Optimization of Vehicle Route using Fulfillment Center Simulation

Adam Carrera, Sivakrish Sivarajah

***Abstract*—We seek to optimize a model of an unmanned fulfillment center. The fulfillment center contains three intakes, and three docks. An algorithm was developed, for one robot, to determine the next optimal destination for minimizing travel cost, therefore maximizing the number of packages transferred, and the number of packages docked. A python script is developed to implement the optimization algorithm and tested with a simulation model of an unmanned fulfillment center.**

I. Background

Our Project is based off of the Kiva Robots employed by Amazon for resource allocation purposes. Data is collected at various stations where items are located which need to be brought to different areas for use. The robots themselves also transmit data on their position as well, in the cloud the information is compiled and the robots are deployed to specific locations where they can move the most amount of items, in the least amount of time.

II. Project Specifics

For our specific project there are 3 intake locations and 3 docks. Packages are continually added to all intake locations on a regular basis. These packages are automatically assigned to be delivered to specific docks. If a robot enters an intake location and has available capacity, the packages are automatically loaded onto the robot. Similarly, if a robot enters a dock, the packages that are slated to be delivered to that dock are unloaded automatically. Robots can be provided with a coordinate pair as a destination and can also be provided with a speed value.

When the robot is at a location it should ideally go to the closest intake and then attend the other location before dropping off the items to the corresponding docks based on which one is closest. However, the robot is limited on the amount of packages it can carry at a time, which for this problem is 50 items, additionally the intakes restock at a certain time interval. Due to this we believed the best approach, in getting the most amount of packages transferred, had to do with the decision making of the robot on where its next destination should be. We needed for the robot to decide on where it should go next whether it be an intake or a dock and which one of those locations it needed to be.

III. Formulating Equation

We thought the best way for the warehouse robot to make this decision was by considering the information it had relating to the following information:

- The number of packages the robot was carrying
- The number of available packages at each intake
- The distance between the current location and each of the other intake and dock locations

Using this data we identified the constraints and boundary conditions required and formulated an equation that we believe will produce the optimal next location for the robot to travel to:

$$\begin{aligned} \max_{x} \quad & f(X) = \frac{P_a}{D_a}x_a + \frac{P_b}{D_b}x_b + \frac{P_c}{D_c}x_c + \frac{P_A}{D_A}x_A + \frac{P_B}{D_B}x_B + \frac{P_C}{D_C}x_C \\ \text{subject to} \quad & P_a + P_b + P_c + P_A + P_B + P_C \leq 50 \\ & x_a + x_b + x_c + x_A + x_B + x_C = 1 \\ & x_a, x_b, x_c, x_A, x_B, x_C \in Z \end{aligned} \tag{1}$$

In this formulated problem, lowercase letters represent docks while uppercase letters indicate intakes. Table I provides information of the variables used in the equations.

We attained the objective function by thinking that the ideal next location should be determined by whichever location has the most amount of packages (either to be picked up or delivered) but we need to travel the least to get to. By maximizing $\frac{P}{D}x$ we ensured that we will pick the location where the ratio of P to D is greatest. the constraint $P_a + P_b + P_c + P_A + P_B + P_C \leq 50$ is used to tell the capacity of the warehouse robot and what it is allowed to carry. This is important in the decision making process for the robot as there is no need to travel to an intake if the robot is already full. The constraint $x_a + x_b + x_c + x_A + x_B + x_C = 1$ & $x_a, x_b, x_c, x_A, x_B, x_C \in Z$ are used to essentially say only 1 location is allowed to be picked for the next stop.

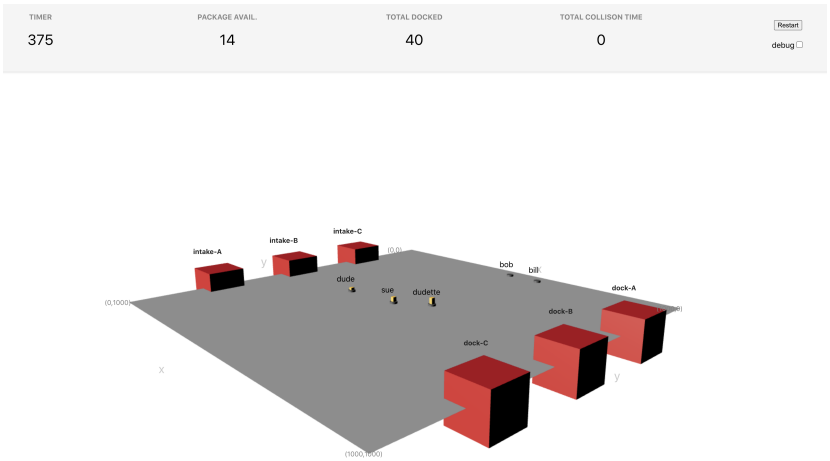

Fig. 1. Unmanned Fulfillment Center Simulation

TABLE I
NOTATION

Variable	Notes	Definition	Representation in Problem
P_i	Subscript is a lowercase letter	Number of packages the Robot is currently carrying that are assigned to be dropped off at the location i	P_a, P_b, P_c
P_I	Subscript is an uppercase letter	Number of packages the Robot can pick up from location I	P_A, P_B, P_C
D_i & D_I	-	Distance from current location to a location denoted by the subscript	$D_a, D_b, D_c, D_A, D_B, D_C$
x_i & x_I	-	Discrete value used to represent if the Robot should travel to the location denoted by the subscript	$x_a, x_b, x_c, x_A, x_B, x_C$

IV. Information about Code

A. Benchmark

A trivial algorithm for package transport is to command the robot back and forth between the same intake and dock. While this does allow for the transport of some amount of packages, the robot will eventually get full of packages that are not marked for its given dock. The robot will no longer be able to pick up or drop off any packages once this occurs. Through testing, we found that this trivial algorithm manages to drop off around 100 packages before this occurs.

B. Implementation

The warehouse simulation is contained in a web application, which communicates with the code base. The codebase is written in Python 3.9.7 using a venv environment for package management. Finally, the simulation is run on a Manjaro XFCE virtual machine with approximately 8 GB of ram and 4 cpu cores. Information about the warehouse simulation can be found at the gitlab repository[1]. Our code base can be found at our github repository[2].

One part of the code base is used to connect to the web application, while the other part is used to implement the optimization algorithm. The algorithm can be implemented with the below script

1) Move robot to first location
2) Determine possible routes given robot location
3) Calculate all D, P and efficiency values for each routes
4) Sort efficiency values, highest to lowest
5) Command robot position with destination associated with highest efficiency value
6) Repeat steps 2 through 5 until simulation is complete

Upon running a simulation for 3500 ticks (simulation time interval), we were able to successfully dock 955 packages.

V. Next Steps

This project is a great introduction into the optimization of a warehouse robot system, to take this problem to the next step we believe focusing primarily on utilizing more robots. In our code we focus on only having 1 robot in the system. It would be beneficial to look at the increased complexity when adding multiple robots. When doing this we would have to consider the robots colliding with one another as something we seek to prevent, and take measures of that in the code. Another area to look into is the effect of increasing the amount of intakes and docks. Our formulated equation would stay similar but with an increased number of variables for the intake and docks respectively. Robot battery life would also be something interesting to consider for the future. In the real world these robots require charge time in between as well. To look into this we would need to collect data on the battery of the robot

[1]https://gitlab.com/systems-iot/syse6301-robotics-fulfillment-project
[2]https://github.com/AdamCarrera/fulfilment-center

and create a set of conditions that instruct the robot to go charge at a certain threshold.

VI. Conclusion

A model of an unmanned fulfillment center was optimized. The fulfillment center contains three intakes, and three docks. An algorithm was developed, for one robot, to determine the next optimal destination for minimizing travel cost, therefore maximizing the number of packages transferred, and the number of packages docked. A python script is developed to implement the optimization algorithm and tested with a simulation model of an unmanned fulfillment center. Upon running the simulation, it was found that the lone robot was able to dock 955 packages over a simulation time of 3500 ticks.

VII. Group Member Contribution

The mathematical theory was primarily developed by Sivakrish Sivarajah with help from Adam Carrera. The code was primarily developed by Adam Carrerea with help from Sivakrish Sivarajah. Both team members contributed equally to the production of the report and presentation.