The Development of Lifefrog

Vu Quang Tuong Nguyen
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
vu-quang-tuong.nguyen@stud.hshl.de

Thanh Long Phan
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
thanh-long.phan@stud.hshl.de

Chen Shih
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
shih.chen@stud.hshl.de

Christian Stratmann
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
christian.stratmann2@stud.hshl.de

Nikolaos Karapoulatidis
Electronic Engineering
Hochschule Hamm-Lippstadt
Erwitte, Germany
nikolaos.karapoulatidis@stud.hshl.de

Üsame Gönül
Electronic Engineering
Hochschule Hamm-Lippstadt
Geseke, Germany
uesame.goenuel@stud.hshl.de

Abstract (Phan) – Rescue Robot has been becoming an important tool for Search and Rescue efforts in recent decades. As a part of the effort to develop new generations of rescue robots, Lifefrog is designed to be a rescue robot specialized in searching for the injured and delivering supplies. Short and mobile, Lifefrog can easily go through narrow space that is not accessible to human beings. It can navigate autonomously, avoid obstacles, and find a way to reach the victims to confirm their status as well as deliver them supplies. Being amphibious is also an important feature of Lifefrog.

To develop the robot in a systematic way, initially requirements were made, and different diagrams were created to help visualize the goals and methods to develop the product. This led to further planning, and sketches of the design were soon finished. The first prototype was 3D modeled in Solidworks and in the process of being produced. The robot must traverse through different terrains, both on land and on water, therefore, it is designed with tracks. The hull is made to be as simple as possible. At the bottom of it, motors and battery are placed. Directly above the battery and the motors, electronics parts are mounted on a platform, which includes a Raspberry Pi, an Arduino Uno, and a motor controller. For navigation, ultrasonic sensors and a camera on top are placed on top, along with microphone, and a compartment at the back to store supplies. Additionally, a pathfinding program was written for simulation scenarios. Named Pathfinder V2, it is the latest version of the pathfinding software. It is determined that in the real world, the robot will have to go in blind with no information about the surroundings at all, find the target, and then return to the starting location. During simulation, Pathfinder V2 has proven to be a success, as it passed all test scenarios although minimal information was given. Another iteration which employs A* was also developed to ensure maximum efficiency. For further development, a physical prototype can be created to with real time mapping and pathfinding.

I. Introduction (Gönül)

In most natural and man-made disasters, the first few hours are very critical for the survival of the victims. In such scenarios, it is important to have support in great numbers to spread through the area, looking for the injured and helping them. Manpower might not always be readily available, and even if that was the case, risking human life is neither ethical nor practical solution, especially in chemical or nuclear threats. Using robots, on the other hand, minimalizes that risk while also offering disposable and low-cost alternative. Scenarios like these were the inspiration, Lifefrog project was based on.

The purpose of this paper is to demonstrate Lifefrog and the path to its development. Lifefrog is an autonomous rescue robot designed for human rescue in disaster areas. While its main functionality is to offer support, Lifefrog also must be mobile and durable enough to withstand the environmental factors. Even though the robot is not large enough to carry the victims by itself, it can ease the work of first responders by locating and providing the injured with supplies depending on the situation. In addition, Lifefrog is designed to be autonomous, in other terms, it locates the victims and draws near them without any external help, hence reduces the required manpower further. Another remarkable feature of the smart rescue robot is its mobility. Strong tracks not only provide traction on

various terrains but also propels Lifefrog on water. Amphibious capabilities are significant and rare aspect that will clearly show their advantages during certain scenarios.

II. Methodology (Nguyen)

To accomplish this task, initially a set of requirements based on the scenario that is given by the stakeholders. Thereafter, models, in this case SysML and UML, is formulated to form a more concrete idea of the design. From the models, paper prototypes can be created, concretizing the ideas. Once the paper prototype is finalized, 3D models can be constructed which will be used for 3D printing. Lastly, the software for the robot is created. Since it is unlikely that the physical prototype will be completed in this constricted amount of time, the software is tested using a limited simulation environment using C.

III. Scenario (Chen)

Lifefrog is an amphibious rescue robot made for scenarios such as natural disasters and serious accidents. Its mission covers navigation in unknown areas, delivering first aid kits to the victims, and assisting in rescuing injured people. All these tasks rely on Lifefrog's high performance and ability to operate in harsh environments.

In most operations, Lifefrog will confront rough terrains, and thus mobility plays an important role. Deep water does not limit Life Frog at all. The robot is waterproof and can travel on the water surface. It is not necessary for the robot to be able to submerge in water. The robot shall be able to bypass obstacles, such as debris of buildings, that are less than 30 centimeters tall. When it comes to climbing, the strong power of Lifefrog supports it to overcome a slope which is up to 35 degrees steep. In some extreme cases, Lifefrog has to enter danger zones like nuclear power plants, buildings on fire, or a polluted factory. Therefore, the whole system is resistant to chemical, biological, radioactive, and nuclear elements.

IV. Requirements (Gönül)

Finding out and analyzing the requirements was the first major step taken in this project. Being aware of the requirements are necessary for a stable system, as most internal and external elements are discussed in this stage of planning. For simplicity, this task was divided into smaller and equally important parts and are briefly explained below.

A. Requirement Elicitation (Gönül)

Initial and most crucial points should be made by the customer (or supervising professor in this case), due to them establishing the main objective of the system. Following a short discussion and clarification of the objective, the business goal was synthesized: Building a robot that can assist in locating and rescuing people. This led to a list of matters to be solved, from basic mobility issues to self-driving and path finding, navigation and even communication. On the other hand, some system constraints were also needed to be attended to. The robot would have to be strong, resilient, and long-lasting while also being mobile and small enough to pass through debris and openings. Having an amphibious system was also a demanded feature, so it would be expected to switch between terrestrial and aquatic environments as well.

To get familiar with the area of interest, background research was beneficial. This, in other terms, meant examining possible application domains and discovering existing systems. For a better perspective, disaster scenarios were analyzed. These scenarios were specifically picked to utilize requirements previously determined, like chemical plant explosion, or Fukushima incident. As for existing systems, research was done on amphibious vehicles, firefighting and EOD robots, communication devices, and drones.

Next step was prioritization of the goals, which would be particularly important in later stages of development. Starting from the most important goal, autonomous navigation could be considered to be the central point of this project. Next, high mobility, with an emphasis on amphibious capabilities, was chosen to be on the second place of priority. With support and rescue on third place, it is indicated to be of utmost importance after fulfilling essential features. In fourth and fifth place are element resistance, and power efficiency, respectively, in order to possess high endurance.

Last step of requirement analysis is the collection of requirements. This includes stakeholder requirements as well as domain requirements. First one describes what stakeholders would expect from this system, like: profitability, safety, and ease of use. Meanwhile domain requirements state programming knowledge, robotic knowledge, and ability to work in various harsh environments.

B. Requirement Analysis (Phan)

After carefully analyzing, several trouble spots were found. First of all, the ability to perform self-driving and

pathing is essential. Going into an unknown territory, the robot has to be able to perform self-driving and pathing to be able to reach the objective. Secondly, as an amphibious rescue robot, it has to be able to float and move on water, which is not an easy task, and also poses a risk of water seepage, which can lead to short circuit. Being able to withstand environmental corrosiveness like fire, chemical substances (acid etc.) is also necessary, which requires good knowledge in materials science. Another trouble spot is the ability to overcome impassable obstacles, having methods to either move around obstacles or go over them is vital. Power capacity and efficiency can also be a bottleneck for the range of operation.

In short, the boundaries of realizing the project are as followed:

- Size
- Mobility
- Range of operation
- Materials
- Budget
- Knowledge
- Current technology
- Carry load (for the supplies)

With that said, the most important requirements can be condensed into: Self-Driving, Pathfinding and Amphibious.

C. Requirement Specification (Phan)

Autonomous Navigation:

To be able to complete its core mission of search and rescue injured people, the robot has to be capable of Autonomous Navigation, including Self-Driving and Pathfinding.

1) Density:
It is required that the vehicle have travel on water. For this to be true, the density of the vehicle must be lower than the density of water which is 997 kg/m^3. However, for the most efficient water flow, only half of the body is allowed to be submerged which meant that the density should be lower than 499 kg/m^3.

2) Mobility:
The robot does not need to be high speed to go through collapsed buildings, therefore max speed of 30 km/h is sufficed. However, it needs to cover different terrains as well as going through narrow space or steep slope, because of that, it needs a max height smaller than 750 mm, max width smaller than 800 mm, and must be

able to climb obstacles of maximum 300 mm height on a $60°$ angle.

3) Hearing Frequency and Contact Range:
The robot needs to communicate with the operator from a far to livestream the video footage, therefore communication is important. As per usual, hearing frequency should be between 20 Hz and 20,000 Hz. Sampling frequency doubles the frequency that is being recorded and should be 40000hz. As for communication range, the robot should be able to communicate at the max range of 3 km.

4) Carrying Load:
It is essential for some missions that the robot carries supplies for the stranded victims. Because of that, the robot should be designed to have a maximum carry load of 30 kg.

5) Energy Capacity:
It is critical for the robot to work in a long period of time. For that, it needs energy. It is determined that the robot needs energy for the max operation time of 24 hours.

6) Toughness:
As a rescue robot traveling through harsh terrain under effect of possible chemical, biological, radiological, and nuclear disasters, toughness is vital. Most important problem is water pressure. The robot should be able to handle water pressure of smaller than 10 ATM.

7) Temperature Resistance:
In order to operate in everywhere in the world of different weather and climate, the robot should be designed to be able to handle a wide range of temperature. It is important that the robot can fully operate between –20ºC and 45ºC. This is due to the constraints of batteries.

V. Research (Chen)
In the requirement elicitation phase, requirements and constraints of the system are identified and specified along with possible scenarios. Subsequently, the development team of Lifefrog started to look for existing technologies which are applicable or are already applied to the field of amphibious vehicles. In the directions of amphibious, robotic arm, and driving methods, the research had been carried out.

The previously elicited requirements are of help with narrowing down the information needed. One of the
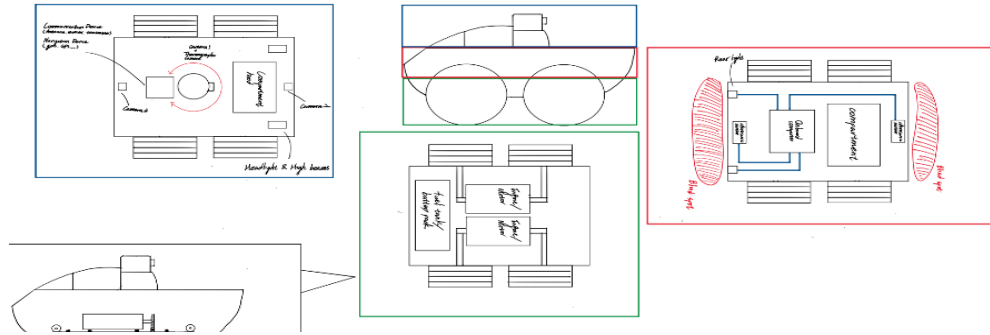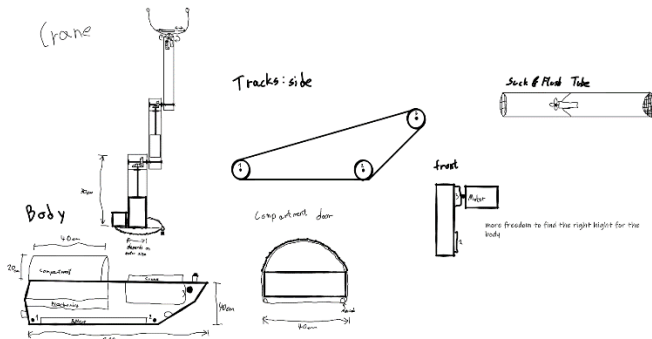
Fig. 1. One of the paper prototypes

Fig. 2. An alternative paper prototypes

Fig. 3. Tracks design

Fig. 4. Wheel design

major focuses is driving methods. Concentrated on common driving methods, the development team analyzed the advantages and disadvantages of wheels and continuous tracks, but also looked out for other possibilities. This research had a significant impact on the design concepts and design decisions of Lifefrog.

1)   *Advantages of wheels:*
- Higher speed - The wheels require less torque to move from a stationary state.
- Maneuverability - In some cases, wheels enable higher maneuverability compared with tracks.
- Lightweight - Weight plays an important role in the performance of vehicles, and wheels are a cut above tracks when it comes to lightweight.
- Simplicity - In comparison to tracks, wheels have relatively simple structure, which makes maintenance easier.

2)   *Disadvantages of wheels:*
- Driving over obstacles - It can be much more difficult for a vehicle to drive over large and vertical obstacles using wheels. In order to resolve this problem, the size of wheels need to be at least twice taller than the obstacles.
- Driving on snow - When driving on soft surfaces such as snow and mud, wheels tend to
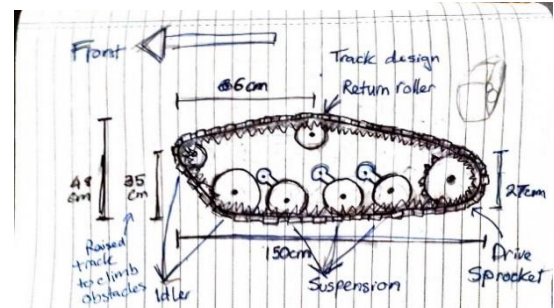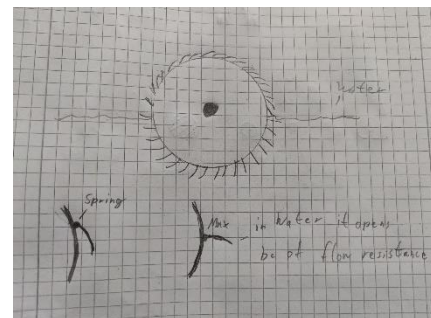
sink in and get stuck because of the lack of surface area.

3)   *Advantages of continuous tracks:*
- Power efficiency - Due to the special design, tracks have higher performance and traction, which leads to a better power delivery efficiency compared with wheels.
- Traction - Tracks have nice performance when driving on slippery surfaces because of high traction.
- Driving on rough terrains - In some extreme situations, wheels would get stuck or damaged. Robust tracks with larger surface area are designed to solve this problem. Furthermore, the continuous tracks are good at climbing ascending and descending stairs.
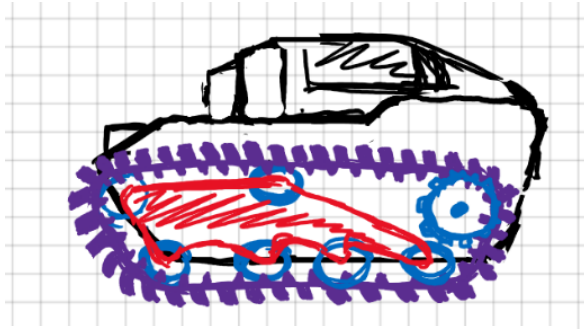- Driving on rough terrains - In some extreme situations, wheels would get stuck or damaged.
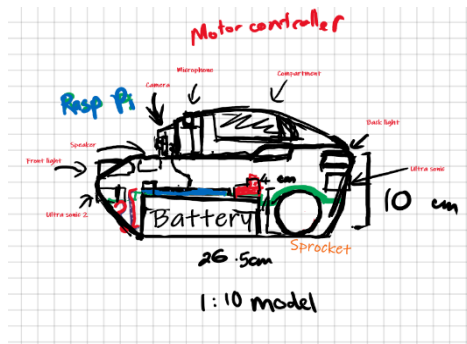
Fig. 5. Final design - side shot

Fig. 6. Final design - internal - side shot

Fig. 7. Final design - internal - bird eye

Robust tracks with larger surface area are designed to solve this problem. Furthermore, the continuous tracks are good at climbing ascending and descending stairs.

4) *Disadvantages of continuous **tracks:***
- Lower speed -Vehicles with tracks usually have lower speed compared with vehicles on wheels because of high complexity and friction. [1]

VI. Design (Karapoulatidis)

D. *Every idea begins as a concept*

Now it is time for creativity and brainstorming. After the scenario, aim, requirements got defined and the research was finished we started to bring our ideas to paper. The strategy consisted of creating several concepts done individually by each team member and discussing them frequently via meetings. Each concept is drawn from several perspective to get a 360° view and to have space for more details. The plan was to vote the best concept of one team member but in the end, it was a combination of several concepts and ideas from every team member which created the final design. This meeting of ideas and thoughts of each team member was present all the time while working out the concepts which was a pleasant experience.
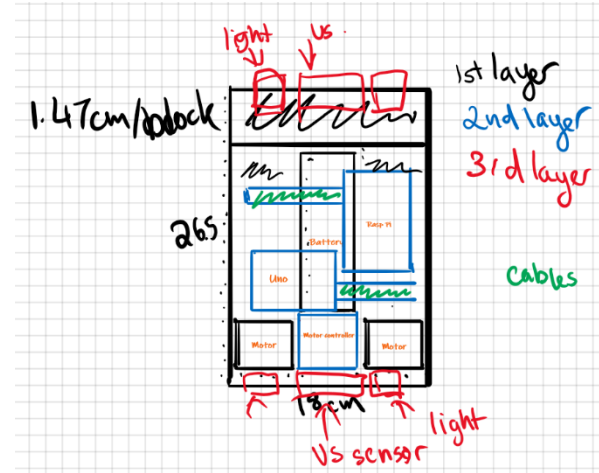
The concepts show design ideas of several parts like the tracks, tires, body, crane, steering mechanism, compartment, and place of sensors.

For the team it was important to make the main dimensions clear from beginning. Considering that the Lifefrog needs to have the ability to move on water it was quiet challenging to find a simple and working solution for that. For the concepts, the team thought about having a wind propeller on top like a hovercraft or just a propeller under the water. To make the propeller under water less vulnerable it came up the idea to put it in a tube. A design idea of a wheel was created that has extra paddles for more force in water which can open automatically due to the flow resistance of water. While the meetings it got more and more clear that it is crucial to make the design as simple as possible so the decision was made to not use a propeller because that would require additional motor, electronics, programming, and mechanics. The agreement was made to use the same drivetrain and tires or tracks to move on water as on land. The challenge regarding that decision was that it has to be sure that enough water is displaced to have a proper velocity on water but also enough stability and off-road capability on land.

E. *The final design*

First the plan was to have a crane attached on the Lifefrog, but the team concluded that it would be too complicated and hard to realize. Considering all pros and cons of tracks and tires the final decision was on using tracks because they are overall better for the use cases (better terrain capability) and easier to implement since no additional steering system is required.
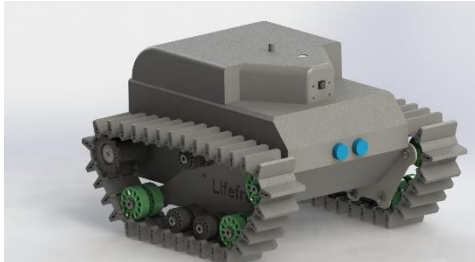
Fig. 8. Final 3D design - internal - side shot

Fig. 10. Final 3D design - side shot

Fig. 9. Final 3D design - internal - side shot

Fig. 11. Final 3D design - side shot

The final design is a tank like looking vehicle with tracks and ultrasonic sensors at the back and front and a camera at the top. The first made dimensions were 26.5cm*10cm*18cm which we adjusted later while the 3D modelling phase. The Lifefrog got separated roughly into 3 layers to also show what's inside. The first layer is the main body/hull plus the battery and motors attached on it at the bottom of the body to keep the center of gravity as low as possible since the motors and battery are the heaviest parts. Keeping the center of gravity low is important for stability, especially in cornering. The second layer is mainly the brain of the Lifefrog with a Raspberry Pi, an Arduino Uno and a motor controller attached on a mounting platform directly over the battery and motors. Also, this layer is the most protected one since it is in the middle which makes sense because the mentioned parts are the most vulnerable ones. The third layer or top layer consists of the compartment, camera, and microphone. With respect to off-road capability, the front is raised to ensure that the tracks have earlier contact to upcoming terrain than the body. Mentioning the tracks, they are designed with long spikes and enough space between the spikes so that enough water gets displaced. If this in real will not work, an alternative design solution was figured out that is easy to attach and would ensure enough force to move on water properly. The talk is of a wheel with long paddles which could be glued on the top back sprocket.
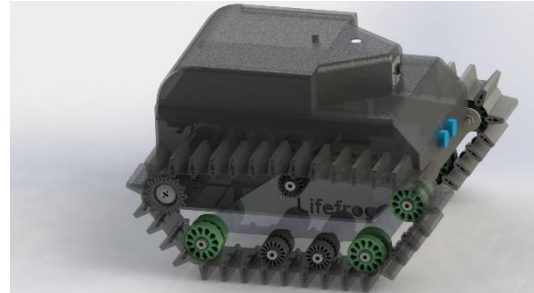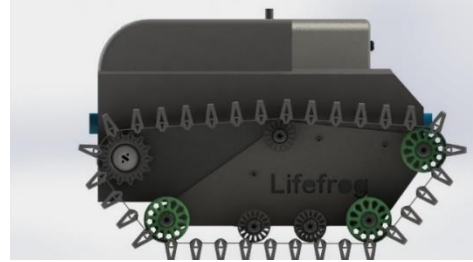
F. 3D Modelling with Solidworks

(Chen, Karapoulatidis)

The 3D modelling phase started with the division of the "to be modelled and printed" parts and distribution of them to the team members. Starting with the body parts which are the "hull" and the "mounting platform", the "hull" has several holes for the motors, ultrasonic sensors, and the holder for the tracks. Additionally, some ribs were added to increase the stability. To attach the "compartment" and the "hood" (which will be explained more detailed later) at the top of the hull the decision was made on using a dove tail. This is a flaring tenon and a mortise into which it fits tightly making an interlocking joint between two pieces. The advantage of a dove tail is that it is simple to print, that it is tight, stable and comes without extra mechanics for locking. The dimensions of the hull are 24.9cm*16cm*10cm.

The mounting platform is modelled in that way that it fits perfectly on the battery and motors and has all needed holes and pillars for mounting the second layer on it.

The top layer consists of two main parts - the hood and the compartment on the rear. There are two electronic devices installed inside the hood. One is a Raspberry Pi camera which captures visual data from the surrounding environment, and the other one is a microphone for communication. Only the lens of the Raspberry Pi camera pokes through the hood, the rest of it is hidden for waterproofing. The compartment is simply a curvy shell that covers the upper back of the vehicle. In order to
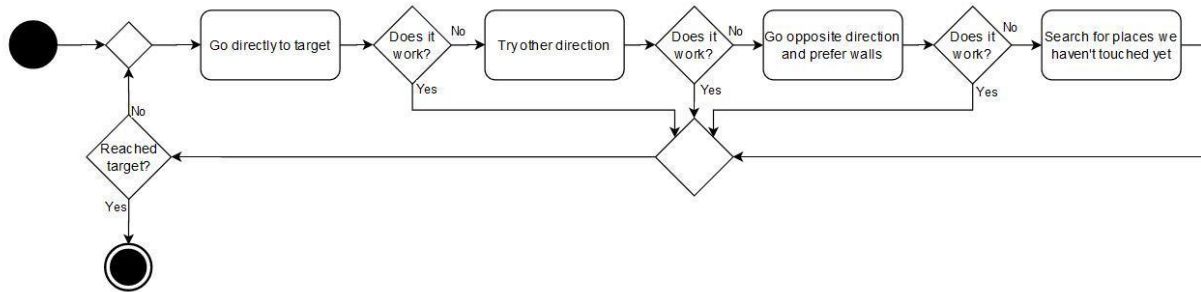
Fig. 12. Pathfinder v.1 activity diagram

install the compartment and hood firmly onto the body of Lifefrog in an aesthetic way, the dovetail joint was chosen over screws and nails for the design.

A dovetail joint is a joinery technique commonly used in carpentry. In Lifefrog, the dovetail consists of pins and tails. Pins and tails look like a trapezoid which is cut in half, and they match perfectly when the dovetail is properly installed. To put on the hood and compartment, one can simply slide them into the hole reserved for the dovetail.

VII. Simulation

A. Pathfinder v.1 (Stratmann)

For the first three simulation tasks, a code was written that can solve the problem by knowing the coordinates of the target but not knowing which way to go and therefore searches a path that leads to the target.

For that, a four-stage plan was created. The lower stages may be faster, but they are worse in solving problems. The higher stages can solve problems quite well and break out of dead ends, but in regard to getting closer to the target, it is more or less counterproductive. Therefore, it always starts with the lowest stage, so it can be assured that it is taking the shortest path.

1) Initialization
First thing to do inside the code is to copy the one-dimensional world twice and convert them into two-dimensional worlds. One world array is used for markings and a second world array is used for resetting the first world array and for checking the untouched world, without markings.

The second and last thing to do is to save the coordinates of the target, the home base, and the robot itself.

2) State Zero
To get to the target in the most direct way, the x and y coordinates of the target and robot are compared. The

code will then reduce the magnitude of the largest distance.

But before actually doing that, it will check the potential new spot for '*' (rocks), 'F' (footprints) and '#' (walls). If the new spot is neither of those, it will move to that spot and leave an 'F' on his old spot. But in case there is an obstacle, it will pass the responsibility to the next state (State One).

3) State One
If the robot got to this state, it is known that the most direct path has failed. But only one axis is checked to reduce the distance to the target. What if the code is trying to reduce the distance of the other axis? In that way the target is still followed, and it could solve the problem of state zero.

So as in state zero, the next move is planned and checked for rocks, walls, and footprints. If everything is fine, it moves, leaves a 'F' behind, and gives responsibility back to state zero. But if it fails again, it passes further to state two.

4) State Two
At this point, it is known that the distance to the target cannot be reduced. This means, it can only increase the distance, which leaves it with two potential directions. To determine which direction it takes, the surroundings of the potential spots is evaluated. Whether there is a wall or a rock, the code will prefer that spot by a higher rating. If there is not anything but empty spaces and maybe some footprints, the spot will only gain a medium rating. That improves efficiency of the robot, as without a rating system the robot would waste a lot of time in a room with a small opening by going "zig zag" from one side to the other side, filling it with footprints, just to find an opening which most likely is not in the middle of the room but near the wall. But all of this will not help if that potential spot has an obstacle on it ('#','*', 'F'). In that case, it will pass further to state three. Otherwise, it will
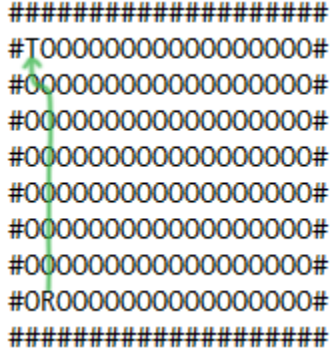
```
####################
#T0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0000000000000000#
#OR0000000000000000#
####################
```

Fig. 13. Working theory of state 0 (green line)

```
####################
#T0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0000000000000000#
#O0#0000000000000#
#O00000000000000#
#O0000000000000000#
#OR0000000000000000#
####################
```

Fig. 14. Working theory of state 1 (yellow line)

```
####################
#T0000000000000000#
#O0000000000000000#
#O00000000000000000#
#O00000000000000000#
###O00000000000000#
#O00000000000000#
#OR0000000000000000#
#O000000000000000#
####################
```

Fig. 15. Working theory of state 2 (orange line)

```
####################
#T00000000000000000#
#O00000000000000000#
#O00000000000000000#
#O00000000000000000#
###O00000000000000#
#O#00000000000000#
#R#00000000000000#
#O00000000000000#
####################
```
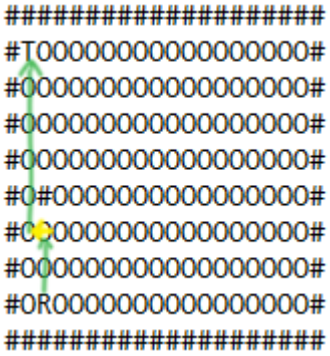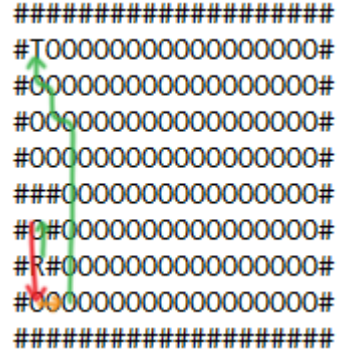
Fig. 16. Working theory of state 3 (red line)

take the spot with the highest rating, leave a 'F' behind, and pass back to state zero.

5) State Three

In this last state, a point is reached where the robot is stuck. It is surrounded by walls, rocks, and footprints. So, the only way to break free is to go back and move on footprints, which was avoided by all previous states.

If the code was not in this state last turn, it does some first initial steps. The robot turns left when it sees a wall in front of it. It will keep the previous direction when it faces a footstep. After that, it will not change direction unless it encounters an obstacle ('#', '*'). When that happens, it turns left. But in addition, it would like to avoid going back, as that could lead to an infinite loop. Instead, it is trying to use a different route, if possible. And as in every state before, it places a footprint after it moved on and let state zero try to progress.

6) Check For Water

At the end of the code, before it actually returns the direction, it checks the untouched copy of the world for water ('~'). If the code plans to go onto a water spot, coming from a land spot and vice versa, it returns the
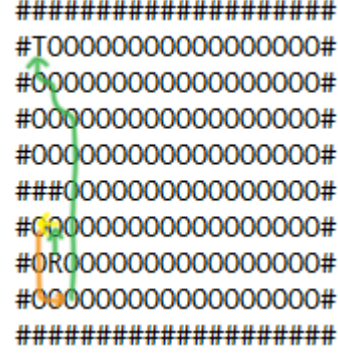
number for transition. When it does that, it also needs to reset the last turn, as it has not moved this turn.

7) Return To Base

After the robot reached the target, the code clears the world from footsteps with the help from the untouched world array and switch the target coordinates with the home base coordinates, as this is the new target. From now on, the code works the same way as it did with the initial target.

B. Pathfinder v.2 (Nguyen)

In reality, there are situations where the target is not known and there is no information on the surroundings due to the impact of the catastrophe and the robot has to be sent in to chart a safe route to the target. To simulate this, the vision of the robot is limited only to its surroundings, and it would not know where the target is. In addition, the robot would not know the map directly and it would have to chart the map on its own as it is moving around.

To accomplish this task, the robot first initializes a blank map with '_' denoted as an undiscovered spot, and places itself in the middle. This means that no matter where the robot is positioned in the world, it can chart the map. The map is 40x40 which is sufficient for the tasks
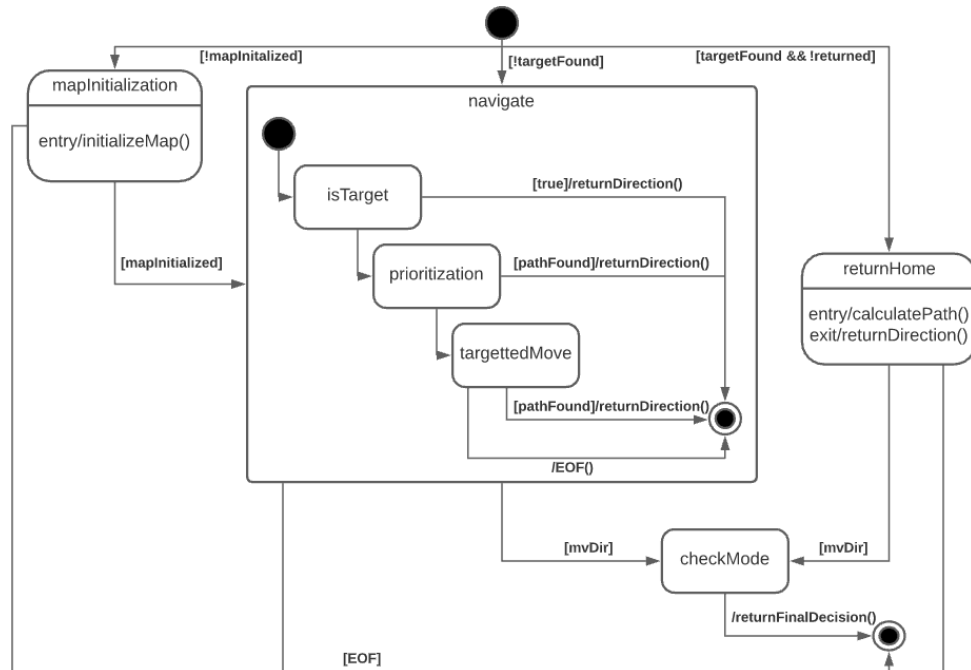
Fig. 17. State machine of pathfinder v.2

at hand. To chart a bigger world, simply increasing the sizes be increased. Once the map is done initializing, the robot would look around itself and save whatever it sees. This initialization is only done on ce for every map.

Subsequently, the robot would start wandering around the world. This is done in two phases, a searching phase, and a return phase.

1) Searching phase

The searching phase has 3 states. The first state is where the robot looks at its surroundings for the target. If the target is not adjacent to the robot, the second state would activate where the robot would move in a sequence to cover as much land as possible. The robot would try to move south first, if that is not possible then west, north, and lastly east with the condition that the cell that they are moving to is an empty spot and that empty spot is adjacent to an uncharted area. This condition ensures that the robot will not just wander around randomly. This state also ignores breakable walls to cover as much area as possible before deciding to break a wall.

Once this condition is no longer achievable by any of the adjacent cells to the robot, the robot would scan its map to find a traversable cell or a breakable wall that is adjacent to an uncharted spot and it would use the previous iteration code to move towards it. This phase is repeated until the target is reached.

2) Returning phase

This phase integrates the last iteration to navigate back to the origin. At this point, the robot has enough information to simply calculate the return path. The method of travel is the same as the last iteration.

Finally, before the move is executed, like the last iteration, the robot would check for the land type. If it is land or water, it would check its current driving mode and change it if it is inappropriate; if the destination cell were a wall, it would destroy the wall in the appropriate direction. The state machine diagram of the pathfinder version 2 can be seen in figure 17.

VIII.   A* Algorithm

For the last task, an additional requirement of energy efficiency was added. While the previous iterations are sufficient to accomplish the task, retrieving the target and returning to base, it is not the most efficient method out there. To ensure utmost efficiency, A* algorithm was employed instead. However, this algorithm requires the target to be known, unlike the previous pathfinder algorithm.

A* built upon the foundation of Dijkstra's algorithm to improve it. Like Dijkstra's, A* algorithm calculates the most efficient path using nodes and by considering the cost to go from one node to another node. However, it also considers the heuristic cost, the distance between
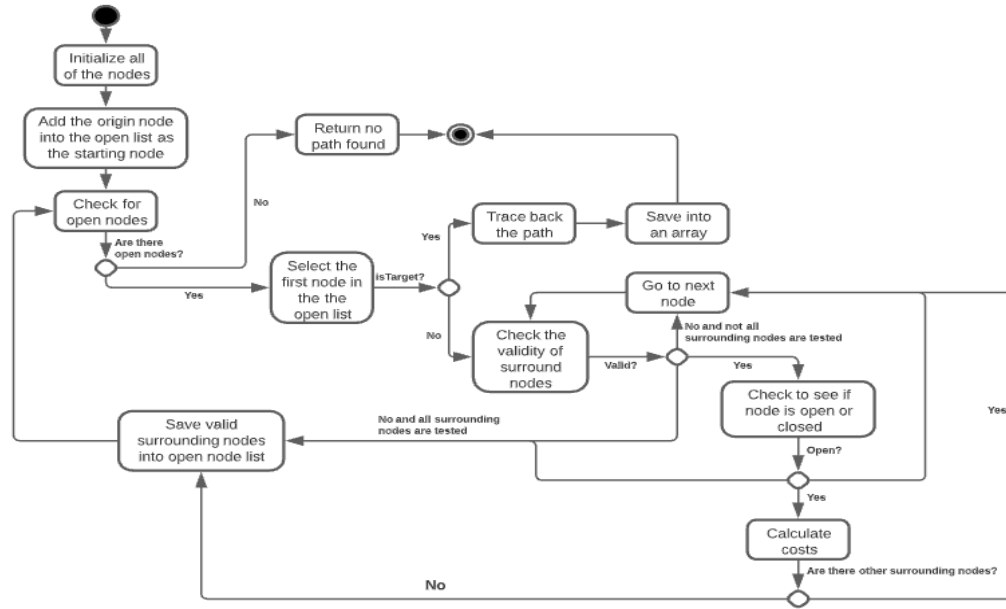
Fig.18. Activity diagram of the A* algorithm

each node to the destination node, which allows the cost to be even more refined. The algorithm always tests the node with the lowest cost first which makes the algorithm less memory and computationally demanding compared to Dijkstra's as fewer nodes are needed to be tested because of a more refined cost. The sum of these two costs is the final value for path calculation [2].

Looking at the definition above, it could be seen that A* heavily reliant on nodes and since each of these nodes are considered as a separate entity, this makes A* an object orientated algorithm. Unfortunately, C is not an object orientated coding language and does not support classes natively, but structure can be used as an alternative. Each node contains the critical information to calculate the best path with the parameter 'g' being the local cost, 'h' being the heuristics cost, and 'f' being the final sum of the two costs. The node also stores the land type of each node to help calculate the local cost. Lastly, to trace the path, the information on the parent node and the travel direction from the parent node to the node is also stored.

To save computation power, the code is only run once, and the path is saved within an array. Figure 18 explains the progression of the algorithm. The code works by initially copying the first node that needs to be tested from a list of open nodes to a temporary variable. This node would then be removed from the list so that it would not be tested again, which would be redundant. Using the temporary variable, the coordinate information is extracted to test the adjacent nodes. The code would

then check to see if the adjacent node is valid, and then check to see if it is the target. If the node is valid but not the target, the algorithm would calculate the cost to move to that adjacent node.

The Euclidean Distance Heuristic formula is used to calculate the distance between the adjacent node and the target node. The local cost is calculated by comparing the land type of the adjacent node to the node being tested. If the two have the same land type, the cost would be the lowest; if the two have different land types, the driving mode must be changed and thus the cost would be moderately high; if the adjacent node were a breakable wall, it would have to be broken before it is possible to be traversed through and thus the cost would be the highest. For the task at hand, it is found that the cost of 1.0, 1.3 and 1.7 respectively is the most appropriate. It is sufficient to ensure that the robot will not go towards a different land type or break walls if it does not have to, while it can also bypass some niche traps that are present in some maps. An example maps 9 where it is better to go around the wall instead of breaking the wall if only one way is considered with 110 and 120 energies, respectively. However, when considering the two-way cost, it is better to break the wall as it would create a shorter path to return later while going around the wall would double the one-way cost. The total costs are 170 for breaking the wall and 220 for going around the wall. The main cause of this complication is because the algorithm is currently modelled for one direction travel and is adapted for two-way travel. Another notable thing

is that if the cost were to be higher, the algorithm as it stands can act somewhat erratically. Alternate values of 1, 3 and 7 were tested. However, using these values, the algorithm is sometimes too scared of breaking walls or moving into different land types. It is unknown whether the cost can be tuned further but from tests, the current values produce the most efficient outcome for every task.

Once the costs are calculated, the node is then saved into the list of open nodes to be tested later. After this is done for all adjacent nodes, the algorithm would then sort through all the open nodes and bring the node with the lowest cost to the front to be tested first. This process would then be repeated until there are no open nodes available, and the target is not reached, which means that there is no possible path, or the target is found, which would invoke the drawPath function to trace back the path and save it into the path array to be used later. The method to switch driving mode and destroy walls is like the previous iterations.

IX.   Conclusion (Gönül)

With the hopes of saving lives in the future, the project has been successfully completed, despite the scarcity of time. Initial diagrams and lists of requirements led to sketches and plans, which later resulted in the first prototype and an efficient pathfinding program. As for the skills and experience gained in the programming exercises, they can be expected to be implemented to the prototype to create real time mapping and pathfinding.

Future of Lifefrog lies in its realization and optimization capabilities. With regards to the model, stronger materials can be used to increase the durability, e.g., carbon fiber body and graphene tracks. Not only they provide a durable structure, but they also keep the required weight below the limit. Additionally, as mentioned, problems can arise if the algorithm only take in consideration of one direction travel so in the future, it is possible to further improving the algorithm by modelling it for two-way travel instead.

X.   References

[1] Lite Trax. 2021. Wheels vs Tracks: Advantages and Disadvantages - Lite Trax. [online] Available at: <https://litetrax.com/wheels-vs-tracks-advantages-disadvantages/> [Accessed 16 July 2021].

[2]"What is the A* algorithm?", Educative. [Online]. Available: https://www.educative.io/edpresso/what-is-the-a-star-algorithm. [Accessed: 16- Jul- 2021].

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

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.

Gönül, Üsame	Berlin, 16.07.2021	
Name, Vorname	Ort, Datum	Unterschrift
Last Name, First Name	Location, Date	Signature

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

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.

Phan, Thanh Long	Nha Trang, Vietnam, 16.07.2021	
Name, Vorname	Ort, Datum	Unterschrift
Last Name, First Name	Location, Date	Signature

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

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.

Nguyen, Vu Quang Tuong	Lippstadt, 16.07.2021	
Name, Vorname	Ort, Datum	Unterschrift
Last Name, First Name	Location, Date	Signature

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

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.
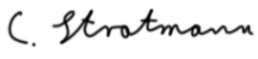
Chen, Shih	Lippstadt, 16.07.2021	*Chen Shih*
Name, Vorname	Ort, Datum	Unterschrift
Last Name, First Name	Location, Date	Signature

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

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.

Stratmann, Christian	Lippstadt, 16.07.2021	C. Stratmann
Name, Vorname	Ort, Datum	Unterschrift
Last Name, First Name	Location, Date	Signature

Eidesstattliche Erklärung

Hiermit bestätige ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen sowie Hilfsmittel genutzt habe. Alle Ausführungen, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind deutlich kenntlich gemacht. Außerdem versichere ich, dass die vorliegende Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Affidavit

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.

Karapoulatidis, Nikolaos	Erwitte, 16.07.2021	
Name, Vorname	Ort, Datum	Unterschrift
Last Name, First Name	Location, Date	Signature