ORA: Investigating Sensor Fusion

Grades 9-12

Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

Welcome to the Teacher Guide for "Investigating Sensor Fusion," a challenge designed to immerse students in the complexities of advanced robotics and artificial intelligence. This lesson focuses on the critical concept of sensor fusion, where data from multiple sources are combined to produce more accurate and reliable results than a single sensor could achieve alone. By using the OzoBlockly Editor and ORA, students will simulate the high-level processing used in modern innovations like self-driving cars and aerospace systems.

In this exercise, students will program ORA to interpret two distinct sensory inputs from specially designed color cards—the printed name of a color and the physical hue of the card background. Utilizing an LLM-powered camera system, the robot must perform optical character recognition and visual analysis simultaneously. The students' primary task is to implement logic that determines whether these data points match, leading the robot to sort the cards into designated "agree" or "disagree" zones based on its findings.

This challenge provides a hands-on opportunity for students to engage with the logic required for machines to navigate conflicting information in real-time. It moves beyond basic robotic movements and into the realm of intelligent decision-making and error detection. As an educator, you can use this lab to spark discussions about how artificial intelligence perceives the physical world and the importance of cross-referencing data to ensure system reliability and safety in complex environments.

Finally, this lesson places a heavy emphasis on industrial safety protocols within the OzoBlockly programming environment. It is vital to ensure that students include speed-limiting blocks at the start of their programs to allow for better control and intervention during the testing phase. You should also verify that the Emergency Stop Button is within easy reach at all times. These precautions not only protect the hardware but also teach students the professional standards necessary for human-robot collaboration.

If you have not already done so, you should now view the accompanying 18-second time-lapse video that was mentioned in the student guide. This video provides a detailed view of the program at work as ORA analyzes the OCR and background color of the cards.

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. Following the "*Initialization*" function is a repeat forever loop. The loop begins with a call to a function that uses an LLM and camera to get the word on the card. Next, a function is called that parses the string obtained by the LLM and camera. The word seen by the camera is stored in the variable OCR. If the word is "Stop", then there is a break out of the loop, ORA moves to the ZERO position, and the program is terminated. If the word is something other than "Stop", presumably a color, then the card is picked up and placed at either the agree or disagree zone, as appropriate.

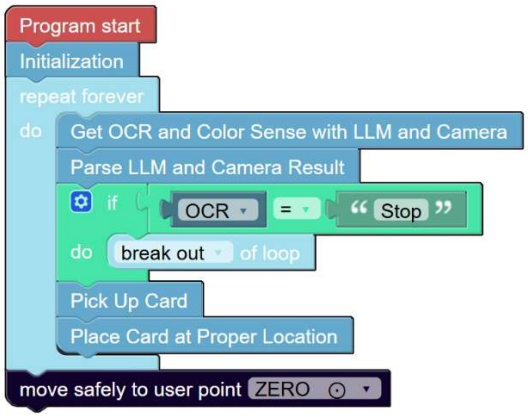

Figure 1

Figure 2 shows the "*Initialization*" function. ORA's speed is set to 60%, though it is wise to go with a lower speed, say 20%, during program development, for safety reasons. The tool is set to the vacuum gripper and is OFF. To improve program documentation, the variable *spaceCharacter* is initialized as a blank space. Finally, the ORA *y* coordinates for the *agree* and *disagree* locations are specified as -200 and 200, respectively.

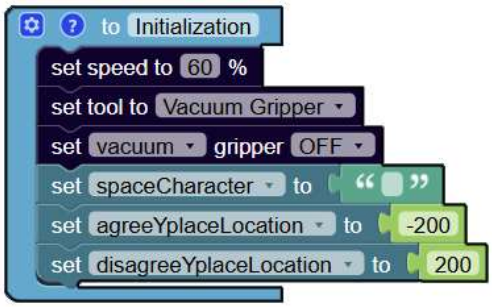

Figure 2

Figure 3 shows the "*Get OCR and Color Sense with LLM and Camera*" function. The prompt has been carefully constructed so that the string variable *result* will be two words, beginning with upper case letters and separated by a space. The first word in *result* is the word that was printed on the card. The second word is the color that was sensed in the background of the card. For example, the string "Green Green" would represent agreement, whereas the string "Yellow Orange" would represent disagreement.

It is strongly recommended to use a *try* block that prints any exception message to the IO console. The most common message might indicate that LLM credits are exhausted, though a message indicating that the LLM is experiencing large volumes of requests has also been observed. In this case the message has suggested "trying later".

The author found the *OpenAI (standard)* LLM to be the most accurate. The value of the *result* variable is printed to the IO console. This is particularly helpful during program development but can be removed later if desired.

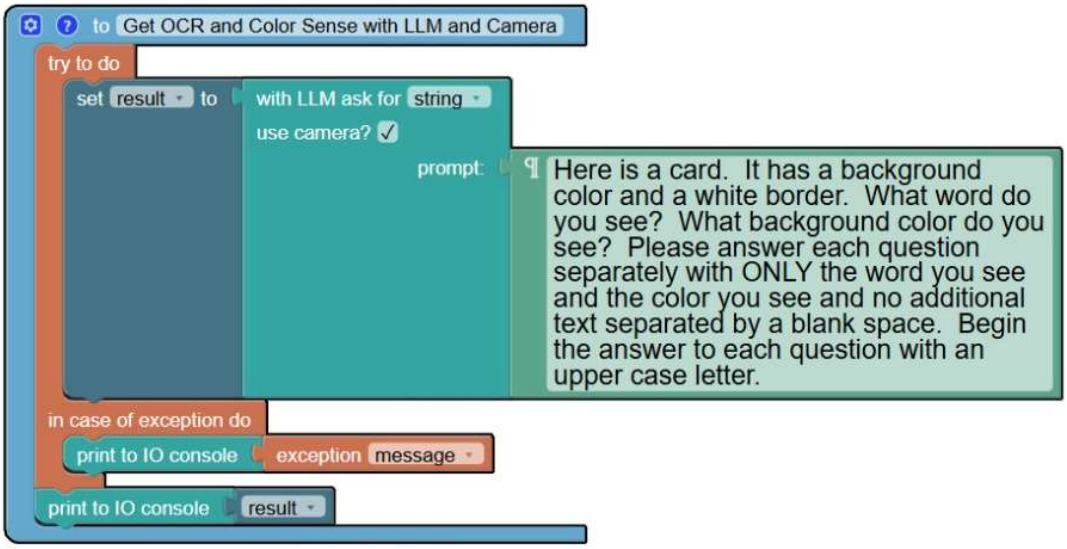

Figure 3

Figure 4 show the "*Parse LLM and Camera Result*" function. The purpose of this function is to take values of the *result* variable, *e.g., Yellow Orange*, and separate out the two words, storing them in the variables *OCR* and *colorSense*. In this case, the variable *OCR* would contain the string *Yellow* and the variable *colorSense* would contain the string *Orange*.

The first block in this function determines the index of the space in the result. The second block captures the substring from the first character in result to the character whose index is one less than that of the space. The third block captures the substring whose index is one greater than that of the space through then last character in the variable *result*. Remember that *zero based indexing* is used in strings, so that the index of the first character in any string is 0 (zero).

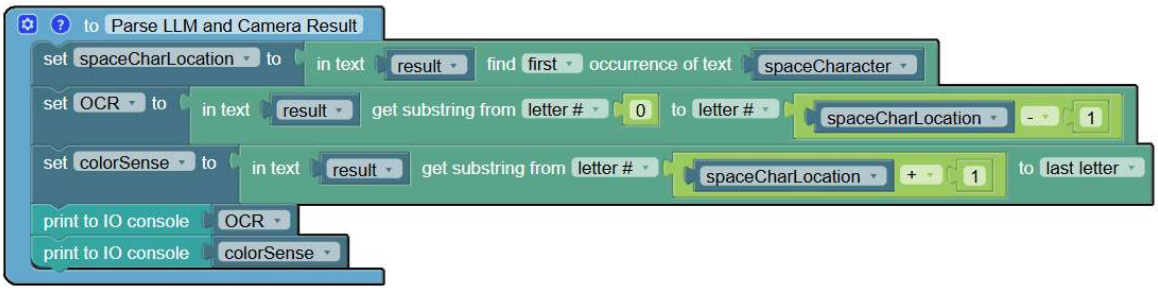

Figure 4

Figure 5 shows the "*Pick Up Card*" function. The first *move* block moves the vacuum gripper to the card at (x, y) = (380, 0) at a height of 30 mm as the card rests in view of the camera. The second move block lowers the vacuum gripper to a height of -4 mm, where it contacts the card. The vacuum gripper is then turned ON, and the card is raised to a height of 30 mm in the third *move* block.

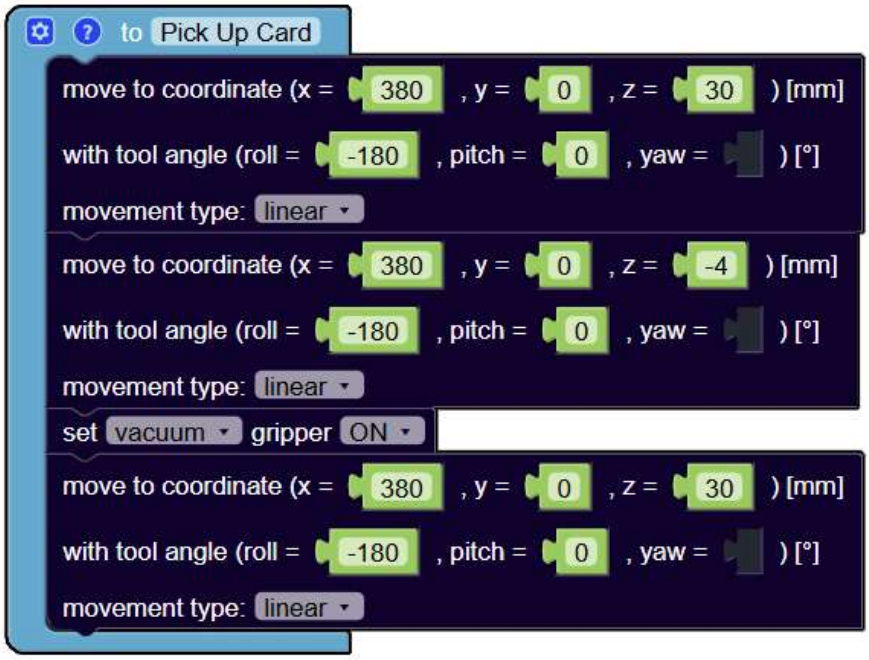

Figure 5

Figure 6 show the "*Place Card at Proper Location*" function. If the *OCR* and *colorSense* variables match, the *y* coordinate of the proper location is set to *agreeYplaceLocation*. Otherwise, the *y* coordinate of the proper location is set to *disagreeYplaceLocation*. The ORA *x* coordinate for both locations is the same, 225 mm.

The first move block moves the card to (225, *y*) at height of 10 mm. The vacuum gripper is then turned OFF, releasing the card to its zone. The second move block then raises the vacuum gripper back to a height of 30 mm.

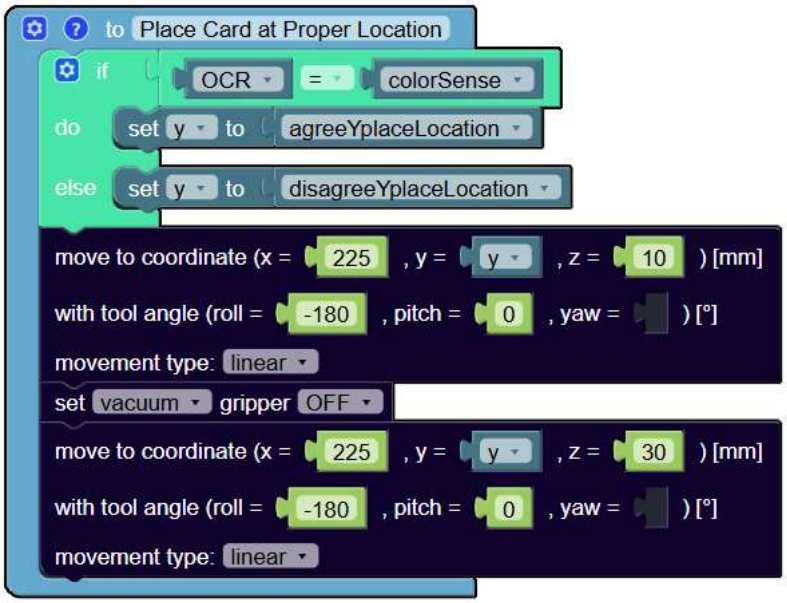

Figure 6

Share Code for this Lesson – x8f45sm

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.