Smart Rescue Robot Team GEPBot+Exemplary

Nnamdi Eljah Obi
Department of Electronics Engineering Hamm-Lippstadt University of Applied Science
Lippstadt,Germany
nnamdi-elijah.obi@stud.hshl.de

Asm Nurussafa
Department of Electronics Engineering Hamm-Lippstadt University of Applied Science
Lippstadt,Germany)
asm.nurussafa@stud.hshl.de

Izuchuwu George Enekwa
Department of Electronics Engineering Hamm-Lippstadt University of Applied Science
Lippstadt,Germany)
lIzuchukwu-george.enekwa@stud.hshl.de

Patrick Nonki
Department of Electronics Engineering Hamm-Lippstadt University of Applied Science
Lippstadt,Germany)
patrick.nonki@stud.hshl.de

***Abstract*— With this documentation we aim to explain the reasons and rationale we took in approaching the tasks given towards the Modelling, Design and Code implementation for our rescue robot. With the aid of Constraint diagrams; block diagrams; used case and sequence diagrams (just to name a few), we were able to carry out detailed analysis as regards different functionalities for our robot. SolidWorks was used during the Design phase for the robot, with which STEP files for different parts of the robot were developed to enable 3-D printing of these parts. In the end, the results achieved were in accordance with the requirement specifications as given by the clients (in this case our professors).**

I. INTRODUCTION

There has been a steady rise in the number of recorded natural and manmade disasters happening around us these past few decades. As the impacts of catastrophes are increasing, the necessity of rescue robotics, which explores solutions to minimize the casualties for all phases of a disaster, has increased as well. As human lives need not to be exposed to unnecessary hazardous conditions, there has been an increase in rescue robot Research and Development, as rescue robots are mainly expected to play influential roles in urban search and rescue (USAR), nuclear field emergency operations, and mine rescue missions. There are instances where the robot might be required to drive through water terrains like in a case where earthquake busts open the underground water channel or in a mine where there's a landslide. A rescue robot needs to demonstrate mobility, dexterity, reconnaissance and exploration capabilities to efficaciously accomplish an USAR mission. These are key capabilities that empowers a rescue robot to efficiently overcome obstacles, remove rubbles, explore and map unknown environments and detect victims. Accordingly, we designed our robot in a waterproof and dust proof casing. The aforementioned functionalities are what we were tasked to embed in a single system. Doing this posed for us various critical challenges, but this was what we turned into our primary motivation for this work.

With the aid of properly sized tires our robot would be able to move through it's given path whether land or water (in this instance, one with a maximum height of 3 meter). Also, it comprises of different sensors for detecting heat signatures, object identification and obstacle avoidance. Lastly, Cameras were also mounted at the upper body part of the robot to enable easy manual remote navigation.

Our robot seeks to recover data through the use of digitalized camera, locate assigned tasks like small object recover, mapping out safest possible path for victim recovery and delivery of essential materials like bottle of water, medicines to victims in time-critical conditions before the human safety personnel arrives the location. It carries out risk assessment of its mapped environment by comparing the current map with the map of its original state before the accident. Using the sensors and camera, rescue personnel can remotely ascertain how critical is the rescue victim condition. Our robot is designed small in order to enable it enter small spaces and pass-through holes that are impossible for human rescuers to enter. This increases the chance to save more victims in a record time than the time it takes human rescuers to break through such obstacles.

The aim of this study is to design and develop a small-scale search and rescue robot that can assists in the recovery and exploration of dangerous areas, which should include both land and water. There are many factors that affect the performance of the model design should be considered. The parameters namely: type of materials used, electronic component, types of control application it uses are further discussed below.

II. REQUIREMENT ELICITATION

At the start of our rescue robot planning, we first of all had to make a requirement elicitation for our robot.

The requirement elicitations are respectively;

a) Establishing objective
b) Organize knowledge
c) Collect requirements

a. ESTABLISHING OBJECTIVES

Business Goals: To rescue human lives and retrieve sensitive data from high-risk disaster areas.

Problems to be solved: Being able to enter into high-risk situations, save lives and retrieve sensitive data, where would be too risky for human rescues to reach.

Systems constraints: Rescues that assist in the recovery and exploration of dangerous areas. Environment also consist of water; hence the rescue robot must be able to drive on water and withstand intense heat pressure.

b. ORGANIZE KNOWLEDGE

Stakeholders' identification: Prof Henkler, Prof Kristian, George, Elijah, Patrick, Nurusaffa and Tasawar.

Goal prioritizing: Ability to rescue human lives from high-risk disasters, ability to retrieve sensitive data for further investigation.

Domain knowledge filtering: Human rescue operation, high temperatures. Cold 3m deep water area, risk for loss of sensitive report data, need to locate and track injured people safely, navigate through obstacles.

c. COLLECT REQUIREMENTS

Stakeholders' requirement: Development of a rescue robot that assists in the recovery and exploration of dangerous areas, the robot should be able to move on land and on water and avoid obstacles. Hence, its environment consists mostly of water.

Domain requirements: Robots must be made with high heat resistive materials, with an ability to withstand high amount of heat, should also be made of waterproof materials and tightly sealed.

Organization requirements: To detect people in danger and send an alert to a rescue crew.

III. MODELING

A. OVERVIEW

This part is intended to describe the main behavior of our system using the System modelling language via diagrams (Use case diagram, Block and internal block diagrams, Requirement's diagram, Activity diagrams, Sequence diagram)

1) Use case Diagram

The Use case diagram shows the different interactions between the user and the interface of the system throughout the different tasks. The diagram is an overview of all the different use-cases linked together and also with the user outside the system. The different use-cases we could define for our system are: detecting obstacle, manipulating objects, moving back and forward, in water driving, scanning area, live broadcasting etc. For the use-case diagrams, we subdivided our work in 3 scenarios:

- The robot use case autonomous here considering our robot acting autonomously
- The robot uses case manual here
- The use case for data retrieval of the environment here

2) Requirements Capture and Diagram

This section lists all the requirements of our robot and their relationship in terms of derivation, containment or refinement. Those applying for our system are:

Human detection, Obstacle detection, Heat resistance, Environment analysis, Detachment Obstacle and water avoidance, Data retrieval, "Move person", Robot walk and Robot float.

Thus the requirements elicitation and validation is presented here and the requirement diagram that applies to all those requirements is presented here.

3) Block and Internal block diagrams

This section lists all the different blocks (parts or functional elements) of our system (robot) and their relationships listed in block definition diagrams and the relationship in terms of energy flow listed in the internal block diagrams.
Considering our system, we assumed the following blocks: Robot, power supply, motor, wheels and sensors.
And following the different scenarios that we have (Movement, floating and rescuing), we created different bdd and ibd accordingly here.

4) Activity Diagrams

This section describes the different actions operated within our system from the start point till the end taking into account decision taking. As for the previous section, it has been divided in 2 scenarios for the diagrams:
- The "float on water" activity here
- The "Rescuing" activity here

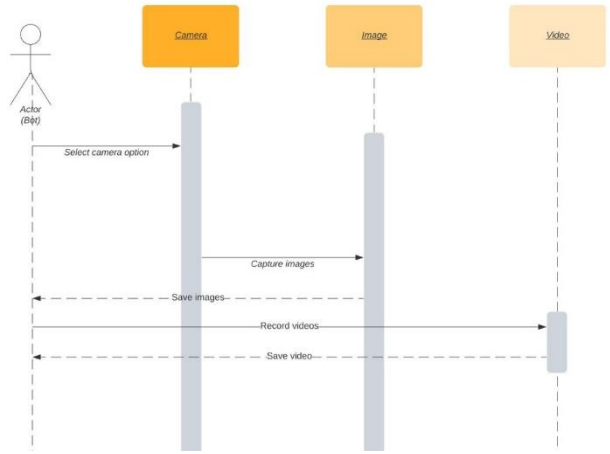
5) Sequence Diagram

This section describes the behavior of our robot in terms of actions operated, considering the objects acting (in terms of action-reaction) in a chronological way.
Like the previous diagrams, we considered 3 different diagrams for the scenarios:

- The “float on water” sequence diagram here
- The rescuing sequence diagram here
- The data retrieval sequence diagram

After the requirement elicitations, we assigned tasks for the system engineering. I worked on the data retrieval part of our robot. For this our robot will have an in-built camera which will be able to capture images and record videos of the rescue premises, these images and videos will then be used for any sort of future investigations.
Below is a sequence diagram of the data retrieval process;

Fig. 1. sequence diagram



The data retrieval mode comprises two (2) main systems: Capture mode and recording mode as shown in the system Sequence diagram in Fig. 1. Fig. 1 consists of three blocks, each containing several components. The robot connects to its system's camera and also controls the switching purpose.

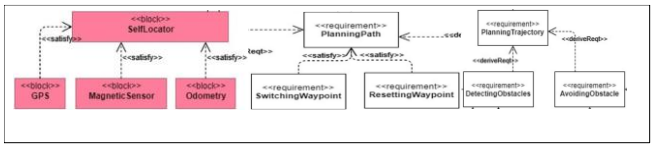
B. MODELLING ASPECTS OF OUR ROBOT

In order to achieve the purpose of implementing a rescue robot, it is very important to adopt a model driven architecture at the beginning. This would help develop the initial ideas independently later on, making it possible to be changed with- out actually implementing the software. Modelling the system would help describe the behavior and internal structure of the system more thoroughly. Therefore, for the purpose of this paper, we have adopted the SysML (System Modelling Language) into standardizing the modelling the approach. The SysML would help us portray the specification, analysis, design, verification and validation of our complex smart rescue robot system and therefore, all of the modelling diagrams are made of SysML constructs. This would help us develop reusable and versatile robot model.
For the modelling part of this paper, we have divided the sections into different scenarios of the robot system which come together as a whole model when combined. To elaborate each of the scenarios, we will be using the requirements diagram, block definition and internal block definition diagrams and to describe the behavioral aspects, we will be using activity and or, sequence diagrams.

1) Context Analysis

We begin by analyzing our robot system in environments. Our robot operates in an outdoor surrounding where the surfaces could be of the type earth, water, concrete, grass etc. The robot must be able to avoid the obstacles, both static and moving obstacles, and make a correct and shortest path in various situations. The full figure of the context analysis (Annex A) shows an example of the context diagram that includes expecting objects in the surroundings and the disturbances in our robot operating environments. Disturbance elements (e.g., physical objects) are considered as the possible distraction sources that may affect the robot’s sensing ability. The human controller is someone who interacts with our robot system. Categorizing the elements in such a way would enable us to derive a requirements diagram for our robot

.



(a) Locating Self and target (b) Path Planning (c) Planning trajectory
Fig. 2: Main parts of the Requirements diagram

Next, we dive into the different individual scenarios of our robot system.

2) Movement

We simplify the movement by limiting our locomotion in the 2D space, i.e., in the vertical direction and in the horizontal direction. Our robot should be able to operate in manual and in the automation modes. In this paper, we mainly focus on automating our process of movement. So, abstracting the requirements, it should be able to locate itself and the target, plan its movement and its trajectory by avoiding obstacles, be able to move in water and so on. This information can be visualized in the snippets from the following Fig. 1. The full requirements diagram of this scenario is seen in Annex B.
So, from this derived requirement diagram, we can now be more specific about the hardware requirements. To do this, we can create a block definition diagram to show the relationship between different components of the system and also be more concrete about the constraints and functions of each of the components. For this arrangement, we have used the Hierarchical Containment Architecture. A snippet of the robot block can be seen in Fig. 2(a). To be able to locate itself and the target, we can use GPS, magnetic sensor and odometry data. And to be able to plan its path and trajectory, we should use bumper sensors, laser sensors and trajectory tracer. Together with all these data, the option of generating a map becomes available. The functions of all these are placed in the main robot system block. Crucial constraints, like the elevation (¡100 mm) , no. of steps (¡ 200) and energy limitations are also seen here. The full block diagram can be found in Annex C.

Chronologically, using an internal block diagram (IBD), we can completely visualize how the interface works among all the components of different blocks.

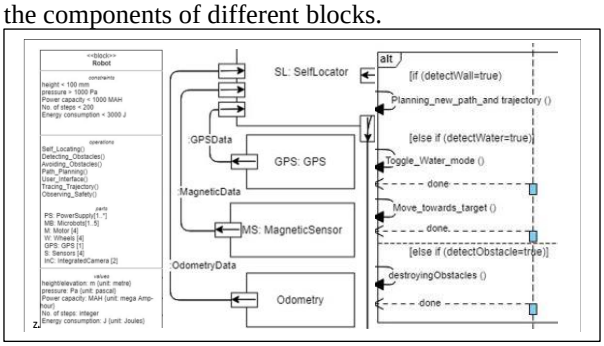


Fig. 2: Snippets from BDD, IBD and Sequence diagram.

A snippet of how the self-locating interface would look, from the IBD diagram is shown in Fig. 2(b). The full internal block definition diagram can be found in Annex D

Next, to explain how this would behave we can transfer all of this information into a sequence diagram. So, initially the robot should be able to locate itself and the target. Next, it plans its path and trajectory and advances towards the target. If during its course it finds a wall, it changes and re-plans its path. Or, if it discovers an obstacle, it should try to destroy it. When it identifies water surface Infront, it toggles to a water-driving mode. This can be showed from a snippet from the sequence diagram in Fig. 3. After it reaches the target successfully, it brings it back home from where it started. The full sequence diagram can be found in Annex E.

IV. DESIGN

This part is intended to define the physical prototyping of our project "Rescue bot" and thus the 3D design, taking into account the different interactions with the environment and its reactions. Before any real technical inputs were made as regards to the design of the robot, adequate level of research was first carried out. Sketches were made for different scenarios of movement. Also, a slide to properly present the idea was also made. In the course of the research, we explored different methods that could enable our robot to meet the given requirements, one of which is to retrieve items. Based on the dimensional restriction given for our robot, the possibility for a small sized robot to pick up human sized objects was not possible hence we had to readjust and eliminate certain functionalities out of our system.

It contains four different parts as follow: Research, analysis, design, and model.

A. Research

a) Movement research

With the aid of technology, we can effectively reduce the risk for loss of human life. Thus, it becomes important to think about designing alternative solutions in case of non-accessible areas by humans for rescuing. During research, we had to put into consideration the carrying capacity and movement speed of the robot both on land and in water. Some techniques we researched about were as follows;

Legged robot

Wheeled robot

Hybrid (combination of wheeled and legged)

It was quite clear that legged robots had a lot of benefits, namely; the capability to overcome and maneuver obstacles easily is one important advantage, while legged robots were too slow. Most legged robots also had limited carrying capacity. Therefore, to easily achieve the movement speed we desire on land, we had to look at the wheeled robot.

Also as was with the legged robot, the wheeled robot would not be able to overcome obstacles easily but its most important advantage was the movement speed and carrying capacity, which is a requirement for our robot.

The Hybrid (wheeled and legged) has both features, its flexibility is excellent but then its carrying capacity might be affected.

Putting our requirements into consideration, we decided to use a wheeled robot. With the help of obstacle detection sensors, our robot can easily map out drive areas with shorter routes, we also included a flexible joint to the wheel, which allows it to climb and drive over little obstacles. Considering the fact that the robot is also supposed to move on water it would be crucial to think about how big the wheels should be but also have less weight, so that it can easily float without being completely submerged and at the same time be able to support its total weight.

b) Material research

Having the given requirement constraints in mind, it was best for us to use lightweight materials for the robot. So it can easily move on land and water without any troubles. It would also be good to think about high heat resistant materials like tungsten having the highest melting point at 3422 ℃ or like fiber that is lighter and reaching the 500 ℃. Thus, we carried out research as regards the best materials suitable for our robot and this was what we discovered:

- Steel is one of the materials used most often by robot builders. This material is a smart choice if we are looking to build a robot that stands up to harsh conditions. As our surrounding could be possibly be surrounded by water , we are looking to use materials that are highly water resistive. Steels can withstand a pressure similar to that of a 50m (165ft) depth.
- Aluminum is also a good material, this is better if we should worry about the robots exterior becoming rusty over time because aluminum does not rust. This material is easier to shape and is lighter, we can also have to use it to cover the fragile parts of our robot.

Rubber will be used for the wheels, the wheels will be made of a special rubber that allows it not to harden when the temperature gets very low. In addition, its wider ridges allows it to get a better grip on snow or mud and to better drive water from damp or wet areas.

B. Analysis and Design

1) Analysis on Powering systems

According to research carried out, we would require two separated power sources to supply energy consumption for

the signal devices and actuators. The rationale behind this separation is the inherent difference in the way energy is being consumed by both. The signal devices for example, such as access points, cameras etc., have often a constant consumption rate through the whole operating time. On the other hand, the robot's actuators often would have significantly sporadic and dynamic energy consumption rate depending on the obstacles on the way or the operations it intends to carry out.

To provide the power source for our robot's devices and actuators, we first need to determine the systems specifications from consumption point of view. The power section of electronic devices needs a 24-V power source with 4 A continuous current. For the sake of the actuators, we similarly need a 24-V power source while its continuous current ranges from 0 to 20 A. The other factor to be considered is the system's burst current, which determines how quickly the batteries are going to be discharged. This factor is only depended on the actuators' behavior due to their stall current when they are applying their maximum torque. Accordingly, the c-rating of the power source must satisfy this characteristic of the system.

Considering the above discussion, we decided in powering our robot with two 24 V lithium polymer battery packs with 10,000 mAh capacity and a c-rating of 10. Having said that, the power section of electronic devices can operate up to 2.5 h while the provided power source for actuator suffices for half an hour on average for each a mission.

2) Body part analysis

For the good behavior of our robot, it was trivial for us to think about a solution that can allow us move across all types of fields. Thus, the junction between the upper and the lower part played a capital role during our designing.

This junction is constituted of 2 parts:

One part looking like a "screw-key" allowing our robot to move easily from the left to the right and vice-versa and attached to a metal rod passing through the whole body.

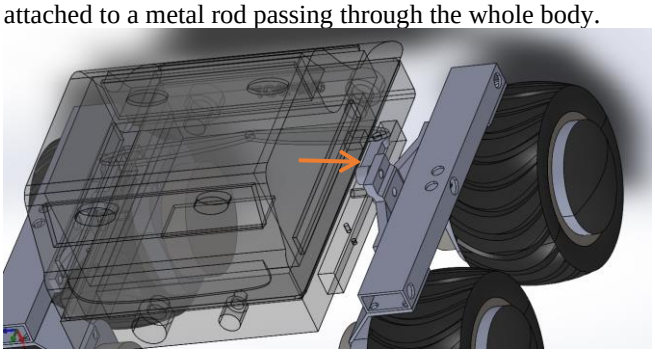
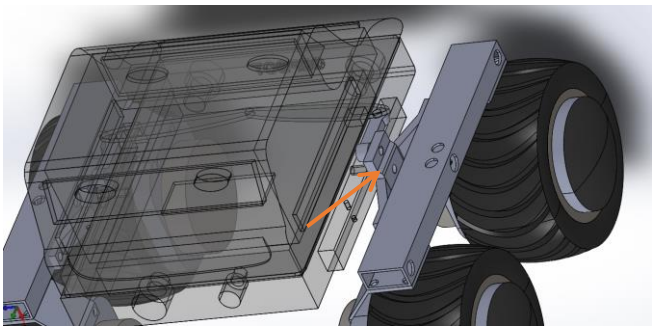


Fig (a): One part looking like a "screw-key"

The other part looking like a "cross" attached both to the rod of the wheels and also to the "screw-key". The connection with the "screw-key" allows a balance weight while allowing the up and down rotation.



Using rotary junctions between wheels and frames, our robot will be able to have an even weight balance as this would aid its movement. After defining the limits of dimensions, we also wanted to be sure that our robot meets the weight requirements defined, thus we used the "Cura" Software to define this limit.

To further aid the interaction of our robot with its surrounding, certain functionalities were added to the upper hood of our robot, namely: cameras, and sensors. Below we will further analyze the functionality and usefulness of each:

Inertial Measurement Unit: we utilize this measurement regarding the changes in the altitude of the robot's platform using a 6-DOF inertial sensor, Xsens MTI-100, to control the laser scanner's stabilizer. The functionality of this stabilizer gets indispensable when the operation terrain is not even.

Thermal Camera: a thermal camera namely Optris PI230, have been installed on the robot which is capable of synchronous capturing of visual and thermal images. This camera is used for detection and position estimation of victims during a mission as a part of the robot's exploration operations. It also ensures robot's visual acuity by providing thermographic information from targeted objects.

CO2 sensor: Here we have a gas sensor, MQ-9, which detects the presence of multiple types of gases in an environment. While this sensor has several applications for a rescue mission, we are more interested in its application to verify a victim's breathing as a vital sign to check the victim current state. Also, we will apply this sensor too verify the level of air pollution within a given space.

Analog Cameras: Analog cameras have been mounted on the robot's upper hood to provide a decent perspective for the robot's operator to monitor the robot and its surroundings through the installed cameras.

IP Camera: As an added incentive to further enhance the robot's ability to Identify its intended target, the operator would require a high-resolution video to complete the inspection task of its detected victim or target as clear as possible. Hence, a Sony high-resolution IP camera has been embodied into the robot's upper hood.

During the designing process, we thought about finding a solution that would be a perfect reflection of what we wanted and what could also reflect the realities of the environment and also the context we are working on.

Thus, we tried to emphasized on two principles aspects: the dimensions and the weight.

- The dimensions we wanted are comprised within these ranges

- Height < 50cm
- width < 70cm
- Length ≈ 50cm

- The Weight we wanted for our robot and that was defined was that the material consumption of the printing material should not be more than 1 kg.

3) Horizontal Rod and part joining rod and motor

In the following figures, Fig 3(a) and Fig. 3(b), the horizontal rod (the road at the sides of the robot) and the small part that joins this horizontal road with the motors are shown. The dimensions shown are all in millimeters.

The importance of the horizontal rod of our robot is of utmost importance since it provides the much-needed flexibility and stability of our robot. This intuitive idea can serve as purposes for connecting the body and motors of the wheel, and additionally, connecting parts that give our robot extra flexibility. On the sides of the rod, there are holes that connect to parts that goes inside the robot, and connects it to the other side of the robot. This serves as a gimbal system that provides our body an extra range of motion both in the vertical and horizontal directions. This essentially solves the problem where the body would simply rather more rigid, which would in turn make the robot much more imbalanced. Full sketches of the these can be found in Annex F and G.

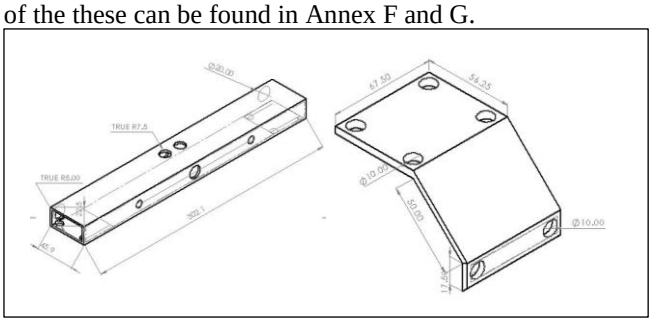


Fig. 3: Sketches of the horizontal rod and the part joining rod and motor.

4) Tire and motor

In the following Fig. 4, the tires of the robot have been given. According to our research, the best way to drive on water is basically to use the same tires we used to drive on land. The combined volume of the air in all the four tires fitted on the robot would make it possible for the robot to float on the water. This is similar to the impact such tires will have when they are used in agricultural machines as they distribute the weight of the machines gently over the soil. So, with a little more energy provided, our lightweight robot can easily drive on water. The directional tire threads have been implemented which are responsible for accelerating smoothly, driving through corners tightly and safely, brake efficiently and it will also extend the battery life by consuming less battery power. The shape of the tire is also crucial for the robot. The shape of the tires is wide because wide tires can drive on bumpy terrain more smoothly. The tires are essentially big enough to carry the weight of the body in such a way that not only it drives smoothly on water, but also gives it extra stability so that it does not topple over, or drown in the water.

We used an individual motor for each tire and the motors are directly connected with tires that supply direct rotational power to tires. Our robot will be able to drive on the water so we need waterproof motors. We figured out two ways to do that. We can simply attach a waterproof motor which is nowadays available and also produced by various manufactures. The waterproof motors are IP68 certified even under extreme conditions and operable in both directions. These kinds of motors are highly efficient, greatly silent, and use less energy and is fully functional under 40m depth in water. It is also possible to add an extra cove to the motor to make it waterproof.

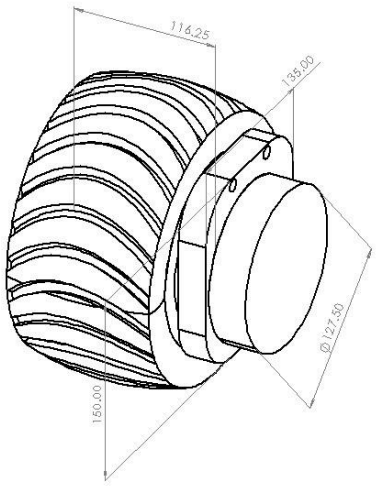


Fig. 4: Tire and motor.

C. FINAL MODEL

The final sketch then resulted to this figure:

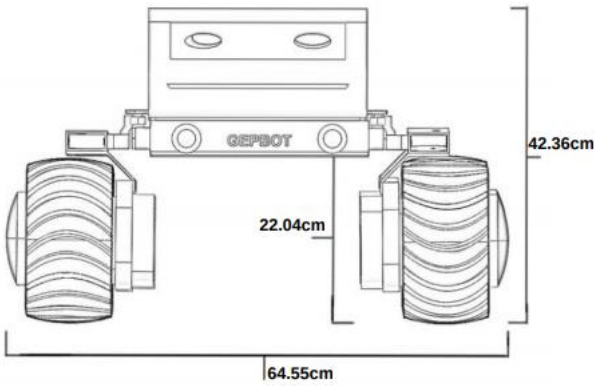


Image 1 : Front view

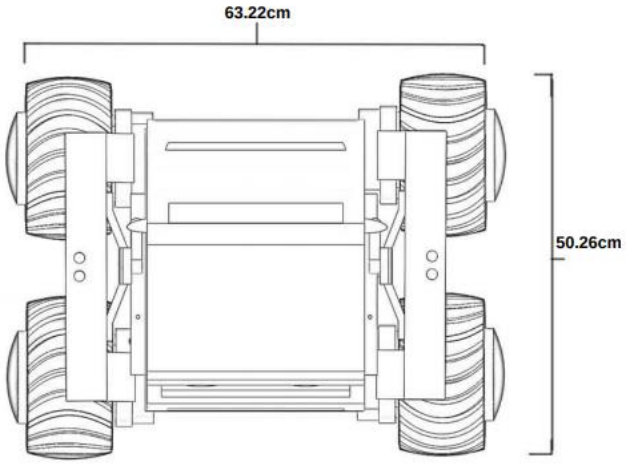


Image 2 : Top view

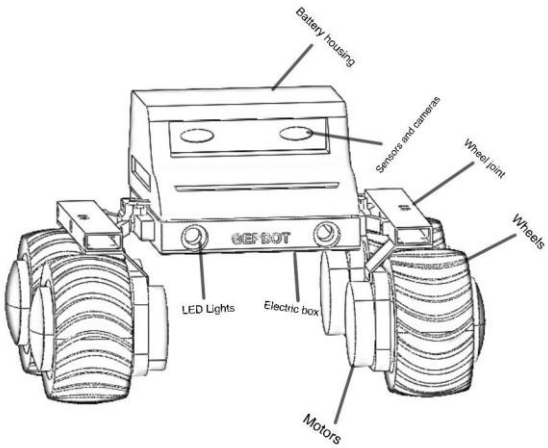
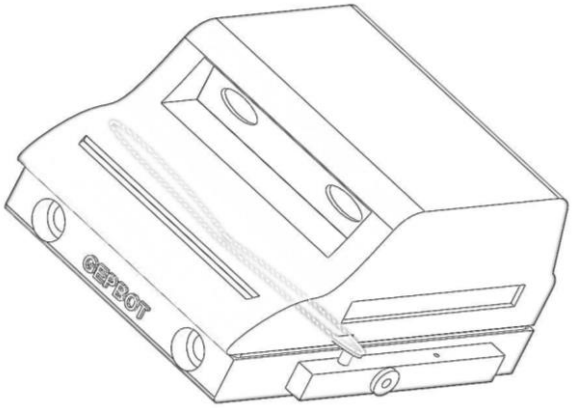


Fig (a): Final design sketch

After defining the limits of dimensions, we also wanted to be sure that our robot meets the weight requirements defined, we then used the "Cura" Software to define this limit. For the final model design of our robot, we decided to choose the Solidworks 2020 app because of his high lots of features and also his ease of use compared to the other software like Fusion 360 or even Rhinoceros.

The upper body will consist of the sensors, the data retrieval cameras and other circuit components.



Upper body diagram

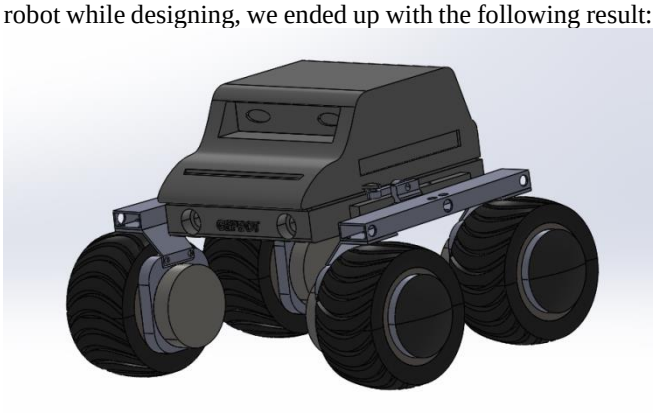
The lower body where GEPBOT is written will carry the most sensitive components of the bot, the Arduino, circuits and the LED lights, The front circle holes will have two white LED light and for the two holes behind, there would be one red LED light to act as the break light and the other a white LED light.

The inner lid will be used to tightly seal and cover the sensitive components inside the lower body.

The length of the main body Fig. (a). is approximately 40cm, the width 30cm and the height 20.3cm.

We had different designs for our robot, we thought to make a design that looks friendly and not frightening because the human mind has a way of imagining live faces in objects, we then decided to add a smiley mark and a human shaped eyes (place for the camera and sensors) both at the front and the back of the bot. The above diagram was the final design we arrived at.

When we then applied the dimensions, we wanted to our robot while designing, we ended up with the following result:



V. IMPLEMENTATION

This section is related to the software part of our robot, referring to the implementation i.e., coding in the C programming language of all the requirements stated before. we decided to separate the degree of complexity of our code and move step by step i.e.:

- Get the representation of the world, the target and a wall
- Toggle the driving mode (from land to water)
- Assume the trajectory from initial point to target and manage energy
- Consider sea target and destroy obstacles

A. TASK 1

This part of our project has to do with the coding of the movement of our robot in C programming language.

This task involves writing a code for the robot to drive successfully to a marked target location and give a feedback that the target has been found.

We were able to solve this task by implementing on an X and Y axis coordinate which we identified as NS and WE coordinates. The code for task one was designed in a way to help the robot find the shortest route to the target. Since we were given the map in a one-dimensional array system, we approached the problem by converting the one-dimensional array map to a two-dimensional array. With this it was easier to track the robot's movement along the x and y axis.

```
int NS_distance= robot_index_NS_cord - target_index_NS_cord;
int WE_distance= robot_index_WE_cord - target_index_WE_cord;

 if(abs(WE_distance) <= abs(NS_distance))
    {
        if(NS_distance >= 0)
        {
            return 1; // north
        }
        else
        {
            return 3; // south
        }
    }
    else
    {
        if(WE_distance >= 0)
        {
            return 4; // west
        }
        else
        {
            return 2; //east
        }
    }
```

The code was tested and was successful on four different maps.

```
//Position in NS and WE coordinate;
robot_index_NS_cord= robot_index / width;
robot_index_WE_cord= robot_index % width;
target_index_NS_cord= target_index / width;
target_index_WE_cord= target_index % width;
int NS_distance= robot_index_NS_cord - target_index_NS_cord;
int WE_distance= robot_index_WE_cord - target_index_WE_cord;
```

B. *TASK 2*

This part involves moving on water, avoiding inside walls, destroying walls and coming back home with target and calculating energy.

In the first task, where we had to navigate around maps without any inside walls, the objective was rather very simple, i.e. to find the target and move towards it with the minimum number of steps possible.

To do this, we simply calculate the distance in the horizontal direction (West to East distance) and in the vertical directions (North to South distance) and then, calculate the distance between our robot and the target and simply move towards these steps. This is done as shown in Fig. 5.

For the next task, the maps get a little more complex. Now, the robot has to face inside walls and avoid them, move on water and then move towards the target in the minimum number of steps possible.

To tackle this, we thought of using the logic that every-time in any direction before moving, the robot would check whether the space in front of it is a wall, or water, or if it is free land surface to move ahead.

If the space Infront of it is a wall when moving in North or South direction, it would try to find a free space to move in the West or East direction. And vice versa, i.e. if the space in front of it is a wall when moving in the West or East direction, it would try to move in the North or South direction. But it would do so, in the direction towards the target.

```
if ( world [robot_index-(width+1)] == 'O' && driving_mode == 0 && coming_from=='O' )
{
    return 1;
}

else if ( world [robot_index-(width+1)] == '~' && driving_mode == 0 && coming_from=='O')
{
    return 5;
}
else if ( world [robot_index-(width+1)] == 'O' && driving_mode == 1 && coming_from=='~')
{
    return 5;
}
else
{
    return 1;
}
```

Fig. 6: Logic to detect water and toggle driving mode.

Fig. 7: Logic to avoid inside walls in a map.

```
if (world [robot_index-1] != '#')
{
    return 4;   // west
}
else if (world [robot_index-1] == '#')
{
    if(NS_distance >= 0)
    {
        if(world[robot_index-(width+1)] != '#')
        {
            return 1;
        }
        else
        {
            if (world[robot_index+(width+1)] != '#')
            {
                return 3;
            }
```

Simultaneously, it would also check if the space in front of it is water surface. If it detects so, it would toggle to water driving mode and only then it would be able to move on water. And then, when it detects land surface Infront of it again, it would toggle back to land driving mode or else, the map fails to work. The implementation of this is shown from snippet of the source code as in Fig. 6 ad 7.

Moving on from this, next we have extra obstacles inside the map , for now which behaves normally like a wall. The main task is to avoid this obstacles, retrieve the target and come back to its base, the original starting position. For each step, toggling of driving mode, the energy is now calculated. The goal is to complete this objective using the least amount of energy.

To visualize this, we used the idea that our robot would use the same logic as before to detect the new obstacles as it did to identify walls. After that, it would use the same logic to navigate around it as before.

For returning back to home, we thought of a simple logic to make this possible. After the robot has reached the target, we simply now set our new destination to our home, using the same logic we did to track our movement towards the target in the first place. This is shown in Fig. 8 and 9. The unexplained printf functions are only for debugging purposes.

```
 else if ( rescued >=1)
 {
     |
      for (int i=0; i<200; ++i){
     if (world[i]=='X')
     {
         target_index=i;
         //The Target Location;
         break;
     }
 }
```

Fig. 8: Logic to return back to base.

```
if(WE_distance >= 0)
{
    if ((world [robot_index-1] != '#')  )
    {return 4;} // west
    else if ((world [robot_index-1] == '#')  )
    {
        if(NS_distance >= 0)
        {
        if((world[robot_index-(width+1)] != '#')  )
            {return 1;}
            else {printf("9043964 \n");}
        }
        else
        {
            if ((world[robot_index+(width+1)] != '#')  )
            {return 3;}
            else { printf("**** \n");}

        }


    }
    else if ((world [robot_index-1] == '*')  )
    {
        if(NS_distance >= 0)
        {
        if((world[robot_index-(width+1)] != '*')  )
            {return 1;}
            else {printf("9043964 \n");}
        }
        else
        {
            if ((world[robot_index+(width+1)] != '*')  )
            {return 3;}
            else { printf("**** \n");}
```

Fig. 9: Logic to avoid obstacles in a map.

Lastly, we work on the last objective where we destroy the obstacles inside the map in each of the four directions, when detected and then move forward and calculate the energy. After that, we run this combined implemented code for all of the maps till now simultaneously and check if it works for each of the given environment. The final source code can be found in Annex J.

VI. SUMMARY

This project involves the designing and development of a rescue robot which will solve problems such as human rescue, the main task of GEPBOT+EXEMPLARY is to enter into high risk situations, save lives and retrieve sensitieve datas, where would be too risky for human rescues to reach. The Robot is more of an autonomous system, designed to overcome obstacles easily, drive on water atleast 3m deep, retrieve datas from the environment and track down humans in danger using sensors and camera. We were motivated to do this project because, robots have been playing an important role in helping rescue teams to perform dangerous missions. Rescue robots will play more important role in future rescue operations and even replace rescuers to perform dangerous missions, we want to be part of the generation to make this huge impact to rescue operations.

REFERENCE

[1] Murphy RR (2014) Disaster Robotics. The MIT Press, New York J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] 2 Kitano H, Tadokoro S, Noda I (1999) RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents' research. IEEE SMC '99 Conference Proceedings. IEEE, Tokyo, Japan, pp 739–743

[3] 3 Tsitsimpelis I et al (2019) A review of ground-based robotic systems for the characterization of nuclear environments. Program Nuclear Energy 111:1

[4] Murphy RR et al (2008) Search and Rescue Robotics. In: Siciliano B, Khatib O (eds) Springer Handbook of Robotics. Springer, Berlin, Heidelberg, pp 1151–1173

[5] Reddy AH, Kalyan B, Murthy CSN (2015) Mine rescue robot system – a review. Procedia Earth Planet Sci 11:457–462

VII. CONTRIBUTION OF EACH MEMBERS IN THE PAPER.

Abstract - Obi Nnamdi Elijah
I. Introduction - Obi Nnamdi Elijah
II. Requirement Elicitation. Enekwa Izuchukwu George
III. Modelling (Systems Engineering) III.A. Overview - Patrick Nonki
III.A.1. Use-Case diagram - Patrick Nonki
III.A.2. Requirements diagram - Patrick Nonki
III.A.3. Block definition and internal block diagram - Patrick Nonki
III.A.4. Activity diagrams - Patrick Nonki
III.A.5. Sequence diagram - Enekwa Izuchukwu George
III.B. Modelling aspects of our robot - Asm Nurussafa.
IV. Design-Enekwa Izuchukwu George, Obi Nnamdi Elijah, Patrick Nonki
IV.A. Research - Enekwa Izuchukwu George, Obi Nnamdi
IV.B. Analysis and Design. - Enekwa Izuchukwu George, Obi Nnamdi Elijah, Patrick Nonki
IV.B.1. Analysis on Powering systems - Enekwa Izuchukwu George, Obi Nnamdi Elijah, Patrick Nonki

IV.B.2. Body part analysis-
IV.B.3. Horizontal Rod and part joining rod and motor- Asm Nurussafa
IV.B.4. Tire and Motor- Asm Nurussafa
IV.C. Final Model
V. Implementation
V.A. Task 1- Enekwa Izuchukwu George, Obi Nnamdi Elijah
V.A. Task 2- Moving on water, avoiding inside walls, destroying walls and coming back home with target and calculating energy- Asm Nurussafa
VI. Summary-Enekwa Izuchukwu George

VIII. AFFIDAVIT (ASM NURUSSAFA)

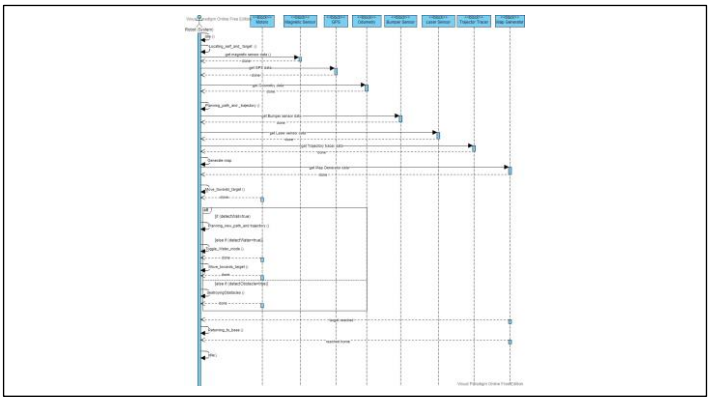
IX. AFFIDAVIT (OBI NNAMDI ELJAH)

X. AFFIDAVIT (NONKI PATRICK)

I hereby confirm that I have written this paper independently and have not used any sources or aids other than those indicated. All statements taken from other sources in wording or sense are clearly marked. Furthermore, I assure that this paper has not been part of a course or examination in the same or a similar version.

Last Name, First Name: Nonki Patrick
Location, Date: Paderborn, 16.07.2021
Signature: Nonki Patrick

XI. APPENDIX



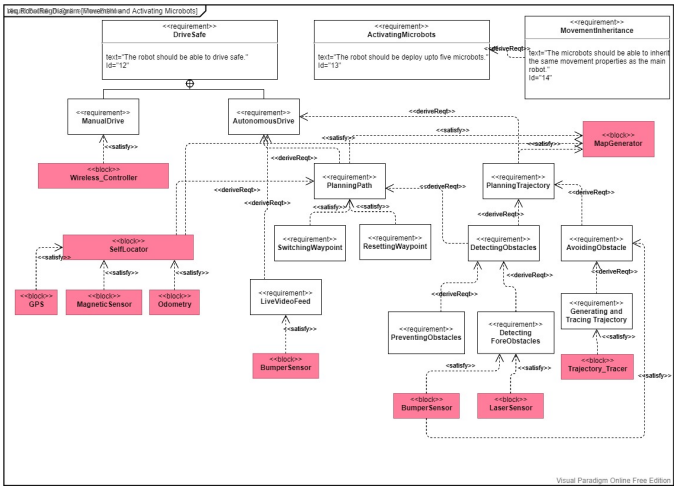


Fig. 11: Requirements diagram- For Movement

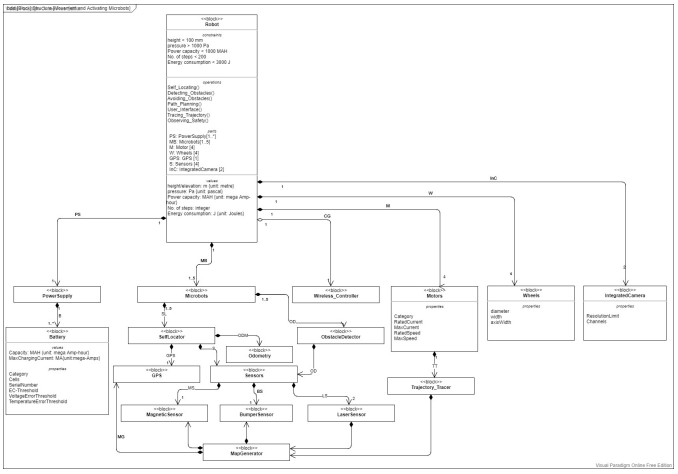


Fig. 12: Block definition diagram from movement.

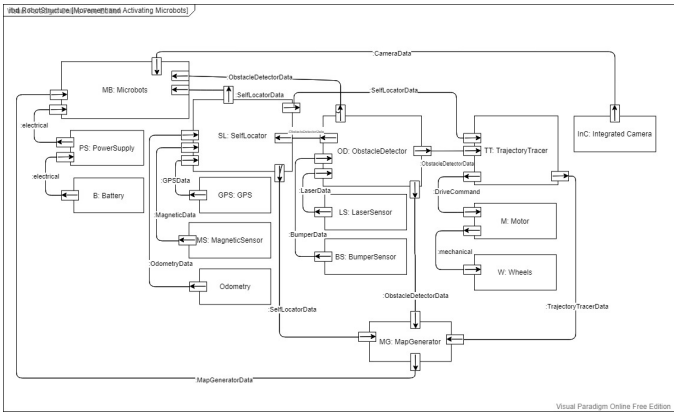


Fig. 13: Internal block definition diagram from movement.

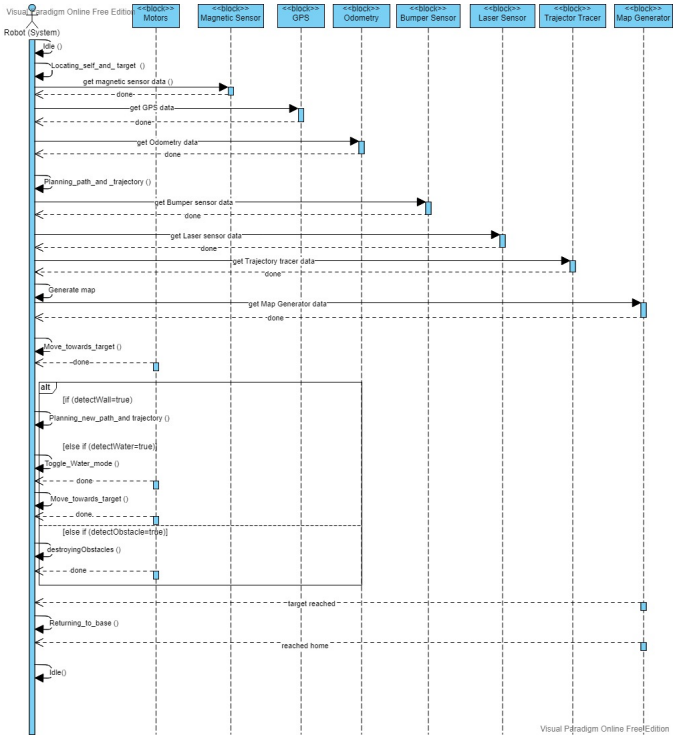


Fig. 14: Sequence diagram for movement.

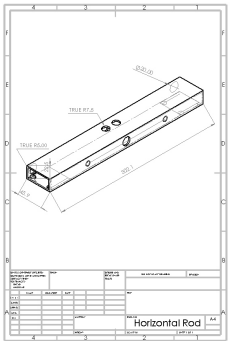


Fig. 15: Sketch for Horizontal Rod.

VIII. APPENDIX

A. Context Analysis

B. Requirements for Movement

C. BDD diagram for Movement

D. IBD diagram for Movement

E. Sequence diagram for Movement

F. Sketch for Horizontal Rod

G. Sketch for Part joining horizontal rod and motor

H. Use case diagram- Automation

I. Use case diagram- Manual

J. Source code for the final task.

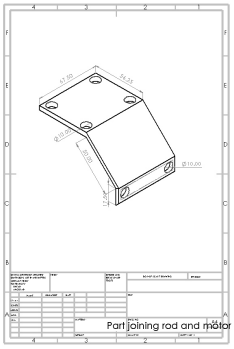


Fig. 16: Sketch for part joining rod and motor.

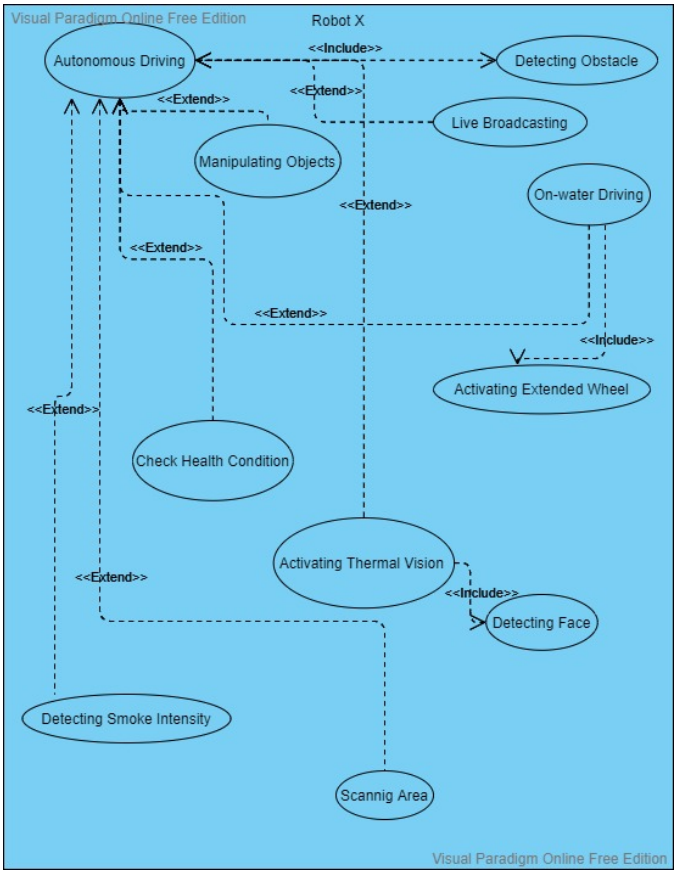


Fig. 17: Use case diagram- Autonomation.

```
/*
A very simple text based simulation of a
    ↪ rescue scenario. Wall, robot and
    ↪ target only.

Copyright 2021 Kristian Rother (kristian.
    ↪ rother@hshl.de)
```

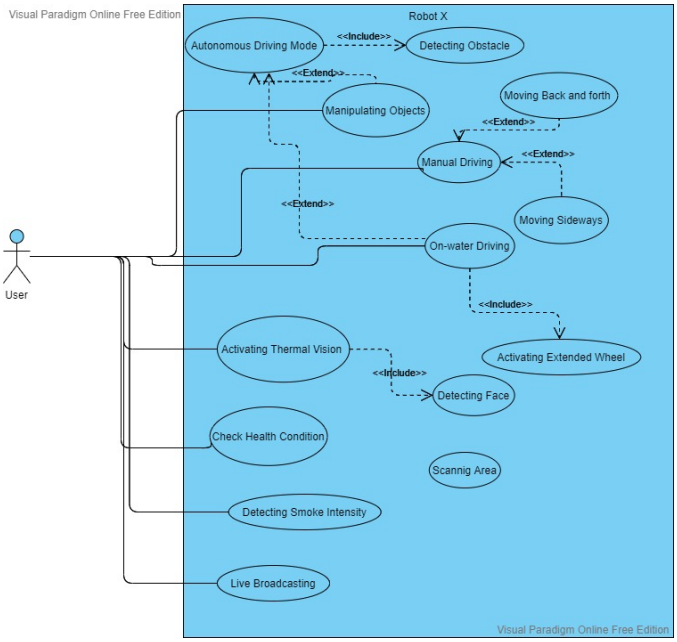


Fig. 18: Use case diagram- Manual.

```
    ↪ COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
    ↪ INDIRECT, INCIDENTAL, SPECIAL,
    ↪ EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO,
    ↪ PROCUREMENT OF SUBSTITUTE GOODS OR
    ↪ SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION
    ↪ ) HOWEVER CAUSED AND ON ANY THEORY
    ↪ OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (
    ↪ INCLUDING NEGLIGENCE OR OTHERWISE)
    ↪ ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF
    ↪ ADVISED OF THE POSSIBILITY OF SUCH
    ↪ DAMAGE.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>



int driving_mode = 0;
int rescued = 0;
int energy = 0;
char coming_from = 'X'; // We start on the
    ↪  space we need to return to

// THIS IS THE FUNCTION YOU IMPLEMENT
int move(char *world)
{
   // Functions for robot and target
      ↪ location
   int robot_index, target_index,
      ↪ target_index_NS_cord,
      ↪ robot_index_NS_cord;
   int width, robot_index_WE_cord,
      ↪ target_index_WE_cord;
   int nexmove=1;

   for (int i=0; i<200; ++i)
   {
      if (world[i]=='\n')
      {
         width= i+1;
         break;
      }
   }



   for (int i=0; i<200; ++i)
   {
      if (world[i]=='R')
      {
         robot_index=i;
         //The robot Location;
         break;
      }
   }



   for (int i=0; i<200; ++i)
   {
      if (world[i]=='T')
      {
         target_index=i;
         //The Target Location;
         break;
      }
   }



   printf("\n width = %d \n", width);
   //Position in NS and WE coordinate;
   robot_index_NS_cord= robot_index /
      ↪ width;
   robot_index_WE_cord= robot_index %
      ↪ width;
   target_index_NS_cord= target_index /
      ↪ width;
   target_index_WE_cord= target_index %
      ↪ width;
   int NS_distance= robot_index_NS_cord -
      ↪ target_index_NS_cord;
   int WE_distance= robot_index_WE_cord -
      ↪ target_index_WE_cord;

   printf (" NS distance= %d, WE distance
      ↪ = %d \n", NS_distance,
      ↪ WE_distance);

   if (rescued < 1)
   {


      if(abs(WE_distance) <= abs(
         ↪ NS_distance))
   {
       if(NS_distance >= 0)
      {
        if ( world [robot_index-(width+1)
           ↪ ] == 'O' && driving_mode ==
           ↪  0 && coming_from=='O' )
        {return 1;}

        else if ( world [robot_index-(
           ↪ width+1)] == '~' &&
```

```
        ↪ driving_mode == 0 &&
        ↪ coming_from=='O')
    {

        return 5;
    }
    else if ( world [robot_index-(
        ↪ width+1)] == 'O' &&
        ↪ driving_mode == 1 &&
        ↪ coming_from=='~')
        {

            return 5;

        }
        else
        {

            return 1;

        }
 }
else
 {

    if (( world [robot_index+(width
        ↪ +1)] == 'O') && (
        ↪ driving_mode == 0) && (
        ↪ coming_from=='O') )

   {printf("south\n");
      return 3;} // South

   else if ( world [robot_index+(
        ↪ width+1)] == '~' &&
        ↪ driving_mode == 0 && (
        ↪ coming_from=='O' ||
        ↪ coming_from=='X'))
    {
        return 5;
    }
    else if ( world [robot_index+(
        ↪ width+1)] == 'O' &&
        ↪ driving_mode == 1 &&
        ↪ coming_from=='~')
        {

            return 5;

        }

    else if ((world [robot_index+(
        ↪ width+1)] == '*') )
    {
        if(WE_distance >= 0)
        {
            if((world[robot_index+1] !=
                ↪  '*') )
            {printf( "hereeeee\n");
                ↪ return 2;} //eest
            else {printf("9043964␣\n")
                ↪ ;}
        }
        else
        {
            if ((world[robot_index-1]
                ↪ != '*') )
            {return 4;} // wast

             else { printf("****␣\n");}

        }


    }
    else
        {
            printf ("Eukajshf\n");
            return 3;

        }
    }
}
else
{
   if(WE_distance >= 0)
   {
      if ((world [robot_index-1] != '#'
          ↪ ) )
      {return 4;} // west
      else if ((world [robot_index-1]
          ↪ == '#') )
      {
         if(NS_distance >= 0)
         {
         if((world[robot_index-(width
             ↪ +1)] != '#') )
             {return 1;}
             else {printf("9043964␣\n")
                 ↪ ;}
         }
         else
         {
             if ((world[robot_index+(
                 ↪ width+1)] != '#') )
             {return 3;}
             else { printf("****␣\n");}

         }


      }
```

```
    else if ((world [robot_index-1]
        ↪ == '*') )
    {
       if(NS_distance >= 0)
       {
       if((world[robot_index-(width
           ↪ +1)] != '*') )
          {return 1;}
          else {printf("9043964␣\n")
              ↪ ;}
       }
       else
       {
          if ((world[robot_index+(
              ↪ width+1)] != '*') )
          {return 3;}
          else { printf("****␣\n");}

       }


    }
    else {return 1;}
}


else
{

    if ((world [robot_index+1] != '#'
        ↪ ) )
    {return 2;} // east
    else if ((world [robot_index-1]
        ↪ == '#') )
    {
       if(NS_distance >= 0)
       {
       if((world[robot_index-(width
           ↪ +1)] != '#') )
          {return 1;}
          else {printf("9043964(3)\n"
              ↪ );}
       }
       else
       {
          if ((world[robot_index+(
              ↪ width+1)] != '#') )
          {return 3;}
          else { printf("****(3)\n")
              ↪ ;}

       }


    }
```

```
       else if ((world [robot_index-1]
           ↪ == '*') )
       {
          if(NS_distance >= 0)
          {
          if((world[robot_index-(width
              ↪ +1)] != '*') )
             {return 1;}
             else {printf("9043964(3)\n"
                 ↪ );}
          }
          else
          {
             if ((world[robot_index+(
                 ↪ width+1)] != '*') )
             {return 3;}
             else { printf("****(3)\n")
                 ↪ ;}

          }


       }
       else {return 1;}
    }
}
}

else if ( rescued >=1)
{

    for (int i=0; i<200; ++i){
   if (world[i]=='X')
   {
      target_index=i;
      //The Target Location;
      break;
   }
}
robot_index_NS_cord= robot_index /
   ↪ width;
robot_index_WE_cord= robot_index %
   ↪ width;
target_index_NS_cord= target_index /
   ↪ width;
target_index_WE_cord= target_index %
   ↪ width;
int NS_distance= robot_index_NS_cord -
   ↪ target_index_NS_cord;
int WE_distance= robot_index_WE_cord -
   ↪ target_index_WE_cord;

if(abs(WE_distance) <= abs(NS_distance)
   ↪ )
{
    if(NS_distance >= 0)
```

```
    {
      if ( world [robot_index-(width+1)
          ↪ ] == 'O' && driving_mode ==
          ↪  0 && coming_from=='O' )
      {return 1;}

      else if ( world [robot_index-(
          ↪ width+1)] == '~' &&
          ↪ driving_mode == 0 &&
          ↪ coming_from=='O')
       {

          return 5;
       }
       else if ( world [robot_index-(
          ↪ width+1)] == 'O' &&
          ↪ driving_mode == 1 &&
          ↪ coming_from=='~')
          {

             return 5;

          }
          else
          {

             return 1;

          }
    }
    else
    {

       if ( world [robot_index+(width+1)
          ↪ ] == 'O' && driving_mode ==
          ↪  0 && coming_from=='O' )
      {return 3;}

      else if ( world [robot_index+(
          ↪ width+1)] == '~' &&
          ↪ driving_mode == 0 &&
          ↪ coming_from=='O')
       {
          return 5;
       }
       else if ( world [robot_index+(
          ↪ width+1)] == 'O' &&
          ↪ driving_mode == 1 &&
          ↪ coming_from=='~')
          {

             return 5;

          }
          else
          {
```

```
             return 3;

          }
    }
}
else
{
   if(WE_distance >= 0)
   {
      if ((world [robot_index-1] != '#'
          ↪ ) )
      {return 4;} // west
      else if ((world [robot_index-1]
          ↪ == '#') )
      {
         if(NS_distance >= 0)
         {
         if((world[robot_index-(width
             ↪ +1)] != '#') )
            {return 1;}
            else {printf("9043964(3)\n"
               ↪ );}
         }
         else
         {
            if ((world[robot_index+(
               ↪ width+1)] != '#') )
            {return 3;}
            else { printf("****(3)\n")
               ↪ ;}

         }


      }
      else {return 1;}
   }

   else
   {
     if ((world [robot_index+1] != '#')
        ↪  )
      {return 2;} // east
      else if ((world [robot_index-1]
          ↪ == '#') )
      {
         if(NS_distance >= 0)
         {
         if((world[robot_index-(width
             ↪ +1)] != '#') )
            {return 1;}
            else {printf("9043964(3)\n"
               ↪ );}
         }
         else
```

```
            {
               if ((world[robot_index+(
                  ↪ width+1)] != '#') )
               {return 3;}
               else { printf("****(3)\n")
                  ↪ ;}

            }


         }
         else {return 1;}
      }
   }

   }
   else {return 1;}

}


// Return target index
int update_world(int movement, char *world
   ↪ , int robot_index, int width) {
   int target_index = 0;
   // NORTH
   if(movement == 1) {
      energy += 10;
      target_index = robot_index-(width+1)
         ↪ ; // +1 for the newline
   }
   // SOUTH
   else if(movement == 3) {
      energy += 10;
      target_index = robot_index+(width+1)
         ↪ ; // +1 for the newline
   }
   // EAST
   else if(movement == 2) {
      energy += 10;
      target_index = robot_index+1;
   }
   // WEST
   else if(movement == 4) {
      energy += 10;
      target_index = robot_index-1;
   }
   else if(movement == 5) {
      energy += 30;
      printf("Toggling mode!\n");
      target_index = robot_index;
      if(driving_mode == 0) { driving_mode
         ↪  = 1; }
      else { driving_mode = 0; }
   }
   printf("Energy used so far: %d \n",
      ↪ energy);

   // ACTION
   if(world[target_index] == 'O') {
      if(driving_mode != 0) {
         printf("FAILURE, tried to drive
            ↪ on land in water mode!");
         exit(1);
      }
      else {
         world[target_index] = 'R';
         world[robot_index] = coming_from;
         coming_from = 'O';
         return target_index;
      }
   }
   // Stay in the same location (used for
      ↪ driving mode toggling)
   else if(world[target_index] == 'R') {
      return target_index;
   }
   // Walls
   else if(world[target_index] == '#') {
      printf("%s", world);
      printf("%c", '\n');
      printf("FAILURE, crashed into a wall
         ↪ !");
      exit(1);
   }
   // Obstacles (can be destroyed later)
   else if(world[target_index] == '*') {
      printf("%s", world);
      printf("%c", '\n');
      printf("FAILURE, crashed into an
         ↪ obstacle!");
      exit(1);
   }
   else if(world[target_index] == '~') {
      if(driving_mode != 1) {
         printf("FAILURE, entered water in
            ↪ land mode!");
         exit(1);
      }
      else {
         world[target_index] = 'R';
         world[robot_index] = coming_from;
         coming_from = '~';
         return target_index;
      }

   }
    else if(world[target_index] == 'T') {
      world[target_index] = 'R';
      world[robot_index] = coming_from;
      rescued++;
```

```
      printf("␣Rescued:␣%d\n", rescued);
      coming_from = 'O'; // Targets are
         ↪ always considered to be on 'O'
         ↪  spaces
      return target_index;
   }
   else if(world[target_index] == 'X') {
      world[target_index] = 'R';
      world[robot_index] = coming_from;
      if(rescued > 0) {
         printf("%s", world);
         printf("%c", '\n');
         printf("SUCCESS,␣target␣returned␣
            ↪ home!");
         exit(0);
      } else {
         coming_from = 'X';
         return target_index;
      }

   }
}

int main() {
   const int MAX_STEPS = 200;
   int step = 1;

   int movement;
   int width = 20; // excluding newlines

   // The maps
   char world1[200] = {
      '#','#','#','#','#','#','#','#','#',
         ↪ '#','#','#','#','#','#','#','#
         ↪ ','#','#','#','\n',
      '#','O','T','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','R','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','#','#','#','#','#','#','#','#',
         ↪ '#','#','#','#','#','#','#','#
         ↪ ','#','#','#','\n'
```

```
   };

   char world2[200] = {
      '#','#','#','#','#','#','#','#','#',
         ↪ '#','#','#','#','#','#','#','#
         ↪ ','#','#','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','R','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','T','#','\n',
      '#','#','#','#','#','#','#','#','#',
         ↪ '#','#','#','#','#','#','#','#
         ↪ ','#','#','#','\n'
   };

   char world3[200] = {
      '#','#','#','#','#','#','#','#','#',
         ↪ '#','#','#','#','#','#','#','#
         ↪ ','#','#','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','R','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','O','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','O','O','O','T','O','O','O','O',
         ↪ 'O','O','O','O','O','O','O','O
         ↪ ','O','O','#','\n',
      '#','#','#','#','#','#','#','#','#',
         ↪ '#','#','#','#','#','#','#','#
         ↪ ','#','#','#','\n'
   };
```

```
char world4[200] = {
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','R','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','T','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n'
};

char world5[200] = {
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','#','#','#','#',
      ↪ '#','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','#','T','O','O',
      ↪ '#','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','#','#','#','O',
      ↪ '#','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','R','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n'
};

char world6[200] = {
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n',
   '#','O','T','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','~','~','~','~','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','~','~','~','~','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','R','O','O','O','O','O','O',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n'
};

char world7[200] = {
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n',
   '#','O','O','~','O','O','O','O','~',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','R','O','O','O','O','~',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','~','~','~','~','O','O','O','~',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','~','T','~','~','O','O','O','~',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','~','O','~','O','O','O','O','~',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','O','O','O','O','O','O','O','~',
      ↪ 'O','O','O','O','O','O','O','O
      ↪ ','O','O','#','\n',
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
      ↪ ','#','#','#','\n'
};

char world8[200] = {
   '#','#','#','#','#','#','#','#','#',
      ↪ '#','#','#','#','#','#','#','#
```

```
        ↪ ','#','#','#','\n',
    '#','O','O','~','O','O','O','O','~',
        ↪ 'O','O','O','O','O','O','O','O
        ↪ ','O','O','#','\n',
    '#','O','O','R','O','O','O','O','~',
        ↪ 'O','O','O','O','O','O','O','O
        ↪ ','O','O','#','\n',
    '#','*','*','*','*','O','O','O','~',
        ↪ 'O','O','O','O','O','O','O','O
        ↪ ','O','O','#','\n',
    '#','O','T','~','~','O','O','O','~',
        ↪ 'O','O','O','O','O','O','O','O
        ↪ ','O','O','#','\n',
    '#','O','O','~','O','O','O','O','~',
        ↪ 'O','O','O','O','O','O','O','O
        ↪ ','O','O','#','\n',
    '#','O','#','O','O','O','O','O','~',
        ↪ 'O','O','O','O','O','O','O','O
        ↪ ','O','O','#','\n',
    '#','#','#','#','#','#','#','#','#',
        ↪ '#','#','#','#','#','#','#','#
        ↪ ','#','#','#','\n'
};

// Set the world to use
char world[200];
memcpy(world, world1, sizeof(world1));

// Initialize target and robot
    ↪ positions
// Assumes only one target, one robot
unsigned int elements = sizeof(world)/
    ↪ sizeof(world[0]);

// Initialize the index of the robot
    ↪ and the target.
// Assumes exactly one robot and one
    ↪ target
int robot_index;
int target_index;

for(int i = 0; i < elements; i++) {
if (world[i] == 'R') { robot_index = i;
    ↪  }
if (world[i] == 'T') { target_index = i
    ↪ ; }
}

// Print the initial world
printf("Starting␣position:␣%c", '\n');
// Debug output
// printf("Robot index: %i / target
    ↪ index: %i %c", robot_index,
    ↪ target_index, '\n');
printf("%s", world);
printf("%c", '\n');
```

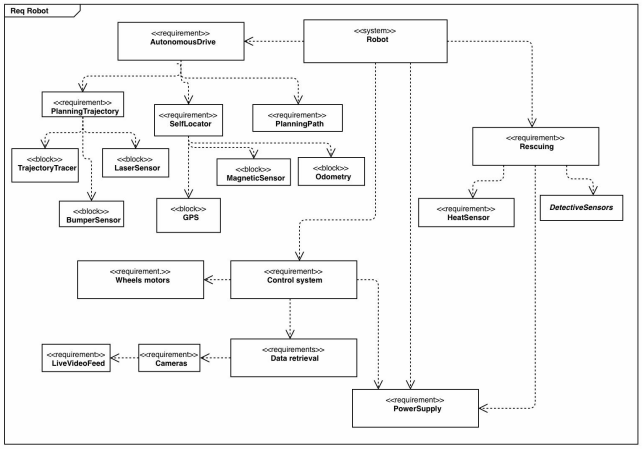


Fig. 19: Combined Requirements Diagram.

```
    while(step <= MAX_STEPS) {
        printf("After␣step␣number␣%i:␣%c",
            ↪ step, '\n');

        movement = move(world);
        robot_index = update_world(movement,
            ↪  world, robot_index, width);
        printf("%s", world);
        printf("%c", '\n');
        step = step+1;
    }
    if(step >= MAX_STEPS) {
        printf("FAILURE,␣maximum␣number␣of␣
            ↪ steps␣exceeded.");
    }

}
```

K. Combined Requirements diagram.

L. Combined Block definition diagram.

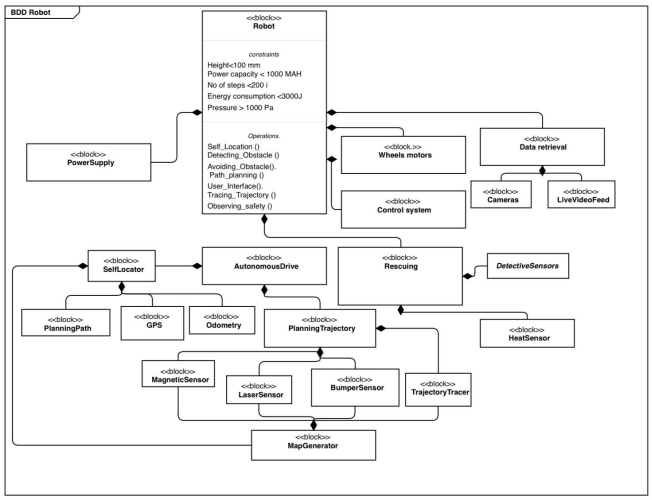


Fig. 20: Combined block definition diagram.