## 1 Theory

Theory behind the design Arrowhead, SoS etc

## 2 Problem formulation

The problem solved by the software is to create a system according to the following high level functional requirements:

1. A user can construct a mission for the robot to do some meaningful work.
2. The mission can be sent to the robot and be executed in a way so the work is done.
3. A higher priority mission can preempt a lower priority mission.
4. Have a distributed, flexible and fault tolerant system.

## 3 Methods

To solve the problem described in section 2 the software from the group working on the snowblower in 2021 was first inspected. It was clear that it did not fullfil our requirements because it was not a distributed system but a monolith with some tightly coupled parts. Futhermore did it not have a flexible enough concept of a mission. Therefore was it decided to build a new system with a new design. This design is described in this section and the implementation is described in section 4.

### 3.1 Software system design

To be able to fulfill the first requirement a flexible mission construct is needed, a design for this construct is described in section 3.1.1. To then be able to execute this mission a system of systems (SoS) was designed and described in section 3.1.2.

#### 3.1.1 The mission construct

A mission is constructed as a series of simple tasks that the robot is to carry out to accomplish some work. For example if the robot have a snowblower accessory attached and we want it to carry out the work of clearing a area of snow a mission like this could be constructed:

1. SET ACCESSORY TILT: to 100%.
2. GOTO: starting point.
3. SET ACCESSORY TILT: to 0%.

4. ACCESSORY COMMAND: hold shute at $30^o$.
5. ACCESSORY COMMAND: start blower.
6. FOLLOW PATH: plow area path.
7. ACCESSORY COMMAND: stop hold shute.
8. ACCESSORY COMMAND: stop blower.

In this example mission, first the accessory is tilted up to lift it upp as much as possible. Then it goes to the starting point. After this it tilts down the accessory, tells the accessory to hold the shute at a angle so the snow is moved to a optimal spot and start the blower. After the accessory is set up it follows a path that covers the area. When it's done clearing the snow it turns of the parts of the accessory it turned on. This shows how a mission can be constructed using a list of tasks.

The type of tasks that can be used is given an overview in table 1.

| Task type | Description |
|---|---|
| GOTO | Move the robot to a specific GPS point. |
| FOLLOW PATH | Make the robot follow a path represented as a list of GPS points. |
| WAIT | Wait for a set amount of time. |
| ACCESSORY COMMAND | Send a command to the accessory. |
| SET ACCESSORY TILT | Tilt the accessory a percentage where 100% is as furthest from the ground and 0% is closest to the ground. |

Table 1: Overview of task types.

### 3.1.2 System of Systems design

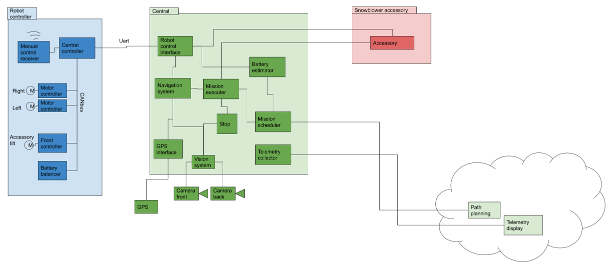

Figure 1: Overview of the software architecture.

The design of the software system is made up of different parts and are shown in figure 1. In figure 1 each system is symbolized by a square with a describing name and lines symbolize communication between systems. Some communication lines are omitted for clarity.

The design was divided into 4 main parts, each part has its own color in figure 1:

The safety critical part is shown in blue and controls the robot directly. This part is described in section (? ref to other Eriks part ?). The part communicate with the rest of the system over a UART [4] link with the Robot control interface system.

The non safety critical part is shown in the green big rectangle. This part handles everything needed to do to carry out a mission and sends the proper commands to either the robot controller or the accessory.

The red part is the accessory, it controls the accessory directly. The accessory is designed to be able to be any kind of accessory. In our test case it is a snowblower.

The last part in light green in the little cloud is running on a server somewhere. It is designed to be the interface between the robot and the world being able to get telemetry from the robot and send missions for it to do.

The green and red parts systems are arrowhead framework systems [1] running in a local cloud on the robot. By using the arrowhead framework these systems must each communicate with each other true well defined application programming interfaces (API). With these APIs a system can easily be changed out with another system implementing the same API or even more then one system implementing each implementing a part of the API. The arrowhead framework also provides late binding between systems, witch means that a system only need to know the address of a service it needs to use when it actually

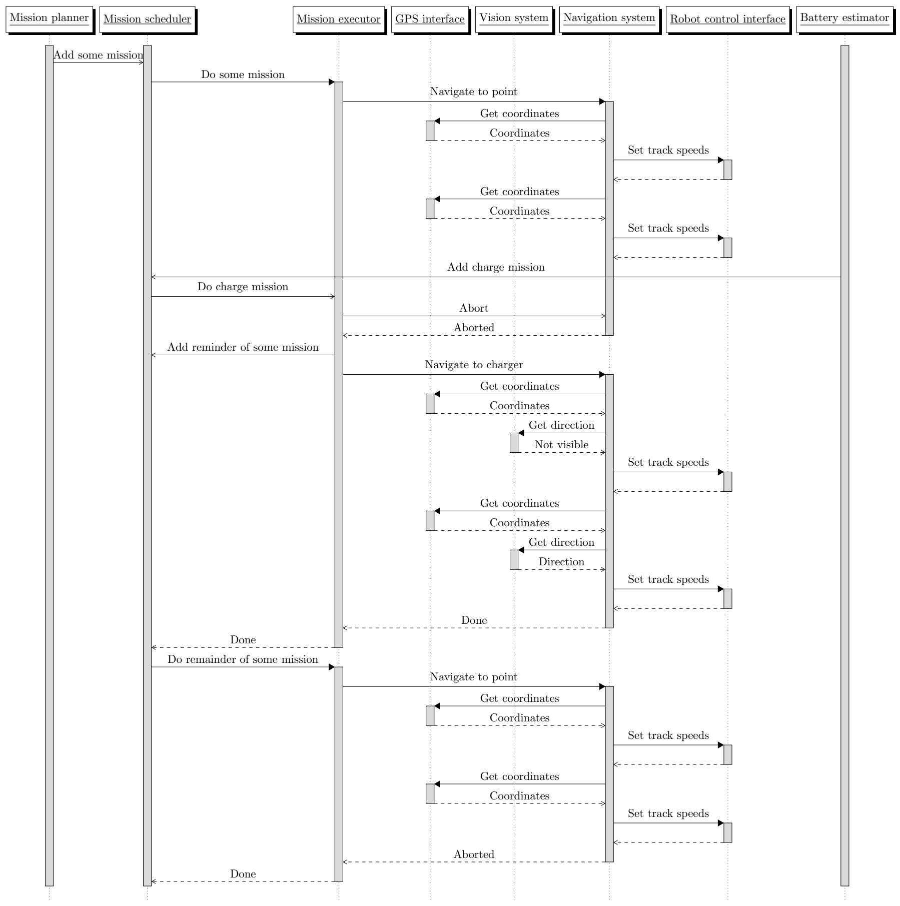

Figure 2: Sequence diagram for snowblower robot systems executing a mission and then interrupt it with a charge mission.

want to use it at runtime. To facilitate this late binding each arrowhead local cloud needs to have a set of core systems [5].

The systems in the cloud down to the right in figure 1 is run in a separate arrowhead local cloud with it's own set of core systems. To be able to communicate with the other local cloud the gateway optional core system [6] was used.

To show how the systems may work together a sequence diagram is shown in figure 2. In the sequence diagram the robot first do a mission where it navigates to a point and is preempted by a higher priority mission to charge the robot. The way preemption is handled is that the mission executor when it's get a new mission will construct a new mission that will carry out the work left on the current mission and send this new mission to the mission scheduler.

## 4 Result

To validate the design described in section 3.1 it was implemented. The arrowhead systems where implemented in Java using a arrowhead application library [2]. For the core system of the arrowhead local cloud core-java-spring [3] was used. Due to time limitations was not the entire design implemented and only the local cloud on the robot was set up. Table (? table ref ?) shows the status of completeness for each system implementation.

## 5 Discussion

How did the software design and implementation meet the design goals?

## 6 Conclusion

What was archived? Further improvements? hej

## References

[1] Jerker Delsing et al. “The arrowhead framework architecture”. In: *IoT Automation.* CRC Press, 2017, pp. 79–124.

[2] eclipse-arrowhead. *application-library-java-spring.* Version 4.4.0.2. Jan. 5, 2022. URL: `https://github.com/eclipse-arrowhead/application-library-java-spring`.

[3] eclipse-arrowhead. *core-java-spring.* Version 4.6.0. Sept. 21, 2022. URL: `https://github.com/eclipse-arrowhead/core-java-spring`.

[4] Umakanta Nanda and Sushant Kumar Pattnaik. “Universal Asynchronous Receiver and Transmitter (UART)”. In: *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS).* Vol. 01. 2016, pp. 1–5. DOI: `10.1109/ICACCS.2016.7586376`.

[5] Pal Varga et al. “Making system of systems interoperable–The core components of the arrowhead framework”. In: *Journal of Network and Computer Applications* 81 (2017), pp. 85–95.

[6] Pál Varga and Csaba Hegedus. “Service interaction through gateways for inter-cloud collaboration within the arrowhead framework”. In: *5th IEEE WirelessVitae, Hyderabad, India* (2015).

## A Documentation