Comau 2020 Multidisciplinary Design Project

Report By: Travis Gurlik (tgurlik)

Other Team Members:
Joey Berman, Shannon Lau, Hannah Moon, Arjun Raman, & Jonathan Wong

Faculty Mentor: Dr. Vineet Kamat
Sponsor Mentor: Joshua Graff (Comau Robotics Engineer)

1. Introduction

This project was an industry sponsored project managed by the Multidisciplinary Design Program (MDP) for the duration of 2020. The sponsor for this project was Comau, an industrial automation and robotics company. Comau is constantly seeking to push technology forward and use it to improve user experiences in a variety of fields. For this project, Comau requested assistance creating an automated system for packing luggage into luggage transportation vehicles at commercial airports.

A. Project Background

One of the slowest processes when preparing a commercial flight is loading the luggage into the airplane. To complete this process, luggage is collected at the check-in counter and stored in the terminal until the plane arrives, at which point it is loaded into luggage cars by hand, transported to the airplane, and loaded into the airplane by hand. This process must be done in reverse to unload an airplane and deliver the luggage to the pickup location. This process requires multiple workers for each step, and the large number of concurrent flights at a typical airport means these workers often have a lot of work to do. This work of constantly lifting heavy bags puts a strain on the workers' bodies and can lead to damage over time. Additionally, any delays during this process may delay flights -- requiring re-planning by air traffic control -- or force customers to wait longer for their bags.

To alleviate these issues, Comau seeks to automate the loading portion of this process. Their solution to these problems is to use a robotic system that will automatically pack incoming luggage into a container. Comau has plenty of experience designing and building industrial robots, but sought an outside perspective on the problem to see if a team unconstrained by Comau's previous work could produce a better, more novel solution. Thus, they approached MDP about sponsoring a student project.

B. Project Overview

Our objective as the team chosen for this project was to design and create a hardware and software system that could be connected to a robotic system provided by Comau to assist with item identification and placement. The robotic system would be created by Comau separately from our system. This means the robotic system would not be controlled by our system; our system would simply tell the robotic system where to place the items. To do this, our system needed to be able to measure the incoming bag, calculate the optimal location for the bag to be placed within the container such that as much space in the container is used as possible, and provide an interface through which a human operator could monitor and interact with the system.

Our system had the following goals and restrictions:

- When an object is placed in the staging area, our system should send the optimal pose (position and orientation) for that object to be placed at within the container to the robotic system, which will handle actually placing the object
- Can only use depth cameras (cameras that are capable of measuring the distance from the camera to any point within the camera's field of view) for sensing
- Should be able to pack a container such that it is at least 85% full before no more objects can be placed in it
- Must be able to recognize when it cannot place any more objects within the container
- Must include a Graphical User Interface (GUI) through which all interaction with the system is performed
- Cycle time (total time between an item being placed in the staging area, the item being placed within the container, and the system being ready for the next item) must be less than ten seconds

Our project thus sought to address four questions:

1. How can we determine the dimensions of an incoming object using only depth cameras?
2. How do we determine the most optimal location within the bin for a given item?
3. How can we display this information and the system status in an intuitive way?
4. How can we connect all of these component programs together into a cohesive system?

2. Methodology

We started by researching some key components of the project: the depth cameras we would be using and how we could apply them to our system, how we could measure the incoming objects with the data these cameras could provide, and how to solve the bin packing problem. We then took our findings and adapted them for our specific problem.

Once we had finished our preliminary research, we split into subteams to handle the various aspects of the project: hardware and segmentation, bin packing, and graphical user interface (GUI). The majority of my work was done with the GUI, but I also assisted with the other components at various points in the project. While my knowledge of these other aspects of the project is incomplete, I know enough about them to provide a general overview of our methods for each.

A. Hardware and Segmentation

The main hardware component of our system is a pair of depth cameras. Depth cameras are special cameras that are able to record the visual data that normal cameras can (the colors of each pixel within the camera's field of view, referred to as the RGB data stream), the distance from the camera to various points within the camera's view (referred to as the depth stream), and the rotation of the camera (referred to as the gyro stream). While solutions that made use of the RGB stream were considered near the start of the project, we ultimately ended up using only the depth and gyro streams.

The depth stream contains distance data in the form of a point cloud; a series of points within three-dimensional space. These point clouds are generated automatically by the cameras, and the points are located relative to the camera's position.

I. Two-Camera Setup

One issue we ran into early into the project was figuring out how to get a point cloud of the entire object; a single camera can generate a point cloud of the object from a certain perspective, but that leaves the cloud with multiple blind spots. To obtain a more complete point cloud, it was decided to use two separate depth cameras and combine their point clouds.

This process required two major problems to be solved. First, we needed to figure out how to physically connect both cameras to a single system. While it would seem like it should be possible to just plug both cameras into the system, it quickly became apparent that some additional hardware parts and modifications were needed to get both depth streams working. Once we got these parts resolved this issue, we then had to figure out how to actually combine the two point clouds. This was achieved by transforming the point cloud from the second camera so it lines up with the first camera's point cloud. This transformation is currently found and entered manually by the user.

II. Segmentation

Once the point cloud is obtained, the object can be measured through a process known as segmentation. First, the point cloud is filtered to only include points within a certain range (specified by the user). This removes most of the background points from the cloud. Next, the object must be isolated from its immediate surroundings. The point cloud should include only the object and the surface it is sitting on at this point, so using the height of the cameras (and potentially some plane detection), the surface can be removed to isolate the object. Finally, we create a minimum oriented bounding box (MOBB) to obtain the dimensions.

The MOBB has three main features. Most obviously, it is a bounding box; this means that it is a rectangular prism (or box) that contains every point within the object's point cloud. It is also oriented, which means that it is rotated to match the rotation of the object instead of being aligned with the camera. Last, and most importantly for this algorithm, it is minimal; it is the smallest box with the given orientation that holds all of the points. Once we have the MOBB, we can easily calculate the length of each of its edges to obtain the dimensions of the object. These dimensions are then passed to our bin packing algorithm.

B. Bin Packing

The particular bin packing problem presented to us in this project is known as an online bin packing problem. In this case, "online" does not mean the system is connected to the internet; rather, it means that each item enters the system one at a time. This is different from typical packing problems, as not having access to all of the items from the start means we cannot determine a truly optimal packing for the container. Instead, we're using a heuristic, which is an algorithm that approximates the solution to a problem.

I. Online Bin Packing Heuristic

The specific algorithm we implemented is called the Online Bin Packing Heuristic. This algorithm uses Empty Maximal Spaces (or EMS's) to track where items can be placed within the container. These EMS's are constructed around each object when they are placed and encompass all of the empty space around that object. These EMS's effectively track where a new item could be placed (via their bottom-back-left corners) and the largest object that could fit in the available space.

As mentioned, new EMS's are created around each object as it is placed. However, the previously existing EMS's must be updated as well. At minimum, any EMS's that intersect the newly placed object must be shortened so they only contain empty space. Depending on where the preexisting EMS's were located, the new EMS's may end up overlapping the old ones, even after the old ones are shortened. We refer to this as the overlap version of the packing

algorithm. Alternatively, the non-overlap version of the algorithm can be used. This version updates the preexisting EMS's such that there is no overlap between them and the new EMS's. This can potentially be more efficient as it allows for adjacent EMS's to be combined into a single EMS in certain circumstances, removing some redundancy and leaving fewer EMS's to iterate through. However, this version of the algorithm was far more difficult to implement as it required allowing objects to be only partially contained within an EMS. In addition, initial tests of this version of the algorithm did not show any notable improvement over the overlap version. For these reasons, we decided to use the overlap version for our final solution.

II. Our Algorithm

Using the Online Bin Packing Heuristic as our base, we created our own bin packing algorithm to suit our project. In our algorithm, EMS's are sorted by their bottom-back-left corners, prioritizing first lower EMS's, then EMS's closer to the back of the container, and finally EMS's closer to the left side of the container. Whenever a new object is detected, the algorithm iterates through all of the EMS's in order and finds the first one the object fits in (and which rotations are required to fit it). The bottom-back-left corner of this EMS is then sent to the robotic system as the target location for the object and the EMS's are updated as needed.

To improve the percentage of the container our algorithm was able to fill, we made a few additional improvements to this basic algorithm. First, since our algorithm needed to place items in locations that they would be able to physically stay in without falling, our initial algorithm constructed the EMS's above new objects to only be directly above the object. However, we quickly found that this was too limiting as it meant the objects had to continually get smaller to be able to place them on top of the earlier objects. To fix this, we added the option to have EMS's hang over the edges of the object. We needed to be careful to ensure the overhang wasn't too large, as our algorithm placed objects at the corners of EMS's instead of in the center of them, meaning allowing too much overhang would cause the next object to not be sufficiently supported.

The second main improvement we made was to allow for vertical rotations in addition to horizontal ones. While the robotic system was not originally intended to be able to rotate items vertically, this change was approved by our sponsor and resulted in a significant increase in our fill percentage.

The third and final major improvement we made to our bin packing algorithm was adding the ability to consider multiple items at once. This was another change from the original system design that our sponsor proposed to help us improve our fill percentage, and it was similarly effective. This change meant that instead of only being able to see the next incoming item, our system was able to view and place any of the next three incoming items. We updated our algorithm to find the best placement for all three items, then actually place whichever one had the most optimal placement (determined by location and required rotations).

C. Graphical User Interface

Our graphical user interface (GUI) is the front-end program that allows the user to interact with our system. While the original plan was to have everything in a single web-based interface, we were unable to get it working properly and thus had to switch to a different structure. Our final GUI consists of two programs: one for visualizing the results of the bin packing algorithm and one for setting parameters and viewing system data.

I. Bin Packing Visualizer

The bin packing visualizer is used to visualize where the bin packing algorithm has placed each item within the container. The visualizer written in Processing, an environment for creating graphical programs in Java. While the original version of this program was intended to be a basic visualizer we could use for testing while the main GUI was still being developed, it was eventually developed into a fully functional program that we could use with our final product.

To communicate with the packing algorithm, the visualizer monitors a particular input file, which the packing algorithm prints its data to. Whenever this file is modified, the visualizer rereads the data and updates the visualization accordingly. The visualization is a three-dimensional view of the container and the objects placed inside, each of which is represented by a differently colored box. The camera can be controlled by moving the moue around the window (though there is a keybind to lock the camera in place), and the arrow keys can be used to view the container from particular directions. For testing purposes, the program is also able to monitor a second input file to allow for easy comparison between two datasets, such as items and EMS's.

II. Data & Parameters Interface

The data and parameters interface (DPI) handles the setting of system parameters (currently only the size of the container) and displays data as it flows through the system, including the dimensions output by the segmentation algorithm and the placement outputted by the packing algorithm. It was originally designed as a web application (using HTML, CSS, and JavaScript) as we decided that would be the easiest way to make a clean and clear interface. However, numerous issues arose during development regarding communicating with the segmentation and packing programs that we were unable to resolve before the end of the project. So we would at least have something functional to use with the system, a terminal-based version of the interface was created instead. This version includes all the functionality the web version was intended to have, but is a bit harder to understand at a glance.

D. Communicating Between Programs

As we were writing multiple separate programs for this system, we needed some way for these separate processes to communicate with one another. As noted above, the bin packing algorithm and the bin packing visualizer communicate by way of a separate file that one program prints data to and the other reads from. While this is sufficient for these purposes, it is a fairly slow communication method and cannot be used between programs on different computer systems. As our system was designed to include two computers -- one to interface with the depth cameras and perform segmentation and one to handle the actual packing -- we needed to find another way for these programs to communicate.

To solve this problem, we used the TCP/IP communication protocol. In short, this protocol allows two systems to communicate with one another via a socket. This socket is opened on a particular port of a web address or network, which both programs are connected to. To create this socket, one program acts as the server and opens the port for communication. Other programs that would like to communicate to the server can then connect to the socket as clients, opening two-way communication between them and the server. Using this protocol, we were able to connect each of our programs together into a single system.

3. Results

Ultimately, we were able to create a fully integrated and fully functional system that solves the problem we were presented with.

For segmentation, we successfully met our goal of calculating the dimensions of incoming objects within a 2.5% margin of error, getting within half a centimeter on many of our test cases.

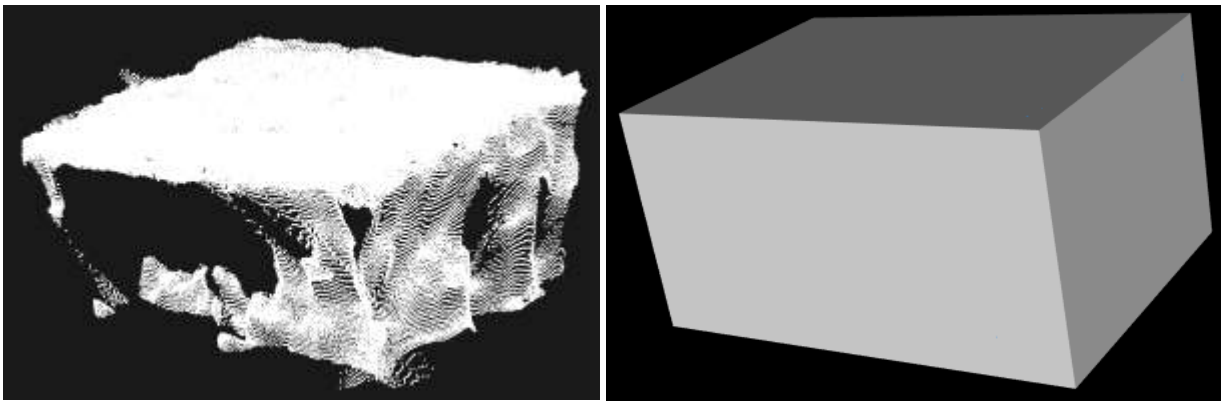

Figure 1: A combined point cloud of a sample object and the generated bounding box.

The packing algorithm was able to quickly and realistically place objects within the container, but ultimately fell short of our goal of filling 85% of the container, with our best result being a little over 75%. Our GUI was able to accurately visualize the output of the bin packing algorithm and proved useful for assisting with debugging and monitoring system status.

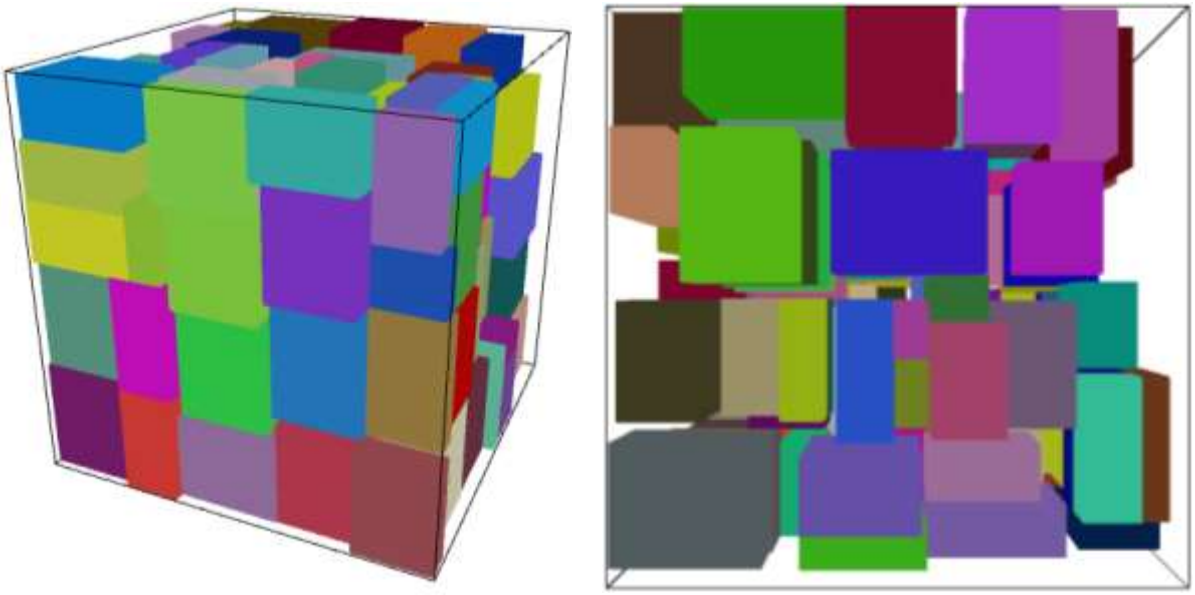

Figure 2: Representations of the output from our bin packing algorithm for our best test case created by our visualizer. The view on the right is from the top of the container.

Our system was able to successfully identify, measure, and place items in a lab setting in under 4 seconds, meeting our time efficiency goal.

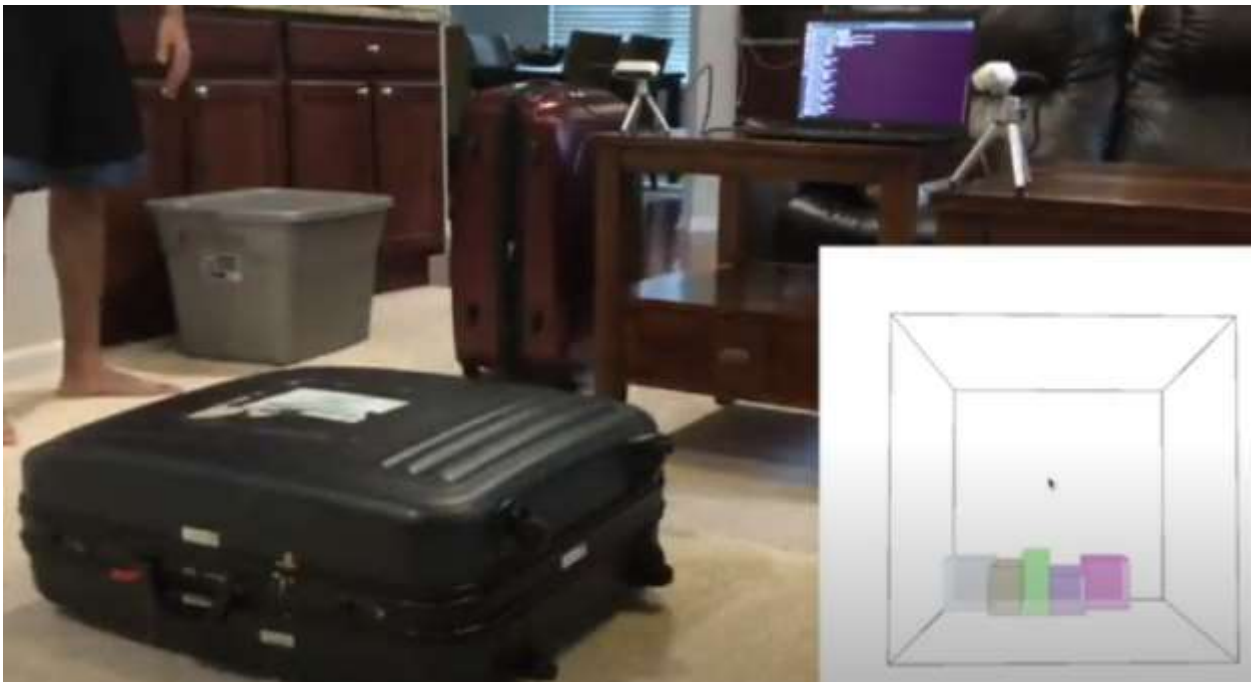

Figure 3: A view of our system running in a lab setting. Objects are placed in the staging area on the left, while the visualizer in the lower right shows placed items within the virtual container.

4. Discussion

In the end, we were able to answer each of the questions that defined our project. Each of these questions helped us learn about new techniques and technology we would not have had an opportunity to learn otherwise.

A. Determining the Dimensions of Objects Using Only Depth Cameras

Our first objective was to identify and measure incoming objects. While there are many ways this can be accomplished, we were tasked with finding a way to solve this problem using only depth cameras.

To solve this problem, we combined the point clouds from two depth cameras, filtered this merged point cloud so it only included the points representing the object, and used the minimal oriented bounding box algorithm to create a rectangular prism representing the object, which we could easily measure.

In developing this solution, we learned about how depth cameras work, including how to read and use each of their data streams. We also learned how to manipulate point clouds through both transformations and filtering and how to use these manipulations to combine two separate point clouds of the same space. Finally, we learned how to create an oriented bounding box for the points in a point cloud to measure the dimensions of an object.

B. Identifying the Optimal Placement for an Object within a Container

Our second objective was to determine the best way to pack a series of items into a container. While bin packing problems are relatively common, online bin packing problems, where the system receives only one item at a time, are less common. The differing properties of the items being packed (some are easily compressed, some are solid, and some have some combination of these) also posed a unique challenge.

Our solution to this problem was to adapt the online bin packing heuristic to better fit our needs. We added the necessary physical constraints, implemented the ability to rotate objects in any direction, provided our own method of comparing various potential placements, and allowed for up to three objects to be considered at once to increase packing efficiency. However, due to the complexity of the problem we were unable to account for the unique properties of the items being packed in the time we had.

Through this process, we learned about various strategies for solving the bin packing problem and how to implement one of those solutions; namely, the online bin packing heuristic. We learned how to modify this algorithm to fit our needs and learned about some ways to optimize how much of the container the algorithm can fill.

C. Creating an Intuitive User Interface

Our third objective was to create a user interface through which a human user could monitor the system and provide any necessary input. This interface needed to be easy for a user to understand and use.

While various complications caused us to change our solution for this problem multiple times, our final solution consists of a Processing-based bin packing visualization program and a terminal-based interface for setting parameters and monitoring system data.

In developing the user interface, we learned a little bit about a lot of things, including rendering in WebGL, using the filesystem in Java, and development with NodeJS and Express.

D. Communicating Between Processes

As our system would consist of multiple processes running side-by-side, we needed to find some way for these processes to communicate with one another reliably and efficiently.

To solve this problem, we utilized the TCP/IP communication protocol, having some programs act as servers while the others were set up as clients. With the exception of the bin packing visualization, all sending and receiving of data between programs was done with this protocol.

This taught us a good deal about what the TCP/IP protocol is, how it work, and how to implement TCP/IP clients and servers in various languages, including Java and C++.

5. Conclusion

Ultimately, I believe this project was a success. While there are certainly aspects of our system that could be further improved, we were able to create a fully functional system that solves the problem presented to us. This is shown in Section 3; we were able to meet, and in many cases, exceed, our goal for accuracy when measuring objects, made great progress on optimizing our bin packing algorithm, and created a fully functional GUI for easy at-a-glance system monitoring.

A. Places for Improvement and Next Steps

While we were able to make excellent progress on this project, there are a few parts of our solution that could be improved should another group pick up the project.

First, despite making continual progress on our bin packing algorithm, we ultimately fell short of our goal of filling 85% of the container. Further improvements to our algorithm could likely be made to cut down on the amount of empty space left between placed objects, particularly when placing objects on top of one another.

Second, our system currently uses a very simple model for incoming objects: it assumes that each object is a rigid rectangular prism. Additional work could be done to allow for more realistic modeling of objects, including allowing for more deviant shapes and accounting for varying rigidities. This flexibility would also help us improve our bin packing algorithm, making the algorithm's placements more accurate to the real container.

Third, our user interface fell short of our initial vision of having a single web-based program. Creating such a program would require implementing the proper rendering in WebGL and fixing the communication issues that prevented us from using a web-based data and parameters interface. Combining the two aspects of the GUI into a single program would make the GUI much easier to use and, for the data and parameters aspect specifically, much easier to understand at a glance.

Finally, due to the pandemic we were unable to test our system with an actual robotic system. As soon as it is safe to do so, our system would need to be tested in an actual work setting. This would bring to light any changes that would need to be made to our system for it to work in its intended environment.