Project 3: Automated Package Delivery Service

Prepared by: Dena Mujtaba and Nihar Mahapatra

All projects have been tested on DECS PCs. You can use the Remote Desktop Connection provided by DECS to access these PCs (as described in the *Guidelines for Software Projects* document posted on D2L) if you do not want to install Anaconda (a free, open-source Python distribution) on your own PC.

Learning Objectives:

- Learn how to implement an automated package delivery service system with ROS.
- Understand scheduling algorithms and learn how to implement them in Python and ROS.

Required Equipment and/or Files:

- Project3sim_<OS>.zip: The simulation program for this project's task (OS corresponds to your computer's operating system – use win64 if on DECS PCs). You can download this from D2L. These files are larger (100+MB) than some of the other files because they provide cross-platform support, obviating the need to install additional software.
- ros_shim.py: The Python file with the classes to use ROS. This file is in the Project 1 folder on D2L.
- You will also need a PC with Python 3 and Jupyter installed. You can also use the DECS lab PCs through the remote desktop feature.

Project Task:

In this task, you will implement an automated package delivery service (APDS) system. With more and more people using online shopping services like Amazon, autonomous delivery options have started to become more prominent and are being tested in different cities [source]. In this project, you will set up a smaller version of this system based on two state machines described in the lecture modules and a new state machine you will design, and use two different scheduling algorithms for package delivery and compare them.

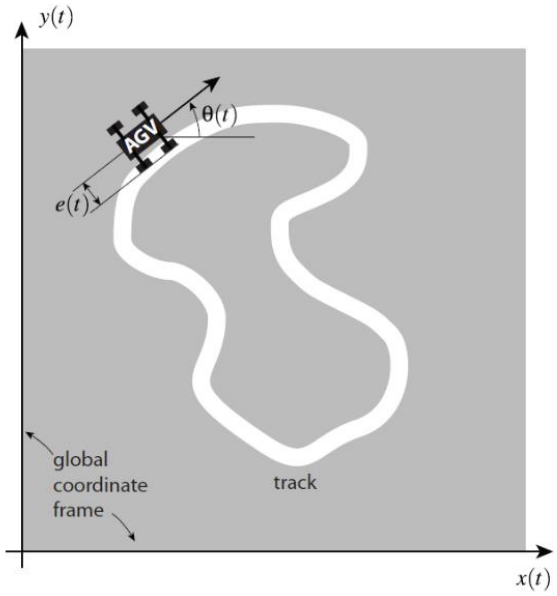

Figure 1: Illustration of the AGV, from Chapter 4, Figure 4.12 of E*dward A. Lee and Sanjit A. Seshia*, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, *Second Edition.*

First, it is essential that you review the following material from the textbook *Edward A. Lee and Sanjit A. Seshia*, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, *Second Edition* to ensure you have the proper background: (1) Section 4.2.3 and especially Example 4.8 on automated guided vehicle (AGV) and Figs. 4.12 – 4.15; (2) Example 5.6 and Fig. 5.6 on servers processing requests; and (3) Chapter 12 introduction (i.e., the introductory paragraph before Section 12.1), Section 12.1 (all four subsubsections), and Section 12.3 covering scheduling concepts and algorithms. The remainder of this document assumes that you have reviewed all of this background. If a certain term or concept after this point in the document is not clear, make sure you review the background material above. We will use two metrics to

compare scheduling algorithms: *maximum lateness* and *total completion time* or *makespan*. Note that the examples from this background material that are referenced in this project will be used with some modifications.

There are two systems in this project that will interact with each other. One is the automated package delivery service (APDS) system that you will design and implement to control an AGV and perform package pickup, scheduling, and delivery. The other is the e-commerce service (ECS) which will provide your APDS the current AGV location, the current time, information about packages to be delivered, and other relevant information to be detailed shortly. The ECS is provided to you as part of the simulation environment. The APDS you will implement will have one vehicle. This vehicle will need to navigate around the street as shown in Figure 1 (i.e., follow the path, in either the clockwise or counterclockwise direction for each package, changing direction between package deliveries if deemed preferable). The delivery route used for this project is illustrated in Fig. 7. Along this path, there are stops, or houses, where the vehicle will need to stop at to deliver a package. The vehicle will need to deliver packages one after another as requests arrive. To deliver a package, the vehicle will need to go to a designated pickup station (to be specified below) to get the package and go to the package's corresponding destination house for delivery. Only one package can be picked up and delivered at a time before the next package is picked up. To deliver these packages, you will implement two different non-preemptive scheduling algorithms i.e., delivery of one package is not interrupted by the presence or arrival of other packages in the package queue. The first is a fixed priority scheduling algorithm called first-come first-served (FCFS) (aka first-in first-out or FIFO). This is a simple scheduling algorithm in which packages are picked up and delivered in the same order as they arrive in the package queue. The second is a dynamic priority scheduling algorithm called earliest deadline first (EDF) or Horn's algorithm described in Section 12.3 of the textbook referenced earlier.

You will implement the APDS system as a composition of three state machines that interact with the ECS system. The first state machine is the AGV state machine. Its purpose is to keep the vehicle on the path and you will set it up similar to (but not same as) the one shown in Figure 2 (*Hint: The origin is at top-left corner with x-value increasing to the right and y-value increasing vertically down, so some speed parameters may need to be modified.)* The AGV state machine will take as inputs: (a) *start* to start the AGV and drive it at a fixed speed, (b) *stop* to stop the AGV, (c) the *x-coordinate* position of the AGV, (d) the *y-coordinate* position of the AGV, (e) the *orientation* of the AGV, and (f) the *displacement e(t)* of the AGV center from the track (see Figure 3 below). The AGV state machine will produce outputs: (a) the *x-velocity* of the AGV, (b) the *y-velocity* of the AGV, and (c) the *angular velocity* of the AGV. AGV displacement relative to track will need to be constrained within the values $\varepsilon_1 = 7ft$ and $\varepsilon_2 = 10ft$ for this project. The *x* and *y* velocities in Fig. 2 have cosine and sine terms, respectively, multiplied by a factor of "10" in three states: *straight*, *right*, and *left*. This "10" factor should be changed as follows. In the *straight* state, replace "10" by "30". In the *right* and *left* states, replace "10" by $\frac{30}{1+ e^{-(t-5)}}$ to determine AGV speed as a function of time *t* in seconds. For the angular velocity in the *right* and *left* states, use $\frac{\pi}{16}$ instead of π (using appropriate signs). Furthermore, you will need to provide a *stop* input once the vehicle reaches the destination house (or is within 4.5 times the ε_2 value noted above), and a *start* input after the package has been delivered at destination house (the time required to place package at destination house after reaching it is explained below in the ROS topics).

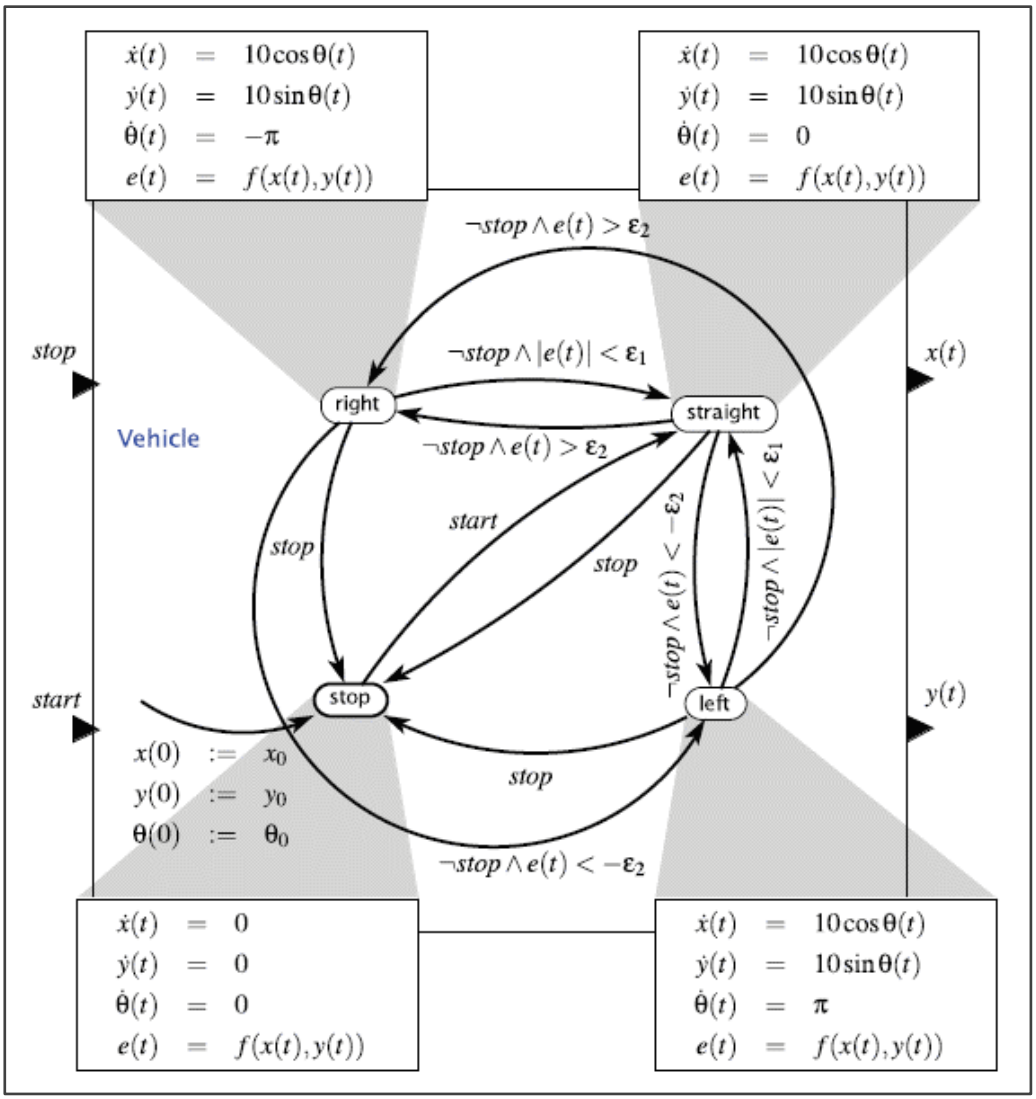

Figure 2: The automated guided vehicle from Chapter 4, Figure 4.13, of E*dward A. Lee and Sanjit A. Seshia,* Introduction to Embedded Systems, A Cyber-Physical Systems Approach, *Second Edition.*

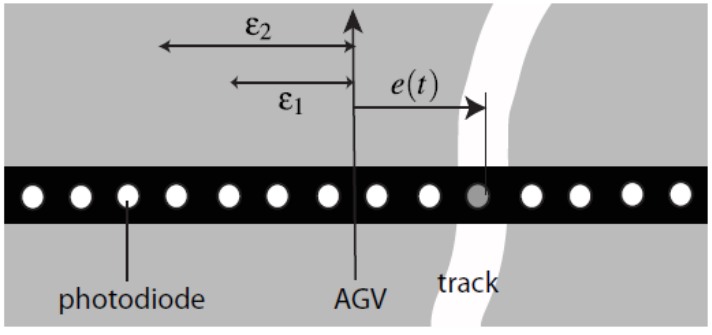

Figure 3: An array of photodiodes under the AGV from Chapter 4, Figure 4.14, of E*dward A. Lee and Sanjit A. Seshia,* Introduction to Embedded Systems, A Cyber-Physical Systems Approach, *Second*

The second state machine is the scheduling service (SS) state machine, to handle package delivery requests. An example (though not the exact setup), is shown in Figure 4, where an input *request* is used to indicate that a package delivery request has arrived, and the SS machine can move between the states *idle* and *serving*. The *pending* variable keeps track of the number of package requests in the queue that still need to be serviced by the AGV. However, the requests received by the SS will need to be processed in accordance with a scheduling algorithm (in the *serving* state) discussed earlier. Each package has a number of parameters associated with it: (a) a package request *arrival time* (this is the time the request arrived in the queue); (b) a *destination house* for the package; (c) a *delivery time* which is the time required for delivery once the vehicle has

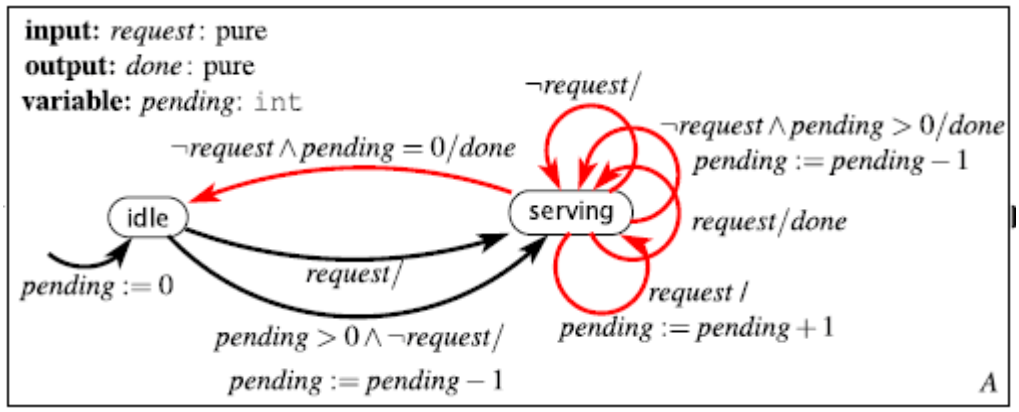

Figure 4: Scheduling service, adapted from Chapter 5, Figure 5.6, of E*dward A. Lee and Sanjit A. Seshia,* Introduction to Embedded Systems, A Cyber-Physical Systems Approach, *Second Edition.*

arrived at the destination house (i.e., the time required to place the package at an appropriate spot near the destination house once the vehicle has arrived at the house), and (d) a *delivery deadline* (i.e., the time by which the package should be delivered). The delivery deadline parameter is used to indicate a constraint on the time of delivery that is preferable to meet or not miss by much, but missing the deadline is not an error. This is an example of a *soft real-time scheduling* problem. Note that the package arrival time parameter needs to be inferred by your program based on the time stamp at the time of arrival, whereas the other package parameters are provided explicitly with the package. Along with "request", another input (from the third state machine described next) is the input "available" which indicates to the SS state macbine that the AGV has returned to the pickup station and is available for the next package pickup. The output for this state machine will be the package ID of the selected package and its destination house and delivery time. After the package is delivered, the AGV needs to return to the pickup station.

The third state machine is the delivery service (DS) state machine which will link the other two state machines (AGV and SS state machines). Once the SS state machine selects a package (if any in the package request queue) that should be processed next based upon the scheduling algorithm, it "informs" the DS state machine (via output to input connection between state machines) which package was selected (i.e., its package ID) and the destination house location and delivery time for the package. Then the DS state machine will instruct (again via output to input connection between state machines) the AGV state machine to go along the track to the detination house, deliver the package, and return to the pickup station. Note that the DS state machine should "inform" the ECS system after package delivery (but before AGV returns to pickup station) by publishing to the "delivered" topic as explained below both the package ID and the time of delivery. Also, after the AGV returns to the pickup station, the DS state machine needs to "inform" the SS state machine that the AGV is "available" for the next package pickup.

The ROS topics to subscribe and publish to/from the simulator for implementing this system are as follows (consider any position variables as relative to the top left corner of the screen):

- `/request` (*List*): The ECS system will publish to this topic a list of packages that have to be delivered. It will publish a custom std_msg of type "**List**" that is a list of std_msgs of type "**Dictionary**", that is a dictionary containing information with the keys "*delivery_time*" (i.e., how long it takes to drop the package off which is the amount of time the AGV should wait at the destination house once it has arrived there), the "*deadline*" (i.e., when the package should be delivered by at the destination house – the time in seconds), "*locationX*" (i.e., the *x* position of the destination house in feet), "*locationY*" (i.e., the *y* position

of the destination house in feet), "*houseNumber*", and "*id*" (i.e., a unique identifier to tell apart this package from the others in the queue).

- */addRequest* (*Dictionary*): If you want to test out different types of packages/deadlines/houses, you can publish to this topic a Dictionary object with the above parameters (except for "*id"* which will be generated by the simulator), instead of using the button in the simulator (which randomly generates a Dictionary delivery object and adds it to the queue). An example is given in Figure 5.

```
pkgRequest = rospy.Publisher("/addRequest", std_msgs.Dictionary)
pkgRequest.publish(
{
    "delivery_time": 1,
    "locationX": 575,
    "locationY": 265,
    "houseNumber": 1,
    "deadline": 500
})
```

Figure 5: Sample code illustrating how to publish to the /addRequest topic.

- */AGVxPosition* (*Float*64): The APDS system should publish to this topic the *x* coordinate of the vehicle on the track (in feet).
- */AGVyPosition* (*Float*64): The APDS system should publish to this topic the *y* coordinate of the vehicle on the track (in feet).
- */AVGanglePosition*(*Float*64): The APDS system should publish to this topic the orientation of the vehicle on the track (in radians).
- */displacement* (*Float*64): The ECS system will publish to this topic periodically the displacement value of the AGV from the center of the track (in feet).
- */time* (*Float*64): The ECS system will publish to this topic periodically the current time of the system. The time in the simulator is set to 2x faster than real time. ROS's read rate of data can be changed to sync with this by using the `rospy.Rate(Hz)` function call, wherein the read rate is provided in Hertz. However, the default setting of 20 Hz is recommended. If you implement an internal clock for your APDS (i.e., to calculate the *t* for the speed, we recommend you increment in your code by 0.1 seconds for every reading from this time topic. This is further illustrated below in Figure 6.

```
#set the speed, assuming you have a global
#variable t defined for time
def speedFn(data):
    global t
    t += 0.1

rospy.Subscriber("/time", std_msgs.Float64, speedFn)
```

Figure 6: Sample code illustrating how to use the data from the /time topic.

- */delivered* (*Dictionary*): The ECS system is subscribed to this topic for the Dictionary std_msg that was delivered. Publish to this topic the ID of the package you have delivered, once the package has been

delivered to the house and the "*delivery_time*" has been fulfilled. If you publish to this topic before the delivery is complete, the package won't be removed from the queue.

- $/pickup$ $(Dictionary)$: The ECS system is subscribed to this topic for the Dictionary std_msg (i.e., pickup) from the pickup station to add to the vehicle. The AGV can hold at most one delivery/package at a time. If you try to pick up a package that is not in the queue, or you are not at the pickup station, the package will not be added to the vehicle.
- $/AGVxVelocity$ $(Float64)$: The ECS system is subscribed to this topic for the *x* velocity of the vehicle (in feet per second).
- $/AGVyVelocity$ $(Float64)$: The ECS is subscribed to this topic for the *y* velocity of the vehicle (in feet per second).
- $/AGVangularVelocity$ $(Float64)$: The ECS is subscribed to this topic for the angular velocity of the vehicle (in radians per second).
- $/algorithm$ $(String)$: The APDS system should subscribe to this topic to switch the scheduling algorithm used. The two outputs are "FCFS" and "EDF", depending on what was toggled in the simulator.

Overall, the steps to deliver a package are as follows:

1. The vehicle will need to drive to the pickup station or be within $4.5\varepsilon_2$ of it. At this point, the package (determined by the scheduling algorithm) will need to be added to the vehicle using the */pickup* topic. *Hint: To get coordinates of the pickup station, you can use the AGV's starting position in the simulator.*
2. The vehicle with the package will need to drive to the correct house described in the Dictionary delivery object's coordinates. *Note: the vehicle can move in any direction on the track (clockwise or counterclockwise), it just needs to follow the track itself as shown in Figure 1.*
3. Once the vehicle is at the house, or within $4.5\varepsilon_2$ of it, the vehicle will need to stop at the house for the "*delivery_time*" in seconds (this mimics going to the house and placing the package).
4. After this wait time, the */delivered* topic will need to be used to publish that the package has been delivered.
5. Next, the vehicle returns to the pickup station to get the next package. If there is no other package in the queue, the AGV still should return to the pickup station.
6. *Additional: If your APDS receives a new request while carrying out a delivery, it should finish the current delivery before processing the next one.*

Implement this A*PDS* by creating one Python program following the code guidelines provided for software projects. The A*PDS* will need to use the simulation environment provided for the project, which can be downloaded from the link above in the Required Equipment section above. This will provide input data and allow you to test different cases for your A*PDS* (you can click the "Add to Q" button to trigger a package delivery request being added to the queue, and the house number for the delivery will be displayed in the simulator queue).

The five test cases we will check for are as follows:

1. The A*PDS* can control the vehicle to drive around the track.
2. The APDS can control the vehicle to drive around the track and deliver one package by stopping at the house by the track.
3. The delivery system can drive around the track and deliver one package and return to the pickup station to pick up another package.
4. When multiple (3+) packages are requested for delivery, the APDS can control the vehicle to drive around the track, stop at the correct houses to deliver the packages with an FCFS scheduling algorithm.

5. When multiple (3+) packages are requested for delivery, the APDS can control the vehicle to drive around the track, stop at the correct houses to deliver the packages with an EDF scheduling algorithm.

Deliveries used for testing are illustrated in Table 1 below:

Package Arrival Time	Delivery Time	House Number	Deadline	Time Package Dropped off (FCFS algorithm)	Time Package Dropped off (EDF algorithm)
2	3	3	140		
3	5	2	1005		
6	1	2	250		
11	6	1	700		
12	2	3	900		
13	1	3	1500		
20	1	2	1500		
22	2	1	5000		

Table 1: Example deliveries used for test cases. You can publish these to the APDS to test with the correct parameters.

Furthermore, the coordinates of different objects in the simulator are shown below in Figure 7.

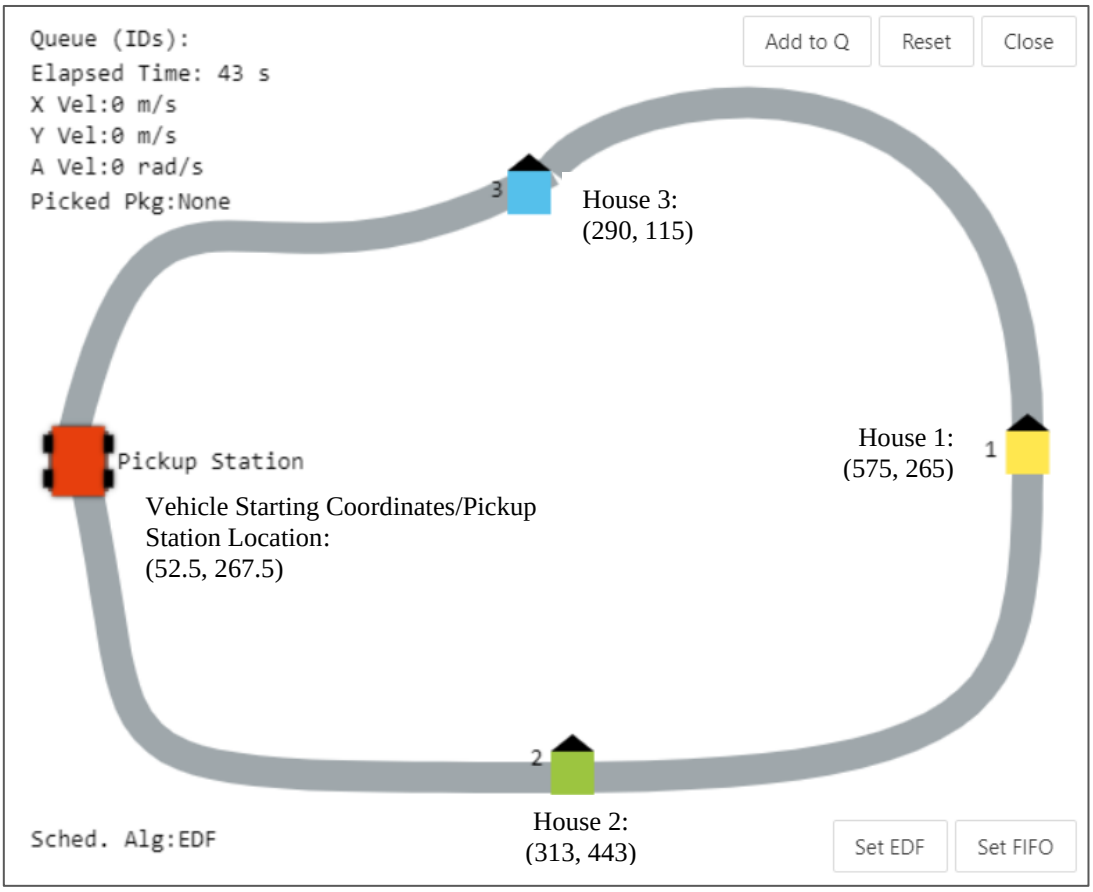

Figure 7: Illustration of the package delivery route.

Final Deliverables and Grading:

You will need to submit the following files to the D2L dropbox:

1. Your Python file(s) implementing your system
2. A PDF of your report answering the project questions.

The following is the breakdown for grading of this project:

- Functionality (55 points total):
    - Test Case One: 10 points
    - Test Case Two: 10 points
    - Test Case Three: 5 points
    - Test Case Four: 15 points
    - Test Case Five: 15 points
- Organization (5 points total)
    - This is according to the code submission guidelines in the *Guidelines for Software Projects* document.
- Report (20 points total):
    - For this project, your report should be a PDF document with your name, NetID, project number, and list of all code files. You will also need to provide a section answering the following questions:
        1. (3 points) What is the purpose of having both thresholds $\varepsilon_1$and ε_2?
        2. (6 points) Provide a diagram illustrating the APDS composition machine, containing the AGV, SS, and DS state machines and their interconnections, and its connection to the ECS system. Label all inputs, outputs, variables, and arcs and draw bounding boxes around state machines.
        3. (4 points) What is the maximum lateness for the package delivery requests in Table 1 for the FCFS and EDF scheduling algorithms? (Note: for this and the next question, provide a copy of Table 1 above with the rightmost two columns also filled in to explain your calculation)
        4. (4 points) What is the total completion time or makespan for the package delivery requests in Table 1 for the FCFS and EDF scheduling algorithms?
        5. (3 points) What are the pros and cons of the FCFS and EDF scheduling algorithms?