Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Model based development of a smart braking system

BSc Project Laboratory
(BMEVIMIAL00)

Author
János Gorondi

Advisor
András Földvári
Simon József Nagy

December 16, 2022

Contents

Chapter 1

Task description

Our teams goal was to design the smart braking system of a car, based on the provided requirements of the clients. Communication of system components is achieved through DDS. This system is able to detect the objects in front of the car and measure the current speed at which the vehicle is moving. Based on this information we are able to calculate the rate of braking required to safely slow down and avoid collisions. With this module intact in the car, possible accidents could be avoided and passenger lives could be saved.

During this whole semester, I was working in a team. We all had the same tasks in the beginning but in the later half of the project, we switched to our individual assignments, which we could choose for ourselves. Our first weeks consisted of determining high level requirements, figuring out possible executions for safety scenarios, building up high level functional components and dividing up the project between ourselves. More on that later on.

The modeled system, that our team thought of, is a very abstract realization of the requirements. Based on our knowledge and available resources, we choose a challenging but not too complex implementation which we could realize. In order to be realistic, and meet the customer's requirements at the same time, we had to decide how to divide up the project and figure out which are the most crucial components in our system.

The spreadsheet below shows each component with their dedicated purpose:

Car brakes	LED which indicates the level of brake intervention
Car speed sensor	Software simulated sensor, presenting realistic data
Car distance sensor	Sensor which continuously measures the distance from object in front
Car monitor	Monitoring software to alter the cars behavior, display data (speed, distance, brake), collect and store logs in a database

In a real world example the braking system would have a camera installed on the car for identifying objects in front. With that additional information our braking calculations could achieve better accuracy. Regard to our situation we left this component out. We also reduced the complexity by making the model unable to move. We were thinking of

using servo motors[1] and setting their rotation according to the braking required, but in the end stayed with the LEDs[2].

[1]Servo motor a rotary actuator that allows for precise control of position, velocity, acceleration.
[2]A light-emitting diode is a semiconductor device that emits light when current flows through it.

Chapter 2

High level requirements

The clients have made the following requests, that we had to fulfil:

- If something gets in front of the car people, animals, etc... then the vehicle should stop and avoid a collision.
- If the collision cannot be avoided, slow down as much as possible.
- Do not cause unnecessary braking.
- Distinguish between cars and pedestrians.
- Be accurate, good and cheap.

According to the customer requirements, we defined the high level system requirements. These requirements consist of other sub components which help to clarify them even further. These conditions are absolutely have to be met and completed.

1. The system must avoid any collision between the car and other objects
   - 1.1 The system must know if an object is in front of it in any traffic situation
     - 1.1.1 The system must be able to measure the distance between itself and the object in front
   - 1.2 The system must be aware of its current speed
     - 1.2.1 The system must be able to alter the speed of the car
   - 1.3 The system must be aware of its surrounding at all times
     - 1.3.1 The system must be able to observe it's surrounding and environment
2. The system must not intervene if its not necessary
   - 2.1 The system must be able to calculate necessary intervention
3. If the collision is not avoidable the system must minimize the damage caused by the car
   - 3.1 The system must determine when the collision is unavoidable
4. The system must be able to differentiate between objects (pedestrians and vehicles)

    4.1 The system must react differently based on the detected object

5. The system must protect all human life.

    5.1 The system must not cause any damage to the cars passengers with its interventions.

6. The system must function under any weather conditions

7. The system must prevent failures from happening as best as it can

8. In case of a failure the system must tolerate it

    8.1 The system must signal when a failure happened

    8.2 The system must record the details of failure

9. In case of an emergency the system must shut down

Since our teams job was not to change the whole automotive industry with this perfect invention, but to model a smart braking system, we tried to meet as many requirements as possible from the above mentioned ones.
The system isn't aware of its current environment due to lack of sensors and since it is not moving it was a necessary simplification that we took. Our model can not differentiate between various types of objects, because we used a simple distance sensor without a camera. Using camera and image processing could be a next step in improving this model. The system is not that fault tolerant, simply due to single point of failures in it, but this could be easily solved introducing some hardware redundancies to the model (having 2 copies of each SBC, microcontroller and LED). The calculation of the precise brake intervention amount could be fine tuned using the PID intervention algorithm.

Chapter 3

Safety scenarios evaluation

For a model to be complete we have to also think about the environment at which our system will be operating. The surroundings bring uncertainty and unpredictability to our calculations, but this problem can not be entirely avoided. Our best practise is to be prepared for as many different outcomes as we can. For this reason our team defined a few real world scenarios and rated them according to their ASIL level (3.1). By doing so we can identify the possible risk and danger connected to these scenarios and devise a plan to take care of them.

Severity	Exposure	Controllability		
		C1 (Simple)	C2 (Normal)	C3 (Difficult, Uncontrollable)
S1 LIGHT AND MODERATE INJURIES	E1 (Very low)	QM	QM	QM
	E2 (Low)	QM	QM	QM
	E3 (Medium)	QM	QM	A
	E4 (High)	QM	A	B
S2 SEVERE AND LIFE THREATENING INJURIES – SURVIVAL PROBABLE	E1 (Very low)	QM	QM	QM
	E2 (Low)	QM	QM	A
	E3 (Medium)	QM	A	B
	E4 (High)	A	B	C
S3 LIFE THREATENING INJURIES, FATAL INJURIES	E1 (Very low)	QM	QM	A
	E2 (Low)	QM	A	B
	E3 (Medium)	A	B	C
	E4 (High)	B	C	D

QM (Quality Management)
Development supported by established Quality Management is sufficient.

A — **lowest ASIL** Low risk reduction necessary
B
C
D — **highest ASIL** High risk reduction necessary

Figure 3.1: ASIL risk spreadsheet

ASIL: Automotive Safety Integrity Level is a risk clarification scheme for Functional Safety for Road Vehicles. It is used in the automotive industry. The ASIL is established by performing a risk analysis of a potential hazard by looking at the Severity(S), Exposure(E) and Controllability(C) of the vehicle operating scenario. The safety goal for that hazard in turn carries the ASIL requirements.[1]

Traffic jam	
Situation	We are waiting in a big traffic jam, the cars start to move. When starting to move forward the car suddenly brakes. The car behind collides with our car.
Failure	The system detected a false emergency.
Safety goal	The car should not be stopped if it's not necessary.
Qualifications	S1, E1, C3
ASIL	QM

Highway	
Situation	The car is moving at a very high speed on the highway. The car suddenly starts to brake. The car behind collides with our car. The passengers are injured.
Failure	The system detected a false emergency.
Safety goal	The car should not be stopped if it's not necessary. The car should brake only as much as it's necessary.
Qualifications	S3, E1, C3
ASIL	A

Parking	
Situation	While performing the system unnecessarily brakes while causing discomfort to the passengers.
Failure	The system detected a false emergency.
Safety goal	The car should not be stopped if it's not necessary.
Qualifications	S0, E1, C3
ASIL	-

Turning	
Situation	The car is turning into a crosswalk. Pedestrians are walking across the crosswalk.The system doesn't stop the car.The car hits a pedestrian or other vehicle.
Failure	The system detected a false emergency.
Safety goal	Objects should also be detected while turning.
Qualifications	S2, E1, C2
ASIL	QM

Pedestrian	
Situation	The car is going at a relatively high speed on a main road. Suddenly a pedestrian steps in front of the car. The pedestrian is hit with full speed.
Failure	The system didn't react to the pedestrian in time.
Safety goal	Objects in front should be detected as soon as possible.
Qualifications	S3, E2, C3
ASIL	B

Deer	
Situation	The car is going with high speed on the highway, suddenly a seer jumps in front. The system is unable to stop the car in time. The deer is hit with full speed, all the passengers are dead.
Failure	The system didn't react to the sudden appearance of the deer in time
Safety goal	Objects in front should be detected as soon as possible.
Qualifications	S3, E4, C3
ASIL	D

We derived the necessary safety goals for our system from the situations, that we tried to fulfil:

Safety goal	Requirement
Objects in front should be detected as soon as possible.	1.1
Objects should also be detected while turning.	1.3
The car should not be stopped if it's not necessary.	2.
The car should brake only as much as it's necessary.	5.1.

Chapter 4

High level functions

We derived these high level functions from the high level requirements. These functions are broken down further into smaller and more manageable components. This process was needed in order to develop our model in parallel and be efficient with the implementation. By braking down the functions we identified three distinct subsystems which we could divide among ourselves. If we each ensure that the functions from the bottom to the top are verified by requirements, than we have a safe system.

1. **Avoid collision**
   - 1.1 Detect objects
     - 1.1.1 Measure distance in front
     - 1.1.2 Identify objects in front
2. **Intervene**
   - 2.1 Change speed
     - 2.1.1 Use breaks
     - 2.1.2 Calculate level of interventions
3. **Notify**
   - 3.1 Users
   - 3.2 System
   - 3.3 Authorities
4. **Collect Data**
   - 4.1 Store data in persistent storage
   - 4.2 Visualisation of data

- **Measuring module**, implemented by János Gorondi
- **Intervention module**, implemented by László Ábrók
- **Monitoring module**, implemented by Regina Bodó

Chapter 5

Common task - Overview

After completing the previously mentioned modelling steps we moved onto the implementation details. The system had to be faulty tolerant, for that reason we distributed it to avoid single point of failure and be able to work on development of the project in parallel.

To simulate a clever braking system, the three most important pieces of information are the **distance** of the system from the object in front of it, the **speed** at which it's moving and the amount of **braking** it can do to avoid a potential accident(5.1). **The measuring module** consisted of several sub-components, but I will go into the details of these later. Its function is to measure, filter and transmit speed and distance data via DDS using Topics.
The intervention module uses the speed and distance data from the measurement module to calculate the amount of braking needed to slow down in time to avoid an accident and injury to passengers. The braking rate is transmitted to the other components via DDS (5.2).
The monitoring module collects the speed, distance and the brake data from other components via DDS than stores them in a persistent database. Data can later be processed and by that potential faulty errors and vulnerabilities can be detected. The received information is logged and displayed on a desktop screen via a graph. Users of the desktop app can alter the current speed of the vehicle or request an emergency shutdown of the system which completely stops it from operating until a reset request is published.

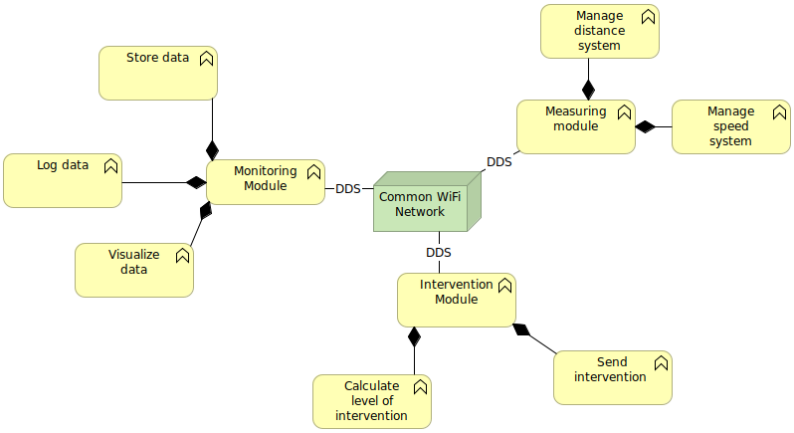

Figure 5.1: Model of the entire smart braking system

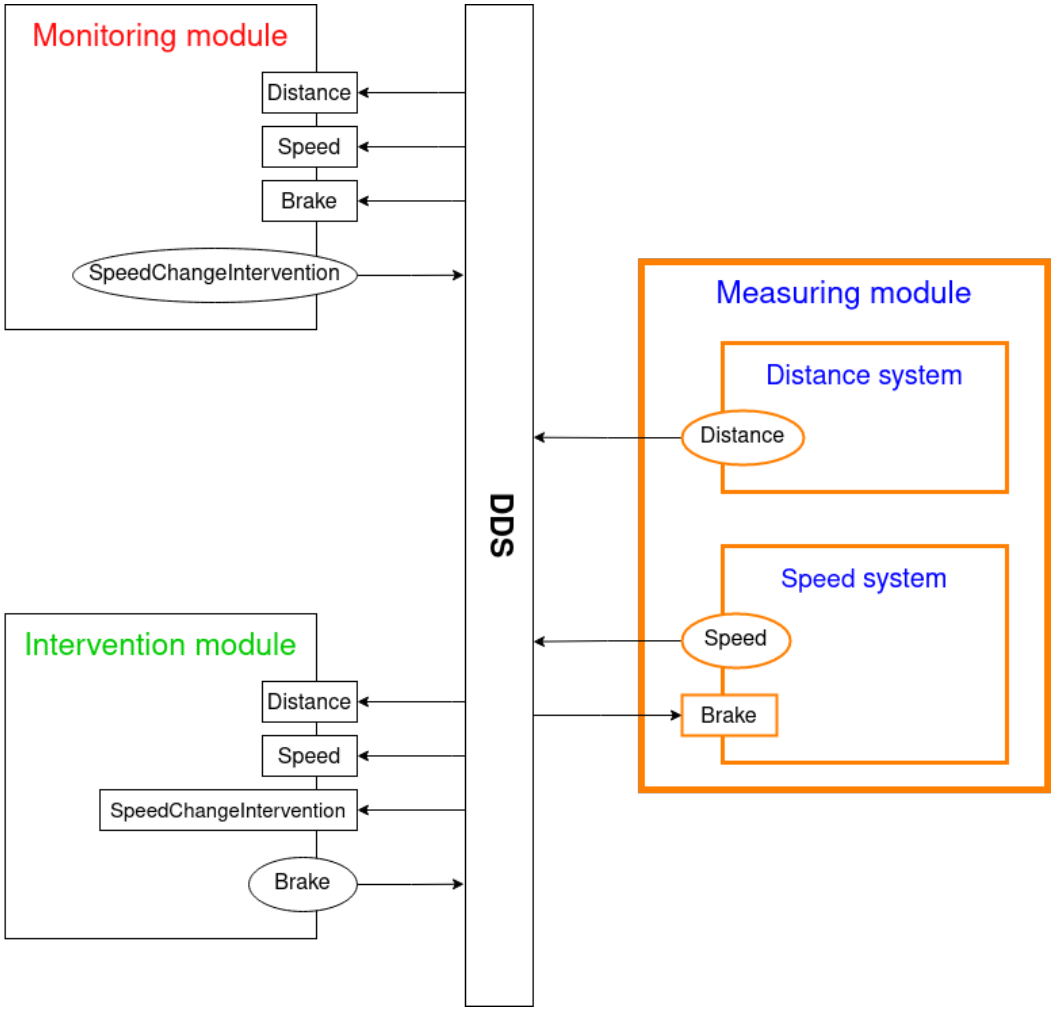

Figure 5.2: Model of DDS communication with Topic between modules. Squares represent Subscribers, circles represent Publishers

Data Distribution Services (DDS) DDS[1] is a communication middleware for Cyber-Physical Systems (CPS)[2]. Middlewares are tools and code libraries for common features. These features are communication, data distribution, authentication and QoS[3] services. We used RTI's Connext DDS in our project. It uses a Publish-Subscribe style communication middleware.
DDS has various components which all play a role at achieving reliable and fast communication within a network. It creates a global data space called **Domain**, in which the communication takes place. **Publishers** and **Subscribers** are the components who send and receive data there. Publishers publish **Topics** into the domain and Subscribers receive those Topics. Topics are global objects where the exchanged information is stored. They have a distinct type, identifier and a QoS service. The Topics are available to everyone who is in the domain, but only Subscribers who subscribe to the given Topic will receive and use it. The Topics are read by **Data Readers** and wrote by **Data Writers**, they can handle the object of a single Topic. (5.3)
These files can largely be generated from **platform independent .idl** extension files, by the RTI Launcher, which greatly helps development. All components of the DDS communication have to be on the same network (Wi-Fi[4]) to be able to identify and discover each other. Data transmission can be synchronous or asynchronous and it can be greatly altered by QoS policies.[2]

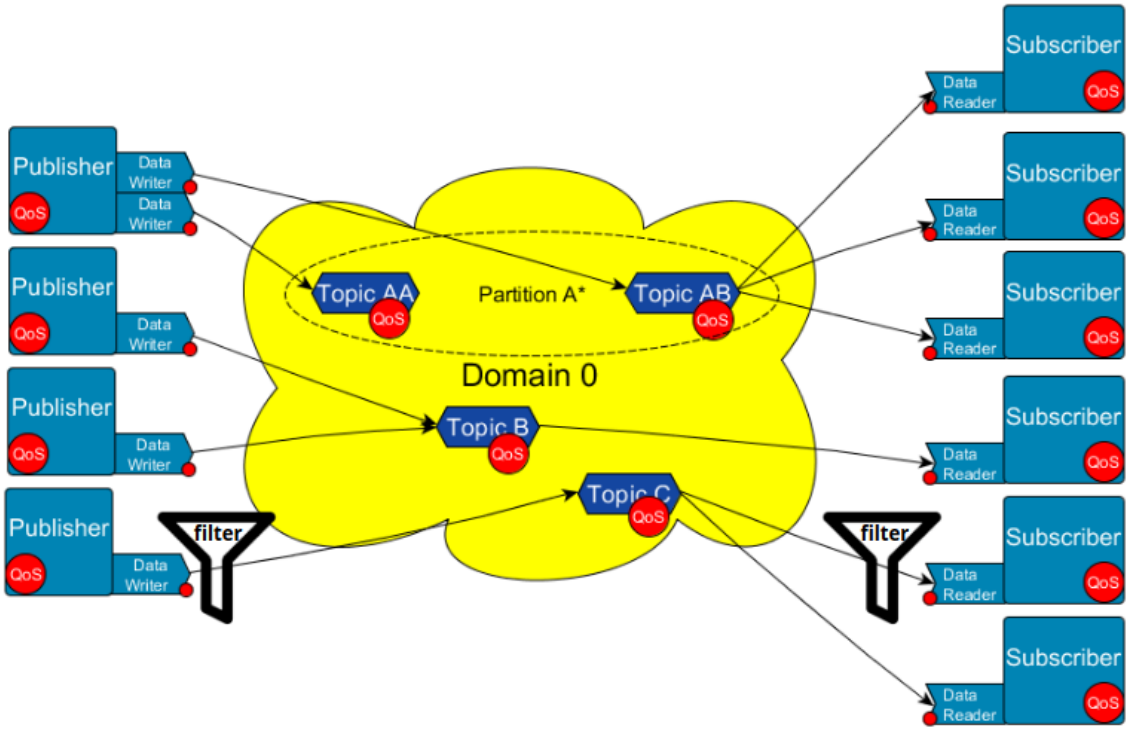

Figure 5.3: DDS Domain and Topic [2]

[1]This paragraph takes complete sentences form Huszerl Gábor's ppt.
[2]Computer systems in which a mechanism is controlled or monitored by computer-based algorithms.
[3]Quality of Service is a network technology that guarantees the rate of success of our application working.
[4]Wi-Fi is a wireless technology used to connect computers and other devices to the internet.

Chapter 6

Personal task - Measuring module

6.1 Task description

My individual task was the modelling, design and implementation of the measuring module. I divided my task into two smaller separable parts. One part was responsible for the handling and transmission of distance data and the other part similarly was responsible for the speed data. I used DDS for reliable communication between the distributed system components (6.1).

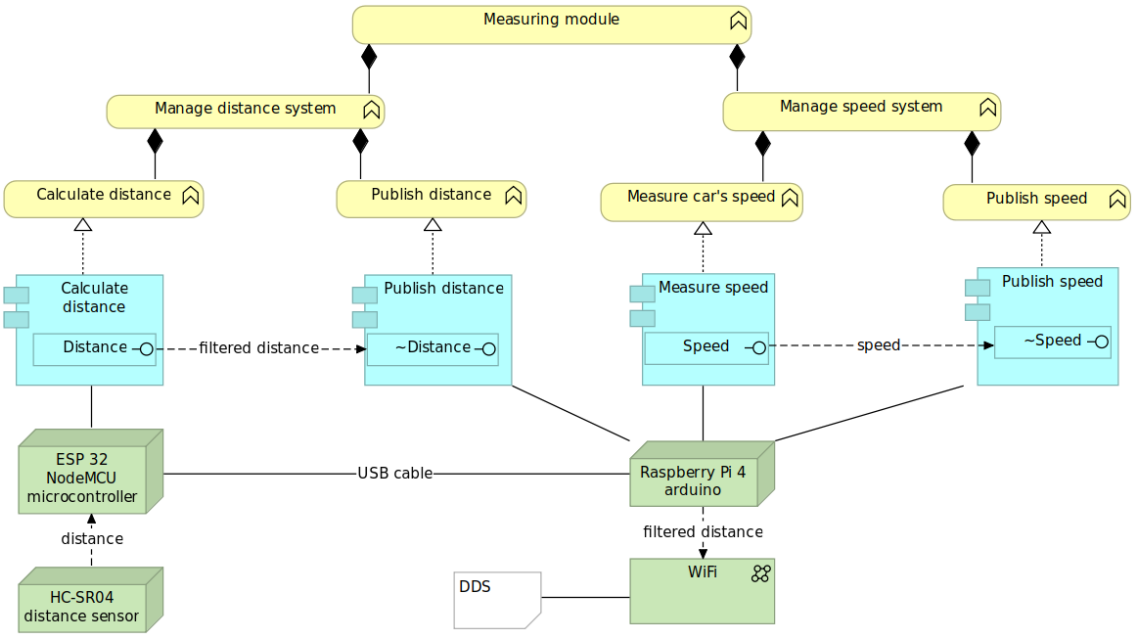

Figure 6.1: Measuring system with all subsystems

6.2 Used architecture

6.2.1 Hardware

ESP32 NODE_MCU_V1.1	ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The distance sensor is connected to its designated pins.[3]
HC_SR04	Ultrasonic sensor that can measure distance. It emits an ultrasound which travels through the air. By that distance can be calculated.[4]
Rasberry Pi 4 Model 4	Raspberry Pi is a series of small single-board computers (SBCs). It is cheap and compatible. It is a small, fully functional computer. It has all qualities of a PC- a dedicated processor, memory, and a graphics driver. It even has its own operating system called Raspberry Pi OS which is an optimized version of Linux. For me it was running java code which used DDS and it was connected to the ESP32 with an USB cable.[5]
USB Cable	Universal Serial Bus, used mostly to connect computers to peripheral devices. They are fast and they carry power as well as signals.[6]
Micro SD card	Small memory storage devices capable of writing and reading data. In my case we installed different operating systems (Ras Pi OS 32, 64-bit, Ubuntu 64-bit) into the Rasberry Pi 4 with it.

6.2.2 Software

Linux	Switched from Windows to Linux for better DDS performante and easier port communication.
VSCode + PlatformIO (C++)	Enables to build, upload, and monitor the code that is developed for ESP32.
Connext DDS (Java)	RTI's DDS communication product, works with university license.
IntelIJ IDE (Java)	Best IDE[1] for Java development, I used it when developed on desktop locally.
RTI launcher	Can not file license file. Used it's generator to generate Java classes from .idl files. You can also set the target platform too.
Archimate	Easy to use modelling tool. Made the diagrams for the project in it.
Rasberry Pi Imager	With it I could write different OS[2] to my microSD card, from where Raspberry installation occured.
VNC viewer	Intuitive tool to ssh[3] into remote device and be able to develop without the need of monitor or mouse.
VSCode + Remote Developer (Java)	Able to ssh into Raspberry, than use it as it was your desktop, with access to file system command line ect...

[1]Integrated development environment is an application that helps programmers develop software.
[2]System software that manages computer hardware, software resources.
[3]Secure Shell is a network communication protocol that enables two computers to communicate.

6.3 Distance measurement and transmission

6.3.1 Task

Distance was measured with an HC-SR04 ultrasonic distance sensor, controlled by dedicated pins. By flashing them at a given rate, the sensor emitted an ultrasonic signal. The ultrasound is reflected from the object in front of the sensor. Given the time between the signal being sent and received, and the speed at which the sound travels, we can calculate our distance from the object in front of us (A.1).
The resulting distance data is then subjected to a filtering algorithm to eliminate sensor errors and outliers (A.2). The resulting final data is transmitted via Serial Port. On the serial port I have set 9600 bits per second, 1 stop bit and no parity bit. The name of USB Serial Port in Linux is "ttyUSB0".

6.3.2 Challenges

During development, the biggest difficulty was not the software, but the hardware caused problems. (A.3).

6.3.3 Testing

Testing a given component independently of other components of the subsystem to see if they can work on their own.
With the Platform IO monitor option, we can test the code loaded on the ESP32 as it automatically displays the serial port data on the console. I tested the sensor to measure the distance of a stationary and moving objects as well. After testing I considered it safe against errors.

6.4 Distribution of distance information via DDS

6.4.1 Task

First I did the development locally on my laptop, later I migrated the code to the Raspberry, I'll write about that later. The task here is to read the value sent by ESP32 from the serial port and then sending distance data via DDS using Topics (6.2).

With the help of the RTI's Launcher and our team's commonly designed *interfaces.idl* file I generated the main skeleton of our project which was responsible for the DDS communication. After modifying them and writing my own classes the implementation was not that hard.
After successful read of the distance data, it is published with the help of the Speed-Publisher to the common DDS Domain in a distance Topic where Subscribers who are interested in it can receive the information (6.3).

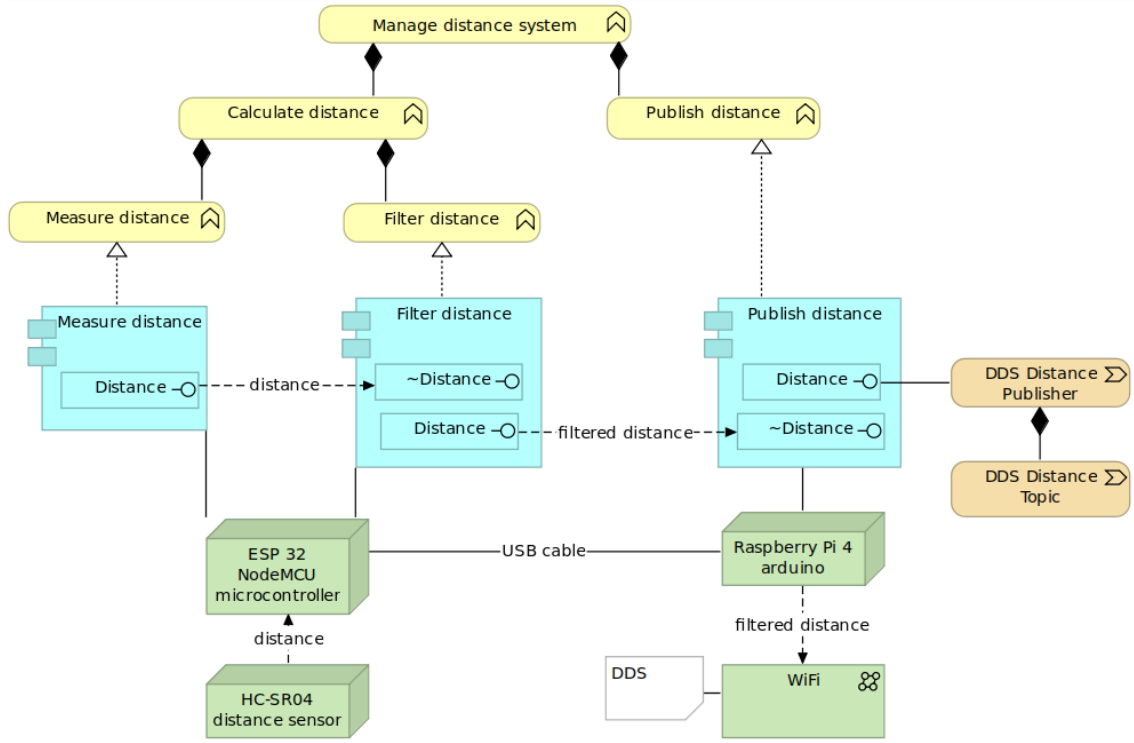

Figure 6.2: Manage distance subsystem

6.4.2 Challenges

This task although on surface looks easy it is not like that at all. If developers do not pay full attention while working with these libraries, it can be quite challenging to find the problem (A.4).

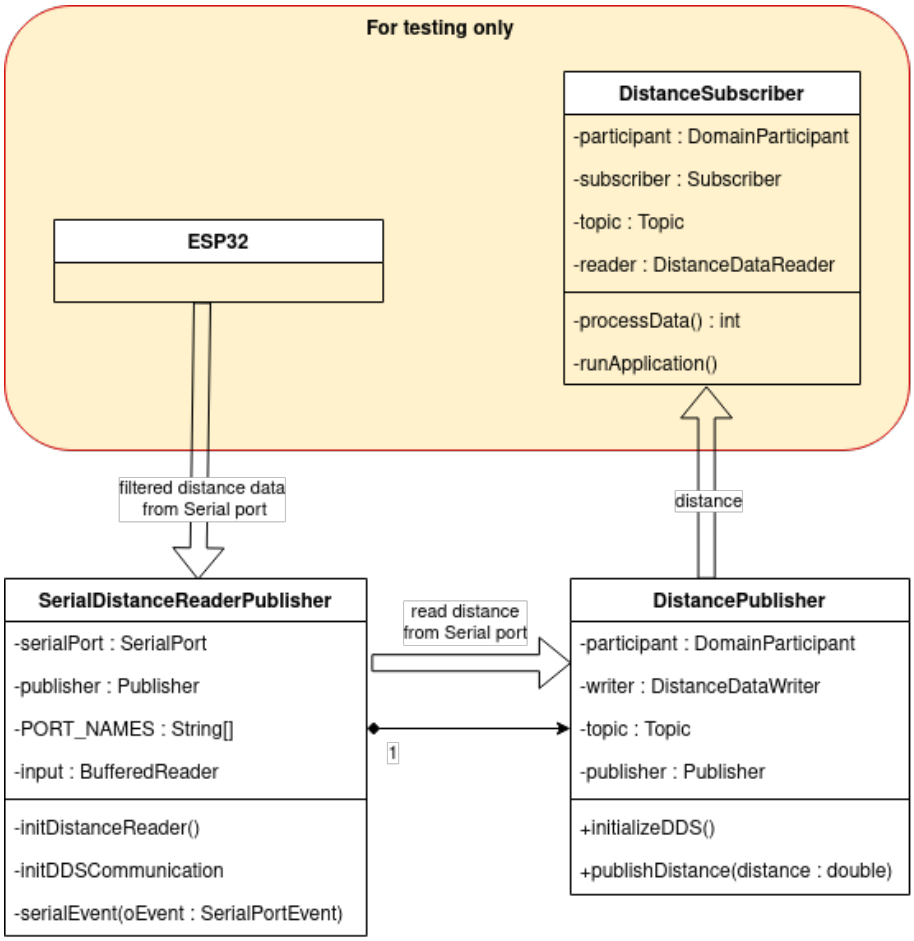

Figure 6.3: Data flow of distance DDS and Serial port communication

6.4.3 Testing

I also implemented the opposite of the DDS components I usually use, each Subscriber had a Publisher and vice versa. I ran them in parallel on my laptop and tested the correctness and accuracy of the communication between them by writing a printout.

6.5 Speed data emulation

6.5.1 Task

Our model is not moving, but to model a smart braking system, it is essential that our car has speed. Speed and distance are used to calculate the appropriate braking intervention needed to ensure a safe journey.
Using a software component, we can simulate sensor data that emulates the real speed of a car. Starting from a base speed, our car accelerates or decelerates randomly, with realistic timing. It sends a brake signal that is fully controllable, automatically adjusting the car's speed to a given amount in Brake Topic (6.4). If you get a "SHUTDOWN" message in the topic, it will immediately stop the car, set its speed to 0, and keep it there until it receives a "RESET" signal, which will cause it to start accelerating again.
I read the Brake Topic with a BrakeSubscriber and send the modified value with a Speed-Publisher(6.5).

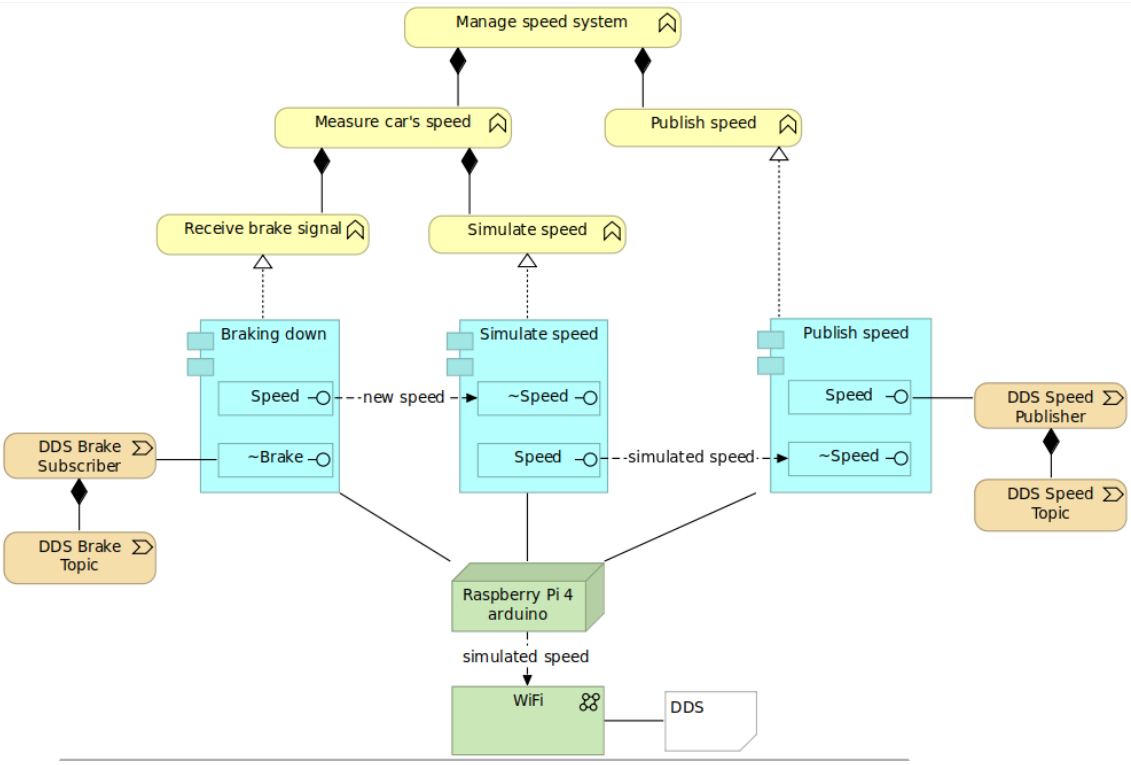

Figure 6.4: Manage speed subsystem

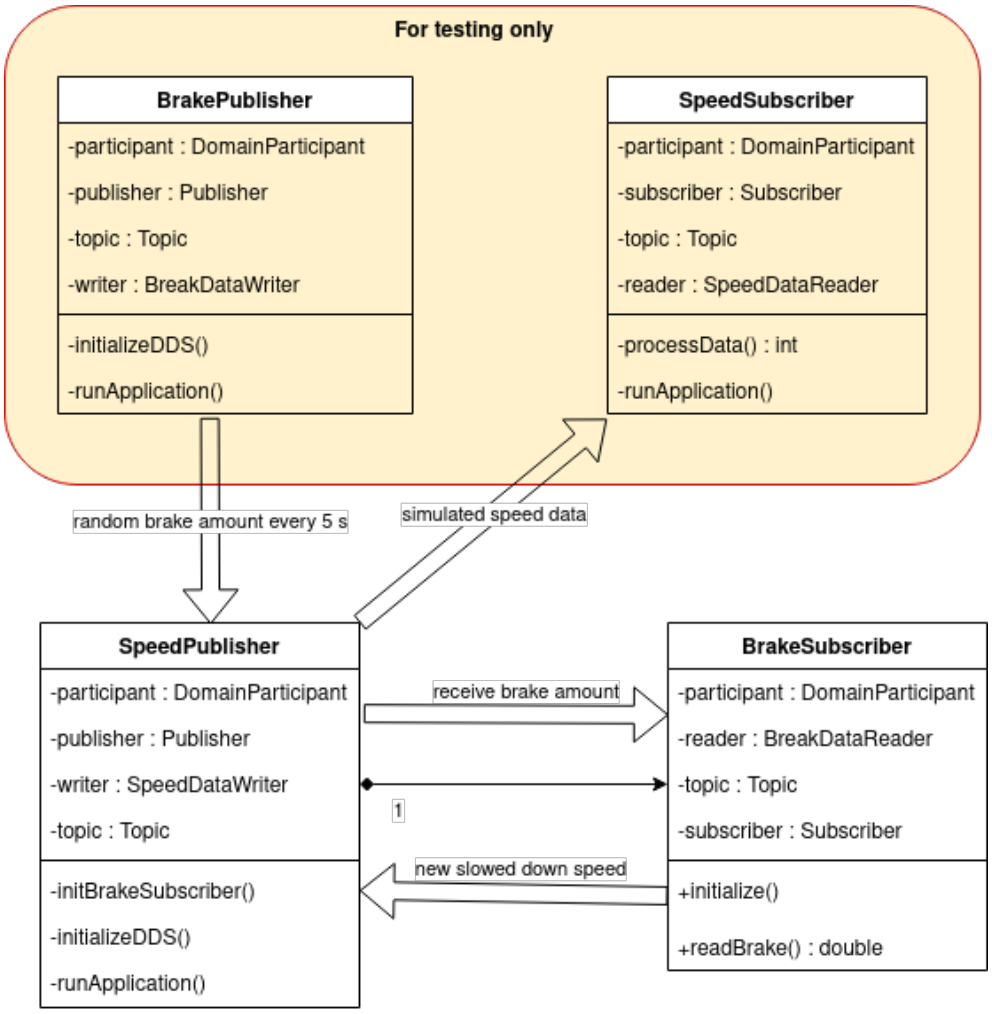

Figure 6.5: Data flow of speed DDS communication

6.5.2 Challenges

Basically, after completing the previous tasks, the creation of this subsystem was easier. Here, too, careful thought had to be given to the correct handling of simulations and the interventions that affect them.

6.5.3 Testing

I created a BrakePublisher that sends a random braking value in a Brake Topic every 10 seconds to the BrakeSubscriber, which sets the speed of the car accordingly. Speed is then emulated by the SpeedPublisher and sent via a Speed Topic. This Speed Topic is then captured by a SpeedSubscriber as testing measures and sent to the console, so that changes can be seen in near real time.
I also tested the periodic sending of the "SHUTDOWN" and "RESET" signals in the DDS communication system explained previously, which also worked as expected.

6.6 Code migration to Raspberry 4

6.6.1 Task

I had to move my existing DDS components running on the local desktop Linux environment to the Raspberry Pi 4 (RPi). I didn't have to write any new source code, but I did have to fully setup the RPi so that it was ready to run the DDS middlewares I wrote. After a lot of trial and error and reading forums, I managed to do this and my task was complete.The Measuring module designed by me can communicate reliably with the rest of the system.

6.6.2 Challenges

Despite the short and simple task description being able to make the RPi able to communicate through DDS was a surprisingly challenging task(A.5).

6.6.3 Testing

I tested communication on DDS by running both speed and distance subsystems on the RPi at the same time.With matching subscriber running on my desktop I printed the speed and distance values on the console.

6.7 Full system integration

6.7.1 Task

At last the final part of our project is left, where each team members work is put into a massive bigger application, modelling a complete system. Locally everyone's implementation was running flawlessly, so logically it should have been enough to connect to the same network, start the Subscribers and Publisher and should be working.
That was not the case, but our bugs and transient data inconsistencies were not too extreme. Regina worked on accepting the Speed Topic as well, which she forgot. László put an outlier filter into action within his BrakePublisher to be more robust. My distance values that I provided through DDS were pretty inconsistent, so I also implemented a small buffering and data filtering functionality into my SerialDistanceReaderPublisher class to improve the quality of my data (6.6).

After these minor corrections we were successfully able to integrate our whole system consisted of the Measuring, Intervention and Monitoring modules. We were overjoyed since it was a visual, yet tangible representation of our hard work and success. It was all worth it.

Figure 6.6: Chart about the communication of the integrated system

- **Orange:** Current speed of car
- **Purple:** Cars distance from object in front
- **Blue:** Braking amount to prevent accident

Chapter 7

Summary

Every component of the Measuring module I designed and implemented was made fault-tolerant as far as my knowledge and resources allowed. This enables it to safely and correctly transmit filtered distance values and emulated velocity information to other components of the reducer via DDS (7.1).

I tried to find the best possible solution to the problem, I don't think I succeeded, but I did my best to complete the task. Transmitting sensor data close to hardware using microcontrollers was a completely new and exciting topic for me. By the end of my assignment, I feel that I had achieved a general level of experience with them.

Learning about DDS has opened up a new field and career opportunity for me in the world of CPSs, which I am very happy about. I would like to develop in this area in the future.

Overall, I am happy to have been given a challenging topic in a theme lab because I have learned and developed a lot while working on it. I consider it a huge success that I managed to complete it in full.

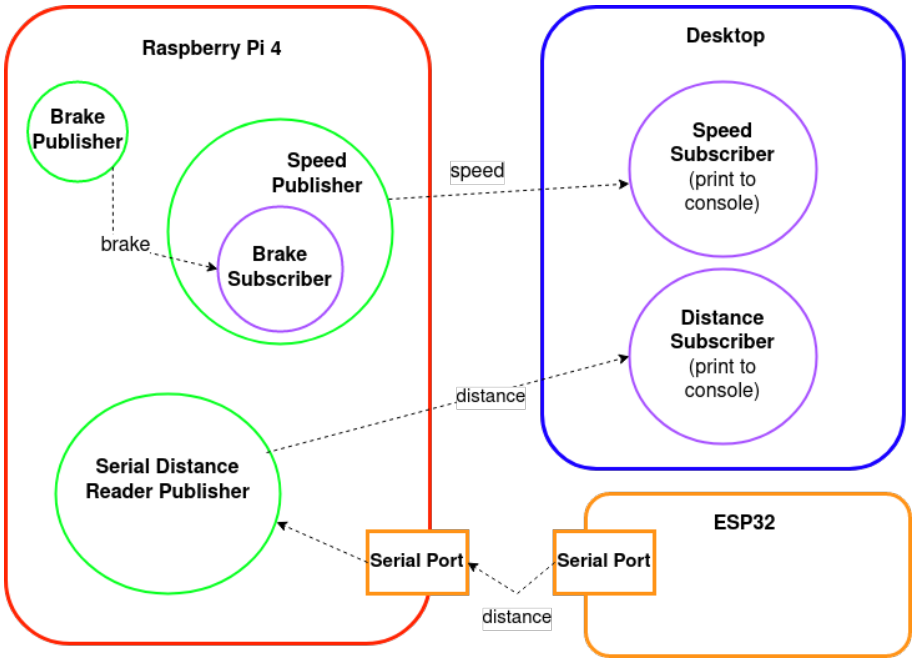

Figure 7.1: Information flow through hardware and software components

Appendix

A.1 Calulation of distance

The timescales required to operate the ultrasonic distance sensor were given in microseconds, and because of the small size of our model, the main unit of measurement I used was cm/microsecond.
In microseconds, we get the time between the arrival and departure of the ultrasonic waves (duration). The speed of sound is 340 $m/second$ which is

$$\frac{340 * 100}{10^{-6}} = 0.034 \ \frac{cm}{microsecond}$$

and this gives the distance travelled in 1 microsecond. This multiplication must be divided by 2, since the sound travels back and forth.

$$\frac{duration * 0,034}{2} = \ distance$$

A.2 Filtering algorithm

While using the distance sensor, I noticed that it sends a lot of random and very erratic values. I wanted to minimize the effect of this. My algorithm measures 20 distances in 10000 microseconds. It compares the value of each distance to the distance measured immediately before it. Depending on the percentage of deviation from this distance, it increases, decreases or leaves it as it is. The resulting 20 values are sorted in ascending order using the quick sort algorithm [7]. I calculate an average for these elements, ignoring the 5 largest and smallest values. These extreme outliers store noise due to sensor malfunction, so they are not counted. The resulting average will be the final distance transmitted on the Serial port every $20 * 10000 = 0.2$ $seconds$ [8].

A.3 Distance subsystem challenges

Several pins of ESP32 were unusable. Finding the only working connection to the distance sensor was difficult. For serial port software communication I used 2 different libraries JSerialComm [9] and RXTXcomm [10]. The latter is easier to use and better, but unfortunately it doesn't compile on the AARCH64 bit processor that Raspberry 4 has. Therefore JSerialComm was left as the final implementation. Also have to keep in mind that USB Serial port is only accessable by one component at a time, so keep an eye for mutual exclusions.

A.4 Distance DDS software challenges

Installing and using DDS on Linux has a lot of pitfalls, some of which I have encountered:

- **Environment variables:**
  - JAVA_HOME for /lib/jvm/<java-version> library
  - NDDSHOME for /opt/<dds-version> dds installation directory
  - LD_LIBRARY_PATH for $NDDSHOME/lib/<OS-and-processor-specific-architecture-version>
  - RTI_LICENSE_FILE for $NDDSHOME/rti_license.dat exact location of license [11]
- Can only guarantee a safe update of environment variables when the machine is rebooted
- For Linux, install DDS in /opt/... as root user, saves a lot of headaches
- Linux host and architecture specific target bundles must also be run with the package generator (rtipkginstaller), only the generated target architecture can be used later in the RTI Launcher [12]
- Always generate your code in the RTI Launcher code generator for the architecture platform of the target device
- Generated code should be developed in IntelIJ if you want to test it locally
- In IntelIJ IDE, include your Eclipse project as import project from existing source. (it generates it by default) (A.4.1) At project structure/modules/sources, set the project as source (blue folder icon) and add $NDDSHOME/lib/java/nddsjava.jar and $NDDSHOME/lib/<OS-and-processor-specific-architecture-version> library as dependency [13].

If you accidentally want to run 32-bit code on a 64-bit processor, you won't be able to.

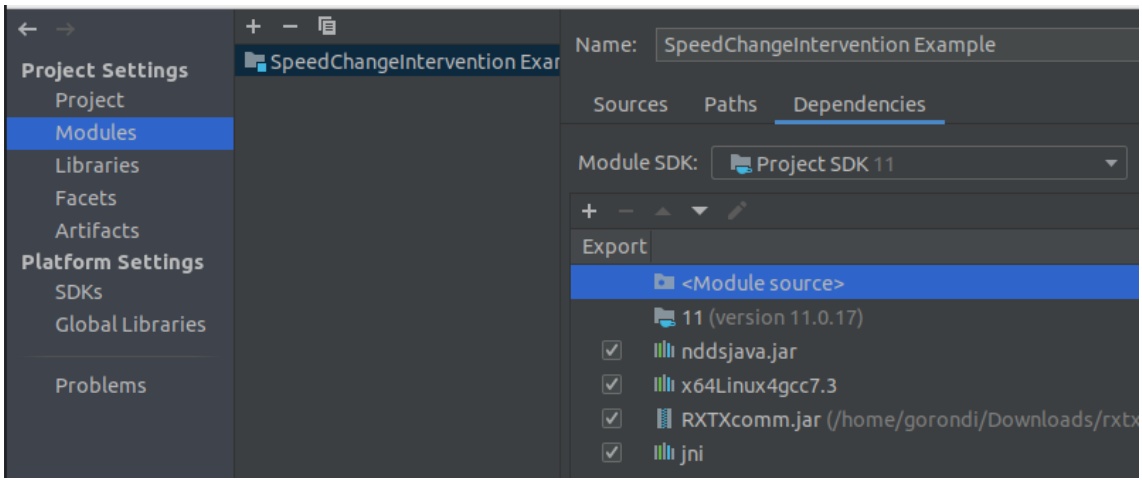

Figure A.4.1: IntelIJ IDE project structure dependencies

A.5 Raspberry Pi 4 migration challenges

While I was trying to install DDS into the RPi microcontroller, I often found myself in a dead end, not knowing what to try next. Below I will try to give a peek into a possible solution which might be able to help anyone who is trying to complete a similar task to mine [14].

1. The OS of host computer (desktop from where files, libraries are migrated) is a Ubuntu 20.04 64bit LTS version. Download the latest available version *.run file* (6.1.1 in my case) for it.
2. Check the processor of your target device (ARMv8 for RPi) and download the target bundles which is compatible with it (ARMv8 target bundle for me). If this step is spoiled you will not run your code on RPi.
3. Run the .run file
4. Generate packages with the .run file created package installer, by running "rtipkginstall <location-of-target-speific-.rtipkg-file>". After this java classes for the choosen platform will be available [15].
5. Open RTI Launcher and generate the DDS java classes from a specified *.idl file* for a selected **platform architecture** (ARMv8). Then at the selected destination *.java files* are generated.
6. Project can be opened in IntelIJ in the following manner described in (A.4).
7. Ssh into the RPi with VSCodes Remote Development extension (A.5.1). Then copy the generated *.java files* into the RPi.
8. Make a ddshome directory, then copy the $NDDSHOME/lib/<OS-and-processor-specific-architecture-version> folder and the $NDDSHOME/lib/java folder from the host to RPi while preserving folder structure [16].
9. Copy the license file to the ddshome directory on target.
10. Set up latest java jdk (jdk 11 for me) into target device.
11. Set up environment variables similarly to host device, just with target specific folder routes (A.5.2).
12. Open project in VSCode Remote developer, there modify *.classpath file* to include your DDS communication specific libraries (A.5.3).
13. Copy the *.java classes* which are already written, into the project folder on the RPi.
14. After every error has gone, it's time to run the program in the target device!

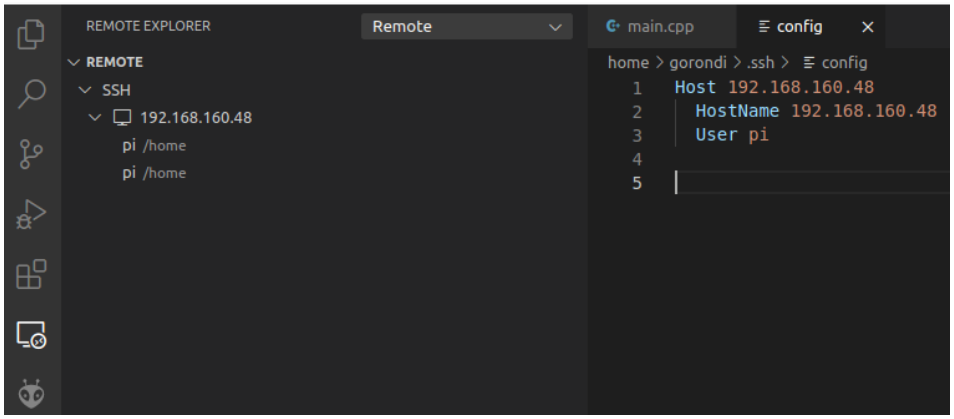

Figure A.5.1: The specific details of the .ssh/config file

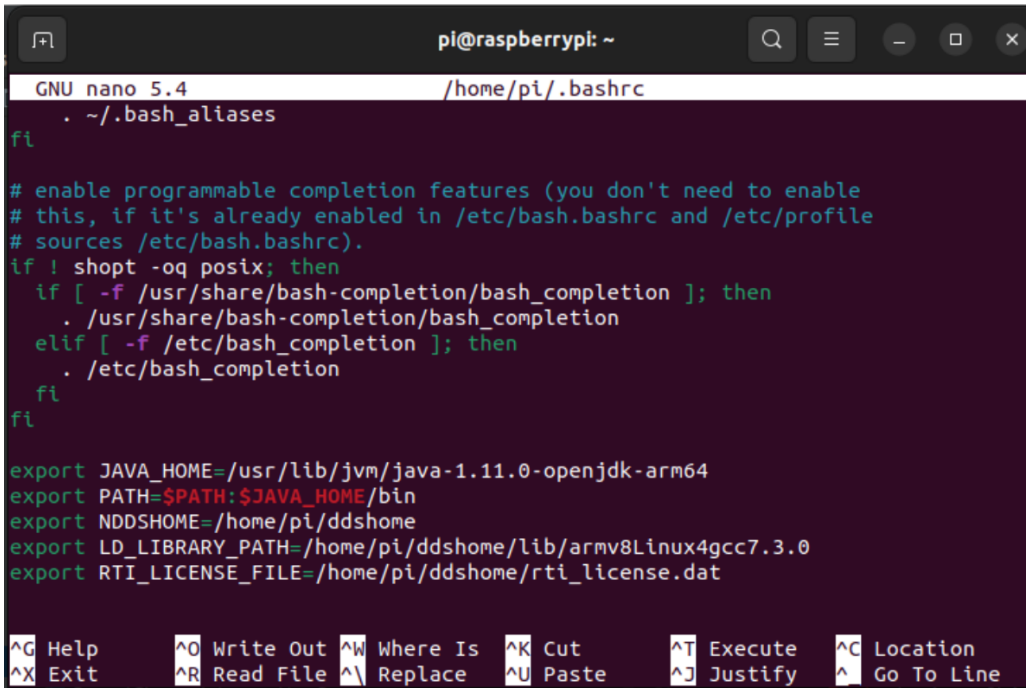

Figure A.5.2: Essential environment variables for DDS on Raspberry Pi 4

Figure A.5.3: The specific .classpath config file for me

Bibliography

[1] ASIL information. URL https://en.wikipedia.org/wiki/Automotive_Safety_Integrity_Level

[2] HUSZERL Gábor. Data distribution services (ppt).

[3] Official espressif site. URL https://www.espressif.com/en/products/socs/esp32

[4] Arduino site for sensor. URL https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6

[5] Wiki for Raspberry. URL https://en.wikipedia.org/wiki/Raspberry_Pi

[6] Site explaining USB cable. URL https://www.l-com.com/frequently-asked-questions%2Fwhat-is-a-usb-cable

[7] C++ implementation of quic sort. URL https://arduinogetstarted.com/faq/how-to-filter-noise-from-sensor

[8] Algoritm which inspired my ways of thinking. URL https://github.com/CoffeeBeforeArch/cpp_data_structures/blob/master/algorithms/sorting_algorithms/quick_sort/quick_sort.cpp

[9] Documentation of JSerialComm library. URL https://fazecast.github.io/jSerialComm/?fbclid=IwAR29qQxFg_CdWzkcOkWI_54cXHuRx_Rzvzf-fcda3XItMX3GZne7kkuWkG8

[10] Arduino tutorial for RXTXcomm library. URL https://xiaozhon.github.io/course_tutorials/Arduino_and_Java_Serial.pdf

[11] DDS host and target files. URL https://community.rti.com/kb/why-do-i-get-error-rti-data-distribution-service-no-source-license-information

[12] DDS host and target files. URL https://www.rti.com/free-trial/dds-files

[13] Incomplete installation tutorial for DDS. URL https://community.rti.com/static/documentation/connext-dds/5.2.0/doc/manuals/connext_dds/RTI_ConnextDDS_CoreLibraries_GettingStarted.pdf?fbclid=IwAR0xAKDiSFuR-mrA0niqOBHmfq3hYTyMS3U497Yhgs1Ub1GZtbl0VE8kkJU

[14] How to install DDS in RPi. URL https://community.rti.com/content/forum-topic/howto-run-rti-connext-dds-raspberry-pi?fbclid=IwAR1fhAPSAJxTVvnoH4HpTgslbOjT4d4Y7Xku88RPT96a4ajtioJfPZYN_HM

[15] Useful thread for DDS RPi integration. Understanding host and target. URL https://community.rti.com/forum-topic/running-rti-installation-raspberry-pi-3

[16] Much needed extention of How to install DDS in RPi. URL https://community.rtidds.com/forum-topic/rti-dds-raspberry-pi?fbclid=IwAR1sSr7YDYfCs5_zT40edcE3bDT8TbrV90T-T1UvoKyn6qqR7ig70PQMXTU#comment-4012