Development of the Open-Source “RobotRun” Robotic Simulation Software

Aleksandr Sergeyev¹, Nasser Alaraje¹, Scott Kuhl², Joshua Hooker², Vincent Druschke², Siddharth Parmar³

¹Electrical and Computer Engineering Program, School of Technology,  
²Computer Science Department  
³Mechanical Engineering Department  
Michigan Technological University, Houghton, MI 49931

Abstract

The rapid growth of robotics and automation, especially during the last few years, its current positive impact and future projections for impact on the United States economy are very promising. This rapid growth of robotic automation in all sectors of industry will require an enormous number of technically sound specialists with the skills in industrial robotics and automation to maintain and monitor existing robots, enhance development of future technologies, and educate users on implementation and applications. It is critical, therefore, that educational institutions adequately respond to this high demand for robotics specialists by developing and offering appropriate courses geared towards professional certification in robotics and automation. In order to effectively teach concepts of industrial robotics, the curriculum needs to be supported by the hands on activities utilizing industrial robots or providing training on robotic simulation software. Nowadays, there is no robotic simulation software available to the academic institution at no cost which limits educational opportunities. As part of the NSF sponsored project, team of faculty members and students from Michigan Tech are developing new, open source “RobotRun” robotic simulation software which will be available at no cost for adaptation by the other institutions. This will allow current concepts related to industrial robotics to be taught even in locations without access to current robotics hardware. The software acts as a simulator where a user can write a program and then view how that program performs when run on a virtual 3D robotic arm displayed on the screen. The RobotRun software shows a 3D, animated rendering of a robotic arm that can be controlled via an intuitive programming language that is similar to the programs used to program real robotic arms. In this paper, authors provide details of the software development and how it can be incorporated in academic curriculum for the institutions that may or may not have an access to the industrial robots.

Introduction

Many existing jobs will be automated in the next 20 years, and robotics will be a major driver for global job creation over the next five years. These trends are made clear in a study conducted by the market research firm, Metra Martech, “Positive Impact of Industrial Robots on Employment”¹. Many repetitive, low-skilled jobs are already being supplanted by technology.
However, a number of studies have found that in the aggregate, the robotics industry is creating more jobs than the number of jobs lost to robots. For example, the International Federation of Robotics (IFR) estimates that robotics directly created 4 to 6 million jobs through 2011 worldwide, with the total rising to eight to 10 million if indirect jobs are counted. The IFR projects that 1.9 to 3.5 million jobs related to robotics will be created in the next eight years².

The rapid growth of robotics and automation, especially during the last few years, its current positive impact and future projections for impact on the United States economy are very promising. Even by conservative estimates¹, the number of robots used in industry in the United States has almost doubled in recent years. In the manufacturing sector, the recent growth was 41% in just three years - the number of robots per 10,000 workers employed in 2008 was 96 and reached 135 in 2011. The automotive sector in the United States relies heavily on robotics as well - China produces more cars than the US, but the number of robots used in vehicle manufacture in China is estimated at 40,000 compared to 65,000 in US. From 2014 to 2016, robot installations are estimated to increase about 6% a year, resulting in an overall 3-year increase of 18% ¹. Likewise, industrial robot manufacturers are reporting 18-25% growth in orders and revenue year on year. While some jobs will be displaced due to the increased rollout of robots in the manufacturing sector, many will also be created as robot manufactures recruit to meet growing demand. Furthermore, jobs that were previously sent offshore are now being brought back to developed countries due to advances in robotics. For example, Apple now manufactures the Mac Pro in America and has spent approximately $10.5 billion in assembly robotics and machinery³. In March 2012, Amazon has acquired Kiva Systems, a warehouse automation robot, and in 2013 deployed 1,382 Kiva robots in three Fulfillment Centers. This initiative has not reduced the number of employees at Amazon; in fact, it added 20,000 full-time employees to its US fulfillment centers alone.

Such rapid growth of robotic automation in all sectors of industry will require an enormous number of technically sound specialists with the skills in industrial robotics and automation to maintain and monitor existing robots, enhance development of future technologies, and educate users on implementation and applications. It is critical, therefore, that educational institutions adequately respond to this high demand for robotics specialists by developing and offering appropriate courses geared towards professional certification in robotics and automation. To provide an effective robotic training to the university and community college students, industry representatives and displaced workers laboratory exercises must be integrated into the classroom environment. However, industrial robots are expensive and not every institution can afford to obtain them. Even though the hands-on training conducted on industrial equipment is preferred, the computer robotic simulation software can be a valuable substitute. Existing industrial robot training software is often too expensive and in many cases way too cumbersome for schools to provide for students or for students to acquire on their own. For example, high schools and community colleges may want to provide students with a basic level of experience with programming industrial robots. If the software is accessible and free, such training software could provide a platform for anyone to learn more about industrial robotics. In this paper, we describe the development of “RobotRun”, a software package that simulates an industrial robot and teach pendant controller. The software allows students to practice basic programming tasks which control the movement and function of the robot. When completed, this open-source program will be suitable for use in high-school outreach activities and in any degree program.

Proceedings of the 2017 Conference for Industry and Education Collaboration
Copyright ©2017, American Society for Engineering Education
which focuses on industrial robotics such as two- or four-year Electrical Engineering Technology programs. “RobotRun” is written in the Java programming language by two students over the course of a summer. It provides a 3D view of a robotic arm, allows the use of different end-effectors, and allows to simulate different factory environments and processes. In addition, the system allows students to learn about controlling the end-effector in different coordinate frames and programming paths that the robotic arm should follow. The teach pendant controller resembles real teach pendants and therefore provides students with a learning experience that can be transferred to real-world industrial robotics applications. This project is a part of a larger collaborative NSF sponsored project between Michigan Technological University and Bay de Noc Community College which aims to develop curricula and training materials to supplement the “RobotRun” software.

**RobotRun Software Core Features**

The RobotRun software is an industrial robotics simulator which simulates the core aspects of using a real robot. The software is free and open source and is aimed at individuals and students who are interested in learning about robotics, but lack access to an expensive industrial robot or access to costly commercial robot simulator packages. The software was developed for usage by the high school, community college, and university classrooms to introduce students to robotics in an accessible way. The software includes a realistic teach-pendant that controls the robot in a way that is similar to how real robots are operated. The core features of the software are already complete. It supports different coordinate frames, collision detection, programming features, end-effectors, and the interactive creation of objects that can be added into the environment for the robot to interact with. We are currently working on providing predefined scenarios which allows the user to create a specific environment aimed at teaching a particular skill with a click of a button. The simulator is written in Processing, a Java-like language that supports multiple platforms.

Current beta version of the software is available online at http://www.cs.mtu.edu/~kuhl/robotics/. The key features currently implemented in the beta version of the software are discussed next.

**End Effectors:**
The robot has a set of attachments, which can be fastened onto the robot’s faceplate: the suction, claw gripper, pointer, glue gun, and wielder end effectors. Only two of the end effectors actually have a function (explain in the robot-part interaction section), while the rest are purely aesthetic.

**Frames:**
The industrial robots operate in different coordinate systems - frames such as world, user, tool, and jog. Frames are used to configure special types of motion commonly used in industrial settings. Some of the frames are predefined and some can be user configures. User frames are comprised of an origin point and a set of three orthogonal unit vectors, which represent the X, Y, and Z axes. A tool frame consists of an offset vector, which defines the frame’s tool tip position with respect to the robot’s faceplate position, and the three orthogonal unit vectors that define the axes of the frame. The user can define ten user frames and ten tool frames. The tool frames axes function strictly as alternative coordinate systems, in which the robot can jog, to the world frame. Though, positions saved in a program are saved with reference to the tool tip of the active tool.
frame, they are never saved with reference to the active tool frame’s axes. Yet, points are saved with reference to the active User frame’s origin and axes, or the world frame in the case that no user frame is active.

A tool frames can be taught with the three-point, six-point, and direct entry method. A user frames can be taught with the three-point, four-point, and direct entry methods. Points are taught in the normal fashion: jog the robot to a position and save the values of the robot’s position with the teach pendant. Any point taught for a frame teaching method that is actively being taught will be displayed in the world environment. The points are color-coded based on the point’s relation to the teaching method. In the example above, the three tool tip points are shown as the gray points (1, 2, and 3), the orient origin point is orange (4), the x-direction point is red (5), and the y-direction point is light green (6). This example only lacks one type of teach point: the origin point, which is only taught in the user frame four-point method and appears blue in the world environment. Additionally, the user can move to a taught point using the teach pendant. Since every frame stores the last value of each teach point associated with the frame, a taught point can be referenced at any time by the user until it is overridden by another value. Alternatively, the direct entry method can explicitly specify a frame. The user can navigate between the different values of the frame entry with the arrow buttons and use the number pad on the pendant to input each value before confirming the entry. Similar to the taught points of a frame, the last direct entry specified for a frame is saved independent of the current value of the frame, and will appear, when the user returns to the direct entry method of that frame again.

A screenshot of the points associated with a tool frame for the six-point method as well as a direct entry approach are shown in Figure 1.

![Figure 1: Screenshot of six-point and direct entry methods display](image)

**World Objects:**
The user can create two classes of world objects: parts and fixtures. The main difference between parts and fixtures is that the robot can only interact with parts; fixtures are coordinate frame references for parts. In any case, world objects have two basic forms: boxes and cylinders. Complex objects can be imported into the program as well. Cylindrical and box-like objects require dimension specifications (radius and height for cylinders and length, height, and width
for boxes), while complex objects require a .stl source file and scale value. As their name suggests, complex objects offer the user the ability to define more complex shapes, which allow the robot to perform more practical tasks. Furthermore, each object has a unique name and a color scheme consisting of an outline color and fill color (except for complex shapes, which only have fills colors), so that they can be easily distinguished from one another. Additionally, the user may also manually specify the position (x, y, and z) and orientation (w, p, and r) of a world object after the object is created and revise its orientation and position whenever necessary. As mentioned before, fixtures can replace the world frame as a part's coordinate frame reference. So, the position and orientation of the part would be with reference to the fixture’s local coordinate system (i.e. its position and orientation with respect to the world frame). The examples of some objects are shown in Figure 2.

![Figure 2: Examples of simple and complex software objects](image)

**Robot and World Object Interaction**

In prior screenshots the parts have a green wire-frame box surrounding them, whereas the fixture has no wire-frame box surrounding it. In addition, multiple wire-frame boxes surrounding the various segments of the robot as it is indicated in Figure 3 (a). These wire-frame boxes visually represent the bounding boxes associated with parts (and the robot). All interactions between the robot and parts involve bounding boxes; fixtures require no bounding box, since they do not interact with the robot. The color of a bounding box acts as a flag for the state of the object associated with itself (part or robot segment). The two of the possible colors of bounding boxes are red and blue. When a bounding box is red, then that box is colliding with another bounding box. Figure 3(b) depicts the robot colliding with a white block.
Figure 3: (a) Robot bounding boxes, (b) Robot colliding with the object

The bounding box color blue is a special type of collision between a part and certain robot end effectors bounding boxes, which are involving in the picking and placing of parts by the robot. Only two of the robot’s end effectors are able to pick up objects: claw gripper and suction. Each has a set of blue bounding boxes shown below, as shown in Figure 4.

Figure 4: Bounding Boxes

In order to pick an object up with the robot, one of blue bounding boxes must be colliding with a part’s bounding box, while no other bounding box of the end effector is colliding with a part. Therefore, the suction end effector must be almost touching the bounding box of a part in order to pick it up. While a part must fit in between the grippers of the gripper claw in order to be picked up by the claw. When a part can be picked up by the robot, its bounding box will appear blue. Although, the collision of a part with the end effector’s normal bounding boxes take precedence over the collision of a part with an end effector’s special bounding boxes. So the part’s bounding box will appear red, if the end effector is colliding with the part even if the end
effector’s special bounding part is colliding with the part’s bounding box. Furthermore, if the robot is currently holding a part, then the part’s bounding box will no longer appear blue, unless it is released and is not colliding with the end effector’s normal bounding boxes. Although the software detects collisions between bounding boxes, no collision handling exists; the user is only notified of a collision when it occurs. Bounding box display is toggleable: the user may hide or show them at will.

**Programming Features**

The user utilizes the teach pendant interface to create, view, edit, and delete programs to manipulate robot state information, including robot position, end effector state, internal register values, and coordinate frames. Programs are composed of a sequential list of instructions that are executed in-order, beginning either at the instruction currently selected by the user, and ending at the last instruction. The program instruction set includes a number of different instruction types, including movement instructions to modify the position and orientation of the robot end effector; register modification instructions for I/O, floating point, and position registers; and control flow instructions in the form of conditional constructs (‘if’ statements), switch statements, labels and jump label instructions, and function calls. Each instruction has a number of individual fields that the user can modify as well. Motion instructions include fields for manipulating motion type, specifying a locally or globally scoped position to move to, specifying which position index to move to, manipulating the robot’s movement speed during the motion, and manipulating smooth interpolation between the current and next point. Supported motion types include joint motion, where movement is performed by simultaneously rotating each joint to a specified angle; linear motion, where inverse kinematics is used to trace a straight line path from the end effector starting point to its destination; and circular motion, which adds an additional point to the movement instruction and traces the robot’s end effector in a circular arc that passes through its starting, intermediate, and end points.

Register instructions allow the user to provide a register of any type and an expression to be evaluated at runtime, the value of which will be stored in the given register if the expression returns a compatible, non-null data type. Register statements can perform operations on boolean, floating point, and positional1 data, and each of these data types correspond to I/O, data, and position registers, respectively; non-matching data types cannot be stored in a register of that type. If an operation is performed on incompatible data types, or if one of the expression elements is uninitialized, the expression will return null, execution will stop, and an error will be generated to notify the user.

Conditional statements also allow the user to define an expression or to compare two expressions’ values; the statement will then execute a jump or call instruction only if the value of the expression or comparison evaluates to true. Case statements can also be used to specify conditionally executed instructions, and accept a data register as an argument; if the data register matches any of the case values specified, the instruction associated with that case is executed. Labels and jumps are used to arbitrarily manipulate program control flow. Labels are placed into programs as a normal instruction would; a jump instruction to that label then moves execution to wherever the specified label is in the program, and the next instruction is considered to be the instruction immediately following that label. Different labels are identified by numerical IDs,
and each label must have a unique ID to ensure that any jump to that ID is not ambiguous.

Finally, call instructions can be used to move execution to another program entirely. A call to a given program will begin executing that program, in its entirety, from its first instruction, and will return execution to the instruction immediately after the call instruction in the calling program once the call is finished executing. By using a combination of movement, register, and control flow operations, the user can specify arbitrarily complex behavior in the robot to complete a variety of tasks.

Implementation Detail: Inverse Kinematics

One of the most important and complex aspects of the “RobotRun” software is the inverse kinematics system. This system calculates how to orient each segment of the arm when given a user-specified position and orientation for the end effector in the world frame. When the end effector is moving along a path, the inverse kinematics problem must be solved for each frame and the user expects the movement to be smooth and natural looking. The problem is made more complex by the fact that there are some end effector positions that are invalid because they are outside of the robot’s range or would cause the robot to self-collide. Our solution, which resulted from making multiple attempts at solving the problem, relies on a well-known approach called the Jacobian Pseudoinverse. The general strategy is to calculate a matrix which represents how small changes to each joint influences the end effector position and orientation, called a Jacobian matrix. Given this matrix and a target point for the end effector, we can calculate how each of the joints need to be changed to get the end effector to its intended position by inverting the Jacobian and multiplying this matrix by the vector pointing from the robot’s current point to its target point. While this calculation should, in theory, result in the exact joint angles needed to move the end effector to the target position, it is not always feasible to obtain the inverse of the Jacobian, either due to time constraints or because the matrix is simply not invertible. To avoid these problems, we instead calculate the pseudoinverse of the Jacobian using a function provided by the Apache Commons Math Library to obtain a reasonable approximation of the matrix inverse. If the joint angles that result from this operation are within a certain margin of error from the target point, the inverse kinematics calculation is considered successful and no further operations need be done; if the resulting position or orientation of the robot is too distant from the target, we begin the inverse kinematics calculation again using the result obtained from the previous calculation as the robot’s new starting position, and continue in this fashion until we either obtain a tolerable result or the calculation limit is reached. By repeatedly performing this series of calculations, the algorithm should converge on a solution that is within our levels of tolerance. Note that, for the purposes of our software, a position consists of 12 floating point values which describe both the Cartesian position and orientation of the robot in space, and the joint rotational values that the robot has when occupying that position in that orientation. With extensive testing and troubleshooting, we have arrived at a solution with runs quickly on commodity hardware and provides a sufficiently accurate solution.

Application Scenarios
The software features discussed above are powerful enough to simulate basic robot functioning required to create applications similar to the industry. Robots are being used drastically across the automation industry for material handling, manufacturing and assembly operations. Efforts have been made to create scenarios that replicate these operations and provide the user a strong foundation of using the different features of the software. Following are the scenarios created using the Robotrun software.

**Pick and place objects from multiple stations:**

The objective of this scenario is to teach the user to create a robotic workcell, as shown in Figure 5, using different fixtures and parts, learn to operate the robot to pick and place these parts on fixtures and create a simple program to record different positions to run the simulation process. Two parts are moved around three fixtures in a cyclic manner picking one part at a time using a vacuum cup selected from the set of tools. The programming involves recording pick and place positions and using I/O instruction to turn the vacuum on and off.

**Grinding a given part using tool frame and creating a user frame:**

A crooked shaped part is attached to the robot face plate as shown in Figure 6 and the conical surface of this part is required to be grinded. While creating the workcell the user creates a cylindrical object representing the grinding wheel. The application demands the user to create a tool frame with the axis of rotation along the pointed tip and use the six-point method. Creating this tool frame helps the user understand the simplicity and comfort of performing this operation. The user frame creates a separate frame of reference for the robot motion. The user inserts a rectangular surface in the robot’s environment and provides it a random orientation. The task is to create a frame of reference using the edges of this surface. When the user has successfully created the user frame the user can jog the robot along the edges of the rectangular surface.
**Welding application for sheet metal using circular instruction:**

There is a sheet metal part available in the software library that is imported twice in the workcell and oriented as shown in Figure 7. The tool used for this operation is a welding tool and user programs the robot to move along the line joining the parts. To accomplish this task, the user first creates the tool frame using three-point method and then uses the circular instruction to program the robot to move in the circular paths. The scenario provides the user another important application of tool frame while performing this task because without it there are high chances of collision of the robot with the parts.

**Gluing application using position registers and Offset instruction:**

Gluing is generally performed by the robot by moving in a zigzag motion along the length of a part. The user inserts a rectangular sheet in the workcell as shown in Figure 8 and uses the glue tool to perform this task. The robot has to perform this motion along the length of the sheet, offset by a certain value along the breadth and repeat the zigzag motion along the length. Firstly, as the robot moves along the rectangular sheet, the user creates a user frame using four-point method. To smartly program this motion the user implements position registers and records the start position in the program. The values of this position register are used to create equations and move the robot to new positions. The offset instruction offsets the value of the position by a certain value and highly simplifies the efforts of programming. There are few other interesting scenarios that include the usage of copying and pasting feature, macro and register equations. All scenarios have been developed with the purpose of highlighting the features by relating them to real time applications. After the completion of these scenarios, the user would have excelled in implementing basic programming of robots with good understanding of using different features for different applications.
Conclusion

In this paper, we described the development of RobotRun. The software seeks to provide industrial robotics education opportunities in a way that is useful and accessible to students and teachers alike. Although work on the software is ongoing, the current system allows a user to control and program the robotic arm and end-effector. The software features a teach pendant system that resembles real-world robots. By creating a realistic learning environment and providing features that will help educators, such as built-in screen-recording and audio recording features, we hope that this system will help increase the amount of robotics education opportunities in K-12 and higher education settings.

Acknowledgements

This project is supported by National Science Foundation grant DUE-1501335.

Bibliography:

1. International Federation of Robotics: Metra Martech Study on Robotics (http://www.ifr.org/uploads/media/Metra_Martech_Study_on_robots_02.pdf)
2. International Federation of Robotics (http://www.ifr.org)

Biographical Information:

ALEKSANDR SERGEYEYEV is Associate Professor in EET program at Michigan Tech. He has a strong record publishing in prestigious journals and conference proceedings such as Measurements Science and Technology, Adaptive Optics, Sensors and Materials, The Technology Interface International Journal, ASEE, IEEE, and SPIE. Dr. Sergeyev is a Co-PI on several NSF and DOL awards, and a PI on multiple significant industry awards.

NASSER ALARAJE is currently EET program Chair and an Associate Professor at Michigan Tech. Alaraje’s research interests focuses on processor architecture, system-on-chip design methodology, field-programmable logic array (FPGA) architecture and design methodology, engineering technology education, and hardware description language modeling. Dr. Alaraje is US Fulbright scholar; he is a member of ASEE, IEEE, and ECETDHA.

SCOTT KUHL is Associate Professor of Computer Science, advisor to the Husky Game Development (HGD) Enterprise, and adjunct faculty in Applied Cognitive Sciences and Human Factors at Michigan Tech. His primary areas of research include immersive virtual reality systems, such as head-mounted displays, and human perception. Dr. Kuhl has worked on NSF and US DOL projects and has received corporate and industry sponsorships for HGD.

JOSHUA HOOKER is a software engineering undergraduate at Michigan Tech. Joshua Hooker is a member of the Husky Game Development Enterprise as well

VINCENT DRUSHKE is currently pursuing an undergraduate degree in computer science.

SIDDHARTH PARMAR is currently pursuing a graduate degree in Mechanical Engineering at Michigan Tech. His professional interests include mechanical design, robotics and automation.