ASSIGNMENT 1 – CAMERA AND JIWY SIMULATOR

Bishoy Essam Samir Yassa Gerges, bishoy.gerges@student.utwente.nl, s2921073, M-ROB
Panteleimon Manouselis, p.manouselis@student.utwente.nl, s3084493, M-ROB

IMAGE PROCESSING WITH ROS2

Camera input with standard ROS2 tools

Showing that the system works:

Figure 1: rqt_graph showing communication between /cam2image and /showimage nodes

Once the node `cam2image` is executed, it retrieves images from the camera and publishes them to the topic `/image`. A message is printed on the terminal to indicate the published image frame number # stamp. While `cam2image` node is active, the node `showimage` subscribes to the `/image` topic to retrieve and display the images in a window. Unfortunately, it was discovered that the displayed images had an incorrect color space. Specifically, it appeared that there was a switch between the R channel and the B channel, causing red objects to appear blue and vice versa. Although this issue was identified, there were no parameters available to manipulate and fix this problem at the current stage.

1. `depth` and `history` are two parameters that affect the quality of service (QoS) settings for ROS 2 communication. `depth` specifies how many messages (N) can be stored in a queue before they are discarded, while `history` specifies whether to keep only the last N messages specified by the `depth` parameter (`keep_last`) or all messages (`keep_all`) in the queue. In other words, the `depth` and `history` parameters provide functionality similar to that of `queue_size` in ROS 1.

2. These parameters might influence the responsiveness of the system by affecting how much memory is used for buffering messages and how often messages are dropped or delivered out of order. For example, a larger `depth` or a `keep_all` history might increase the memory usage and latency of the system, but also reduce the chance of losing messages. Conversely, a smaller `depth` and a `keep_last` history decreases the memory usage and latency of the system, but also increases the chance of losing messages.

3. The given parameter values of `depth:=1` and `history:=keep_last` are good choices for usage in a sequence controller because in real-time control (tracking), latency should be minimum, and the control command should be based on the most recently sensed value. The use of these parameter values means that only the most recent image is stored in the queue which is suitable for a sequence controller since it guarantees that the sent control command is based on the most recent (~live) image. Older images should be discarded as they are not important anymore (they should not affect the currently sent command).

4. By using the command `ros2 topic hz image`, it was shown that the frame rate of the generated messages is about 30 Hz. That means that a message is sent every ~33ms.

```
bishoy@Bishoy:~$ ros2 topic hz image
average rate: 29.924
        min: 0.031s max: 0.037s std dev: 0.00206s window: 31
```

Figure 2: inspecting the default publishing frequency of the image topic using the terminal.

5. One can influence the frame rate by using the `frequency` parameter when running the `cam2image` node. For example, changing the frequency to 1 Hz is achieved by running the following command: `ros2 run image_tools cam2image --ros-args -p depth:=1 -p history:=keep_last -p frequency:=1.0`
   The frequency is changed as shown:

```
bishoy@Bishoy:~$ ros2 topic hz image
average rate: 1.000
        min: 0.999s max: 1.001s std dev: 0.00060s window: 3
```

Figure 3: inspecting the publishing frequency of the image topic after changing it to 1 Hz.

6. There is a maximum for this frame rate which is determined by the acquisition time for a single image by the camera which is in this case 33ms (30 fps) (i.e., we cannot get images at a faster rate than that determined by the hardware capabilities). This can be checked using the following cmd:
   `v4l2-ctl -d /dev/video0 --list-formats-ext`

```
bishoy@Bishoy:~$ v4l2-ctl -d /dev/video0 --list-formats-ext
ioctl: VIDIOC_ENUM_FMT
        Type: Video Capture

        [0]: 'MJPG' (Motion-JPEG, compressed)
                Size: Discrete 1280x720
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 960x540
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 848x480
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 1280x720
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.033s (30.000 fps)
```

Figure 4: Terminal cmd to show the camera fps.

Adding a brightness node and Adding ROS2 parameters

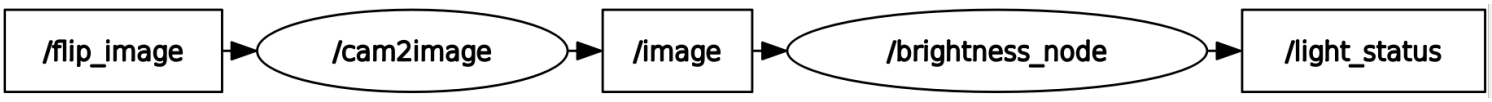

Figure 5: rqt_graph showing the communication between the /brightness_node and /cam2image nodes

We created the node `brightness_node` which subscribes to the `image` topic and publishes a boolean message every 33ms to the topic `light_status` indicating if a light spot is detected (`true`) or not (`false`). It works as follows:

1. It converts the ROS image to CV (OpenCV) image using cv_bridge.
2. Converts the RGB image to either:
    a. A single-channel grayscale image.
    b. Or the V channel of its HSV color space. The V channel includes the brightness information of the image.
3. Calculates the mean (average) of the intensity values of the converted single-channel image.
4. If the mean is greater than a set threshold, then it publishes `true` over the `/light_status` topic and `false` otherwise.

For this node we constructed two parameters:

1. The `threshold` parameter (default is `128`): to set the threshold value for the intensity (int).
2. The `conversion_mode` parameter (default is `gray`): to control the conversion method to either grayscale (`gray`) or the V channel of the HSV space (`hsv`). There were no noticeable differences between the performances of both modes. Both modes were implemented as valid options.

Showing that the system works:

- The parameter threshold can be set during start time of the node as follows:

```
bishoy@Bishoy:~$ ros2 run assignment1 brightness_node --ros-args -p threshold:=80
[INFO] [1678291143.556414073] [brightness_node]: Brightness: 39.200390
[INFO] [1678291143.556664284] [brightness_node]: Current set threshold: 80
[INFO] [1678291143.589036918] [brightness_node]: Brightness: 39.200390
[INFO] [1678291143.589123376] [brightness_node]: Current set threshold: 80
```

Figure 6: changing the default threshold of the /brightness_node at start time

- It can also be changed during run time by using the command:
  `ros2 param set /brightness_node threshold <value>`
  For example, using the above command for threshold value of 2, changes the threshold as shown:

```
[INFO] [1678291939.832949241] [brightness_node]: Current set threshold: 80
[INFO] [1678291939.865525468] [brightness_node]: Brightness: 39.234348
[INFO] [1678291939.865576322] [brightness_node]: Current set threshold: 80
[INFO] [1678291939.898771830] [brightness_node]: Brightness: 39.142643
[INFO] [1678291939.898863348] [brightness_node]: Current set threshold: 80
[INFO] [1678291939.931821021] [brightness_node]: Brightness: 39.142643
[INFO] [1678291939.931875826] [brightness_node]: Current set threshold: 80
[INFO] [1678291939.964772307] [brightness_node]: Brightness: 39.142643
[INFO] [1678291939.964849495] [brightness_node]: Current set threshold: 80
[INFO] [1678291939.997408700] [brightness_node]: Brightness: 39.172436
[INFO] [1678291939.997482110] [brightness_node]: Current set threshold: 80
[INFO] [1678291940.030619673] [brightness_node]: Brightness: 39.172436
[INFO] [1678291940.030710605] [brightness_node]: Current set threshold: 80
[INFO] [1678291940.063437162] [brightness_node]: Brightness: 39.172436
[INFO] [1678291940.063564772] [brightness_node]: Current set threshold: 2
[INFO] [1678291940.096776444] [brightness_node]: Brightness: 39.165115
[INFO] [1678291940.096902645] [brightness_node]: Current set threshold: 2
[INFO] [1678291940.129465283] [brightness_node]: Brightness: 39.165115
[INFO] [1678291940.129592713] [brightness_node]: Current set threshold: 2
[INFO] [1678291940.162846083] [brightness_node]: Brightness: 39.165115
[INFO] [1678291940.162914982] [brightness_node]: Current set threshold: 2
[INFO] [1678291940.195843606] [brightness_node]: Brightness: 39.008190
```

Figure 7: terminal messages showing the change from threshold of 80 to 2 as a result of using ros2 param set command during run time

Running the `brightness_node` prints the current mean value of the intensity of the pixel values of the converted single-channel image (step 3 above). This can be used to test and try what the mean value is when the light spot is on and that when the light spot is off. This is helpful to set the suitable threshold.

The published message can be monitored using the cmd: `ros2 topic echo /light_status`

```
bishoy@Bishoy:~$ ros2 topic echo /light_status
data: false
---
data: false
---
```

Figure 8: monitoring the /light_status topic using the ros2 topic echo /light_status command

Simple light position indicator

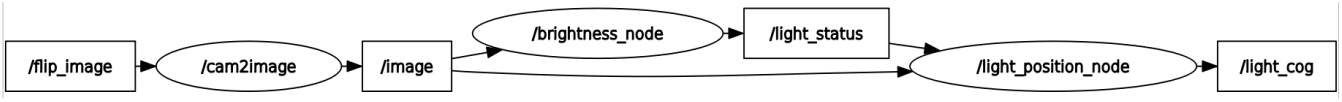

Figure 9: rqt_graph showing the communication between the /light_position_node, /brightness_node and /cam2image nodes

We developed the `light_position_node` to subscribe to the `image` topic and the `light_status` topic to determine the presence of a light spot. Whenever a light spot is detected, the node applies the steps specified in the manual (1.1.4) to calculate the center of gravity (COG) of the spot. The node then publishes the COG position via the `light_cog` topic. Additionally, the COG is visually highlighted with a red circle and its position is displayed on the image, as shown below:

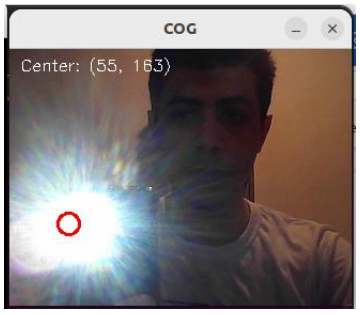

Figure 10: /light_position_node displays and marks the COG position on the image

Design choices:

For ease of use of the node and to make it flexible under different conditions (e.g., different project, webcam or lightning conditions), the node is made such that it calculates and displays the histogram of the current image. This histogram can be used to determine the suitable threshold to binarize the image. For example, in the below histogram, we see a spike (resembling the bright spot) at a pixel value greater than 246. The rest of the image falls mostly below the pixel value 136. So, a suitable threshold can be chosen between 136 and 246. For flexibility, this threshold can be changed using the parameter `pixel_threshold` of the node.

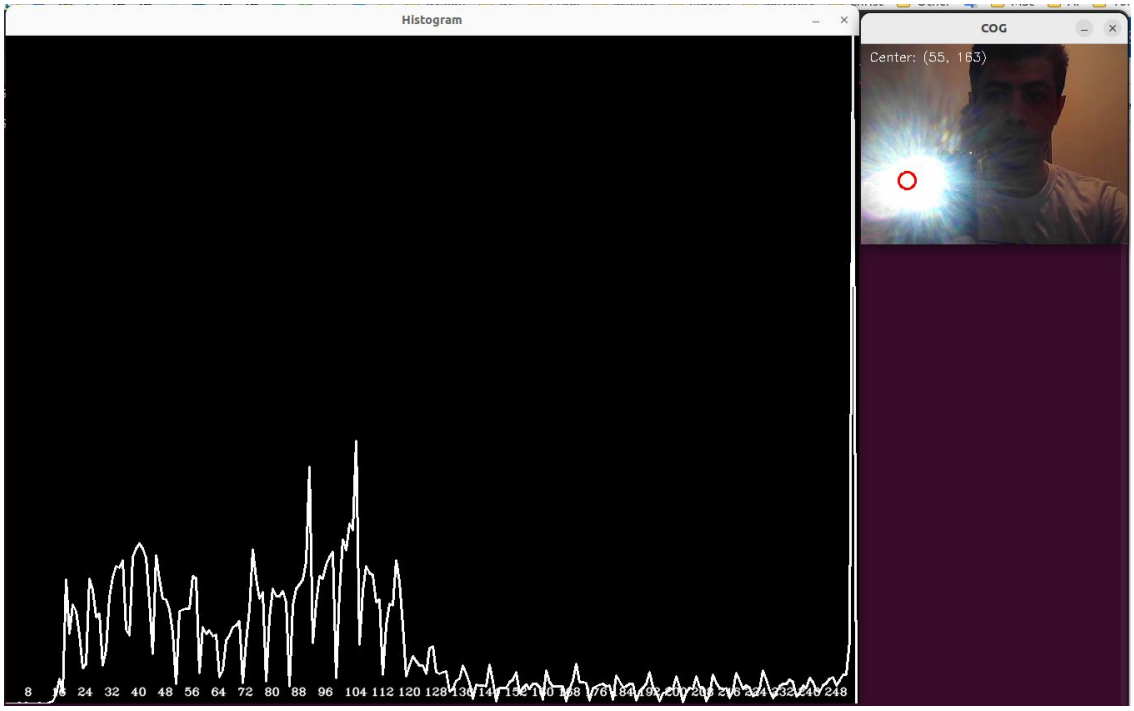

Figure 11: histogram of the current image to determine the suitable threshold value

SEQUENCE CONTROLLER

Unit test of the Jiwy Simulator

To evaluate the performance of the Jiwy Simulator's controller, we designed and implemented a ROS2 node called "jiwy_unit_test". This node generates a series of setpoints for the simulator to track. The setpoints are published to the "/setpoint" topic at a frequency of 10 Hz (every 100 ms). Each setpoint consists of a 2D point (x, y) within a specific range that corresponds to the “physical limits” defined in the Jiwy Simulator. Specifically, the x-coordinate of the setpoints is randomly generated between -0.8 and 0.8, and the y-coordinate is randomly generated between -0.6 and 0.6.

To test the controller's ability to track the setpoints accurately and quickly, we intentionally made each setpoint independent from the previous one, meaning that there is no correlation or pattern between the generated setpoints. This design choice allows us to observe how the controller responds to sudden changes in the setpoints, and how well it tracks them. By analyzing the behavior of the Jiwy Simulator's controller output (published in the “/position” topic) in response to the generated setpoints, we can evaluate its performance in terms of accuracy, speed, and stability. This information can help us fine-tune the controller's parameters and improve the overall performance of the Jiwy Simulator. In the figure below one can observe how the “jiwy_unit_test” node publishes to the setpoints topic to which the jiwy_simulator subscribes.

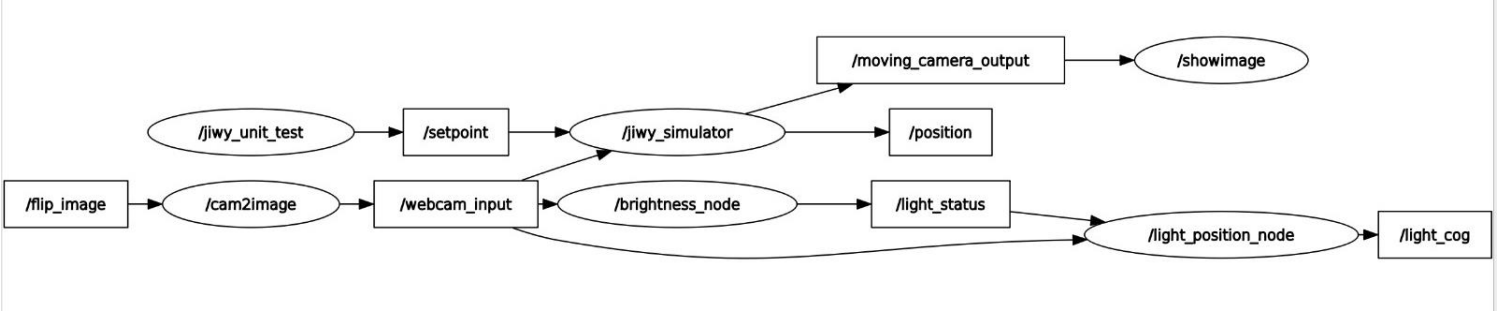

Figure 12: Rqt_graph showing the jiwy_unit_test node and its interaction with the jiwy_simulator

To evaluate the system's performance, we plotted the setpoint and actual position of the system over time. The actual position is the output of the system, which responds to the setpoint with a first-order system. The following figures illustrate the relationship between the setpoints and actual positions for various time constants (τ). The first three graphs represent the x-coordinate of the setpoint (in red) and the actual position (in blue) for different time constants, while the last three graphs depict the y-coordinate of the setpoint (in red) and the actual position (in blue) for the same set of time constants. By analyzing these plots, we can observe the effect of the time constant on the system's performance.

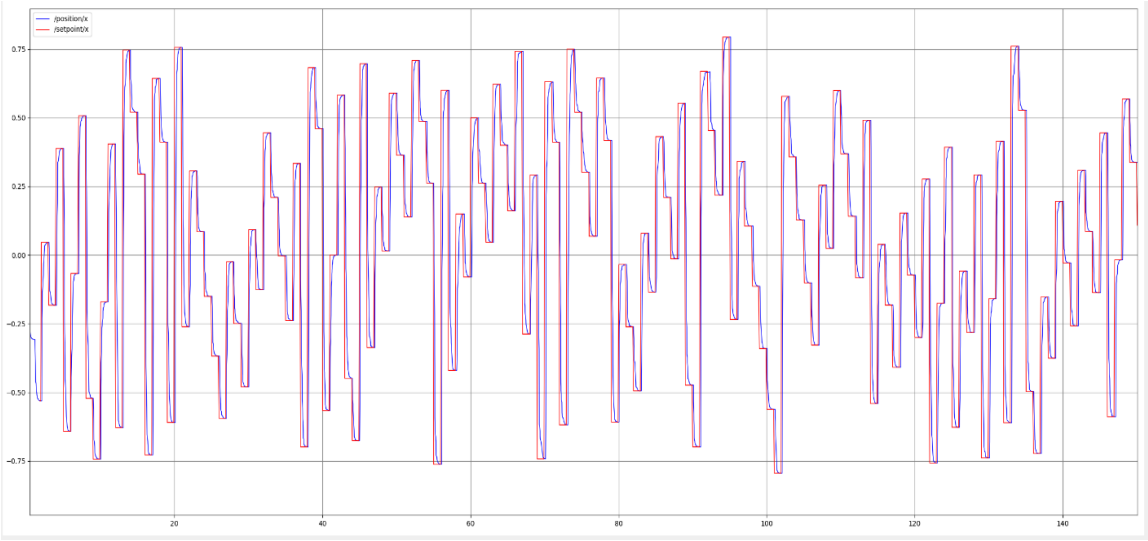

Figure 13: x-coordinate of both setpoint (red) and actual position (blue) for $\tau = 0.1$

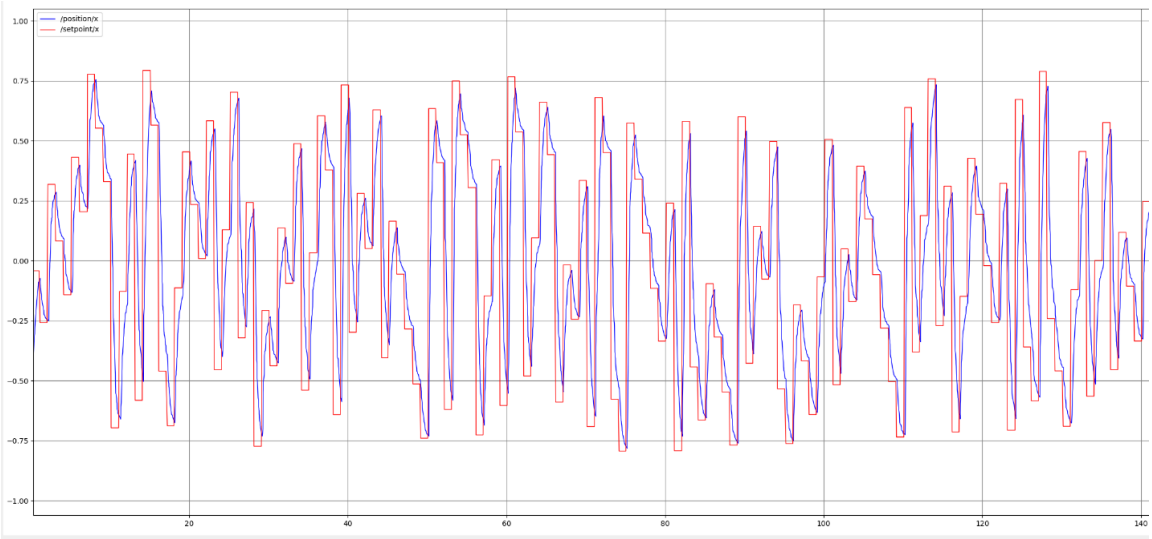

Figure 14: x-coordinate of both setpoint (red) and actual position (blue) for $\tau = 0.3$

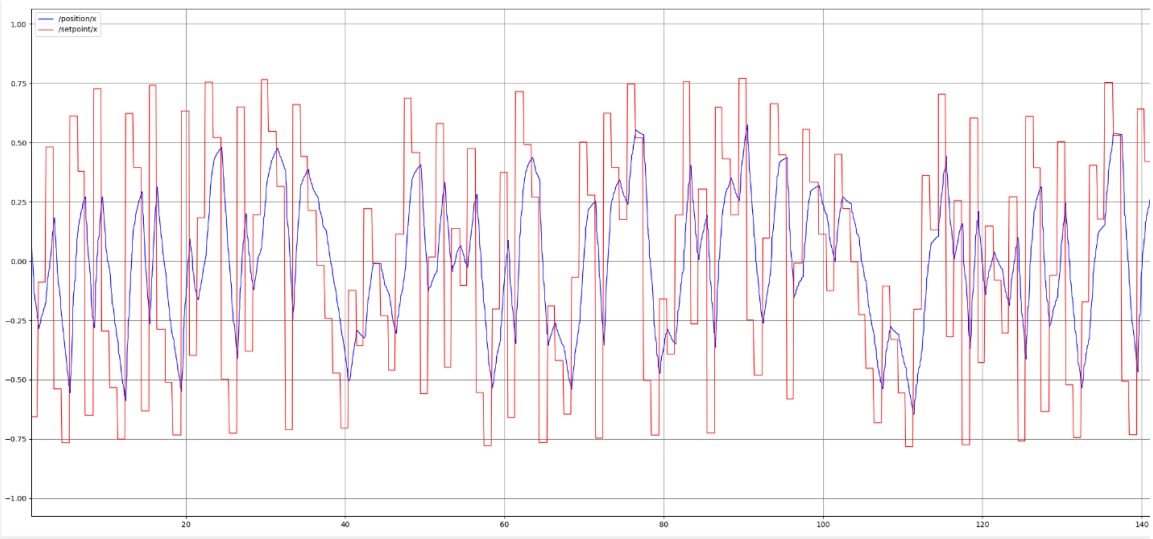

Figure 15: x-coordinate of both setpoint (red) and actual position (blue) for $\tau = 1.0$

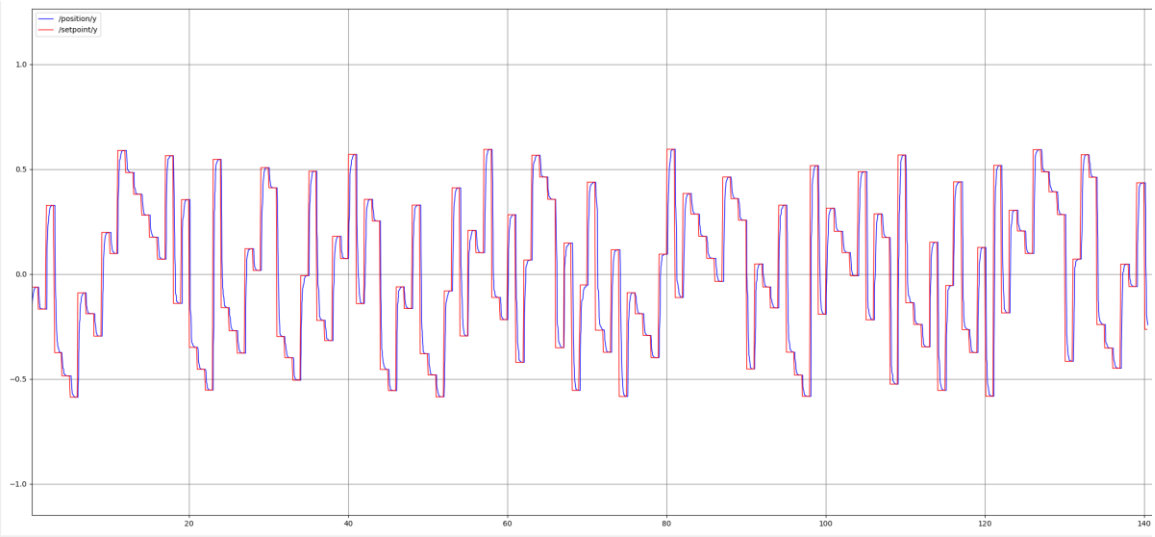

Figure 16: y-coordinate of both setpoint (red) and actual position (blue) for $\tau = 0.1$

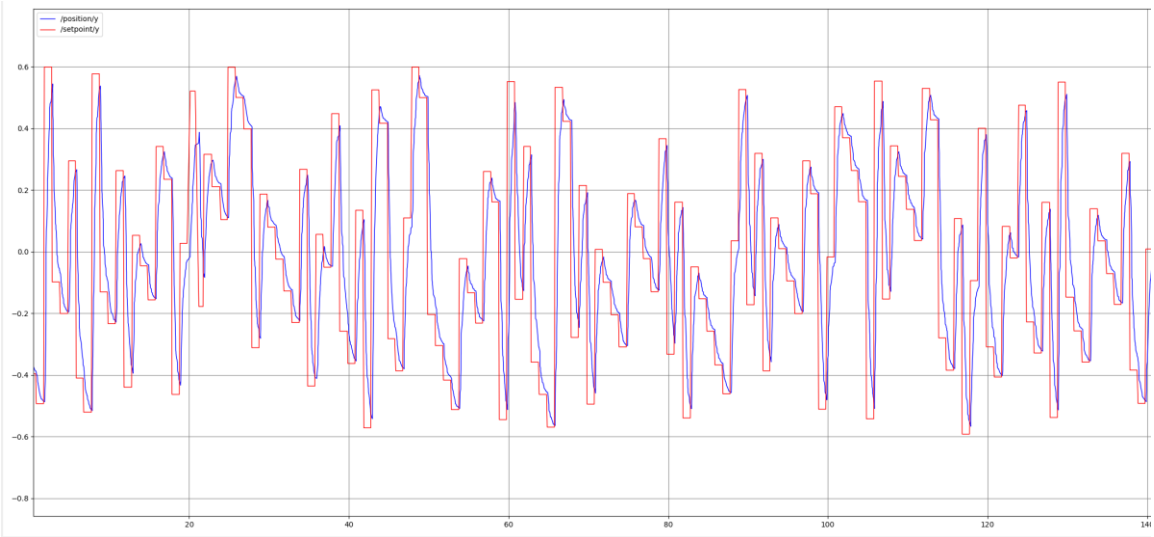

Figure 17: y-coordinate of both setpoint (red) and actual position (blue) for $\tau = 0.3$

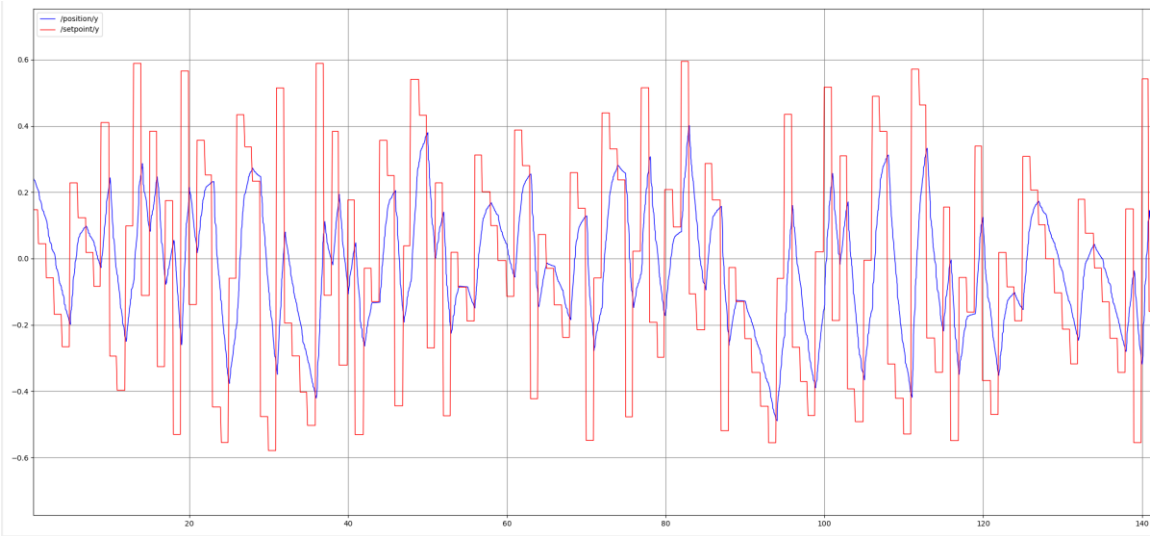

Figure 18: y-coordinate of both setpoint (red) and actual position (blue) for $\tau = 1.0$

To explain the results, we need to look at the time constant of the first order system. The time constant is a measure of how fast the system responds to changes in the input and is inversely proportional in magnitude to the pole of our system ($\tau = \frac{1}{\alpha}$), where $-\alpha$ is the pole's location. In our case, the first order system is being represented by the Jiwy Simulator node. The time constant is the time it takes for the system to reach 63.2% of its steady-state value in response to a step input. In Figure 13, Figure 14, Figure 16 and Figure 17 the time constant of the system is relatively small (between 0.1 and 0.3), and as a result the system will respond quickly to changes in the input. This means that the actual position will track the setpoint more closely, and the error between the setpoint and actual position will be small. Conversely, if the time constant of the system is large (Figure 15 and Figure 18), the system will respond slowly to changes in the input. This means that there will be a larger error between the setpoint and actual position. One characteristic of first order systems is that they do not exhibit overshoot, which was confirmed by observing the graphs.

Integration of image processing and Jiwy Simulator

In this part of the assignment, we created the node jiwy_sequence_controller that makes the Jiwy Simulator follow a bright light positioned in front or near our webcam. This node subscribes to three topics (namely, the /light_cog, /light_status, and /webcam_input topics). The /light_cog topic provides the position of the detected light, the /light_status topic provides the status of the detected light (i.e., on or off), and the /webcam_input topic provides the dimensions of the image captured by the webcam. Note that the /image topic is remapped to the /webcam_input topic. This is done in the launch file for each node that publishes to the /image topic.

The jiwy_sequence_controller node then calculates the setpoint that needs to be sent to the Jiwy Simulator to track the detected light. It does this by converting the position of the detected light from pixel locations to pan and tilt radian angles using the following equations:

$$x = \frac{2 * x_{cog} * x_{limit}}{width_{image}} - x_{limit} \qquad (1)$$

$$y = -\frac{2 * y_{cog} * y_{limit}}{height_{image}} - y_{limit} \qquad (2)$$

The minus term (–) in equation (2) is necessary since the direction of y is inverted in jiwy_simulator (see line 145 in jiwy_simulator.cpp). If no light is detected, the camera is made to "look" in the center (i.e., (0,0)). The code then publishes this setpoint on the /setpoint topic.

In the figure below one can observe how the "jiwy_sequence_controller" node, subscribes to the three aforementioned topics whilst it publishes to the setpoints topic to which the jiwy_simulator subscribes.

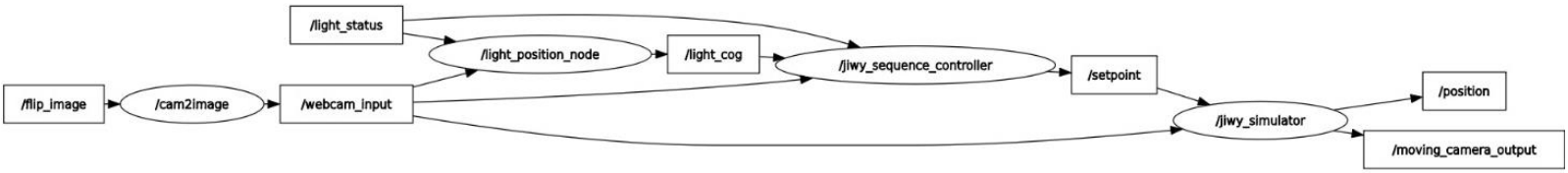

Figure 19: rqt_graph depicting the jiwy_sequence controller and its interaction with the jiwy_simulator node

The system we created is an open-loop system. The control action which is performed in the jiwy_sequence_controller node (i.e., the creation of the setpoint sequence based on the center of gravity of the light)

is independent from the output of the system (i.e., the actual position published in the position topic). More specifically, our sequence controller sends signals to the plant of the system (i.e., the jiwy simulator) and the system responds according to its pre-determined dynamics. The output of the system (the actual position) is not used to adjust the input signal (i.e., no feedback to the sequence controller).

On the contrary if we take an internal look to the jiwy_simulator node, one can observe that the node constitutes a closed loop system on its own. The jiwy simulator node receives the desired position (through the topic setpoint). The error is then calculated ($e = x_{desired} - x_{actual}$) and is used as an input to the controller. The controller performs a forward Euler calculation using the time step (which is approximately equal to 10ms) and the time constant (τ):

$$x_{new} = x_{actual} + \left(\frac{e}{\tau}\right) * (time\ step)$$

The new true position is fed back to the "internal" closed-loop system (jiwy simulator) in order to calculate a new error (the desired position remains the same for some time). This process repeats approximately 100 times per second, whereas a new desired position is fed to the system every second (can be controlled based on how frequently we publish to the setpoint topic). As mentioned in the previous section by increasing the value of the time constant, one would make the systems response slower (i.e., the tracking would be more inaccurate).

Our system is similar to an open loop system that is comprised by a sequence controller and the servo motor. The servo motor itself has an internal closed-loop control while the system (sequence controller and the servo motor) is open loop. This is because the servo motor doesn't provide any feedback to the sequence controller.

ROS2 is appropriate for this type of system as it provides a flexible and modular framework for building complex robotic systems. When it comes to programming Soft Real Time (SRT) processes like image processing, AI, and motion planning, ROS can be an ideal choice. This is because ROS helps in managing the development of such algorithms and their interactions with inputs and outputs. This, in turn, frees up more time and resources to focus on refining the algorithm itself. ROS2's modular architecture allows different parts of the system to be developed and tested independently, making it easier to develop and debug complex systems. In this case, the different parts of the system (i.e., the node that captures the image from the webcam, the node that processes the image to determine the position of the light, and the node that sends the appropriate setpoint to the Jiwy Simulator) can be developed and tested independently, and then integrated together to create the complete system.

However, there are some limitations to using ROS2 for real-time systems. One limitation of ROS2 is that it is not a hard real-time system, meaning that it cannot guarantee that a particular computation will be completed within a specific amount of time. This can be a problem for certain applications where real-time performance is critical, such as in safety-critical systems or high-performance control systems.

Another limitation is that the performance of ROS2 can be affected by factors such as network latency, message size, and system load. These factors can lead to delays and inconsistencies in the timing of messages, which can be problematic for systems that require precise synchronization and coordination (i.e., hard real-time systems).