Improvements made:

* Pizza is assigned on distance basis as each pizza gets assigned to closest available working robot
* Amount of code needed to be written is reduced as all the features are capsuled in few compulsory object bound method

Ecosystem.assign()

Robot.deliver()

Robot.move()

Ecosystem.update()

* Most of the codes are written in a user friendly way. It doesn’t give error to common mistakes.
* Robot kinds are assigned pizzas that are within radius of 300 unit. On the other hand, no restriction on pizza target.
* Charging Algorithm:
* An effort was made to estimate time and energy it would consume to get to pizza and its target. This was not accurate and caused several problems, such as robot started hesitating/ going back and forth and conditions change. This leads to waste of energy. Hence more robots got broken than simple percentage charging algorithm.
* I could reduce the problem by improving the calculation of energy and time. This would require additional inspection on how energy is spent and on what formula.
* Effort was already made to contemplate the line of code that manage the soc quantity. Further debugging and inspection could be made.
* Further debugging
* Separate class was created for each registerable type. This enables class to have attributes and in-bound methods which can also be inherited down to other similar classes.

List of features added:

* Deliverable\_list
* Station\_list
* Pizza\_assignment
* Ecosystem.pizza\_assign()
* Robot.deliver():

Optimal charging

When broken or turned off, robot drops the pizza and assigns it to other robot

Effort was made to find better algorithm to charge robots

* Robot.move()

Each classes might have their unique attributes that special to the only class that owns based on their characteristics.

Solar\_charger: could use illuminance value to charge robot more realistically whilst maintaining carbon-emission-free.

Repait\_station: fixes damaged robots instead of charging

Recycling\_station: recycles broken robots and create new one when collected enough materials.

This coursework should have highlighted the use of OOP in software design.

Summarise how the 4 main concepts of OOP (see WSA010 JNB08 Structured Programming.ipynb) have been employed in the Robot Ecosystem. Your answer should make specific reference to the objects, methods, attributes and any other code you have deployed in the courswork, and discuss OOPs suitability for this sort of application.

 (500 words minimum - 1000 words maximum)

OOP:

OOP, as the name suggests, interprets every code as an object. Every object in OOP belongs to a type. A new type can be created using “class” function and then it can instantiate an object which belongs to that new type created. OOP is very powerful and diverse way of programming as each object is in-dependent and self-containing, and it maintain its structure within itself. You can access data within data that’s also can be within another data. Simple example would be a value returned by an in-bound function of an object.

OOP is the most used programming language type. OOP’s 4 fundamental concept include encapsulation, abstraction, inheritance, and polymorphism. Abstraction lets you simulate real life entities thus OOp is the most suitable choice for this course work.

Encapsulation:

Encapsulation encapsulates attributes and methods in a class closing direct access and protecting its data from unwanted changes. Encapsulation is used from start to finish in the course work. Starting with import function, it brings libraries that encapsulates useful functions which can then be accessed. Then functions and validation tools are created, followed by the ecosystem class. When ecosystem object is instantiated, programmer can access ecosystem object and pull or push data from anywhere after the instantiation. Same for any code that follows.

Abstraction:

Abstraction helps programmers to represent real world entities into a set of class that can return desired outcome. In this course work, robot, drone, droid were abstracted to model its functionality. They were not actual robot, droid, nor drone but can give you similar outcome and function alike what they were in real life. Robot, Drone, Droid classes were created with its attributes and methods that are designed to deliver functionality of real-world robot system, along with charging station. They have a mission to deliver items which, in this course work, are pizzas. This enables us to see how a robot would perform with given algorithm without actually needing to create robot system and louds of pizzas. This helps us simulate real life objects before we do it in real life.

Inheritance:

Classes in OOP can inherit from another class that will be entitled as the parent class. Inherited classes are child classes of a parent class that mimics all its attributes and methods. Furthermore, you can add or override on these attributes and methods if the need be to customize and differentiate from other child classes. In this course work, droid and drone classes inherit from robot class.

Polymorphism:

Polymorphism lets you handle classes from the same family in a similar way and they can be passed around without needing to change much. In this course work, ecosystem and robot, droid, drone classes were tightly bound to work together to produce the abstracted outcome or simulation. Ecosystem takes value for each class objects and treat them in the same way to produce the desired functionality. Robot, Droid, Drone also takes value from ecosystem and pass it around each other to produce certain value.