Advanced Software Development for Robotics Assignment set 3 — Integration, Controlling a robot

Group10: Tongli Zhu(s3159396), Thijs Hof (s3099571)

Statement regarding use of AI tools

1. Code: We had a problem with executables, we took the idea from ChatGpt and used "chmod +x".
2. Report: Question 3.4.2- the third method was suggested by ChatGpt 4.0 after we did not find the answer in the literature, it seems like a valid method to us.

Assignment 3.1: RELbot CPS architecture(the high-level block diagram of the structure)

The structure of the Cyber-Physical System (CPS) is shown as Figure 1, which is divided into four main modules: the personal laptop, the Raspberry Pi, the interface part and the mechanical structure of the RELbot.

3.1.1 Hardware Connections and Interfaces

1. **Personal Laptop to Raspberry Pi:** Connected via a local area network.
2. **Raspberry Pi to Interface Circuit Board:** Connected through the SPI protocol.
3. **Interface Circuit Board to Mechanical Parts:** Connected through the PMOD connectors.
4. **Camera:** Directly connected to the Raspberry Pi via USB for real-time video capture.

3.1.2 Software Deployment and Task Allocation

The core software is deployed on the Raspberry Pi, including two parts: ROS and Xenomai, each running on different processor cores to ensure system real-time performance.

3.1.2.1 ROS Part

Responsible for image processing tasks. It uses the `cam2image` node to capture video frames from the camera and sends the image data to the `lp_Indicator_node`. This node processes the image, identifies the position of the brightest point, and publishes this location information via the `light_position_data_topic`. Subsequently, the `wheel_control_node` calculates the set speed values for the left and right wheels (`left_setpoint_vel`, `right_setpoint_vel`) based on the received position information.

But compared with Assignment Set 1, we made some special modifications to Assignment Set 3.

1. **Change in Position Determination Mechanism:** Previously, the system used the simulator's position data to judge the relative position to the brightest point. Now, it directly uses the center point of the image to determine the robot's position.
2. **Message Type Conversion:** Attention should be paid to message type conversion when matching topics in RosXenoBridge to ensure data format consistency.

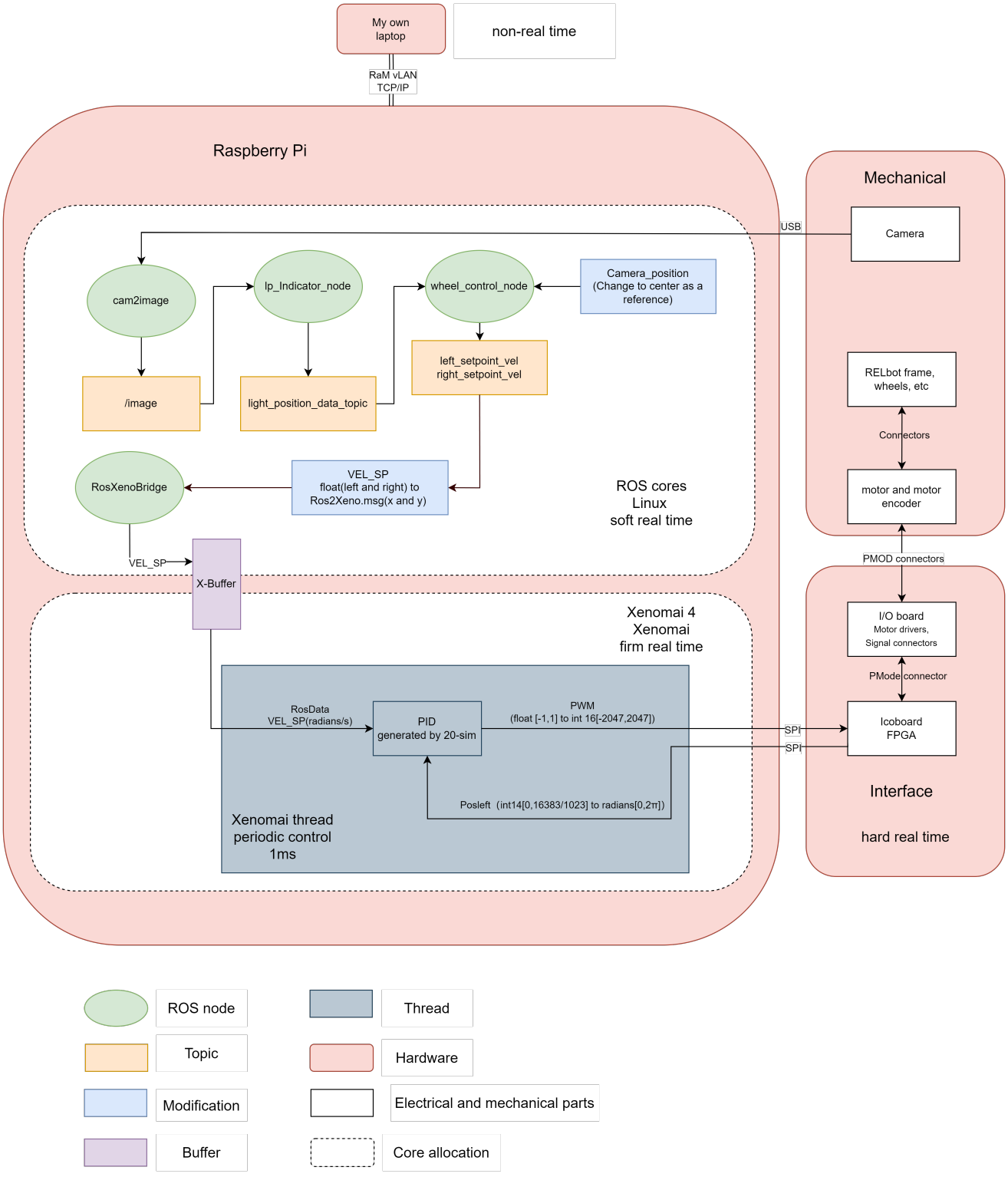

Figure 1: RELbot CPS architecture(the high-level block diagram of the structure)

3.1.2.2 Xenomai Part

The calculation of control commands is deployed in a Xenomai thread, set to a cycle of 1ms. Using the PID controller generated by 20-sim software, a continuous loop control of two inputs (VEL_SP, Posleft) and one output (PWM) is performed. The data conversions involved are as follows:

- **Speed Unit Conversion:** Convert VEL_SP from radians per second to degrees to fit the PID controller.
- **Position Unit Conversion:** Convert Posleft and Posright from an integer range [0, 16383] to radians.
- **PWM Output Mapping:** Map the output of the PID controller from a duty cycle to an integer range compatible with the motor control requirements.

3.1.3 Real-time Interaction

After processing by the Raspberry Pi, information interacts with the interface and mechanical parts in real-time to realize the final functionality.

Assignment 3.2: ROS 2 on the Raspberry Pi of the RELbot

3.2.1 Show that the system works

Design choices

We follow this part as the Figure2. Since we are CBL students, we use the Sequence-Controller issued by TA. At the same time, as the requirements of the experiment changed (from the brightest point to the green Cube), we made the following adjustments to the original program:

1. We modified the message type and name published by the original image processing node to light_position. The purpose of this is to unify it with the Sequence-Controller subscription.

2. We conducted the pre-test on the VM. The purpose of doing this is safing time because the time in the laboratory is limited and it needs to be efficient, but it should be noted that the Raspberry Pi has to use another architecture of Sequence-Controller.

3. Since the requirements of the experiment have changed now, we have modified the image processing part. In the previous version we applied a threshold to the gray-scale image. The updated version applies a threshold to the red, green, and blue channels separately to find which parts of the image match the color of the cube. The thresholds that are used were obtained by sampling the colors from two faces of the cube, one of which was exposed to direct light, the other was in a shadow. In this way the cube will be detected more reliably when the shadows change as it moves around.

Furthermore a dilation and erosion step were added. By performing erosion we get rid of other smaller objects in the image that have a color similar to the cube. But if holes occur in the detected cube it might be eroded away. To prevent that a dilation step is applied before performing the erosion to ensure that these holes are closed first.

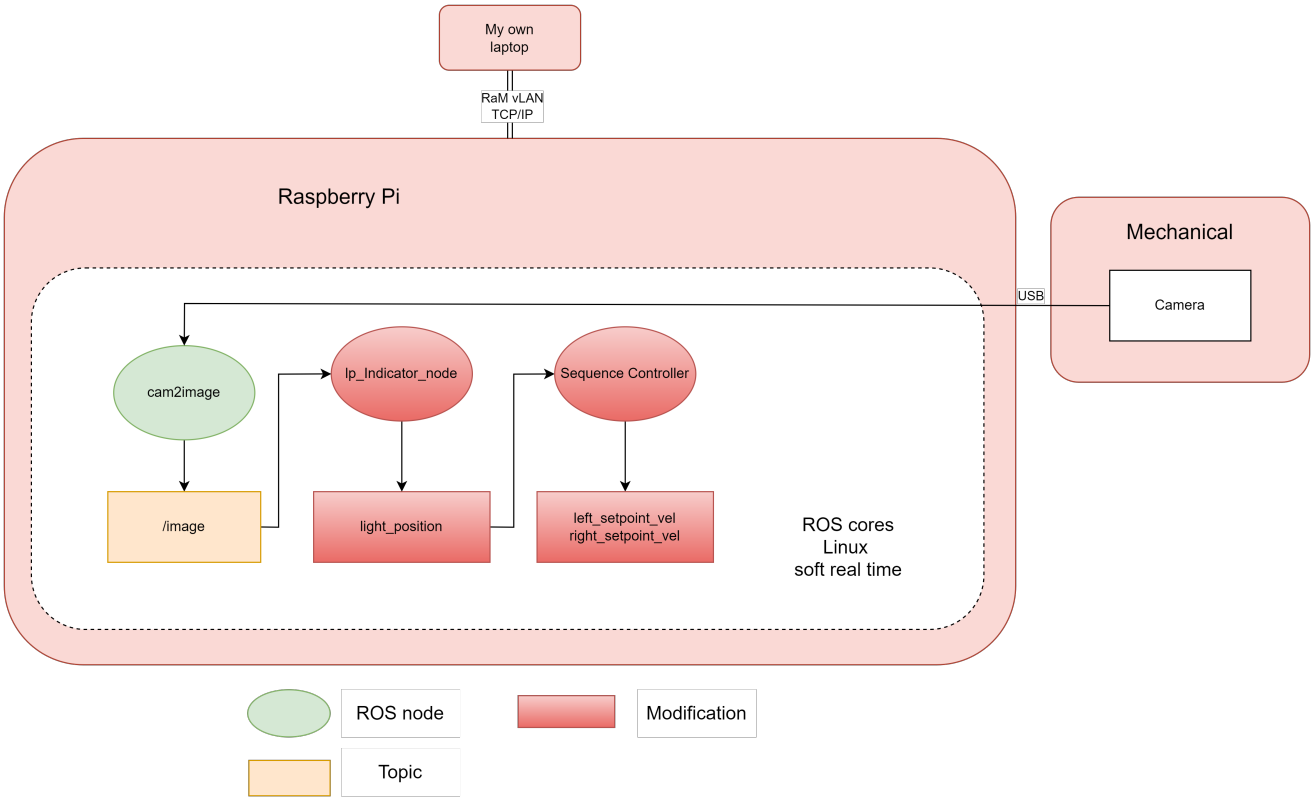

Figure 2: Architecture diagram for ROS module

Record and Store the results of your tests

The rqt diagram is shown as Figure 3, which proves our nodes work well on the laptop.

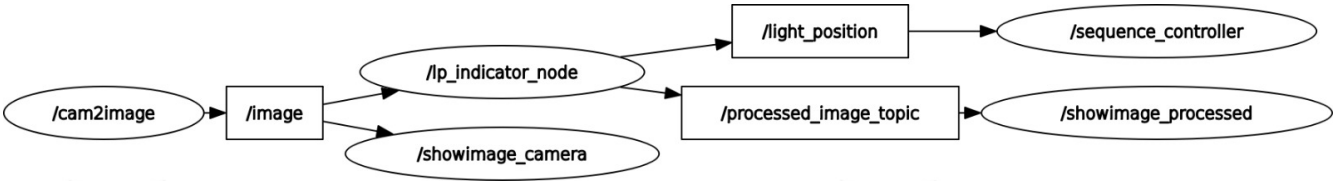

Figure 3: RQT diagram

The test results after adjusting the appropriate parameters can be seen in Figure 4, it shows very stable performance.

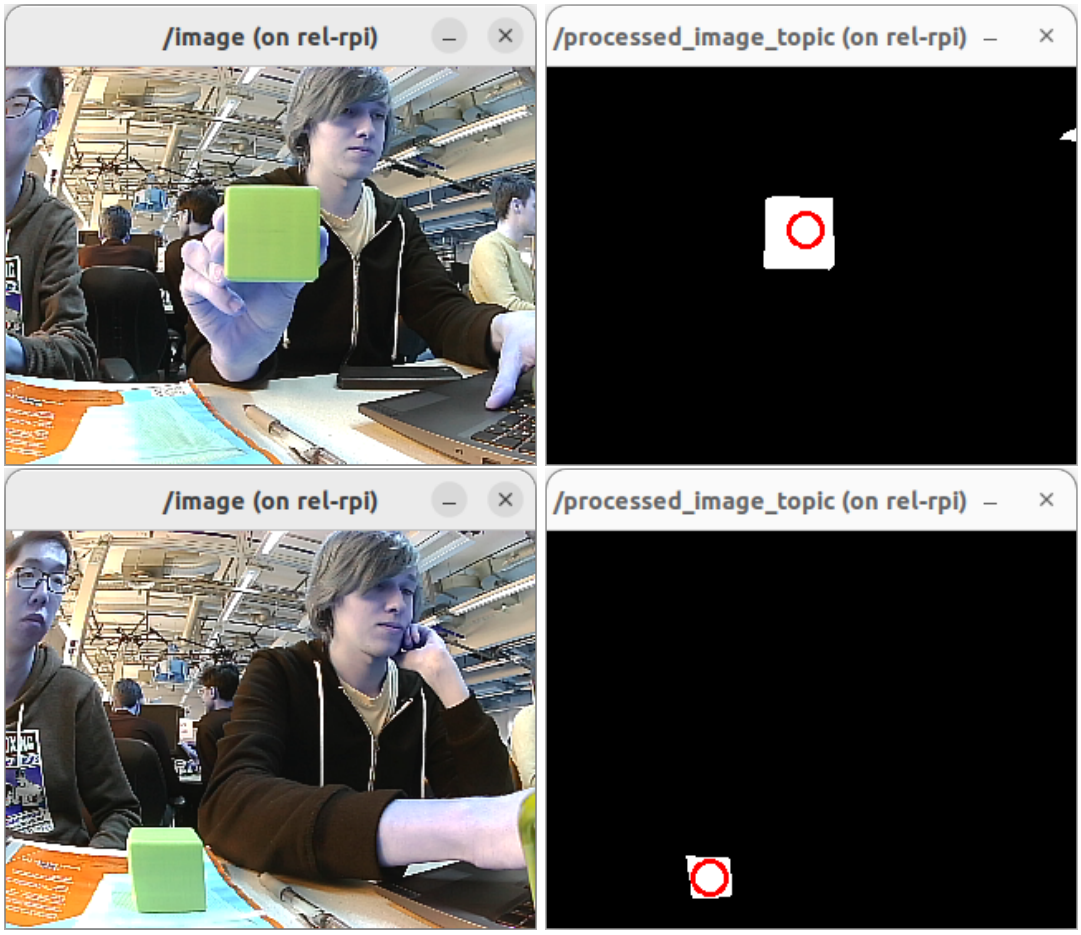

Figure 4: The results for ROS2 part

3.2.2 Questions

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

After just connecting to the sequence controller at the beginning, the general functionality is no problem. But with the changes in image processing functions, we encountered some bugs, but we fixed them in the end.

The processor loads are shown as Figure 6 and Figure 5. The main reason is the hardware performance difference, our own laptop is equipped with a more powerful CPU (with 8 CPU cores, 16 threads) and more RAM (32.0 GB), which is much higher than the Raspberry Pi configuration, so running the same workload on the Raspberry Pi will naturally result in higher resource usage.

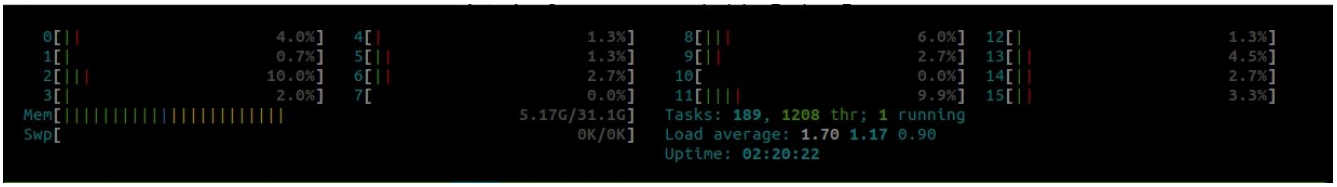

Figure 5: The processor load for own laptop

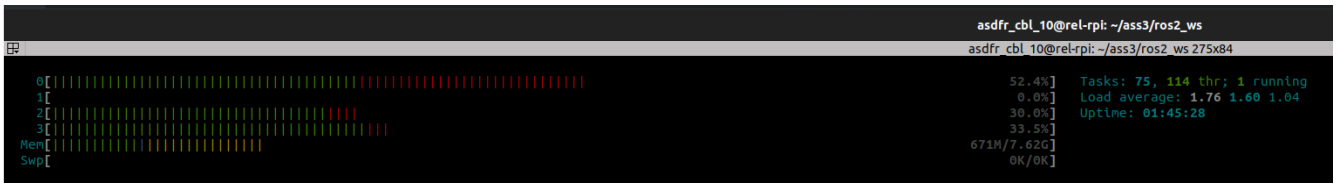

Figure 6: The processor load for Raspberry Pi

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?

Yes, we have made the following changes to migrate the code to Raspberry Pi:

1. We no longer need to use the image_tools_sdfr tool, because Raspberry Pi is a Linux system and can use image_tools directly.

2. We need to change sequence_controller_x86_64for Intel x86 64-bit processors as in Windows laptops to sequence_controller_aarch64 for Raspberry Pi.

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?

We indeed made changes in the nodes, but not because of 1.2.3 (we used sequence control directly). The main work focused on the image processing part and the adaptation part with sequence control, which could be seen in the previous section "3.1 Show that the system works".

Assignment 3.3: FRT software architecture on RELbot's Raspberry Pi

3.3.1 Show that the system works

Design Choices

As illustrated in Figure 7, this is the architecture diagram for our test bed. Firstly, since we must use the existing sequence controller and ROS-Xenomai-Bridge, but discovered that the message types published and subscribed between these two nodes do not match, we specifically wrote a converter node to convert floating point velocity values (left and right wheels) into Ros2Xeno.msg (x and y).

Secondly, we replaced the previous image processing node with a new test node called velocity_node, which sends some fixed velocity values. The reason for this is that the ROS code before the sequence controller has been verified to be valid in part 3.2. Now, this test node effectively takes over this part. Compared to the previous start-up of many nodes, the new node significantly simplifies the debugging process and prevents accidental modifications to previously verified nodes during debugging.

Thirdly, in order to create an "empty" loop controller, we chose not to modify the 20-sim software to avoid introducing new errors. Instead, we adopted a "bypass" method, that is, letting the set_velocity values pass through the bridge and then directly assigning these values to PWM, which can be seen as "short-circuiting" the PID part. This design is intended to facilitate the analysis of the relationship between velocity settings and wheel rotation, the positive value for the left wheel PWM indicates forward rotation, and the negative value for the right wheel PWM indicates forward rotation.

In addition, the ultimate purpose of building this test bed is to facilitate debugging and give confidence to our project. Consider our project divided into three main parts: 1) image processing, 2) information transmission (including PID control), 3) driving the wheels. If the experiment on the test bed is successful, our subsequent work will focus on testing PID control. Of course there is also the integrated image processing part, but this will involve relatively little debugging work since we have already conducted relevant tests in Section 3.1.

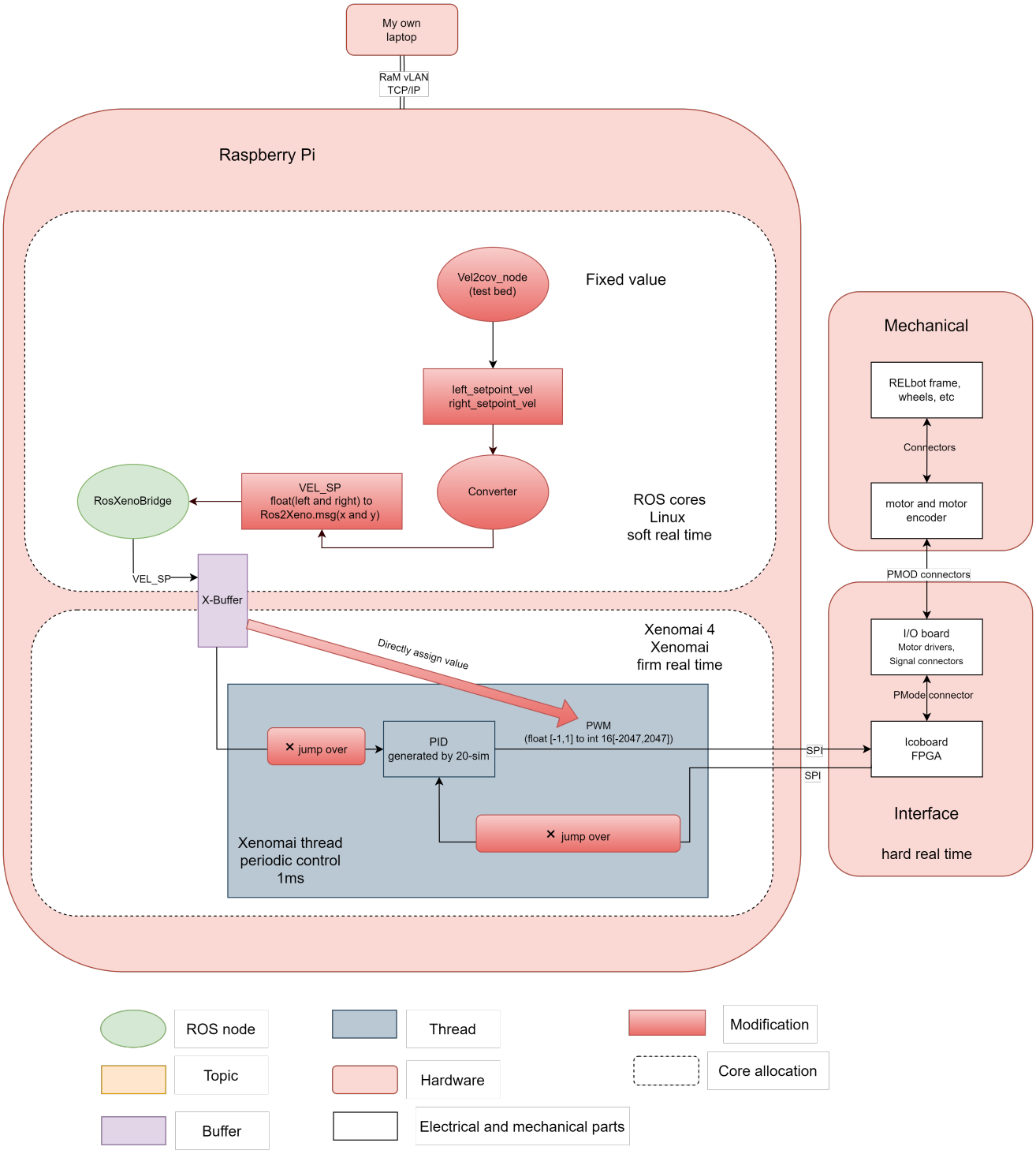

Figure 7: Architecture diagram for the test bed

Record and Store the results of your tests

We conducted a series of experiments to test the PWM sign and wheel rotation direction by assigning the constant value to PWM. The results of test can be seen in the Figure 8. The movement of a single wheel can be seen in the video.

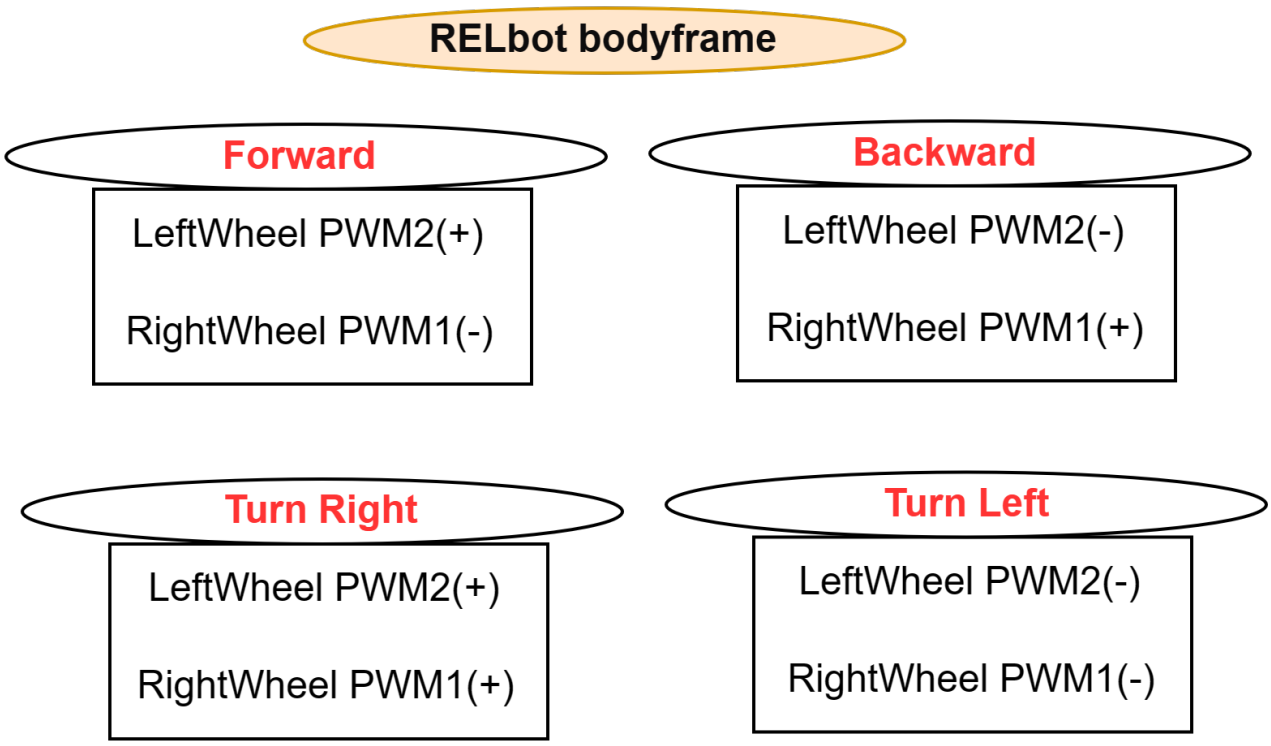

Figure 8: The results of tests(Correspondence between pwm sign and wheel rotation direction)

Questions

1. What are advantages and disadvantages of separately testing the FRT skeleton part first, as done in this assignment?

Advantage

1. **Efficiency in Modularization:** Modular adjustments significantly enhance the overall efficiency. Without separate testing, when problems occur, it would be difficult to determine whether they are in the ROS part or the FRT part. This could inadvertently affect the previously tested ROS components, leading to inefficient debugging efforts.

2. **Understanding Control Logic:** Currently, our understanding of the principles of PID control is still preliminary. This testing step allows us to grasp the specific relationship between the set_point_vel values and the wheel movements. Through intuitive experiments, we lay a solid foundation for subsequent debugging if there are any abnormalities in wheel direction or speed.

3. **Significance of "milestones":** Our ultimate goal is to make the robot move to the green cube. The FRT part is directly related to mechanical control. Testing this part not only achieves motor activation but also represents a milestone for our project, greatly boosting our confidence in testing.

Disadvantage

1. **Attention to Module Integration:** Special attention must be paid to the interaction and integration between modules. The part from ROS to the sequence controller is "created" by us now. We need to ensure that the conditions of the new test node output are exactly the same as in previous tests, which take more time and resources.

2. What is the advantage of starting with an 'empty' loop controller in 20-sim and use that as a starter for Assignment 3.4?

Modular Isolation

Through the task of modular decomposition, especially the isolation and testing of the loop controller, we could more clearly identify the source of issues. If the loop controller were tested directly within the full system setup, it would be challenging to discern whether the problem was with the loop controller or the Bridge. By isolating it, once debugging is successful, the focus will then shift solely to adjusting the loop controller, greatly simplifying the problem-solving process.

Understanding the Impact of PID Control

Through this test bed, we have gained an understanding of how the output of the PID controller affects wheel movement. This comprehension is intuitively important for the task 3.4.

In the previous phase 3.2, we understood what information was published from the ROS sequence controller. In phase 3.3, by setting the PID outputs directly, we could determine which output values were reasonable and could effectively drive the wheels to turn. Additionally, by printing the information returned by the encoder, we obtained insights into the returned position values (Pos) and the relation between the sign of inputs and the wheel's direction, as the Figure 10. In the upcoming phase 3.4, should there be a failure in control, we can print the input and output values of the PID to determine which part of the process is problematic.

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

Assignment 3.4: Loop-Controller algorithm in FRT part

3.4.1 Show that the system works

Design choices

In this section, we focus on how to get the loop controller to work properly. Initially, we need to clarify the inputs and outputs of the loop controller. Our task is to build a bridge, which is the preProc() and postProc(), to filter and transform the information from the existing code, ensuring it is delivered correctly.

The conversion relationships are defined in the following table:

Type	Name	Source	Range	Unit	Target	Range	Unit
Output	SteerLeft	$y[0]$	-	int	FpgaOutput.pwm1	[-2047, 2047]	int
Output	SteerRight	$y[1]$	-	int	FpgaOutput.pwm2	[-2047, 2047]	int
Input	VEL_SPLeft	RosData.x	-	rad/s	$u[2]$	-	rad/s
Input	VEL_SPRight	RosData.y	-	rad/s	$u[3]$	-	rad/s
Input	PosLeft	FpgaInput.channel1	[0, 16383]	1/16383 rev.	$u[0]$	-	rad
Input	PosRight	FpgaInput.channel2	[0, 16383]	1/16383 rev.	$u[1]$	-	rad

Table 1: Conversion relationships

1. SteerLeft and SteerRight (PWM) values outside this range are clipped to -2047 or $+2047$.
2. The left wheel corresponds to RosData.x and pwm/encoder channel2, the right wheel to RosData.y and pwm/encoder channel1.

For $u[2]$ and $u[3]$, $y[0]$ and $y[1]$, they can all be processed through simple assignment or sign change. The biggest challenge of this project is in $u[0]$ and $u[1]$ calculation. Figure 9 shows how the encoder values are converted to an angle. It is mainly divided into three steps: One is Determine encoding wrapping, the other is Transform, and the third is Integral.

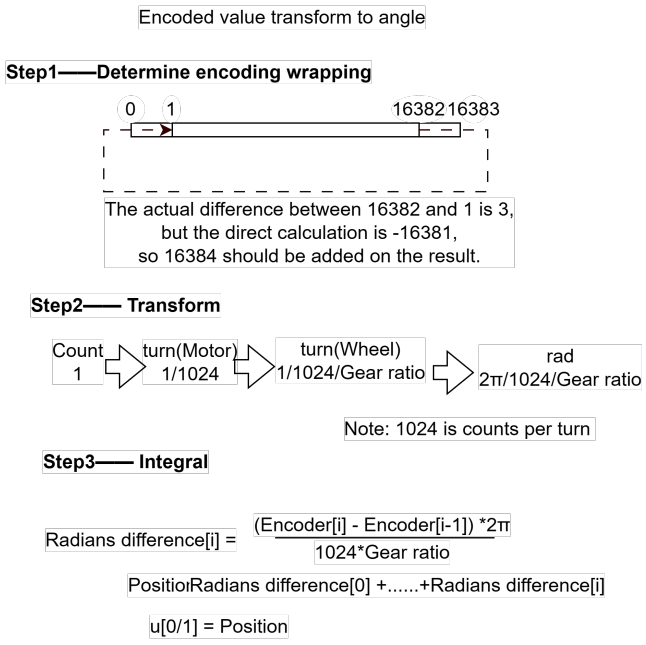

Figure 9: Encoded value transform to angle

After designing the `preProc()` and `postProc()` functions, we will need to modify the `Velocity_node` (test bed) to change it into a node that periodically sends `VEL_SPLeft`, to simulate the functionality of going straight and then turning. For example, sending a velocity every second, sending a forward velocity for the first three seconds, and sending a zero velocity for one wheel in the fourth second to simulate the turning process. The sign of the velocity values depends on the conclusions tested in section 3.3. Taking "forward" as an example, we detail the corresponding relationship between the positive and negative signs of the parameters and the wheel direction as shown in Figure 10.

Finally, the skeleton diagram of the entire part is shown in Figure 11.

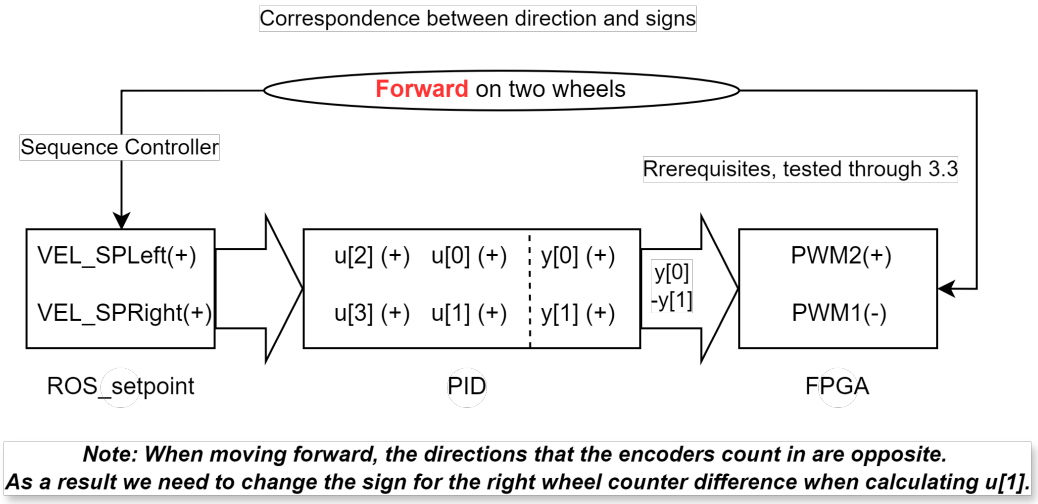

Figure 10: Correspondence between direction and signs

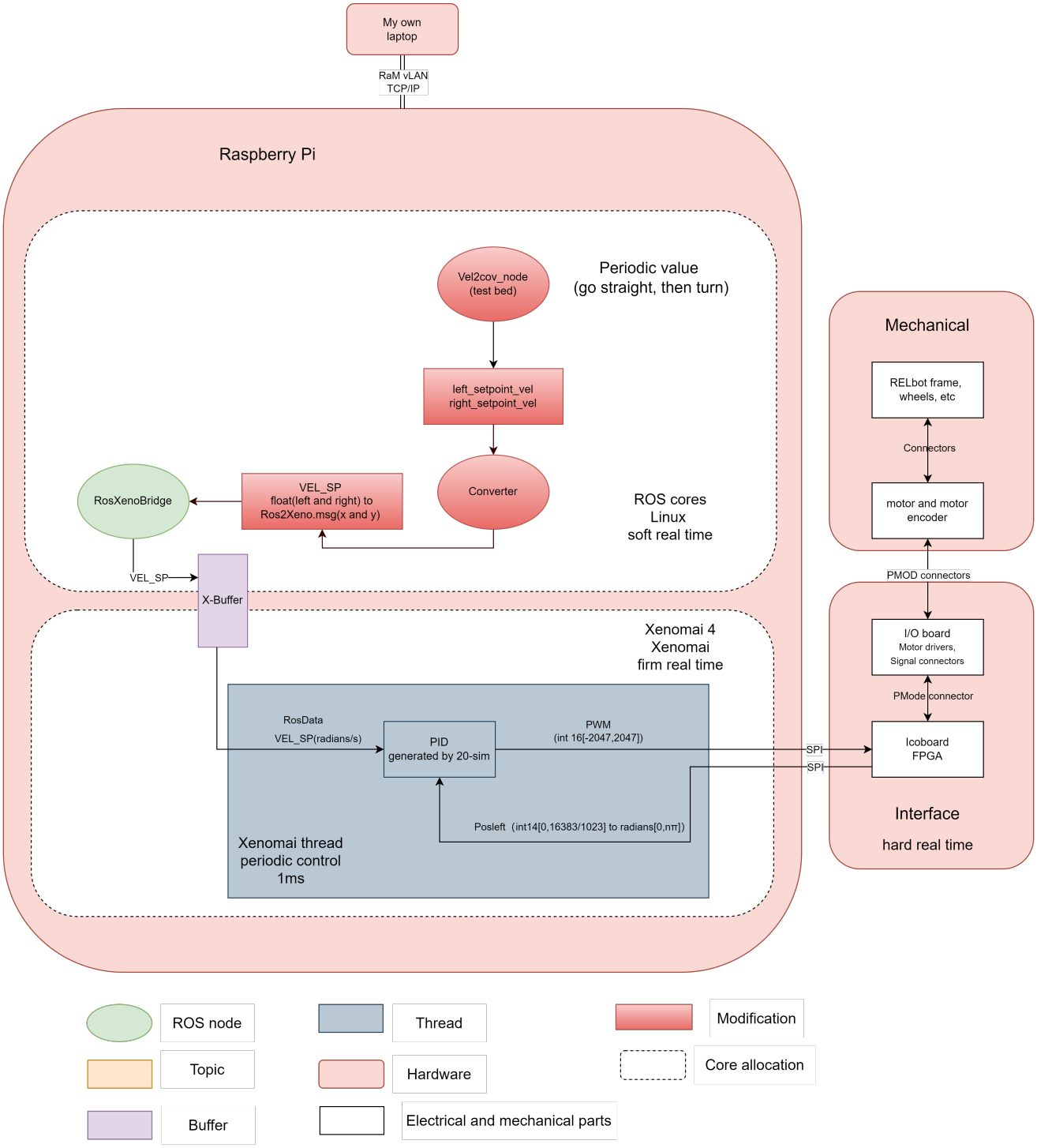

Figure 11: Architecture diagram for the Loop Controller

Record and Store the results of your tests

We conducted a periodic setpoint with a cycle of 4 seconds. During each cycle, the forward speed is set for the first 2 seconds, and for the latter 2 seconds, the right wheel is set to a negative value to perform a right turn. The speed setpoint was adjusted multiple times due to limitations of the experiment, which was carried out on a table. Therefore, a lower speed was chosen to ensure the safety of the robot, preventing it from falling off the table due to excessive speed. Finally, a speed of 1.57 (half of π) rad/s was selected, achieving an approximate 90-degree turn. For the detailed process, see the video.

Other discussion

We have some uncertainties regarding the parameter "1024 counts per turn," specifically, it's essential to clarify how the encoder counts. Common counting methods include:

- Single-phase counting: The encoder counts at one edge of a signal (for example, the rising edge), thus each revolution results in 1024 counts.
- Dual-phase counting: If the encoder counts both the rising and falling edges of a single signal, then the actual count per revolution is double, i.e., 2048.
- Quadrature counting: If the encoder monitors two signal lines (typically A and B), and counts at both the rising and falling edges of each signal line, the actual count per revolution quadruples, i.e., 4096.

Considering our project, we have not received specific answers in the Canvas discussion window; therefore, we assumed that "1024 counts per turn" had already accounted for the above counting rules. Later we found that the calculated angles had to be divided by 4 to get the correct values, this might mean that quadrature counting is implemented.

3.4.2 Questions

1. What are the advantages and disadvantages of doing code generation (as opposed to writing the controllers directly in C/C++)?

Advantages:

- **Rapid Development:** Using code generation tools can quickly produce executable code from visual models, significantly shortening the development cycle.
- **Reduced Errors:** Code generation can decrease the likelihood of human coding errors, as the code is automatically generated from verified models.
- **Intuitive simulation:** Although the simulation results are different from running on the hardware, compared to recompiling and running to the Raspberry Pi every time, 20-sim allows us to make a preliminary judgment on the operating function of the PID as Figure12, and then adjust the parameters.

Disadvantages:

- **Limited Flexibility:** Generated code may be difficult to customize, as exemplified by the hardcoded controller named `LoopController()` in our project.
- **Timeliness and Dependency Issues:** Generated code may be complex in structure, challenging to manually adjust and optimize, especially when integration with larger systems is required. Moreover, reliance on specific tools for updates, discontinuation, or licensing changes can impact the ongoing updates and maintenance of the project.

2. Implementing theM&A block by embedding it in the controller loop (preProc() and post-Proc()), 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?

Using preProc() and postProc(): This method has the high modularity and is easy to maintain.

Integrated with Loop Controller(Directly modify the Loop Controller code): This method is easy to implement, but as an integrated module, it's not friendly for debugging or maintenance.

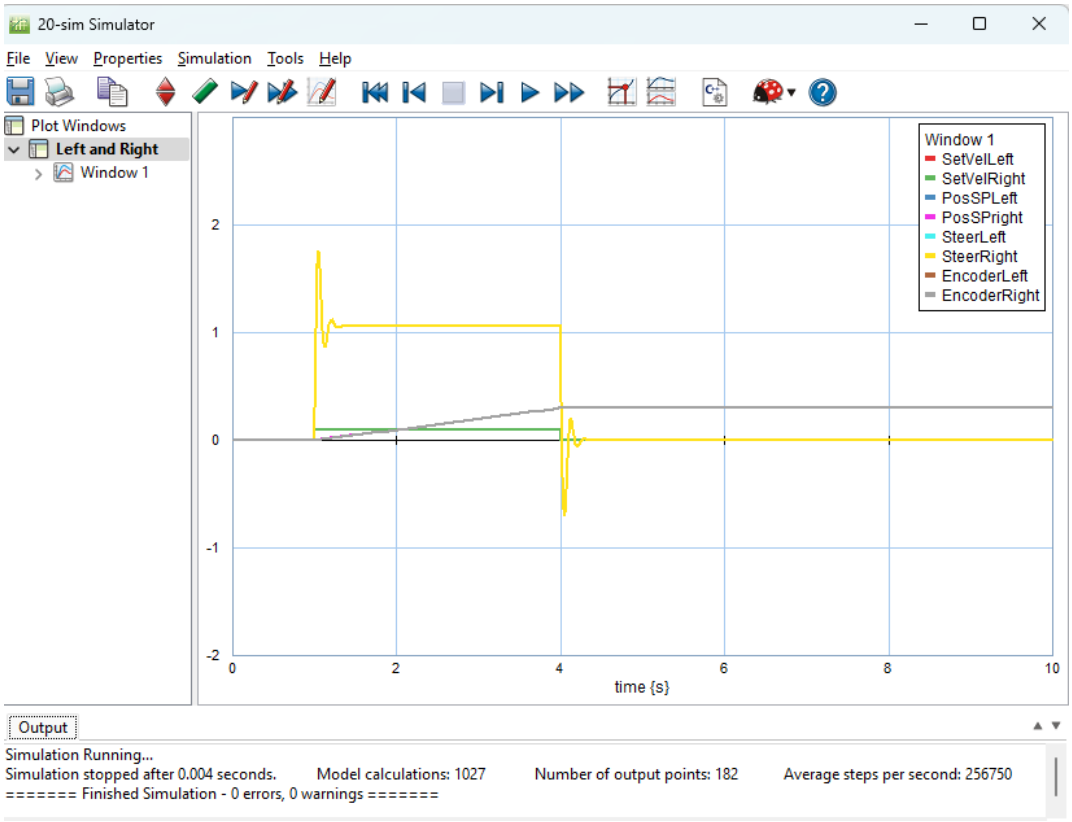

Figure 12: 20-sim simulation result

Separated Thread: Design the M&A function as a separate thread or process that communicates with the main control loop via a message queue. This offers strong modularity and high maintainability, but the implementation complexity is high.

In PID control systems, it is often necessary to ensure that the control algorithm is closely synchronized with the sampling period to ensure the stability and response speed of the control effect. The PID algorithm itself is usually not computationally intensive and can be completed in a short time, so executing it within the main loop usually does not have a negative impact on performance. Considering maintainability and debugging convenience, we think using preProc() and postProc() is the best approach.

Assignment 3.5: Final integration

Show that the final system works

Design choices

Because 3.1 has tested the function from the "start" to the "sequence controller", and 3.4 has tested the function from the "sequence controller" to the "end", and the output of the sequence controller and the test node is consistent, it is natural to compile together at this time.

Record and Store the results of your tests

During the experiment, an oversight occurred: we initially tested the virtual machine on our laptop and inadvertently failed to update the sequence_controller on the Raspberry Pi. Additionally, due to time constraints, we were unable to perform a final test or record a video of the process. However, we are confident that our code functions correctly, and we invite the TA to verify its operation.

Questions

Compare the resulting system block diagram with the block diagram from Assignment 3.1. Indicate the differences, and describe why the final implementation deviated from the original diagram.

Upon reviewing all the steps, compared to version 3.1, we have made the following modifications:

- **lp_indicator_node:** We have updated the message we publish from the brightest point being the position of a green cube, which involves changes in image processing. Additionally, the type of message to be published also needed to be changed to match the requirements of the Sequence Controller.
- **Sequence Controller:** This is a requirement given by the course, and we must use it.
- **Converter:** Since the Sequence Controller did not provide source files (src files), we had to create our own conversion node, in order to transform the `set_velocity` command into a type that can be subscribed to by RosXenoBridge.
- **PWM conversion:** Initially, we assumed that the PWM values needed to be scaled from a range of -1 to 1 up to -2047 to 2047. However, after discussions with the teaching assistant and other groups, we learned that the PID controller outputs, $y[0]$ and $y[1]$, were already scaled to the [-2047, 2047] range. Consequently, we did not implement scaling in our final experiment, which yielded satisfactory results. Nonetheless, we recognize this as an area for future improvement. To ensure accuracy, we should independently verify the output values $y[0]$ and $y[1]$ by printing them, rather than relying solely on external advice.

The final architecture diagram can be shown as Figure 13.

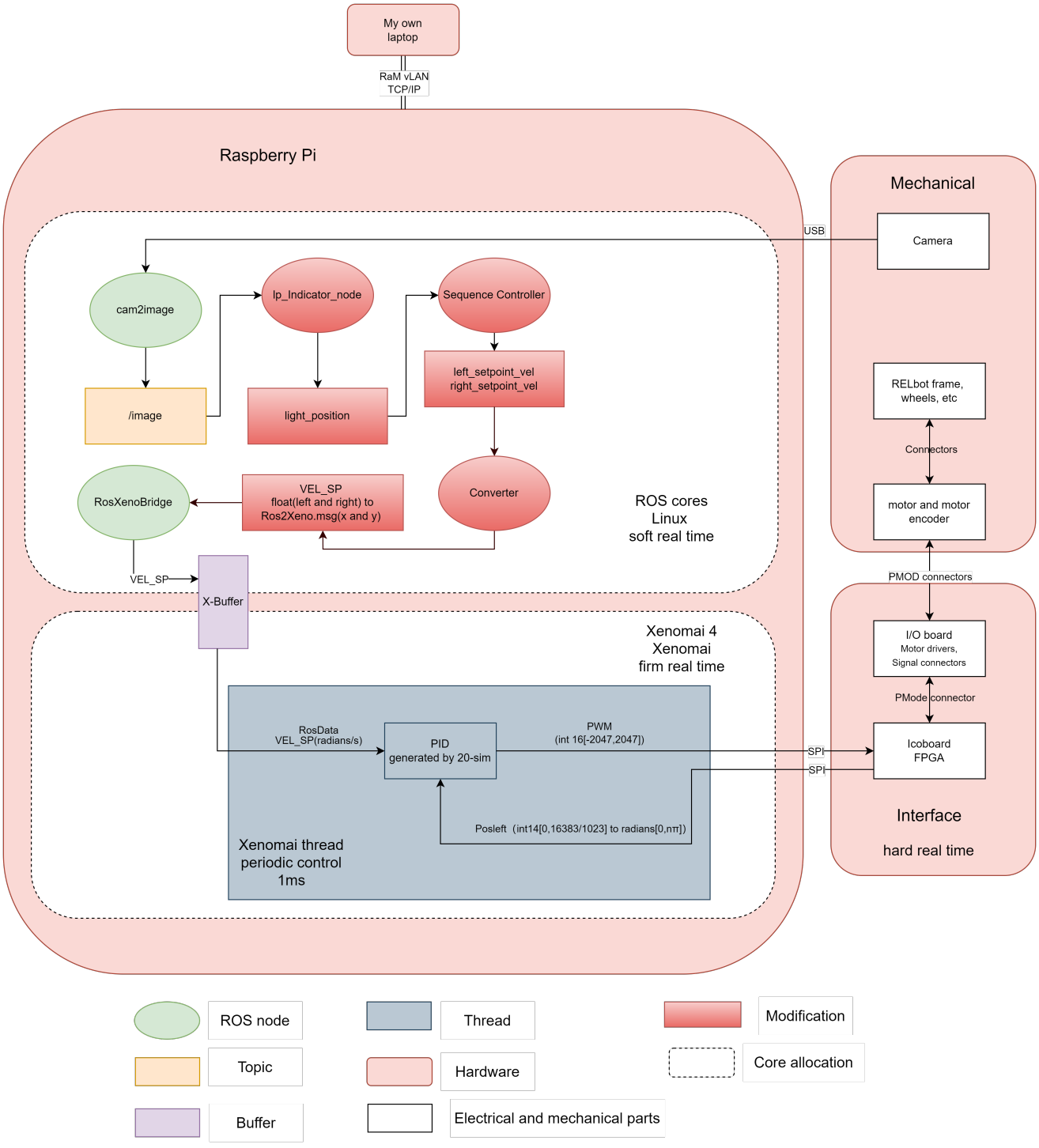

Figure 13: Final architecture diagram

A Appendix - Video

Related videos can be accessed via the following link: Assignment3-Group10-Video