

Timon Rupelt

Project Report
**Deployment of Reed
Sensors for Monitoring
Flood Protection Gates**

Faculty of Engineering and Computer Science
Department of Computer Science

Supervising examiner: PROF. DR. THOMAS C. SCHMIDT

**HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG**
Hamburg University of Applied Sciences

Contents

1 Introduction	3
2 Hardware	4
2.1 Requirements for the Mounting	4
2.2 Sensor Mounting	5
2.3 Compute Hardware and Sensors	8
3 Software	9
3.1 Main Software Components	11
3.2 Gate State	13
3.2.1 Total Agreement	15
3.2.2 Sequenced Agreement	16
3.2.3 Majority Voting	16
3.2.4 Weighted Voting	16
3.3 Ticket System	17
3.4 State Reporting	18
4 Deployment	20
4.1 Observations	22
5 Conclusion and Outlook	24
References	26

***CBOR* – Concise Binary Object Representation**

Abstract In the context of the RESCUE-MATE research project, which aims to improve situational awareness in storm flood scenarios, it is necessary to equip floodgates with sensors that enable remote monitoring of their state. This report showcases the development of a prototype system utilizing reed sensors for this application. Four approaches with varying strictness allow evaluating multiple and possibly inconsistent values to produce a single reliable gate state. Purpose-designed circuitry and mounting hardware allow the sensors and magnets to be mounted on a real floodgate. The deployment saw the use of three sensors configured with a sequential majority vote for state determination on a swing gate. The system successfully detected gate closures and openings under real-world conditions but failed to deliver long-term data due to a software bug and a failing logging system.

Keywords: reed sensor, reed switches, proximity sensors, flood gate monitoring, riot, rescue-mate, LoRa, LoRaWAN

1 Introduction

This project aims to create a prototype system for monitoring the state of floodgates for a deployment in the port of Hamburg. This is part of the research project RESCUE-MATE [1], which aims to investigate the creation of a platform for dynamic situation overview for emergency situations, such as flooding. It should integrate various data sources from environmental sensors to social media, to improve the understanding of a situation for emergency services and other personnel. As part of this, the “Resilient Sensor Networks for Dike Monitoring and Emergency Communication” subproject undertaken by the INET research group at HAW Hamburg, is to increase the availability of information on dikes and floodgates.

Floodgates of varying sizes and closing mechanisms would need to be equipped with sensors, to give reliable information about their state. In this context, the previous works [2] and [3] have investigated the use of reed sensors for this monitoring task in laboratory environments. Reed sensors or reed switches are electromagnetic switches [4]. Applying or removing an external magnetic field forces two internal contacts to open or close. This mechanism can be used as a simple proximity sensor.

Based on the previous works, this work takes the next step to develop a prototype system and deploys it on-site on a real floodgate in a real environment. This system uses reed sensors to detect changes in the gate position, determine its new state, and report this state to a data sink. It needs to be extract reliable information about the gate under real, uncontrollable and changing conditions. To this end, software was developed to record sensor values of an configurable number of sensors and evaluate them to determine a single gate state. Multiple options for handling inconsistent sensor values and varying degrees of strictness were explored and implemented. The deployment on a real floodgate, required the design of mountings and housings for the sensors for attachment to the gate and protection from the weather. The deployment itself showed success of the core behavior of the system but failed to deliver long-term data due to bugs and a failing logging system.

The requirements for the hardware and its implementation are discussed in Section 2. The developed software is described in Section 3. Section 4 describes the deployment and the observations about the behavior of the system. Finally, Section 5 draws conclusions and outlines the next steps regarding future enhancements and improvements.

2 Hardware

The deployment requires computational hardware and specifically designed housings and mounts for the sensors. This section describes the design process for the parts that will be produced using 3D printing later on, as well as the microcontroller, which serves as the core of the system and the used sensors.

2.1 Requirements for the Mounting

The previous work has shown that reed sensors have great reliability in a controlled environment and for a high number of activation cycles [3]. But in a real environment with uncontrollable conditions, it is hard to trust a single sensor.

For this reason, three reed sensors are to be used in this prototype, for verification and redundancy. Using just a single sensor does not allow verifying the correctness of a sensor in case of sensor failure. Using two sensors poses the challenge of determining which sensor is the correct one. Three sensors are the minimum number of sensors, allowing to handle these challenges.

The sensors are to be mounted on the frame in a row along the gate on the non-hinge side. The counter magnets are mounted on the gate itself, specifically on the non-hinge end of the gate, opposite of the sensors on the frame when the gate is closed. As the end of the gate and the frame against which it closes are parallel, all the sensors would be triggered at roughly the same time (within a few milliseconds).

To add another option for verifying the correctness of the sensor outputs, the activation order can be utilized. The ‘correct’ order is determined by the physical placement of the sensors and the current ‘phase’ of operation, meaning closing or opening. The first sensor to activate in the opening phase, is the last to deactivate in the closing phase.

To achieve a more isolated and sequential triggering, the sensors should be placed so that each one is slightly offset backwards from the previous one, as visualized in Figure 1. This offset should be adjustable to allow for fine-tuning for the sensors activation range and to handle mounting restrictions. The opposing magnets should also be adjustable.

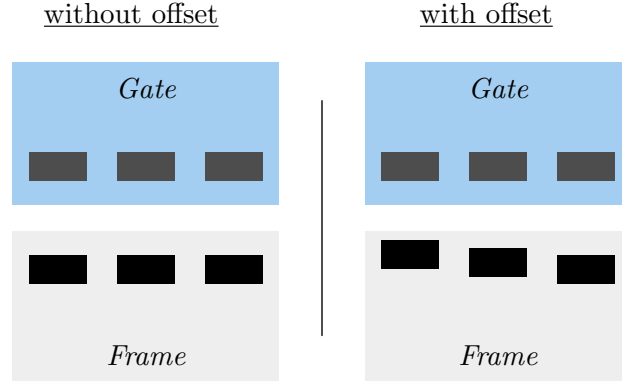


Figure 1: Sketch demonstrating the positioning offset of the sensors.

These requirements had to be met in the design of the mounting hardware for the sensors and magnets.

2.2 Sensor Mounting

Specific mounts were developed to attach the sensors and the magnets to the frame and the gate respectively. This means, allowing the fastening on the respective part of the gate or frame. At the same time the sensors and magnets need to be adjustable in their relative positing, as describe in Section 2.1.

This is achieved by using two separate parts: the housing of the sensors and magnets, and the mounts that attach the housing to the frame or gate. This way, when deploying the sensors on another gate, only the mount that attaches to the gate itself needs to be modified, not the housing.

The housing needs to fit all sensors and the hardware that enable external connections. The sensors are arranged in a row, 15 mm apart facing the edge of one of the longer sides. This spacing is determined by the restrictions imposed by the cables of the sensors, which need enough space to be routed away.

The movement of the sensors are restricted by guide rails on each side which only allows for forwards or backwards movement. To secure them in place they are pressed against two bars above them by using a screw. For this two smaller fasteners are *connected* with the sensors. The lower part allows the sensor to lie flat on the base surface. The upper part is wider than the sensor. This part will lift up when screwed against the sensor itself and press into the upper guiding bars. The pressure secures the sensor in place.

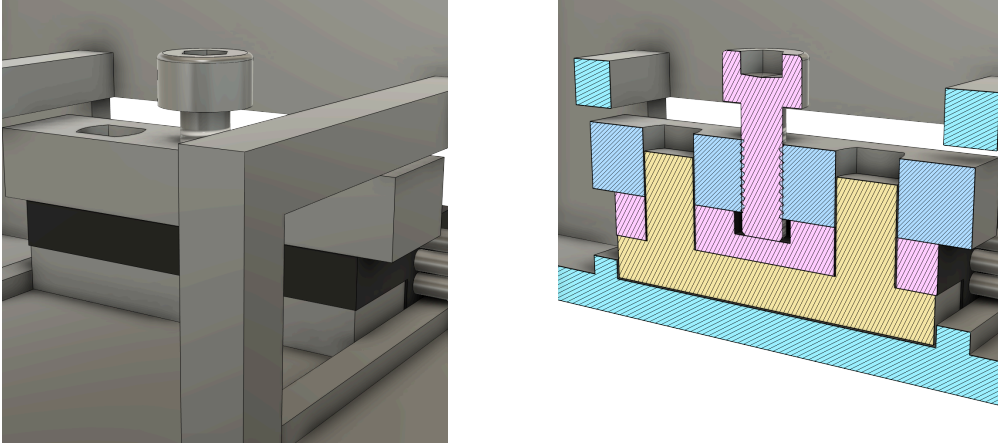


Figure 2: 3D model view of the sensor and the mechanism to secure the it. The normal view is on the left. A cross-section view of the sensor and the surrounding parts is on the right. **Cyan**: the housing, **pink**: the sensor and the screw, **yellow**: lower fastener part, **blue**: upper fastener part.

This mechanism is shown in Figure 2 with a normal view and a cross-section view of one sensor placed within the guiding rails with both fastener parts.

The sensor cables are connected to a circuit board via *JST-XH* connectors, which route the sensors to an 8-pin *Switchcraft EN3P8M* plug connector. This allows the sensors to be connected from the outside using a single shielded cable.

Multiple small *feet* with holes are spaced 20 mm apart on the outer edge of the housing. This allows the housing to be secured to the base mount using M3 screws.

Since the sensors and magnets have the same form factor, the housing is mostly the same, except for the plug and circuit board. Figure 3 shows 3D models of both housing variants.

To secure the housings on to the gate and frame, the mount needs to be able to attach to the gate or frame and allow the housing to be screwed on top. The frame mount is just a 22x8 cm plate with a grid of ca. 3 mm holes with 10 mm spacing. The plate will be attached to the frame with adhesive.

The mount for the gate (used for this deployment) is more complex, because there is less surface area, and it must not obstruct the closing of the gate. It is fitted to the shape of the end part of gate, so it can be put over it from above. It is secured with three screws on the side that are driven into the gate, holding it in place by applying pressure. To not interfere with the closing of the gate, a slice of the side wall towards the frame is left

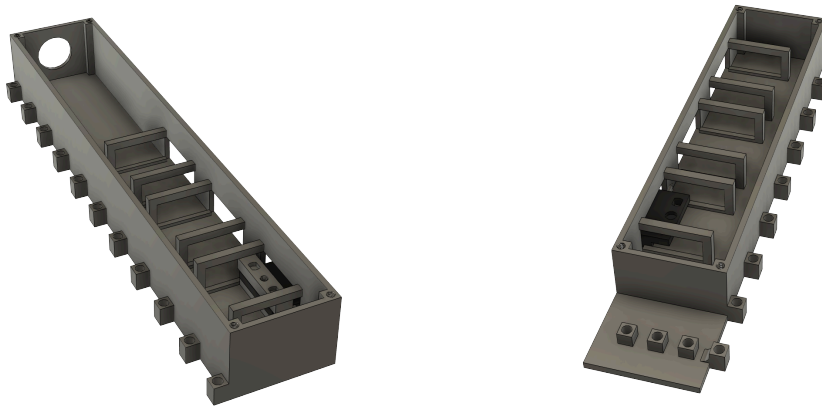


Figure 3: 3D models of the housing of the reed sensors (left) and the housing of the magnets (right).

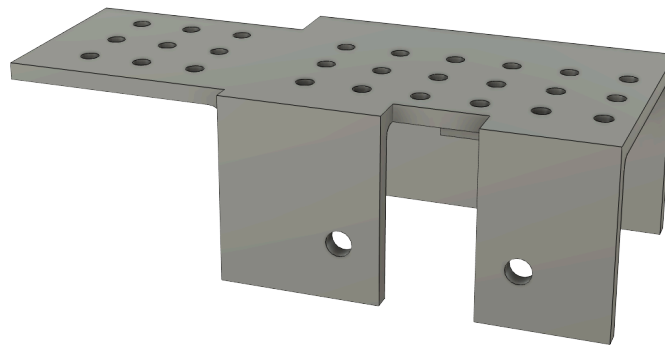


Figure 4: The 3D model of base mount for the frame.

open to allow the frame to press into the neoprene of the gate. This is necessary to seal it off completely. On the upper surface is the same grid as on the plate for the frame. The 3D model of the base mount is show in Figure 4.

While designing, the technical drawings of the gate have been used as a references as well as some own measurements. Unfortunately, some dimensions were not specified in

the drawings and some measurements were not as precise as needed. This lead to some problems during deployment described in Section 4.

2.3 Compute Hardware and Sensors

The core of the system is the *Adafruit Feather nRF52840 Sense*¹. The nRF52840 SoC is built around a 32-bit ARM Cortex-M4 CPU running at 64 MHz with 1MB Flash Memory and 256 KB RAM. Equipped with 21 GPIO pins, six of which are 12-bit ADC, as well as SPI and I2C support for all pins, it is well suited for this purpose of reading sensors in a low-power application. It has already been used in a previous work [3]. It is extended with an expansion board to add a RF69 LoRa radio (based on Semtech SX127x LoRa module) for remote communication.

The microcontroller is encased in a waterproof box with connectors of the same rating. These allow powering and communication via USB, as well as a connection to the reed sensors via the same 8-pin *Switchcraft EN3P8M* plug connector as described in Section 2.2.

The reed sensors used in this prototype are MS-324-4-3-0500² by PIC. These are *form C* reed switches, meaning it changes between two contacts when operated, resulting in two states (Normally-Open and Normally-Closed). They have been evaluated for their usability in this application as part of a previous work [3]. Laboratory tests showed generally consistent readings and reliability across many activation cycles. The wiring is shown in Figure 5.

¹<https://www.adafruit.com/product/4516#technical-details>

²<https://www.pic-gmbh.com/en/products/ms-324-4>



The received sensor values are then evaluated with the selected *multi-sensor mode*, defining the approach to determine a single gate state. These are different approaches to *data fusion* with varying strictness concerning inconsistent values and the dependency on order. These approaches are described in Section 3.2. The determined gate state and additional information are then reported via LoRaWAN to the remote endpoint. The data is encoded using the Concise Binary Object Representation (CBOR) data format as described in Section 3.4.

The implementation of these stages is visualized in an sequence diagram in Figure 6. Here, two sensor events arrive after which the timer expires triggering the state determination. A third event occurs during the determination phase resetting the timer again, but not triggering the determination yet.

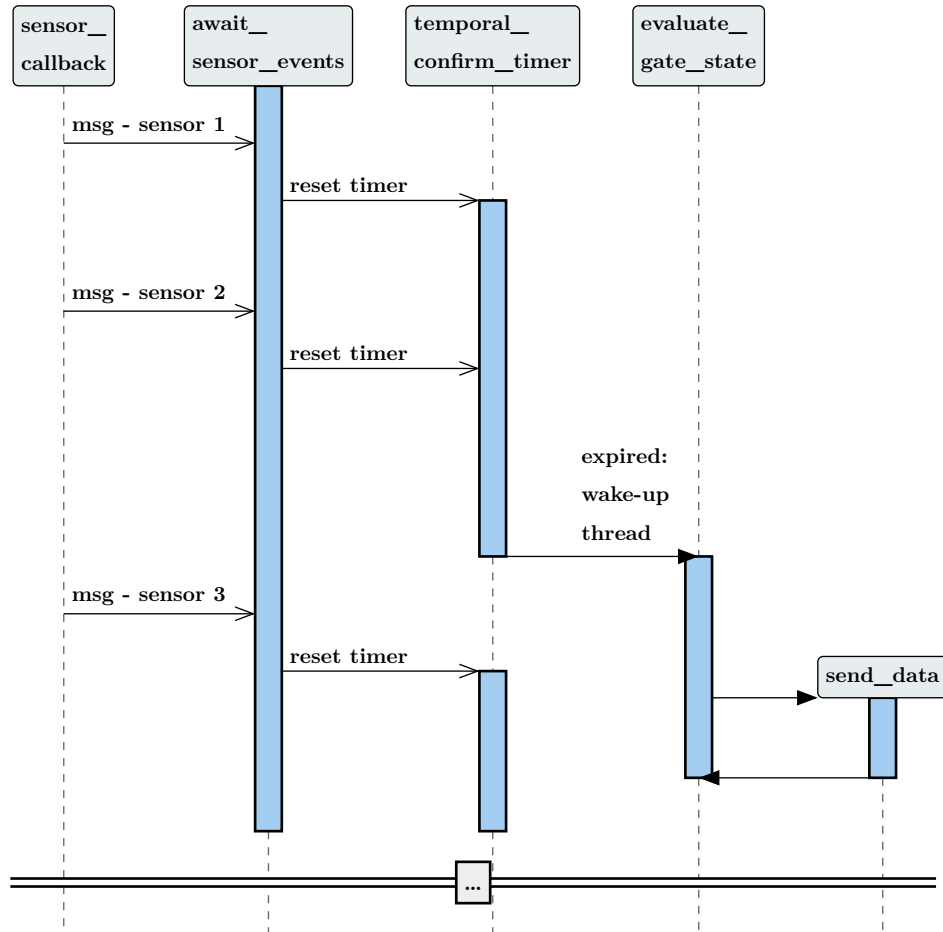


Figure 6: Sequence diagram visualizing the system's behavior from new events until sending the data to the data sink.

The developed software uses the real-time operating system RIOT [5] which is designed for resource-constrained embedded devices. It provides implementations for embedded operating system functionality, a variety of technologies and standards and supports a wide range of microcontrollers, sensors, and actuators.

The following sections present the software components used to implement the described behavior.

3.1 Main Software Components

The main software components and their relations are shown in class diagram in Figure 8.

A configured sensor receives a unique sensor id. The additional value id is assigned per unique value provided across all sensors. This is because some sensor can provide multiple values. This is the case with the reed sensors used here.

A sensor value is represented by a struct type `sensor_value_state_t` which is to abstract the underlying sensor. It contains three types which are used to identify which sensor it belongs to. The `sensor_type_t` is an enum type representing the type of the underlying supported sensor. The `sensor_id_t` and `value_id_t` are typedefs for different unsigned integer types. These three values are encoded into a single integer value using bit-shifting, used as a combined unique identifier in internal messages and for reporting (see Section 3.4). Regarding the reporting message, this will save space when using more than three sensor values, because fewer bytes are sent per sensor value. To minimize the size of the message even more, the integer types of `sensor_id_t` and `value_id_t` are configurable by defining the minimal number of bits required to uniquely identify all configured sensors. By default both are a `uint8_t` supporting up to 256 different sensors and values. A visual representation is shown in Figure 7 with 3 bits used for the sensor id and 4 bits for the sensor type and the value id.

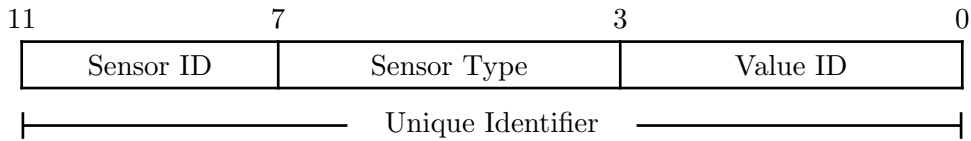


Figure 7: Bit field representation of the encoded sensor id (3 bit), sensor type (4 bit), value id (4 bit)

The current read sensor value is stored as a `uint32_t` providing enough space for larger values of sensors measuring distance as well. The `event_counter` stores how often a sensor

value has triggered. If this value exceeds a threshold, the value is masked, indicated by the `is_masked` flag. The `latest_arrive_ticket` is used to determine the order of arrival between values when evaluating the gate state. The ticket system is introduced in Section 3.3. The flag `is_out_of_sequence` is set during the determination phase if the value has arrived out of sequence, which is defined by the physical placement, the operational *phase* (*closing* or *opening*) and the selected multi-sensor mode.

The state of the gate is represented via a struct type `gate_state_t`. The type stores the determined position of the gate (open or closed) and all registered *sensor_value_states* used to determine that position. The configurable `multi_sensor_mode_t` specifies how the sensor values are evaluated to determine the current gate position. The possible modes differ in the way values are interpreted and if the order matters in which the values arrive. The basic modes are:

1. Total Agreement
2. Sequenced Agreement
3. Majority Voting
4. Weighted Voting

As opposed to `SEQUENCE`, `PARALLEL` indicates that order does not matter. This is the default for *Total Agreement*, *Majority Voting* and *Weighted Voting*. However it is also possible to combine these modes with the `SEQUENCE` dependency. The four modes are described later in the following section.

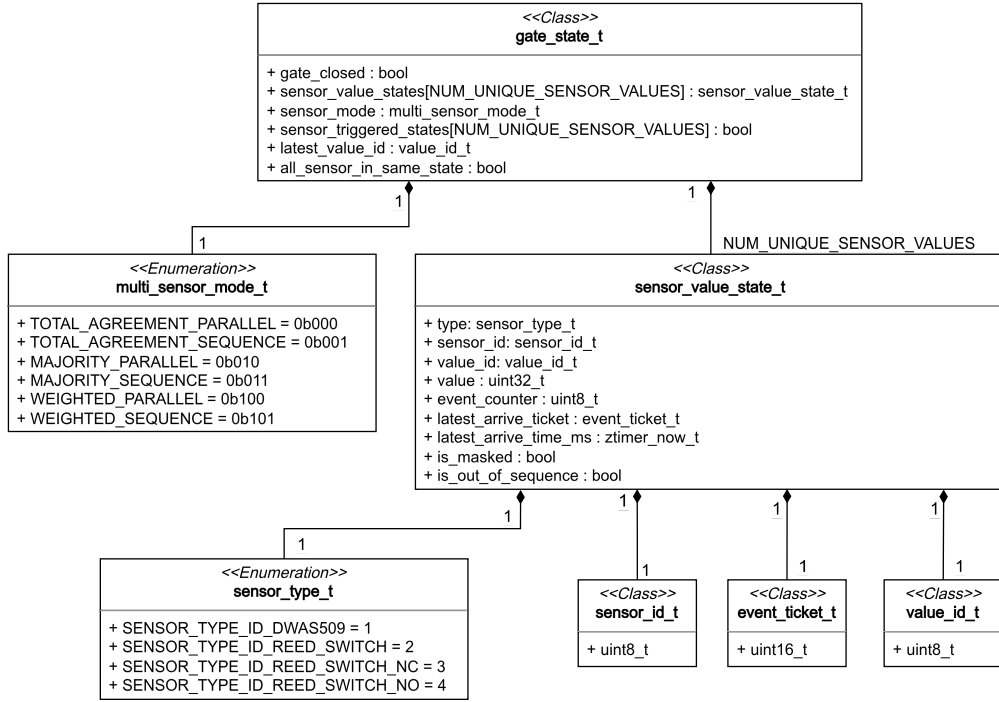


Figure 8: Abstract ‘class’ diagram of the main software components.

3.2 Gate State

The main routine `await_sensor_events` is waiting for new sensor events to arrive using a RIOT-OS message queue. When a sensor is activated, an interrupt is triggered, which results in the corresponding call of the respective `sensor_callback` function. The callback sends a message containing the encode sensor id, type and value id to the main routine. When the message arrives, the values corresponding instance of `sensor_value_state_t` is updated with the current value, the new ticket, etc. Additionally, the `temporal_confirm_timer` is reset. This will happen every time a new event arrives until the timer expires.

When multiple sensors are used and the gate is closed or opened, the sensors will trigger in short succession. The timer prevents the system to start the process of determining the gate state too early. Only when all sensors have arrived, the process should be started. However, it is not guaranteed that all sensors will trigger.

When the timer finally expires, a thread running the `evaluate_gate_state` routine is woken up. In this routine the current sensor values are evaluated depending on the chosen

multi-sensor mode to determine the current gate state. The activity diagram displayed in Figure 9 visualizes this process. The areas with a gray background are different depending on the chosen *multi-sensor mode*.

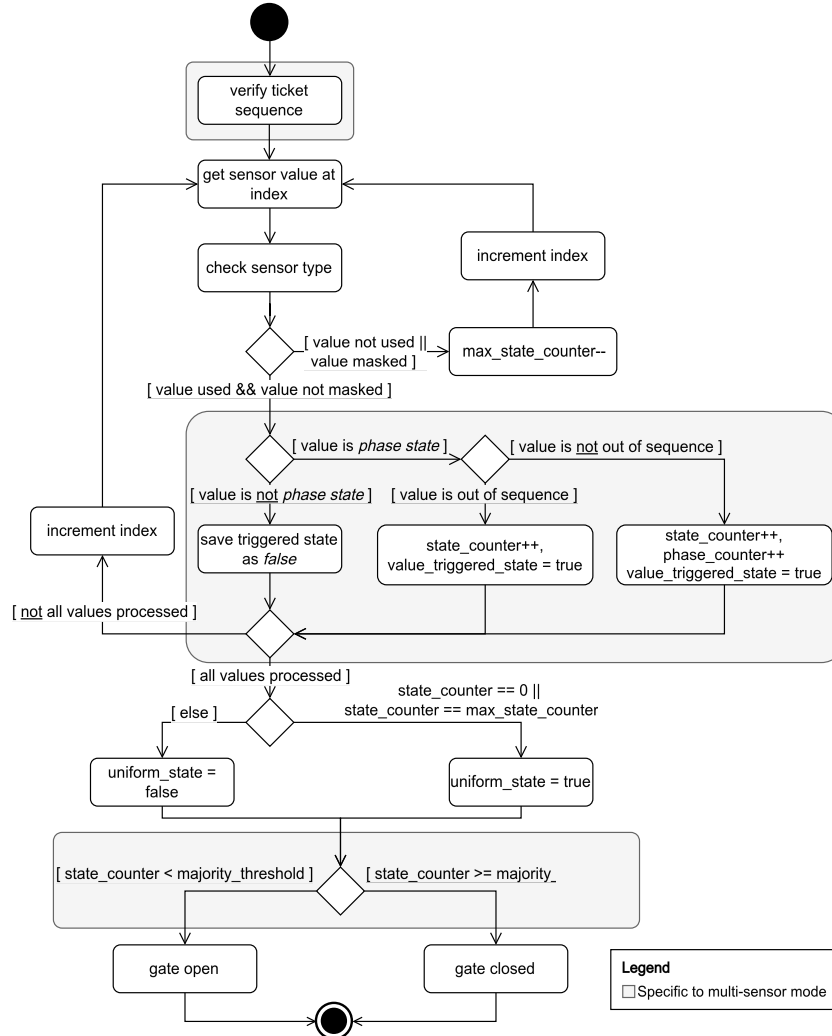


Figure 9: Activity diagram visualizing the *state determination* process performed in the `evaluate_gate_state` (see Figure 6) routine. The gray areas are specific to the configured multi-sensor mode. The diagram shows the behavior for Majority Voting or the specific variant of Total Agreement (with the majority set to the max number of values.)

All configured sensor values are evaluated to determine a single state. The initial step is to determine the current *phase* which is either *closing* or *opening*. This determines the ‘correct’ value to check against. When using a multi-sensor mode with the sequence requirement, the phase defines the order in which the sensor have to arrive. A sensor

value failing this check has its `out_of_sequence` flag set. When a sensor value is *masked*, its evaluation is skipped and it is excluded from the total of valid values, which is used to determine a uniform state, where all values are the same. The uniform state is needed to reset the global ticket value.

When a sensor value is not masked, it is verified with the approach of the select *multi-sensor mode*. There are four possible approaches for *data fusion* provided in this work. These approaches differ in their strictness concerning inconsistent values and the dependency on order. Depending on the phase of the gate (*opening* or *closing*), the sensor values also need to be interpreted in a different ways. These approaches are described in the following and visualized in Figure 10.

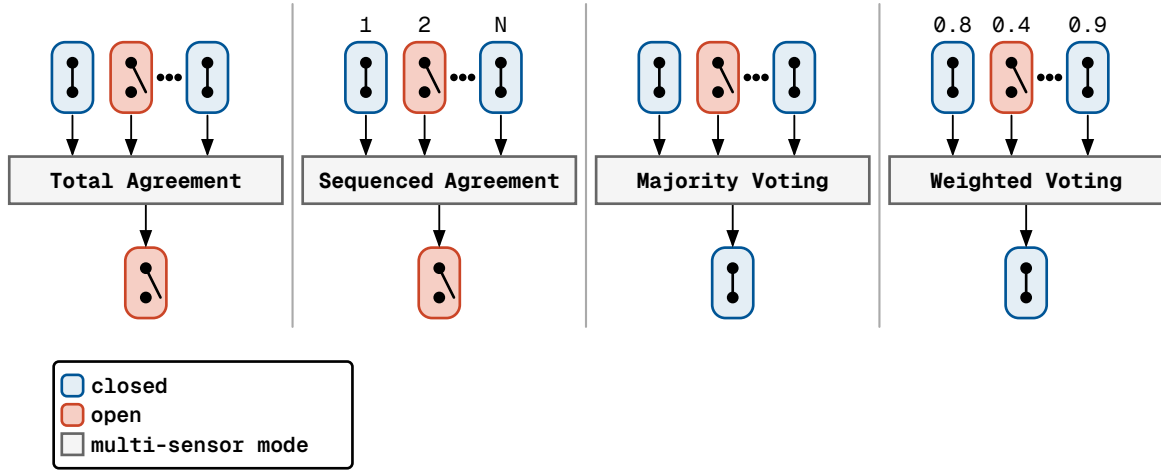


Figure 10: An abstract visualization of the four multi-sensor modes, evaluating an arbitrary number of sensor values with only the second value indicating *open*. Depending on the mode, the value itself or the order has different influence on the determined state.

3.2.1 Total Agreement

Total Agreement is a variant of majority voting with the majority being the total number of participants. In this case, all sensor values must indicate the same state. Switching to the new state is only possible if it has full support from all sensors. If the current gate state is *open* and three out of four sensors indicate *closed* no change of the internal gate state will be made. The same applies if the states are reversed. The advantage of strictness of this approach is also its biggest drawback, which is the inability to handle malfunctioning sensors. It is either all or nothing.

3.2.2 Sequenced Agreement

Sequenced Agreement is an extension of *Total Agreement* requiring all sensor values to arrive in the correct order in which they are configured. If a value is determined to be *out of sequence* based on its ticket, the flag `out_of_sequence` of its corresponding `sensor_value_state_t` instance is set. Furthermore, the value will not be considered when determining the gate state. In this case, this would mean one less value that could support the new gate state, preventing a state change.

This approach adds another layer of validity to the recorded values. It can catch possible malicious manipulation or shifting of a sensor mounting. This approach inherits the same drawback as *Total Agreement* and adds strictness of the order.

3.2.3 Majority Voting

With *Majority Vote* a state change is only possible if the number of supporting values reach a certain threshold. The majority can be defined as, e.g., 50 % or 75 %. This allows for more flexibility in handling inconsistent values. For example, with a majority of three out of four, a state change can be conducted even if one sensor value does not support the new state. With this, and depending on the configured majority, a malfunctioning sensor does not have to inhibit the system from performing a state change.

A more strict extension of this is to define the closed state as a *critical state*, which always needs the majority to remain in this state. With a three-out-of-four majority, this means that, as soon as the closed state has only two out of four values supporting it, it switches into the open state. But when in the open state, it only switches to the closed state when reaching the three out of four majority.

3.2.4 Weighted Voting

The *Weighted Voting* system refines the *Majority Voting* approach by adding more flexibility to adjust to sensor behavior. Every sensor value is assigned a weight, which can be updated based on the sensor's past influence on state changes and monitored behavior in between. A sensor with a history of being inconsistent in respect to the determined gate state will have a lower weight and therefore less influence in the decision.

This adaptive approach requires recording the sensor behaviors when the gate is at rest and the value it indicates compared to the determined gate state. Furthermore, correcting overwrites from personnel, if the determined gate state is incorrect, needs to be registered

as well. With this, the reliability of a single sensor can be adapted over time. This rating can also be used to determine when a sensor may need to be replaced.

3.3 Ticket System

All modes with the **SEQUENCE** dependency, require the sensor values to arrive in the correct order to be valid. The time between sensor events arriving can be very long. When there is a flood warning and the gate is closed, it will stay closed for at least multiple hours. The gate remains in the open mostly otherwise for long periods.

A timestamp using local system time without explicit timer overflow handling to store elapsed time is therefore not suitable. Another option would be the use of a global time base and real global timestamps instead. This however requires synchronization with a global time base.

An alternative approach is the use of a *logical clock*, a relative ticket system. Each event receive a ticket number from a global ticket counter. With this, it doesn't matter whether two events are a few milliseconds or days apart. These tickets can then be compared to each other to determine if a value has arrived *out of sequence*. In this case, the sensor values `out_of_sequence` flag is set.

The correct order depends on the current *phase* of the gate. During the *closing* phase, the ticket order must correspond to the configured order based on the physical positioning as mentioned in Section 2.1 (see Figure 1). The 'first' configured sensor must be the first to arrive. When in the *opening* phase, this order is reversed.

In order to avoid running into the same problem of a value overflow with the global ticket counter at some point, the counter and all sensor tickets are reset when all sensor values are in the same state when evaluating the gate state. In this situation, the gate is either fully open or fully closed (if no sensor is malfunctioning). The tickets of the previous completed phase (all sensor values have the same state) are not important to determining if a sensor is out of sequence in the current phase. The global counter is reset and all sensor values get assigned a new ticket starting from the reset counter value. Depending on the *phase* (opening or closing) the order in which the new tickets are assigned to the sensor values is reversed. The ticket system and the resetting mechanism is visualized in Table 1. Here, the situation is after the system's start where all sensor values start with a ticket of 0. In each row a new event has arrived and has received a new ticket.

Global Ticket	Phase	S1	S2	S3	Note
1	closing	1	0	0	First event after the start of the system.
2	closing	1	2	0	
3	closing	1	3	0	Gate moves back again, deactivating the sensor
4	closing	1	4	0	
5	closing	1	4	5	All are activated: Tickets are reset to (1,2,3)
4	opening	1	2	4	
5	opening	0	5	4	
6	opening	6	5	4	All are deactivated: Tickets are reset to (3,2,1)

Table 1: Visualization of the distribution of tickets and the resetting of tickets. Colors: Sensor is **not activated**, Sensor is **activated**.

Table 2 shows two other scenarios in which one sensor is out of sequence. In scenario 1, sensor S2 is triggered first. After the event of S1 arrives, S2 is flagged out of sequence. In scenario 2, S3 is triggered before S3, but only becomes out of sequence when S3 arrives.

Scenario 1					Scenario 2				
Global Ticket	Phase	S1	S2	S3	Global Ticket	Phase	S1	S2	S3
1	closing	0	1	0	1	closing	1	0	0
2	closing	2	1	0	2	closing	1	0	2
3	closing	2	1	3	3	closing	1	3	2
4	opening	1	2	4	4	opening	1	2	4
5	opening	1	5	4	5	opening	1	5	4
6	opening	6	5	4	5	opening	6	5	4

Table 2: Visualization of the distribution of tickets for two scenarios where values are *out of sequence*. Colors: Sensor is **not activated**, Sensor is **activated**, Sensor is **out of sequence**,

3.4 State Reporting

When a state determination is triggered by new events arriving, the state of gate is determined and transmitted to the data sink along with some other information. The data is encoded in the CBOR data format defined in the RFC 8949 Internet Standard [6]. It is based on the JSON data model and allows for extremely small code size, fairly small message size, and extensibility. The sent encoded message data consists of multiple

entries used for identifying the data origin and interpreting the sensor data, as well as the sensor data itself.

An example of a message payload with three sensor values is shown in Listing 1. The `l1` and `l2` entries in the message indicate the location with the polder number and the gate number respectively. Three entries `rb`, `ib` and `vb` encoded the number of bits that are used for the respective data types to store the sensors type, ID and value ID. These are needed to decode the encoded sensor id, type and values id. The four arrays `i`, `v`, `c`, and `k` hold the global encoded id, the current sensor values, its current event counter and its latest arrived ticket.

The current gate state is stored in the `g` entry and the sequence number of the message in `s`. The `t` entry holds the local system timestamp of the moment of state determination.

```
{
  "l1": SENSOR_CONFIG_LOCATION_POLDER,
  "l2": SENSOR_CONFIG_LOCATION_GATE,
  "tb": SENSOR_ENCODE_TYPE_BITS,
  "ib": SENSOR_ENCODE_SENSOR_ID_BITS,
  "vb": SENSOR_ENCODE_VALUE_ID_BITS,
  "i": [
    ENCODE_SENSOR_TYPE_IDS(sensor_1,type_1,value_id_1),
    ...,
    ENCODE_SENSOR_TYPE_IDS(sensor_n,type_n,value_id_n),
  ],
  "v": [
    <sensor_value_1,>
    ...,
    <sensor_value_n,>
  ],
  "c": [
    <sensor_event_counter_1,>
    ...,
    <sensor_event_counter_n,>
  ],
  "k": [
    <sensor_latest_arrive_ticket_1,>
    ...,
    <sensor_latest_arrive_ticket_n,>
  ],
  "g": <current gate state>,
  "s": <sequence number>,
  "t": <timestamp of gate state>
}
```

Listing 1: The abstract structure of the CBOR message payload.

4 Deployment

The focus of the project was the deployment of a first prototype in on site on a real flood gate in the port of Hamburg. The gate, which has been selected for deploying the prototype on, is a manually closeable swing-gate. It has already been the target in a previous work [3], in which first tests and measurements were done. The gate is depicted in its open state in Figure 13.

The first steps were to mount the sensor and magnets base mounts on the frame and gate respectively. The base mount for the magnet housing was fastened only using the pressure of the screws pressing into the gate body. As already mentioned in Section 2.2, the absence of some precise dimensions while designing, lead to the dimensions of the part being slightly inaccurate, meaning that the shape did not quite fit the gate. The part required some manual modifications to meet specification, as can be seen in Figure 11 on the right. After the magnet housing was done, the base plate for the sensor housing was fastened to the frame using assembly adhesive.



Figure 11: Side view of the mounted housings. The sensor housing mounted on the frame (left). The housing of the magnets on the gate (right).

After both base mounts had been mounted, the housings were attached to the respective mounts, so that there were almost touching when the gate was closed, as seen in Figure 12. The next step was to adjust the positions of the sensors and magnets itself as shown in Figure 1 in Section 2.1. This was done by checking the output of the software, to see the order the sensors were triggered in.



Figure 12: Top view of the mounted housings for the sensors and the magnets without the lid and with the gate closed.



Figure 13: Full view of the open floodgate with the mounted housings on the gate and the frame.

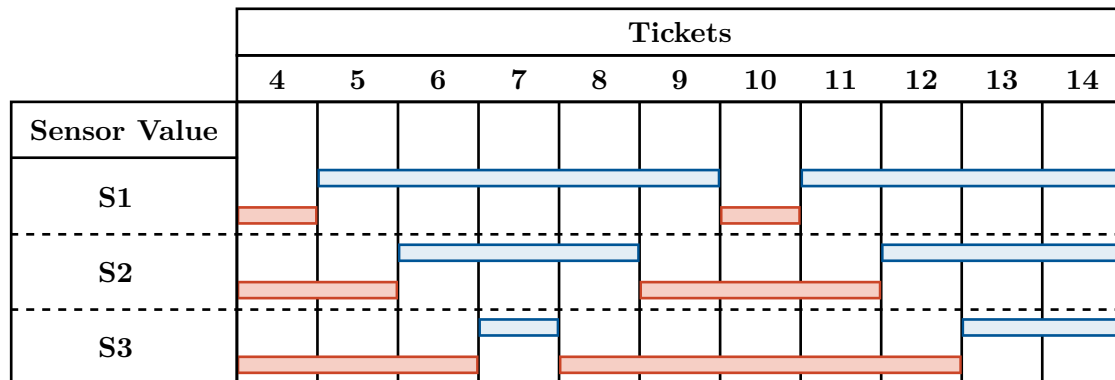
The device is currently powered via USB from a nearby system and not battery-powered, as it is planned for the long term. The software for this deployment has been configured to use a *sequential majority vote* with a two-out-of-three majority. Because the gate is not

operated regular (normally only when there is a flood warning), the software has been extended to trigger the described process by itself, for the purpose of gathering data. The sensor values messages are manually triggered in a regular interval of five minutes. In addition, someone on site will occasionally close the gate and note the time. This allows the data sent to be validated against a ground truth.

4.1 Observations

The gate is heavy and requires some momentum to reach the point where the handles can be fixed on the frame. This momentum leads to bouncing when the gate hits the frame. This leads to the sensors oscillating between both states multiple times before settling in the closed state. This could be observed via the console output of the software shown in Listing 2. The ‘bouncing’ behavior is also visualized in Figure 14.

This behavior is an important observation that highlights the difference of testing in a controlled, simulated environment and the real-world conditions. However, the temporal confirmation timer (described in Section 3) ensures that this behavior does not cause any problems for the state determination. The process of evaluating the sensor values does not begin until the gate has settled and the sensors are no longer triggered.



Legend: open closed

Figure 14: Visualization of the trigger behavior for the sensors caused by the bouncing of the gate when closing. First all sensor trigger, indicating a closed state and then open again before settling in the closed state.

```

1 # Sensor ID | Type | Value | Ticket
2 [DEBUG] Sensor 0 (Type: 4): 1, 5
3 [DEBUG] Sensor 1 (Type: 4): 1, 6
4 [DEBUG] Sensor 2 (Type: 4): 1, 7
5 [DEBUG] Sensor 2 (Type: 4): 0, 8
6 [DEBUG] Sensor 1 (Type: 4): 0, 9
7 [DEBUG] Sensor 0 (Type: 4): 0, 10
8 [DEBUG] Sensor 0 (Type: 4): 1, 11
9 [DEBUG] Sensor 1 (Type: 4): 1, 12
10 [DEBUG] Sensor 2 (Type: 4): 1, 13
11 [DEBUG] ----
12 [DEBUG] State (triggers, ticket): 1 (3, 11), 1 (3, 12), 1 (3, 13), 678 (9, 1),
13 [DEBUG] MAJ_SEQ for CLOSED: 3 > 2 ?
14 [INFO] Phase is CLOSING and is COMPLETED, State: CLOSED
15 'AC626C311828626C320E62696203627462046276620461698418401901411902421903136176840101
    02A661638400000009616B84010203046167F56173182461741A002862940000000000000000000000
    000000000000000000000000000000000000000000000000000000'

```

Listing 2: Console output of the software during the closing of the gate. Lines 1-9 show the triggered sensors, their values and tickets. Line 12 shows the state of all sensor values with their event count and ticket. Line 13 shows the result of the gate position determination for Majority Vote with Sequence. Line 14 shows the validated and new state. Line 15 is the cbor-encoded message.

The data about the gate state and the sensors values is successfully send to the data sink when operating the gate and also every 5 minutes.

During development and deployment some issues arose.

A problem that arose during development is the unsuitability of the used *Adafruit Feather nRF52840 Sense* when using multiple sensors. The chip supports only up to 8 interrupt lines (or GPIO tasks and events channel³), three of which are needed for the *Adafruit LoRa Radio FeatherWing* extension board. The remaining 6 interrupt lines are not enough to use both normally-open and normally-closed pins of all three (or more) reed sensors at the same time. This was the reason only the NO-pins were used for this deployment.

Another problem occurred with the software, where LoRaWAN packets could no longer be sent after a while. After a thorough investigation, this turned out to be an issue within RIOT involving the interaction between the LoRaWAN network interface (*netif*) and the *PKTQ* module. Using the *PKTQ* module, which is queuing and resending packets when a network interface is busy, leads to packets not being fully released by the LoRaWAN interface. Furthermore, when the *netif* was actually busy, the packets were stored in the queue without the data. As a result, the resend of the queued packets failed. This issue has now been resolved⁴.

Due to the above-mentioned issue with the messages and the message storage of the LoRaWAN network server not working, no long-term data was gathered that could be evaluated.

5 Conclusion and Outlook

The main objective of this project was the deployment of a prototype device to monitor the state of a flood gate. Based on the insights from previous works, three reed sensors have been used for this task. Hardware parts have been manufactured, to allow the mounting of the sensors to the gate. The developed software determines the gate' state based on the sensor values via a *sequenced majority vote*. Despite the need for some last-minute adjustments, the prototype was successfully installed on the gate and put into operation.

Further work is required to extend the software to be self sufficient and reliable and, suitable for use with various types of flood gates. This includes the reworking of some parts and the addition of necessary features. As for the mounting of the sensors and magnets, an alternative vertical arrangement could be considered, to lessen the obstruction when the gate is open.

³https://docs.nordicsemi.com/bundle/ps_nrf52840/page/gpiote.html

⁴<https://github.com/RIOT-OS/RIOT/pull/22771>

The additional features for the software include energy management to allow the system to run on a battery charged by a solar panel. Additionally, some system to track sensor health or reliability is needed that monitors sensors behavior. There is also a need for a feature that allows authorized personnel to be notified, when the reliability drops to a low level, enabling preventive maintenance to be carried out.

References

- [1] RESCUE-MATE, “RESCUE-MATE.” Accessed: Mar. 06, 2025. [Online]. Available: <https://www.rescue-mate.de/>
- [2] T. Rupelt, “Investigating Reed Sensors for Monitoring Closing Mechanisms of Flood Protection Gates,” technical report, 2024. [Online]. Available: https://inet.haw-hamburg.de/teaching/ss-2024/project-class/fw1_timon_rupelt.pdf/view
- [3] T. Rupelt, “Investigation of using Reed Sensors for Monitoring Flood Protection Gates,” technical report, 2024. [Online]. Available: https://inet.haw-hamburg.de/teaching/ss-2025/project-class/prj1_timon_rupelt.pdf
- [4] Standex-Meder Electronics, “Reed Technology.” Accessed: Mar. 06, 2025. [Online]. Available: <https://standexelectronics.com/resources/technical-library/technical-papers/technical-databook-reed-switch-technology/>
- [5] E. Baccelli, C. Gündogan, O. Hahm, P. Kietzmann, M. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählich, “RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT,” *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4428–4440, Dec. 2018, [Online]. Available: <http://doi.org/10.1109/JIOT.2018.2815038>
- [6] C. Bormann and P. Hoffman, “Concise Binary Object Representation (CBOR),” IETF, RFC 8949, RFC 8949, Dec. 2020. [Online]. Available: <https://doi.org/10.17487/RFC8949>