ORA: AI-Guided Sorting and Coordinate Transformation Challenge

Grades 9-12

Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

Designed to immerse students in the dynamic realm of industrial automation, this lab combines artificial intelligence with robotics, offering a comprehensive learning experience. Students will take on the role of robotic programmers as they direct ORA to perform a complex sorting operation that emulates real-world smart manufacturing processes.

This engaging challenge goes beyond traditional programming, as students will combine ORA's integrated camera and Large Language Model (LLM) to analyze shapes on a grid. The ability to identify and locate objects introduces an essential component of robotics: the integration of vision-based AI. Through the process of coordinate transformation, students will learn to convert two-dimensional paper grid coordinates into ORA's three-dimensional workspace, allowing ORA to navigate accurately to each shape's center.

In addition to enhancing their technical skills, students will undertake a critical thinking exercise as they develop the logic necessary for ORA's pick-and-place routines. By sorting shapes into designated bins based on their type, students will draw parallels to advanced sorting systems in fulfillment centers, showcasing the collaborative potential of AI and robotics in handling complex tasks. Through this hands-on experience, your students will cultivate essential skills such as problem-solving, creativity, and teamwork, all while engaging with cutting-edge technology.

The accompanying resources, including visual aids and instructional videos, are tailored to support educators in guiding students through this transformative challenge. By the end of this lab, students will emerge with a robust understanding of how AI and robotics work together, preparing them for future innovations in technology and automation.

You should also view the accompanying 15-second time-lapse video that was mentioned in the student guide. This video provides a detailed view of the program at work as ORA moves the circles and squares from the grid to their respective bins.

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. In the solution presented here, variable and function names have been created in a way that promotes self-documentation.

Figure 1 shows the main program and the "*Initialization*" function. The blocks in the initialization function are self-documenting. Note that the speed is set to a low value of 20% for safety.

The main program begins with a call to the initialization function. The blocks in the repeat loop are executed six times, once for each of six shapes that are manually placed one-at-a-time on the grid. Each time through the loop, four functions are called in succession: *Capture Image, Compute Coordinates, Pick Up Shape from Grid, and Move Shape to Its Bin*.

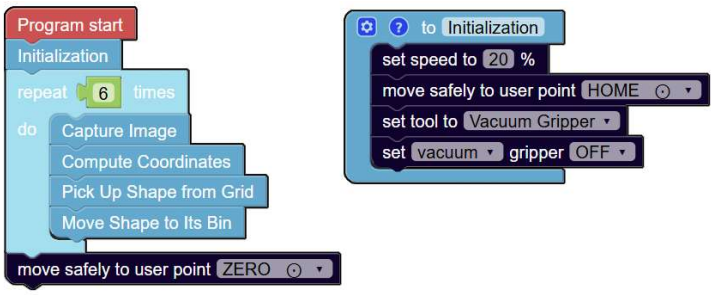

Figure 1

Figure 2 shows the "*Capture Image*" function. The LLM is asking for a string and is using a camera. The prompt has been designed to return the letter C for a circle or R for a rectangle. The letter is followed by a blank space, which is in turn followed by the x and y coordinates of the center of the shape. Theses coordinates are separated by a blank space. The resultant string is stored in the variable *dataFromLLM*. An example value for this variable would be the string *C 120 80*, indicating a circle at paper grid coordinates (x, y) = (120, 80). The prompt was designed so that the variable *dataFromLLM* would be relatively easy to parse.

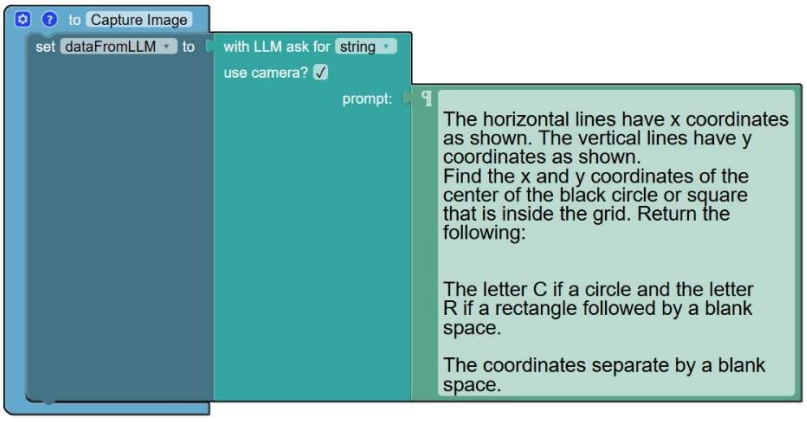

Figure 2

Figure 3 shows the "*Compute Coordinates*" function. This critical function does two things. First, it parses the *dataFromLLM* variable so that the shape is stored in the string variable *shape*, and the x and y paper coordinates are stored in the numeric variables *xCoord* and *yCoord,* respectively. Keeping in mind that the characters in a string have indices starting at zero, the first and second

blank spaces in the variable *dataFromLLM* string are stored in the variables *spaceCharacterIndex* and *spaceCharcterIndex2*, respectively. The paper grid variables *xCoord* and *yCoord* are converted to numbers using the *convert to* block. The *print to IO console* blocks are particularly useful when developing and debugging the program. If desired, then can be removed from the final program.

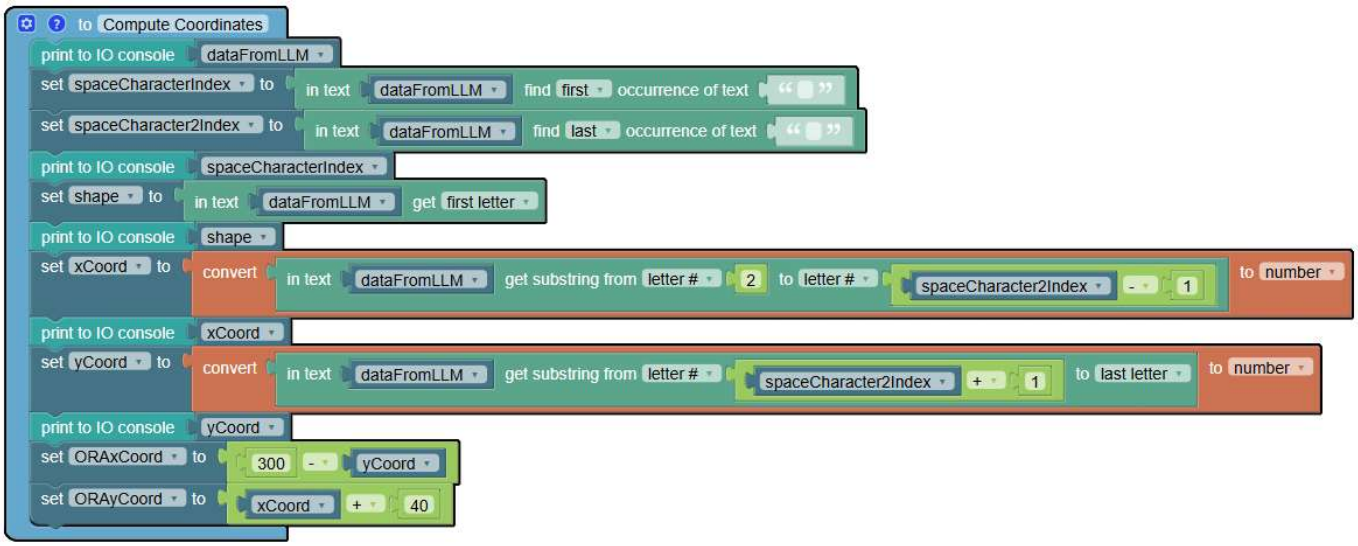

Figure 3

The final two blocks in the *Compute Coordinates* function are where ***coordinate transformation*** takes place, *i.e.*, converting from paper grid coordinates to the ORA coordinate system. Figure 4 is used as an aid to understand this transformation.

Everything shown in red relates to the ORA coordinate system, with the origin (0, 0) at the center of the ORA base. The +y axis is to the right, the -y axis is to the left, and the +x axis is downward. The paper grid is shown in black with its +y axis upward and its +x axis to the right. The upper left corner of the paper grid, with coordinates (0, 160) was placed at ORA coordinates (140, 40). Each of the squares on the paper grid is 40 mm by 40 mm. This means that the upper right corner of the paper grid has ORA coordinates (140, 200). In a similar manner, you can determine that the lower left and lower right corners of the paper grid have ORA coordinates (300, 40) and (300, 200), respectively.

Table 1 shows the mapping of the x paper coordinates to the y ORA coordinates. Visual inspection of the data in Table 1 shows that $\mathbf{y_{ORA} = x_{paper} + 40}$. Table 2 shows the mapping of the y paper coordinates to the x ORA coordinates. Again, visual inspection shows that $\mathbf{x_{ORA} = 300 - y_{paper}}$. The two equations in bold face above explain the final two *set-to* blocks in Figure 3.

In general, visual inspection is not this easy. Therefore, students can always make use of the following two equations when two points (x1, y1) and (x2, y2) lie on a straight line. The slope m is given by the equation $\mathbf{m = (y2 - y1)/(x2 - x1)}$. The equation of a straight line with slope m and y-intercept b is given by y = mx + b. Therefore, b can be found by using the equation $\mathbf{b = y - mx}$ and substituting the values of x and y by the corresponding values for any point on the straight line.

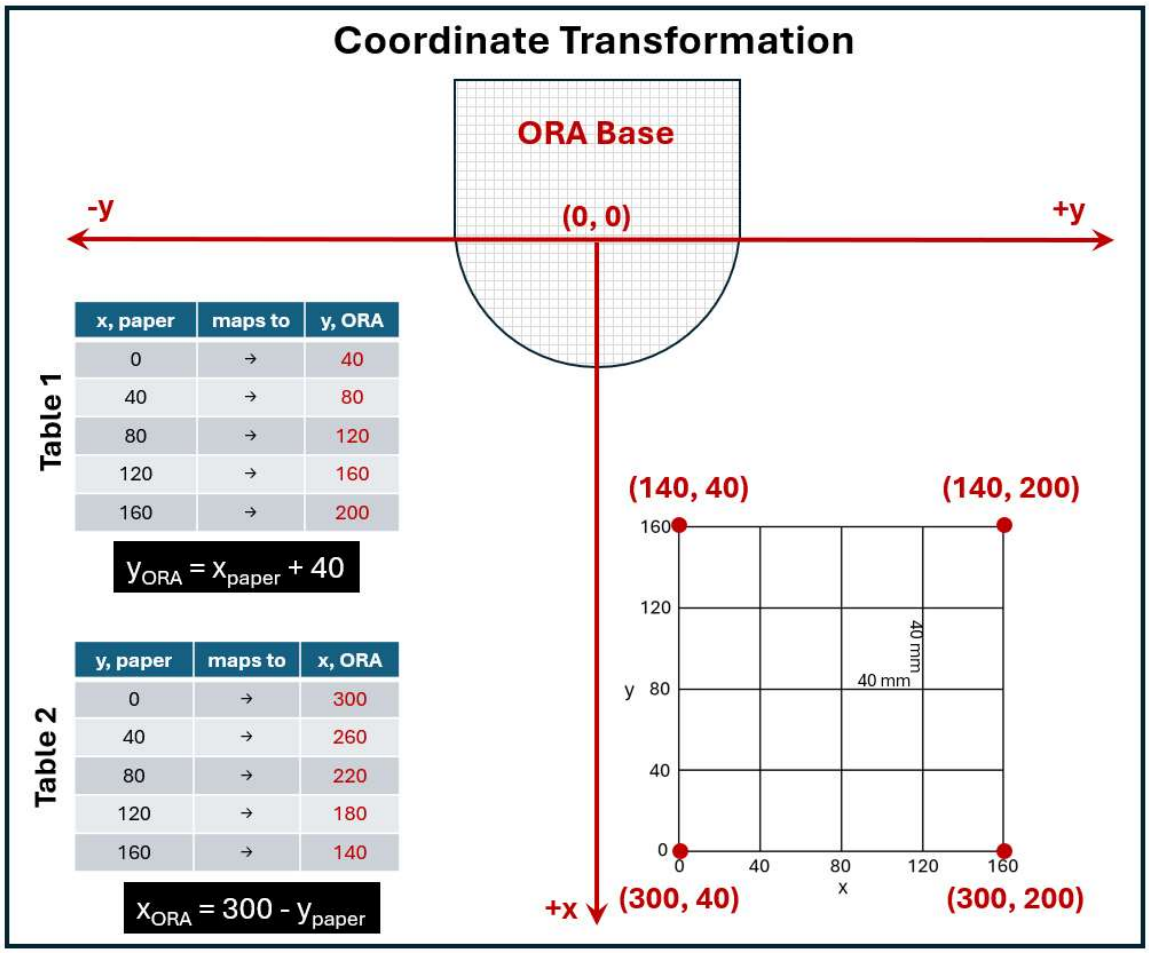

Figure 4

Figure 5 shows the "*Pick Up Shape from Grid*" function. The first *move* block moves the gripper to the location (ORAxCoord, ORAyCoord) of the shape at a height of 80 mm. The second *move* block lowers the gripper so that the suction cup is in contact with the shape. The vacuum gripper is then turned ON, and the shape is lifted back to a height of 80 mm.

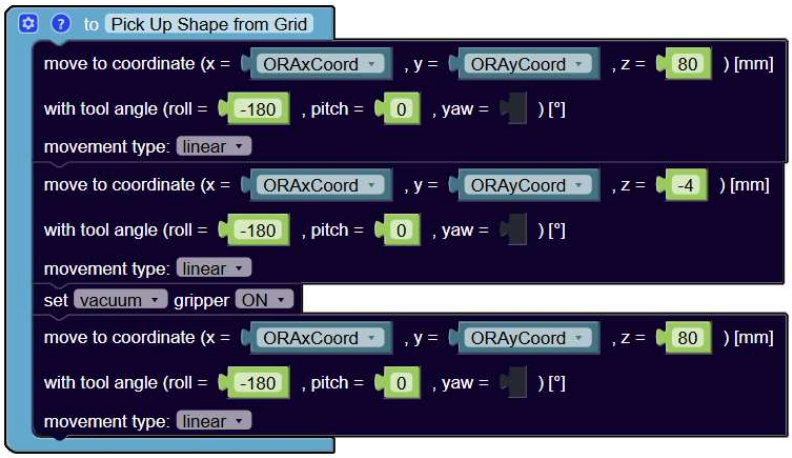

Figure 5

Figure 6 shows the "*Move Shape to Its Bin*" function. An *if-do-else* block determines if the shape is a circle or a rectangle and sets the y placement location variables appropriately. The x placement location is 225 for both shapes. The first *move* block moves the shape to a height of 80 mm, the second *move* block lowers the shape to 10 mm, the vacuum gripper is turned off releasing the shape to its bin, and finally the gripper is moved back to a height of 80 mm.

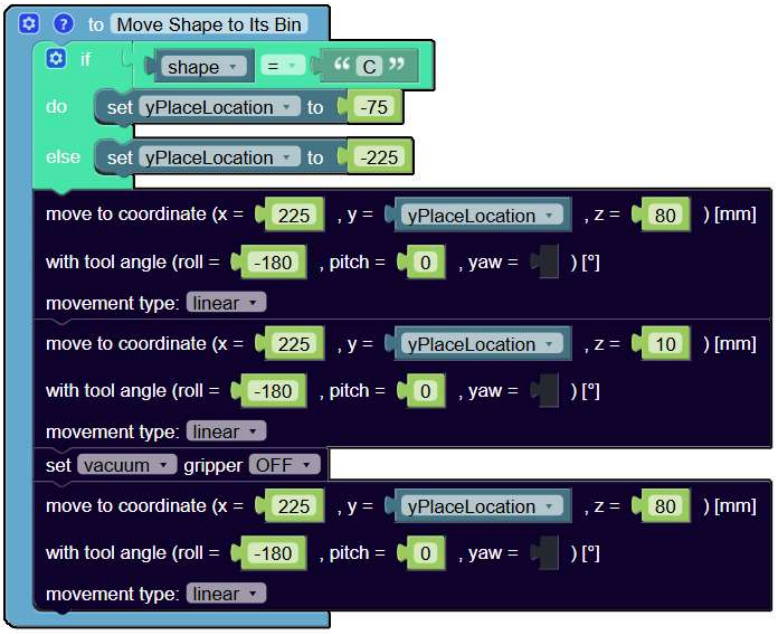

Figure 6

Share Code for this Lesson – bzgd2nz

Additional Tips

1. The "Standard" LLMs seem to be *much* more accurate than the "Mini" LLMs.
2. The camera should be placed so that the paper grid fills most of the camera's field of view, giving better resolution of the image.
3. It is best if the circles and squares are centered only on the nine internal grid coordinates where grid lines intersect. The LLMs do not seem to do well with interpolation.
4. The squares can be at angles as long as they are centered on grid line intersections.
5. The paper grid should be well lit to improve the camera's view.
6. Adjustments may be necessary if the size of the square on the printed paper grid is not reasonably close to 40 mm by 40 mm.

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.