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. After this each system is described section 3.2 to 3.11.

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

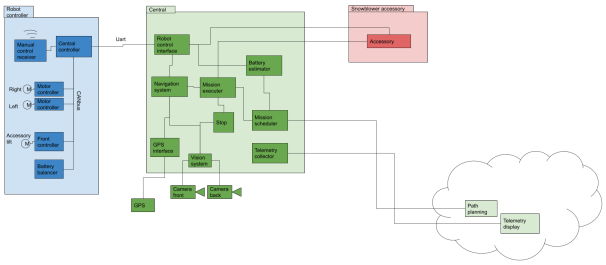

Figure 1: Overview of the software architecture.

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

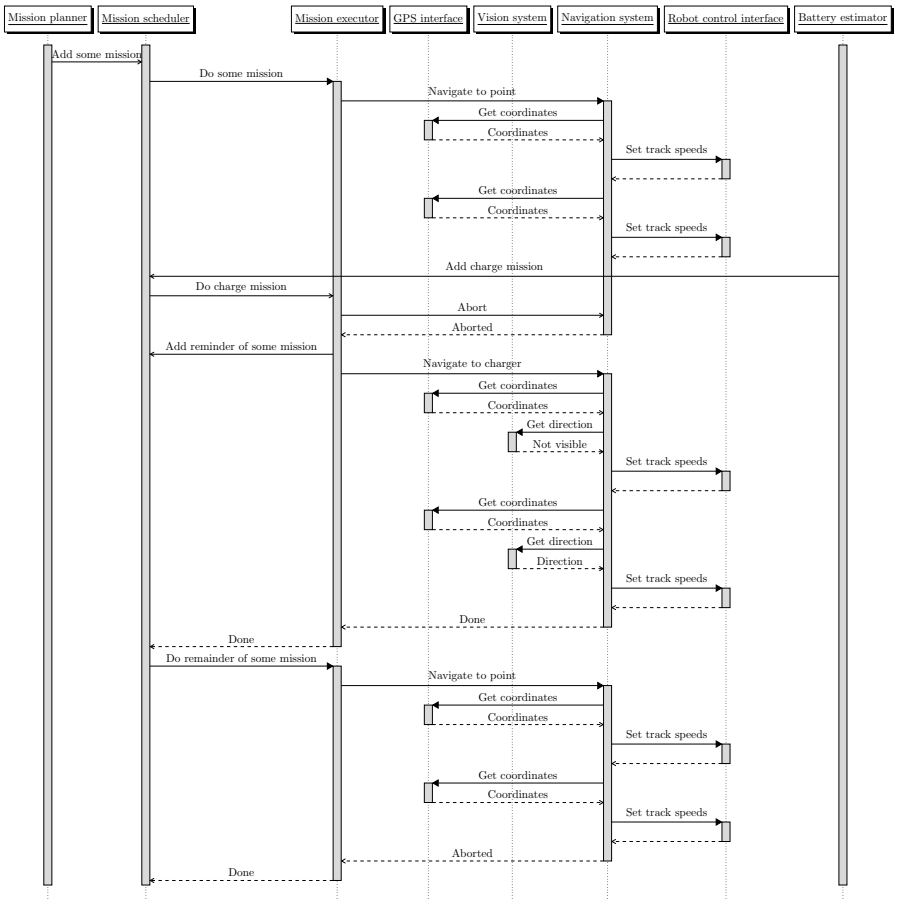

Figure 2: Sequence diagram for snowblower robot systems executing a mission and then interrupt it with a charge mission.

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.

3.2 Accesory control

The accessory control system takes accessory commands and carries them out by controlling the accessory. What accessory commands ara available depend on the accessory and each accessory type need its own accessory control. To be able to know what accessory is attached a get accessory type service is provided and to validate that the command can be interpreted by the specific accessory controller the accessory type is sent alongside each command.

3.3 Battery estimator

The battery estimator gets information battery from the robot control interface and estimates battery life left. When the battery life left is to short it can issue a high priority go charge mission if appropriate.

3.4 GPS interface

The GPS interface gets information from the GPS unit and makes it available to the other systems thru provided services.

3.5 Mission executor

The mission executor handles the execution of the missions. When its do mission service is consumed it will take the mission and for each task send the right service request with the right data. It can be looked upon as a translator between mission task and service request. It also sends a add mission request to the mission scheduler with a mission that will do the remaining work of a mission interrupted by another mission to enable preemption.

3.6 Mission scheduler

The mission scheduler as the name implies schedules the missions. It schedules the mission by highest priority first and if same priority oldest first. The function of the mission scheduler can be described by the state diagram shown in figure 3. When in the send mission state either the new mission or the first mission in the queue is sent to the mission executor.

3.7 Navigator

The navigator navigate the robot by getting position and heading from the GPS interface or a relative heading and distance from the Vision system and comparing it to where it should be send the appropriate track speeds to the robot control interface. It can receive either a GPS point to navigate to, a path represented by a list of GPS points to follow or a marker id to navigate to.

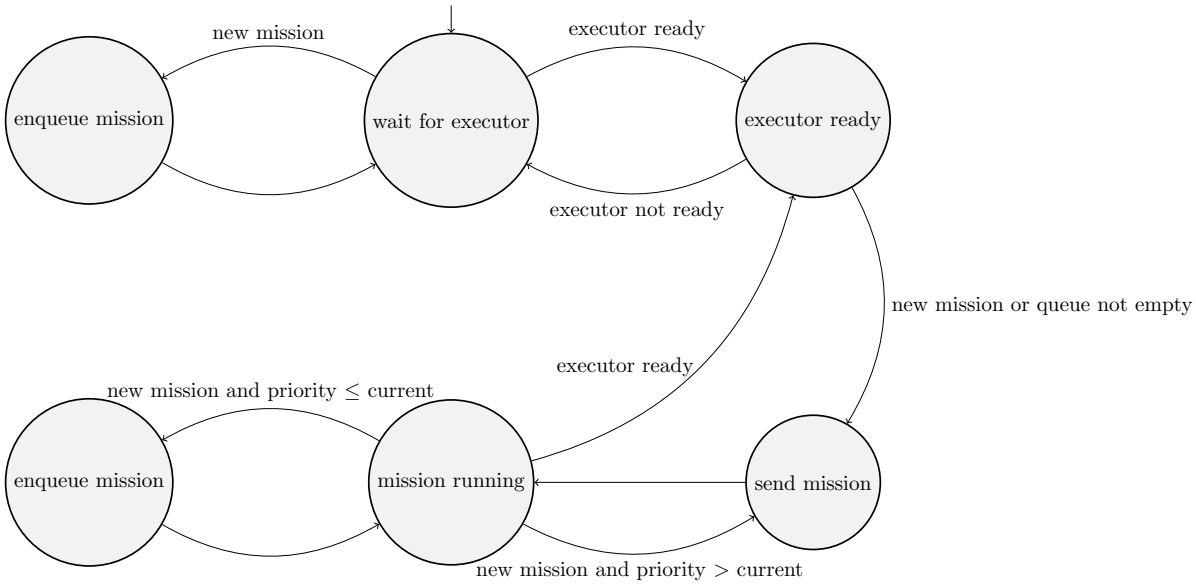

Figure 3: State diagram for mission scheduler.

3.8 Robot control interface

The robot control interface communicates with the safety critical system that directly controls the robot. It can receive requests to set the track speeds, tilt accessory etc.

3.9 Stopper system

The stopper system sends stop requests to all applicable systems when it receives a stop request.

3.10 Telemetry collector

The telemetry collector collects telemetry from most other system to relay it to the user via for example a web interface running on some server somewhere.

3.11 Vision system

The vision system is connected to stereo cameras. It uses the cameras to identify when something is to close in front of the robot and stop it by a request to the stopper system. Other then identifying obstacles it can also get the distance and bearing to a fiduciary marker. For more information about the camera system se section (? ref to Antons part ?).

System	Implementation status
Accessory control	Initial version written but not tested and integrated.
Battery estimator	Nothing done.
GPS interface	Done apart from telemetry.
Mission executor	Done for follow path and go to point tasks, no telemetry or mission reconstruction on interruption.
Mission scheduler	Done apart from telemetry.
Navigator	Done for go to point and follow path.
Robot control interface	Done for set track speeds.
Stopper system	Nothing done.
Telemetry collector	Nothing done.
Vision system	Not fully tested and implemented with arrowhead.

Table 2: Implementation status of all the systems.

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 2 shows the status of completeness for each system implementation. To test the parts of the system that where done a mission that started with a go to point task then a follow path task and finally another go to point task was sent to the mission scheduler. After this the robot first navigated to the point then followed the path and finally navigated to the point showing that the mission scheduler, mission executor, navigator, GPS interface and robot control interface for this case worked as intended.

5 Discussion

As shown in section 4 a mission was able to be constructed that did meaningful work in this case moving the robot around so the first and second requirement was met. For the third requirement the design was done to enable thru the mission executor adding a new mission for the work left to do in the interrupted mission to the mission scheduler. This was not implemented as seen in table 2 so is to be seen how well this works in practice. When it comes to the fourth requirement it is hard to say wether or not the design fulfill it. Altho the design is clearly distributed and parts of the distributed system was shown to work in practice. Furthermore was the system made to be fault tolerant by separating the system into safety critical and non safety critical parts. By doing this the safety critical system can ignore the instruction from the non safety critical system if an operator turn a switch on the remote. The flexible part of the requirement is somewhat fulfilled by having a system of systems with well

defined APIs whish enables a system to be changed out to another system that implement the same API.

6 Conclusion

To conclude was the requirements met even thou many features are missing from the implementation.

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).