DP06: Prometheus AI Integration
-- Meta Reasoner Layer

By
DP 06:
Zisheng Wang 260619653
Li Qian 260617009
Jiachen Ju 260627548
Supervisor: Joseph Vybihal

Abstract

The primary goal for DP06 team is to design an algorithm for META Reasoner layer within the prometheus AI model. It consists of a data structure and reasoner that handles the abstract representation of the current local map of the robot and knows how to draw conclusions from that representation. The motivation of implementing this algorithm is to translate, paraphrase real world information into data structure that could be understood, so that the robot will have an conception of how the real world is constructed using surrounding objects. This algorithm can serve as a potential solution for a rescue/exploration robot with better memory efficiency. During this semester, we implemented the path generation algorithm, which takes a list of goals as inputs and generates an efficient path based on the map generated from exploration (the exploration and map building is implemented during first semester of DP) in order to achieve the goals while minimizing the distance travelled and battery usage. Instead of using tree-like data structure from last semester, a graph-like data structure is used in this semester, because it is a better representation for a clear relationship between node and its neighbours. In the following report, more detailed information regarding the data structure and algorithm will be provided. Since the surrounding environment is unknown to robot most of the time, an enormous map might be constructed after exploration phase. Therefore, a plan generation process in this scenario can be both time and memory consuming. The method to prevent certain scenario happening is to generate a simplified version map that contains only the necessary goal nodes, and then derives plan based on the simplified map.

Acknowledgment

Simulator team contributes to set up unity for META Reasoner layer demo.
Dr. John B. Matthews, MapVisualizer is inspired by his code

DP06: Prometheus AI Integration 1

Abstract 2

Acknowledgment 2

List of abbreviations 4

Introduction, Motivation and Objective 5

Background 5

Requirements and Constraints 6

High level requirements for META 6

Constraints 7

Design 8

System Design 8

Data structure 9

Algorithm 10

Exploration phase 11

1. Exploring the map 11

Path Generation Phase 12

2. User Set Goal 12

3. Make Plan 12

4. Decide to execute next plan step or not 12

5. Execute plan step 13

6. Status update 13

7. Need to make new plan? 13

8. Unexecuted plan step left? 13

Solution to Requirement 13

Solution to Constraints 14

Tools 15

Other Classes 15

Result 16

Impact on Society and the environment 18

Report on Teamwork 19

Conclusion 19

References 20

List of abbreviations

META	META Reasoner layer is the data structure and reasoner that handles the abstract representation of the current local world of the robot, and knows how to draw conclusions from that representation.
Node	A data structure we defined for storing objects that the robot discovered during traversing and building the map.
LocalMap	A graph like data structure that we defined for storing the map for a small local environment.
GlobalMap	A graph like data structure that we defined for storing the map for a large environment that contains multiple LocalMaps.
Edge	The link connecting 2 different local map or global map nodes
MST	Minimum Spanning Tree
BFS	Breadth-first search
DFS	Depth-first search
Data Structure	A data structure is a specialized format for organizing and storing data.

Introduction, Motivation and Objective

The robot for this design project consists of sensors, actuators and AI model. The AI model is defined to be a multi-level algorithm consisting of four layers (Neural Network, Knowledge Node, Expert System and META-Reasoner), similar to an operating system or a network stack. The AI model can be used to classify, store, track, and translate high-order state information for a robot. The model deals with raw sensor datas and translate them into an abstract representation of the current environment explored by robot. META Reasoner layer is the data structure and reasoner that handles this abstract representation of the current local world of the robot and draws conclusions from that representation. The final goal for DP06 is to construct map using abstract representation, find most efficient path to achieve specific goal required by user on local map or global map. The goal for this semester is to generate an efficient path to achieve goals step by step based on the local map we constructed last semester. This AI model can be used by a rescue or exploration robot which could accomplish some tasks that are difficult to achieve by human-being but easily achievable by robot.

Background

The full AI model consists of four layers[1], they are Neural Network, Knowledge Node, Expert System (Recommender System) and META-Reasoner as shown in Figure 1.

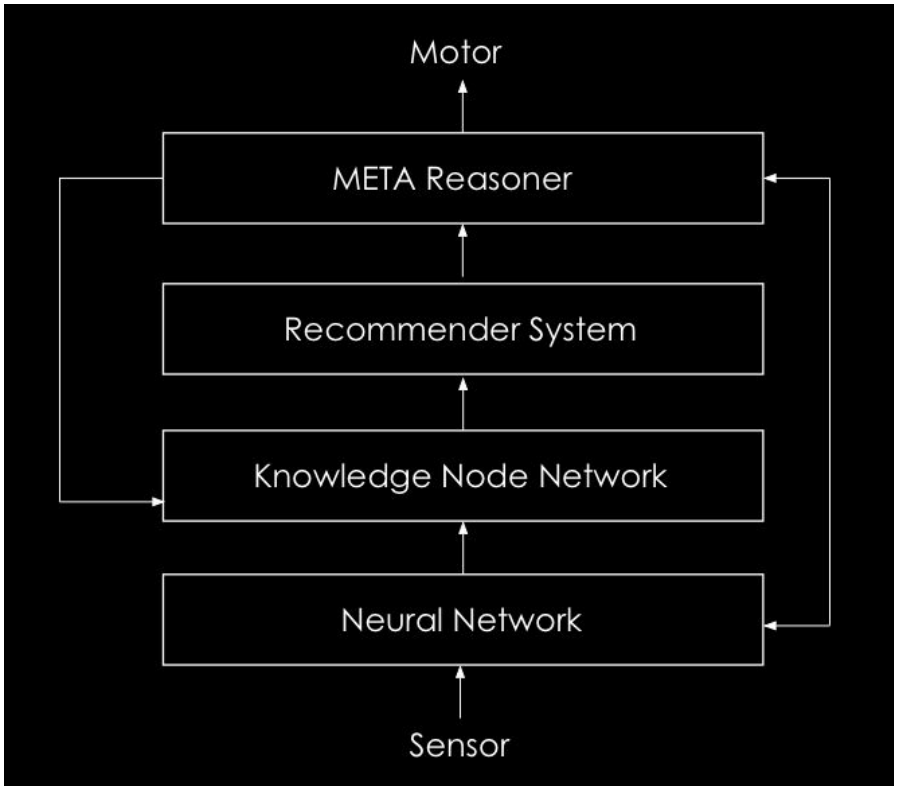

Figure 1: AI Model Diagram

The AI robot model begins with sensor input and ends with actuator output controlled by the our META reasoner. The raw data from the sensors is first processed by the Neural Network and Knowledge Node Network before entering the Recommendation System. The Recommendation System generates a recommendation using the processed data from the previous stages. The META reasoner then acts as the "brain" of the AI robots by determining if the recommendation from the Recommender System is acceptable based on state information. If the META is satisfied with the recommendation, it issues commands to the actuators. Otherwise, it sends a message back into the lower layers to produce a new recommendation or query for additional information. Overall, the META attempts to maintain a true world view and balance its goals while maximizing its resource efficiency.

Requirements and Constraints

High level requirements for META

1. It should take inputs from lower layer, and translate the inputs into a representation of the surroundings of the robot (a map or a localmap as we call it).
2. An efficient algorithm that allows the robot to explore an environment that it is unknown to the robot and construct a graph of localmaps for that environment. When the graph of localmaps has grown to a point that uses too much memory, the graph of localmaps would be saved into a

globalmap. And eventually represent the world with a graph of globalmaps. (This requirements was achieved during the first semester of DP)

3. If user sends a specific goal that would like to be achieved, the META would generate an efficient path while taking the distance it has to travel, the system status (namely the battery status) and the goal's priority into consideration.
4. When generating the path, higher priority goals should be achieved before lower priority ones, unless no extra distance will be travelled.
5. Robot should not run out of Battery before visiting all goals. If the robot estimates that it might run of batter soon, it should go to one of the battery stations to charge itself.
6. (Low priority requirement) Simulate the algorithm by controlling the simulator.

Constraints

1. Limited CPU power and memory.
2. The robot should consider the effects that its current system status has on its performance and make adjustments to its estimated path accordingly. However, when battery is low, the sensor inputs may not be as accurate and the motors may not perform as instructed.
3. Objects may be removed from their original position
4. Since the other layers of the AI model are still works in progress, we had to simulates the inputs and the outputs to the simulator.

Design

System Design

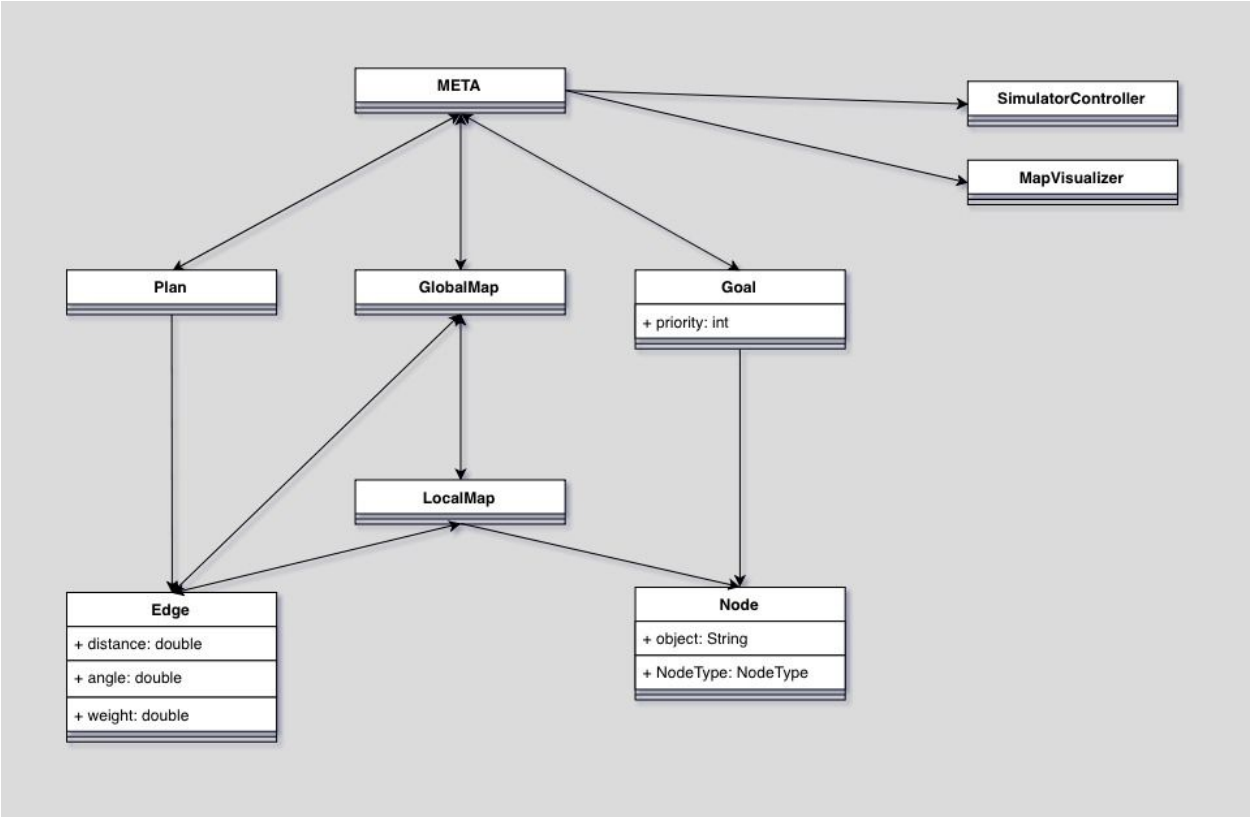

Figure 2: META UML Class Diagram

META takes input from lower layer, makes decisions among recommendations in order for the robot to efficiently complete the goals instructed by user. Since the other layers of the AI model are still works in progress, therefore, the META layer is still connecting with simulator for testing purpose. The map visualizer is used to visualize the constructed map. The objective for this semester is implementing path generation algorithm. Therefore, the map generated from last semester's algorithm will be used as an input. Compared to last semester, the map data structure switched from tree to graph. META takes constructed global map and a goal list as inputs, and then generates plan as output.

The data structure that is used to help the decision making and routing is presented in Figure 2.
The updated UML diagram of META-Reasoner system consists 8 main classes. The simulator controller class is used as an interface that allows META layer to communicate with simulator. The MapVisualizer class is used to help user visualize the graphic map. Edges and Nodes are the most fundamental unit of a map. Edge class connects two nodes and contains the detail information such as distance, angle and weight between start node and destination node. (The initialized weight is calculated based on distance,

the value will be adjusted based on the real world executing results and current system status while executing the plan). The start and destination location could be either local map or global map.
The LocalMap class has a zero to one association with Node class (each localmap contains at most one object) and an zero to many association with Edge class (each localmap may connects to zero or many other localmap nodes). The GlobalMap class is used to store multiple localmaps. It has a zero to many association with itself and a zero to many association with Edge class. Plan class is used to store the generated plan. It has a zero to many association with Edge class that is used to show the direction of path. The Goal class is a data structure to store input goals. It has an attribute "priority" that indicate the order for the specific goal to be achieved.

Data structure

In the first semester, our data structure is a tree of trees, where the inner tree represents a localmap and the outer tree represents the whole global map. Tree like data structure works perfectly in the exploring phase. However, it has poor performance when searching the shortest path. Because META frequently searches the shortest path in the path generating phase, we decided to revise the tree data structure into a graph data structure for better performance. This is the only change we made to the algorithm from the first semester.
To implement a graph data structure for path generation phase, we define our data structure with the following classes:

Node	A node stores a name string, and a type (Must be one of the following: "WALL, WAYPOINT, OBJECT, BATTERY")
LocalMap	A localmap stores exactly one node and a list of edges it connects.
GlobalMap	A globalmap stores exactly one localmap as its root, and a list of all edges in the globalmap.
Edge	An edge stores its starting localmap and destination localmap, and a distance between them.
Goal	A goal stores a node and its priority
Plan	A plan stores a list of edges representing a series of movement.

The data structure of META reasoner is a graph of nodes and edges. Each node can be a wall, an object, a waypoint, or a battery station. Each edge stores the distance and relative angle between two nodes. The information for each node and edge object is coming from Knowledge node layer, which processes sensor data into a prediction. And then META will transform this prediction into its own data structure.

An example of map constructing: Assume there is an object in front of the robot. After analyzing the object according to the shape and color, Knowledge node layer reports that the object has 70% likelihood to be a desk, 30 degree to the left of our current position, around 4.56 meters away. In this case, META will receive the prediction from Knowledge node layer in a form:

Node Type: Object
Probability: 70%
Node Name: Desk

This information is interpreted as: A desk is detected with a 70% of confidence, and it is an Object type. And information of the distance and angle of the detected object,

Distance: 4.56 meters
Degree: 30 degree

This information is interpreted as: The desk object is at 30 degrees to the current position, and it is *4.56* meters away.

Meta then generate :

1. A Node of type Object, name=desk
2. A Localmap of the desk Node, Localmap(desk)
3. (Optional) A Node of current position as a waypoint, if meta is not close enough to any existing Nodes.
4. (Optional) A Localmap of the waypoint (if a new waypoint Node is created in step 3), Localmap(current)
   (Optional) An Edge, start=last node traveled, destination=current position, distance=estimated from battery usage, angle=last robot turn record.
5. An Edge, start=Localmap(desk), destination=Localmap(current position), distance= *4.56*, angle=30

To sum up, whenever a new object is detected, META will generate 3 (or 6 if current position is not in the data structure) data instances to record the object`s relative position for later searching and pathing to use.

Algorithm

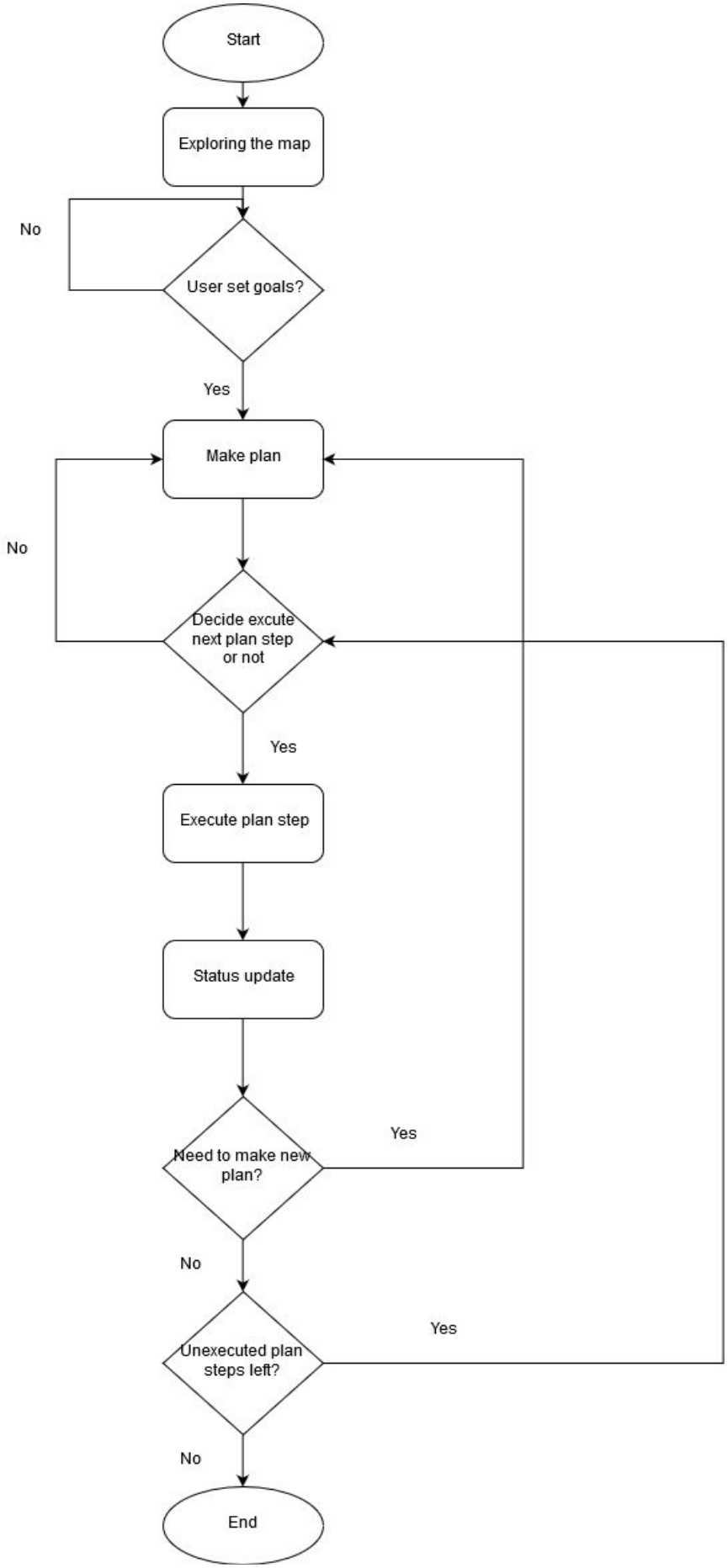
Start
Exploring the map
No
User set goals?
Yes
Make plan
No
Decide excute next plan step or not
Yes
Execute plan step
Status update
Need to make new plan?
Yes
No
Unexecuted plan steps left?
Yes
No
End

Figure 3: Algorithm Flowchart

Exploration phase

1. Exploring the map

The exploration algorithm is inherited from the first semester. Here is a brief description of the algorithm: The robot starts in the exploring phase. The robot aims to explore the undiscovered map as much as possible by always traveling to the furthest unvisited spot, spin around to find the next furthest spot, and travel to that spot and so on. Meanwhile, META will construct the graph data structure from the low layer inputs.

Path Generation Phase

The objective in this phase is to execute a plan that minimizes the distance to travel while making sure that all goals are visited in the order of their priority. And one important constraint is that the robot should decide when to charge its battery so it battery will not run out before visiting all goals.

2. User Set Goal

When the robot receives goals input in the form of (ObjectName, priority) from the user, the robot enters path generation phase.

3. Make Plan

3.1 Build complete goal graph

Find all the Localmap instances that contains the a goal in the graph data structure, and generates a graph namely “Goal Graph” of those goal nodes. Each edge between two goal nodes is a set of edges of the shortest distance between two nodes calculated from the original graph. The shortest distance is calculated by using a floyd warshall algorithm, which calculates the shortest distance between all pairs of nodes in the whole graph and their paths with a O(V^3) time complexity. We execute floyd warshall algorithm upon request of making a new plan, and store the all-pairs shortest distance to save subsequent calculations.

3.2 Build a Minimum spanning tree from the goal graph

Generate a minimum spanning tree from this goal graph. This is done by Kruskal’s Minimum Spanning Tree Algorithm with a time complexity of $O(E \log V)$ time, where E is the number of edges in the graph and V is the number of nodes. Now all the subsequent calculations are only scoped in this MST, which only has a very few edges and nodes comparing to the original graph.

3.3 Generate Plan

Build a plan from the minimum spanning tree, while enforcing the priority rule. We take the advantage that the edges in the minimum spanning tree are the shortest distance to connect all the goal nodes, we can traverse each edge(some edges are traversed more than once) to visit all goal nodes. We now only need to

decide the order of selecting edges. For each step: We generate a single step plan to the branch that contains the highest priority goal, and remove the last visiting node from the MST, regenerate edges connecting the next node and the disconnected nodes. We repeat this process until all goals are visited in our plan.

4. Decide to execute next plan step or not

Now we have a plan consisting of many steps, we execute the plan step by step. For each plan step, we determine if our battery can support us travel to the next destination node and from there to a nearest battery station. If yes, execute that step; if not, generate and execute a plan that redirects the robot to a closest battery station for charging.

5. Execute plan step

A plan step is a series of connected edges. The robot will travel the edge1.distance at edge1.angle to a next node, node2, and from there, travel edge2.distance at edge2.angle and so on.

6. Status update

After executing a plan step, META updates 4 informations:

1. Its battery status
2. Real distance traveled for each edge and update that edge's real distance
3. Missing objects(*Not yet implemented, left for future improvement)
4. MeanError. This value is used to estimate the difference between theoretical values(distance) and the real values.

7. Need to make new plan?

There are two cases that META has to make a new plan:

1. When the battery left cannot support the robot to travel to the next destination node and from there to a nearest battery station, or
2. There are objects (landmarks) missing. (* Not yet implemented, left for future improvement)

8. Unexecuted plan step left?

Jump to step 4 to execute unexecuted plan steps.

Now we will discuss how we designed our solution in order to fulfill the requirement under the given constraints. Since this project is a two-semester project, we already designed part of the solution, "Map exploration algorithm" to resolve some requirements, we have discussed these design briefly above. We name the new designs we introduced in the second semester as "Path Generation algorithm". This

algorithm is based on the output of map exploration algorithm and a list of goals to be achieved. We did some minor revision to the first semester`s data structure to better support the path generation algorithm.

Solution to Requirement

Requirement	Solution
1	Refer to Map exploration algorithm designed in the first semester. (Slightly revised for the second semester)
2	Refer to Map exploration algorithm designed in the first semester. (Slightly revised for the second semester)
3	Refer to Path Generation algorithm, step 3 and step 4. We decide to design a shortest path algorithm that best suits META's data structure for efficient pathing, and then apply constraints checks to make sure the pathing plan satisfies all constraints.
4	Refer to Path Generation algorithm, step 3.3. When generating each plan step from the minimum spanning tree, we do depth-first-search on goal priorities to find the branch with the highest priority sum.
5	Refer to Path Generation algorithm, step 4, step 6, and step 7. For every movement, battery consumption is predicted before the robot starts to move, so META will not execute plan steps that puts META in a position where no battery charging station can be reached. Also we performs a battery threshold check before META executes every plan step, in case the consumption is poorly estimated.
6	Requirement.6: We have not integrated mapVisualizer with META. This is left for future improvement.

Solution to Constraints

Constraints	Solution
1	We keep the data structure concise and easily accessible. We aim to simplify the data involved in all pathing computations as much as we can while keeping all the desired information (Refer to Path Generation algorithm, step 3.1 and step 3.2). We choose to run a O(v^3) all pairs shortest path algorithm in step 3.1 and store the result for

	subsequent use, rather than run O(v^2) two-nodes shortest path algorithm everytime we need a shortest distance.
2	This constraint is considered in both Map exploration algorithm and Path Generation algorithm. We store relative distance and angles to minimize the scope of every inaccurate sensor data. Information such as edge distance may be verified when traveling through that edge by averaging the real distance traveled and the stored theoretical distance.
3	We have not designed a solution to this constraint. This is left for future improvement.
4	This constraint mainly affects the testing process of our design. We have to enter the inputs from other layers to META manually when running unit tests.

Tools

In terms of tools, we used Java as the primary programming language to implement our data structure and algorithm. We used IntelliJ IDEA as an integrated development environment (IDE) for its code auto completion and static grammar checking features. For teamwork, we used VS code as a text editor with Live share feature to work on some components together, we also used git and GitHub for vision control, collaborating and issue tracking. We also used JUnit framework to write our unit testing for code quality assurance. We also integrated our code with the simulator provided by the simulator team. Since the simulator was implemented in Unity and C#, we had to create a class called “SimulatorController” to communicate with the simulator through a TCP connection details on this class is explained in the following section. In order to better visualize the graph-like map data structure we constructed, we created a tool called “MapVisualizer”. The details on “MapVisualizer” is also explained in the following section.

Other Classes

SimulatorController:
The simulator controller class is used to communicate with the simulator from our META layer. Simulator scripts are written in C# and Unity by the simulator team, but our AI model is implemented using Java. Therefore, a communication protocol is needed. A TCP connection is established on “localhost” on port 2222 and port 4444. Port 2222 is used by our META to send commands to simulator and port 4444 is used to receive data from simulator.

Map Visualizer:
The MapVisualizer class is used to help human reader visualize the graph-like map constructed by META. The MapVisualizer class contains two classes, namely an Edge class and a Node class. Node class corresponds to each node in our tree-map data structure and Edge class is used to represent the

relations between each parent node and its children nodes in the tree-map data structure. MapVisualizer class takes advantage of the Abstract Window Toolkit (Java AWT) and implements Runnable interface. MapVisualizer class utilizes the observer design pattern in which the MapVisualizer class is a separate thread running besides the META main thread and periodically checks if the tree-map is updated. If the map is updated, we will redraw all nodes on the screen. See figure 6 above for an example of treemap after exploration.

Result

After finishing implementation of the algorithm, a test case using unit test is generated to test the correctness of our algorithm.
The procedure is shown below:

1. Generate a global map with several local map in it, the map is constructed by using the data structure we designed in semester 1. The figure below is used to visualize the input map for a better understanding.

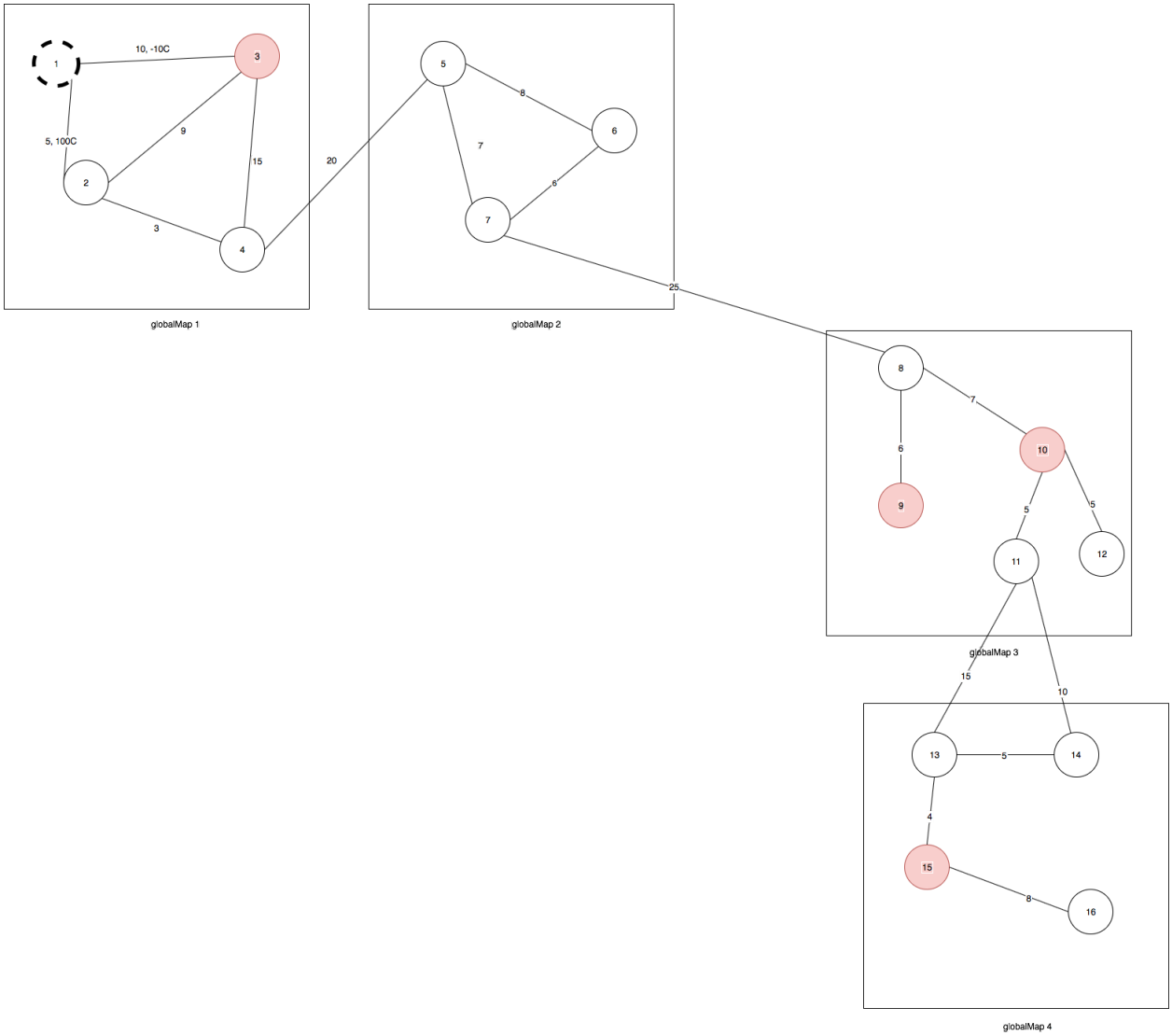

Figure 4: Map instance for Junit Test

2. The order of goal list according to the priority,

Node	Priority
Node 15	2
Node 10	1
Node 3	6
Node 9	4

Table 1: Goal Priority for Nodes

is Node 3 -> Node 9 -> Node 15 -> Node 10 (larger number means higher priority)

3. We have current robot status,

Location: Node 1

Battery Level: 10%

4. After execute the program with test input derived from step 1, 2 and 3, we obtained,

```
CurrentBattery= 10.0
CurrentBattery= 10.0
planlist left: 4
Battery low, go charging now
CurrentBattery= 10.0
Excute charge plan now, plan destination=o2
CurrentBattery= 9.153576190301193
CurrentBattery= 6.154874443242056
Charging complete.
CurrentBattery= 100.0
planlist left: 4
Excute plan now, plan destination=o9
CurrentBattery= 95.143228850653
CurrentBattery= 93.82034391356808
CurrentBattery= 85.62582930866236
CurrentBattery= 83.71780693370329
CurrentBattery= 70.06616504142339
CurrentBattery= 67.56326014168624
Goal : o9 visited
CurrentBattery= 67.56326014168624
planlist left: 3
Excute plan now, plan destination=o3
CurrentBattery= 64.6557305147896
CurrentBattery= 63.9258635679436
CurrentBattery= 59.25929206497747
CurrentBattery= 57.60655464892378
CurrentBattery= 56.1464016881963
CurrentBattery= 55.9431501694428
Goal : o3 visited
CurrentBattery= 55.9431501694428
planlist left: 2
Battery low, go charging now
CurrentBattery= 55.9431501694428
Excute charge plan now, plan destination=o2
CurrentBattery= 53.47417559190392
CurrentBattery= 52.01766199999702
Charging complete.
CurrentBattery= 100.0
planlist left: 2
Excute plan now, plan destination=o10
CurrentBattery= 97.3669525660097
CurrentBattery= 96.05033075604068
CurrentBattery= 92.75538358825857
CurrentBattery= 84.49966962127506
```

Figure 5: Test Case Output

At first, the robot only has 10% battery left. Therefore, instead of going to the first goal, it goes to the battery station. The robot then decides which goal node to visit by considering the combination factor of distance and priority. Moreover, the robot will continue to visit goal nodes until it finishes visiting all goals or runs out of battery.

Impact on Society and the environment

Since META reasoner is only a layer of the Prometheus Project, the impact of META reasoner on society and environment should not be determined individually. Therefore, the following section addresses the impact of the Prometheus Project.

a) Use of non-renewable resources:
   Most rescue and search robots need to be endurable[6]; therefore, current robot usually is fuel powered, which uses non-renewable resource. However, in the near future, the fuel can be replaced by solar energy, wind energy and biological energy etc.
b) Environmental benefits:
   With environment exploration function, the AI model can be used to build agricultural robot, which can help to improve environment. For robot built in other purpose, the benefit to environment is using green materials that reduce pollution as much as possible.
c) Safety and risk:
   Artificial intelligence is designed to learn from interactions with its surroundings and alter behavior accordingly, which brings a lot of benefits to human. Since AI can address problems more effectively than human-being, it has huge potential to improve life for everyone. However, this technology could go awry by illegal application. Also, an AI misunderstanding its surrounding might lead to deadly action as a result.[1]
d) Benefits to society:
   The robot can save a lot of human resources. In current society, most rescue operations mainly depend on humans. However, humans have limitations comparing to robot. First, it is low-efficient for human to do rescue during night time. Secondly, human has speed limitation, they cannot search as fast as robot. Thirdly, it is dangerous for human to search in unknown environment; the unfavorable condition might be weather, wild beasts, or lackness of air. Lastly, there is no terrain restriction for robot. It can either dive into water or fly into sky.

Report on Teamwork

During this semester, all the teammates are active and responsive to resolve problems. Team also meet with project supervisor frequently .

[1] Tesla car accident, in which the vehicle mistook a white truck for a clear sky.

We followed a traditional waterfall software development process, which goes through requirement analysis, design, implementation, and test stages. We spent some time to thoroughly understand what META layer is used for and how to build the data structure effectively, and discuss and improve our design. We did not implement any java code until we feel confident about our design.
Therefore, most of the coding work was done in second half of the semester.

Zisheng Wang	Read material to understand the META layer, Design algorithm in pseudo code, transfer algorithm to Java, Write report.
Qian Li	Read material to understand the META layer, Design algorithm in pseudo code, transfer algorithm to Java, Write report.
Jiachen Ju	Read material to understand the META layer, Design algorithm in pseudo code, transfer algorithm to Java, Write report.

Conclusion

In design project, AI model decomposes how human behaves into layers; each layer deal with different aspect of data and this requires the communication between layers. The team has explored and studied different algorithms of exploring map from a brute-force algorithm to an algorithm that behaves more human-like. At the end of this semester, the robot is able to explore local map, build tree map using detected landmarks with accurate sensor data and generate plan for received goal list. The things we learned throughout the course of our design project include the following:

1. We greatly improved the efficiency of our algorithm; for example, we simplified the map to become more goal-oriented in order to save memory while generating a plan.
2. The design project is a good opportunity to apply knowledge we learnt from our courses, such as finding the minimum spanning tree for a weighted graph
3. Consolidate our understanding for different data structure, such as tree and graph

If we have more time, we would focusing on visualize the generated plan by connecting to robot's motor and integrate the META layer with other three layers indicates in Figure 1.

References

[1]J. Vybihal, "Full AI Model", 2017. [Online]. Available: https://docs.google.com/document/d/1FBEm6MZ8kCdUFxLMYMHy_w977qjolDdEL_Ux_5dnE_g/edit?usp=sharing. [Accessed: 4- Dec- 2018].

[2]J. Vybihal, "META Reasoning", 2017. [Online]. Available: https://docs.google.com/document/d/1IBExD3UBCMwKmmk6EmLn_2X7J6CK9A9_bCIV2e7tPQ8/edit?usp=sharing. [Accessed: 4- Dec- 2018].

[3] *Dr.John.B.Matthews*, “GraphPanel”, 2012. *GraphPanel - Dr.John.B.Matthews*. [online] Available at: https://sites.google.com/site/drjohnbmatthews/graphpanel [Accessed: 4- Dec- 2018].

[4]J. Vybihal, 2017. [Online]. Available: https://docs.google.com/document/d/1nnQdnvgzcjZJexT5fi54CmH7acatw3QdQoXzFrpGInc/edit?usp=sharing. [Accessed: 4- Dec- 2018].

[5]J. Vybihal, "META Problem Statement", 2018. [Online]. Available: http://docs.google.com/document/d/10b4A7i3pO1k0Y8p4qQStD6NnWfY34vn_VzQN1hNtJmU/edit?ts=5a7cc010. [Accessed: 4- Dec- 2018].

[6]W. Yang and Y. Zhang, "Applications of renewable energy for robots", *World Automation Congress (WAC)*, 2012.