Advanced Software Development for Robotics
Assignment Manual (v2024.2)

Jan Broenink, et al.

Thursday 6th March, 2025

Contents

0 Introduction

0.1 Context

The goal of this practical work is get familiar with soft real-time and hard real-time software development, and the combination of it, in the context of controlling robotic/mechatronic systems (physical machines). We use the RELbot, the in-house developed education mobile robot of the Robotics Education Lab of MSc Robotics.

For the soft real-time part (Assignment set 1 and 3), we use ROS2 (Robotic Operating System 2), version Jazzy, heavily used in modern robotics. The algorithms are programmed in C++. For the development environment, we use a mainstream development tool, VS Code, and run ROS in a dedicated environment, using the virtualisation tool WSL (on Windows) or Multipass (on Mac). ROS needs a particular version of Linux to run in (Ubuntu 2024.04), and that is provided via the virtualisation tools.

Next to this, migrating to developing for the RELbot is straightforward, as the embedded computer is just another dedicated environment, connected to the host machine via a normal network.

For installation, see the Software-Tools Install Manual (Broenink et al., 2025).

For the hard real-time part (Assignment sets 2 and 3), we work with a Raspberry Pi4, using a specifically configured Linux (Ubuntu with the Xenomai real-time patch) and programs written in C++. The Raspberry Pi4s are available at the RaM Lab. In Assignment set 2, only a bare Raspberry Pi4 is needed, without additional hardware. In Assignment set 3, the RELbot is controlled using its Raspberry Pi.

The focus in the assignments of this course lies on developing *good, well-structured solutions*. Code that is 'hacked together' and lacks good structure or programming style does *not* receive full points, even if the output of the program is perfect.

0.2 Organisation

The practical assignments consist of three assignment sets, and must be done in groups of 2; all three assignment sets with the same team of 2 students. Results must be submitted via the UT Canvas page of the course (other ways of submission are *not* accepted). You must supply a report (as PDF) as well as a structured zip file with *all* source code (separated per sub-assignment). The code should be such that it can be compiled (and run) by the teaching assistants on their own computer. Deadlines are published on Canvas.

General requirements for the reports are:

- Reports must be compact: only answers need to be given, but always provide a motivation.
- Use a readable, *one*-column layout.
- Include screen dumps of models / diagrams and results in the report.
- For ROS2 programs, *always* put a node/topic graph in the report (`rqt_graph`).
- For ROS2 programs, submit the structure as indicated in The *Software-Tools Installation Manual*, the appendix on *Structuring ROS2 projects* (Broenink et al., 2025, Appendix F).
- Put also relevant code snippets in the report, but never put the full code in the report, not even as an appendix.
- The sentence *Show that the system works* as regularly written as part of an assignment task, means that you set up, run, and report on some tests to make clear that the system does what you specify it should do. You should address and discuss any anomalies.

Note that the teacher only reads the report and the teaching assistants also check the code in the zip file.

0.2.1 CBL version versus Classical version of the course

MSc-Robotics students who take this class as compulsory course must do the CBL-variant of this course. This means, they can skip certain sub-assignments (those are indicated), and contribute to their CBL case using material from this course. Next to the *framing* Section of the CBL manual, more information is in Appendix E.

Those who do *not* need to do the CBL version, do the *Classical* version of this course, that is all the assigments in this document.

0.3 Support from the Teaching Assistants

There is a weekly Q&A directly after the lecture, so Wednesdays or Tuesdays, 12:45-13:30, in CR 3446, the large meeting room of RaM, directly starting in the first week. You could use the first occasion for support on installing the tools.

During the Lab sessions of Assignment Set 2 and 3, teaching assistants are of course present in the Lab.

You can ask questions regarding the assignments (most notably Assignment Set 1) via *Discussions* available on the Canvas page.

At times when teaching assistants are present (Wednesday's Q&A, Lab sessions) we run the queueing tool Horus to handle online questions synchronously: https://horus.apps.utwente.nl/278.

0.4 Responsibility and fraud

All group members share responsibility for the work handed in. When you distribute work among group members, check each other's work, discuss the work among all group members, do you understand it all?

You must submit original work.

Plagiarism, i.e. copying of someone else's work without proper reference, is a serious offence which in all cases will be reported to the Examination Board. Refer to UT's Student Charter Student Charter (2020).

In cases where a non-trivial amount of work is copied *with* proper reference, indicate which parts are copied and which parts are your own original work. The copied work will not be considered for grading.

0.4.1 What constitutes fraud?

Material written by Arend Rensink is reused here.

When it comes down to handing in assignments, every year there are students who do not understand the borderline between, on the one hand, cooperating and discussing solutions between groups (which is allowed), and on the other, copying or sharing solutions (which is forbidden and counted as fraudulent behaviour). Here are some scenarios which may help in making this distinction.

- **Scenario.** Peter and Lisa are quite comfortable with programming and have pretty much finished the assignment. Mark and Wouter, on the other hand, are struggling and ask Lisa how she has solved it. Lisa, a friendly girl, shows her solution and takes them through it line by line. Mark and Wouter think they now understand and go off to create their own solution, based on what they saw. Is this allowed or not?

Verdict. No problem here, everything is in the green. It is perfectly fine and allowed for Lisa to explain her solution, even very thoroughly. The important point is that in implementing it themselves and testing their own solution, Mark and Wouter are still forced to think about what is happening and will gain the required understanding, though probably they will not get as much out of it as Lisa (explaining stuff to others is about the best possible way to learn it better yourself!).

- **Scenario.** The start is as in the previous case. However, while Mark and Wouter implement their own solution, inspired by that of Lisa, some error crops up which they do not understand. Lisa has left by now; after they mail her, still trying to be helpful she sends them her solution for them to inspect. They inspect it so closely that in the end their solution is indistinguishable from Peter and Lisa's, except for the choice of some variable names and the comments they added themselves. Is this allowed or not?
  Verdict. This is now a case of fraud. Three are at fault: Lisa for enabling fraud by sending her files (even if it was meant as a friendly gesture) and Mark and Wouter for copying the code. Peter was not involved, developed his own solution (together with Lisa) and is innocent.
- **Scenario.** Alexandra and Nahuel are not finished, and the deadline is very close. The same holds for Simon and Jaco. On the Friday night train home, Jaco and Nahuel meet and during the 2-hour train ride work it out together. They type in the same solution and hand it in on behalf of their groups. Is this allowed or not?
  Verdict. This is also a case of fraud. Actually there are two problems here. The first is that both Nahuel and Jaco handed in code on behalf of their groups that had been developed by them alone, without their partners. This is unwise and against the spirit of the assignment (Alexandra and Simon also need to master this stuff!) but essentially undetectable and not fraudulent. The second problem is that the solution was developed, and shared, in collaboration between two groups; this is definitely forbidden. All four students are culpable; Alexandra and Simon cannot hide behind the fact that they did not partake in the collaboration, as they were apparently happy enough to have their name on the solutions and pretend they worked on it, too.

Note that we are not on a witch-hunt here: let us stress again that cooperating and discussing assignments is OK, even encouraged; it is at the point where you start copying or duplicating pieces of code that you cross the border.

0.5 Document History and Disclaimer

When you find a mistake or have remarks about this document, please contact one of the TAs or the teacher.

As we also continuously improve this document, newer versions appear regularly, see its date and version number. Changes are summarized in the Changelog, in Appendix A. The latest version is on Canvas. Be sure to always use the latest version of this guide.

This document has seen quite some versions over the years. Contributors to this document, next to the teacher, are: Gijs van Oort, Luuk Lenders, Ilyas Raoudi, Jasper Vinkenvleugel, Jurrijn Pott, Frank Hooglander, Arnold Hofstede, and Nick in 't Veld.

Development and documentation of the RELbot is by Gijs van Oort, Sander Smits, Marcel Schwirtz, Marjon Kuipers, Artur Gąsienica, and Jan Broenink.

Special thanks to Arend Rensink for the Subsection "What constitutes fraud?".

1 Assignment 1 — ROS2

In this assignment you get familiar with ROS2 and build some projects with it. See the Software-Tools Install Manual (Broenink et al., 2025) for how to get access to ROS2 and the required software.

1.0 Assignment 1.0: Getting familiar with ROS2

ROS2 has good tutorials on the website:
`http://docs.ros.org/en/jazzy/Tutorials.html`.
Do (at least) the following tutorials:

1. Using turtlesim and rqt
2. Understanding nodes
3. Understanding topics
4. Launching nodes
5. Creating a workspace
6. Creating a package
7. Writing a simple publisher and subscriber (C++)
8. Using parameters in a class (C++)

You do not have to hand in anything for this assignment.

1.1 Assignment 1.1: Image processing with ROS2

To get more familiar with ROS2, we do some simple image processing work. However, the focus of this assignment is *not* the image processing itself, but rather the structure of the nodes and topics in ROS2.

1.1.1 Camera input with standard ROS2 tools

Due to Multipass VM or WSL Linux not being able to access the webcam directly, the webcam stream must be sent over an UDP channel. For this, we provide a videoserver script and an updated `image_tools` ROS2 package, see the *Software-Tools Install Manual*, the appendix on *cam2image* (Broenink et al., 2025, Appendix D). It is available on Canvas.

Use the specific `image_tools` ROS2 package (from Canvas), and the standard `showimage` node of the standard `image_tools` ROS2 package to capture webcam footage, publish to a channel and view the channel.

The standard `showimage` node can be started using the following command in a separate shell, X-server enabled (argument `-X`):

```
ros2 run image_tools showimage
```

For the `cam2image` node, set history to `keep_last` and depth to 1. This can be done in two ways:

1. Add and modify the parameters in the `cam2image.yaml` inside the package config folder. This is now not needed, but can be useful in Assignment 1.1.3.
2. Include the parameters as *ROS arguments* in the run command as follows:

```
--ros-args -p depth:=1 -p history:=keep_last
```

Show that the system works.

Note: you might check the tips in Section 0.2 on structuring the ROS2 files, finalising this assignment, and producing the report.

Questions

1. Describe in your own words what the parameters `depth` and `history` do and how they might influence the responsiveness of the system. Are the given parameter values good choices for usage in a sequence controller? Motivate your answer.
2. What is the frame rate of the generated messages? How can you influence the frame rate? Is there a maximum? What determines the maximum?

1.1.2 Adding a brightness node

Create a node that determines the average brightness of the image and, using some threshold, sends on a new topic if a the light is turned on (it is light) or off (it is dark).

Show that the system works.

1.1.3 Adding ROS2 parameters

Change the node in such a way that the threshold is a ROS2 parameter. Show that it is settable from the command line when starting the node (`ros2 run ... --ros-args -p ...`) as well as during run time (`ros2 param set ...`).

Show that the system works.

1.1.4 Simple object-position indicator

A very simple way of detecting the position of a coloured object or bright light in an image is the following:

1. Gray-scale the image.
2. Apply a threshold on the specific colour or brightness of each pixel.
   If the threshold is well-chosen (and there are no other parts with the same colour or brightness in the image), the result is a black image with a large white dot where the coloured object or bright thing is. You may need to reduce background light.
3. Compute the 'center of gravity' (CoG) of the white pixels. This CoG (in pixel coordinates) gives an indication of the position of the centre of the object.

Create a node that, given a camera input, outputs the position of the coloured object or bright light (if there is any) in pixel coordinates. The node should be easy to use, e.g., also when using in a larger project, on a different webcam or with different lighting conditions.

Show that the system works. Also, discuss your design choices.

Note: In Assignment 3, you have to follow a small green ball with the RELbot, using it's front camera. With this subassignment, you could prepare for that.

1.2 Assignment 1.2: Sequence controller

Assignment 3 is on controlling a differential drive robot, called the *RELbot* (Figure 1.1). This robot is driven by two electromotors and when using on a flat surface, it can be considered as a rigid body that can move in 2D space: driving forward or backward or turning. For describing the pose (position and orientation) of the robot, we only need x, y and θ_z, rotation around the vertical axis (z-axis), see Appendix G of the Software-Tools Install Manual (Broenink et al., 2025).

In order to prepare for the final assignment set, in this assignment we explore the ways of creating a sequence controller in ROS2 and control a simulation model of the RELbot.

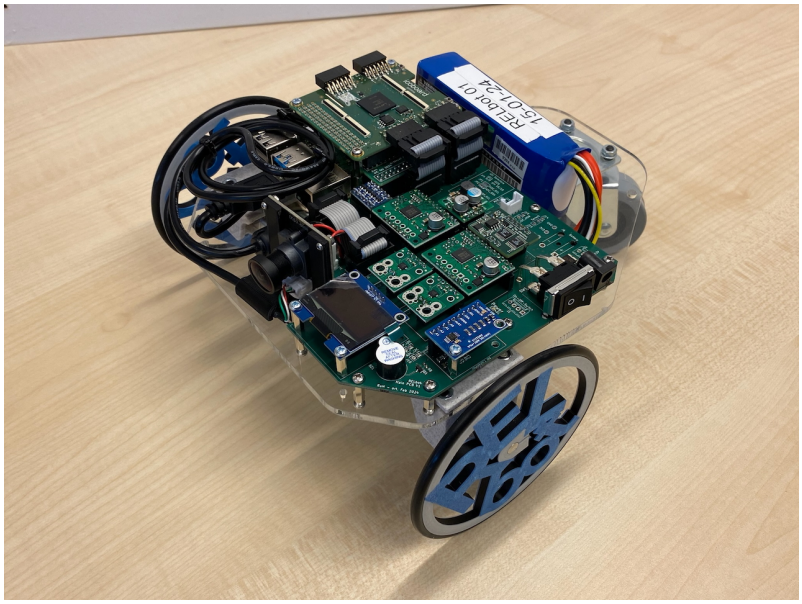

Figure 1.1: Version 1.0 of the differential-drive robot *RELbot*, without top plate and battery cover.

1.2.1 Unit test the sequence controller using the RELbot Simulator

A ROS2 node simulating the RELbot behaviour is available for this assignment (see Canvas). The node needs a webcam stream from a steady webcam (e.g., the webcam of your computer), this is achieved by using the previously used cam2image tool. For details about the use of the RELbot node, see Appendix G of the Software-Tools Install Manual (Broenink et al., 2025).

Create a node that generates a sequence of setpoints and test it using the RELbot Simulator node.

Also create a launch file that sets up all required nodes.

Show that the system works and discuss the design decisions you made.

Questions

1. Plot the setpoint and actual position (use of `rqt` is ok). Explain the results. Do not forget to look at the time constant of the first order system.

1.2.2 Integration of image processing and RELbot Simulator

This sub-assignment can be seen as an intermediate step towards the ultimate goal of having the RELbot Simulator use its own (moving) camera output. In this sub-assignment the data from the your webcam is used directly (software development goes small, testable, steps at a time).

Extend the sequence controller ROS2 node such that the RELbot Simulator follow a coloured object or bright light that you move around in front of your webcam. Keep the old functionality available. Use the steady (non-moving) stream of your laptop camera (not the `/moving_camera_output`) as an input for the light position indicator node of Section 1.1.4 and compute appropriate setpoints from that. Remember to keep the system modular (i.e., keep the node(s) easily re-usable; thus not too specialised).

Show that the system works and discuss the design choices you made.

Questions

1. Is the system you created an open-loop or closed-loop system (why)?

2. Is ROS2 (which is a non-realtime platform) in general, appropriate for this type of systems? Why? Are there any limitations (i.e., when do things go wrong)?

1.2.3 *Classical only:* Closed loop control of the RELbot Simulator

Classical Only
This sub-assignment is *not* for CBL students. Only for those who take the *Classical* version of ASDfR.

In this part you make RELbot look at the coloured object or bright light again, but this time the input of the light-position indicator should be the `/moving_camera_output` of the RELbot simulator (just as in the real case, where the camera mounted on RELbot moves around during operation).

In order to make this work, you need to steer the RELbot pose (position and orientation), $\mathbf{x}_{\text{RELbot}} = [x_{\text{RELbot}} \quad \theta_{z,\text{RELbot}}]^T$, towards the (absolute) position of the light $\mathbf{x}_{\text{light}}$ such that the error, $\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{RELbot}}$, becomes zero.

A convenient controller choice is a first order controller, where the rate of change of the RELbot orientation, $\dot{\mathbf{x}}_{\text{RELbot}}$, is proportional to the error, i.e.,

$$\dot{\mathbf{x}}_{\text{RELbot}} = \begin{bmatrix} \dot{x}_{\text{RELbot}} \\ \dot{\theta}_{z,\text{RELbot}} \end{bmatrix} = \frac{1}{\tau} \cdot \left(\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{RELbot}}\right) = \frac{1}{\tau} \cdot \begin{bmatrix} x_{\text{light}} - x_{\text{RELbot}} \\ \theta_{z,\text{light}} - \theta_{z,\text{RELbot}} \end{bmatrix}; \quad \mathbf{x}_{\text{RELbot}} = \int \dot{\mathbf{x}}_{\text{RELbot}} \, \mathrm{d}t, \tag{1.1}$$

where τ is some time constant (a good start would be $\tau = 1\text{s}$).

Notes

- The position of the coloured spot / bright light in the camera image is precisely $\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{RELbot}}$.
- Moving in the x direction means zooming in or zooming out.
- Moving around the z axis (changing θ_z) means horizontally moving the moving-camera cut out over the webcam image.
- The only way to influence the camera orientation is by sending a setpoint $\mathbf{x}_{\text{set}}$ to the simulator node. Fortunately, the RELbot can track its setpoint rather well, so we can assume that $\mathbf{x}_{\text{RELbot}} = \mathbf{x}_{\text{set}}$, resulting in

$$\dot{\mathbf{x}}_{\text{set}} = \begin{bmatrix} \dot{x}_{\text{set}} \\ \dot{\theta}_{z,\text{set}} \end{bmatrix} = \frac{1}{\tau} \cdot \left(\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{RELbot}}\right) = \frac{1}{\tau} \cdot \begin{bmatrix} x_{\text{light}} - x_{\text{RELbot}} \\ \theta_{z,\text{light}} - \theta_{z,\text{RELbot}} \end{bmatrix}; \quad \mathbf{x}_{\text{set}} = \int \dot{\mathbf{x}}_{\text{set}} \, \mathrm{d}t. \tag{1.2}$$

Develop the system described above. For the integration (second equation of (1.2)) you can use the Forward Euler integration method.

Show that it works (i.e., that the simulated RELbot can track the object). Also discuss your design choices.

Questions

1. Is the system you created an open-loop or closed-loop system (why)?
2. Is ROS2 (which is a non-realtime platform) in general, appropriate for this type of systems? Why? Are there any limitations (i.e., when do things go wrong)?

Note – for both CBL and Classical versions of the course

Keep the nodes you have programmed for this assignment set, as you need them again in Assignment 3.

2 Assignment 2 — Timing in Robotic / Cyber-Physical Systems

2.0 Introduction

In this assignment, you will analyse the timing of a Robotic or Cyber-Physical System, in which the SRT parts are ROS2 nodes. More specifically, it is about the timing of threads in the SRT and in the FRT part of the Raspberry Pi. By doing so, you can find out what SRT and FRT imply with respect to timing of the loops. In particular, what jitter appears in control loops. Factors such as computational load and how busy a processor is obviously affect this, and thus require investigation. Furthermore, of importance is characterising roundtrip times: a bidirectional, synchronised communication between two nodes, like in closed-loop control systems. And of course, how such a roundtrip behaves under several load conditions.

Next to this, by characterising the Raspberry Pi with respect to timing, using a kind of bare implementation (that is, without the ROS2 stuff), you are also setting up a basic timer experiment on the Raspberry Pi.

As preparation for the assignment in Chapter 3, the last part of the current assignment (Section 2.4) is on constructing diagrams of the ECS architecture of the RELbot system.

2.0.1 Raspberry Pi / RELbot in the Lab

The larger part of this assignment set consists of doing timing measurements on a Raspberry Pi. They are situated in the RaM Lab (Carré 3434). We have 6 Raspberry Pis in the form of RELbots available at the same time.

For this assignment, you do *not* need to use the same Raspberry Pi / RELbot during all time slots. The Raspberry Pis are identical and you should have a copy of you files on your local computer anyway, so switching to a different Raspberry Pi should not be a problem.

Each group is allowed to use 2 two-lecture-hour time slots per week on a Raspberry Pi / RELbot, during the weeks this assignment is scheduled. Timeslots are available on Canvas for claiming.

You *must* subscribe to a time slot in Canvas before you are allowed to use it.

See the Software-Tools Install Manual (Broenink et al., 2025) for how to get access to the Raspberry Pis / RELbots in LabOne of RaM, CR 3437.

The time you have available at a Raspberry Pi is limited. Therefore it is vital to be well prepared:

- The programs of Section 2.1 and Section 2.3 (Classical only) can be fully prepared and tested at home on your Virtual Machine. If you program correctly, the code can run both on the VM and on the Raspberry Pi without any code modifications. Note that you *must* recompile though, throw away all object files and executables before recompilation, to force the build system to run a complete compile process again.
- The program of Section 2.2 can not be fully tested on the VM because your VM has no Xenomai functions. However you can do a best-effort to write the program without compiling and testing the final version. VS Code's intellisense / autocorrect already helps preventing typos. Note that most EVL / Xenomai 4 commands have different names than the comparable POSIX commands, such that compiling at 'home' (uh... on your VM) might not help that much. In case the difference is only function parameters, you can keep them during development at the POSIX settings and only change them to the EVL / Xenomai 4 version when you work on the Raspberry Pi.
- Do any post-processing and report-writing later. However, you could do some initial (sanity) check of the results in the lab, to prevent leaving the lab with useless data.

2.0.2 Documentation and References

- Documentation for all POSIX C functions is conveniently embedded in Linux; you can use the command-line command `man` for this, e.g., from the Linux command line:
  `man pthread_create`
  Alternatively, an online version of the man-pages is at
  https://man7.org/linux/man-pages/index.html
- Specifically for documentation on `CLOCK_REALTIME` and `CLOCK_MONOTONIC` (used in Section 2.1), refer to the manpage of `clock_gettime` (they are described in many other man-pages as well, but in less detail).
- Documentation on Xenomai 4 is at
  https://evlproject.org/overview/.

2.1 Timing of periodic non-realtime thread (Ubuntu)

In Linux there are two ways to get timed periodic execution of code:

- using `clock_nanosleep`, in which you repeatedly tell the process to sleep until a certain absolute time, or
- using `timer_create` and `sigwait`, in which you set a periodic timer once and subsequently call `sigwait` each sample to sleep until the next time the timer fires (interrupt).

Usually, the part of the code that is executed periodically is situated inside a `while` loop (or `for` loop in case you only want to run for a known, finite number of iterations).

Write a program that spawns a POSIX thread which has a `while` (or `for`) loop inside that contains timed periodically-executed code. Choose either `clock_nanosleep` or `timer_create`/`sigwait` to accomplish this. The code should be executed every 1.0 ms. Let the loop do some computational work so that it actually spends some time inside the loop.

Run the program twice: without and with extra processor load (from another program).

Investigate the timing of the loop. Discuss any particularities. Does extra computational load significantly affect the timing performance?

- The timing measurements should be taken when the program runs on the Raspberry Pi 4. However, you can prepare and test the program at home. See Section 2.0.1.
- Use `clock_gettime` for accurate time measurement.
- Avoid printing to screen/file IO inside the loop as this may heavily influence the timing of the loop.
- Make sure that the program exits nicely; e.g., the main thread waits for the thread to end, files and other resources are properly closed etc.
- For computational work, consider doing some math. Experiment a little with the amount needed to get a nice processor load (do not flood it though).
- You can use the command line tool `htop` to view the processor load.
- To generate extra processor load extra, you can use the command line tool `stress` (both on the virtual machine and the Raspberry Pi). For this, open an extra terminal or extra SSH connection and start `stress` in there before running your program. Think of suitable parameters yourself (motivate them).

Questions

1. Explain what timing aspects we are interested in and why. Keep in mind that the course is on software development *for robotics*; a typical use case here is doing closed-loop control with a given sample time. Give at least two ways of representing the measured data. Discuss advantages and disadvantages of them.

2. Explain why you have chosen the method you used (either `clock_nanosleep` or `timer_create`/`sigwait`). What are advantages and disadvantages of both methods?
3. What is the difference between `CLOCK_MONOTONIC` and `CLOCK_REALTIME`? Which one is better for timing a firm real-time loop?
4. When using `timer_create`/`sigwait`, what happens with the timer you initialised when the program ends? And what happens when it crashes due to some error?
5. Is it a good idea to use the approach used in this subassignment for closed-loop control of a robotic system? Why (not)?

2.2 Timing of periodic realtime thread (Xenomai)

Make a variant of the program of Section 2.1 in which the periodic loop runs real-time under Xenomai.

Investigate the timing of the loop, that is, display the jitter in a diagram. Discuss any particularities. Does extra computational load significantly affect the timing performance?

- This program must run on Xenomai on the Raspberry Pi 4. Obviously, you can *not* fully prepare and test this at home on your VM. See Section 2.0.1.
- Most POSIX functionality can stay the same; however for EVL / Xenomai 4 names of those functions are different.
- Take care to make the program real-time capable.
- See Appendix B for details on how to compile for EVL / Xenomai 4.
- You can still use `htop` to show processor load. However, this command only shows load from processes run on Linux, not from those running on EVL / Xenomai 4. To show the load by EVL / Xenomai 4 processes, use the command `evl ps -l`. Apart from the load (`%CPU`), this command shows some other statistics, such as the number of *mode switches* (`ISW`) made.
- It is important to be aware of which CPU cores are reserved for Xenomai tasks on the Raspberry Pi. By default, Linux attempts to schedule processes on all available cores. However, you can isolate specific cores for Xenomai by adjusting the kernel boot parameters. The Raspberry Pi is configured such that Linux only schedules processes on cores 0, 2 and 3, leaving core 1 reserved for Xenomai. Even with this setting, you must still explicitly assign a process or thread to a EVL / Xenomai 4 core by setting its thread affinity.

Hint: Keep in mind that factors such as memory allocation, system calls easily disrupt the real-time behaviour. Efficient real-time programming requires careful consideration of these aspects to prevent unnecessary jitter or delays in your system's response times.

Questions

1. For each POSIX / EVL / Xenomai 4 command used in your program, explain *in your own words* what it does and why you need it for a real-time program. If the command has arguments that influence the real-time behaviour (e.g., the scheduling type `SCHED_X`, or the clock type `CLOCK_X`), discuss why the chosen option is the best for real-time performance.
2. Compare the timing performance of this program with the program from Section 2.1. What can you conclude on timing with respect to real-time capability?
3. Is it a good idea to use the approach used in this subassignment for closed-loop control of a robotic system? Why (not)?

2.3 *Classical only:* Timing of two communicating ROS2 nodes

Classical Only
This sub-assignment is *not* for CBL students. Only for those who take the *Classical* version of ASDfR.

Create two ROS2 nodes, `Seq<nn>` and `Loop<nn>`, where `nn` is your group number. Let *one* node provide the timer running on 1 kHz. Let `Seq<`*`nn`*`>` send a message each sample time to `Loop<`*`nn`*`>`. Let `Loop<`*`nn`*`>` send a message back to `Seq<`*`nn`*`>` *in reply to* the message it got from `Seq<`*`nn`*`>`.

Run the program twice: without and with extra processor load. Investigate the timing, that is show jitter, round-trip time, and jitter of round-trip time graphically. Discuss any particularities. Does extra computational load significantly affect the timing performance?

- Extra provisions might be needed to get the ROS2 nodes running due to the mutual dependency w.r.t. inputs (subscribers).
- Again, the timing measurements should be taken when the program runs on the Raspberry Pi 4. However, you can prepare and test the program at home. See Section 2.0.1.
- Take into account the hints in Section 2.1.
- To match which sent and received messages belong together `Seq<`*`nn`*`>`, it is wise to send unique messages (e.g., with unique reference numbers).
- When testing the round-trip time of messages, take into account that the policies which are set for the communication between ROS2 nodes affect the results. An example is enabling or disabling message buffering. Recall from Section 1.1.1 that you can affect these policies with `ros2 run ... --ros-args -p ...`
  For a list of arguments, see https://docs.ros.org/en/humble/Concepts/About-Quality-of-Service-Settings.html
- Usually, a ROS2-node executes indefinitely and is stopped by completely stopping the program containing the node with `Ctrl+C`. However, when doing measurements, you probably want to run the experiment for a known, finite amount of time and do some post-processing in the same program afterwards. One way of accomplishing this, is to stop the node from within, which is done by calling `rclcpp::shutdown()` inside its functions. Doing this is not mandatory though.

Questions

1. Compare the results from this subassignment with the results from Assignment 2.1. Discuss the results.
2. Motivate your tuning of communication policies, that is, which parameters you have used, or not.
3. Discuss the analogy between the node architecture used in this subassignment and a typical robot control application. Put in your answer also the reasoning in what node you have placed the timer.
4. Is it a good idea to use the approach used in this subassignment for closed-loop control of a robotic system? Why (not)?

2.4 RELbot CPS Architecture Diagrams

In this subassignment, you will construct diagrams to think ahead for the realisation of the ECS software in the assignment of Chapter 3. Doing so, gives you a flavour of working along approaches of *model-based software development.*

Details on the RELbot including its hardware are in the Software-Tools Install Manual (Broenink et al., 2025).

The Xenomai 4 - ROS2 Framework, second generation (XRF2), provides two pieces of functionality:

- Communication software to let ROS2 programs and EVL programs communicate with each other, while adhering to both the ROS 2 structure and respecting timing demands of EVL at the corresponding side of the communication, of course.
- A skeleton for code to be run on EVL / Xenomai (firm real-time) in a while loop, whereby the computation part to execute is (preferably) generated by 20-sim and a FSM to coordinate which computation part must run in which *logical* state of the robot.

See Appendix C for more info on using XRF2. Background information and rationale for its design are in the MSc report of (Raoudi, 2024).

Block diagram

Construct a high-level block diagram of the structure of the Cyber-Physical System for which you are developing the embedded software. Cover the complete CPS / Robotic system.

- Detailing should be to the level of ROS 2 nodes and topics or the control stack as regularly shown on the lecture slides; similar detail is expected for the other parts. So, distinguish essential functionality and draw that as separate items (blocks).
- Clearly indicate the boundaries between non/soft/firm/hard real-time of the Cyber part.

Details of each block

Per item and communication channel, specify at least the following aspects:

- For each item:
  - Name.
    The name must clearly indicate the function of the item; you may add a concise description separately from the diagram.
  - Environment (e.g., ROS2, Non-RT, Xenomai, mechanical).
  - Sample time (if applicable).
- For each communication channel:
  - Signal Name.
  - Type of channel (e.g., ROS topic, XDDP, IDDP, SPI, electrical).
  - Data type and size for vectors / arrays.
  - For signals representing physical quantities: unit (deg, rad, ...).

To avoid cluttered diagrams, or too small text sizes in the diagrams, you can use colour coding or abbreviations (adhere to conventions and standards). While doing so, do provide a legend or a list explaining the abbreviations.

FRT code parts to 'fill' the FSM

Specify what computation must be executed in the *user-programmable states* of the FSM of XRF2. Write it as pseudo code, and clearly indicate to which state of the FSM it belongs.

Specify which commands must be sent over the command channel (`XenoCmd`) to let the computation parts run.

The XRF2 FSM is shown in Figure 3.10 of Raoudi (2024), and on Slide 3-52 (Slide 52 of Lecture 3). The XRF2 implementation overview is shown in Figure 3.8 of Raoudi (2024), and on Slide 3-54.

Questions

1. Why are the states needed as you specified here? And why are the states you did *not* specify here *not* needed?

2. Assume the RELbot has a basic robot arm with basic gripper on its top, like in Figure 2.1. What states of the FSM would need computations? Indicate what functionality these (extra) computations must have.
3. Assume distance sensors are mounted next to the license plate for detecting obstacles (for instance Time-of-Flight sensors). In what part in the CPS should these be placed? In what block of the control stack should the sensor signal go to, and what function should it feed?

Figure 2.1: Idea of a simple robot arm that could be placed on top of the RELbot (Waveshare, figure is adapted).

3 Assignment 3 — Integration, Controlling a robot

This assignment still needs to be checked.
Furthermore, some parts will be adapted, as these were unclear last year.

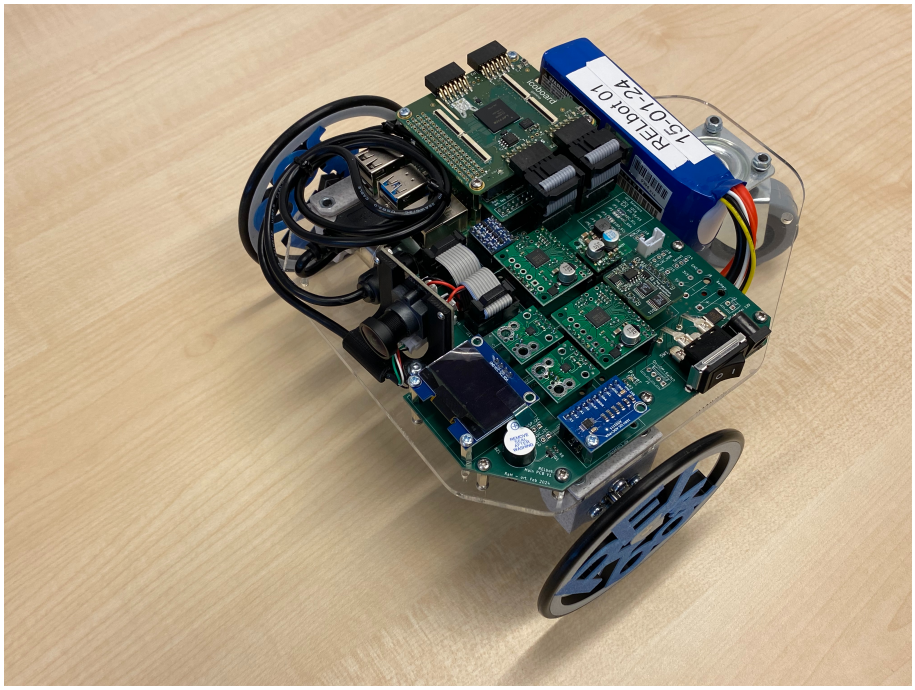

Figure 3.1: RELbot v1.0.

3.0 Introduction

In this final assignment set, you will control the real RELbot (version 1.0, Figure 3.1), such that it follows a bright light. For this, you combine the ROS 2 nodes you made in Assignment Set 1 with a firm real-time controller part. All run on the Raspberry Pi of the RELbot, the SRT parts on Linux (Ubuntu 22.04) and the FRT parts run on EVL / Xenomai 4.

3.0.1 RELbot in the Lab

The larger part of this assignment set consists of developing code for and on the RELbot (on its Raspberry Pi actually). They are situated in the RaM Lab (Carré 3434), the same location as for Assignment Set 2. From 2 April onwards, we have 6 RELbots available at the same time (next to 9 RELbots for the course SDfR also at the same time).

For this assignment, you do *not* need to use the same RELbot during all time slots. The RELbots are identical (except a small variation in some physical parameters) and you should have a copy of you files on your local computer anyway, so switching to a different RELbot should not be a problem.

Each group is allowed to use 2 two-lecture-hour time slots per week on a RELbot, during 3 weeks on Mondays, Wednesdays (of course not during the lecture of 3 April), and Thursdays during week 14, 15, and 16 (Quartile week 8, 9, and 10) so 2 — 19 April.

You *must* subscribe to a time slot in Canvas before you are allowed to use it.

See Appendix **??** for details on how to log in to the Raspberry Pis of the RELbots. Furthermore, presenting graphical output on your laptop of a program while running it on a RELbot using *X-forwarding* is explained **??**.

The time you have at a setup is limited, therefore it is vital to be well-prepared. Just like for Assignment Set 2.

Handing in a ROS2 project needs some specific action, see **??**. Just like for Assignment Set 1.

3.0.2 Planning and intermediate feedback

For each subassignment you have the opportunity to let a TA check your work. During that check, you must demonstrate to the TA that the subassignment works. The TA can then provide you with tips. Use this check as a dedicated feedback moment.

To support you keeping on track, a suggested planning for this assignment set is given in Table 3.1.

Note that week 13 is shorter, due to late publication of this Assignment Set and UT being closed on Friday.

Week	Assignment	Topic
13*	3.1	Block Diagram
14	3.2	ROS on RPi of RELbot (or separate one)
15	3.3 + 3.4	LoopController code; Tests on Robot
16	3.5 + 3.6	Full integation; Extras

Table 3.1: Suggested planning.*Needs updating*

3.1 Assignment 3.1: ROS 2 on the Raspberry Pi of the RELbot

Reproduce the experiment from Section 1.2.3 on the Raspberry Pi of the RELbot.

Show that the system works.

- Re-use the nodes you developed in Assignment Set 1.
- The RELbot USB camera is connected to the Raspberry Pi.
- `rqt` is not installed on the RELbot's Raspberry Pi (it can*not* easily be installed). Fortunately, `showimage` *is* available.
- To show graphical results, you can use *X forwarding*, the same as during ROS 2 development using the VM for Multipass we provide. See **??** for details. The RELbot's Raspberry Pi does not have a graphical environment, so it is quite obvious that you cannot see contents of graphical windows directly on the Raspberry Pi.
- You can use the command line tool `htop` to view the processor load.
- Store results of this subassignment separately, that is, prepare it for submission (see **??**), and set it apart in a separate directory.

Note: Due to a small delay in the production of the actual RELbots, we might provide you with an Raspberry Pi plus webcam only. This is unfortunately an example of hardware not being ready in time to develop / test other parts of the CPS, stressing the need and usefullness of simulation during the development process.

Questions

1. Does the system perform as well as on your own laptop? Why? What is the processor load?

2. Did you have to make changes to the experiment to get it running on the Raspberry Pi? If so, why was it necessary? Could you have avoided it if you had done the experiment in Assignment 1.2.3 differently?
3. Did you have to make changes in the nodes you developed in Assignment 1.2.3? If so, why was it necessary? Could you have avoided it if you had developed the nodes in Assignment 1.2.3 differently?

Sign off and results

You can show the result of Assignment 3.1 to one of the TAs in order to receive feedback. You can use this as a dedicated feedback moment.

3.2 Assignment 3.2: FRT software architecture on RELbot's Raspberry Pi

In this assignment, you develop the architecture (software structure) of the Firm Real-Time software for the RELbot. This is the FRT part *without* adding the loop-controller code and other functionality. It is also about using the brand new ROS 2 – Xenomai 4 bridge and framework, and test it separately. See Appendix C on how to use the ROS 2 – Xenomai 4 bridge and framework.

Set up the FRT architecture and connect that to both the ROS 2 side and the RELbot hardware, using the structure of Figure C.3, but *without* any controller functionality. In fact, you are creating a test bed for the architecture of this part of the system.

Construct a separate test bed to test your FRT part with its connections, whereby:

- The ROS 2 node *only* generates signals to feed to the FRT part.
- The FRT program passes values to its neighbours (ROS 2 node and RELbot hardware)
- the RELbot is on its stand, to prevent moving away. You do drive the motors, so the wheels turn!!

Show that the system works, discuss your design choices. Record / Store the results of your tests to add to your report of this assignment set.

Remarks and Tips

- *Keep the robot on its stand!!*
  This to prevent unintendedly running away of the RELbot, which can cause serious damage.
- Make a separate ROS 2 node with essential functionality for testing purposes only. You could use ROS parameters to select different input patterns, to have a variaty of tests easily available.
- One of the ways to produce the architecture for the loop controller is to create an 'empty' loop controller in 20-sim and generate code from that, and use that in the framework. This might ease the work in the next assignment.
- You may only extend the parts of the ROS 2 – Xenomai 4 bridge / framework as indicated in Figure C.1 and Figure C.2. You would destroy modularity otherwise. If you need extra functionality, use the places where it is allowed to extend / add code. If you think you absolutely need to update code elsewhere, discuss with the teaching assistant first.
- Be careful what diagnostic / debug statements you use in the Xenomai 4 part, as stepping out the *out-of-band* situation is quite easy, and often happens unintendedly. You could use `evl_print()` to show results / numbers in a Xenomai 4 terminal window.
- To test the signal path of the encoder signals, you can carefully rotate the wheels by hand.
- To test the signal path of the PWM signals, you can measure the result of the wheel movement by some external means.

- Be sure to have the motors turn positively (from the point of view of steering value) when the robot has to move forward (camera faces in the forward direction). The encoders should then also give their pulses such that a increasing value of the angle is produced.

Questions

1. What are advantages and disadvantages of separately testing the FRT skeleton part first, as done in this assignment?
2. What is the advantage of starting with an 'empty' loop controller in 20-sim and use that as a starter for Assignment 3.3?

Sign off and results

You can show the result of Assignment 3.2 to one of the TAs in order to receive feedback. You can use this as a dedicated feedback moment.

Note: Store results of this subassignment separately, that is, prepare it for submission, and set it apart in a separate directory, adhering to the **??** for the ROS 2 part. Describe your tests / experiments and add relevant test data to the report.

3.3 Assignment 3.3: Loop-Controller algorithm in FRT part

Complete the Firm Real-Time software for the RELbot, by putting the loop-controller algorithm in the FRT software architecture of the previous assignment (Assignment 3.2). The loop controller is available in the 20-sim model of the RELbot (simplified version), see Appendix D.

Generate the loop-controller code from the 20-sim, put that in your FRT software framework.

Functionality of the Measurement & Actuation (M&A) block can be put in the functions `preProc()` and `postProc()`. Motivate the choice of what you put in these parts, especially on scaling factors.

You can reuse / extend the testbed of Assignment 3.2. You could even do a few tests that are the same as in the previous assignment.

Show that the system works and validate the correctness of controller as final test by letting the RELbot ride in a straight line, and let it make a 90° corner.

Remarks and Tips

- *Keep the robot on its stand, except for the final test!!*
  This to prevent unintendedly running away of the RELbot, which can cause serious damage.
- Appendix D describes a 20-sim model with a very simple dynamic model of the RELbot and an appropriate loop-controller.
- Keep in mind the block diagram you made in Assignment **??**.
- Make sure the directions (i.e., which direction is called positive) adhere to those mentioned in Appendix **??**.
- For the M&A (Measurement and Actuation) functionality, use the `preProc()` and `postProc()` functions respectively, see Appendix C describing the ROS 2 – Xenomai 4 framework.
- Do first some basic tests, with the RELbot on its stand.
- *Let the RELbot ride only after you have checked its loop-controller functionality thoroughly while keeping the robot on its stand.*
- ***!!! Check first with the TA before let the robot ride on the table / floor !!!***
- You are not allowed to modify the files generated by 20-sim, as that would destroy modularity. Of course, updating values of parameters is allowed, as this is *doing experiments* using the generated code.

- You may only extend the parts of the ROS 2 – Xenomai 4 bridge / framework as indicated in Figure C.1 and Figure C.2. You would destroy modularity otherwise. If you need extra functionality, use the places where it is allowed to extend / add code. If you think you absolutely need to update code elsewhere, discuss with the teaching assistant first.
- If you do not want to use 20-sim, you can use any other modeling package (e.g., Simulink) for code generation. In that case, you need to design the position controller yourself, in such a way that it fits to the provided interface.
  Note that you *must* do code-generation from some modeling package; writing the controller directly in C/C++ is not accepted.

Note that testing precise controller functionality is not easy without letting the RELbot ride, as unfortunately there is no simulation model of the RELbot hardware available that runs on Xenomai or can be connected to the FPGA.

Questions

1. What are the advantages and disadvantages of doing code generation (as opposed to writing the controllers directly in C/C++)?
2. Implementing the M&A block by 'embedding it in the controller loop (preProc() and postProc())', is just one way to do it. Describe three alternatives (including the one just mentioned) to implement the M&A block in the framework you are using. Discuss advantages and disadvantages (in terms of maintainability, ease of implementation, modularity, computation speed, ...). Is the chosen implementation your preferred one? Why?

Sign off and results

You can show the results of Assignment 3.3 to one of the TAs in order to receive feedback. You can use this as a dedicated feedback moment.

Note: Store results of this subassignment separately, that is, prepare it for submission, and set it apart in a separate directory, adhering to the **??** for the ROS 2 part. Describe your tests / experiments and add relevant test data to the report.

3.4 Assignment 3.4: Final integration

Finally, integration time!!

Combine the image-processing and loop-controller parts to let the RELbot follow a bright light / spot.

Create the full system of 'RELbot following a bright light' by composing all blocks you have built and individually tested in the previous sections.

Show that the final system works; discuss your design choices.

Draw a block diagram of the architecture / structure of the total system (CPS), such that it describes the results obtained in this last assignment.

Note: this block diagram must be a representation of the real situation. Obviously, it is *not* identical to the answer of Assignment **??**.

Remarks and Tips

- *Keep the robot on its stand, especially during the first tests of the integrated system!!*
  This to prevent unintendedly running away of the RELbot, which can cause serious damage.
- You might tune the controllers such that the RELbot reacts rather slow, to prevent unintendedly riding too fast. After some first tests you could let the RELbot ride faster. However, this is absolutely not necessary.

- Generate some measurements to underpin that the RELbot really does what you describe in the report about its functioning.
- Photos or videos are appreciated as 'proof' of a functioning RELbot. In case your videos are really large w.r.t. file size, a link to some shared space is appreciated.

Questions

1. Compare the resulting system block diagram with the block diagram from Assignment **??**. Indicate the differences, and describe why the final implementation deviated from the original diagram.

Sign off and results

You can show the results of Assignment 3.4 to one of the TAs in order to receive feedback. You can use this as a dedicated feedback moment.

Note: Store results of this subassignment separately, that is, prepare it for submission, and set it apart in a separate directory, adhering to the **??** for the ROS 2 part. Describe your tests / experiments and add relevant test data to the report.

3.5 Assignment 3.5: Extra functionality (Bonus assignment)

In this bonus assignment you can add functionality to the, already working, setup. Some ideas:

- Start-up procedure
- Safety layer: RELbot stops after seeing an obstacle.
- Advanced behaviour in the case no bright light is seen.
- Extra safety: stop the movement of the whole robot if one of the degrees of freedom does not function as expected.
- Your own ideas...

Describe what extra functionality you want to make and how it will influence the behaviour of the whole system. Explain how you implemented the extra functionality. Motivate the design choices. Show and discuss test results.

Note: As with all assignments, the focus is on developing *good, well-structured solutions*. A very cool feature which is just 'hacked' into the system will not give you points.

Sign off and results

You can show the results of Assignment 3.5 to one of the TAs in order to receive feedback. You can use this as a dedicated feedback moment.

Note: Store results of this subassignment separately, that is, prepare it for submission, and set it apart in a separate directory, adhering to the **??** for the ROS 2 part. Describe your tests / experiments and add relevant test data to the report.

3.6 Final Submisison of Assignment Set 3

For the final submission, obviously combine all results of the subassignments of this Assignment Set. Keep in mind the following:

- Adhere to **??** for providing ROS 2 results of the subassignments.
- For each subassignment in which you produce Xenomai 4 software, submit a structure as indicated in Appendix C, most notably Figure C.1. Indeed, the untouched files (red-coloured annotations) appear several times in you submission. Just accept this inefficiency.
- Provide readme files containing the build and run commands, allowing for easy checking your software.

- Of course, adhere to Section 0.2 w.r.t. what to submit.
- The work must be submitted at least three days before the first of your group takes the oral exam.

A Changelog

- **Version 2024.2, 6 March 2025**
  - Assignment 2 complete.
  - **Note** Assignment 3 must still be checked and updated.
- **Version 2024.1, 6 February 2025**
  Initial version, consisting of:
  - First version of the Assignment Manual.
  - **Note** Assignment 2 must still be updated (until 2.3 ready).
  - **Note** Assignment 3 must still be checked and updated.

B Working with the Raspberry Pi

This appendix gives some hints on how to develop, run and debug code on the Raspberry Pi from within Visual Studio Code (VSCode). It continues on **??**, so read that first.

B.1 Compiling/debugging generic C/C++ code on the Raspberry Pi

Use the Remote-SSH extension to login onto the Raspberry Pi. Make sure you have installed the C/C++ Extension Pack, both locally and on the Raspberry Pi.

The following, standard, `tasks.json`, `launch.json` and `c_cpp_properties.json` should get you started. When using these, you can press F5 to build and debug the C-file you are currently editing.

B.1.1 tasks.json (Raspberry Pi)

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C++: gcc build active file",
      "command": "/usr/bin/g++",
      "args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}",
        "-lpthread","-lrt"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": "build",
      "detail": "compiler: /usr/bin/gcc"
    }
  ]
}
```

B.2 launch.json (Raspberry Pi)

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "gcc - Build and debug active file",
      "type": "cppdbg",
      "request": "launch",
```

```
      "program": "${fileDirname}/${fileBasenameNoExtension}",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${fileDirname}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ],
      "preLaunchTask": "C/C++: gcc build active file",
      "miDebuggerPath": "/usr/bin/gdb"
    }
  ]
}
```

B.3 c_cpp_properties.json (Raspberry Pi)

```
{
  "configurations": [
    {
      "name": "Linux",
      "includePath": [
        "${workspaceFolder}/**"
      ],
      "defines": [],
      "compilerPath": "/usr/bin/gcc",
      "cStandard": "gnu99",
      "cppStandard": "c++14",
      "intelliSenseMode": "linux-gcc-arm"
    }
  ],
  "version": 4
}
```

Note that you can also use the same ways of compiling C/C++ code that are described in **??**.

B.4 Compiling for Xenomai on the Raspberry Pi

B.4.1 Introduction

Xenomai 4 is a framework that adds a new core to the Linux kernel called the EVL core, which can be used as a high-priority execution stage that enables real-time behaviour. This approach is described as a "dual kernel" model. To be able to use Xenomai 4 and its userspace utilities, the Linux kernel needs to be specificially compiled with the Xenomai 4 patches. This means you will not be able to compile and test code written for Xenomai 4 on your own computer, but only on the Raspberry Pi devices provided for this course, which are set up with Xenomai 4.

To get started with `libevl`, the userspace library for creating programs that use Xenomai 4, a regular C or C++ program needs to be written that uses `pthread_create` to spawn a new

thread. This thread can then be attached to the EVL core by using the `evl_attach_`-family of functions. More on these functions can be found in the documentation.

The newly created thread is now an EVL thread, which is either in-band or out-of-band, where the latter is the one that guarantees real-time behaviour. Whether it is in-band or out-of-band depends on the scheduling settings, so make sure these can be set explicitly: see `PTHREAD_EXPLICIT_SCHED` for this.

Setting the correct scheduling policy and priority is very important to make sure a thread is actually running out-of-band. By default, the policy is set to `SCHED_WEAK`, which is a non-real-time policy. Other options can be found in the documentation. Decide for yourself what the best approach for your application is, and then set this policy explicitly, either when starting the thread initially (using the POSIX functions) or later when the thread is attached to the EVL core (using the `libevl` functions). When a thread is attached the function `evl_is_inband` can be used to check the state of the current thread.

While attached to the EVL core, make sure only functions internal to EVL are used as opposed to services from the system C library. Calling functions from the system C library is possible, but the side effect is that the process will be moved in-band, which does not guarantee real-time behaviour. For example, consider using `evl_printf` instead of `printf` (while keeping in mind the overhead of such a "heavy" I/O operation).

When a thread is created and attached, it is pinned to the CPU core it was originally scheduled on, and inherits the scheduling policy that was set during creation. While the dual kernel design does not make this strictly necessary, on the provided Raspberry Pi images a CPU core is intentionally kept free from other processes such that your real-time process can use all CPU time. Functions such as `sched_setaffinity` have to be used to schedule the thread on a certain CPU core, but keep in mind that this is a function that is provided by the system C library and not by Xenomai, which means that it should be executed before attaching to the EVL core.

Good practice for using Xenomai is to check the return values of functions to determine where something goes wrong. For example, the function `evl_attach_thread` returns the file descriptor of the new thread on success (which is a positive integer), and reserves all negative values for error codes. These error codes can be "decoded" to a descriptive string using the function `strerror`, which can then be printed to the console:

```
efd = evl_attach_self("evl_thread");

if (efd < 0) {
  evl_printf("error attaching thread: %s\n", strerror(-efd));

  // handle error...
}
```

These possible return values are listed in the documentation for all functions.

B.4.2 Compilation

To compile a program for real-time running in Xenomai, you need to make the compiler link to real-time functions instead of the normal non-realtime variants. In order to do this easily, Xenomai provides a `pkg-config`-file, which can output all required compiler and linker flag in an easy manner. The file is located in `/usr/evl/lib/pkgconfig/evl.pc`. In the simplest case, where you want to compile and link a C-file (or multiple files) in one `gcc` command, the required flags can be obtained as follows (try it to have an idea what is going on):

```
pkg-config /usr/evl/lib/pkgconfig/evl.pc --cflags --libs
```

Note that this only adds the flags that are required to use Xenomai headers and link against EVL, so any other required libraries (`pthread`, for example) need to be added by hand. One can embed the output of `pkg-config` directly inside a new command with the bash command substitution feature[1], for example the compile command. With that, compiling a file, say `test.c` for Xenomai from the command line becomes as easy as:

```
gcc test.c -o test $(pkg-config
   /usr/evl/lib/pkgconfig/evl.pc --cflags --libs)
```

This can also be used in `tasks.json`; the required arguments should just be given as one of the `"args"`. Unfortunately, VSCode does not support shell substitution in `tasks.json`. To circumvent this issue, we need to manually add the required compiler flags to our file. This results in the following contents:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C++: gcc build active file (Xenomai 4)",
      "command": "/usr/bin/g++",
      "args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}",
        "-I/usr/evl/include", "-L/usr/evl/lib", "-levl"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": "build",
      "detail": "compiler: /usr/bin/gcc"
    },
  ]
}
```

B.4.3 launch.json (Raspberry Pi, Xenomai)

Running a real-time program in Xenomai can only be done as root, i.e., you should run the program (say, `test`) as `sudo ./test`

This can be done from within VSCode, although it seems that using breakpoints does not work out-of-the-box. A small script at `/usr/bin/gdb-sudo` can be created on file system of the Raspberry Pi which simply invokes the debugger `gdb` as root:

```
#!/bin/bash
sudo /usr/bin/gdb "$@"
```

[1] When using `$(command)` in a command, the `command` is executed and its output is substituted there

Now in `launch.json` the only thing to do is to change the `"miDebuggerPath"` from `"gdb"` to `"gdb-sudo"`.

C XRF2, the Xenomai 4 – ROS 2 framework

This appendix needs to be updated, as it currently describes XRF1, which is significantly different to use.
Essence of new parts have been presented during the first 3 lectures.

The Xenomai 4 – ROS 2 framework, second generation (XRF2) provides a skeleton to set up data communication between a ROS 2 node and code running on EVL / Xenomai 4. This code consists of a controller generated by 20-sim, and pre- and post-processing functions that are called before and after each call to the 20-sim generated controller.

Note: The controller function must be called `LoopController()`. Indeed, hardcoded. Sorry for that. We did not have time to use the submodel name of the code generation output of 20-sim. This will of course be taken care of in a later version.

C.1 Structure of the Framework

The file structure of the framework is shown in Figure C.1.

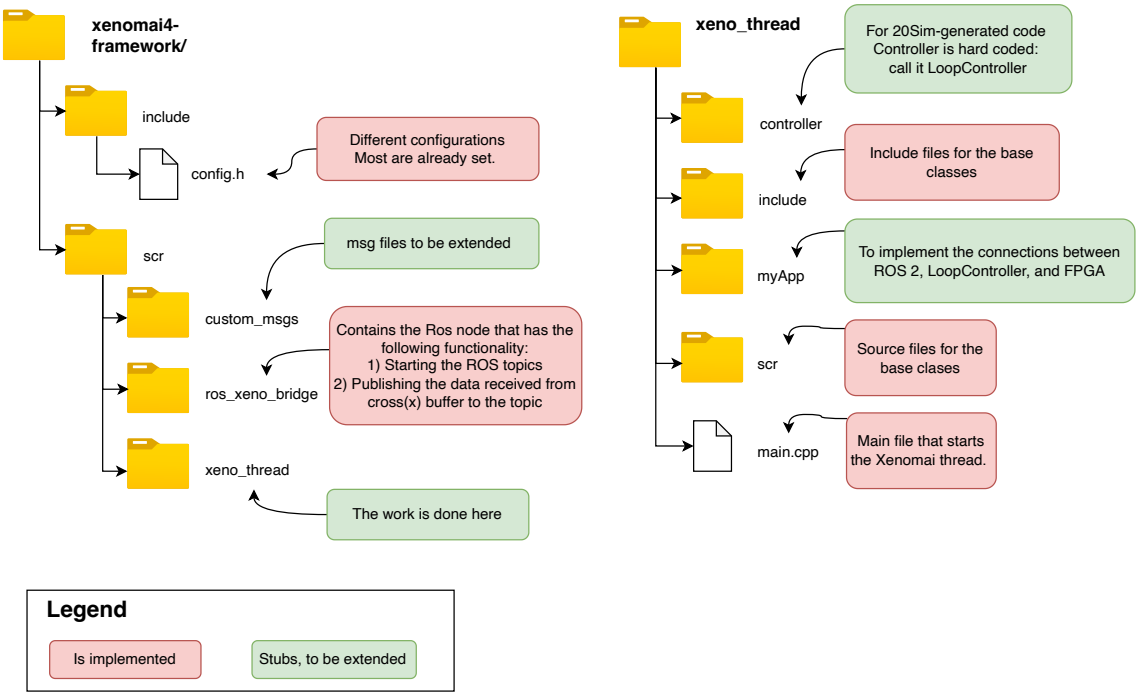

Figure C.1: File structure of the framework. Files in green annotated folders must be extended

The class diagram of the framework is shown in Figure C.2.

C.2 Using the framework in your project

The framework is available on the RELbots in the lab.

The *ToDo.txt* file, available on Canvas, contains the steps you have to perform to extend the provided code. It also contains the commands to build and run the augmented framework, both on the Linux / ROS 2 side and the Xenomai 4 / EVL side.

Note: Always start the Xenomai 4 side before starting the ROS-Xeno-Bridge ROS 2 node.

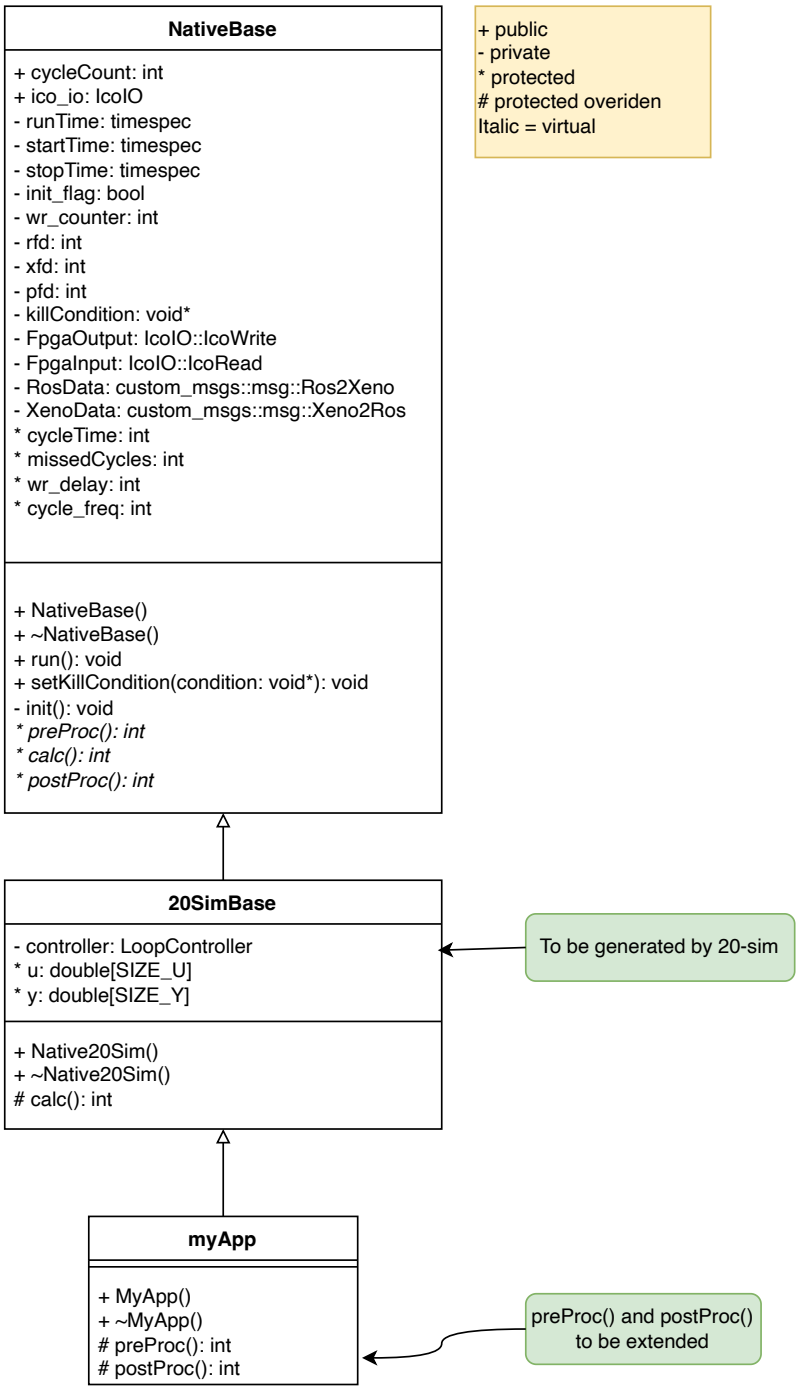

Figure C.2: Class diagram of the framework. Parts in green must be extended

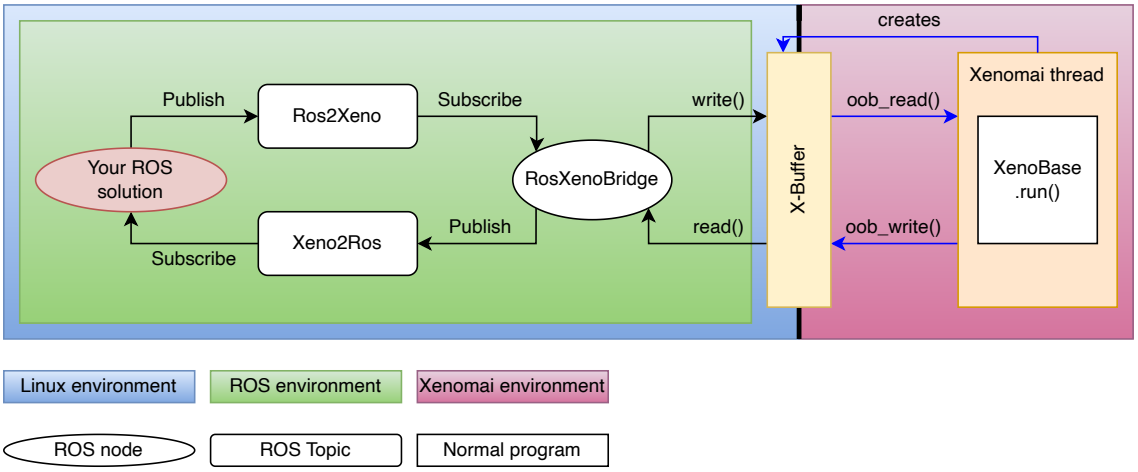

Figure C.3: Structure of ROS-Xenomai-Bridge connected to a ROS 2 node and a Xenomai program

D 20-sim and the 20-sim model of RELbot v1.0

D.1 20-sim

20-sim (Controllab Products, 2008) is a modeling and simulation package by Controllab Products. It consists of a model editor, a simulator and much more.

D.1.1 Installation

It only runs on Windows. The current version can be downloaded from `https://www.20sim.com/downloadnext/` skipping filling in your personal data. The license key is avaliable on Canvas.

Note that you do not need the newest version, although the license key provided on Canvas is tight to the current version, and has a restricted time period of validity.

For Mac users, you would need a virtual machine running Windows. Virtual Box runs only on X86-based Macs and for ARM-based Macs, you can use UTM as virtual machine engine. Parallels is another good virtualisation tool for all kinds of Macs, but needs license costs.

In case none of your group succeeds in running 20-sim you can request the generated code, and use that, instead of checking the model and generating code yourself.

D.1.2 Code generation

Code generation is a function found in the simulator (`Model|Start Simulator`), not in the editor (this is because the code generator needs already an checked-and-precompiled model and that's done when invoking the simulator). The code generation function is in the simulator in the menu `Tools|Real Time Toolbox`, or use the Code Generation button. You need to generate a *C++ class for 20-sim submodel*.

You can only generate code of individual submodels; if you want to generate a single C++ class from multiple submodels then you need to group the submodels first ('implode').

The default location where the generated files are saved is `c:\temp`.

D.2 The 20-sim model

We provide a 20-sim model of RELbot: both the mechatronic part (*plant*) and a loop controller, see Figure D.1. For the plant (mechatronics) part, it is assumed that

- the RELbot ride on an flat (horizontal) surface, and
- the RELbot rides straight forward and backwards, or takes corners rather slowly.

This model is available on Canvas.

The Loop Controller itself is shown in Figure D.2.

The velocity setpoint is integrated to an angle (rotational position), as the plant delivers an encoder value, which is a angle (rotational position), and we assume a rotational velocity as input.

The measured values, *PosLeft* and *PosRight*, are in radians, and the steering values *SteerLeft* and *SteerRight* are in percentages, implying / assuming the I / O part considered ideal (this is the controllaw-design point of view, i.e. step 2 in the method).

The *VEL_SP* block produces velocity setpoints for the LoopController, and can be seen as a sequence controller, but also as input signals to test the rest of this model.

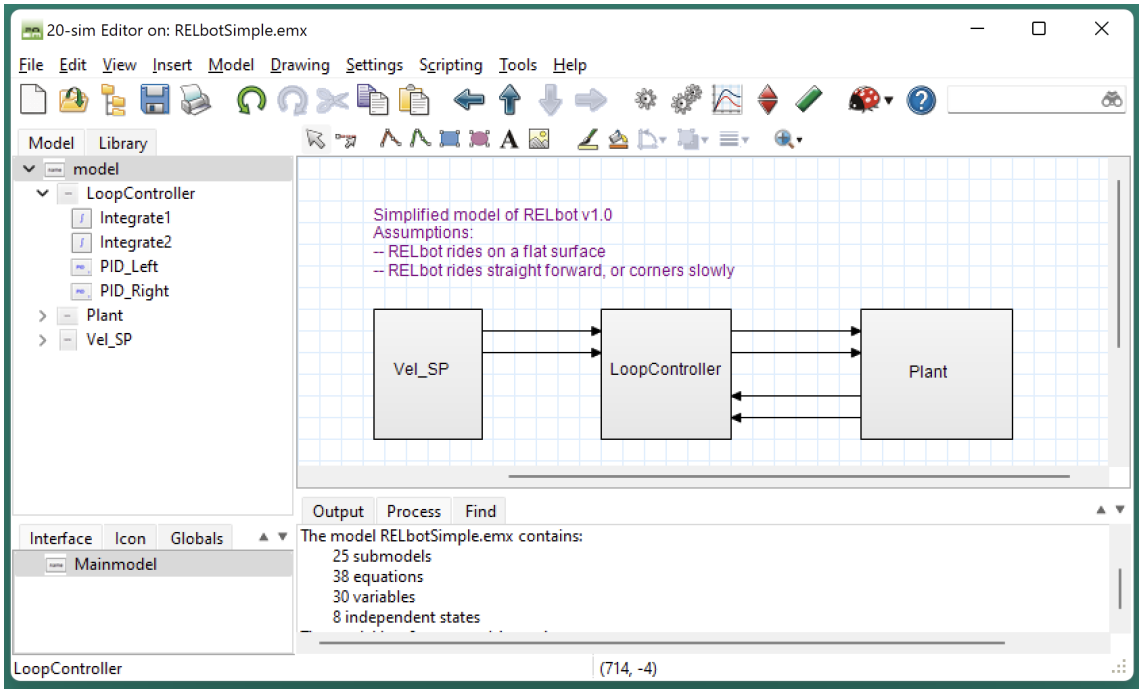

Figure D.1: Top-level model of RELbot, shown in 20-sim's Model Editor.

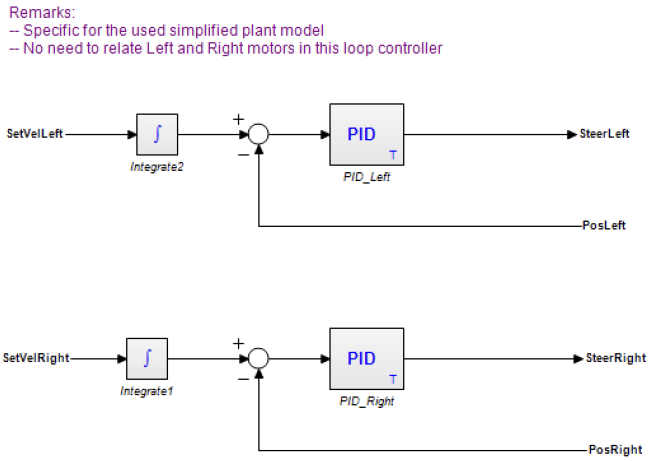

Figure D.2: The contents of the ControllerTilt block from Figure D.1.

E Course-specific framing for CBL projects

As stated in the CBL manual, the course-specific framing is:

- SDfR
  Describe the CPS architecture of the robotic system.
- ASDfR
  Describe the CPS architecture of the robotic system. Include also the real-time aspects.

Further details are presented here. First the common things, then specifics for ASDfR, and also in case your group has both SDfR and ASDfR students on board.

E.1 Describe the CPS architecture of your robotic system – both SDfR and ASDfR

It is on presenting the architecture of the software, interfacing to hardware, and hardware of your robotic system / embedded control system / cyber-physical system.

From the point of view of developing software for robots, the terms embedded control system and cyber-physical system are often used interchangeably.

Provide at least the following

- Block diagram of the 'top level' of the system (high-level block diagram), comparable to ROS2 nodes, and use a similar detail for non-ROS2 parts.
- Next to this block diagram, specify of each software part, next to its name in the diagram:
  - Description of its function.
  - Where it runs, that is on which processor (type), if applicable.
  - Sample time (if applicable).
  - Interfaces do not need to be specified but may be mentioned, as details of the connected communication channels also indicate what is produced as output, expected / received as input.
- Specify for each communication 'channel', next to its name in the diagram:
  - Type of channel (ROS, data, network package, electrical signal).
  - Data type, size of data.
  - Physical quantity and unit, in case physical signals are communicated.
  - Sample time (if applicable).
- Specify for the non-software parts:
  - Description of its function.
  - What domain it is in (electrical, mechanical...).
  - If available and useful, more domain-specific details.
  - Interfaces do not need to be specified, unless the communication channels are not described separately (for instance user interface, interface to environment). Then describe that interface.

E.2 Specifics for ASDfR

In addition to the information stated for SDfR, add:

- Augment the block diagram to indicate timing regime: Non / Soft / Firm-Hard real-time / Physics / etc. You may indicate those timing regimes by drawing (dotted) lines to separate those areas.
- What timing support each software block needs to have: SRT, F/HRT, or NonRT.
- What QoS and timing behaviour each communication channel needs to have. You may use ROS2 terminology, if applicable and useful.
- How connections between SRT and F/HRT parts are realised.

E.3 Combined SDfR and ASDfR

In case your CBL group has both SDfR and ASDfR students, you may specify both parts as a combination.

E.4 Hints on producing diagrams

Diagrams drawn on paper are accepted as long as they are readable and complete. A few suggestions for making diagrams on a computer:

- Draw.io (has both an online version on `draw.io` and an offline version (Help|Get Desktop in the online version)).
- UMLet (`https://www.umlet.com/`), specific for UML diagrams.
- Your own favorite application.

Both suggested tools have integration into VS Code; check the VS Code Extensions list.

Bibliography

Broenink J, van Oort G, Lenders L, 2025 "RELbot software tools and RELbot simulation (v2024.n)"

Controllab Products, October 2008 "20-sim" URL http://www.20-sim.com/

Raoudi I, 2024 "ROS2 – Xenomai4 Real-time Framework on Raspberry Pi" MSc Thesis 078RaM2024, University of Twente URL https://cloud.ram.eemcs.utwente.nl/index.php/s/g6SijsJ4P8igszo

Student Charter, 2020 "Student Charter at University of Twente" URL https://www.utwente.nl/en/ces/sacc/regulations/charter