

Advanced Software Development for Robotics Assignment Manual (v2022.1)

Gijs van Oort and Jan Broenink

Wednesday 8th February, 2023

Contents

0	Introduction	1
0.1	Context	1
0.2	Organisation	1
0.3	Support from the Teaching Assistants	2
0.4	Responsibility and fraud	2
0.5	Document History and Disclaimer	3
1	Assignment set 1 — ROS2	4
1.0	Assignment 1.0: Getting familiar with ROS2	4
1.1	Assignment 1.1: Image processing with ROS2	4
1.2	Assignment 1.2: Sequence controller	5
2	Assignment set 2 — Timing in Robotic / Cyber-Physical Systems	8
3	Assignment set 3 — Controlling the real Jiwy	9
A	Changelog	10
B	Getting Linux and ROS2	11
B.1	ROS2 on virtual machine (Virtualbox)	11
B.2	Installing ROS2 directly on Linux	11
C	Ubuntu22.04-RAM Virtual Machine — User Manual	12
C.1	Introduction	12
C.2	VM Contents	12
C.3	VM User name/password	12
C.4	Installation	12
C.5	Tips & Tricks	13
D	Visual Studio Code	14
D.1	C/C++ Extension Pack	14
D.2	Configuration files	14
D.3	VSCoDe and ROS2	14
D.4	Using VSCoDe for remote editing	15
E	Jiwy simulator	17
E.1	Dependencies	17
E.2	Input topics	17
E.3	Output topics	18

E.4 Parameters	18
E.5 Differences with a real Jiwy	18
E.6 Getting and compiling the node	18
Bibliography	20

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, i.e. physical machines.

For the soft real-time part (Assignment set 1 and 3), we use ROS2 (Robotic Operating System 2), version Humble, heavily used in modern robotics. The algorithms are programmed in C++. For the development environment, you can either install ROS2 on your *own* laptop, or install a virtual machine running ROS2, see Appendix B.

For the hard real-time part (Assignment sets 2 and 3), we work with a Raspberry Pi 4, using a specifically tuned Linux (Raspberry Pi OS with the Xenomai real-time patch) and programs written in C++. The Raspberry Pi's are available at the RaM Lab. In Assignment set 2, only the bare Raspberry Pi is needed, without additional hardware. In Assignment set 3 a robotic system (a two-DOF webcam movement unit) is controlled with the 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`).
- 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 design, 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, which is indicated in the *framing* Section of the CBL manual.

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

0.3 Support from the Teaching Assistants

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

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

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.

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 authors: Arnold Hofstede, Muriel vd Spek, Timon Kruiper, Alejandro Lopez Tellez. Marcel Schwirtz supported on the hardware.

Special thanks to Arend Rensink for the subsection “What constitutes fraud?”.

1 Assignment set 1 — ROS2

In this assignment you get familiar with ROS2 and build some projects with it. For Assignment sets 2 and 3 you *must* use Linux. Therefore it is recommended to already start using Linux from the first assignment set (switching halfway is not a good idea). See Appendix B for how to get access to Linux 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/humble/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

Use ROS2 standard tools to capture webcam footage, publish to a channel and view the channel. You can use the following commands¹:

```
ros2 run image_tools cam2image --ros-args -p depth:=1 -p history:=
↳ keep_last
ros2 run image_tools showimage
```

Show that the system works.

Note: you might check the tips in Section 0.2 on finalizing 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?

¹If it does not work, you may try to add to the first line: `-p device_id:=n` where `n` is 0, 1, ... This will pick a different camera stream from the list of available streams (check with `ls /dev/video*`).

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 light position indicator

A very simple way of detecting the position of a bright light in an (otherwise no so bright) image is the following:

1. Gray-scale the image.
2. Apply a threshold on the brightness of each pixel.
If the threshold is well-chosen (and there are no other bright parts in the image), the result is a black image with a large white dot where you shine the light. 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 center of the light.

Create a node that, given a camera input, outputs the position of a 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.

1.2 Assignment 1.2: Sequence controller

The final assignment set of this course is on controlling a pan-tilt camera unit, called *Jiwy* (Figure 1.1). 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 Jiwy.

1.2.1 Unit test of the Jiwy Simulator

A ROS2 node simulating the Jiwy 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). For details about the node, refer to Appendix E.

Create a node that generates a sequence of setpoints to test the Jiwy 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. Don’t forget to look at the time constant of the first order system.

1.2.2 Integration of image processing and Jiwy Simulator

This sub-assignment can be seen as an intermediate step towards the ultimate goal of having the Jiwy Simulator use its own (moving) camera output. In this sub-assignment the data from

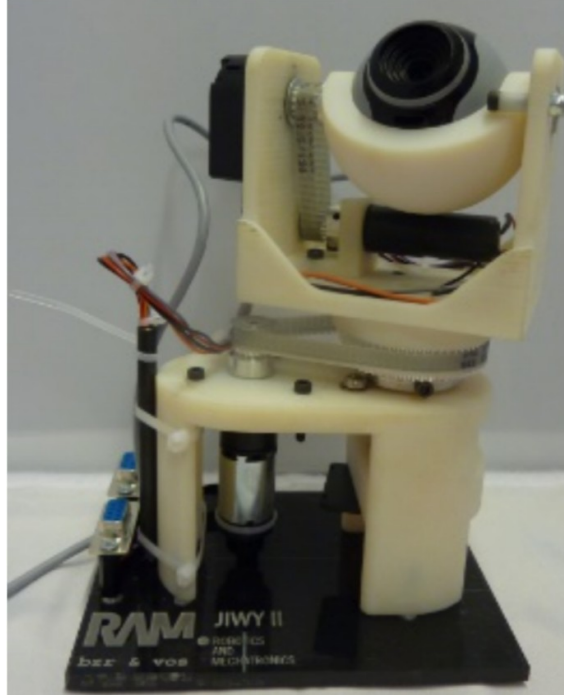


Figure 1.1: The pan-tilt camera unit *Jiwy*.

the (stationary) webcam is used directly (software development goes small, testable, steps at a time...).

Make a new ROS2 node that makes the Jiwy Simulator follow a bright light that you move around in front of your webcam. 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 nodes easily re-usable; thus not too specialized).

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 Jiwy 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 Jiwy look at the bright light again, but this time the input of the light position indicator should be the `/moving_camera_output` of the Jiwy simulator (just as in the real case, where the camera mounted on Jiwy moves around during operation).

In order to make this work, you need to steer the camera orientation, $\mathbf{x}_{\text{jwy}} = [x_{\text{jwy}} \ y_{\text{jwy}}]^T$ (in radians), towards the (absolute) position of the light $\mathbf{x}_{\text{light}}$ (in radians) such that the error $\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{jwy}}$ becomes zero. A convenient controller choice is a first order controller, where the rate of change of the Jiwy orientation, $\dot{\mathbf{x}}_{\text{jwy}}$, is proportional to the error, i.e.,

$$\dot{\mathbf{x}}_{\text{jwy}} = \begin{bmatrix} \dot{x}_{\text{jwy}} \\ \dot{y}_{\text{jwy}} \end{bmatrix} = \frac{1}{\tau} \cdot (\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{jwy}}) = \frac{1}{\tau} \cdot \begin{bmatrix} x_{\text{light}} - x_{\text{jwy}} \\ y_{\text{light}} - y_{\text{jwy}} \end{bmatrix}; \quad \mathbf{x}_{\text{jwy}} = \int \dot{\mathbf{x}}_{\text{jwy}} dt, \quad (1.1)$$

where τ is some time constant (a good start would be $\tau \approx 1$ s).

Notes:

- The position of the bright light in the camera image is precisely $\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{jiwy}}$.
- The only way to influence the camera orientation is by sending a setpoint \mathbf{x}_{set} to the simulator node. Fortunately, the Jiwy can track its setpoint well, so we can assume that $\mathbf{x}_{\text{jiwy}} = \mathbf{x}_{\text{set}}$, resulting in

$$\dot{\mathbf{x}}_{\text{set}} = \begin{bmatrix} \dot{x}_{\text{set}} \\ \dot{y}_{\text{set}} \end{bmatrix} = \frac{1}{\tau} \cdot (\mathbf{x}_{\text{light}} - \mathbf{x}_{\text{jiwy}}) = \frac{1}{\tau} \cdot \begin{bmatrix} x_{\text{light}} - x_{\text{jiwy}} \\ y_{\text{light}} - y_{\text{jiwy}} \end{bmatrix}; \quad \mathbf{x}_{\text{set}} = \int \dot{\mathbf{x}}_{\text{set}} dt. \quad (1.2)$$

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 Jiwy can track the bright light). 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

Keep the nodes you have programmed for this assignment set; you might need them again in assignment set 3.

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

In this assignment, you are going to analyze 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 communication between ROS2 nodes. By doing so, you can find out to what extent ROS2 is SRT-capable. In particular, what jitter appears in the communication between ROS2 nodes. 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 4 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 4.

This assignment set is to be handed out later.

3 Assignment set 3 — Controlling the real Jiwy

This assignment set is to be handed out later.

A Changelog

- **Version 2022.1, 8 February 2023**

Initial version, consisting of:

- Chapter 0 (Introduction)
- Chapter 1 (Assignment set 1)
- Appendix A (Changelog)
- Appendix B (Installing ROS2)
- Appendix C (Visual Studio Code)
- Appendix D (Ubuntu Virtual Machine)
- Appendix E (Jiwy Simulator)

B Getting Linux and ROS2

For the first assignments you need ROS2 on your own computer. For the later assignments you *must* use Linux. These requirements combine into two options for software installation:

- *Run ROS2 on a virtual machine (Virtualbox) running Ubuntu 22.04.*
This is the recommended way. We provide a virtual machine which has everything you need pre-installed.
- *Run Ubuntu natively on your laptop and install ROS2 yourself.*
If you happen to run Ubuntu on your laptop you can install ROS2 directly on it. Note that in the later assignments you need 20-sim as well. You can either use a separate Windows PC for that or install it on Linux by using Wine (installation instructions will follow in an update of this manual).

B.1 ROS2 on virtual machine (Virtualbox)

See Appendix C.

B.2 Installing ROS2 directly on Linux

Installation instructions are on the ROS2 site: <https://docs.ros.org/en/humble/Installation.html>. Installing binary packages is usually easier than compiling from source.

For Ubuntu users, there is a convenient script that does the full installation unattended; have a look at https://github.com/Tiryoh/ros2_setup_scripts_ubuntu. We used this script to install ROS2 on the virtual machine provided, as follows:

```
wget https://raw.githubusercontent.com/Tiryoh/ros2_setup_scripts_ubuntu
↪ /main/ros2-humble-desktop-main.sh
chmod 755 ros2-humble-desktop-main.sh
./ros2-humble-desktop-main.sh
```

C Ubuntu22.04-RAM Virtual Machine — User Manual

C.1 Introduction

This document/chapter¹ gives some information on how to work with the *Ubuntu22.04-RAM* Virtual Machine. It was prepared for usage with the courses *Advanced Software Development For Robotics* (ASDFR), *Programming 2* (optional) and *Software Development For Robotics* (SDFR), but of course usage is not limited to these courses; anyone who can benefit from this readily-installed VM can use it.

C.2 VM Contents

The Virtual Machine does *not* contain course-specific software (such as templates for assignments; these have to be downloaded by the students from Canvas). However, it does contain software and tools that are used by the courses, being:

- Ubuntu Desktop 22.04
- Visual Studio Code (VS Code) with some extensions:
 - cpptools, cpptools-extension-pack *Working with cpp*
 - cmake-tools *Working with cmake*
 - doxygen *Doxygen support*
 - githistory, gitlens *Better support for git from within VS Code*
 - latex-workshop *Work with L^AT_EX files*
 - remote-ssh *Directly edit files on a remote target such as a Raspberry Pi*
- ROS2 (Humble Hawksbill)
- SDL2
- Various small tools, e.g., git, vim, wget, ...
- Wine (required for 20-sim)
- 20-sim 5.0.2 (license key until 1 February 2024)

C.3 VM User name/password

There is one user installed:

- User: ram-user
- Password: ram-user

This user has admin rights.

C.4 Installation

1. Download and install Virtualbox (www.virtualbox.org). Version 7.0 is recommended.
2. If your host OS is Linux, you might need to add yourself to the `vboxusers` group. This can be done by typing into a terminal (on your host OS):

```
sudo usermod -a -G vboxusers `whoami`
```

and logout/login afterwards.
3. For webcam support (required for the assignments): go to www.virtualbox.org/wiki/Downloads and download and run the Virtualbox Extension Pack.
4. Download the virtual machine file `Ubuntu22.04-RAM.ova`. Where you get it may vary. If you are attending a course, look on Canvas. Alternatively, ask the person you got this user manual from. The file is approximately 11 GB.
5. In Virtualbox, *Choose File | Import appliance* and import the virtual machine.

¹Depending on if the text was embedded in a larger document

6. Start the virtual machine.
7. Depending on the configuration, you may need to log in. For usernames and passwords, see Section C.2.
8. To use the webcam in the virtual machine, from the drop down menu item *Devices|Webcams* of the Virtualbox window, check the appropriate webcam. Test if this works by typing (in a terminal on the virtual machine):

```
ros2 run image_tools cam2image --ros-args -p show_camera:=true
```

C.5 Tips & Tricks

- The `ros2` command is available directly (i.e., *source*ing the ros installation directory is not needed). If you don't want this, comment out the lines

```
source /opt/ros/humble/setup.bash
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
```

from the file `~/.bashrc`.

D Visual Studio Code

At this moment, Visual Studio Code¹ is the most-used editor for coding. It is available for Windows, Mac and Linux and has an enormous amount of available extensions. We encourage you to use this editor (if you don't already). In the remainder of this chapter we will use the term VSCode for the editor. On the internet there are thousands of helpful pages on how to use VSCode in specific circumstances. Also, the application itself provides help; e.g., there are quick tutorials linked on the opening screen.

D.1 C/C++ Extension Pack

To make use of all features listed below, make sure you have the *C/C++ Extension Pack* extension installed. This pack contains nine useful extensions for developing C/C++ code, including the Remote-SSH extension (Section D.4).

D.2 Configuration files

Configuration of VSCode usually goes per project/(VSCode-)workspace. When you open a folder in VSCode, it creates a directory `.vscode` in the base directory, containing one or more settings files. These files are human-readable (and -editable):

- `.vscode/settings.json` General project settings such as the color scheme used.
- `.vscode/tasks.json` Compiler build settings
- `.vscode/launch.json` Debugger settings
- `.vscode/c_cpp_properties.json` Settings for correctly parsing/checking C/C++ files. For example, you supply include-paths here so that the VSCode can look up header files and use the information in there for code completion, real-time error checking, hinting etc.

You can edit these files so that they optimize the integration for your system. For example, they can contain paths to header files that VSCode then can use for syntax highlighting and code-completion. Some VSCode extensions provide (some of) the configuration files automatically.

D.3 VSCode and ROS2

There is a nice VSCode Extension to make Visual Studio Code work properly with ROS2. The extension is inconveniently called ROS, but it works for both ROS(1) and ROS2. This extension configures VSCode (via the files described in Section D.2 so that it does code-completion, building and debugging (including breakpoints) of ROS nodes.

Install the ROS extension from Microsoft (ms-iot.vscode-ros) yourself; it is not in the Virtual Machine that we provided.

D.3.1 Preparing your workspace for VSCode

The extension works not great for individual nodes; if you want debug a node you *must* create a launch file for it and configure your workspace accordingly. Do the following:

1. Create a launch file in the appropriate directory
(`<ros_ws>/src/<package name>/launch/mylaunchfile.launch.py`). See the ROS2 tutorials.
2. Add the following line to `package.xml`:
`<exec_depend>ros2launch</exec_depend>`
3. Add the following lines to `CMakeLists.txt`:

¹<https://code.visualstudio.com/>

```
!# install the launch directory
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}/
)
```

4. Remove the directories `build`, `install` and `log` from your workspace (to allow VSCode to do a complete rebuild of the workspace).

D.3.2 Opening the Workspace and enable ROS Extension in VSCode

In VSCode, choose File|Open Folder and open the root of your workspace folder (i.e., the folder where `src`, `build`, `install` and `log` are situated normally). This acts as the *base directory* for VSCode. VSCode now automatically recognizes the ROS2 workspace. Hence it will (later) create `.vscode/c_cpp_properties.json` and `.vscode/settings.json` with ROS-friendly settings. This enables syntax highlighting and code completion (even of your own custom message types!).

D.3.3 Building your ROS workspace

Press Ctrl-Shift-B to build your entire workspace. VSCode asks you what/how to build:

- Choose 'Colcon: build' as build task.

This should invoke `colcon build` with some additional parameters for convenient debugging.

D.3.4 Debug a node (with the built-in debugger)

This requires a `launch.json` file to be automatically generated. For this:

- Go to 'Run and Debug' in the side bar (Ctrl-Shift-D)
- Close all files in VSCode (failure to do this will not give the ROS option in the next step)
- In the side bar, choose 'Create a launch.json file'.
- Choose ROS as debugger
- Choose ROS Launch
- Choose the appropriate package name and node name.

Now a `launch.json` file is created. To start debugging, set some breakpoints in your `cpp` files and press the green triangle in the 'Run and Debug' sidebar (or press F5).

D.3.5 Run a node

The equivalent of `ros2 run` is:


- Press Ctrl-Shift-P to open the command window of VSCode
- ROS: Run a ROS Executable
- Choose the package and node name

D.3.6 More information on VSCode and ROS2

A good source on how to work with VSCode and ROS2 is the *Visual Studio Code ROS Extension* video series on Youtube: <https://www.youtube.com/playlist?list=PL2dJBq8ig-vihvDVw-D5zAYOArTMIX0FA>.

D.4 Using VSCode for remote editing

You can use VSCode on your own computer work with files on a remote computer, for example a Raspberry Pi. This is not limited to editing the files, but you can also compile, run and debug remote files as well as using `git` on those files.

In order to use this feature, you need the *Remote - SSH* extension of VSCode installed. Also, you need SSH access to the remote computer (test in a terminal window). Then, to use it, press the -icon on the lower left of the screen.

After connecting, VSCode automatically downloads some 'server code' to the remote computer to support the editing/developing process. However, not all functionality is available right away. In particular, for syntax checking and debugging/running C/C++ files you need to install the 'C/C++ Extension Pack' extension on the remote computer. This is how to do it:

1. In VSCode, log in to the remote computer as described above.
2. Press Ctrl-Shift-P to open the Command Palette
3. Choose the *Remote: Install Local Extensions In 'Remote'* command
4. Choose the *C/C++ Extension Pack* extension.

E Jiwy simulator

Provided for Assignment set 1 is a ROS2 node that simulates the Jiwy behaviour. The node uses the webcam of your computer (or, actually it accepts an image stream on a topic; you can offer webcam footage to that) as input.

Instead of physically rotating the camera, it outputs a part of the full camera view in a topic. Additionally, it outputs its actual pan and tilt angle. See Figure E.1.

As an input, the node expects a setpoint, i.e., a desired pan and tilt angle. Dynamics are modelled as a rather slow (to see the effect) first order system. End stops are included as well.

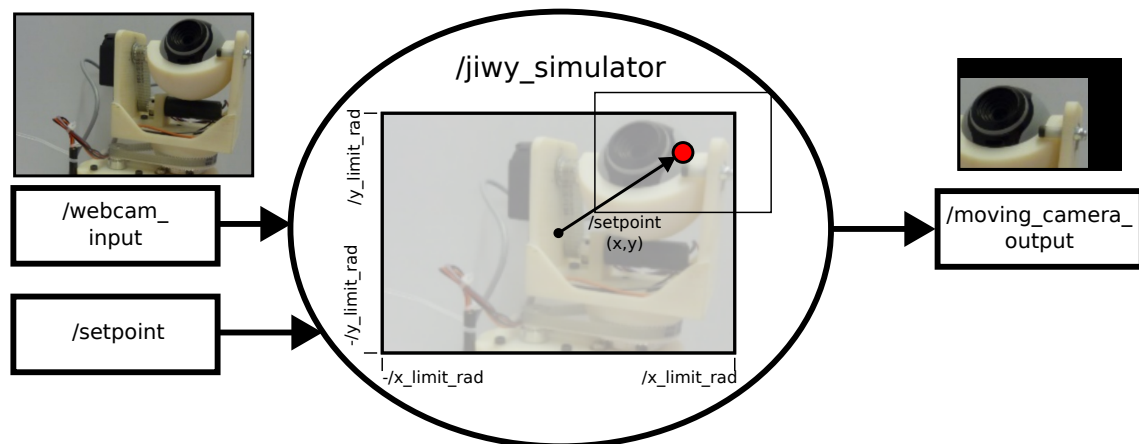


Figure E.1: Schematic drawing of the functionality of the Jiwy Simulator node.

E.1 Dependencies

The node used the custom message type `asdfr_interfaces/msg/Point2`. Before invoking the node, make this message type known to ROS2 (in the `ros2_nodes` directory, invoke `. install/local_setup.bash`).

The message type of `Point2` is:

```
~$ ros2 interface show asdfr_interfaces/msg/Point2
float64 x
float64 y
```

E.2 Input topics

`/setpoint`

The desired pan (x) and tilt (y) angle of the simulated camera in radians.

- The message type is `asdfr_interfaces/msg/Point2`.
- A positive pan (x) angle makes the camera look to the right.
- A positive tilt (y) angle makes the camera look upward.
- `[x=0, y=0]` makes the camera look straight forward.
- Setting a setpoint outside the working area (i.e., beyond the end stops) is allowed; the camera will just move to the end stop and not further.
- You can publish to this topic as often (or as little) as you like.
- The initial setpoint is `[x=0, y=0]`.
- You can give setpoints to the Jiwy Simulator from the command line with the following (standard) ROS2 command:
`ros2 topic pub --once /setpoint asdfr_interfaces/msg/Point2 "{x: 0.3, y: 0.2}"`
 Note that you *must* place a whitespace after the `:`'s. Also make sure you have executed `. install/local_setup.bash` first.

`/webcam_input`

A webcam stream of a stationary webcam (e.g., your laptop's webcam).

- Consider using `ros2 run image_tools cam2image` with appropriate topic remapping.

E.3 Output topics

`/position`

The actual pan (x) and tilt (y) angle of the simulated camera in radians.

- This topic is published to at approximately (but not necessarily precisely) 100 Hz.
- Refer to the `/setpoint` topic for details about the positive orientation and zero angle.

`/moving_camera_output`

The simulated output image of the simulated camera.

- This topic is published to whenever a new message from `/webcam_input` arrives.
- It is a part of the original input image; padded with black pixels if necessary.
- The width and height of the output image are half the width and height of the input image (rounded down if necessary).
- The center of the output image is looking at `/position`.
- The position scaling is such that at the end-stop positions the center of the output image coincides with the edge of the input image.

E.4 Parameters

`/tau_s`

The time constant of the first-order dynamics system, in seconds. You can freely change this while running.

`/x_limit_rad`

`/y_limit_rad`

The orientation, in radians, at which an end stop is modelled. A negative endstop is at `-*_limit_rad`. The values influence the scaling from pixels to orientation angles; see above (`/moving_camera_output`). You can freely change these parameters while running. Doing so gives an instantaneous jump in camera image and may give an instantaneous jump in `/position`. In particular, you can adjust the end stops so that they align with the physical field of view of your webcam.

E.5 Differences with a real Jiwy

This simulation is a very crude one; there are dozens of differences. The most important ones are:

- The physical Jiwy setup accepts PWM signals as inputs which encode the motor voltage. Normally, one would add a (software) closed-loop position controller to this which determines these signals based on setpoints \mathbf{x}_{set} (this is what you will develop in Assignment set 3). So, in fact, this simulator does not simulate the bare Jiwy setup, but rather the setup plus the closed-loop position controller.
- The field of view of the physical Jiwy is much larger than the field of view of the simulator:

	Simulator	Physical Jiwy	Unit
Pan	-0.8...0.8 (adjustable)	$\approx -3...3$	rad
Tilt	-0.6...0.6 (adjustable)	$\approx -1...1.5$	rad

- In the physical Jiwy, the camera image rotates (and, thus, distorts) while in the simulation the camera image translates over the input image.
- The speed of the dynamics is not the same; also the physical jiwy acts as a 4th order system (though dominant 2nd-order behavior); the simulation generates first order trajectories only.

E.6 Getting and compiling the node

E.6.1 Getting

You can download the code from Canvas. Unzip into the directory of your choice.

E.6.2 Compiling

Make sure that the ROS2 commands are sourced. Since this is a normal ROS2 package, the standard ROS2 compilation instructions apply:

```
cd <your_root_dir>/ros2nodes
colcon build
```

If this fails, you may have luck trying to build the `asdf_interfaces` package first and sourcing it before building the `jiwy_simulator` package:

```
cd <your_root_dir>/ros2nodes
```

```
colcon build --packages-up-to asdfr_interfaces # Build only asdfr_interfaces
. install/local_setup.bash
colcon build # Build all
```

Bibliography

Student Charter (2020), Student Charter at University of Twente.

<https://www.utwente.nl/en/ces/sacc/regulations/charter>