TASK BRIEF

Sim-2-Real Experience

Submission details:

Type:	"Lab book".
Length:	Various, see marking guides below for details.
Weighting:	14% of the overall subject grade.
Due Date:	Various submissions over the period of semester weeks 5-8 (inclusive), see marking guides below for details.
Instructions:	• Complete in teams of 2. • Upload "lab book" documents to LMS. • You must be physically present during your workshop session to complete the oral parts of the assessment.

The following learning outcomes are demonstrated through this assessment activity:

- **ILO4)** Compare and contrast the complexities of implementing decision-making and learning algorithms on real-world systems.
- **ILO5)** Communicate and collaborate on AI and decision-making projects through formats such as: technical reports, presentations, and informational videos, and reflection.

PURPOSE STATEMENT

Gaining hands-on experience is the primary purpose of this task. We focus on some challenges associated with transferring the performance of a policy from simulation -to- achieving comparable performance on the real-world robotic system, i.e., **"sim-2-real"**. The following is a high-level design process for synthesizing policies (where "sim-2-real" is captured by Steps 3-5):

(Step 1) Define and justify specifications for the high-level behaviours that the robot should perform, and translate these to an initial set of quantitative and/or qualitative specifications that the finalized autonomous robot should achieve.

(Step 2) Select sensors and actuators, potentially multiple options, that are expected to be appropriate for achieving the desired behaviours and specifications.

- There is almost always a trade-off that needs to be considered between the cost (of the sensors and actuators) and the performance (of the real-world robot), hence multiple options are generally considered.

(Step 3) Conduct initial experiments with the real-world robot to get to know its capabilities and its parameters, including the performance of the sensors and actuators selected in Step 2.

(Step 4) Perform simulation-based synthesis and analysis of multiple policies.

- This step includes selecting multiple parameter options for the same policy type, as well as considering multiple different policy types, such as: PID; linear state feedback; non-linear feedback; "optimal" designs (e.g., MPC); learning-based designs (e.g., RL).
- It is common practice to first design "simpler" policies, and then progress to more advanced policy techniques if the specifications are not met.

(Step 5) Transfer to the real-world system and test the most viable policies.

- Real-world testing is often cost and time intensive, hence, test only the "best" candidates.

(Step 6) Iterate back-and-forth between real-world results and simulation-based developments as guided by the specific robotic system and desired behaviours, i.e., iterate across Steps 1-5.

The actual design process may from the steps above, with some steps performed in parallel, and other steps iterated multiple times. Experience and engineering judgement determine the process. When in doubt, build up your experience base by following the steps in the order above.

OVERALL SYSTEM AND TASK DESCRIPTION

Overall description (i.e., "Step 0")

- The baseline robotic system we use is a 1:10 scale remote controlled (RC) car.
- The operating environment is a flat-floor space with a long continuous line on the floor.
- The car is equipped with:
  - A camera for detecting the line.
  - A computer for implementing sensor processing and feedback policies.
  - A motor for actuating the main drive wheels, and a servo for actuating the steering.
- The overall goal is for the car to autonomously follow the line.

Desired system bahaviour (i.e., "Step 1: define and justify specifications")
The policies that you design should aim to satisfy the specifications given in the following table.

Specification	Justification
(Spec.1) The car autonomously follows a continuous, unobstructed line from start to finish.	This is the minimum behaviour that completes the task.
(Spec.2) The center of the car should not deviate more than half a body width from the line.	This is representative of a lane width constraint, with a lane twice the width of the car (i.e., quite generous).
(Spec.3) The line tracking behaviour should be representative of natural driving behaviours, e.g., minimal oscillations and minimal jerky movements.	If a human is on-board the vehicle, then this represents a ride-comfort specification. If a human is observing the vehicle, then this represents a specification for gaining the human's trust of the autonomous vehicle's behaviour.
(Spec.4) The sequence of drive commands does not exceed the maximum allowed command nor exceed the maximum rate-of-increase command.	The maximum allowed command is representative of a speed limit. The maximum rate-of-increase is representative of a comfort constraint for forward acceleration, as well as reducing the likelihood of wheel slippage.
(Spec.5) Traverse the line from start-to-finish in minimum time.	This is closely tied to (Spec.4) and captures the intention that the car should, where possible, operate close to the speed limit.

Robotic System (i.e., "Step 2: sensor and actuator selection)"
The robotic system we use for this sim-2-real experience is an off-the-shelf 1:10 scale remote controlled (RC) car, specifically a Traxxas Slash (model 58034-61). To enable autonomous operation of the car it is retro-fitted with:

- A Raspberry Pi 4 for reading sensor data, processing sensor data, implementing decision-making policies, and communicating with your personal computer.
- A Logitech StreamCam web camera for visual feedback.
- A PCA9685 PWM Driver for commanding the drive motor and steering servo of the RC car.

Software for sensor and actuator interfaces (i.e., between Steps 2 and 3)
In order to be able to perform "Step 3: conduct initial experiments", we need to be able to retrieve data from the sensors selected in Step 2 and send commands to the actuators selected in Step 2. We have prepared software for retrieving images from the web camera, for adjusting parameters of a line-detection algorithm, and for commanding the drive motor and steering servo. Instructions for using this software are split across the 4 pages of this appendix.

Sim-2-Real (i.e., Steps 3, 4, and 5)
These steps are in your hands as the AI4R student engineers responsible for designing policies that convert our fleet from RC cars to autonomous cars! See Parts A and B below.

TASK DESCRIPTION - PART A - Getting to know your car

(Task A.1) Collect a few image from your car's camera while it is stationary.

- Instructions for using the image collection interface are given on the frame capture page.
- "a few image" means:
  - Different angles of the line in the image
  - Different curvature of the line.
  - Line at the edge of the image.
  - Line entering half-way up image.
  - Neighbouring line visible.
  - Intersection of lines.
  - Noisy image.
  - Different lighting conditions (ask teaching team to dim lights for some time).

(Task A.2) Detect the line within the stationary images by testing out a full range of parameters for the line detection class provided (instructions are given on line-detection testing page). Find and describe the limits of the line-detection algorithm, i.e.:

- Discuss which parameters and their range of values are most important to ensure usable line-detection on the majority of the images.
- Discuss the kinds of images that are at the limit of the line-detection algorithm provided.
- Discuss what challenges you faced during test and how you overcame them? (Where "how" refers to: adjustments to the line detection parameters; and/or conditions that you require of the operating environment; and/or other aspects that you changed.)

(Task A.3) Get certified for manual operation of the car, (details distributed during the workshop).

(Task A.4) Collect a stream of frames the camera while the car is making dynamic movement under your manual control of the car.

- The collect instruction are the same as Task A.1, except that you set the number of saved frames to be a higher value.
- For example, 50 images at 10 FPS frame-rate corresponds to nominally 5 seconds of data.
- Perform some turns of the camera during data collection in order to introduce noise/blur.

(Task A.5) Detect the line within the stream of images using the line detection class provided.

- Test out a range of parameters for detecting the line within the stream.
- Draw on your experience gained in Task A.2 for how the parameters influence the quality and reliability of line detection.
- Keep a record of the line-detection parameters and the resulting line-detection algorithm computation time for **3 (or more) choices** that span a range from fast computation time (< 30ms) to slow computation time (> 100ms).

(Task A.6) Benchmark the performance of line-detection on the image stream.

- A "ground truth" measurement is not available (which is a common challenge for low-cost consumer grade robotics!).
- Select line-detection parameters that you observe to detect the most accurate position of the line within the image (this can be a different set of parameters to anything you have previously considered, i.e., with a higher computation time).
- Use the benchmark interface (instructions are given on the benchmarking page) to compare the performance of the 3 (or more) choices of line-detection parameters from Task A.5.

(Task A.7) Perform real-time tests of your best choice(s) of line-detection algorithm on the live camera stream.

- Have the car stationary for these tests. (Progress to manual driving if time-permits).
- Keep a record of the line-detection computation-time. Compare to the Task A.5 results.
- Keep a record of overall frame-rate (which includes computation times for: image capture; line detection; and any other overheads).
- Perform a test for determining approximately how much buffer (in milliseconds) you need between the line detection computation time and the requested frame-rate of the camera.

(Task A.8) Explain and justify your final choice of line-detection parameters that you plan to use for control-loop control.

(Task A.9) Suggest other testing, data collection, investigation, and/or development you would need to do for verifying real-time line-detection algorithms for a specific application.

NOTE: download the saved images at end of session so that you can complete the processing in the next session (or for processing at home, instructions provided separately for installing the line detection class on your personal laptop).

MARKING GUIDE - for PART A

Your team of 2 is required to prepare and submit to LMS a "lab book" document that contains for the following:

- For (Task A.1): One example of an image collected.
- For (Task A.2): Briefly address the 3 discussion points.
- For (Task A.4): One example of a noisy/blurry image collected as part of the stream.
- For (Task A.5)
  - List or tabulate your 3 (or more) choices for line-detection parameters. Tabulate the computation time results for each choice.
  - One example of an image that shows the line-detection result for each of the 3 (or more) parameter choices.
- For (Task A.6):
  - List or tabulate the line-detection parameters you use as the benchmark, i.e., the parameters that give the most accurate position of the line within the image.
  - One example of an image that shows the line-detection result of using the benchmark line-detection parameters.
  - Tabulate the performance of the 3 (or more) parameter choices from Task A.5 against the benchmark.
- For (Task A.7):
  - Compare the real-time line-detection computation time results with those from Task A.5.
  - Describe the test you performed for determining the required computation buffer, and report the results.
- For (Task A.8): as stated, briefly explain and justify your final choice of line-detection parameters that you plan to use for control-loop control.
- For (Task A.9): as stated, briefly suggest other testing, data collection, investigation, and/or development you would need to do for verifying real-time line-detection algorithms for a specific application.

Submission details:

Type:	"Lab book" uploaded to LMS.
Length:	As short as needed to address the points above, please be concise and complete.
Weighting:	5% of the overall subject grade.
Due Date:	31 August 23:59 (Thursday of semester week 6). Any extension will be indicated by the due date on Canvas.
Instructions:	You must be physically present during your workshop session to conduct hands-on experiments with the cars.

TASK DESCRIPTION - PART B - Simulation and real-world implementation of policies

The overall goal of Part B is that you design through simulation 2 (or more) policies for the 1/10 scale car to follow a line and then you implement those policies on the real car. The focus for simulation is on fidelity and on empirically testing the robustness to stochastic influences. The focus for real-world implementation is observing similarities and differences between the simulated and real-world behaviour.
The tasks below are a guide for achieving this overall goal.

(Task B.1) Get familiar with the gymnasium simulation environment that is provided.

- A video tutorial is provided on LMS for giving you a "tour" of the simulation environment. The "tour" takes you through the following:
  - The Autonomous-line following gymnasium(-like) simulation environment that is provided as a **Python notebook** in the Sim-2-Real Part B assignment on LMS.
  - The Python notebook provided is self-contained in the sense that it includes all the relevant modelling equations that it implements and descriptions of the classes provided.
  - The `RoadEnv` class provided is for building a line to follow and for generating camera-based line-detection measurements.
    - You do **not** need to delve into the details of this class.
    - You **do need** to review the convention for a how a line is described and constructed.
  - The `BicycleModelKinematic` class provided is for simulating the time-evolution of a car.
    - You do **not** need to delve into the details of this class.
    - You **do need** to review the modeling section titled "Selecting and describing a kinematic bicycle model" in order to understand the convention, units, and meaning of each of the states and actions.
  - The `AutonomousLineFollowingGym` class is for combining the above two classes into a simulation environment.
    - You **do need** to delve into the details of this class.
    - You **do need** to adjust the `__init__` function of this class so that it is simulating the car and road that you want it to simulate.
    - You should **assume** that all the values hard-coded into the `__init__` function need to be adjusted and/or checked for correctness.
    - You should **not** need to edit the the `reset` nor the `step` functions.
    - You **do need** to understand the convention, units, and meaning of the feedback measurement that are returned by the `reset` and `step` functions, i.e., the variables `uv_coords` and `info_dict`.

(Task B.2) Specify baseline parameters for the car and the road

- Specify parameters for the 1/10 scale cars that we are working with for this subject. The parameters that need to be specified are:
  - The length of the car between the front and rear axles.
  - The maximum allowed steering angle request.
  - The maximum allowed car velocity request.
  - The maximum possible rate-of-change of steering angle.
  - The maximum possible rate-of-change of car velocity.
- As necessary, make simplifying assumption or perform simple tests to estimate some of the parameters.
- Construct a road that is representative for the 1/10 scale cars, i.e., the overall length and maximum curvature should match with the space we are operating in and curvature we can achieve with the tape we are using.

(Task B.3) Comparison of numerical integration techniques

- Setup your code to simulate the car for:
  - A fixed sequence of actions as a function of time, i.e., open-loop control.
  - Multiple numerical integration techniques (keeping all other simulation parameters unchanged).
  - Analyzing the difference between the trajectories computed by each numerical integration technique.
- Use your code to investigate how the difference in trajectories is influenced by the integration time-step and by the speed and/or aggressiveness of the open-loop sequence of actions.
  - As a starting point for your investigation, you can determine the "extremes" for the integration time-step.
  - Is there a time-step below which the integration techniques all compute essentially identical trajectories?
  - Is there a time-step above which the difference is trajectories is concerning?
- Decide on an integration time-step and numerical integration technique to use for the remainder of Part B.
  - In making your decision, you should give consideration to the computation time of each numerical integration technique.

(Task B.4) Specify camera parameters

- In order to simulate the real 1/10 car more accurately, we need to simulate the measurements that are generated by the line-detection algorithm, i.e., we need to simulate the details that you investigated in Part A.
- The line-detection algorithm computes the pixel location of the line within a camera image, hence, we need to transform body-frame Cartesian-coordinates of the line into pixel locations in the camera image.
- The `RoadEnv` class already implements this body-frame -to- camera-frame transformation, which requires you to specify the following parameters for simulating the specific camera on the 1/10 cars:
  - The translation from the camera-frame origin to the body frame origin (which is the front axle) expressed in coordinates of the body frame, i.e., $T^{(B)}_{(C)\to(B)}$.
  - The angle that the camera tilts downwards from the horizontal (this is 25 degrees for the baseline build of the 1/10 cars we are using).
  - The resolution of the camera (e.g., 1920x1080, 1280x720, 960x540, 640x480)
  - The matrix of intrinsic camera parameters.
- A full understanding of the matrix of intrinsic camera parameters is beyond the scope of this task (and this subject). A brief explanation and parameter values are provided at the end of this document.
- For each of these camera parameters, specify the uncertainty in the parameter value (this may come from uncertainty in how accurately you can measure the parameters, or it may come from the real car pitching and rolling as it drives).
- For one fixed car location relative to the road, generate a camera image (i.e., line-detection pixel coordinates) for many different camera parameter values within the ranges that you specified in the previous dot-point.
  - "Many" is pointing towards the industry lecture from Michael Crump where he talked about Monte Carlo simulations; hence "many" is at least 100 different samples, and potentially even 1,000 or 10,000 samples.
- Show, through either plots or tables, how the line-detection results are influenced by the parameter uncertainties.

(Task B.5) Simulation-based synthesis of feedback policies

- **Design two policies for the 1/10 car to autonomously follow the road-line that you specified in Task (B.2).**
- This task is now getting to the "core" of this subject, i.e., policy design and sythnesis.
- The previous tasks are quite prescriptive; this task is purposely less prescriptive.
- In designing your policies, you should:
  - Use only 1 numerical integration technique based on your results of Task (B.3).
  - Add noise to the line-detection pixel data based on Task (B.4) and based on your experiences in Part A.
  - Add any other noise or parameter uncertainty that you consider appropriate, for example, an offset in the steering angle.
  - Perform simulations that demonstrate the empirical robustness of the policy in the face of the uncertainties being considered, i.e., how does the policy performance vary across many simulations with different noise/uncertainty.
  - Consider whether the road you specified in Task (B.2) is challenging enough to "stress-test" your policies,
- The two policies you design should have either distinctly different behaviour, or be distinctly different algorithms, so that it is sufficiently meaningful to compare their performance in real-world implementation.
- As the real 1/10 car we are using does not have speed feedback, your simulation-based policy should set the velocity request to a fixed value and consider only the steering angle request action for achieving the goal of following the line.
- Any and all types of policies are valid for this task. The kinds of policies you can consider are:
  - Proportional control (e.g., the steering angle request is proportional to the lateral deviation of the car from the line.)
  - Rule-based control (e.g., a fixed steering angle request for different ranges of feedback measurements).
  - Pure pursuit controller.
  - Stanley controller.
  - MPC policy using a linear model derived by linearizing the kinematic bicycle model equations-of-motion.
- You need to specify 1 (or more) performance metric(s) that you use for assessing the performance of your policies.

(Task B.6) Real-world implementation

- Implement and test your two policies from Task (B.5) on the real 1/10 cars.
  - Details will be distributed separately for how to implement a policy on the real car.
- Compare the real-world performance to the simulation performance. Compare both the:
  - Performance of the same policy between simulation and reality.
  - The relative performance of the policies in simulation, versus the relative performance of the policies in reality.
- The setup with the real 1/10 cars does not have any ground truth feedback of the position nor heading angle of the car. Hence, the only data you have for assessing performance is the line detection data, and your observations of the car. Hence, you may need to get a bit creative with a performance metric for the real car.

MARKING GUIDE - for PART B

Your team of 2 is required to prepare and submit to LMS a "lab book" document that contains for the following:

- **For (Task B.1)** Get familiar with the gymnasium [**0%**]
  - Nothing needs to be included for this task.
- **For (Task B.2)** Specify baseline parameters for the car and the road [**1%**]
  - State the parameter values that you use for the 1/10 car model, with 1-3 sentences for each parameter explaining how you arrived at that value.
  - Include one plot of the road you constructed.
- **For (Task B.3):** Comparison of numerical integration techniques [**2%**]
  - Figures, descriptions, and discussions of the analysis you performed and the insights you gained for comparing numerical integration techniques.
  - State and explain your decision for the integration time-step and numerical integration technique that you use for the remaining tasks.
- **For (Task B.4):** Specify camera parameters [**1%**]
  - State the parameter values that you use for simulating the camera, and state the uncertainty distribution that you choose for each parameter. Include 1-3 sentences for each parameters explaining how you arrived at the value and uncertainty distribution.
  - Figures, descriptions, and discussions of the analysis you performed and the results for how line-detection is influenced by uncertainties in the camera parameters.
- **For (Task B.5):** Simulation-based design of feedback policies [**4%**]
  - Describe the policy types you use.
    - State clearly what are the parameters of the policy for which you need to synthesize parameter values (e.g.: proportional gain parameter; look-ahead distance parameter).
    - Describe clearly how the line-detection measurements are used in the policy, including any parameters for converting the pixel-measurements to a feedback-signal for the policy.
  - Explain how you synthesized the parameter value(s) for your policies, including figures and derivations as necessary to support your explanations.
  - Describe the performance metrics you use for assessing the performance of your policies.
  - Figures and discussion of the performance of your policies.
- **For (Task B.6):** Real-world implementation [**1%**]
  - Describe and discuss your observation and experience of transferring your policies from simulation to reality.
  - If applicable, provide any performance metrics that you were able to obtain for the performance of the real car.

Submission details:

Type:	"Lab book" **pdf** uploaded to LMS.
Length:	As short as needed to address the points above, please be concise and complete.
Weighting:	9% of the overall subject grade.
Due Date:	18 September 12:00 (Monday of semester week 9). Any extension will be indicated by the due date on LMS.
Instructions:	You must be physically present during your workshop session to conduct hands-on experiments with the cars.

ADDITIONAL INFO - INTRINSIC CAMERA PARAMETERS

The matrix of intrinsic camera parameters is typically written as follows:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where:

- f_x : is the focal length of the camera lens (units: m) multiplied by the pixel density (units: pixels/m) in the x-direction of the camera frame, hence, overall this is the focal length in units of pixels in the x-direction.
- f_y : is the same as f_x but for the y-direction of the camera frame, hence, overall this is the focal length in units of pixels in the y-direction.
- c_x : is the camera-frame x-direction coordinate of the optical center of the camera lens (units: pixels), relative to the origin of the camera's pixel array.
- c_y : is the same as c_x but for the camera-frame y-direction coordinate.

For cameras that are the same make and model, these 4 intrinsic parameters vary slightly from camera-to-camera due to manufacturing tolerances. Measuring the parameters directly is not practical for an off-the-shelf camera product, hence a calibration procedure is used to estimate the intrinsic parameters. If you are interested, you can read more about the intrinsic parameters and calibration procedure here.

We performed the calibration procedure for 10 of the Logitech StreamCam cameras, with the results tabulated below. For each parameter, we provide a standard deviation for uncertainty in the parameter estimate that arises from uncertainties in the calibration procedure.

Intrinsic camera parameters of "StreamCam" for resolution 1920x1080

Cam #	f_x		f_y		c_x		c_y	
	mean	std.	mean	std.	mean	std.	mean	std.
1	1430.7	0.52	1426.6	0.47	980.9	0.80	517.7	1.25
2	1413.4	0.58	1409.8	0.51	965.3	0.80	535.2	0.90
3	1416.2	0.59	1412.9	0.50	959.7	0.70	568.5	0.81
4	1426.8	0.51	1422.8	0.39	971.7	0.84	536.4	1.65
5	1423.2	0.40	1419.5	0.43	950.5	0.52	518.6	0.92
6	1416.0	0.40	1411.0	0.39	945.2	0.90	531.0	2.08
7	1428.0	0.40	1424.4	0.41	932.6	0.33	542.3	1.00
8	1419.3	0.83	1415.3	0.82	981.9	0.83	533.4	1.63
9	1418.6	0.37	1414.4	0.38	960.8	0.74	521.7	1.74
10	1419.8	0.43	1414.8	0.32	958.9	0.38	534.1	0.94

Data for other resolutions continued on the next page.

Intrinsic camera parameters of "StreamCam" for resolution 1280x720

Cam #	f_x		f_y		c_x		c_y	
	mean	std.	mean	std.	mean	std.	mean	std.
1	951.6	0.38	948.7	0.33	654.9	0.54	342.4	0.77
2	946.3	0.26	943.7	0.37	640.5	0.56	354.7	0.71
3	942.0	0.39	939.2	0.39	641.3	0.38	369.4	0.77
4	949.7	0.95	947.5	1.03	646.2	0.59	352.4	1.15
5	950.9	0.14	948.7	0.12	636.0	0.25	350.1	0.85
6	944.8	0.23	942.2	0.29	634.3	0.34	352.4	0.79
7	949.2	0.38	947.0	0.38	622.4	0.34	364.8	0.45
8	950.6	0.39	947.2	0.36	654.0	0.42	351.1	0.55
9	942.7	0.35	940.3	0.31	635.8	0.47	347.8	0.72
10	946.4	0.33	943.5	0.27	640.5	0.18	350.5	0.67

Intrinsic camera parameters of "StreamCam" for resolution 640x480

Cam #	f_x		f_y		c_x		c_y	
	mean	std.	mean	std.	mean	std.	mean	std.
1	632.7	0.27	631.0	0.27	329.7	0.52	230.5	0.63
2	628.7	0.54	626.2	0.57	320.5	0.59	223.7	0.50
3	635.4	0.56	633.4	0.54	319.2	0.33	244.3	1.27
4	636.1	0.66	634.7	0.64	326.1	0.40	238.0	0.64
5	634.7	0.20	633.2	0.17	317.3	0.17	232.4	0.50
6	630.6	0.19	628.6	0.16	313.8	0.25	229.0	1.03
7	633.4	0.42	631.4	0.40	309.6	0.43	236.5	0.65
8	632.2	0.28	630.0	0.25	329.0	0.33	234.7	0.90
9	628.7	0.20	627.0	0.16	318.9	0.32	229.3	0.31
10	631.9	0.20	629.4	0.20	320.6	0.23	237.9	0.35