OzoBlockly Editor Challenge:

Factory Part Inspection – ORA Responds to Hand Gestures

Grades 9-12

Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

This OzoBlockly activity invites students to step into the role of automation engineers, exploring the cutting-edge intersection of robotics, computer vision, and human-computer interaction. In this simulation, students will program the Ozobot ORA to act as a factory transport unit that sorts manufactured parts based on visual commands rather than traditional keyboard inputs.

The challenge focuses on creating a system where the robot responds to specific hand gestures—such as a "thumbs up" for good parts, "thumbs down" for discards, and "two fingers" for rework—simulating real-world industrial environments where intuitive controls enhance safety and efficiency. The lesson also introduces the concept of a "wake-up" signal to prevent accidental commands. By guiding ORA to move items between inspection and destination stations, students will gain practical experience with logic loops, conditional programming, and event handling. This guide provides the necessary resources and solutions to help you facilitate this hands-on exploration of modern automation technologies.

If you have not already done so, you should now read through the *Student Guide* for this challenge for details on what is expected from the students.

Example Solution to this Challenge

The solution to this challenge provides the opportunity for your students to develop an OzoBlockly Editor program that is modular, dividing the program logic into functions each having a specific purpose. Figure 1 shows the main program, which contains the major functions. The *repeat forever* loop is used to allow an indefinite number of inspections which can be stopped by the inspector, terminating the program by a *break out of loop* block.

The *"Initialization"* function provides some preliminary settings. The *"Capture Image of Hand Gesture"* function is where the LLM and connected camera take a snapshot of the inspector's hand gesture. The *"Notice that Image Has Been Captured'* function simply informs the inspector when the photo of the hand gesture is complete, so the inspector can stop the hand gesture. A nested series of *ifs* then move the package to the correct location on the factory floor, based upon the instruction from the hand gesture. The use of *else if* gives priority to good, then rework, and finally to discard items, based upon factory statistics indicating that good items are most likely, then rework, and finally discard items are least likely.

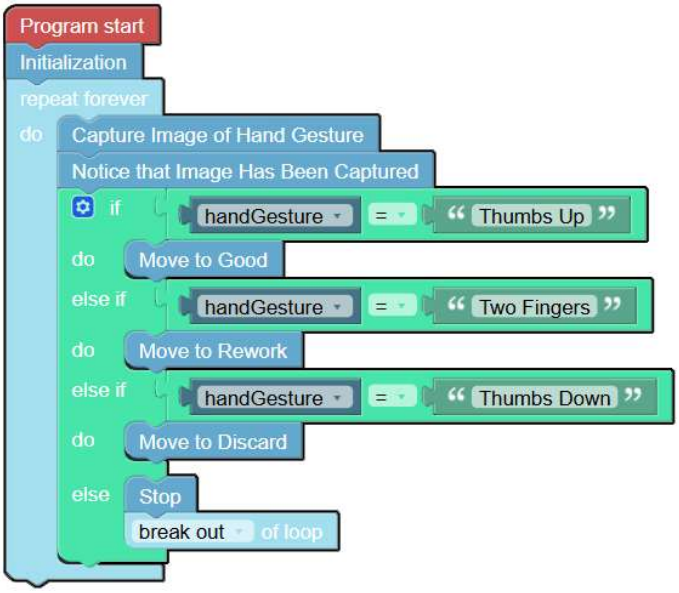

Figure 1

Figure 2 shows the "*Initialization*" function. Ari's top light color is set to white, ORA's speed is set to 20% and is moved to the HOME location. The tool is set to the vacuum gripper and turned OFF.

Figure 2

Figure 3 shows the "*Capture Image of Hand Gesture*" function and the *"Notice that Image Has Been Captured"* function. The prompt was worded so that the string variable *handGesture* would ultimately contain the either *Thumbs Up*, *Thumbs Down*, *Two Fingers*, or *Stop*. **Note that using the *Google (standard)* LLM seems to provide the most accurate determination of the hand gesture.** Printing the value of the variable *handGesture* to the IO console is optional, but quite helpful, particularly when developing/debugging the program.

The *"Notice that Image Has Been Captured"* function lets the inspector know that the photo has been taken. Two sensory signals are used—visual and sound. First, the top light of Ari is changed to red. Second, an audible beep is played by Ari.

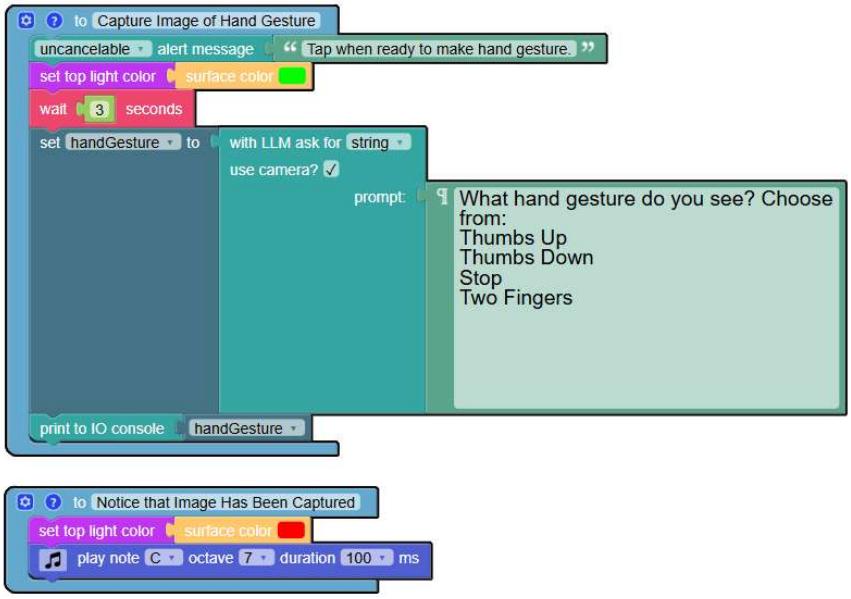

Figure 3

Figure 4 shows the *"Move to Good", "Move to Rework", and "Move to Discard"* functions. Each one of these three functions first calls the *"Pick up Part at Inspection Station"* function. Then the coordinates (*xDestination*, *yDestination*) are set, specifying the location of the destination station. Finally, the *"Place Part at Destination Station"* function is called.

Figure 4

Figure 5 shows the "*Pick up Part at Inspection Station*" function. The first *move to coordinate* block moves ORA's arm to the inspection station at a height of 30 mm. The second move to coordinate block lowers the arm to 2 mm, so that the vacuum gripper can contact the part. The vacuum gripper is turned ON, and the part is lifted to a height of 30 mm.

Figure 5

Figure 6 shows the *"Place Part at Destination Station"* function. The (*x, y*) coordinates are set to (*xDestination, yDestination*) at a height of 30 mm. The part is then lowered to 5 mm, and the vacuum gripper is turned OFF, releasing the part at the station.

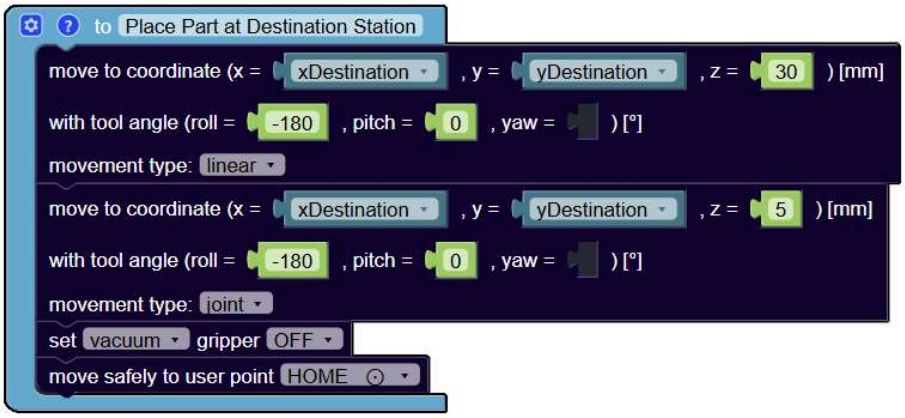

Figure 6

Figure 7 shows the "Stop" function that was initiated by the stop hand gesture. Ari plays a note five times to indicate that the inspection process is about to stop. Finally, Ari's top light is set to white and ORA's arm is moved to the ZERO location

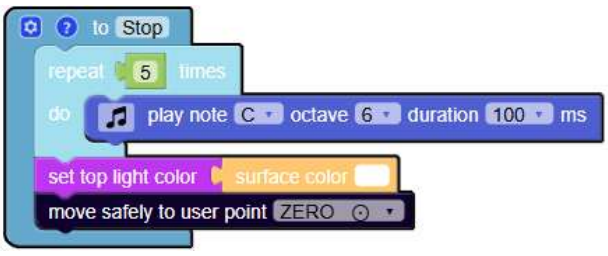

Figure 7

Share Code for the Solution – 884z3fd

Link to 3D STL File - Package Holder

ISTE Standards (International Society for Technology and Education)

Empowered Learner 1.1.d: Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

Knowledge Constructor 1.3.d: Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Computational Thinker 1.5.c: Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

CSTA (Computer Science Teachers Association) Standards

2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
2-AP-19	Document programs in order to make them easier to follow, test, and debug.
3A-DA-11	Create interactive data visualizations using software tools to help others better understand real-world phenomena.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3B-CS-02	Illustrate ways computing systems implement logic, input, and output through hardware components.
3B-AP-08	Describe how artificial intelligence drives many software and physical systems.
3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects.