R&D engineer · robotics, control and data-driven modelling · Munich

Anas Makdesi

Models, estimators and controllers you can trust.

Modelling, estimation, optimization and control for robots and machines, from first principles to tested software. PhD in automatic control (CentraleSupélec), postdoc at LMU Munich, and earlier a robotics team lead in industry.

PROJECT PORTFOLIO

01	ddma: data-driven models and abstractions	p. 2
02	Safety filter for learning-based control	p. 3
03	Real-time MPC: a car, a quadrotor and a robot arm	p. 4
04	Safe learning-based MPC: learn while you drive, never leave the safe region	p. 5
05	Safe operating envelopes straight from data	p. 6
06	Surveying robot: survey-grade positioning on a construction site	p. 7
07	3-DOF helicopter rig: from circuit board to closed-loop control	p. 8
08	Sewer inspection robot: electronics for a robot that works inside pipes	p. 9

WEBSITE	anas-makdesi.github.io
EMAIL	anas.makdissi@gmail.com
GITHUB	github.com/anas-makdesi
LINKEDIN	www.linkedin.com/in/anas-makdissi
GOOGLE SCHOLAR	scholar.google.com/citations
LOCATION	Munich, Germany

Each project has its own page on the website, with its code on GitHub where it is open source: links and QR codes on every page.

ddma: data-driven models and abstractions

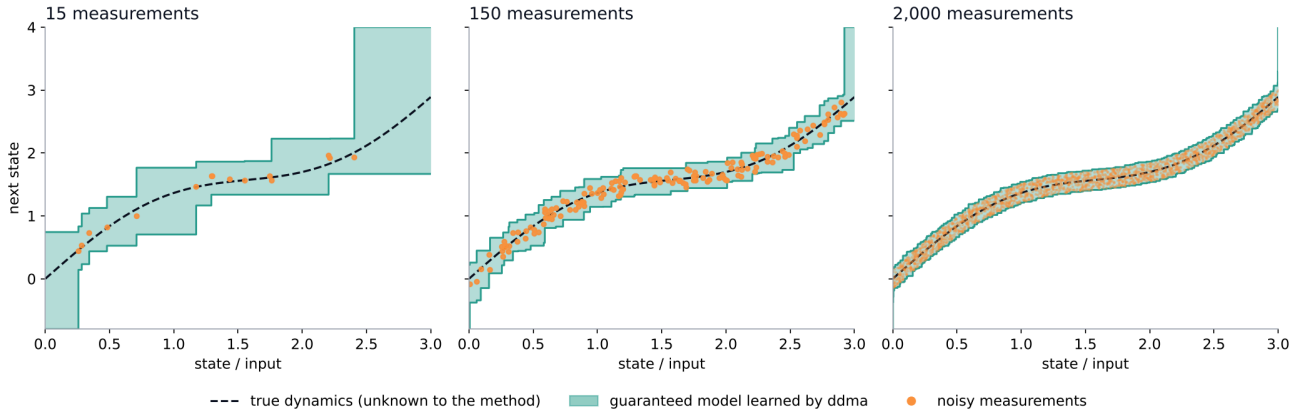
Safe controllers learned from data. An open-source library that turns logged sensor data into a model with guaranteed error bounds, for monotone systems and for any system with bounded derivatives, then into a safety filter that keeps any controller, including a learned one, inside its limits. Python and MATLAB on one C++ core.

10^8
measurements processed in one run

0.3 s
to learn a model from 10 million measurements

0
safety violations in every closed-loop test

490+
automated tests, Python and MATLAB checked against each other



The guaranteed model tightens around the true dynamics as measurements come in

HOW IT WORKS

1
Logged data
Measured transitions (state, input → next state), noisy and unevenly spread.

2
Model with error bounds
The tightest set of possible behaviours consistent with the data. The real system is guaranteed to be inside it.

3
Safe operating region
The largest set of states from which the limits can always be kept, whatever the disturbance does.

4
Safety filter
Wraps any controller (PID, MPC, an RL policy). It passes commands through and corrects only those that would lead out of the safe region.

STACK C++17 · OpenMP · Python · NumPy · SciPy / HiGHS · pybind11 · MATLAB / MEX · Octave · pytest · GitHub Actions · matplotlib

PROJECT PAGE
anas-makdesi.github.io/projects/abstraction-toolbox

CODE
github.com/anas-makdesi/ddma

Safety filter for learning-based control

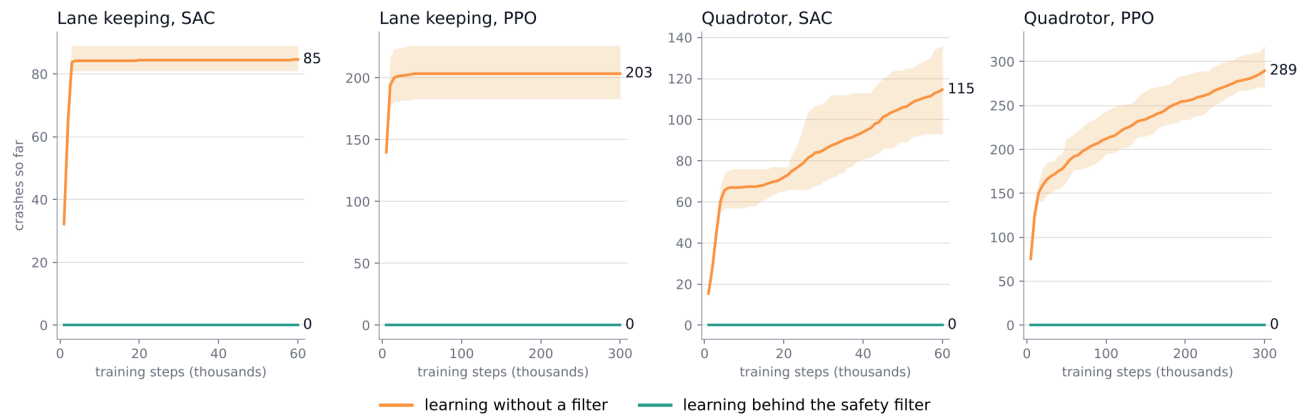
A learning controller that cannot crash, even while it learns. A filter learned from data sits between the controller (reinforcement learning, DeePC or MPC) and the machine, passes every safe command and corrects only the unsafe ones. On a lane-keeping car and a quadrotor in a corridor: 81–318 crashes per training run without it, none with it.

0
crashes in 40 training runs behind the filter (81–318 per run without it)

0
crashes in 480 test episodes with the filter, for every controller

0.1 %
performance cost for controllers that are already safe

0.1–0.2 ms
per filter call



Crashes while learning, with and without the safety filter: two benchmarks, two learning algorithms, five runs each

HOW IT WORKS

1
Logged data
Sampled transitions of the machine (state, command → next state). The equations are never given.

2
Certified safe set
With my toolbox ddma: a model with guaranteed error bounds, then the states from which the limits can be kept forever, and the commands that do it.

3
Filter in the loop
At every step: a safe command passes unchanged, an unsafe one is replaced by the nearest safe command.

4
Any controller on top
Reinforcement learning (SAC, PPO), DeePC, MPC. The controller is in charge of performance, the filter of safety.

STACK Python · PyTorch · Stable-Baselines3 · Gymnasium · CVXPY · OSQP · ddma · pytest · GitHub Actions · matplotlib

PROJECT PAGE
anas-makdesi.github.io/projects/safety-filter

CODE
github.com/anas-makdesi/safety-filter

Real-time MPC: a car, a quadrotor and a robot arm

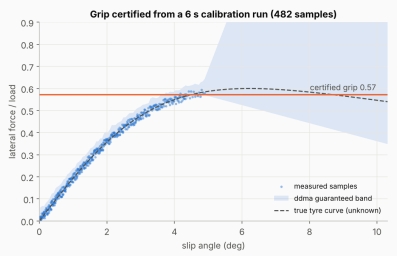
Nonlinear model predictive control that runs in milliseconds, on three platforms: a car at the handling limit, a quadrotor in aggressive flight and a 7-joint robot arm. Each controller beats the one used in practice, is timed step by step and runs as a C++ ROS 2 node. On each, a limit learned from data with ddma (tyre grip, rotor thrust, joint friction) keeps the plan within what the hardware can give.

0.4–3.5 ms
median solve per control step, on the full nonlinear model

3 platforms
a car, a quadrotor and a 7-joint arm, each against the controller used in practice

0
spins or crashes when the hardware degrades, planning with limits certified from seconds of data

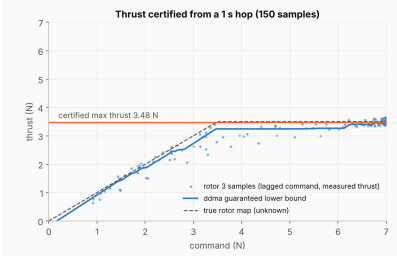
10^{-6}
agreement between each C++ ROS 2 controller and its Python version



Car at the handling limit

Drives close to the tyres' limit where linear MPC spins; on a slippery road it plans with a grip level certified from six seconds of driving.

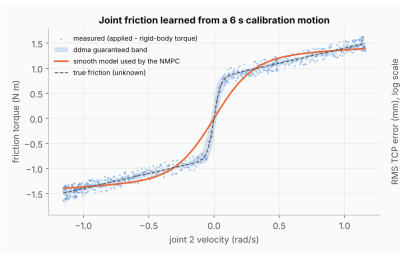
≈ 1 ms solve · 9 % faster lap · 0 spins on a slippery road



Quadrotor in aggressive flight

Commands the four rotor thrusts directly through a figure-8 at up to 12 m/s; with a damaged propeller it flies on thrust limits certified from one second of data.

≈ 0.4 ms solve · 2.6 cm at 12 m/s · flies with a damaged rotor



7-joint robot arm

Commands a Franka Emika Panda's joint torques around a fast circle within every joint's limits, with a friction model learned with a certified bound.

≈ 3.5 ms solve · 0.04 mm at 1.35 m/s · 1 kHz torque loop

HOW IT WORKS

1
Model
The platform's nonlinear model in CasADi, with the limits that matter on the real machine: grip, rotor thrust, joint torques and speeds.

2
Optimise in real time
acados, multiple shooting, one SQP iteration per control step, warm-started from the last plan; checked against a solve to convergence.

3
Certify the hardware
ddma turns seconds of measured data into a limit that is guaranteed to hold; the plan never asks for more. Set at run time, without rebuilding the solver.

4
Deploy
Generated C code inside a ROS 2 Jazzy C++ node, next to a simulator node running the machine at 1 kHz.

STACK acados · CasADi · OSQP · IPOPT · Pinocchio (tests) · Python · C++ · ROS 2 Jazzy · ddma · pytest · GitHub Actions · Docker

PROJECT PAGE
anas-makdesi.github.io/projects/realtime-mpc

Safe learning-based MPC: learn while you drive, never leave the safe region

A predictive controller that gets its performance from a model learned from data and its safety from a second, guaranteed model built from the same data. Both keep learning during operation. A robot trained only below 1 m/s learns to lap at 1.9 m/s without ever leaving its certified safe region; without the certificate, the same controller crashed in one of five runs.

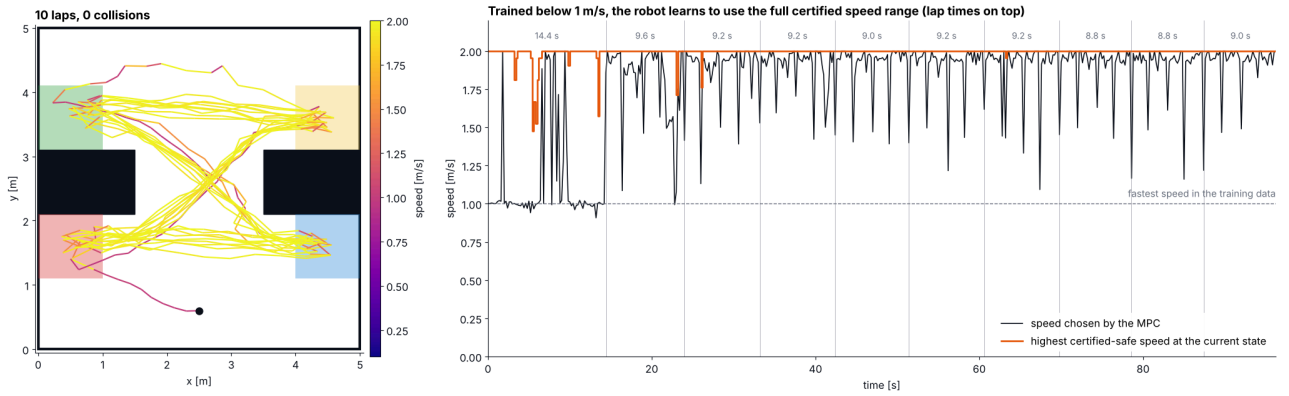
0
collisions and 0 steps outside the certified safe set, in 50 laps (5 runs)

1.1 → 1.9 m/s
mean lap speed, after training only below 1 m/s

14.4 → 9 s
lap time, from the first lap to the later ones

< 0.1 s
to compute the safe set over 100,000 states and 1,600 inputs

~1.5 ms
to learn from each new measurement, both models included



Ten laps of online learning: the trajectories coloured by speed, and the speed over time against the highest speed certified safe

HOW IT WORKS

1
Logged data
Measured transitions (state, input → next state). One long recording is enough; independent samples are not needed.

2
Guaranteed model → safe actions
A range the true next state is proven to fall in. From it, for every state, the commands that keep the system inside its limits forever, whatever the disturbance does.

3
Learned model
A smooth best-fit model of the dynamics, constrained to stay inside the guaranteed one. This is what makes the scheme work.

4
Predictive controller
MPC plans with the learned model for performance, but may only pick from the safe actions. Always feasible, always safe.

STACK Python · NumPy · SciPy · numba · ddma · pytest · GitHub Actions · matplotlib

PROJECT PAGE
anas-makdesi.github.io/projects/safe-learning-mpc

CODE
github.com/anas-makdesi/compatible-mpc

Safe operating envelopes straight from data

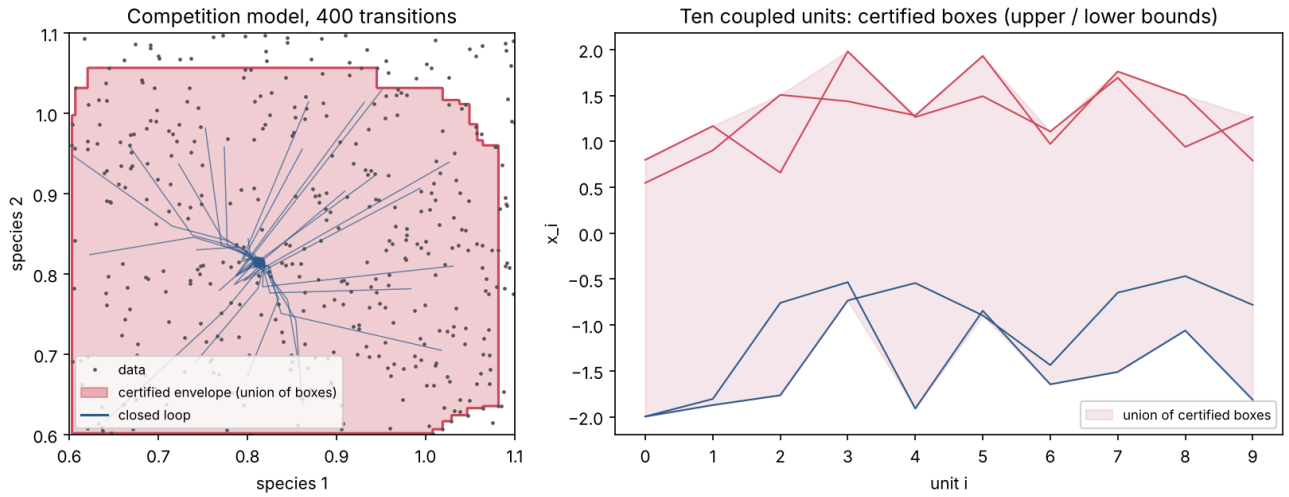
Give it logged measurements of a system you have no model of, and it returns the region where the system can be kept safe forever, together with the commands that keep it there. No model, no grid, and a guarantee that is a proof, not a probability. From 1,000 measurements of a three-room cooling system it certifies 98 % of the safe temperature range in under a second; a grid-based method needs 200,000 to certify 74 %.

1,000 → 98 %
measurements in, share of the safe temperature range certified (three rooms, one cooling input)

200× less data
than a grid abstraction of the same system, which certifies 74 % from 200,000

< 1 s
per envelope in every example, up to ten dimensions and 500,000 pairs of measurements

0
violations in closed loop, under random worst-case disturbances



Left: the certified envelope of a two-species competition model from 400 measurements, with closed-loop trajectories under a controller that harvests as much as the envelope allows. Right: the certified boxes of a ring of ten coupled nonlinear units, from 1,000 measurements

HOW IT WORKS

1
Logged data
Measured transitions (state, command → next state), from normal operation or a simulator. Plus simple prior knowledge: how the variables push on each other, or bounds on how fast the dynamics can change.

2
Order structure
Many systems are *monotone*: more of one variable never means less of another. Systems that are not get *lifted* into a monotone system twice the size that tracks a lower and an upper bound.

3
Envelope search
Find the measurements (or pairs of them) whose box of states provably maps back inside itself, and combine them. Only comparisons and small LPs: no gridding of the state space.

4
Safety controller
For every state in the envelope, the commands that keep it there. Use it as a safety filter around any controller, learned or hand-tuned.

STACK Python · NumPy · MATLAB / Octave · pytest · GitHub Actions

PROJECT PAGE
anas-makdesi.github.io/projects/data-driven-invariance

CODE
github.com/anas-makdesi/ddma

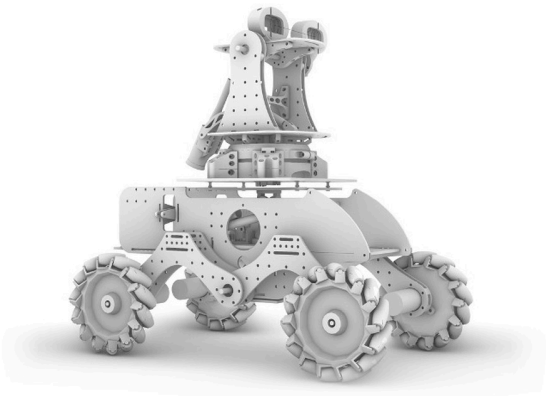
Surveying robot: survey-grade positioning on a construction site

A mobile robot that sets out points and measures what was built, keeping a construction site in sync with its digital twin. I designed its localization: a camera looking through a total station's telescope, combined with marker detection and surveying methods. Led a team of five.

< 1 mm
positioning accuracy target

In + out
indoor and outdoor construction sites

5
engineers in the team I led



CAD render of the surveying robot

HOW IT WORKS

1
Markers
Printed markers, with colours and materials chosen through indoor and outdoor tests.

2
Camera in the telescope
A commercial camera looks through a total station's telescope and detects the markers automatically.

3
Surveying methods
Precise angles from the total station and the marker detections feed standard surveying methods.

4
Navigation
The robot uses its position to reach set-out points and to place its measurements in site coordinates.

STACK Python · OpenCV · C/C++

PROJECT PAGE
anas-makdesi.github.io/projects/surveying-robot

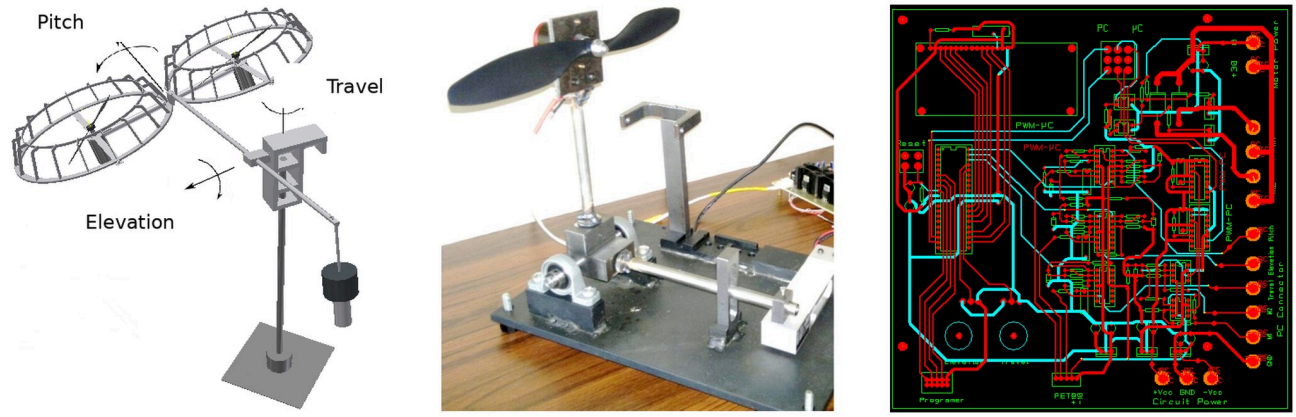
3-DOF helicopter rig: from circuit board to closed-loop control

A twin-rotor helicopter lab rig with three rotating axes and only two motors. I built its electronics (sensor conditioning, motor drivers, a custom PCB), identified its model on a thrust test bench, and designed and tested linear, gain-scheduled and sliding-mode controllers on the real hardware.

3 axes, 2 motors
underactuated: two inputs control three angles

≈ 1 s
pitch settling time on the real rig

2 ways to run it
stand-alone microcontroller or a PC running MATLAB in real time



The rig (CAD model), the thrust test bench and the custom control board

HOW IT WORKS

1
Mechanics
Added the helicopter head on a rotary joint, two DC motors with 12-inch carbon propellers in protective cages, and a counterweight.

2
Electronics
One board filters and scales the three angle sensors and drives both motors, with a switch that hands control to a microcontroller or to the PC.

3
Model
Equations of motion from first principles; thrust curves measured on a test bench; masses and inertias from measurement and a CAD model.

4
Control
Controllers designed from the model and tuned on the physical rig, axis by axis.

STACK MATLAB / Simulink real-time · Microcontroller · Analog electronics · PCB design · Autodesk Inventor

PROJECT PAGE
anas-makdesi.github.io/projects/helicopter-rig

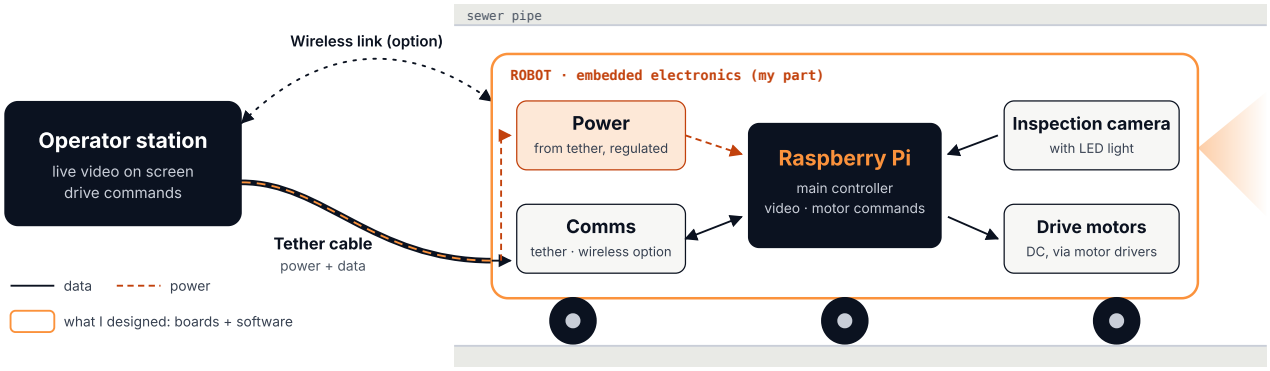
Sewer inspection robot: electronics for a robot that works inside pipes

A remotely driven robot that goes into sewer pipes and sends live video back to an operator. I designed its embedded electronics and wrote its software: a Raspberry Pi controller, motor drives, camera and lighting, and a tether that carries power and data, with a wireless option. Delivered to the client and field-tested.

Tether + wireless
power and data over one cable, wireless as an option

Live video
inspection camera with its own light

Field-tested
delivered to the client



Block diagram of the robot's electronics: operator station, tether and wireless link, power, Raspberry Pi, camera with light, drive motors

HOW IT WORKS

1
Drive in
The operator steers the robot into the pipe from a station at the surface.

2
Light and film
An inspection camera with an LED light films the pipe wall ahead.

3
Stream back
The Raspberry Pi sends the video to the operator and receives drive commands.

4
Stay connected
A tether carries power and data; a wireless link is available as an option.

STACK Raspberry Pi · Python · C/C++ · PCB design

PROJECT PAGE
anas-makdesi.github.io/projects/sewer-robot