ECE 470 Project Final Report

Ali Albazroun (aia)

Team Name: Book Worm

Codebase: https://github.com/Ali-7800/ECE-470-Final-Project

Video: https://youtu.be/NlzXUuGl1ps

1. Abstract

Book Worm is a system of robots and conveyor belts that sort books based on two criteria: color (which represents the topic of the book), and size (the length of the longest edge of the book). A pair of UR3 robots was used in this system to sort the books using inverse kinematics. A color sorting UR3 robot separates the books based on the RGB color data from a vision sensor. It can sort any set of colored books into stacks, with a maximum of 3 stacks (3 colors) with 3 books each. Then a size sorting UR3 robot the books in descending order where the largest book is placed on the bottom, based on data from two parallel opposite proximity sensors. This system can be adapted to sort any cuboid shaped object such as packages.

2. Introduction

Robots are used in a variety of different applications to automate tasks that would be to repetitive or dangerous for humans. One of these tasks is sorting, where a robot orders a set of objects in a particular fashion or groups them into different categories based on one or multiple criteria. The goal of this project was to combine both kinds of sorting (ordering and categorization) instead of just one like in [4] where objects were sorted by color and [3] where bricks were sorted by either color or length into bins. Hence, books were chosen as the sorted objects because they could be categorized by color and ordered by size. The color and size sorting algorithms were inspired by the Unsupervised Robot Sorting in [2] which relies on clustering similar objects together instead of predefined classes like in [5] or known objects like the shapes in [1]. Lastly, some of elements of this project were inspired by the \”Am I two?\”\n\”No, UR3.\” project, most notably the proximity sensor controlled conveyor belts and the tables for perception.

ECE 470 Project Final Report

3. Method

The robot system has four main components that interact with each other: the color sorting robot, the size sorting robot, the perception system, and the conveyor belt system. The following sections will summarize how each component works.

3.1. Conveyor belt system

The system is comprised of three belts, left, middle, and right as viewed from the top. Each of them have two modes: moving and stopped.

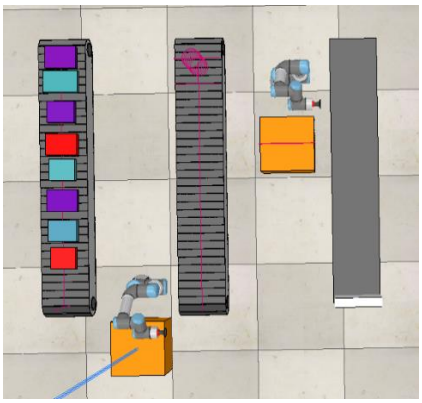

Figure 1- The left, middle, and right conveyor belts

3.1.1 left Conveyor

The goal of the left conveyor is to move each book to the proper position for the color sorting robot to pick up. This conveyor is equipped with two proximity sensors which act as the perception mechanism for it. The first proximity sensor runs along the entire conveyor belt and its purpose is to determine whether there are still books on the conveyor belt. The second proximity sensor is located at the end and its purpose is to determine whether the book reached the pickup position.

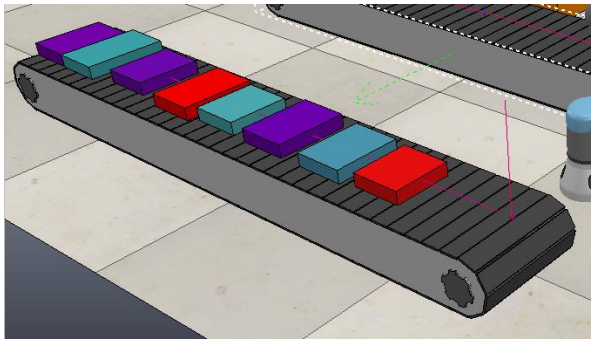

Figure 2- Sideways view of the left conveyor

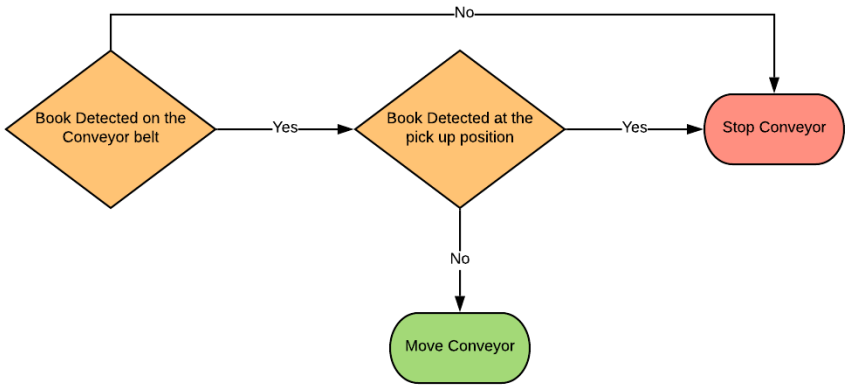

Figure 3- left conveyor state determination block diagram

3.1.2. Middle Conveyor

The middle conveyor has four proximity sensors, three of which are used to determine the state of the conveyor and a cylindrical one is used for the size sorting algorithm which will be discussed later in the perception section. This conveyor also communicates with the robots and the left conveyor to determine its state. The first proximity sensor (the one along the whole conveyor belt) serves a similar purpose to the first one in the left conveyor, it determines whether there are stacks of books on the conveyor or not. Then there are two proximity lying sideways at the end of the conveyor. One is used to determine the if a book stack reached the size sorter's pickup position and the other is to determine if there is more stacks to be sorted so the conveyor does not move them before the robot is ready to sort them.

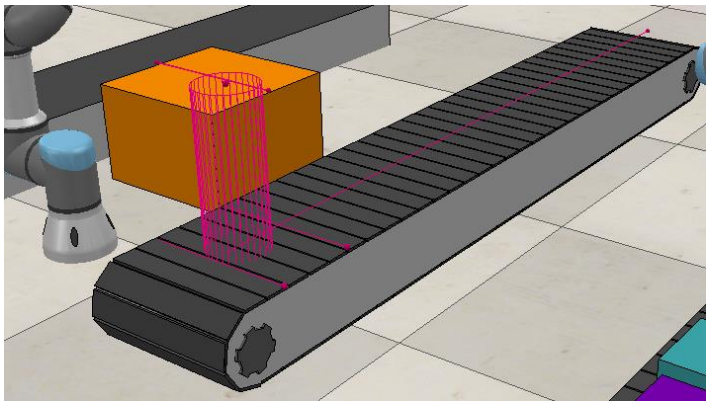

Figure 4- Sideways view of the middle conveyor

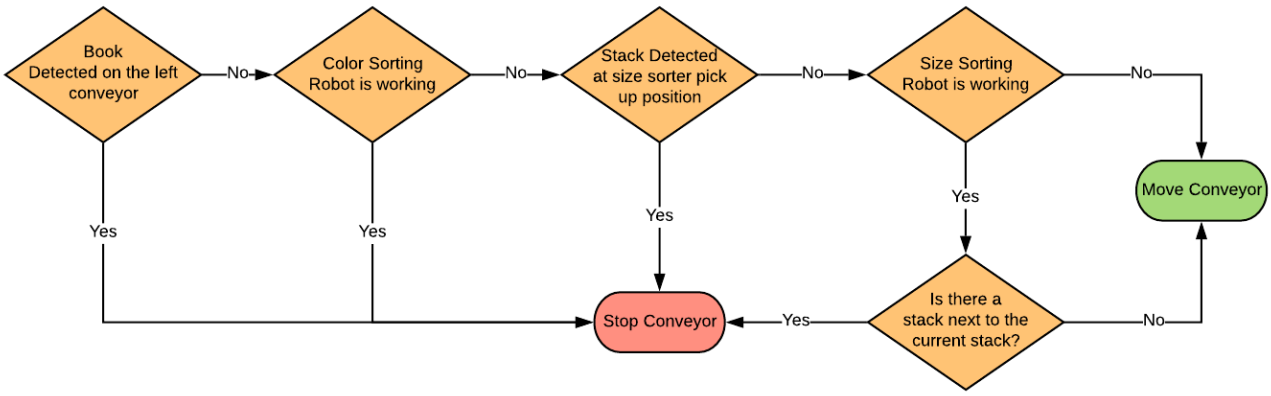

Figure 5-Middle conveyor state determination block diagram

3.1.3. Right Conveyor

This conveyor does not have any proximity sensors but primarily relies on the state of the size sorting robot. If the size sorting robot is working the conveyor stops moving, otherwise it moves. Notice it is a different kind of conveyor belt because of CoppeliaSim simulation limitations which prevent three conveyors of the same kind from working together properly. However, the conveyor now kind of resembles the top thicker part of an earth worm which is appropriate since the whole system is supposed to resemble one. Also, notice the wall at the end of the conveyor which stops the books from moving even if the conveyor is moving.

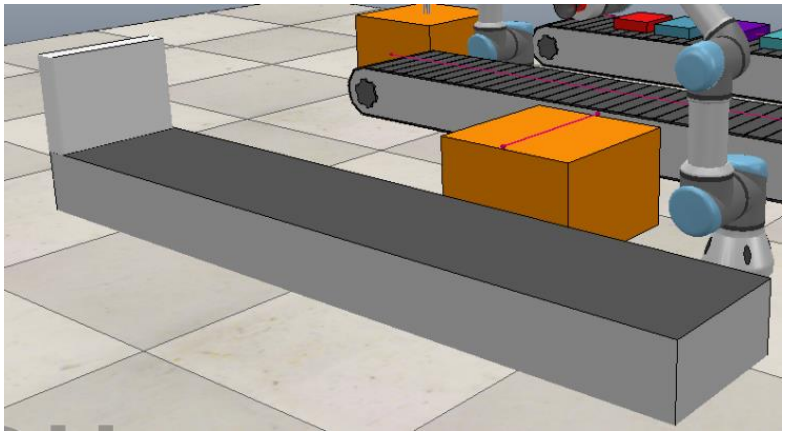

Figure 6- Sideways view of the right conveyor

3.2. Perception System

The perception system is also comprised of three elements, the two orange sorting tables, and the height cylindrical proximity sensor.

3.2.1. Color Sorting Table

This table is located next to the color sorting robot and is responsible of receiving the color data. It is equipped with a vision sensor that reports the RGB values of the books placed on the table using the *sim.simxGetVisionSensorImage* command.

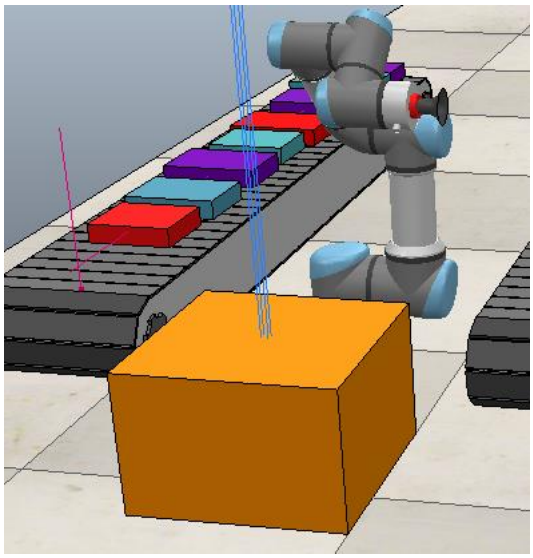

Figure 7- Color sorting table

3.2.2. Size Sorting Table

The size sorting table is equipped with two parallel opposite proximity sensors which measure how far is the book from the right and left edges of the table. This is done because the size robot might not pick up a book perfectly from the middle which causes it to be closer to one of the proximity sensors than the other, so if there were only one proximity sensor it could potentially report a smaller book to be bigger because it is closer to it. The book length is calculated using the following equation:

$$L_{book} = L_{table} - d_{right\ edge,book} - d_{left\ edge,book} \quad (1)$$

After it is calculated, the book length is sent and received using the *sim.simxsetFloatSignal* and *sim.simxGetFloatSignal* commands.

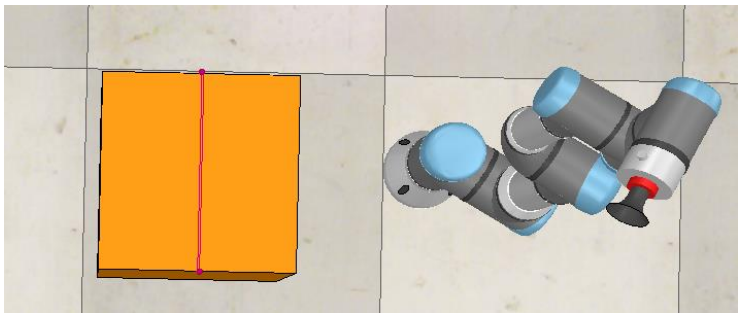

Figure 8-Top view of the size sorting table

3.2.3. Cylindrical Proximity Sensor

This sensor is used for two purposes related to the inverse kinematics calculations of the size sorting robot. First, to determine the height of the top book in the stack so that the size sorting robot can pick it up with knocking the other books in the stack. Second to determine the thickness of each book in the stack to calculate the appropriate height to place the book later during the restacking process. Each of the thickness of each book is calculated this way:

$$T_{previous\ book} = h_{previous\ book} - h_{current\ book} \quad (2)$$

and

$$T_{bottom\ book} = h_{bottom\ book} - h_{conveyor} \quad (3)$$

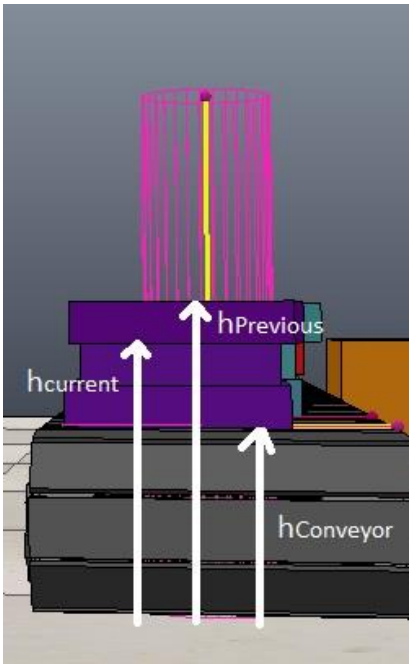

Figure 9- Cylindrical proximity sensors with book heights

Lastly, a cylindrical proximity sensor is used instead of a normal linear one because of the possibility that a book is small enough that it the linear proximity sensor misses it and measures the height of the book below it which would miss up all the later inverse kinematics calculations.

3.3. Sorting Robots

3.3.1. Joint Moving Function

At the beginning of this project I was using a joint moving function that was similar to the one used by the \"Am I two?\"\n\"No, UR3.\" project, which works pretty well at picking and placing individual books such as in the color sorting portion. However, when I started working on the size sorting portion I realized that this was not going to work. This is because the original joint moving function moved the joints simultaneously which caused the robot to take down the book stack while it is picking the top book. To fix this problem, I had to define a new joint moving function that took this into account. The new function I defined moved the books in a sequence of motions that would start by raising the second and the third joints of the UR3 robot moving the base of the robot and the sixth joints first, moving joints 3 through 5 next, then lastly lowering the second joint to the appropriate position. This is the code for the joint moving function:

```
def moveJoints(robot,angles):
    #set joints 2 and 3 to zero to stop the robot from knocking everything in its path
    sim.simxSetJointTargetPosition(clientID, robot[1][1], 0, sim.simx_opmode_oneshot)
    sim.simxSetJointTargetPosition(clientID, robot[1][2], 0, sim.simx_opmode_oneshot)

    #rotate joint 1 and joint 6 to appropriate position first
    sim.simxSetJointTargetPosition(clientID, robot[1][0], angles[0], sim.simx_opmode_oneshot)
    time.sleep(2)
    sim.simxSetJointTargetPosition(clientID, robot[1][5], angles[5], sim.simx_opmode_oneshot)
    time.sleep(0.5)

    #rotate joints 3 through 5
    for i in range(2,5):
        sim.simxSetJointTargetPosition(clientID, robot[1][i], angles[i], sim.simx_opmode_oneshot)
        time.sleep(0.5)
    time.sleep(0.7)
    #lastly rotate joint 2
    sim.simxSetJointTargetPosition(clientID, robot[1][1], angles[1], sim.simx_opmode_oneshot)
    time.sleep(0.5)
```

This new motion function is more natural since it moves the robot like how a human controller would move it. Although it made the motion slower, it fixed the problems caused by the old function.

3.3.2. Inverse Kinematics Function

Analytic inverse kinematics was used to determine the joint angles from the world coordinates of the scene. The equations used were the same equations used for lab 5. However, there were minor differences such as the equations for going from the world frame to the base frame which used position finding function to find the world coordinates of the robot, and the lengths of the end joints of the robot. This function will be referred to as $invk(x, y, z)$ for the rest of the document.

3.3.3. Color Sorting Robot

Unlike the \"Am I two?\"\n\"No, UR3.\" sorting algorithm, the color sorting algorithm used in this project was made to be as general as possible by clustering similar colors together in a similar fashion to [2]. This is done by first defining an empty list each sorting round, that adds a new color each time a new color of book is scanned by the color sorting table. Each time a new book is scanned by the vision sensor its RGB values are compared to the RGB values of every color in the color list, and if the new RGB values is within a range rng of the RGB values of one of the colors in the list, the new book is assigned that color, otherwise it is registered as a new color. The book is then placed on the middle conveyor using inverse kinematics based on its order of its color in the color list as follows:

$$stack\ position = invk(x_0, y_0 + k(color\ order - 1), z_0) \quad (4)$$

Where (x_0, y_0, z_0) is the position of where the first stack is placed, and k is a constant the represents the distance between the stack's centers. We can see that if the book has the same color as the first color in the color list it will be placed in the position $(x_0, y_0 + k(1-1), z_0) = (x_0, y_0, z_0)$, but it if it has a the same color as the third color in the list we get $(x_0, y_0 + k(3-1), z_0) = (x_0, y_0 + 2k, z_0)$.Which means it will be placed 2k units away from the first stack position in the y direction.

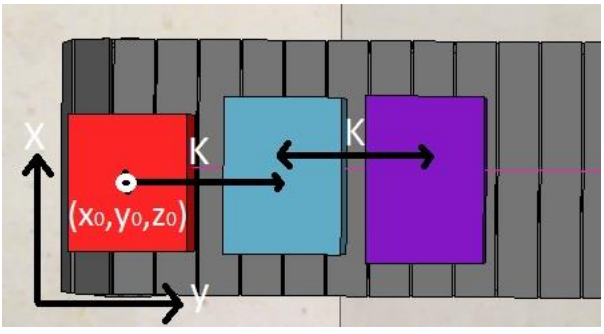

Figure 10- book positions depending on the order

This method has comes with some limitations, as the robot cannot keep moving k units away each time a new color is scanned forever due to physical limitations. Also, there is a set limit to how high can the stacks be.

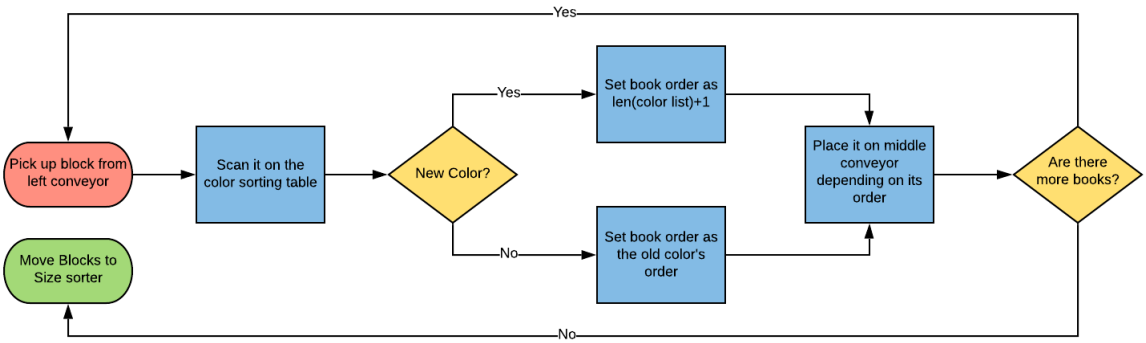

Figure 11- Color sorting routine block diagram

3.3.3. Size Sorting Robot

Similar to the color sorting algorithm, we start with an empty size list. The robot picks the top book using the height data from the cylindrical proximity sensor and the inverse kinematics function:

$$top\ book\ position = invk(x_1, y_1, h_{current}) \tag{5}$$

Where (x_1, y_1) are the coordinates of the size sorting pickup position.

After that it places the book on the size sorting table to get its size which is then added to the size list in a tuple along with its thickness and its order in the stack. After getting the book's size it is moved to the right conveyor using inverse kinematics and its order in the stack (top being first and bottom being last):

$$book\ position_{right\ conveyor} = invk(x_2, y_2 - k(stack\ order - 1), z_2) \tag{6}$$

Where (x_2, y_2, z_2) are the position where the first book is placed on the right conveyor. This is similar to how the books were placed previously on the middle conveyor using equation (4). When the last book in the stack is placed on the right conveyor this way, the size list is sorted by size using the python *sorted* function in descending order. Using this new list, the books are moved to the position of the first book in that list and stacked on top of each other using the thickness data

from the cylindrical sensor. This process is repeated as many times as the number of elements in the color list which equals the number of stacks.

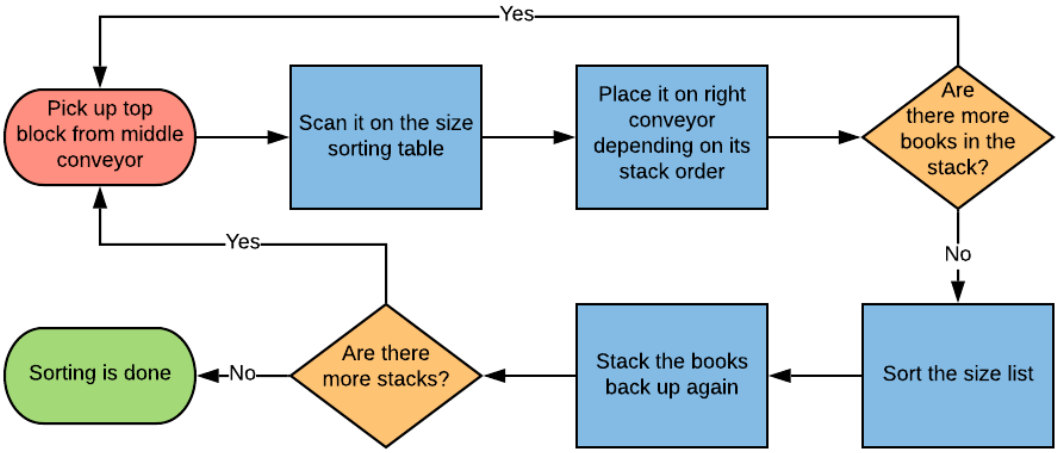

Figure 12- Size sorting robot routine block diagram

Similar to the color sorting robot it has some inherent limitations to physical constraints. Another limitation of this size sorting mechanism is that it assumes all books have the same width, where book in general are not like that. A more general algorithm would check both dimensions of the book and determine which is the length and which is the width and use the greater one for sorting.

ECE 470 Project Final Report

4. Experimental Setup

To test the sorting capabilities of the whole entire Book Worm system, I am going to run three experiments with multiple trials. Two experiments for the individual sorting systems and one for the system as a whole.

4.1. Color Sorting Robot Experiment

For this experiment I am going to run 15 trials of the following:

- Choose a set of three random colors using a color randomizer and change the book colors to those color in a random order, but the maximum number of books for any color has to be three since that's the range of the UR3's gripper.
- Run the Color sorting routine.
- Record success or failure and record the reason for failure (sorting error or stack fell down)

The goal of this experiment is to calculate an approximate success rate for the color sorting based on the number of successful trials.

4.2. Size Sorting Robot Experiment

For this experiment I am going to run trials of the following:

- Prepare 5 stacks of books (three books each) that vary in size ($0.15m \pm constant$).
- Run the size sorting routine twice while changing the order of the books in the stack.
- Record success or failure and record the reason for failure (sorting error or stack fell down)
- Plot the successes vs size variation on a log scale plot.
- Lower the variation in size for the next trial.
- Stop when you reach the smallest size variation in CoppeliaSim (0.00001m).

The goal of this experiment is to find what is the smallest variation between book sizes such that size sorting robot is still able to sort more than 50% of the books.

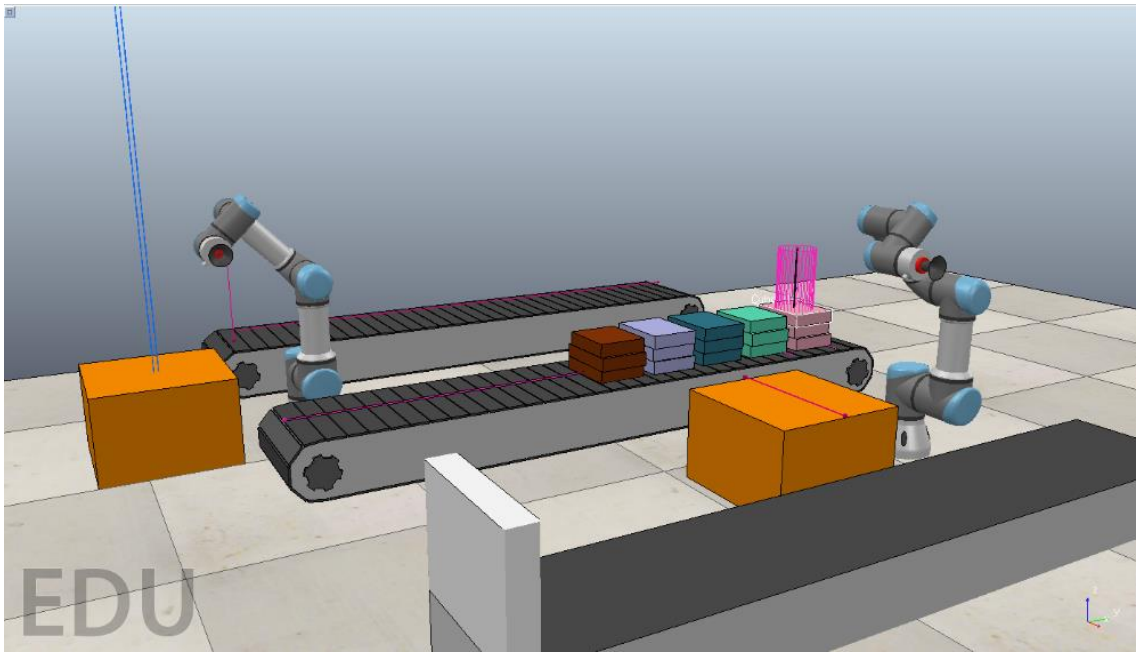

Figure 13- Size sorting experiment setup, notice how small variation in size are

4.3. Full Sorting Routine

For this experiment I am trying to simulate what would a typical sorting run for the whole Book Worm system would look like. To achieve this, I am going to run 15 trials of the following:

- Choose a set of three random colors using a color randomizer and change the book colors to those color in a random order, but the maximum number of books for any color has to be three since that's the range of the UR3's gripper.
- Scale the books with random scaling by CoppeliaSim.
- Run the whole sorting routine.
- Record the number of successfully sorted stacks and the number of unsuccessfully sorted stacks and record the reason for failure (sorting error or stack fell down) and when it failed (during the color sorting, during the size sorting, or neither.)

The goal is of this experiment is to calculate two approximate values, the success rate defined by the number of stacks sorted correctly over the total number of stacks sorted, and the success rate defined by the number of successful trials (all books we sorted correctly during the trial) over the total number of trials.

Lua code used to randomize colors and sizes:

```
    --Randomize Colors
    RGB1 = {math.random(),math.random(),math.random()}
    RGB2 = {math.random(),math.random(),math.random()}
    RGB3 = {math.random(),math.random(),math.random()}

    sim.setShapeColor(cuboid,nil,0,RGB1)
    sim.setShapeColor(cuboid0,nil,0,RGB3)
    sim.setShapeColor(cuboid1,nil,0,RGB2)
    sim.setShapeColor(cuboid2,nil,0,RGB2)
    sim.setShapeColor(cuboid3,nil,0,RGB1)
    sim.setShapeColor(cuboid4,nil,0,RGB3)
    sim.setShapeColor(cuboid5,nil,0,RGB2)
    sim.setShapeColor(cuboid6,nil,0,RGB3)

    --Randomize Sizes
    sim.scaleObject(cuboid,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid0,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid1,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid2,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid3,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid4,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid5,0.9+0.2*math.random(),1,1,0)
    sim.scaleObject(cuboid6,0.9+0.2*math.random(),1,1,0)
```

4.4. Notes

The first simulation after starting the CoppeliaSim scene will not be recorded because for unknown reasons the communication between CoppeliaSim and Python does not work properly during the first simulation causing the program to stop for no reason. Also, you can see that the code for the full color randomization assigns different colors in the same order, but those color assignments change each trial manually.

5. Data and Results

5.1. Color Sorting Robot Experiment

The 15 trials have been conducted and the results are outlined below:

Table 1-Color sorting experiment results

Variable	Success Rate	Number of Successful trials
Numerical Value	100%	15

The color sorting algorithm proved to be quite robust during the experiments where it was able to distinguish between closely colored books like the ones in figure 14 below. You can see from the table that it was able to successfully sort all the books in all the 15 trials.

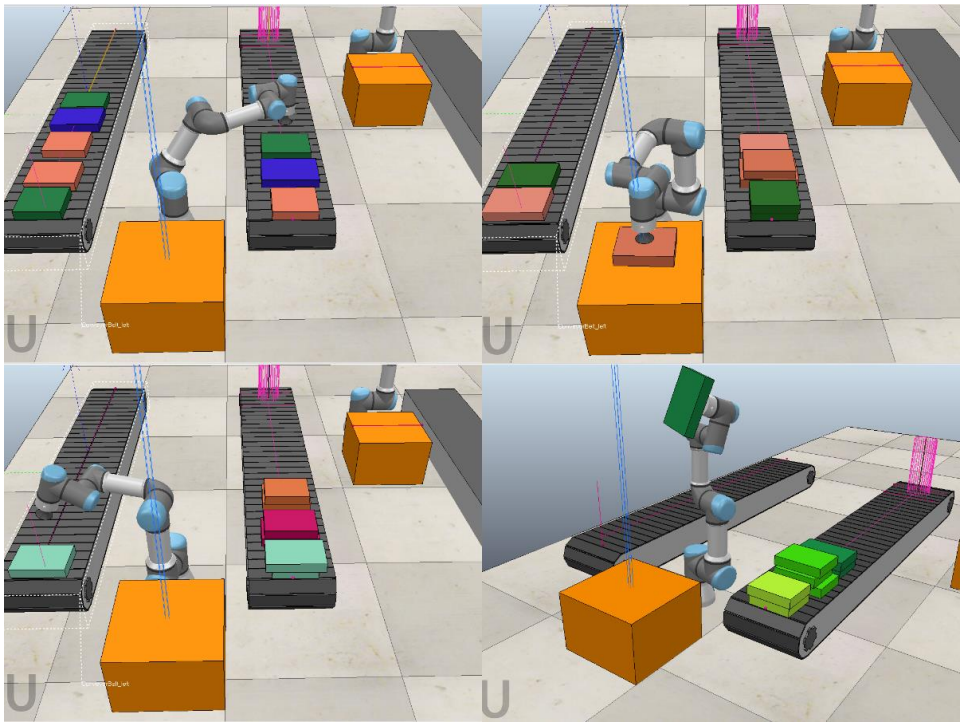

Figure 14- Color sorting experiment trials

5.2. Size Sorting Robot Experiment

6 trials were conducted, and the number of successfully and unsuccessfully sorted stacks vs the variation in book sizes in meter are plotted on a logarithmic scale in figure 15.

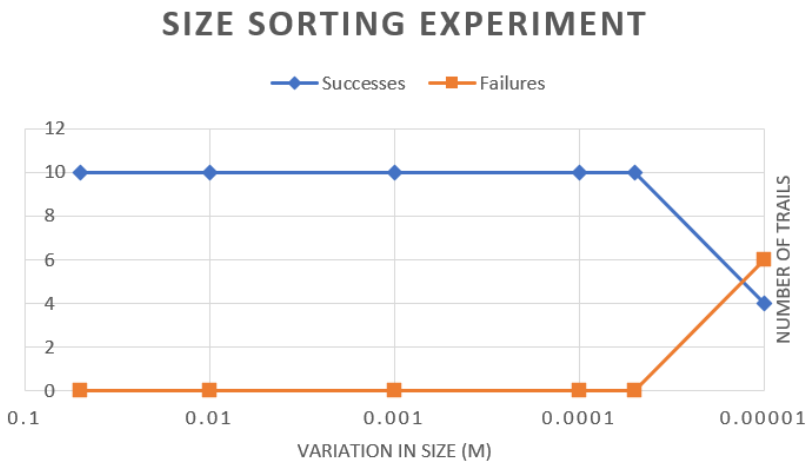

Figure 15-Successful and unsuccessful sorts plotted against the size variation of the trial

The size sorting algorithm also proved to be quite tough, it was successfully sorting all the book stacks until it reached the smallest size variation in CoppeliaSim were it was only able to sort 4 out of 10 stacks successfully. All the errors were sorting errors and not stacking errors. This is most likely because the two proximity sensors used to measure size are not precise to within $10^{-5}\ m$. For this project's purposes, the book sorting should be precise to within a centimeter.

5.3. Full Sorting Routine

15 trials of the whole color sorting routine were conducted, the data and results from those trials are outlined in the table below.

Table 2- Full sorting routine experiment results

Variable	Successfully sorted stacks	Successful trials	Success rate by stacks sorted	Success rate by trials
Value	44	14	97.78%	93.33%

Only a single stack in a single trial failed to be sorted during this experiment. The reason for failure was that the size robot sorting arm failed at stacking one of the books correctly which caused the book to fall of the conveyor belt. This makes sense because the results of the size sorting experiment would suggest that the robot would be able to successfully sort the books within the size variations used in this experiment (>> 10 microns), which it did in this particular trial. However, the size sorting experiment does not consider the errors in stacking caused by the color sorting robot but assumes perfectly stacked books. This in turn reduces the success rate of the size sorting robot by causing failures in stacking.

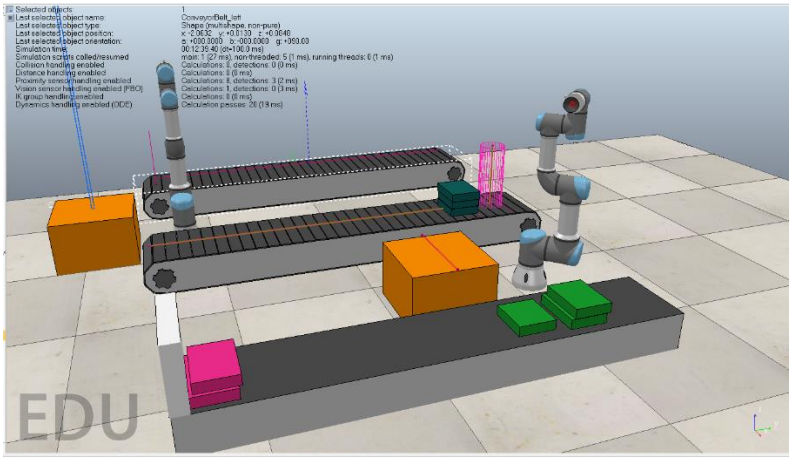

Figure 16- Books stack reaching the end the full sorting routine experiment

ECE 470 Project Final Report

6. Conclusion and Recommendations

The Book Worm robot system was able to successfully sort books by color and size at an outstanding rate of success. It was interesting to see how simple algorithms for color and size sorting could successfully sort a wide variety of book colors and sizes without the need for prior knowledge on those colors and sizes.

Some recommendations on how to increase the viability of this robot system:

- Using robots with a workspace that is greater than the UR3's workspace, this will allow the system to reach farther along the conveyor belt (more colors could be sorted) and higher normal to them (more sizes could be sorted).
- Introducing some cameras on top of the pickup positions, this would allow for the implementation of some computer vision algorithms to find the book's center position and in its orientation. This in turn could be used to improve the accuracy of the inverse kinematics and eliminate stacking errors.
- Installing a second pair of proximity sensor orthogonal to the original pair on the size sorting table, this would allow the robot to check both of the book's dimensions and sort using the greatest of the two.
- Investigate joint moving functions that vary with the angle inputs using conditional statements instead of having a static function that does the same motions for any angle input. This could reduce the number of unnecessary motions especially during the size sorting routine where the robot arm would go up and down with book unnecessarily which made the size sorting routine too slow.
- Add a second cylindrical proximity sensor at the color sorting pickup position, this would allow for height data to be collected which would improve the accuracy of the color sorting robot's inverse kinematics.

Overall, I had a lot of fun working on this project and seeing how far I can push it. If I had more time I would have definitely implemented some of the recommendations listed above, but I am still satisfied by the end the result and all the positive feedback I got on it.

7. References

[1] Garad, Vinayak. "Others: Object Sorting Robot Based on Shape - Published by Priya Vinayak Garad in IJARIIT Journal." *IJARIIT*, IJARIIT, 1 Jan. 1970, www.ijariit.com/manuscript/object-sorting-robot-based-on-shape/.

[2] Guerin, Joris, et al. "Unsupervised Robotic Sorting : Towards Autonomous Decision Making Robots." *International Journal of Artificial Intelligence & Applications*, vol. 9, no. 2, 2018, pp. 81–98., doi:10.5121/ijaia.2018.9207.

[3] Gupta, Megha, and Gaurav S. Sukhatme. "Using Manipulation Primitives for Brick Sorting in Clutter." *2012 IEEE International Conference on Robotics and Automation*, 2012, doi:10.1109/icra.2012.6224787.

[4] Meng, Ong Kok, et al. "Robotic Arm System with Computer Vision for Colour Object Sorting." *International Journal of Engineering & Technology*, vol. 7, no. 4.27, 2018, p. 50., doi:10.14419/ijet.v7i4.27.22479.

[5] Mezei, Ady-Daniel, et al. "Sorting Objects from a Conveyor Belt Using POMDPs with Multiple-Object Observations and Information-Gain Rewards." *Sensors*, vol. 20, no. 9, 2020, p. 2481., doi:10.3390/s20092481.