is not as flexible as the external mode feature described above.

Robot Genetic Algorithm Project

Robotic locomotion, particularly when it relies on jointed actuators resembling legs, can be a challenging project if one tries to develop a program "from scratch" to control and coordinate multiple legs. In this laboratory project students work in teams to explore how a genetic algorithm can be used to evolve a program to make a 6-legged Lego MindStorms™ robot walk. Figure 3 shows the robot chassis used in the project. Each team was supplied a pre-built chassis so that students would not have to divert their attention from the programming aspects of the project. Besides reinforcing the course lectures on the basic organization of genetic algorithms, the project required students to hypothesize and evaluate how different parameters of the standard genetic algorithm model (mutation rate, population size, crossover strategy, convergence criterion) will affect how quickly the robot learns motor control settings that enable the robot to walk in a coordinated forward-direction manner.

In the rest of this section's discussion, in Figure 3, the motors driving each of the front pairs of legs will be referred to as *motor A* and *motor C*. The motor driving the rear leg will be referred to as *motor B*.

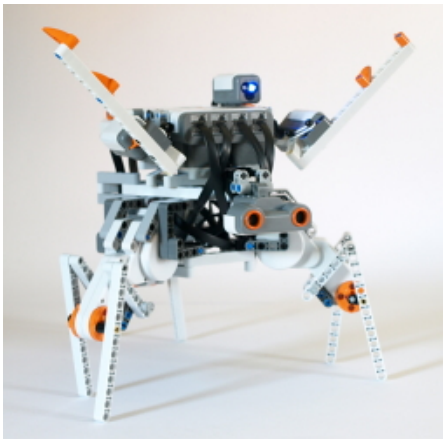

Figure 3. Spider Robot

Figure 4 shows the VU-LRT Simulink model used by the project to implement the genetic algorithm as a control loop, a common idiom for Matlab programming.

Each block represents a high-level view of the stages of the algorithm. They encapsulate low-level Matlab code and/or functions from the blockset. When students click once on a block, a "mask" is revealed. Masks show settings of control parameters for the block as editable numeric or text fields, which the student can edit to modify the block's output behavior. Students can also select a block and then use the Edit menu in the window to look under the mask to see and edit the code that uses the control parameters.

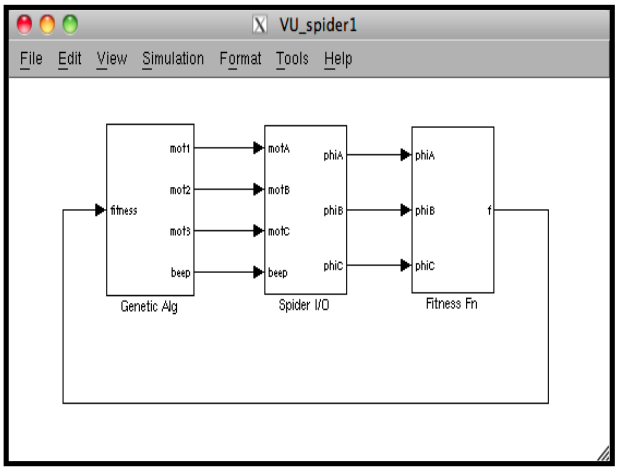

Figure 4. Simulink model for Genetic Algorithm

In Figure 4, the three blocks are labeled Genetic Alg, Spider I/O, and Fitness Fn. The code and control parameters within the Genetic Alg block perform the work of crossover and of mutating members of the chromosome pool in the project, then decoding a chromosome's

contents into motor speed settings to be passed to the next block. The Spider I/O block applies the motor settings to the robot's motors and provides as output the observed rotation (alternatively, observed forward motion) generated by the motor settings. This block is built from subblocks defined in Figure 3. In this project the students were given this code, but a variation on the project for students with significant Matlab background could include formulation of this code by the students. The Fitness Fn block uses the leg motion observations to calculate a fitness value for the chromosome that drove the motors, and returns that value to the Genetic Alg block.

The Genetic Alg block maintains an internal state variable to record the best fitness value observed so far, and then selects another chromosome from the pool to start the cycle again. The cycle is iterated for each gene in the pool. Once all chromosomes have had a chance to control the robot's motors, a short beep is emitted, crossover and mutating are applied to the pool again, and the process continues as described before. When convergence on a reasonable set of motor parameters is achieved, the Genetic Alg block causes the robot to emit a long beep, then sets its beep output high to prevent further iteration.

Chromosomes are 16-bit integers. Each motor is controlled by a 4-bit value that describes direction and speed. Thus, only 12 bits of the 16 bits are used. The code given to students does not optimize for 12-bit vectors because the project asks the students to identify possible optimizations themselves. Each motor is activated for the same duration. A variation of this project could include encoding the duration for each motor activation and sequence of motor activations, but in this study this was not built into the project so as to cut down on the size of the chromosome space.

The Genetic Alg block had a mask field that allowed students to select the type of crossover strategy used, and to change it as they needed. Strategy 0 has no crossover and only applies the mutation process to each chromosome. Strategy 1 creates offspring whose high order bits always come from the fittest chromosome and whose low order bits always come from the given member of the population. Strategy 2 creates offspring whose low order bits always come from the fittest chromosome and whose high order bits always come from the given member of the population. Strategy 3 creates offspring by randomly selecting for each population member whether to follow Strategy 1 or Strategy 2. The Genetic Alg mask also contained a field for the mutation rate and the number of bits that are changed when a mutation occurred.

Each team was asked to establish a baseline performance for the average number of generations it took in 7 runs for their robot to converge to forward walking behavior under the initial conditions of 20% mutation rate, at most 2 bits will be flipped in a mutation, and crossover strategy 3 in effect. The chromosome population was fixed at 10, with offspring replacing parents in each iteration. Each run took approximately 3 minutes for a robot to complete.

After the baseline was established, student teams chose one of three parameters (mutation rate, number of bits flipped, crossover strategy) to modify, based on their understanding of the algorithm from readings and lectures, in order to attain quicker convergence rates. Teams were also asked to identify and make deeper changes to the source code of the system in order to optimize the system for faster convergence.

Student Evaluation

The project as described was conducted in a Machine Learning course in the Spring 2011 semester at a liberal arts college. Eleven students were enrolled, with a variety of majors and programming backgrounds represented. Four teams of 2 or 3 students were organized. Each team had at least one computer science or computer engineering major and at least one non-computing major. All but one of the students had taken a prior non-robotics programming course. All students in the course reported having programming experience with Java. Only two of the students in the course reported prior experience with Matlab. The laboratory was conducted over a one-week period as an open lab, after the genetic algorithms lecture was completed. The instructor provided a 50-minute overview of Matlab. Students had handouts detailing the menus, buttons, dialog boxes, etc., that they would need to interact with in order to modify and cross-compile their genetic algorithms to the NXT Spider robots.

After completing the projects, but before they had received a grade, students were asked to take a survey of their experience. They estimated the amount of time they spent working personally on the module and the amount of time they spent working personally on other projects in the course. Using a 5-point Likert-like scale, they indicated the extent to which they agreed or disagreed with each of eleven statements related to the robotics project, which appears in Table 1, ranked by level of agreement. Note that for nine of the eleven items, the preferred response is *Agree*. For item no. 6 (ranked 10^{th} in the chart) the preferred response is *Disagree*. For item no. 11 (ranked 7^{th}) there is no preferred response.

They also responded to four open-ended questions related to the robotics module: (1) What was the most interesting aspect of the module? (2) What aspect of the module presents you with the greatest difficulty? (3) What changes would you recommend for this module's write-up? (4) Based on your experience with this module, what comparisons do you draw between block-diagram programming and text-based programming?

Conclusions

On average, students reported spending 2.6 hours per week working on the project related to the Robotics module, with the reported time invested across students ranging from a low of one hour to a high of four hours per week. Students also reported spending an average of 4.9 hours per week on other programming projects in the course, with the reported time invested across students ranging from a low of two hours to a high of twelve hours. It appears that the project is within current course norms in this matter, although the students did observe that the baseline work felt somewhat tedious.

In a review of the survey's Agree-Disagree statements, four thematic areas related to the project were identified, which we will discuss in turn, where appropriate drawing on responses in the survey and the open-ended questions.

The Robotics Platform: The project's use of robots as real-world objects which the students programmed to perform as specified. Participant responses to the statement "The module provided adequate background on the robot platform for my team to program it" generated more agreement across the group than any other statement in the survey: 10 participants were in agreement, one was neutral, and none were in disagreement. In responding to the question "What was the most interesting aspect of the module", every response focused on some aspect of getting the robots to perform and observing them as they did so. For example:

> "To see genetic algorithms control something in physical reality rather than just simulation."
> "Watching the robot improve through generations"
> "The robot itself moving based on criteria the lab team set"

It should be noted that in the open-ended request for suggestions for the write-up about the module, only one student indicated preference for software-based simulation.

Matlab: The use of Matlab as a tool for programming the robots. In responding to the agree/disagree statements in the survey, participants acknowledged that Matlab enabled them to change the robots' embedded code quickly and to focus on high-level design instead of low level programming. The agreement on Matlab's capacity for making changes in embedded code quickly was second, with 10 respondents registering agreement (Strongly Agree or Agree), 1 "in the middle" (Neutral), and zero registering disagreement (Disagree, Strongly Disagree).

Participants also tended to disagree with the statement that the module made them spend more time using Matlab than on achieving learning objectives. Nonetheless, a sizeable proportion of participants requested more background on using Matlab as a tool for the project. In most cases, the requests came from non-computing majors who indicated that although they had no difficulty working on the project via the parameter fields in the control block masks, they wanted to be able to learn more about Matlab programming than the handouts provided. So much for the paper's original assumption that non-computing majors will shy away from text-based programming!

Learning objectives: The survey addressed learning objectives in two questions: "The learning objectives for this module were clear to me," and "The module's project helped me achieve the learning objectives." For each of these statements, a little more than half of the respondents (6 of 11) indicated agreement. For the statement about clarity, the 5 of the 11 indicated neutrality. No one disagreed with the statement. For the statement about the project being a help in achieving learning objectives, six people were again in agreement, one person was in disagreement and the remaining four indicated neutrality.

Looking to the Future: Considerations about the module's future use and desirability. The two questions on the survey which addressed future issues (recommending the use of the module in future offerings of the course, and having increased interest in taking a course that used a similar module) were met with more neutrality than agreement. More people agreed (4) than disagreed (1) that they would recommend using the module in the course in the future. Most (6 of 11) were neutral about the idea.

Ranked last among the statements, with only one person in agreement it, two in disagreement, and eight neutral about it, was the statement about having an increased interest in taking a future course if the student knew the course used a module like the robot module. While the authors derive solace in the neutrality of the students, more work needs to be done in later course offerings to ascertain whether there is a gender effect (there were 4 females and 7 males in the class) occasionally seen in the literature in which women's enthusiasm for robotics is more dependent on the type and style of team work associated with the project, or an effect from perceived tedium in the baseline data gathering stage of the project.

The overall conclusions we derive from this case study are that the active-learning nature of the robotics project did lead to expected indications of student engagement, and that Matlab/Simulink has potential for bridging the gap in diverse-major AI courses, but that more investigation will be necessary to understand how much more Matlab background needs to be provided to students in order to increase their comfort with Matlab without overshadowing the active-learning focus on direct adjustment of a genetic algorithm.

References

[1] Greenwald, L., and Artz, D., "Teaching artificial intelligence with low cost robots," In Accessible hands-on artificial intelligence and robotics education, ed. L. Greenwald, Z. Dodds,

A. Howard, S. Tejada, and J. Weinberg, pp. 35-41. Technical Report SS-04-01. Menlo Park, CA: AAAI Press, (2004).

[2] Coradeschi, S., and Malec, J., "How to make a challenging AI course enjoyable using the RoboCup soccer simulation system, in RoboCup-98: Robot soccer world cup II: Lecture notes in artificial intelligence, vol. 1604, pp.120-124, ed. M. Asada and H. Kitano. Berlin: Springer, (1999).

[3] Goldweber, M., et al. "The use of robots in the undergraduate curriculum: Experience reports," Panel at 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, North Carolina..

[4] Klassner, F., and Anderson, S., "Lego MindStorms: Not Just for K-12 Anymore" IEEE Robotics and Automation Magazine, vol. 10, no. 2, June 2003. pp. 12-18.

[5] Summet, J., et al, "Personalizing CS1 with robots" Proceedings of the 2009 SIGCSE Symposium, Chattanooga, TN, March 4-7, 2009. pp. 433-437.

[6] N. McNulty, "Understanding Technology through Robots and Multimedia", National Science Foundation DUE #0088370 (2001).

[7] Behrens, A., et al. "MATLAB meets LEGO Mindstorms: A freshman introduction course into practical engineering", IEEE Trans. on Education, Vol.53, No.2, (2010), pp. 306-317.

[8] Peyton-Jones, J., McArthur, C. W., Young, T. A., The VU-LEGO Real Time Target: Taking Student Designs to Implementation, Proceedings of the 2011 American Society for Engineering Education Conference and Exposition (ASEE-2011).

[9] Parallax Inc, website: http://www.parallax.com/

[10] LEGO MindStorms NXT, website: http://www.lego.com/

[11] Korebot II, website, http://www.k-team.com/mobile-robotics-products/korebot-ii

[12] T. Chikamasa, "Embedded coder robot NXT instruction manual", www.mathworks.com/ matlabcentral/fileexchange/13399/, 2009.

[13] McNinch, L. C., Soltan, R. A., Muske, K. R., Ashrafiuon, H., Peyton-Jones, J. C. "An Experimental Mobile Robot Platform for Autonomous Systems Research and Education", Proceedings of the 17th IASTED International Conference on Robotics and Applications, (2009): 412-418

[14] McNinch, L. C., Soltan, R. A., Muske, K. R., Ashrafiuon, H., Peyton-Jones, J. C. "Application of a Coordinated Trajectory Planning and Real-time Obstacle Avoidance Algorithm". Proceedings of the 2010 American Control Conference, June 30-Jul 2, Baltimore MD, (2010).

[15] NXT OSEK/JSP, website, http://lejos-osek.sourceforge.net/

[16] VU-LEGO Real Time Target, website, http://www.mathworks.com/matlabcentral/fileexchange/29857-vu-lego-real-time-target

Acknowledgements

The authors gratefully acknowledge support for this project from the National Science Foundation (DUE No. 0837637), and the MathWorks Inc. This work is neither endorsed nor maintained by the LEGO group. MindStorms, NXT, and LEGO are trademarks of the LEGO Group.

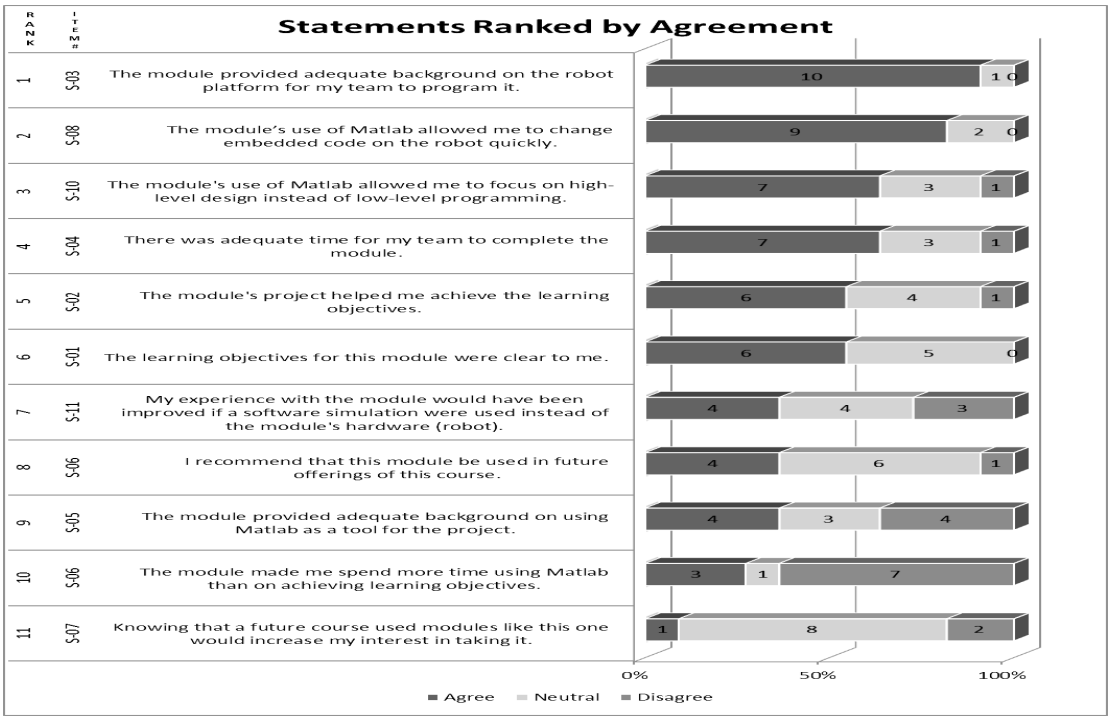

Table 1. Student Survey Summary