Software design
Team project – Deliverable 1

Team number: 2
Team members:

Name	Student Nr.	E-mail
Ruben van der Ham	2592271	2592271@student.vu.nl
Harsh Khandelwal	2593809	2593809@student.vu.nl
Rico Mossinkoff	2589295	rmf700@student.vu.nl
Ewoud Vermeij	2595391	2595391@student.vu.nl
Lucas Faijdherbe	2594812	2594812@student.vu.nl

Table of Contents

1. Introduction

Author(s): Ewoud Vermeij, Rico Mossinkoff, Lucas Fajidherbe, Ruben van der Ham

In this introduction we will give a short description about the implementation of our ROVU system. In ROVU, robots have the mission to cover as much as possible of the area they reside in. With cover we mean that if we divide the area to be monitored in a grid of points, the robots need to make sure that of each point one or more photos are taken. We will define the key aspects of our ROVU system that makes it possible to achieve this goal.

Operator
A mission in ROVU does not start by it self. It is initiated by an operator. Before a ROVU mission is started, the operator must define some features of the environment where the ROVU system will be deployed. Here we should think of the possibility to choose for outer walls in the environment and the presence of obstacles. Also picking the type of obstacles should be possible. However, there will be a limit on the number of obstacles in an environment. It should not be possible to put hundreds of objects in the environment. When the mission is started, the central station takes over control. This means the operator can only monitor the ROVU system doing its job and is not able to interfere with the system. We decided to make our system completely autonomous, because the objective is to create a system with robots that cover as much as possible. If an operator can interfere with system, it's not just the system who does all the work. The system will stop when the mission is complete.

Environment & System
Since environments can be different every time, we decided to make the environment unknown to our ROVU system. With an unknown environment, we need proper obstacle avoidance strategies for our robots and a way to map the environment when obstacles are found. To achieve this, our ROVU system consists of at least one and at most 4 robots and one central station. The limit of 4 robots has to do with our strategy. A short description of our strategy:

> *At the start of the mission each robot will be placed in one of the corners in parallel with the wall it has on its left. When the robots start moving, they will move straight forward. If an obstacle or a visited gridpoint is found it will turn right and explores further. In this way the robot(s) explore the area in a spiral. The central station will intervene when 2 or more robots are about to collide and also keeps track of visited points and obstacle. Robots take pictures at every grid point so that we are sure all grid points will be covered.*

Robots

There is only one type of robot. Each is equipped with sonar sensors to detect obstacles and a camera to make pictures of grid points it passes. The behavior of the robot will be as follows. A robot will always move straight forward unless an object or visited grid point is detected in front of the robot. When that happens, the robot will turn right until it can drive straight in parallel with the detected object or grid point(for example, finding a wall and following it). When a passageway on the left is found, the robot will turn left and move forward again [1]. When a wall is found the robot will turn right again and looks for a passageway on the left. Because each robot will explore from the outside of the environment to the inside, getting stuck in a cycle will not happen easily. However, when 2 or more robots are about to collide or get stuck somehow, the central station will intervene in the behavior of the robot. Beside sensing obstacles and making pictures, a robot is capable of sending and receiving messages of the central station. A robot sends it location every time a grid point is visited and can get specific orders from the central station which it must obey. With robots unable to disobey orders, the central station has full control over the mission.

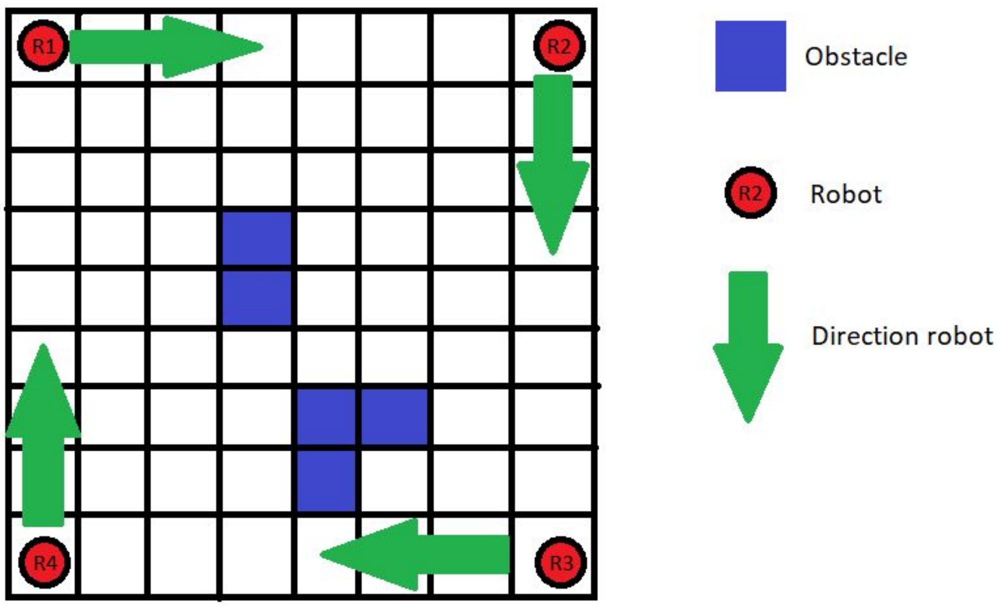
Central station

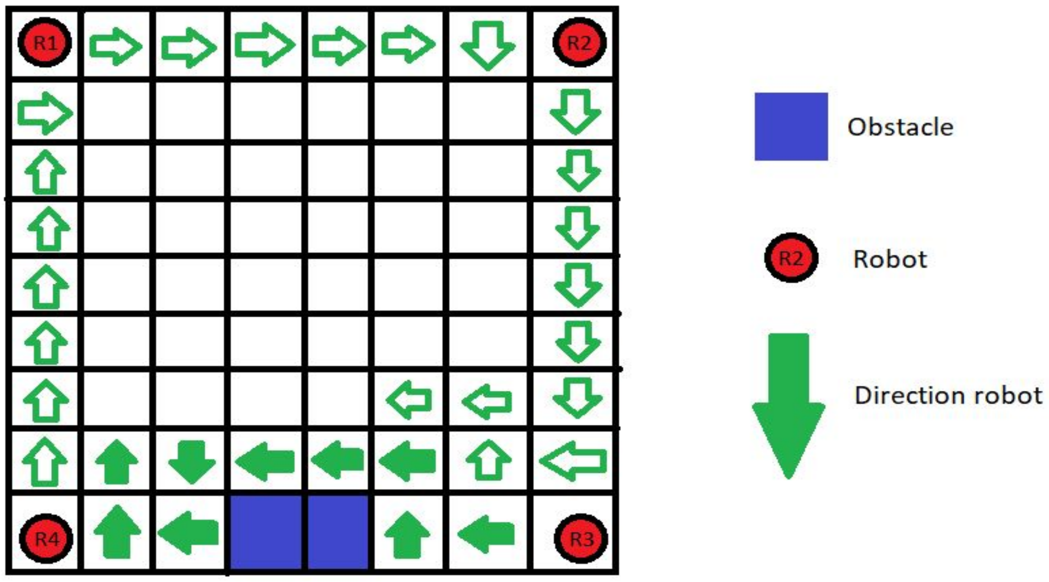
The central station can be seen as an all-seeing eye watching over the robots. The central station sees the environment as a grid of points that must be covered by one of the robots. Each grid point is assigned a certain status. This way the central station has a detailed view over the area and is more capable of making proper decision to cover the area as much as possible when guiding the robots. The statuses are:

1. Unknown. Zero knowledge of this grid point.
2. Accessible. This means that a sonar sensor of a robot has covered this grid point and did not find any obstacle.
3. Obstacle. This means a grid point is identified as an obstacle.
4. Covered. This is a grid point covered and photographed by a robot
5. Inaccessible. This means a certain grid point is inaccessible. For example, when an area is surrounded by walls.

In our system, covered means that a robot has been on that grid point physically. However, for our project, covered means that a picture has been taken of the grid point. With our strategy, letting the robots visit all grid points in a spiral, we are confident that at least 70% of the coverable area will be covered. This is because the grid points on the photos will be photographed multiple times. The central station can also give orders to the robots. This can be for changing the direction or making it stop for a specific amount of time. Another order could be directing a robot to a specific grid point that has not been covered yet. Also, it is the central stations job to guide a robot in a different direction when 2 or more robots are about to collide. Orders to guide robots usually only happens when they get stuck in a situation or when some points are still not covered. The idea behind the central station is that 1 entity keeps track of the progress of the mission and not all robots. In this way it is easier to guide the robots as a boss instead of having robots communicating with each other for decision making. You could see the mission this way in 2 phases. In phase one the robots will mostly move autonomous, mapping the area. The second phase robots have mapped most of the environment and the central station will guide the robots to areas which have not been identified yet. We handle a at least 70% coverage policy to define the mission as complete. When the only statuses of grid points consist of obstacle or covered, the mission is complete and the central station will order the robots to get back to their starting position.

Below the introduction we have a visual example of our strategy.





2. Requirements Specification

Author(s): Ewoud Vermeij, Rico Mossinkoff Lucas Fajidherbe, Ruben van der Ham

2.1 Requirements

Functional requirements

#	Short Name	Description
F1	Environment mapping	The rovers shall map every point in the environment, determining if a point is free, or if there is a static object on the point.
F2	Obstacle avoidance and identifying	When running into an object, a rover shall move around it and identify it by following the “wall” of the obstacle.
F3	Grid point avoidance	The robots shall avoid visited grid points and handle them as special objects.
F4	Grid point mapping	The rovers shall take a photo of every grid point encountered, and send it to central station. A grid point shall then be marked as ‘visited’ by the central station.
F5	Communication	Rovers shall send information about obstacles, visited grid points and positioning to central station. Central station shall send this information to the other rovers.
F6	Environment covering	The rovers shall cover every accessible grid point in the environment.
F7	Order following	A robot, even though it is working autonomously, shall always follow orders given by the central station. These orders have highest priority.
F8	Rover navigation	A rover shall move straightforward, unless an obstacle is detected. If there is an obstacle, the robot shall try to move right, and otherwise try to move left.
F9	Mission completion	If all grid points are covered, the mission is complete. The rovers shall return to the base, and stand-by.

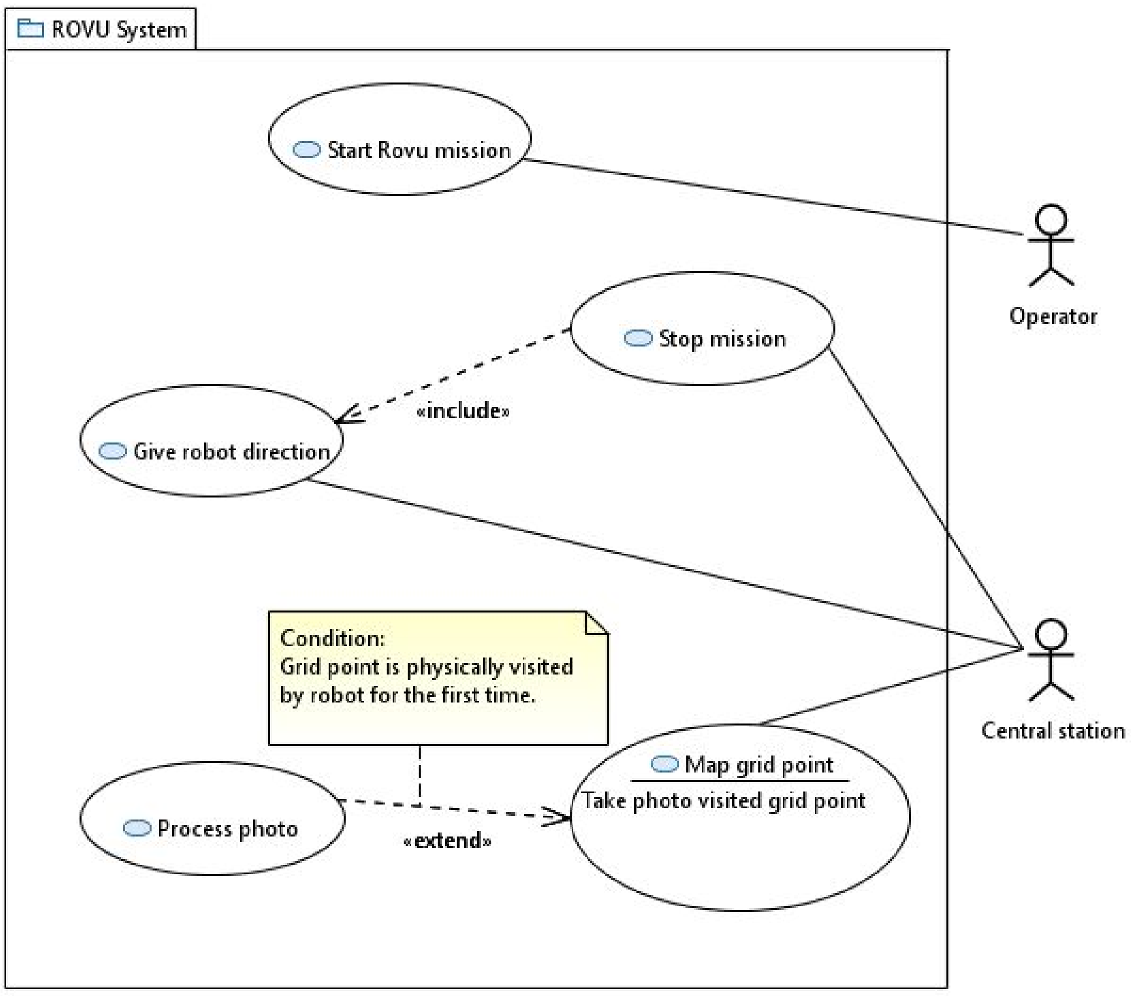
Non-functional requirements

#	Short Name	Description & reasoning
NF1	Robot - CS communication	[PERFORMANCE] A robot shall report his location to the central station every time it has reached a new grid point. This makes the central station more capable of mapping the environment in a detailed and clear way.
NF2	Photo handling	[SPACE] Central station receives photos from the visited grid points, and shall store these. The stored photos should be ordered, and at the end of the mission, should be retrievable/downloadable as a whole. Making it downloadable is necessary for checking the pictures after the mission.
NF3	Response time	[PERFORMANCE] The response time of a robot for changing direction to avoid obstacles or order following should be less than 0.1 seconds. If the robot has a slower response time, obstacle avoidance and environment mapping could be less precise and endanger the mission.
NF4	Initiating mission	[OPERATIONAL] An operator without technical knowledge should be able to initiate and define a mission in ROVU. This makes the system more accessible and no technical knowledge is needed to start a specific mission.
NF5	Following orders	[DEPENDABILITY] A robot shall not be able to ignore orders of the central station. This gives full control to the central station which keeps track of the mission. If robots could ignore orders the mission would be less likely to be completed.
NF6	Programming language, environment and API	[DEVELOPMENT] For the development of the ROVU system the programming language Java shall be used. The simbad API and Eclipse shall be used as library and environment for development. This is necessary for the project.
NF7	Robot efficiency	[EFFICIENCY] The robots should visit each grid point for the least amount as possible. This makes the system more efficient, decreasing the necessary time needed to complete the mission..

2.2 Use Cases

Author(s): Ewoud Vermeij, Rico Mossinkoff, Lucas Fajidherbe, Ruben van der Ham

2.2.1 Use Case Diagram



2.2.2 Use Case Descriptions

Name	Start ROVU mission
Short description	The operator starts the ROVU mission.
Precondition	Operator has set the features for the to be monitored environment where the robots will perform on their mission
Postcondition	Phase 1 of the mission has started
Error situations	The operator has given invalid features for the environment
System state in the event of an error	The mission does not start.
Actors	Operator
Trigger	Operator wants to monitor a ROVU mission
Standard process	1. Operator chooses for an outer wall or not. 2. Operator chooses number of boxes in environment. 3. Operator chooses number of arches in environment. 4. Operator confirms chosen features.
Alternative processes	(2’) Number of boxes passes maximum amount. (3’) Number of arches passes maximum amount.

Name	Stop mission
Short description	The mission is stopped when the mission goal has been reached. The central station sends an order to the robots to return to the central station
Precondition	At least 70% of the grid points has been covered
Postcondition	The mission has been stopped and the robots are back in their starting position
Error situations	Robots get stuck in certain position not being able to return to the starting position.
System state in the event of an error	Central station is required to give robot a specific order to get out of a situation where it is stuck.
Actors	Central station

Trigger	70% of area is covered
Standard process	1. Central station calculated percentage of covered area to be higher than 80% 2. Central station orders robots to return to starting position. 3. Robots reach starting position. 4. Mission is ended.
Alternative processes	(3’) Robot gets stuck in certain position. (4’) Central station gives direction to robot. (5’) Mission is ended.

Name	Give robot order
Short description	Give the robot an order to do something.
Precondition	Danger of collision, uncovered grid point in phase 1, robot is in a loop or stuck
Postcondition	Robot has been given an order to: - Change direction - Go to a specific grid point - Exit a loop - Stop mission
Error situations	Order cannot be followed
System state in the event of an error	Robot stops moving
Actors	Central station
Trigger	Danger of collision, uncovered grid point in phase 1, robot is in a loop or stuck
Standard process	1. Central station checks the current state of the environment and the robots 2. Central station chooses which order he wants to give 3. Robots perform the given commands
Alternative processes	(3’) Robots cannot perform command (4’) Robots stop moving.

Name	Process photo
Short description	Central station receives photos from a rover visiting an unknown grid point, and stores these.
Precondition	Grid point is not physically visited yet by the rover
Postcondition	Photo has been taken by the rover and the point is marked as visited by central station.
Error situations	Rover camera not functioning
System state in the event of an error	Robot returns to home station
Actors	Central station
Trigger	Photo has been taken by rover
Standard process	1. Central station receives the photo 2. Photo is saved by central station
Alternative processes	(1’) No photo has been received (2’) Robot is called back to home station

Name	Map grid point
Short description	Central station maps a coordinate grid point (and its surrounding coordinate grid points) as accessible, inaccessible, obstacle or covered.
Precondition	Robot sent its current coordinate grid point to the central station
Postcondition	Central station has mapped the coordinate grid point to a specific status.
Error situations	Grid point is already mapped as visited
System state in the event of an error	Central station does nothing
Actors	Central station
Trigger	Robot sent its current coordinate grid point
Standard process	1. Central station receives a coordinate grid point (and its surrounding grid points)

	2. Central station maps the grid point (and its surrounding grid points) to a specific status: - covered - obstacle - accessible - inaccessible
Alternative processes	(2’) Central station does not map the grid point as its already mapped (2’) If a surrounding grid point is already mapped as visited, it won’t change that grid point.

3. Implementation remarks

Author(s): Harsh Khandelwal

As required for, we have kept the initial implementation of our system minimalistic. An environment enclosed with walls and a few boxes as obstacles was our model to run the robots upon. We found a way to make the robots move randomly, and also a way to move deterministically without collisions at all. Further, we decided on a way to map the environment and for the robots to find their coordinates.

We reengineered the example platform(environment, robot and main) provided and built our system upon that. The code mentioned in [2] offered us insights on how to use 12 sonar sensors instead of 4 to get measurements from all 4 quadrants and avoid obstacles. By doing this, we could move the robot deterministically, avoiding collisions at all. Further, every 10th virtual second, we would make the robot make a random move in order to not get stuck in loops. The methods moveDeterministic() and moveRandom() in Robot class achieve the said things.

The performBehaviour() in Robot calls all the method and additionally, every 10th second with making the random move, it also prints the coordinates of the robot. The z value of the coordinate was nullified to create a 2d mapping. The coordinates are rounded to nearest 0.05 value for simpler mapping.

The environment is mapped in a Hashtable with Vector3d objects as key and String as value. In this way, every coordinate can be assigned one of the 5 states (unknown, inaccessible, accessible, obstacle, covered). The getAreaInCoordinate() method in Environment does this by initializing every coordinate with a step of 0.05 to “unknown”.

In future, we aim to retrieve coordinates of the robots from their objects in the Main class and update the values of the worldMap Hashtable object. This way, we will have a listed representation the map. Moreover, we will try to get images captured by robots and store them in a data structure in Main. We will still need to look into a way for the Main to direct robots to a certain coordinate and also an algorithm to determine landlocked regions of the map.

4. References

References here.
[1] https://www.ibm.com/developerworks/library/j-robots/
[2] https://github.com/jimmikaelkael/simbad/blob/master/src/contribs/bench/Benchmark.java