Lesson

SIMULATION OF A ROBOTIC WAREHOUSE

CREATED BY

Richard Born
Associate Professor Emeritus
Northern Illinois University
rborn2@niu.edu

TOPICS

Robotics, Programming

GRADES

7-12

METHOD

OzoBlockly

DURATION

60 minutes

Simulation of Robotic Warehouse Picking

By Richard Born
Associate Professor Emeritus
Northern Illinois University
rborn2@niu.edu

Introduction

We humans don't seem to be built for repetitive tasks. Over time we tire, slow down, make errors, and even take on physical problems. A very repetitive task is the picking of items from shelves in a warehouse to fulfill customer orders. Here is a situation for which robots provide a significant advantage over manual methods. Companies have developed robotic warehouse picking systems using a variety of techniques including the use arms that can grab fairly heavy items from a shelf and the use of suction to hold lighter items such as health and beauty goods, pharmaceuticals, and groceries. Some of these robotic pickers even have vision systems. Picking rates of 200 per hour, typical of human picking speed, are possible with today's robotic technologies, making them competitive with manual techniques, and without the associated disadvantages.

A nice way to introduce students to robotic warehouse picking systems is to simulate these systems using either Ozobot Bit or Evo. This is exactly what we'll be doing in this lesson. Students will study an OzoBlockly program in which Ozobot traverses a warehouse floor while stopping to "pick" items from shelves in order to fulfill customer orders. Students will compare picking rates for two layouts for shelves, an existing layout and a new proposed layout. Customer orders will be simulated by the use of random numbers in OzoBlockly, and students will learn how the use of a *seed number* allows comparing the two shelf layouts by *controlling the random input variables*.

Problem Statement

Dawn t' Dusk, a (fictitious) nationwide "mail" order house that specializes in certain household goods for the 21st century, receives hundreds of orders each day, some through snail mail, some via telephone, and some from their Web site. They have been experiencing growth since the company began operation five years ago, and are currently looking at how to deal temporarily with this growth before their new warehouse is complete a year from now. They are hoping that some modest modifications to their existing facility will get them through the next year, helping them to handle increased orders from their continually growing customer base.

Figure 1 shows their existing (left) warehouse picking layout and proposed (right) layout. Both have 16 shelf regions at which the robot can stop to pick goods for orders. The long black lines show the path of the robot, and the intersections with the short black lines show where the robot stops to pick items from the shelves. The proposed layout places the shelves into a single row of shelves rather than two separate rows. The proposed layout has the advantages that (1) it takes up less floor space, freeing space for other tasks, and (2) the total length of the path followed by the robot in traversing the shelves is less, allowing more items (and hence orders) to be picked per hour. ***Dawn t' Dusk* management would like to know, in advance of**

implementing the proposed layout, how much the order picking rate would increase with the proposed layout. As you can imagine, doing a simulation before implementing a proposed system could be much less costly than carrying out the physical implementation only to find out that it doesn't work as anticipated. This is a very common use for simulation modeling.

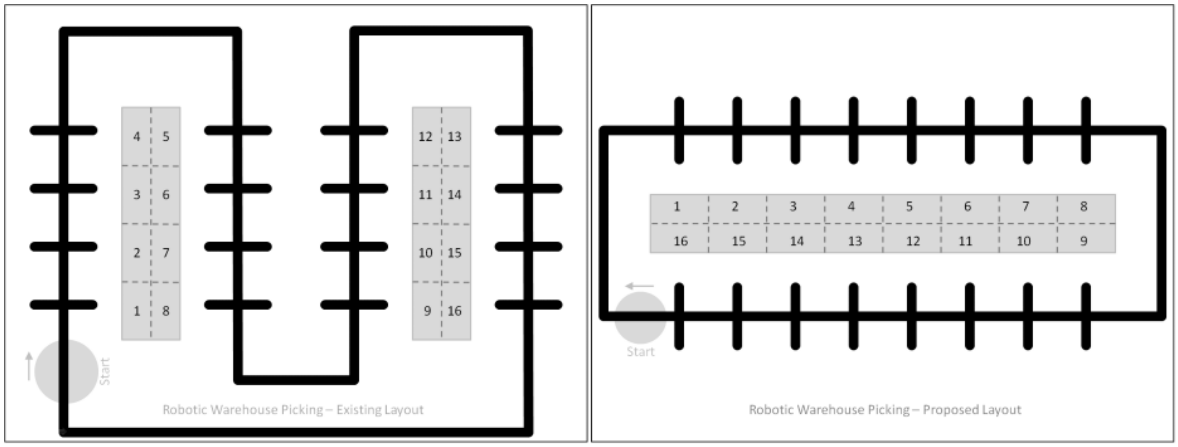


Figure 1

As shown in Figure 1, each layout consists of 16 locations where the Ozobot robot can stop to pick up items for an order. All 16 locations are equally likely and randomly selected. You can assume that the maximum number of picks for a given order is 6, and the probabilities for orders with 1, 2, 3, 4, 5, or 6 picks are all equally likely. Any given stop for a pick is just for a single item at the shelf. Each run of the OzoBlockly program should simulate Ozobot picking a total of five different orders. It should be noted that the final two pages of this document contain larger OzoMaps of the two layouts that can be duplicated for students to use with their Ozobots and this lesson.

Random Number Generation Primer

A key component of any simulation model is the production of random numbers. These are used to represent things such as customer arrivals, service times, and in the case of our robotic warehouse picking simulation, the number of picks for each order and shelf location for each pick.

Truly random numbers can be obtained by rolling a die, or drawing well-mixed, numbered ping-pong balls from a container, as done in lotteries. Computers, however, produce what are referred to as *pseudorandom* numbers. If you look up the prefix *pseudo-* in a dictionary, you will find typical definitions including *false*, *unreal*, *pretended*, and *having the appearance of*. Pseudorandom numbers are not actually random, but have the appearance of being random, as the resulting sequence of numbers exhibits many of the statistical properties of randomness. Digital computers produce them from *mathematical algorithms* beginning with what is known as a *seed number*. If the seed number is kept the same each time a program is run, then the sequence of "random" numbers will be the same every time the program is run. If the seed number is different each time the program is run, then the sequence of "random" numbers will differ with each run. Rather than putting the word *random* inside of double-quotation marks, we'll just use *random* from here on out to mean *pseudorandom*.

An OzoBlockly program running on Bit will produce the same sequence of random numbers each time the program is run *if a block from the math group "set random number generator seed" appears near the start of the program before any calls for random numbers*. The seed number cannot exceed 127. The same is true for Evo, but for a given seed number, the sequence of random numbers produced by Bit and Evo are different. If the seed number is not set, then the program on Bit or Evo will produce a different sequence of random numbers each time the program is run.

If you look at the JavaScript Preview of the program in OzoBlockly, you will notice that inclusion of a block "set random number generator seed" to 57, corresponds to the JavaScript statement:

```
system_RandomGeneratorSeed(57);
```

So what happens if you *don't* include a block to set the random number generator seed? Most system random number generators use a parameter from the system's clock to set the seed value. The result is that each time the program is run, the sequence of random numbers is different.

You might be wondering what all this talk about randomness has to do with our robotic warehouse picking simulation model. Recall that we are comparing two layouts for the picking area of the warehouse—the existing layout and a proposed new layout, and we are interested in determining how much the order picking rate would increase under the proposed layout. If we run Ozobot on these two layouts, we need to properly control variables. ***Specifically, we want to make sure that any differences in order picking rates are due to differences in the layouts and not due to differences in the number of picks for each order and shelf location for each pick.*** The number of picks for each order and shelf location for each pick are random processes in our simulation model. To make sure that these random numbers are the same for each layout, we simply include a block to "set random number generator seed" to a specific value. The random streams will then be the same for both layouts, as you will see when you run the OzoBlockly program. You have controlled these random variables. Any differences in picking rates for the two layouts must be due to differences in layout design.

Robotic Warehouse Picking Video

Before studying the OzoBlockly program, it is suggested that the video *RoboticWarehousePicking.m4v* accompanying this document be viewed now to clarify what the OzoBlockly program does. In order to keep the video short, some of the wait blocks have shorter wait times than in the actual OzoBlockly program. You will notice that when Bit/Evo is moving, a green LED is displayed. When he has stopped to pick an item from a numbered shelf, a red LED is displayed. When picking for an order is complete, Bit/Evo stops at shelf 16 momentarily and blinks a white LED a few times to indicate the order is complete. He then proceeds to pick the next order. In the video, two different orders are picked. The first order shows five picks, from shelves 1, 3, 7, 9, and 12. The second order shows three picks, from shelves 7, 12, and 16.

Array Declaration and Main Program

Figure 2 shows an array declaration block and the blocks making up the main program. The array declaration block can be placed anywhere in the OzoBlockly workspace. It declares an array called *stopLocations* and has a length of 17. Therefore, array subscripts vary from [0] through [16]. We will not make use of the element with the subscript [0] to simplify things a little. The element with the subscript [1] refers to shelf 1, subscript [2] to shelf 2, and so on through shelf 16. The purpose of the array is to indicate which shelves Ozobot is to stop at while picking the order. If an array element has a value of 1, then Ozobot is to stop at that shelf location. If the value of the element is 0, then Ozobot should not stop at that shelf location.

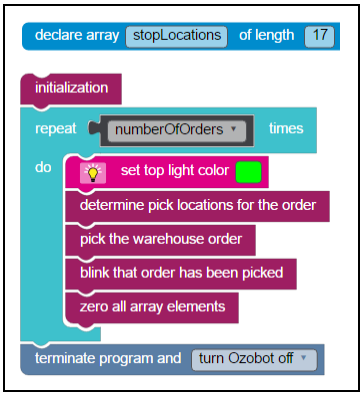


Figure 2

The first thing done in the main program is some housekeeping through the initialization subroutine, discussed in the next section of this document. Then a loop is executed once for each order that Ozobot is to pick. In this loop the LED is set to green to show that Ozobot is moving. Then a subroutine is called to randomly pick locations for the order. Next a subroutine is called which has Ozobot actually picking the order. When the order has been picked, a subroutine is called that causes Ozobot to blink several times to indicate this fact. The last task in the loop calls a subroutine to reset all elements of the array to zero to get ready for the next order. When the loop has completed all iterations, the program is terminated an Ozobot is turned off.

"Initialization" Subroutine

Figure 3 shows the blocks for the initialization subroutine, which does some one-time things at the start of the program.

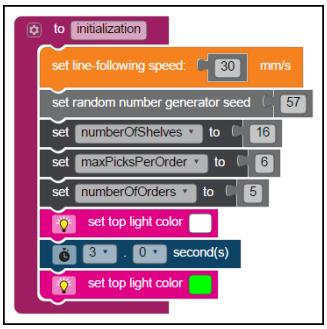


Figure 3

The line-following speed is set, and the random number generator seed is set. Then the values of three program variables are set. The number of possible stop locations (number of shelves) is set to 16. The maximum number of picks for an order is set to 6, and the number of orders that Ozobot is to pick is set at 5. The LED is initially set to white for three seconds. This delay allows the student to start Ozobot while holding it and then setting it down on the origin before it starts moving. Finally, the initialization subroutine sets the LED green when Ozobot begins his path through the shelves.

"Determine Pick Locations for the Order" Subroutine

Figure 4 shows the blocks making up the "determine pick locations for the order" subroutine.

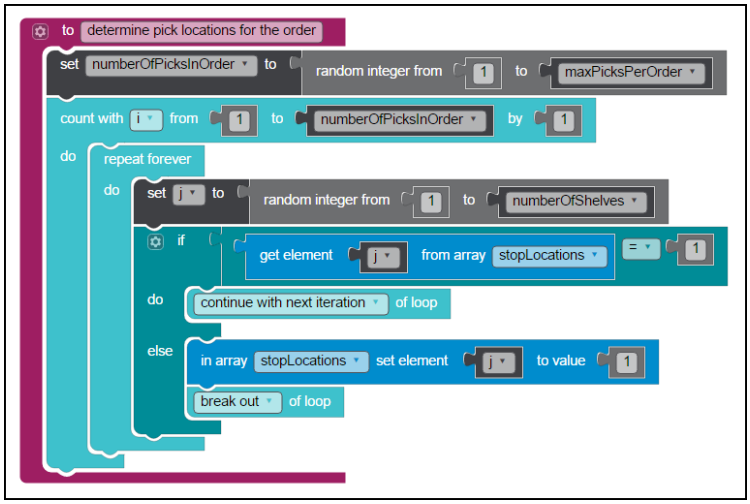


Figure 4

The subroutine begins by setting a variable holding the number of picks for the order to a random value from 1 to the maximum number of picks allowed for the order. Then a loop with a loop counter *i* is executed once for each of the picks in the order. Nested in this loop is a *repeat forever* loop. The *repeat forever* loop will actually stop looping because of the use of the *break out of loop* block in the loop. Within the *repeat forever* loop, a variable *j* is set to a random integer between 1 and the number shelves (16). An *if…else* block then checks to see if the *j*'th element of the array of stop locations is equal to 1. If it is, then that shelf location is already in the picks and next iteration of the *repeat forever* loop tries again. If the *j*'th element is not equal to 1, then the array location is set to one to indicate that the shelf is now a pick. In this case, we break out of the *repeat forever* loop and continue with the next pick

"Pick the Warehouse Order" Subroutine

In this subroutine Ozobot either moves past a shelf if there is not a pick there, or stops at the shelf if there is a pick at that shelf. This is accomplished by a loop for which the loop counter *k* varies from 1 to the number of shelves. Inside this loop, Ozobot follows the line to the next intersection and picks the direction straight when arriving at the intersection. The *k*'th element of the array of possible stop locations is checked to see if it has a value of 1. If it does, then that means that Ozobot is to stop there and pick an item from shelf *k*. Ozobot stops, sets the LED to red, and waits for 5 seconds, during which time the item is removed from the shelf. If

the *k*'th element is not 1, then it is 0, and Ozobot moves forward and continues with the next iteration of the loop.

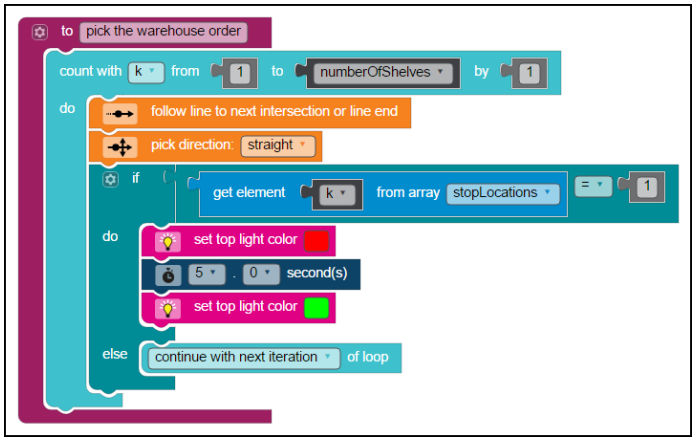


Figure 5

Here is an excellent time to discuss the amount of detail that needs to be included in a simulation model. With some (perhaps a lot!) of additional programming effort, we might have been able to make Ozobot turn toward the shelf momentarily in the event that an item is to be picked from the shelf. But one needs to consider if doing so will help our goal of accurately determining the difference in picks per hour for the existing and proposed warehouse layouts. We already have Ozobot waiting 5 seconds to allow for the actual picking process from the shelf. Adding additional turning motion for Ozobot is not likely to affect the number of picks per hour, since we would then reduce the wait time so that the turning motion plus wait time is still 5 seconds. And we have saved the time required to program this additional movement. We should also mention that in a real robotic warehouse picking operation, the robot may in fact not turn toward the shelf. Rather it might "simply" grab the item from the shelf with an arm and drop it into a receiving container.

The Remaining Two Subroutines

Figure 6 shows the blocks for the final two subroutines of our OzoBlockly program, "blink that order has been picked" and "zero all array elements".

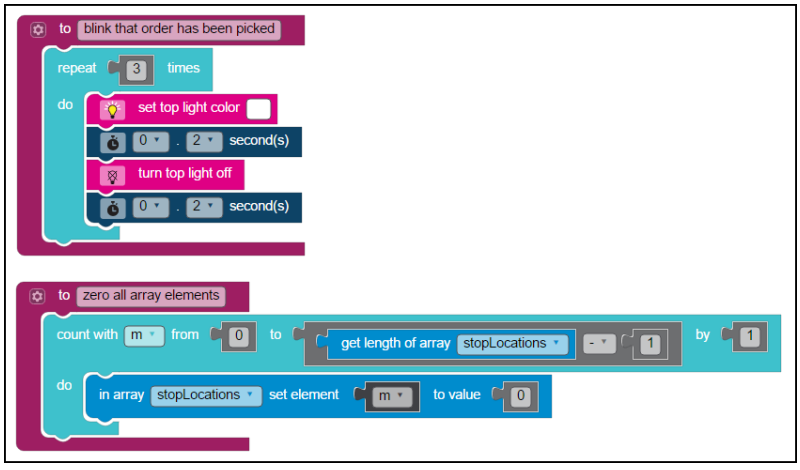


Figure 6

The blocks for the "blink that order has been picked" subroutine are quite straight forward—the LED blinks white three times, on for 0.2 seconds each time.

A couple of points should be mentioned with regard to the "zero all array elements" subroutine. You need to remember that indexing of array elements starts with [0], and therefore the last element in an array of length *n* has the index [*n* – 1]. This explains the "count with m" loop parameters. Secondly, in our simulation model, the maximum number of array elements that would be set to 1 while picking an order is 5. So there is really no need to reset *all* of the array elements back to 0. But the extra blocks required to set only specific array elements back to the value 0 would require an *if* block that *might*, in fact, consume more CPU cycles than those required to set all of the elements back to 0, especially considering the fact that our array has only 16 elements in total.

Robotic Warehouse Picking Experiment

Now that we have a thorough understanding of the OzoBlockly program and know that our goal is to determine the picking rate increase possible with the proposed layout, we can begin thinking about an experiment using Ozobot to solve this problem. In addition to Bit or Evo, the OzoMaps on the last two pages of this document and the data tables on pages 9 and 10, each student group will need a timer capable of measuring time to seconds—a stopwatch or cell phone timer would do just fine.

Each student lab group can be assigned a seed number to use in the program and then asked to run the OzoBlockly program once for each of the two layouts. They can collect and record their data on a copy of the data table on page 9 of this document. An example data table that is completely filled out appears on page 8. This should serve as an example for the teacher in the event that some students are not sure what to do. The shelf stop locations should be the same for the existing and proposed layouts. (As you know, those are controlled by using the same random seed for both layouts.) The time to pick all five orders is measured from the time that Ozobot starts moving until the LED flashes white after picking order number 5. The total number of picked items is obtained by counting the shelf location stops in the five orders. (Remember our assumption that each stop represents a single item being picked.) Dimensional analysis makes it easy to convert picks per minute to the more standard measure of picks per hour. The percent change is the ratio of the difference between the proposed an existing picking rates to the existing rate, multiplied by 100.

As the lab groups complete their data tables, they can be asked to record their final percent change in a copy of the summary data table, intended for class discussion. Based upon experiments done by the author of this document and shown in an example data table on page 8, it appears that most students will find ***picking rates increase approximately 25% with the proposed layout***. The author obtained similar results for both Evo and Bit, and the resulting percent change seemed to be about the same regardless of what seed number was used.

It should be mentioned that when doing simulations, ***conclusions should never be drawn based upon a single simulation run***. Multiple runs should be made and averages, or even fancier statistics such as standard deviations, should be computed. A single run may be an outlier, such as the 31% increase found by Bob and Roger, shown in the example summary table on page 8. Averaging multiple student group results is a nice way to minimize the effect of outliers on the final conclusions.

Data Table: Simulation of Robotic Warehouse Picking

Ozobot: Bit ☒ Evo ☐ Seed Number 57

Order #	EXISTING LAYOUT Shelf Stop Location Numbers						PROPOSED LAYOUT Shelf Stop Location Numbers					
1	1	2	5	11	13	14	1	2	5	11	13	14
2	4	6	12	14	15		4	6	12	14	15	
3	8	14					8	14				
4	1	3	7	9	12		1	3	7	9	12	
5	7	12	16				7	12	16			
Time to pick all 5 orders	4'23" = 263 seconds						3'33" = 213 seconds					
Total number of picked items	21						21					
Picking rate (picks per hour)	A $\frac{21 \text{ picks}}{263 \text{ s}} \times \frac{3600 \text{ s}}{1 \text{ hr}} = 287$ picks/hr						B $\frac{21 \text{ picks}}{213 \text{ s}} \times \frac{3600 \text{ s}}{1 \text{ hr}} = 355$ picks/hr					
% Change	(B - A)/A x 100%						(355 – 287) / 287 x 100% = 24% increase					

Simulation of Robotic Warehouse Picking
DATA SUMMARY
% Change in Picking Rate (Existing to Proposed)

Student/Group	% Change
Rex, Wally	26 %
Chuck, Ron	28 %
Sue, Amy	24 %
Bob, Roger	31 %
Mary, Kayla	26 %
Scott, Chelsea	26 %
Average→	26.8 %

Data Table: Simulation of Robotic Warehouse Picking

Ozobot: Bit ☐ Evo ☐ Seed Number _____

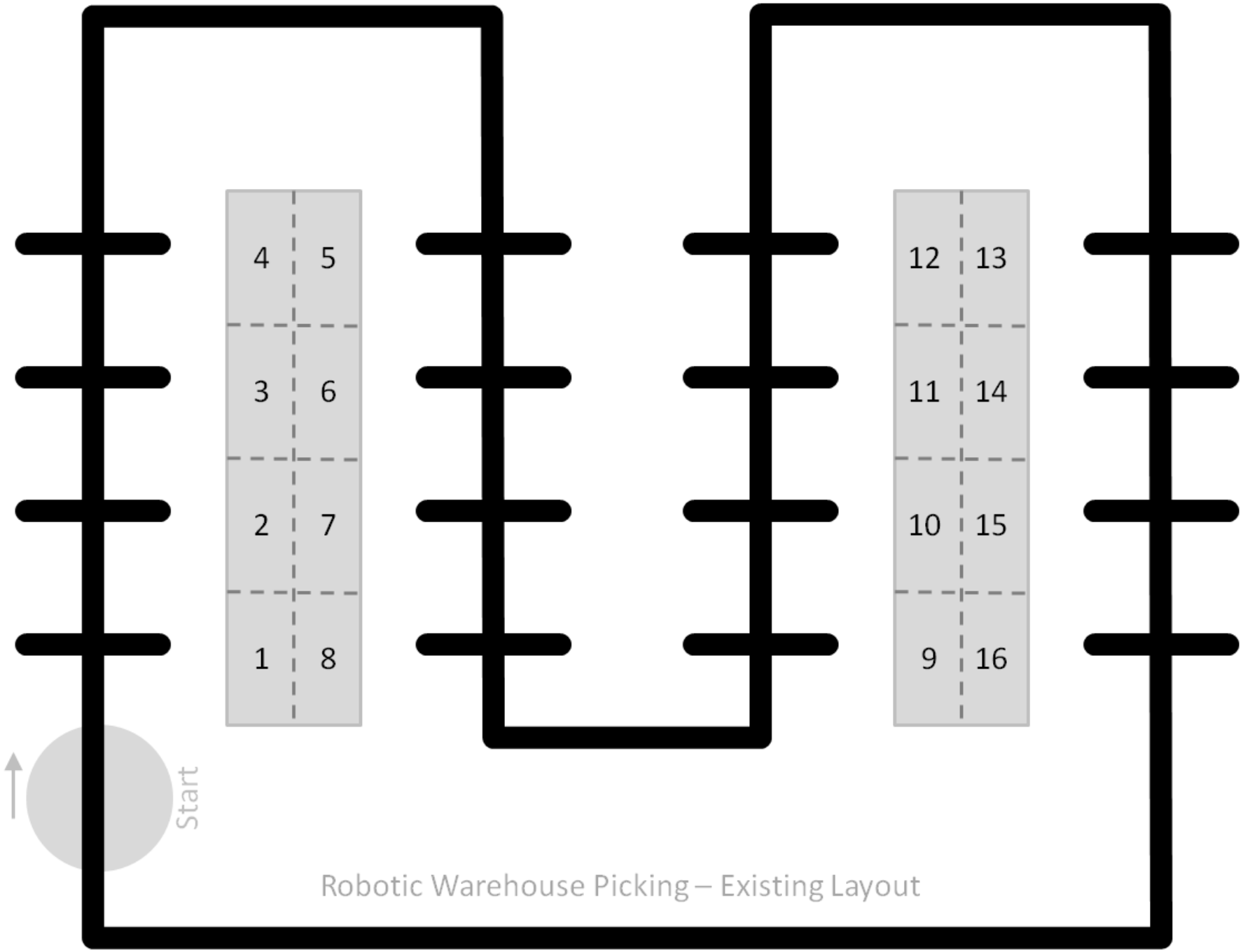
Order #	EXISTING LAYOUT Shelf Stop Location Numbers						PROPOSED LAYOUT Shelf Stop Location Numbers					
1												
2												
3												
4												
5												
Time to pick all 5 orders												
Total number of picked items												
Picking rate (picks per hour)	A						B					
% Change	(B - A)/A x 100%											

Simulation of Robotic Warehouse Picking

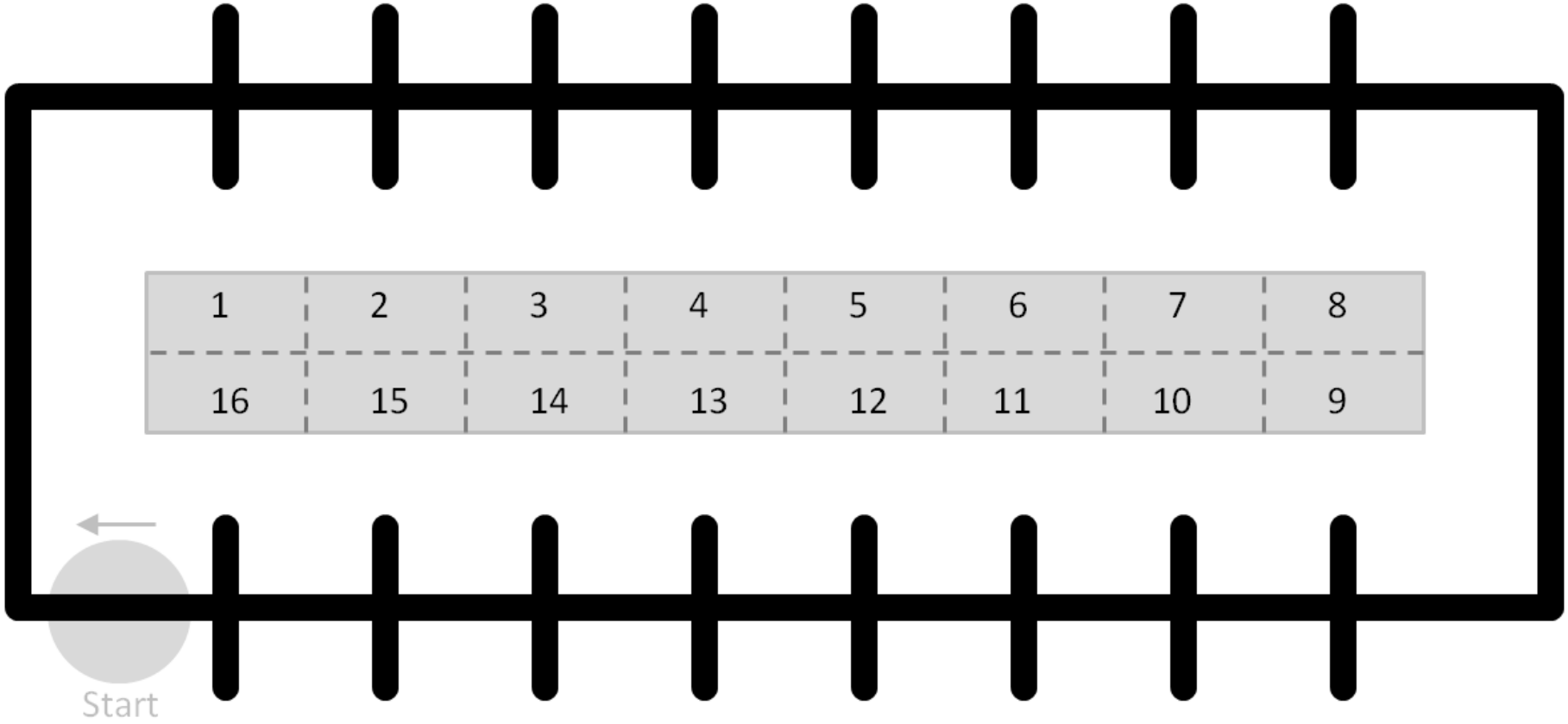
DATA SUMMARY

% Change in Picking Rate (Existing to Proposed)

Student/Group	% Change
Average→	



Robotic Warehouse Picking – Existing Layout



Robotic Warehouse Picking – Proposed Layout