ASEE'S VIRTUAL CONFERENCE
At Home with Engineering Education
JUNE 22 - 26, 2020 #ASEEVC
Paper ID #29357

Development of a MATLAB/ROS Interface to a Low-cost Robot Arm

Prof. Robert L. Avanzato, Pennsylvania State University, Abington

Robert Avanzato is an associate professor of engineering at the Penn State Abington campus where he teaches courses in electrical and computer engineering, computer science, and robotics. His research interests are mobile robotics, computer vision, intelligent systems, collaborative virtual environments and innovative education.

Development of a MATLAB/ROS Interface to a Low-cost Robot Arm

Abstract

Development of a MATLAB Robotics Toolbox ROS interface and educational resources for a low-cost robot arm (Dobot Magician) in a senior-level robotics design course are described. The objectives of the study were to evaluate the effectiveness of a MATLAB interface to ROS services to control an articulated robot manipulator and conveyor belt in a laboratory setting. Laboratory exercises are described that expose students to the MATLAB/ROS interface, the basics of robot manipulator programming and an introduction to computer vision. Student survey data shows a positive response to the MATLAB/ROS strategy with the robot. The significance of this study is that a low-cost robot arm with a professional-level ROS/MATLAB software interface can greatly improve student access to advanced, hands-on, project-based education in intelligent manufacturing and Industry 4.0.

1.0 Introduction and background

Robot Operating System (ROS) is an open source, Linux-based robotics development and deployment system which supports many commercial and research robots, including mobile robots, underwater robots, aerial robots and robot arms (manipulators) [1]. ROS provides a structured development and deployment software architecture, with a distributed model, across a variety of sensor and hardware platforms. Although software development in ROS is primarily implemented in C++ and Python languages (other languages are supported), the MATLAB Robotics System Toolbox also provides a ROS interface. This MATLAB interface enables engineering students to more easily communicate with ROS-enabled robots. The MATLAB/ROS solution provides students with a more intuitive and interactive programming environment, visualization tools, and integration of other MATLAB toolboxes such as computer vision. This approach ultimately enables students to perform advanced projects in robotics across a variety of robot platforms.

This paper describes the development of a MATLAB/ROS interface and educational resources for a low-cost robot arm (Dobot Magician) in a senior-level robotics design course. Laboratory exercises were developed and tested using MATLAB Robotics Systems Toolbox and a ROS-enabled Dobot Magician robot arm. Topics covered basic interfacing MATLAB and ROS services, pick and place operations, control of a conveyor belt and IR sensor, and computer vision for color sorting and shape detection of objects (rejection of defective parts on a conveyor belt). The Dobot is a commercially available, low-cost, 4 degree of freedom (DOF) robot arm with 0.2mm precision and is capable of pick and place as well as laser engraving and 3D printing. This robot has a small footprint and is an ideal device for a classroom or laboratory setting.

The robotics course emphasizes hands-on projects and design and includes modules on mobile robotics and manipulators as well as specialized robotics projects. The robotics course is designed with a systems design and manufacturing emphasis. ROS and MATLAB are used in the other modules in the course and the students have prior experience with using MATLAB and ROS to control a Turtlebot 2 mobile robot and a Husky mobile robot. An educational goal of the robot manipulator module is to incorporate robot hardware that support ROS and MATLAB and specifically introduce the students to ROS services (which is not previously covered in other course modules). The other educational goal of this module is that students be exposed to programming a robot manipulator solution that supports integration of a conveyor belt, camera and sensors, and which simulates a real-world environment. Finally, the robot manipulator and software tools should seamlessly support the introduction of basic computer vision techniques in the context of a manufacturing or production environment.

Due to the growing importance of ROS in research and commercial robotics, engineering educators are introducing ROS into the engineering curriculum [2], [3], [4], [5], [6], [7]. As can be noted from the literature, ROS is powerful technology but has a steep learning curve, which makes the interface with MATLAB an attractive alternative for educators. Several educators have reported results integrating MATLAB Robotics Toolkit with ROS middleware for mobile robotics [8], [9]. to improve student accessibility. One researcher has surveyed 75 robotics educational programs (undergraduate and graduate) and indicated MATLAB is the most commonly used software language in robotics programs [10]. It was also reported in the same study that there was a general need for a low-cost robot manipulator for educational use.

The methodology of this research project is the develop laboratories to introduce students to the MATLAB programming interface to the ROS control of the Dobot robot arm system and to progressively advance student programming skills leading to the introduction of basic concepts in computer vision. Finally, the students were surveyed to assess the effectiveness of the hardware and software tools introduced in the lab projects. The primary focus is to evaluate the effectiveness of MATLAB programming interface to the ROS control of a robot manipulator, conveyor belt and sensors in the context of hands-on laboratory projects.

This paper is organized as follows: section 2 describes key features of the Dobot robot manipulator, section 3 introduces ROS fundamentals, section 4 discusses the MATLAB/ROS interface, and section 5 describes the laboratory projects. Finally, section 6 provides a summary of student feedback, conclusions and future directions.

2.0 Dobot robot manipulator

As mentioned, the Dobot Magician (Shenzhen Yuejiang Technology Co., Ltd.) is a low-cost, 4 DOF, articulated robot manipulator with 0.2mm precision [11]. This robot is commercially available in two versions: a basic (standard) version (USD$1400) with suction cup, vacuum pump, gripper, pen module, 3D printing support; and an educational version (USD$1700) which adds capabilities for laser engraving, joystick control, Wi-Fi and Bluetooth connectivity. The project described in this paper is compatible with the basic version of the Dobot Magician. A conveyor belt system, compatible with the Dobot Magician and available from the same vendor, is also available for USD$560. The conveyor belt system includes a color sensor and an IR sensor for detecting the presence of objects on the conveyor belt. The web camera (Logitech

C920; USD$70) and camera stand (USD$30) used in our project were purchased separately. The size if the base (footprint) is 158mm (6.2 in.) by 158mm (6.2 in) making it ideal for a classroom or laboratory bench environment (see figure 1). There are 12 input/output ports including support for ADC and PWM. A serial communication port and control for two stepper motors is featured.

There is programming language support for Python, C++, C#, Java, Visual Basic, MATLAB (DLL), and others. Mobile iOS and Android support are included, as well as PLC and microcontroller support (Arduino). A ROS package with Dobot support is also provided and was utilized in this project. All API documentation and demonstration programs are provided on the product website. A client desktop software application (Magician Studio) is available to operate the robot and to provide an interface to robot arm accessories. Using the Studio software, it is possible to directly control any of the joint angles and set the XYZ Cartesian position of the end effector. There is also a teach and replay programming mode and a pen writing feature in the Studio client that can be used as a student orientation to the robot operation.

It should be noted that MATLAB can be interfaced to the Dobot robot through a DLL installation which would not require ROS. In our study, we evaluated the use of ROS and the MATLAB interface to ROS to program the robot. Our requirement of ROS is that ROS/MATLAB is used in other modules of the robotics course (mobile robotics) and the ROS ecosystem offers other tools and networking capabilities that may support advanced projects.

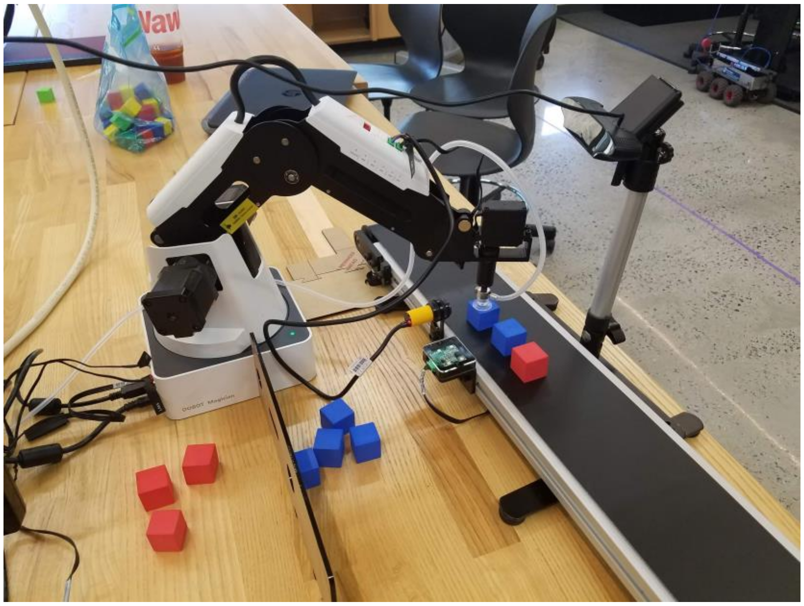


Figure 1: Dobot Robot Arm, Conveyor Belt, Sensors and Camera

3.0 Introduction to ROS

The Robot Operating System (ROS) is experiencing increased use in research robotics, commercial robotics and engineering education [12], [13], [14]. ROS has a rather steep learning curve and requires Linux OS. (It should be noted that ROS2 will have Windows OS support [15]). A complete description of ROS is beyond the scope of this paper. Several key concepts of ROS that relate to the robot arm project will be introduced here.

Firstly, a "node" in ROS is a software entity represented by executable code (for example, a sensor driver or an algorithm to analyze sensor data). ROS nodes are distributed across a network and registered by a master node that comprise a ROS system. ROS nodes can be distributed across hardware platforms but there is only one master node. Secondly, communication among nodes is accomplished with two major mechanisms: topics and services. Nodes communicating using topics are designated as either publishers or subscribers. Publisher nodes will publish or send data anonymously to a topic (which specifies the data message type and format) and subscribers retrieve data from the topic of interest. The communication is anonymous in that there is no handshaking between the subscriber nodes and the publisher nodes. Topics behave as a "channel" or "pipe" to facilitate communication among nodes. It is a many-to-many model but, in many cases, there is one publisher (for example, a node publishing images from a camera) and one or more subscribers (for example, nodes displaying and analyzing camera images). Topics are used extensively with mobile robotics projects. For example, velocity commands are sent to the mobile robot platform via topics and sensor data (such as camera data and odometry data) is exchanged and shared using topics. The diagram below (Figure 2) outlines the topic model for communication in ROS.

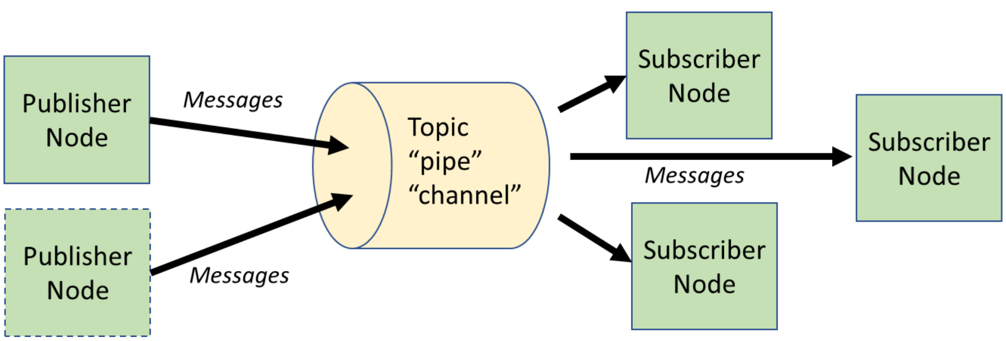


Figure 2: ROS Topics Model

The ROS interface for the Dobot Magician robot arm uses ROS "services" to control the robot and communicate with other nodes in the ROS system. The server node (which also communicates with the robot firmware) takes requests from client nodes which are created by the programmer to operate and control the robot. Notice that there is handshaking in the service model (see figure 3). After a request is sent from a client node to the service node, a response message is returned. A timeout error occurs if the response is not received after a request is invoked.

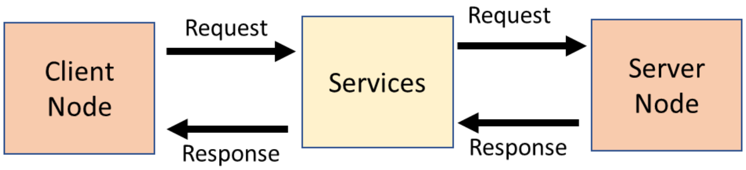


Figure 3: ROS Services Model

There are 61 services defined in the Dobot Robot ROS package. The service files and the server program (DobotServer) are provided by the Dobot vendor. In our ROS architecture there will be one Dobot Server node and multiple service clients which will send commands with data messages to the server. Below is an example of a service definition to move the robot end-effector to a specific XYZ coordinate (filename: SetPTPCmd.srv). The ptpMode sets the trajectory between points (linear or jump). The r coordinate is the rotation angle of the end-effector. The data sent in the request from the client is above the dashed line (---) and the data returned from the server back the client is below the dashed line.

uint8 ptpMode
float32 x
float32 y
float32 z
float32 r
---
int32 result
uint64 queuedCmdIndex

Associated with that service definition is a service request message type and service response message type. The service request message type (as obtained in MATLAB) is provided below using the MATLAB "rosmsg" command:

>> **rosmsg show dobot/SetPTPCmdRequest**

uint8 PtpMode
single X
single Y
single Z
single R

As will be discussed in the next section, the service message types for the Dobot services are required to be created as custom messages in MATLAB.

4.0 MATLAB/ROS interface

The overall system architecture for the ROS/MATLAB environment with robot system is shown in Figure 4. Each computer was configured with virtual machine software (VMware) and MATLAB (2019a). A virtual machine was configured with Linux Ubuntu 16.04 and the ROS packages were installed. The Dobot ROS packages were built and the DobotServer node and necessary files were created in the Linux environment. The robot is connected by a physical USB cable and is visible to the Linux virtual machine. The conveyor belt, color sensor and IR sensor are connected directly to the robot device. The USB web camera was accessed by the MATLAB environment in Windows. The configuration of the virtual machine with the configured ROS packages can be performed one time and then the virtual machine image can be saved for distribution to other lab computers.

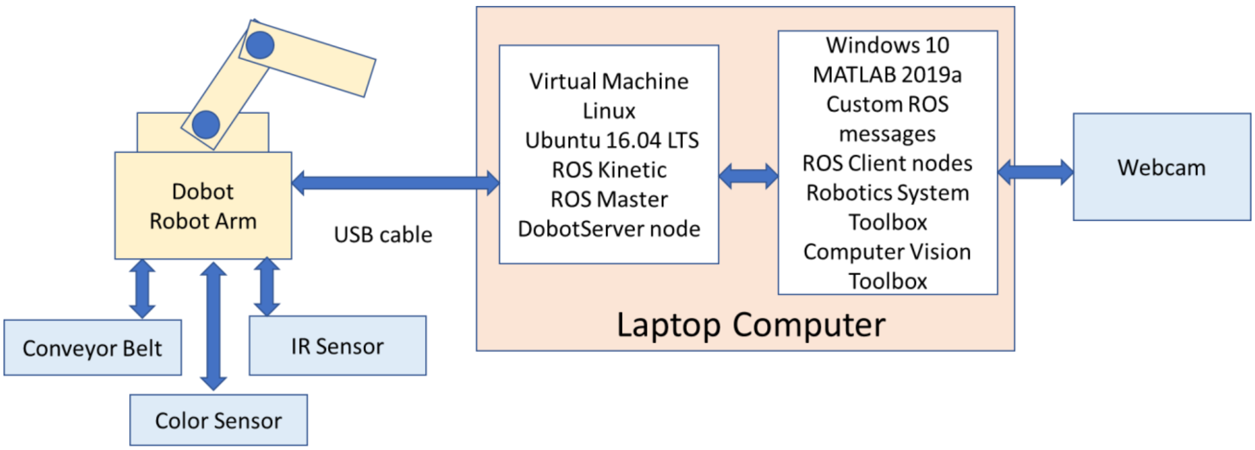


Figure 4: Robot System Architecture

In the MATLAB environment, it is necessary to first create custom messages for the Dobot services. The MATLAB utility for this operation is "rosgenmsg" (Robotics System Toolbox Add-on) [16]. This procedure requires a folder of all Dobot service definitions which are available from the Dobot ROS package. This operation is performed one time and then the generated custom message files can be distributed and copied to the lab computers. After the custom message folder is installed, it is necessary to perform two operations in MATLAB: 1) add a path name to a .jar file to the javaclasspath.txt file, and 2) use the "addpath" command to add a path to the installed custom message folder.

To initiate a MATLAB/ROS robot session, the virtual machine is started with the Linux/ROS distributions and a terminal window is opened and "roscore" is executed to run ROS and create the ROS master node. Then another terminal window is opened in Linux to run the Dobot server ("rosrun dobot DobotServer ttyUSB0"). This server node makes a direct connection to the robot arm using the USB connection. In the MATLAB environment, we start ROS and connect to the ROS master in the virtual machine by entering "rosinit(ip address of virtual machine)". Once the ROS network is connected, the students are not required to perform any operations in the Linux environment. This situation is desirable in our case since the majority of students had no prior experience with Linux OS commands. However, students with familiarity with ROS and Linux can access many advanced resources if needed for special projects.

Initiating a ROS service request from MATLAB generally consists of 4 steps: 1) creating a client, 2) creating a message, 3) filling message with data, 4) calling the service. Here is an example of a service request to move the end effector to a specified location in Cartesian space.

```
>>  client - rossvcclient('/DobotServer/SetPTPCmd')     % create client based on service
>> cmd = rosmessage(client)                             % create an empty message
>> cmd.PtpMode = 0;              % 0 → jump mode;  2 → linear mode
>>  cmd.X = 200;                 % fill message with XYZ data (in mm)
>> cmd.Y =  0;
>> cmd.Z = 50;
>> cmd.R = 10;                   % set rotation angle of end-effector = 10 degrees
>> call(client, cmd);            % call the service to move the robot
```

Below is an example of calling the GetPose service to display the current Cartesian coordinates and rotation of the end-effector. In this service, no message data values are sent to service.

```
Pose_client = rossvcclient('/DobotServer/GetPose');
Pose_cmd = rosmessage(Pose_client);
call (Pose_client, Pose_cmd)

MessageType: 'dobot/GetPoseResponse'
      Result: 0
         X: 272.1753
         Y: 7.1691
         Z: 16.5563
         R: 1.5088
      JointAngle: [4×1 single]
```

As can be observed, the MATLAB programming required to call the Dobot ROS services are very straightforward and intuitive. In the next section, the laboratory exercises will be discussed.

5.0 Robotics laboratory projects

This section will describe the introductory robot arm laboratories which were completed by student teams using the MATLAB/ROS programming environment and the robot arm setup. Key MATLAB code examples will be provided as well.

5.1 Laboratory #1: Stacking blocks

In this lab the students were required to program the robot arm to stack 3 blocks (cubes with length of 25mm (1 inch)). The blocks were originally positioned horizontally on the table and the goal is a stack of vertically arranged blocks in another specified location. This lab emphasized the programming technique of using coordinate offsets to calculate all other block target positions based on one known block position and all relevant distances between blocks. The suction cup end-effector was used in this lab. The "jump" mode was used to move the robot arm and the jump height was adjusted in the program to add clearance. The jump height was achieved by a ROS service call to the Dobot server.

```
SetJumpClient = rossvcclient('/DobotServer/SetPTPJumpParams');
SetJumpCmd = rosmessage(SetJumpClient);
SetJumpCmd.JumpHeight = 50;          % set jump height to 50mm to be safe
SetJumpCmd.ZLimit = 100;
call (SetJumpClient, SetJumpCmd);    % do this one time in your program
```

5.2 Laboratory #2: Removing blocks from conveyor belt

This lab required students operate the conveyor belt and IR sensor to detect and stop the conveyor belt when objects (25mm cubes/blocks) reached a target position on the belt. The MATLAB code was able to start and stop the conveyor belt based on the IR sensor output using appropriate ROS services. The blocks were then removed from the conveyor belt using the suction cup and moved to a target position to be deposited.

5.3 Laboratory #3: Sort colored blocks from conveyor belt using color sensor

In this lab, randomly placed blue and red blocks were removed from the conveyor belt using the suction cup and each block was placed 1cm above an electronic color sensor. Based on the readings of the color sensor, the blocks were sorted into a container of red blocks or a container of blue blocks. The color sensor was activated, and the sensor readings were accessed using ROS services through the MATLAB interface.

```
GetColorClient = rossvcclient('/DobotServer/GetColorSensor');
GetColorCmd = rosmessage(GetColorClient);
result = call (GetColorClient, GetColorCmd);
%  access result.R, result.B, result.G for sensor readings
```

The sensor, which was provided in the conveyor belt kit, did not perform as described and was only able to distinguish blue and non-blue blocks in our experiments. We have identified and tested a replacement sensor (Adafruit TCS34725) which can be used with an Arduino microcontroller and then interfaced to the robot arm I/O ports for future laboratory exercises.

5.4 Laboratory #4: Sort colored blocks from conveyor belt using overhead camera

This lab required use of the webcam and stand and required students to develop a simple MATLAB image processing algorithm to detect and differentiate between red and blue blocks. It also required the student to crop the image using MATLAB commands to yield a single object (block) in the image to be processed. Students were introduced to the RGB images, binary images and how to create a simple color detector. Basic image processing commands to filter the image, remove noise, and the regionprops function were also introduced. Using the computer vision algorithm in MATLAB, the randomly sequenced red and blue blocks (which were centered on the conveyor belt with variable spacing) could be color sorted by the robot arm into separate containers. The control of the robot, conveyor belt, sensors, as well as the image processing were all developed and tested in the same interactive MATLAB environment, which simplified the development and prototyping time.

5.5 Laboratory #5: Sort colored blocks from conveyor belt using overhead camera to determine position

This lab was an extension of laboratory #4 except the colored blocks were not centered on the conveyor belts (see figure 5 below). The students were required to develop, test and implement a calibration step and transform the centroid position of the block from image coordinates to robot coordinates. Using the coordinate system transformation, the colored blocks (randomly separated and randomly displaced from the center line of the conveyor belt) could be removed from the conveyor belt with the suction cup and sorted by color. The webcam and MATLAB color algorithm were used with the regionprops function to obtain the centroid (x, y position) of the blocks in the image. It should also be noted the IR sensor data was used to stop the conveyor belt when a block was detected, but the IR sensor does not return distance from sensor thereby motivating the computer vision localizing solution.

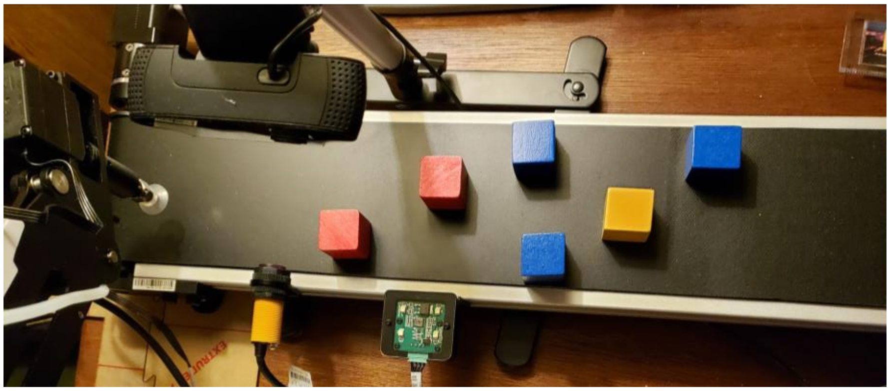


Figure 5: Sorting Blocks with Camera (identifying offset position)

5.6 Laboratory #6: Sort colored blocks from conveyor belt using overhead camera to determine shape

In this lab objects in the shape of square blocks and in the shape of cylinders were sorted by color and shape. The shape was analyzed in MATLAB using the circularity index which compares object perimeter to area using the 'circularity' property of the regionprops function. Circularity can also be computed by calculating the perimeter from object border data and the area data separately. The circularity index is 1 for a circle and approximately 0.7 for a square. The overhead camera was utilized, and the students were able to sort the blocks into 4 bins according to color and shape (see figure 6).

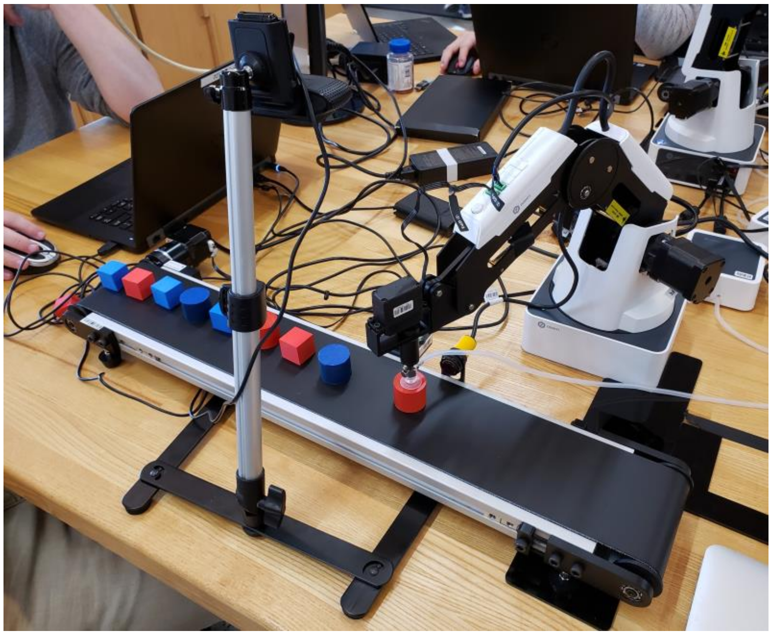


Figure 6: Sorting objects by shape detection

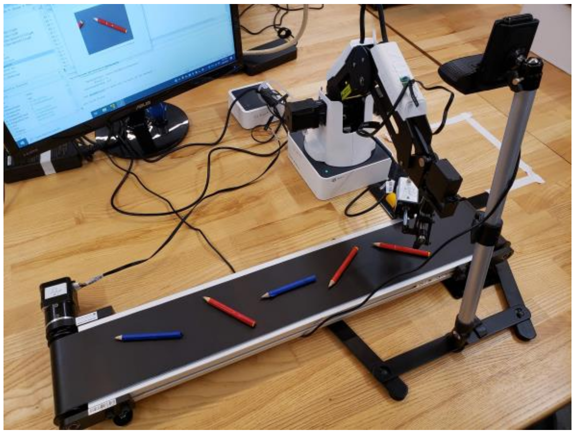
5.7 Laboratory #7: Pick up pencils from conveyor belt and stack in box

In the previous labs, the students used the suction cup end-effector to pick up the objects. In this lab, the pneumatic gripper (2 positions; fully open/fully closed) was used to pick up small pencils from the conveyor belt. The pencils were centered, with random spacing, and oriented parallel to the length of the conveyor belt. The goal was to gain experience operating the gripper and the servomotor connected to the gripper. Once the pencils were picked up by the gripper, the pencils needed to be rotated 90 degrees and stacked in a box. No camera was required in this lab. ROS services were used to open and close the gripper as well as control the servomotor to rotate the gripper mechanism.

5.8 Laboratory #8: Pick up pencils from conveyor belt and stack in box with orientation (web camera). Defective pencil detection option

Lab #8 is an extension of Lab #7 but in this case, the pencils are in random angles from the center line of the conveyor belt (see figure 7). The overhead camera is required to analyze the image of each pencil and determine the rotational orientation of the pencil so that the gripper can be oriented under software control to match the orientation of the pencil. The pencil is then picked up and stacked horizontally in a box in an ordered fashion with all pointed ends in the same direction. The angle of orientation is determined by the image in MATLAB using the ‘orientation’ property of the regionprops function. Red and blue pencils at various orientation angles were successfully picked up by the gripper mechanism. The students also modified the MATLAB code to identify yellow pencils by modifying the color detection algorithm. There was also an extra credit option to identify defective pencils which were shorter than standard 89mm (3.5-inch) length and to discard these rejects. Another optional challenge was to use computer vision tools to determine the orientation of the pencil (that is, determine which end is pointed).

Educators can modify and expand these labs and the requirements within the labs to meet the objectives of their courses.



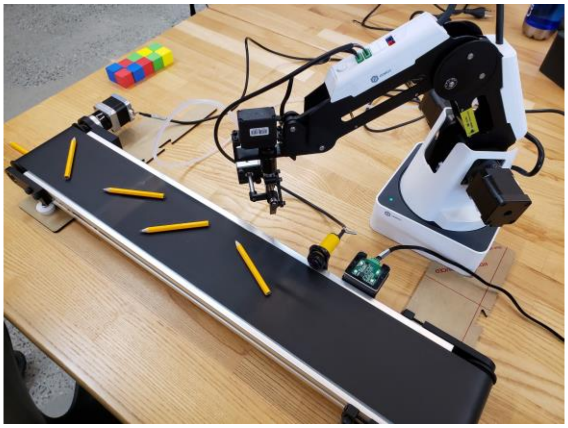


Figure 7: Pencil Pick-and-Place with Gripper

6.0 Conclusions and future directions

The case study evaluating the labs using a ROS/MATLAB interface to a Dobot robot arm was successfully implemented in fall of 2019 and early spring of 2020 in a senior-level robotics design and applications course. A total of 12 students (4 teams of 3 students each) were able to successfully complete the robot labs as described above. The set of labs was designed to be completed in a 4- to 6-week time frame assuming 2 scheduled lab sessions per week. This short time frame allows students to participate in more advanced robotics projects and additional modules in the course.

When students were asked in a survey (n =11) to rate the effectiveness of the Dobot robot hardware system, the response average was 4.0 (5=excellent, 4=good, 3=average, 2=below average, 1=poor). When asked to rate the effectiveness of using MATLAB and ROS software to program the Dobot robot arm and accessories, the student response average was 4.5. The rated effectiveness of the computer vision component was 4.4 (average score), and the effectiveness of the labs to train students in ROS services and ROS technology was reported as 4.5 (average score) from the student survey. One comment from the student feedback was that the robot arm would lose calibration at times and we discovered that it is important to keep the end-effector in the valid workspace of the robot.

One of the advantages of MATLAB is the support for interactive development, testing and visualization. External monitors were provided for each team so that all team members could participate in the coding and testing. Engineering students often enter a robotics course with varied programming backgrounds and experience levels – the MATLAB/ROS environment mitigated this issue. The computer vision concepts were successfully integrated into the robot arm, conveyor belt and sensor labs to allow the simulation of a real-world manufacturing, assembly or production environment. The small footprint, functionality and configurability of the Dobot Magician robot and conveyor belt was highly advantageous for an educational laboratory environment.

Challenges of this ROS/MATLAB approach described in this paper include: 1) a virtual machine software must be setup and installed with ROS and the Dobot ROS build files on the student computers and 2) custom message files must be installed and configured in MATLAB environment of the machines. These steps were relatively minor and were performed prior to the robot arm lab module and the students were insulated from these setup details.

It is hoped that the results of this research project will encourage other educators to develop additional resources for MATLAB and ROS programming of low-cost robot manipulators that are effective in the classroom and laboratory. These results also have significance to the introduction of modern robotics concepts, including industrial robots and intelligent manufacturing, into lower division engineering courses, K-12 and STEM activities.

7.0 References

[1] https://www.ros.org/ [Accessed April 26, 2020]

[2] S. A. Wilkerson, J. Forsyth, C. Sperbeck, M. Jones, and P. D. Lynn, “A Student Project using Robotic Operating System (ROS) for Undergraduate Research,” *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017. Available: https://peer.asee.org/27515 [Accessed April 26, 2020]

[3] A. Yousuf, W. Lehman, M. A. Mustafa, and M. M. Hayder, “Low-Cost Robot Arms for the Robotic Operating System (ROS) and MoveIt,” *2016 ASEE Annual Conference & Exposition*, New Orleans, Louisiana, June 2016, 10.18260/p.25584

[4] K.A. Khan and J. Ryu, “ROS-based Control of a Manipulator Arm for Balancing a Ball on a Plate,” *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017. Available: https://peer.asee.org/28809 [Accessed April 26, 2020]

[5] C. Luo, J. Wang, W. Zhao, and L. Wang, “Multi-Lab-Driven Learning Method Used for Robotics ROS System Development,” *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017. Available: https://peer.asee.org/28692 [Accessed April 26, 2020]

[6] A. Araújo, D. Portugal, M.S Couceiro, et al., “Integrating Arduino-Based Educational Mobile Robots in ROS,” *J Intell Robot Syst* **77,** 281–298, 2015. Available: https://doi.org/10.1007/s10846-013-0007-4 [Accessed April 26, 2020]

[7] S. Schiffer, et al (ed.), “Teaching Robotics with ROS,” *European Robotics Forum 2018 Workshop Proceedings of the Workshop on Teaching Robotics with ROS* (held at ERF 2018), Tampere, Finland, March 15th, 2018.

[8] R. L. Avanzato and C. G. Wilcox, “Work in Progress: Introductory Mobile Robotics and Computer Vision Laboratories Using ROS and MATLAB,” 2018 *ASEE Annual Conference & Exposition*, Salt Lake City, Utah. June 2018. Available: https://peer.asee.org/30072 [Accessed April 26, 2020]

[9] N. Rosillo, N. Montés, J. P. Alves, and N. M. F. Ferreira, “A Generalized Matlab/ROS/Robotic Platform Framework for Teaching Robotics,” In: Merdan M., Lepuschitz W., Koppensteiner G., Balogh R., Obdržálek D. (eds) *Robotics in Education. RiE 2019*. Advances in Intelligent Systems and Computing, vol 1023. Springer, Cham.

[10] J. M. Esposito, “The State of Robotics Education: Proposed Goals for Positively Transforming Robotics Education at Postsecondary Institutions,” *IEEE Robotics &Automation Magazine*, vol. 24, no. 3, pp. 157-164, 2017.

[11] https://www.dobot.cc/dobot-magician/product-overview.html [Accessed April 26, 2020]

[12] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ROS2*,” Proceedings of the 13th International Conference on Embedded Software (EMSOFT ’16)*. Association for Computing Machinery, New York, NY, USA, 2016, Article 5, 1–10.

[13] M. Quigley, B, Gerkey, and W. Smart, *Programming Robots with ROS*, O’Reilly Media, Inc., 2015.

[14] C. Fairchild and T. Harmon, *ROS Robotics By Example*, 2nd edition, Packt Publishing, 2017.

[15] J. Lentin and J. Cacace, *Mastering ROS for Robotics Programming*, 2nd edition, Packt Publishing, 2018.

[16] https://www.mathworks.com/help/releases/R2019a/robotics/ug/create-custom-messages-from-ros-package.html [Accessed April 26, 2020]