Designed and engineered, artfully. Silicon to software.

MSc Computer & Embedded Systems Engineering, TU Delft (2026–2028) · BSc Computer Science, Manchester (2022–2026)

INDEX / ALL PROJECTS

01	Quant Trading System (Automaton-QTS)	QUANT
02	Cellular-Automaton SoC Accelerator	QUANT
03	Aero-Lab — Quadrotor Flight Sim	DRONES
04	RRHC vs MPCC: A Reactive Racing Controller That Beats Online Optimisation	MOTORSPORT
05	Stump — 16-bit RISC CPU closed on Spartan-6 silicon	QUANT
06	Multicore Computing Labs	QUANT
07	ONI — Local-LLM Agent Harness	OTHER
08	ARMED — Git for Ableton Sessions	MUSIC
09	FSAI Driverless Racing Simulator	MOTORSPORT
10	velox — CommonRoad C++ Port	MOTORSPORT
11	Slit Light Interference Simulator	OTHER
12	CarrLane — Engineering Part Catalog Chatbot	OTHER
13	AUBRY — Autonomous Cinematography Drone	DRONES
14	Proce-Tree — Procedural Tree Generator	OTHER
15	FSAI Unity Driving Simulator	MOTORSPORT

FIELD / QUANTITATIVE FINANCE

01 / SYSTEMS · CONTROL

Quant Trading System (Automaton-QTS)

A multi-signal trading engine, an agent-based market simulator, and a relation-typed contagion-propagation graph that I proved is dead in equities and alive in crypto.

Python | PyTorch | torchdiffeq | NautilusTrader | hftbacktest | scipy

CODE	TESTS	CRYPTO CONTAGION (RESEARCH-STAGE, N=3)	TERRA LINKED-PEER SIGNAL (RESEARCH-STAGE, N=3)
45.7k LOC Python	**1,314 tests, 129 files**	**+9.2%/event**	**p=0.0009 @72h**
LIVE-PAPER PHASE-1 GATE **built, never run**			

ROLE

Sole author. Built the signal/risk/execution core, the human-gated LLM oversight trail, an agent-based market simulator for synthetic data, and the relation-typed propagation-graph research line (equity negative + crypto positive).

REPO / https://github.com/RandevRanjit/Automaton-QTS

02 / HARDWARE

Cellular-Automaton SoC Accelerator

A from-scratch SystemVerilog drawing unit that renders Wolfram cellular automata — programmable rule as a pure lookup, LFSR-seeded, taken through Vivado to an Artix-7 and demoed live on a monitor.

SystemVerilog | Vivado | Artix-7 | RISC-V assembly | Questa

ACCELERATOR (BLOCK)	WHOLE SOC FILL	RULE SPACE	FUNCTIONAL COVERAGE
4,433 LUT HANDSHAKE SVA **5 asserts + 1 cover**	**96.0% BRAM**	**256 rules**	**100% stmt/ branch/cond/ FSM**

ROLE

Designed unit_automaton from scratch (programmable rule-as-lookup, 32-bit LFSR seed, 3-state compute/ write FSM, byte-lane framestore writes with req/ack backpressure), wrote a coverage-driven testbench to 100% statement/branch/condition/FSM coverage, integrated it into the drawing engine replacing a dummy slot, drove it from a RISC-V MMIO program, and closed it through Vivado synthesis + implementation on Artix-7 with a live monitor demo.

PRIVATE · CASE STUDY (NO CODE LINK)

---

05 / HARDWARE

Stump — 16-bit RISC CPU closed on Spartan-6 silicon

A multi-cycle 16-bit RISC processor built bottom-up from RTL primitives in Verilog, then synthesised, placed-and-routed, and timing-closed on a real Spartan-6 at 98% slice occupancy — the first design I carried through the full FPGA back-end flow.

`Verilog-2001` `Xilinx ISE` `Spartan-6 (xc6slx4)` `ModelSim` `Stump assembly`

SLICE LUTS	OCCUPIED SLICES	SLICE REGISTERS	TIMING SCORE
2,068 / 2,400 (86%)	**591 / 600 (98%)**	**1,052 / 4,800 (21%)**	**0 (timing met)**

ROLE

Authored the datapath (ALU, barrel shifter, sign-extender, 8-register file with R0=0 / R7=PC, operand muxes), the 3-state control FSM (FETCH/EXECUTE/MEMORY) and the control-decode logic, and implemented the LDST and BCC instruction paths. Top-level board wrapper and 1.5k-line memory model were provided. Wrote test programs in Stump assembly including a Battleship game driving a memory-mapped 8×8 LED matrix.

PRIVATE · CASE STUDY (NO CODE LINK)

---

06 / SYSTEMS

Multicore Computing Labs

Three parallelism labs on a 72-core 2-socket NUMA machine — a 132× temporal-blocking Jacobi stencil, a 30.7× NUMA-aware vecadd, and dining philosophers scaling near-linearly to 1.04M meals/s.

C | C++20 | pthreads | OpenMP | std::barrier | NUMA

TEMPORAL-BLOCKING STENCIL **132.5× @ 72 cores** SYNC REDUCTION (STENCIL) **1 barrier / 128 iters**	OPENMP STENCIL **128.6× @ 72 cores**	NUMA VECADD SPEEDUP **30.7× @ 72 threads**	FINE-SPINLOCK PHILOSOPHERS **1,039K meals/s**

ROLE

Sole author (s94810rr), three labs. Implemented NUMA first-touch pthreads vecadd; four-variant dining philosophers (coarse/fine × mutex/spinlock) with lock-ordering deadlock prevention; and a temporal-blocking 1-D Poisson stencil in both std::thread+std::barrier and OpenMP. Benchmarked on mcore72 (2× Xeon Platinum 8452Y, 72 cores, 2 NUMA nodes) through the OGE batch queue.

PRIVATE · CASE STUDY (NO CODE LINK)

FIELD / DRONES

03 / CONTROL

Aero-Lab — Quadrotor Flight Sim

Hand-rolled 6-DOF quadrotor dynamics plus four racing controllers — PID, MPCC (acados SQP-RTI), a solver-free RRHC, and a PPO policy — compared under a machine-enforced fairness contract, with the RRHC now re-ported to deterministic bare-metal C as declared prior work for a TU Delft MSc.

Python | NumPy | SciPy | acados | CasADi | Gymnasium

TESTS **381 tests** CODE **~9.0k src LOC**	CONTROLLERS **8 registered**	RRHC VS MPCC **~22x less compute**	C PORT VS PYTHON **max |Δω| 1.4e-12 rad/s**

ROLE

Sole author. Hand-rolled the 6-DOF rigid-body dynamics (RK4, motor lag, gyroscopic torque, drag), the analytic differential-flatness inner loop shared by every controller, an acados SQP-RTI MPCC, a CTBR PPO env

+ policy, and a fairness contract (pre-flight + watchdog + post-hoc audit) that enforces identical plant, course, and control rate across all controllers. Since July 2026 also the offline table-bake and bare-metal C reimplementation of the RRHC, validated bit-close against the Python reference.

PRIVATE REPOSITORY

13 / CONTROL

AUBRY — Autonomous Cinematography Drone

A camera drone that trails you, leads you and frames the shot itself, answers with a nod or a shake, and decides between those behaviours on its own — four controllers with potential-field obstacle avoidance and a decoupled virtual gimbal, all riding the aero-lab racing stack's inner loop untouched.

Python | NumPy | SciPy | VisPy

CONTROLLERS	SOURCE	TESTS	OBSTACLE AVOIDANCE
4 cinematic	**981 LOC**	**56 AUBRY tests**	**APF + 4-pass detour**
INNER LOOP **shared, RRHC untouched**			

ROLE

Sole author. Designed four cinematic controllers on top of the existing quadrotor stack — a trailing controller that keeps a target arc-length gap behind a moving subject via a rolling breadcrumb spline with iterative obstacle-detour resampling; a framing controller that leads the subject in a relative-pose "front selfie" with artificial-potential-field repulsion and a decoupled virtual gimbal computed in the runner layer; a gesture controller that perturbs a hover reference into nod/shake/bob/pirouette primitives; and a companion arbiter that picks between come, watch, follow and hover from scripted stimuli, gluing a gesture onto each transition. All four reuse the shared differential-flatness inner loop without touching RRHC.

PRIVATE REPOSITORY

FIELD / MOTORSPORT

04 / CONTROL

RRHC vs MPCC: A Reactive Racing Controller That Beats Online Optimisation

Final-year dissertation, submitted April 2026 — a model-free heuristic racing controller benchmarked against an IPOPT-solved MPCC under an enforced fairness contract, and shipped with a 13k-word report and a rendered screencast of the argument.

Python | NumPy | SciPy | CasADi | IPOPT | Optuna

LAP TIME VS MPCC	PER-STEP COMPUTE	DEADLINE OVERRUNS (ZOH)	CROSS-PHASE SIGNIFICANCE
12.2% faster	**153x cheaper**	**0 (vs 1,870 MPCC)**	**Wilcoxon W=253, p=1.9e-5**

ROLE

Sole author (dissertation). Designed and built the RRHC controller, the MPCC/IPOPT baseline, the 3-DOF Fiala-tyre plant, the fairness-contract harness, the five-phase statistical evaluation, and the animated screencast that presents the result.

PRIVATE REPOSITORY

---

09 / SYSTEMS · CONTROL

FSAI Driverless Racing Simulator

Architected and authored the simulation engine (~29k of 34k LOC C++) — RK4 adaptive sub-stepping over three CommonRoad models (7/9/29 state), an OpenGL stereo camera with PBO readback, and a software CAN bus to a separate VCU process. On an 11-person Formula Student AI team I wrote the physics, IO, CAN and infra; perception by teammates.

C++17 | CMake | Eigen | OpenGL | SDL2 | ONNXRuntime

CODEBASE	VEHICLE DYNAMICS	STEREO CAMERA	REAL-TIME BUDGETS
~29k LOC (of 34k)	**RK4, 3 models (7–29 state)**	**PBO readback + ring buffer**	**ns clock, 4 subsystems**

ROLE

Architect and sole author of the simulation engine (~29k of 34k LOC). Designed and wrote the velox physics core (RK4 + adaptive sub-stepping, all three CommonRoad vehicle models), the sim loop, the OpenGL FBO stereo camera with PBO readback, the AI-to-VCU CAN interface and UDP S-VCU link, the ns-budget timing + swappable-clock infrastructure, and all common/IO/control libraries. 11-person team project; the ONNX/SIFT/Kalman perception pipeline was teammates' work — it plugs into the camera and pose feeds I built.

REPO / https://github.com/RandevRanjit/FSAI-Simulation-C

---

10 / SYSTEMS · CONTROL

velox — CommonRoad C++ Port

~10.6k LOC C++20 port of the CommonRoad vehicle models (ST / STD / MB) into a reusable static library, `velox`, behind one daemon-style API — derivative outputs matched against the Python reference to 1e-13, plus a ~4.5k LOC TypeScript SDK whose JS backend is parity-checked field-by-field against the native one.

C++20 | CMake | TypeScript | Python | YAML | SDL2/ImGui

CODEBASE	SELECTABLE MODELS	DERIVATIVE PARITY VS PYTHON	TEST SUITE
~10.6k LOC C++	ST / STD / MB (29-state)	ST/STD 1e-13, MB 1e-7	19 CTest targets

ROLE

Sole author. Ported the CommonRoad ST/STD/MB vehicle models from the academic Python reference to a C++20 static library (`velox`), built the `SimulationDaemon` API with a `ModelTiming` sub-step scheduler, Pacejka tyre model, EV powertrain controller, staged low-speed/loss-of-control safety, and a TypeScript web SDK with a JS backend parity-checked against the native one.

REPO / https://github.com/RandevRanjit/CommonRoad-CXX-Port

---

15 / CONTROL · GRAPHICS

FSAI Unity Driving Simulator

An earlier Unity/C# Formula Student driving sim — procedurally-generated cone circuits via complex Fourier synthesis, rescaled until the tightest corner clears a minimum radius; a dynamic bicycle model (Pacejka tyre, kinematic blend through standstill) driven autonomously by a dual-lookahead path-follower with live telemetry.

C# | Unity 6 | System.Numerics

TYRE MODEL	TRACK GENERATION	LOW-SPEED MODEL	C# ENGINE
Pacejka magic formula	complex Fourier synthesis	kinematic blend ≤3.5 m/s	~2.6k LOC

ROLE

Sole author (Formula Student). Wrote the dynamic bicycle vehicle model (Pacejka tyre, aero down/drag-force, low-speed kinematic correction), the procedural cone-track generator (a complex-Fourier path with curvature-derived corner radii and a minimum-radius rescale), the autonomous lookahead path-following controller, and the real-time telemetry + CSV logging — all C# on Unity 6 (6000.0.22f1).

PRIVATE REPOSITORY

FIELD / MUSIC

08 / SYSTEMS

ARMED — Git for Ableton Sessions

A macOS menubar app over a Rust daemon that gives Ableton Live real version control — diffing the music, not the bytes. Semantic .als diff, per-song variations that never touch HEAD, and a byte-splice merge engine that edits the project XML in place. Signed, notarised and universal at 0.2.8 — and deliberately not published yet.

Rust | tokio | libgit2 (git2-rs) | roxmltree | flate2 | SHA-256

CODEBASE **27.5k Rust · 11.9k Swift**	.ALS → TYPED AST **1,443 LOC parser**	SEMANTIC DIFF **22 delta variants**	BYTE-SPLICE MERGE **6,017 LOC engine**
PER-FILE BRANCH COMMIT **HEAD/index untouched**	TESTS **257 daemon tests**	MERGE FUZZ SOAK **2,160 cases, 0 CRITICAL**	BUILD STATE **0.2.8, unpublished**

ROLE

Sole author. Built the whole system end to end — the Rust daemon (FS watcher, async/sync IPC boundary, git object layer, per-song ref namespaces, project discovery, launchd integration), the 1,443-LOC Ableton gzip-XML parser, the semantic differ, the 6k-LOC byte-splice merge engine and its corpus fuzzer, the clap CLI, the SwiftUI menubar app and standalone window, the signed/notarised release pipeline, and the static marketing site.

PRIVATE REPOSITORY

FIELD / OTHER

07 / SYSTEMS

ONI — Local-LLM Agent Harness

102k-LOC single-crate Rust harness driving a local llama.cpp / MLX server through a 38-tool fenced-code protocol, a graph working-memory, a tree-sitter code graph and an in-loop LSP client — 2,656 tests, µs/ns Criterion latency contracts, and a scored experiment behind every feature that shipped.

Rust | tokio | reqwest | tree-sitter | tantivy | criterion

CODEBASE	TESTS	TOOL SURFACE	PARSER BUDGET
102.1k LOC Rust	**2,656 tests**	**38 fenced tools**	**<100 µs**
TOOL DISPATCH BUDGET	LOOP-DETECTOR BUDGET	BACKENDS	BEST SCORED EVAL RUN
<5 ms/call	**<50 ns/KB**	**3 backends**	**9/9 on L1, ×9**

ROLE

Sole author. Designed and built the whole harness: the OMTF v0.4 fenced-code tool protocol and the parser that normalises three native XML dialects into it, the backend seam over three inference servers, the graph working-memory with BFS gather and harness-managed compaction, a tree-sitter code knowledge-graph, an in-loop LSP diagnostics client, the LIT/SIT stream-loop detectors, the two-role delegate orchestrator, and the process-group SIGKILL sandbox. Every behaviour-changing feature is greenlit, rollback-tracked, and either validated or killed by a scored run.

PRIVATE REPOSITORY

---

11 / GRAPHICS

Slit Light Interference Simulator

A 3D single-slit Fraunhofer diffraction lab — the sinc² intensity envelope solved per-fragment on the GPU, wrapped in a Three.js scene with draggable apparatus, live wavelength/slit/distance controls and a real-time measurement readout.

JavaScript | Three.js | GLSL | Tweakpane | Webpack

DIFFRACTION MODEL	SHADER PROGRAMS	ENGINE	WAVELENGTH→COLOUR
sinc² Fraunhofer, GPU	**3 programs, 6 files**	**~1.65k LOC**	**380–780 nm CIE→RGB**

ROLE

Sole author. Wrote the interference vertex/fragment shader pair (sinc² envelope, SI-unit parametrisation), the wavelength→RGB conversion, the full Three.js scene graph, the draggable transform-controlled apparatus, the live measurement panel, and the particle-haze system with its sorted exponential-search culling.

REPO / https://github.com/RandevRanjit/Slit-Light-Interference-Sim

---

12 / SYSTEMS

CarrLane — Engineering Part Catalog Chatbot

Three-tier function-calling assistant that turns natural-language part queries into typed API calls over a scraped engineering catalog — built during an industry internship.

Python | FastAPI | SQLModel | Node/Express | React | Chakra UI

CATALOG (PROTOTYPE)	COMPANY CATALOG	LLM TOOLS	API TESTS
314 parts	**10,000+ parts**	**9 typed function schemas**	**8 async endpoint tests**
DATES **Jun–Aug 2025, Chennai**			

ROLE

AI / Software Engineering Intern. Built the catalog ingestion, the typed REST API over it, and the function-calling orchestration layer that maps natural-language queries to catalog lookups.

PRIVATE REPOSITORY

---

14 / GRAPHICS

Proce-Tree — Procedural Tree Generator

A real-time procedural tree grown from a resource-flow model — branches that steer away from their own crowded canopy, cross-sectional area conserved at every fork, depth-decaying splits, and camera-facing instanced leaves, in Three.js.

JavaScript | Three.js | Vite

GROWTH MODEL	BRANCH STEERING	LEAVES	ENGINE
resource-flow + area	**canopy-density avoidance**	**instanced billboards**	**~520 LOC**

ROLE

Sole author. Designed the growth algorithm — a binary-splitting branch model fed by a "growth resource" that is consumed in proportion to cross-sectional area and conserved across each fork, with branches that compute a weighted-average canopy position and grow *away* from it — plus the Three.js scene, the per-frame branch geometry, and the camera-facing instanced leaf billboards.

REPO / https://github.com/RandevRanjit/Proce-Tree