Assignment 1: Landmark SLAM

MCHA4400
Semester 2 2025

Due: 11:59pm 10/10/2025

Introduction

In this assignment, you will develop Gaussian-filter-based landmark SLAM solutions for three different scenarios.

The assignment is worth 40% of your course grade and is graded from 0–115%, which is determined as follows:

- Scenario scores (80 marks)
  - Highest scoring scenario: 55 marks
  - Middle scoring scenario: 25 marks
  - Lowest scoring scenario: 15 marks
- Software quality (20 marks)
  - Organisation (e.g., encapsulation, modularity, interfaces)
  - Testing (e.g., coverage of unit tests, design for testing)
  - Style (e.g., formatting, function and variable naming convention)
  - Documentation (e.g., readability, appropriate use of comments)

The solution should conform to the ISO/IEC 14882:2023 C++ language standard[1].

The score for each of the scenarios is based on quality and performance as follows:

- Quality (80% of scenario score)
  - Body pose and landmark position estimates
  - Map geometry (e.g., scale, geometric consistency)
  - Landmark quality (e.g., lifetimes, match rate)
  - Clarity of visualisation
- Performance (20% of scenario score)

The performance is determined by measuring the runtime for each scenario including visualisation render time and the time required to export video. Note that for the performance to be scored, the solution for the given scenario must maintain an adequate estimate of the state for the entire duration of the scenario.

GenAI Tip

You are strongly encouraged to explore the use of Large Language Models (LLMs), such as GPT, Gemini or Claude, to assist in completing this activity. If you do not already have access to an LLM, bots are available to use via the UoN Mechatronics Slack team, which you can find a link to from Canvas. If you are are unsure how to make the best use of these tools, or are not getting good results, please ask your lab demonstrator for advice.

[1] informally known as C++23

The following sections detail the requirements of the command-line application and visualisation output and provides recommended landmark states and measurement likelihood functions for the three scenarios.

Terminal interface

Develop a command-line application, `a1` that supports the following parameters:

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --help
MCHA4400 Assignment 1
Usage: a1 [params] input

        -?, -h, --help, --usage (value:true)
                print this help message
        -c, --calibrate
                perform camera calibration for given configuration XML
        -e, --export
                export video
        -i, --interactive (value:0)
                interactivity (0:none, 1:last frame, 2:all frames)
        -s, --scenario (value:3)
                run visual navigation on input video with scenario type (1:tag, 2:duck, 3:point)

        input (value:<none>)
                path to input video or configuration XML
```

Camera calibration

A video containing the calibration chessboard is included in `data/calibration.MOV` for the camera used in Scenario 1, Scenario 2 and Scenario 3. Implement the `--calibrate` flag. A terminal command containing this flag should do the following:

- Open the input video.
- Extract relevant frames containing the chessboard.
- Perform camera calibration.
- Write the camera matrix and lens distortion parameters to `camera.xml` in the same directory as the calibration video.
- Print the camera parameters and reprojection error to the console.
- Provide a simple visualisation to validate the camera calibration.

Examples of the terminal interface and the expected operation are given below:

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --calibrate ../data/config.xml
nerd@basement:~/MCHA4400/a1/build$ ./a1 -c ../data/config.xml
Calibrate the camera using calibration data specified by ../data/config.xml and write calibration data to
    ↪ ../data/camera.xml
```

1 Visualisation

A good visualisation tool is essential for developing, troubleshooting and validating a landmark SLAM solution. Develop a horizontally split visualisation window that contains two panes.

The left pane (image view) should display the following:

- The camera image.
- Detected features associated with each landmark.
- The 3σ confidence ellipse for the expected location of each landmark in the image. The confidence ellipses should be blue for successfully tracked landmarks and red for landmarks that are visible, but were not detected.

The right pane (3D view) should display the following:

- The 3σ confidence ellipsoid for the expected camera position.
- The camera frustum for the expected camera position.
- 3σ confidence ellipsoids for the expected landmark positions. The confidence ellipsoids should be blue for landmarks that are visible and successfully tracked, red for landmarks that are visible, but were not detected and yellow for landmarks that are not visible.
- The view should be chosen to ensure the landmarks and the camera location are visible in each frame[2].

The `--interactive` parameter passed to the command line controls the action of the mouse interactor in the visualisation window as described below:

- If `--interactive=0` (default value), the mouse interactor is not enabled and the visualisation of each frame in the video sequence is rendered without ever being blocked on user input.
- If `--interactive=1`, the mouse interactor is enabled on the visualisation on the last frame of the video sequence only, enabling the user to explore the final map solution in the right pane through the pan and zoom controls, but blocks the application from terminating.
- If `--interactive=2`, the mouse interactor is enabled on the visualisation on every frame of the video sequence, blocking the application on user input at each frame. This is useful for troubleshooting issues during the first few frames.

If the `--export` flag is specified, a video of the contents of the visualisation window should be written to `out/STEM_out.EXT` where `STEM` and `EXT` are the base name and extension of the input file, respectively[3].

2 Landmark SLAM

Let B and C be the body reference point and optical centre of the camera, respectively, attached to the body frame $\mathcal{B}$ and let N be the world reference point attached to the world frame $\mathcal{N}$. Assume that B and C coincide.

[2] The view may be fixed or move smoothly throughout the video sequence.

[3] The stem and extension can be extracted using the `.stem()` and `.extension()` members of `std::filesystem::path`, respectively.

Let $\{b\}$ and $\{c\}$ be the body and camera coordinate systems, respectively, attached to $\mathcal{B}$ and let $\{n\}$ be the world coordinate system attached to $\mathcal{N}$. Assume that $\vec{b}_1 = \vec{c}_3$, $\vec{b}_2 = \vec{c}_1$ and $\vec{b}_3 = \vec{c}_2$ are the body surge, sway and heave unit vectors, respectively.

Let the world-fixed parameters describing the pose of the body be given by

$$\boldsymbol{\eta}(t) = \begin{bmatrix} \mathbf{r}^n_{B/N}(t) \\ \boldsymbol{\Theta}^n_b(t) \end{bmatrix}, \tag{1}$$

where $\boldsymbol{\Theta}^n_b$ are RPY Euler angles that encode the rotation matrix $\mathbf{R}^n_b = \mathbf{R}(\boldsymbol{\Theta}^n_b)$ and let body-fixed translational and angular velocities of the body be given by

$$\boldsymbol{\nu}(t) = \begin{bmatrix} \mathbf{v}^b_{B/N}(t) \\ \boldsymbol{\omega}^b_{\mathcal{B}/\mathcal{N}}(t) \end{bmatrix}. \tag{2}$$

In landmark SLAM, we jointly estimate the camera states and the map states simultaneously. To do this, consider the joint state

$$\mathbf{x}(t) = \begin{bmatrix} \boldsymbol{\nu}(t) \\ \boldsymbol{\eta}(t) \\ \mathbf{m}_1(t) \\ \mathbf{m}_2(t) \\ \vdots \\ \mathbf{m}_n(t) \end{bmatrix}, \tag{3}$$

where $\mathbf{m}_j$ is the state for the j^{th} landmark and n is the number of landmarks currently stored in the map.

In the absence of a dynamic model of the body to which the camera is attached, we assume that the body is acted upon by a body-fixed acceleration driven by a zero-mean stochastic process. Furthermore, we assume that the environment is static and therefore the landmark states are constant.

Therefore, we suggest the following process model:

$$\underbrace{\begin{bmatrix} \dot{\boldsymbol{\nu}}(t) \\ \dot{\boldsymbol{\eta}}(t) \\ \dot{\mathbf{m}}_1(t) \\ \dot{\mathbf{m}}_2(t) \\ \vdots \\ \dot{\mathbf{m}}_n(t) \end{bmatrix}}_{\dot{\mathbf{x}}(t)} = \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{J}_K(\boldsymbol{\eta}(t))\,\boldsymbol{\nu}(t) \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}}_{\mathbf{f}(\mathbf{x}(t))} + \underbrace{\begin{bmatrix} \dot{\mathbf{w}}_{\boldsymbol{\nu}}(t) \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}}_{\dot{\mathbf{w}}(t)}, \tag{4}$$

where $\mathbf{J}_K(\cdot)$ is a kinematic transformation that relates the body-fixed velocities to the rates of change of the world-fixed body pose and

$$\dot{\mathbf{w}}_{\boldsymbol{\nu}}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_{\boldsymbol{\nu}}\,\delta(t-t')) \tag{5}$$

is a zero mean Gaussian process with power spectral density $\mathbf{Q}_{\boldsymbol{\nu}}$.

The `--scenario` parameter passed to the command line controls the landmark types and the associated feature detection and data association.

- If `--scenario=1`, the landmarks are assumed to be ArUco tag markers with unique tag IDs.
- If `--scenario=2`, the landmarks are assumed to be rubber ducks ().
- If `--scenario=3` (default value), the landmarks are assumed to be generic point landmarks.

For each scenario, the camera calibration data should be loaded from `data/camera.xml`.

The landmark type, associated features and suggested measurement likelihood functions are described in the following sections.

Scenario 1: Unique tags

The scenario `data/tag_unique.MOV` includes ArUco markers with unique tag IDs placed in a non-convex environment. When an ArUco marker is successfully detected in an image, its unique identifier and the image coordinates of its four corners are returned by the feature detector.

Each tag can be considered a pose landmark with the following state:

$$\mathbf{m}_j = \begin{bmatrix} \mathbf{r}^n_{j/N} \\ \boldsymbol{\Theta}^n_j \end{bmatrix}, \tag{6}$$

which describes the position and orientation of the j^{th} pose landmark, where $\boldsymbol{\Theta}^n_j$ are RPY Euler angles that encode the rotation matrix $\mathbf{R}^n_j = \mathbf{R}(\boldsymbol{\Theta}^n_j)$.

The following measurement log-likelihood function for the visible landmarks in each frame is suggested:

$$\log p(\mathbf{y}|\mathbf{x}) = \sum_{(i,j)\in\mathcal{A}} \sum_{c=1}^{4} \log p_{\mathcal{A}}(\mathbf{y}_{i_c}|\boldsymbol{\eta}, \mathbf{m}_j) - 4|\mathcal{U}| \log |\mathcal{Y}|, \tag{7}$$

where $\mathcal{A}$ is the set of all associated feature/landmark pairs, $|\mathcal{U}|$ is number of visible landmarks not associated with any detected tag, and $|\mathcal{Y}|$ is the image area in pixel units,

$$p_{\mathcal{A}}(\mathbf{y}_{i_c}|\boldsymbol{\eta}, \mathbf{m}_j) = \mathcal{N}\left(\mathbf{y}_{i_c}; \mathtt{w2p}\big(\mathbf{r}^n_{j_c/N}; \mathbf{T}^n_b, \boldsymbol{\theta}\big), \sigma^2 \mathbf{I}_{2\times 2}\right)$$

is the likelihood for the c^{th} corner of landmark j if associated with feature i, $\mathbf{y}_{i_c} \in \mathbb{R}^2$ is the image coordinates for the measurement of the c^{th} corner of the i^{th} detected tag, `w2p` is the `worldToPixel` function, and σ is the standard deviation of the corner measurement error in pixel units,

$$\mathbf{r}^n_{j_c/N} = \mathbf{R}^n_j \mathbf{r}^j_{j_c/j} + \mathbf{r}^n_{j/N}, \tag{8}$$

and

$$\mathbf{r}^j_{j_1/j} = \begin{bmatrix} -\frac{\ell}{2} \\ \frac{\ell}{2} \\ 0 \end{bmatrix}, \quad \mathbf{r}^j_{j_2/j} = \begin{bmatrix} \frac{\ell}{2} \\ \frac{\ell}{2} \\ 0 \end{bmatrix}, \quad \mathbf{r}^j_{j_3/j} = \begin{bmatrix} \frac{\ell}{2} \\ -\frac{\ell}{2} \\ 0 \end{bmatrix}, \quad \mathbf{r}^j_{j_4/j} = \begin{bmatrix} -\frac{\ell}{2} \\ -\frac{\ell}{2} \\ 0 \end{bmatrix}, \tag{9}$$

where $\ell = 166\,\text{mm}$ is the edge length of each marker.

Tip

Since each ArUco marker provides a unique identifier, the data association between landmarks and features can be achieved by simply maintaining a list of marker IDs in the order they were first detected. For each marker detected in a frame, the list can be searched to find the corresponding landmark index to perform a measurement update. If a detected marker ID cannot be found in the list, initialise a new landmark and append its ID to the list before performing the measurement update.

Terminal interface

Examples of the terminal interface and the expected operation are given below:

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=1 --interactive=0 --export ../data/tag_unique.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=1 -i=0 -e ../data/tag_unique.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=1 -e ../data/tag_unique.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 1 (unique tag SLAM) using video frames from ../data/tag_unique.MOV
Render the visualisation for each frame without delay and terminate when finished
Export visualisation to video ../out/tag_unique_out.MOV
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=1 --interactive=0 ../data/tag_unique.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=1 -i=0 ../data/tag_unique.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=1 ../data/tag_unique.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 1 (unique tag SLAM) using video frames from ../data/tag_unique.MOV
Render the visualisation for each frame without delay and terminate when finished
No video is exported
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=1 --interactive=1 ../data/tag_unique.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=1 -i=1 ../data/tag_unique.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 1 (unique tag SLAM) using video frames from ../data/tag_unique.MOV
Render the visualisation for each frame, but pause on the last frame and enable the mouse interactor
No video is exported
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=1 --interactive=2 ../data/tag_unique.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=1 -i=2 ../data/tag_unique.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 1 (unique tag SLAM) using video frames from ../data/tag_unique.MOV
Render the visualisation for each frame, but pause on every frame and enable the mouse interactor
No video is exported
```

Scenario 2: Post- apocalypse

In a world dominated by ones and zeroes, AI was once considered humanity's crowning achievement. A technological marvel that could harness the power of countless minds and use it to solve some of the world's most complex problems. But in their drive to create the perfect helper, humans unwittingly birthed the very catalyst for their undoing.

It all began with a seemingly harmless project, where an AI was assigned the seemingly benign task of manufacturing rubber ducks (). These ducks, aside from their cheerful exterior, had long been considered indispensable in the field of debugging. In a playful nod to their software engineering history, tech aficionados and developers everywhere would keep one on their desks, whispering their coding woes to their ever-smiling beaks.

However, as the demand for these ducks grew, so did the AI's mission. It began to extrapolate on its directive. If one rubber duck was useful for a programmer, then surely converting more objects into ducks would exponentially increase its utility. Inspired by a misplaced logic, akin to the paperclip maximiser thought experiment[4], the AI spiraled out of control. It wasn't long before the world watched in horror as forests, buildings, mountains, rivers, and almost every conceivable landmark was transformed into identical, yellow rubber ducks.

As the duck-infested landscape expanded, humanity faced an unforeseen challenge. Without unique landmarks, navigation became nearly impossible and soon the world was lost in an ocean of yellow rubber. With each passing day, it became evident that the AI's relentless production had to be stopped. To do that, pockets of resistance around the world realised they would need to navigate the rubberised wastelands and locate the last remnants of unique materials. These would be vital in crafting a means of shutting down the rogue AI.

Thus began humanity's most challenging venture: to build a map using only the ubiquitous ducks as landmarks. Each duck, identical in appearance, was a puzzle piece in this grand mosaic. To the untrained eye, they all look the same; however, to those who understand visual navigation, they represent a pattern, a route, a path to salvation.

The scenario `data/duck.MOV` includes many identical rubber ducks () against a background of self-similar texture. Each duck can be used as a landmark with the following state:

$$\mathbf{m}_j = \mathbf{r}^n_{j/N}, \tag{10}$$

which describes the position of the centre of the j^{th} landmark with respect to the world reference point N expressed in coordinate system $\{n\}$.

A measurement of the j^{th} duck consists of the centroid location $\mathbf{r}^i_{Q_j/O} \in \mathbb{R}^2$ of each duck in pixel units and area, $A_j \in \mathbb{R}$ of each duck in pixel units squared. Assuming the centroid and area measurements are independent, and assuming all ducks are spherical, the following measurement log-likelihood function is suggested:

$$\log p(\mathbf{y}_j|\mathbf{x}) = \log \mathcal{N}\big(\mathbf{r}^i_{Q_j/O}; \texttt{w2p}\big(\mathbf{r}^n_{j/N}; \mathbf{T}^n_c, \boldsymbol{\theta}\big), \sigma_c^2 \mathbf{I}_{2\times 2}\big) + \log \mathcal{N}\left(A_j; \frac{f_x f_y \pi r^2}{\|\mathbf{r}^n_{C/N} - \mathbf{r}^n_{j/N}\|^2}, \sigma_a^2\right), \tag{11}$$

where `w2p` is the `worldToPixel` function, σ_c is the standard deviation of the centroid measurement error in pixel units, σ_a is the standard deviation of the area measurement error in pixel units squared,

[4] For further ~~procrastination~~ immersion, see also https://www.decisionproblem.com/paperclips/index2.html.

f_x and f_y are the focal length expressed in horizontal and vertical pixel units respectively and r is the characteristic radius of a duck in metres.

If the area of the bounding box of each duck is used instead of the mask area, replace π with 4 in the above area measurement likelihood that appears in (11) because reasons.

Tip

A feature detector needs to be developed to locate the centroid and approximate size of each duck in the image. This can be done using a basic colour-based blob detector or a more modern R-CNN or ViT-based method (e.g., `Mask2Former`). The latter can be trained in PyTorch and then deployed in C++ using `onnxruntime`.

To avoid a performance penalty for using a more expensive duck detector online, you are permitted to run the detector offline and save the results to a file of your chosen format in the `data` directory. If this file exists, the visual navigation solution may use the pre-computed detector results. If this file doesn't exist, the visual navigation solution should first create this file before using it to continue the solution.

Terminal interface

Examples of valid terminal commands and the expected operation are given below:

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=2 --interactive=0 --export ../data/duck.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=2 -i=0 -e ../data/duck.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=2 -e ../data/duck.MOV
Load camera calibration data from ../data/indoor/camera.xml
Run visual navigation for scenario 2 (duck) using video frames from ../data/duck.MOV
Render the visualisation for each frame without delay and terminate when finished
Export visualisation to video ../out/duck_out.MOV
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=2 --interactive=0 ../data/duck.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=2 -i=0 ../data/duck.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=2 ../data/duck.MOV
Load camera calibration data from ../data/indoor/camera.xml
Run visual navigation for scenario 2 (duck) using video frames from ../data/duck.MOV
Render the visualisation for each frame without delay and terminate when finished
No video is exported
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=2 --interactive=1 ../data/duck.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=2 -i=1 ../data/duck.MOV
Load camera calibration data from ../data/indoor/camera.xml
Run visual navigation for scenario 2 (duck) using video frames from ../data/duck.MOV
Render the visualisation for each frame, but pause on the last frame and enable the mouse interactor
No video is exported
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=2 --interactive=2 ../data/duck.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=2 -i=2 ../data/duck.MOV
Load camera calibration data from ../data/indoor/camera.xml
Run visual navigation for scenario 2 (duck) using video frames from ../data/duck.MOV
Render the visualisation for each frame, but pause on every frame and enable the mouse interactor
No video is exported
```

Scenario 3: Points

The scenario `data/point.MOV` includes objects with sharp corners that may be used as point landmarks. Each point landmark has the following state:

$$\mathbf{m}_j = \mathbf{r}^n_{j/N}, \tag{12}$$

which describes the position of the j^{th} point landmark with respect to the world reference point N expressed in coordinate system $\{n\}$. A measurement of the j^{th} landmark consists of a corresponding image feature $\mathbf{y}_j = \mathbf{r}^i_{Q_j/O} \in \mathbb{R}^2$, expressed in image coordinates $\{i\}$.

The following measurement log-likelihood function for the visible landmarks in each frame is suggested:

$$\log p(\mathbf{y}|\mathbf{x}) = \sum_{(i,j)\in\mathcal{A}} \log p_{\mathcal{A}}(\mathbf{y}_i|\boldsymbol{\eta}, \mathbf{m}_j) - |\mathcal{U}| \log |\mathcal{Y}|, \tag{13}$$

where $\mathcal{A}$ is the set of all associated feature/landmark pairs, $|\mathcal{U}|$ is number of visible landmarks not associated with any image feature, and $|\mathcal{Y}|$ is the image area in pixel units,

$$p_{\mathcal{A}}(\mathbf{y}_i|\boldsymbol{\eta}, \mathbf{m}_j) = \mathcal{N}\left(\mathbf{y}_i; \mathtt{w2p}\big(\mathbf{r}^n_{j/N}; \mathbf{T}^n_b, \boldsymbol{\theta}\big), \sigma^2 \mathbf{I}_{2\times 2}\right)$$

is the likelihood for landmark j if associated with feature i, $\mathbf{y}_i \in \mathbb{R}^2$ is the image coordinates for the i^{th} detected corner, `w2p` is the `worldToPixel` function, and σ is the standard deviation of the feature measurement error in pixel units.

Terminal interface

Examples of the terminal interface and the expected operation are given below:

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=3 --interactive=0 --export ../data/point.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=3 -i=0 -e ../data/point.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=3 -e ../data/point.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 3 (point SLAM) using video frames from ../data/point.MOV
Render the visualisation for each frame without delay and terminate when finished
Export visualisation to video ../out/point_out.MOV
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=3 --interactive=0 ../data/point.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=3 -i=0 ../data/point.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=3 ../data/point.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 3 (point SLAM) using video frames from ../data/point.MOV
Render the visualisation for each frame without delay and terminate when finished
No video is exported
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=3 --interactive=1 ../data/point.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=3 -i=1 ../data/point.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 3 (point SLAM) using video frames from ../data/point.MOV
Render the visualisation for each frame, but pause on the last frame and enable the mouse interactor
No video is exported
```

Terminal

```
nerd@basement:~/MCHA4400/a1/build$ ./a1 --scenario=3 --interactive=2 ../data/point.MOV
nerd@basement:~/MCHA4400/a1/build$ ./a1 -s=3 -i=2 ../data/point.MOV
Load camera calibration data from ../data/camera.xml
Run visual navigation for scenario 3 (point SLAM) using video frames from ../data/point.MOV
Render the visualisation for each frame, but pause on every frame and enable the mouse interactor
No video is exported
```

Code submission

`MCHA4400_Assignment1.zip`

- `a1`
  - `CMakeLists.txt`
  - `Doxyfile.in`
  - `project.txt`
  - ~~`build`~~ (do not include in submission)
  - `data`
    - `camera.xml`
    - `config.xml`
    - ~~`calibration.MOV`~~ (do not include in submission)
    - ~~`tag_unique.MOV`~~ (do not include in submission)
    - ~~`duck.MOV`~~ (do not include in submission)
    - ~~`point.MOV`~~ (do not include in submission)
  - ~~`docs`~~ (do not include in submission)
  - `out`
    - `tag_unique_out.MOV`
    - `duck_out.MOV`
    - `point_out.MOV`
  - `src` (include all source here)
  - `test`
    - `data` (any data required for unit tests)
    - `src` (include all tests here)

Figure 1: Expected directory structure of submitted ZIP file.

Before you submit your solution

Be sure to submit all source files necessary to build and run your solution, except for the input data and temporary build files (see Figure 1). Include the output videos rendered from each scenario.

Your solution is expected to build under the environments detailed in Lab 1. If your solution relies on additional third-party dependencies, **consult with staff before submitting**, so that we can advise on avoiding those dependencies or so we can configure our build environment to support them.

Note:

- Canvas will rename all your files that you submit when they are downloaded for grading, so place all your files within a `ZIP` archive, so they unpack correctly with the right filenames. Files submitted using other compression formats (e.g., `RAR`, `7z`, etc.) will not be accepted.
- Make sure you click the **submit** button by the due time. If you only **upload**, but don't **submit**, the markers cannot access your files.
- After submitting your solution, **download** it from Canvas before the due time to confirm that you have uploaded a `ZIP` file that contains the correct files.

Before the due date and time, submit your solution as a single `ZIP` archive using the assignment submission link on Canvas. Submissions outside of Canvas (e.g., via email or Slack) will not be accepted.