SOFTWARE ENGINEERING M3 (2019-2020) ASSESSED EXERCISE

Overview

This is the description for the Software Engineering M3 assessed exercise. This exercise is worth 50% of your marks for the course. This document is split into two sections:

1. **Product Description**: This is a description for a new software product that an end-user wants to have developed. It provides high level information about what the user envisages the product will do.
2. **Tasks**: For this exercise, you will be acting as both a software architect and software developer. Your boss has tasked you with converting the product description into documentation for their development team using the software engineering techniques you have learned, and then assigned you to build two of the classes that they need in Python. The tasks section describes what you need to produce.

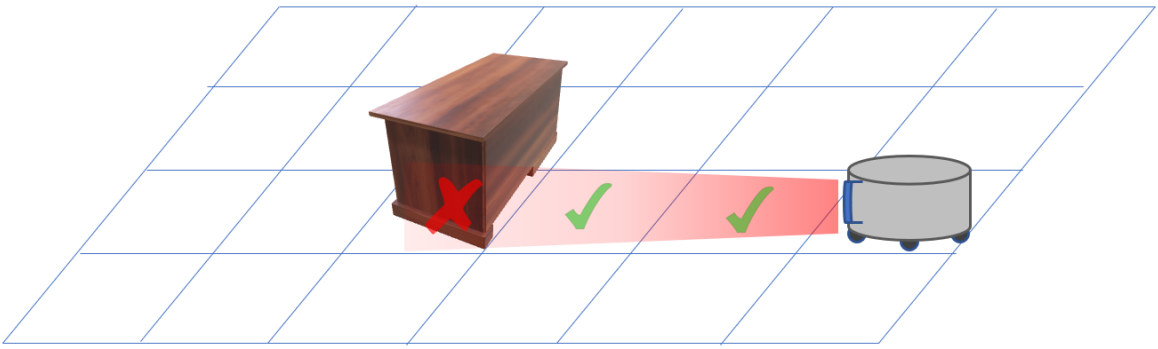
Guidance:

- For all tasks, feel free to add comments to your diagrams and code explaining any additional assumptions that you have made.
- Not all of the information in the description is relevant to the tasks that you are assigned.
- The number of marks in each case is for you to see how much weight each section has. Your final mark will be provided as a grade on the University scale (A1, A2, etc)

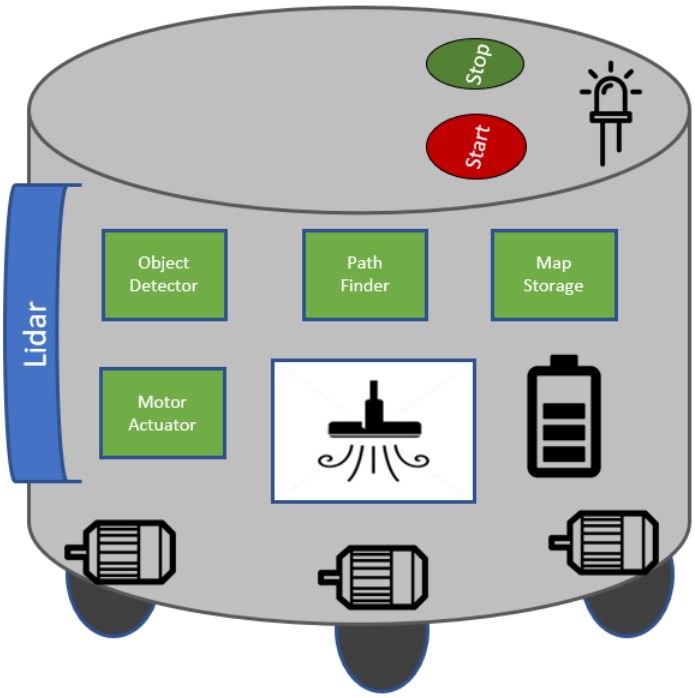
Product Description

RoomRestore Technologies

Automated Hoover (Ah!)



RoomRestore is a new company that specializes in designing and manufacturing small-scale automatic hoovers (vacuum cleaners) for use in busy offices. RoomRestore is developing a new 'Ah!' line of automated hoovers, where their primary selling point is the integration of accurate LIDAR (Light Detection and Ranging) sensors that can rapidly detect and estimate the size of nearby objects at a range of up-to 10 meters, reducing the need for janitorial staff.



The primary components of an Ah! Series hoover are the wheels (with associated electric motors), the hoover attachment, the battery, the LIDAR sensor and four chipsets that run the software (Object detector, Path Finder, Map Storage and Motor Actuator). The object detector takes raw signals from the LIDAR sensor and determines 1) if an object is directly in front of the hoover, and 2) provides topology information for the space 10 meters in front of the hoover. The Map Storage receives topology information from the Object Detector and maintains a map of the explored area of the room. The Path Finder uses the current mapping information to direct the hoover to areas of the room that have not been hoovered and do not contain obstacles, as well as periodically rotating the hoover 360" to provide updated map information of the surrounding area. The Motor Actuator controls the wheels. It receives input from the

Path Finder during normal operation, but that signal may be overridden by the Object Detector if the area directly in front of the hoover becomes blocked. On top of the hoover there are two buttons, start and stop, along with an LED that indicates whether the hoover is on (with a steady light), and flashes if the battery is low. The stop button disables the software chipsets and hoover attachment, while the start button re-enables them. The default hoover attachment can be swapped out with other attachments designed for different floor types. Which attachment is installed determines the target movement speed of the hoover.

RoomRestore engineers are subcontracting you to develop parts of the software of the hoover, focusing on the Object Detector, Path Finder and Motor Actuator components. When powered, the LIDAR sensor acts as the main driver of the software, and periodically calls the Object Detector when it has new sensor readings available. The Object Detector receives LIDAR distance estimates (in meters) for the area directly in front of the hoover roughly every 200ms (milliseconds). The LIDAR scan has a resolution of 300 (wide) x 200 (tall) evenly distributed data points. These data points are exported as a list with 60k entries. If any data point is equal to or less than 0.2 then the Object Detector calls the stop function of the Motor Actuator. Once the obstacle is removed, the Object Detector calls the continue method of the Motor Actuator. In cases where there is no blocking object, the Object Detector sends both the data point array and the distance to the nearest object to the Map Storage. The Motor Actuator receives input from both the Object Detector and Path Finder components. The Motor Actuator has three states that it can be in – traveling, blocked or stopped. As mentioned above, the Object Detector notifies the Motor Actuator if there is an object directly in front of the hoover, which changes the desired motor state from traveling to blocked (which in turn halts the hoover). Similarly, calling the continue function sets the state back to traveling. When first started or a path has been explored, the Motor Actuator contacts the Path Finder for a new path, which returns both turn information (degrees to turn left or right) and a desired distance to travel (which may be 0). The hoover will then turn the specified number of degrees and then travel the distance. If the desired motor state switches to traveling, it checks to see what hoover attachment is connected and based on that sets its speed via a wheels library (which provides turnLeft, turnRight and targetSpeed methods).

Another development team is working on the other hoover components. They have provided you with three python classes for you to use in your solution:

```python
from abc import ABC, abstractmethod

class HooverAttachment(ABC):

    def __init__(self, name, attachmentSpec):
        self.attachmentName = name
        self.attachmentSpec = attachmentSpec
        super().__init__()

    @abstractmethod
    def getTraverseSpeed(self): pass
```

```python
class PathFinderResponse:

    # Direction is a boolean, true is left, false is right
    # Distance is in meters
    def __init__(self, direction, degrees, distance):
        self.direction = direction
        self.degrees = degrees
        self.distance = distance
```

```python
from abc import ABC, abstractmethod

class PathFinder(ABC):

    def __init__(self, mapStore, attachment):
        self.__mapStore = mapStore
        self.__attachment = attachment
        self.__updateMapInXTurns = 0
        super().__init__()

    def getNewPath():
        self.__updateMapInXTurns -= 1
        if (self.__updateMapInXTurns <= 0):
            direction = True
            degrees = 360
            distance = 0
            self.__updateMapInXTurns = 2
            return PathFinderResponse(direction, degrees, distance)
        else:
            if isThereUnexploredRoomArea():
                currentMapData = self.__mapStore.getMapData()
                traverseSpeed = self.__attachment.getTraverseSpeed()
                availablePaths = generateAvailablePaths(traverseSpeed, currentMapData)
                return selectPath(availablePaths)
            else:
                return

    @abstractmethod
    def isThereUnexploredRoomArea(self): pass

    @abstractmethod
    def generateAvailablePaths(self, speed, mapData): pass

    @abstractmethod
    def selectPath(self, availablePaths): pass
```

Tasks

---

1. Identify the stakeholders in the system.

- RoomRestore
- Other Development Team (if not from RoomRestore)
- RoomRestore's Clients (likely, other businesses)
- Myself as an independent architect and developer

Assumption: I work as an external consultant and my future success depends on my performance with the RoomRestore Project

[2 marks]

---

2. Describe **two requirements** that you have **extracted from the description** above (two sentences each). For each, state whether it is functional or non-functional.

   - **Detect object**: the hoover must be able to detect objects in its path. It is a functional requirement that has a dedicated chipset that runs a software to meet this requirement

   - **Start and Stop**: the hoover must be able to be turned on and off. This is another functional requirement achieved by the buttons on the hoover itself

[2 marks (1 for each)]

---

3. Describe **two** additional **requirements** that you are assuming for the remainder of your design but are **not mentioned in the description** above (two sentences each). State whether they are functional or non-functional.

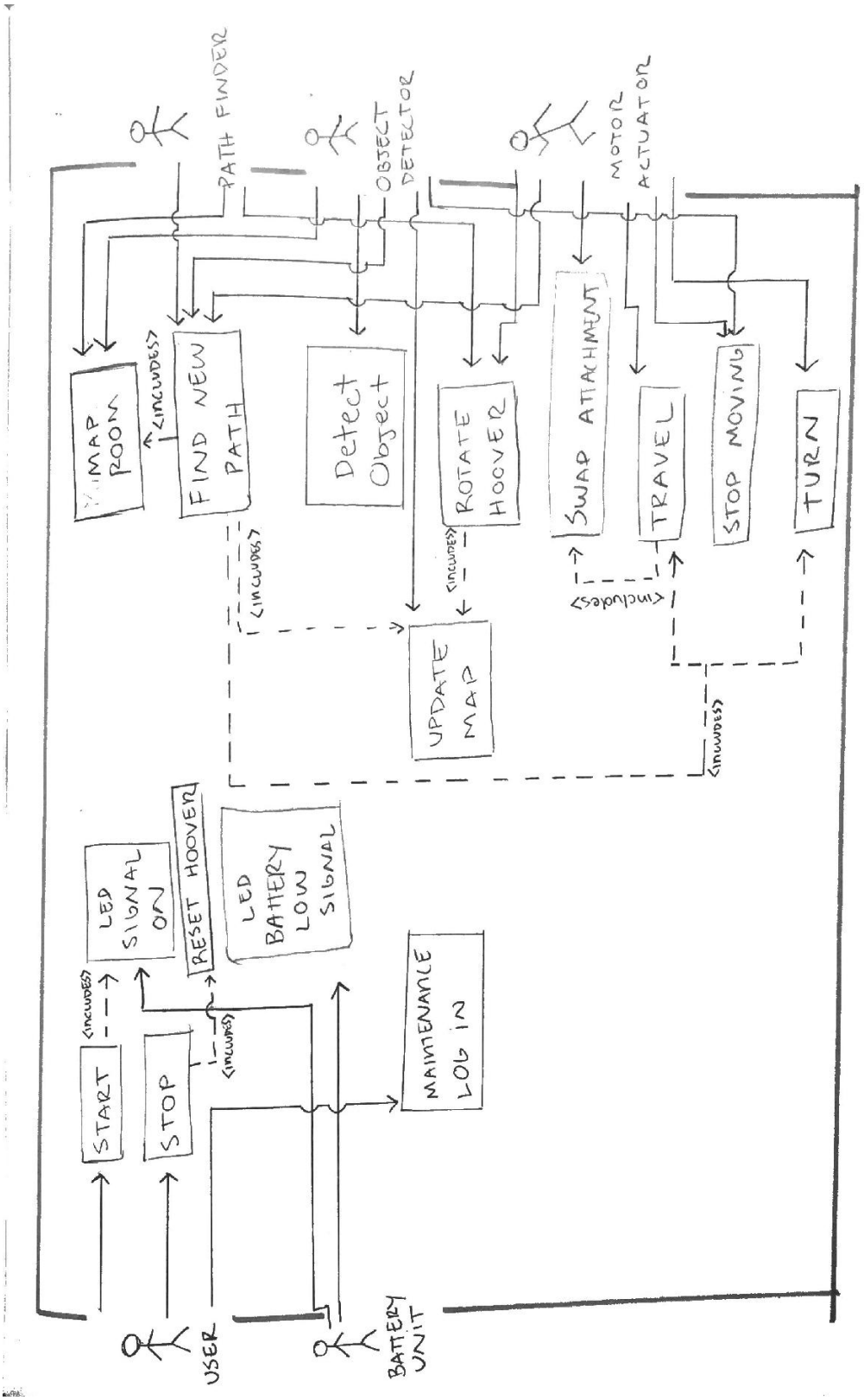
   - **Reset Hoover and Map**: user should be able to reset the hoover and recorded map in case there are any malfunctions (ie: hoover is stuck hitting the wall repeatedly) or once the hoover is finished exploring all paths. Functional requirement. This can be achieved by pressing holding the STOP button for 10 seconds

   - **Maintenance Security log-in**: The hoover has a log-in port that can be accessed by maintenance personnel. Log-in information is required so that the hoover is not accessed by the user. This is a non-functional requirement.

[2 marks (1 for each)

---

4. Draw a **use case diagram** to give a global view of how the hoover operates. Actors may be humans or software components. You must include a use-case called 'Find New Path',



[6 marks]

---

6. Write **use case scenarios** for two of your use cases.

Title**: Find New Path**

Goal: Find an unexplored or obstacle free path for the hoover

Actor: Path Finder

Preconditions: Map of the room by Object detector and Path Finder data of updated map

Trigger: Motor Actuator contacts the Path Finder for a new path

Course of Events:

- When first started, blocked (assumption stated above) or a path has been explored, the motor actuator calls the path finder for a new path.
- The path finder checks if the map is updated, if not it updates the map.
- The path finder uses the map of the room stored with data collected from the Object Detector and the periodical map updates check if there are unexplored areas
- If there are unexplored areas, the Path Finder returns turn information (degrees, and turn direction left or right) and a desired distance to travel (which may be 0) to the motor actuator.
- **Assumption:** If there are no unexplored areas, the Path Finder specifies the path back to towards where it first started (via turn direction, degrees and distance) and idles once it reaches its destination. See post conditions for outcome.
- The motor actuator will then turn the specified number of degrees in the direction specified and then travel the distance specified

Postconditions: The hoover will travel in the direction and for the distance specified. When the motor state switches to traveling, it checks to see what hoover attachment is connected and based on that sets its speed via a wheels library. In the case where there are no unexplored areas, the hoover goes back to the position it started and from here it can be reset by the user by pressing and holding the stop button for 10 seconds.

Title: **Detect Object**

Goal: To detect obstacles in the hoover's path and map out the area

Actor: Object Detector

Preconditions: LIDAR Sensor is engaged

Trigger: LIDAR Sensor calls the Object Detector when it has new readings available

Course of Events:

- The Object Detector receives LIDAR distance estimates (in meters) for the area directly in front of the hoover roughly every 200ms (milliseconds).
- If any data points imported from the LIDAR sensor is equal to or less than 0.2 then the Object Detector calls the stop function of the Motor Actuator.
- In this case, the motor actuator calls the Find New Path use case
- In cases where there is no blocking object, the Object Detector sends both the data point array and the distance to the nearest object to the Map Storage and the map is updated

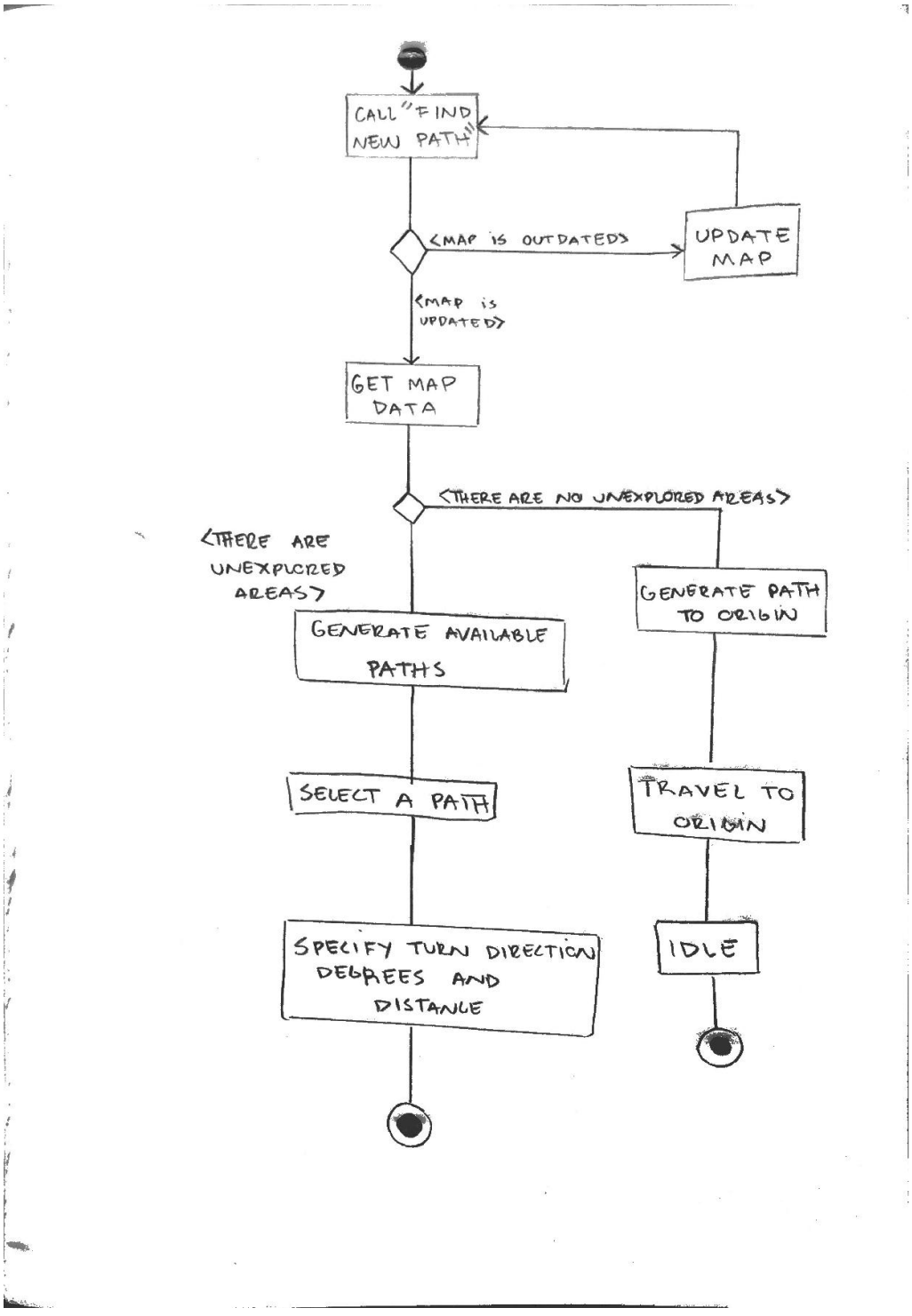
Postconditions: If there is an obstacle, the hoover 'Finds a new path' as explained above. If there is no obstacle, it continues on its intended patt

[6 marks (3 for each)]

---

7. Draw an **activity diagram** for the use case ‘Find New Path’.

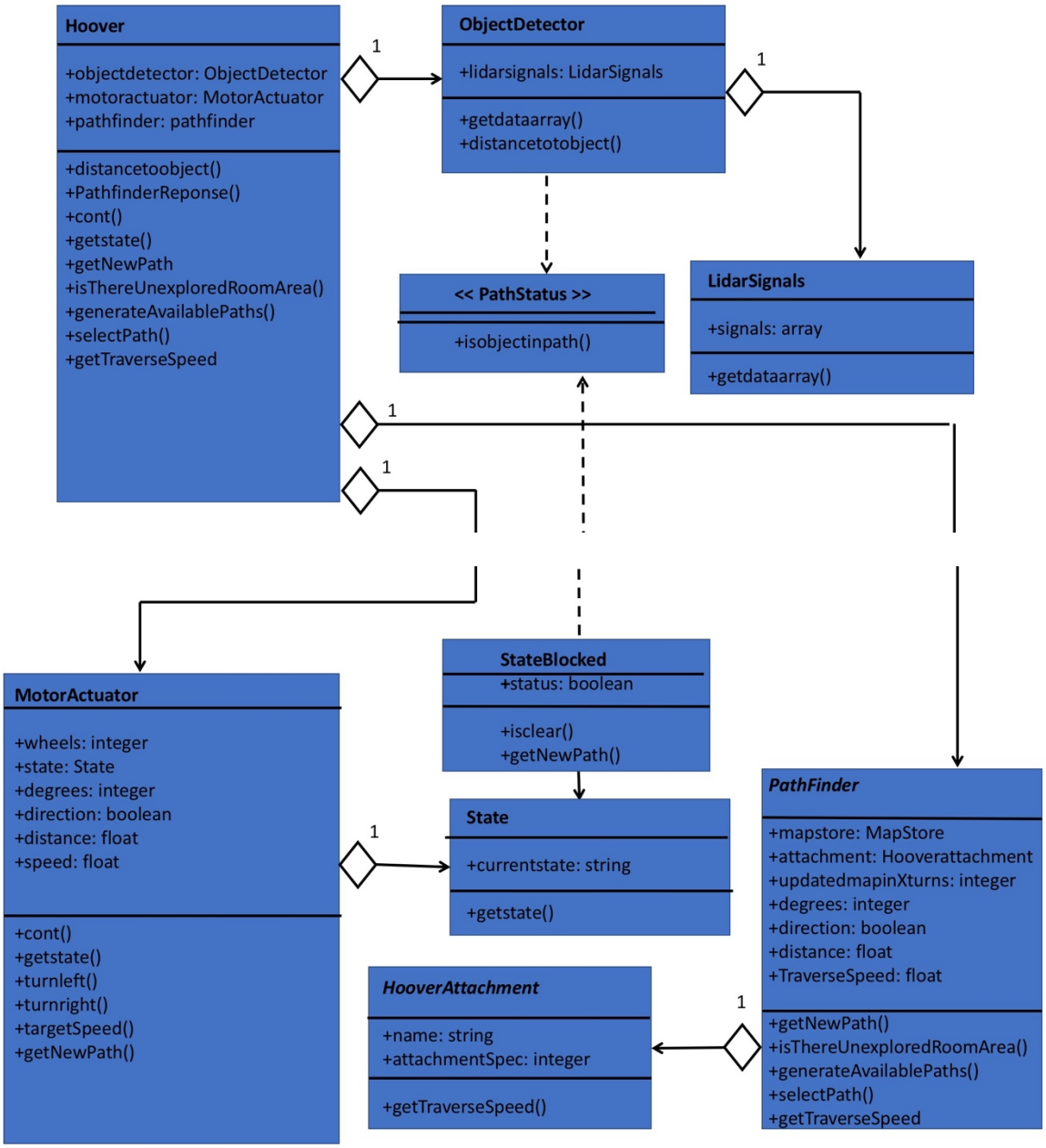
See use case scenario for details



[6 marks]

---

8. Design a **class structure** (present your solution as a class diagram) that represents the objects used by the Object Detector and Motor Actuator components. Your solution should include examples of *inheritance*, *abstract classes or interfaces*, and *aggregation or composition.*



Assumptions for class structure:

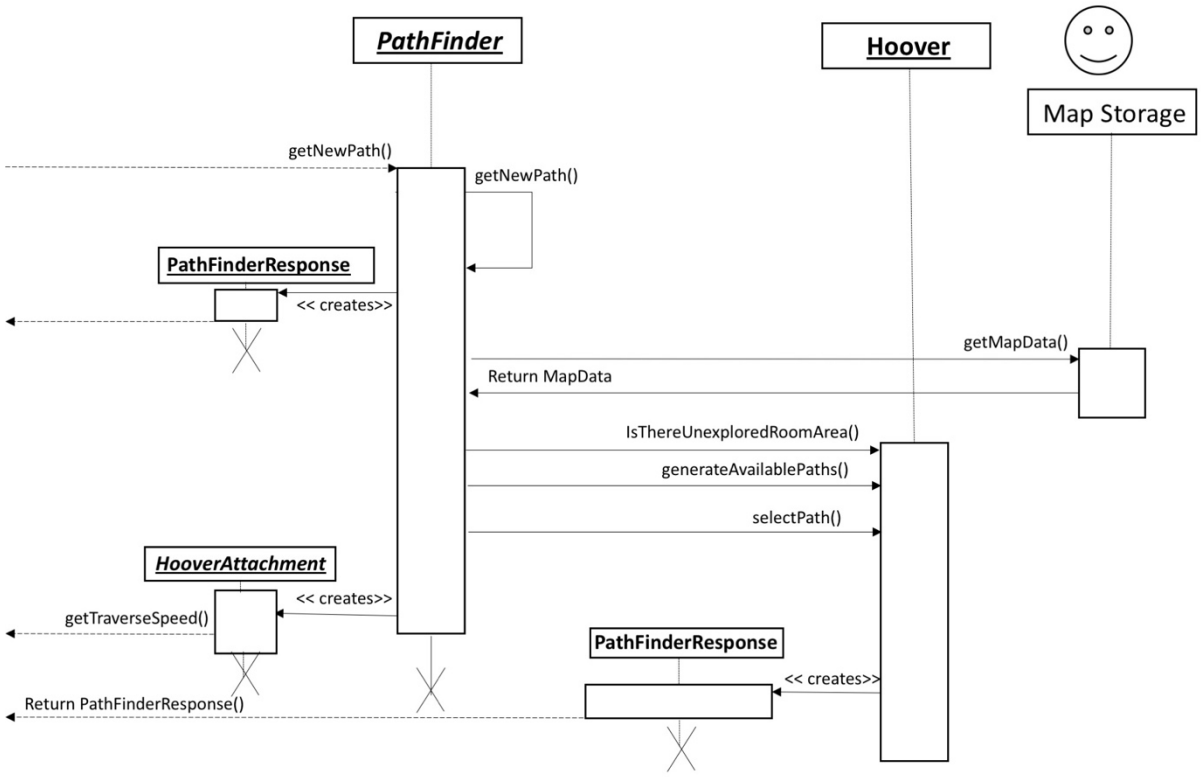
- I am excluding all classes, objects and methods that are not used directly by the Motor Actuator or the Object Detector.

- The definition of some of the abstract methods was left to the Hoover Class, leaving only the relevant ones to the Motor Actuator.
- Composing classes for the PathFinder class were left out of the diagram as they are considered outside of the scope of the question (ie. MapStore, PathFinderResponse)
- There is a flow of information between the Motor Actuator, StateBlocked and the PathFinder class. That's why they both call functions and attributes from this abstract class – however the Hoover class defines the abstract methods except for the relevant ones for the Actuator: getNewPath(), getTraverseSpeed()

[10 marks]

---

9. Draw a **sequence diagram** for the use case ‘Find New Path’.



[6 marks]

10. Provide a Python implementation of the Motor Actuator class based on your above designs, following good software engineering and development practices you have learned about. You should include either a screenshot or source of your class in Python here.

Assumptions:

- Since the question asks for Motor Actuator class, the interface “StateBlocked” will receive a pseudo array representing what would come from the Object Detector
- Even though not directly linked in the class structure diagram, the StateBlocked call methods from the PathFinder class.
- This is supported by the classes described in the brief above: HooverAttachment, PathFinder and PathFinderResponse

In [59]:
```
class MotorActuator:

    # This class is composed of standard attributes except for "state"
    # which is a class of its own
    # the Methods would be called by the Hoover depending on the informa
tion
    # from the PathFinder
    # The arguments for this class also come from the Hoover class
    # which in turn gets a lot of them from the PathFinder class
    # The methods are called in the actual code

    def __init__(self, currentstate, wheels, degrees, distance):
        self.wheels = 0
        self.__currentstate = state
        self.__degrees = 0
        self.__direction = None
        self.__distance = 0
        self.__speed = 0

    def cont(self):
        self.___state.currentstate = "travel"

    def getState(self):
        return self.__state.currentstate

    # depending on the string returned by the above function, the
    # Motor Actuator class can be called in the code and its attributes
    # set to the appropriately and the target speed adjusted.
    # For example, if getState()  returns "stopped" then the
    # instanced Motor Actuator class is
    # called as: a = MotorActuator(stopped, 4, 0, 0) and the method is c
alled
    # a.targetspeed (0)


    def turnleft(self):
        self.__direction = False

    def turnright(self):
        self.__direction = True


    def targetspeed (self, speed):
        self.__speed(speed)

    # in order to implement these specifications, the information
    # in this class is sent to the actual motor hardware.
    # for attribute of speed is turned from an integer into a voltage
    # within the wheel's microcontroller. The same applies for the
    # rest of the attributes
```

In [58]:
```
class state:

    #state of the hoover could be travel, stopped or blocked
    #here it is inialized as travel, and it could be easily chnaged

    def __init__ (self, currentstate):
        self.currentstate = "travel"


    def getState(self):
        return self.currentstate
```

In [55]:
```
class stateblocked (state, PathStatus):

    #This class is for when the object detector halts the hoover
    #in this case this subclass has an additional attribute, which
    #if true, the isblocked() method calls the getNewPath() from
    #PathFinder and sets the state to blocked
    #the omission of the "__" in the currentstate attribute carries it
    #over onto the "state" class

    def __init__ (self, currentstate, status):
        state.__init__(self,currentstate)
        self.status = True

    def isclear(self):
        if self.status == False:
            super().getNewPath()
            self.currentstate = "blocked"
        else:
            return

    # this abstract method below returns a boolean
    # The stateblocked class has the attribute "status" which can be
    # modified depening on the output of the below method.

    def isobjectinpath(self, pathstatus, array1, value):
        for x in array1:
               if x <= value:
                    self.pathstatus = False
                    return False
        return True

    # when this method is called, it takes in an array
    # from the Object detector and
    # a target value as as an argument
    # if any data point in the list is less than or equal to a
    # target value (in this case 0.2), then the Path is blocked (False)
```

In [57]:
```
from abc import ABC, abstractmethod

class PathStatus(ABC):

    def __init__(self):
        self.pathstatus = True

        #status can be True for Clear or False for blocked

    @abstractmethod
    def isobjectinpath(self, pathstatus, list1, value): pass
```

In [ ]:

In [ ]:

In [ ]:

[10 marks]