Optimal real-time landing using deep networks

Carlos Sánchez and Dario Izzo
Advanced Concepts Team (ESA)

Daniel Hennes
Robotics Innovation Center (DFKI)

Learning the optimal state-feedback using deep networks

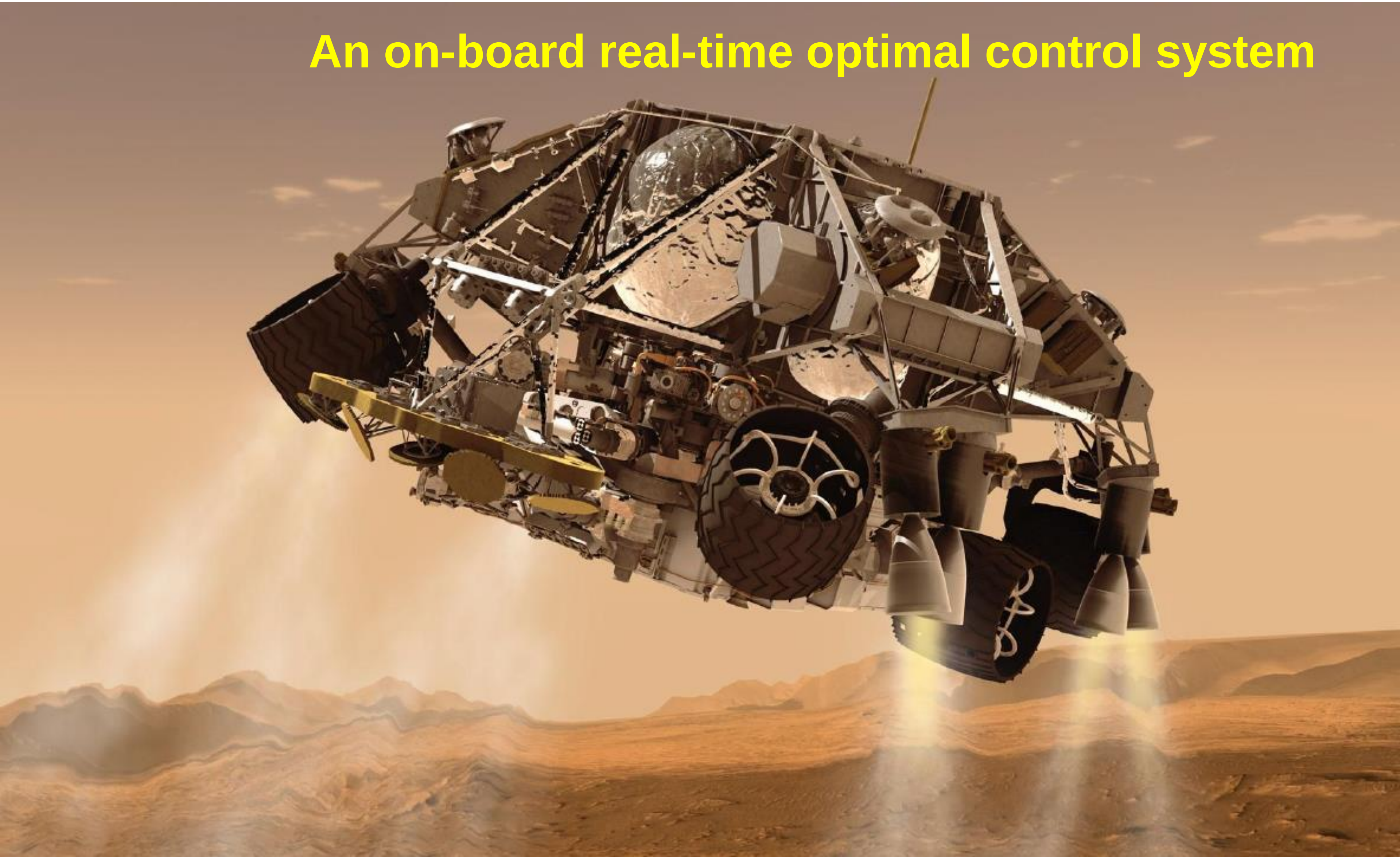
Rocket Landing Simulation

Watch the video clip

“Deep Learning in Deep Space”

https://youtu.be/8sDldAK4O0E

Goal is to make



Dynamic Systems

- The mathematical model of a system usually leads to a system of equations describing the nature of the interaction of the system.
- These equations are commonly known as governing laws or **model equations** of the system.
- The model equations can be :

  time independent ➔ steady-state model equations

  time dependent ➔ dynamic model equations

In this course, we are mainly interested in dynamical systems.

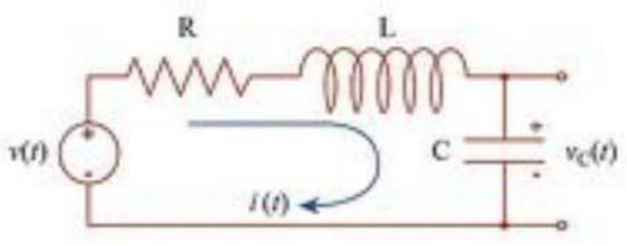
➔ Systems that evolve with time are known as **dynamic systems.**

Examples of Dynamic models – RLC Circuit

- Linear Differential Equations

$$\dot{x} = Ax + Bu$$

- Example RLC circuit (Ohm‘s and Kirchhoff‘s Laws)



$$\begin{pmatrix} \dot{i} \\ \dot{v}_C \end{pmatrix} = \begin{bmatrix} -R/L & -1/L \\ 1/C & 0 \end{bmatrix} \begin{pmatrix} i \\ v_C \end{pmatrix} + \begin{pmatrix} 1/L \\ 0 \end{pmatrix} v$$

$$\mathrm{x} = \begin{pmatrix} i \\ v_C \end{pmatrix}, \ \dot{\mathrm{x}} = \begin{pmatrix} \dot{i} \\ \dot{v}_C \end{pmatrix}, \ A = \begin{bmatrix} -R/L & -1/L \\ 1/C & 0 \end{bmatrix}, \ B = \begin{pmatrix} 1/L \\ 0 \end{pmatrix}, u = v$$

- Nonlinear Differential equations : general cases

$$\dot{x} = f(x(t), u(t), t)$$

Simulation ...

In mathematical systems theory, simulation is done by solving the governing equations of the system for various input scenarios.

- ➔ **This requires algorithms corresponding to the type of systems model equation.**
- ➔ **Numerical methods for the solution of systems of equations and differential equations.**

Optimization of Dynamic Systems

- A system with degrees of freedom can be always manipulated to display certain useful behavior.
- Manipulation ➔ possibility to control
- Control variables are usually systems degrees of freedom.

We ask:
What is the best control strategy that forces a system to display required characterstics, output, follow a trajectory, etc ?

➔ Optimal Control
➔ Methods of Numerical Optimization

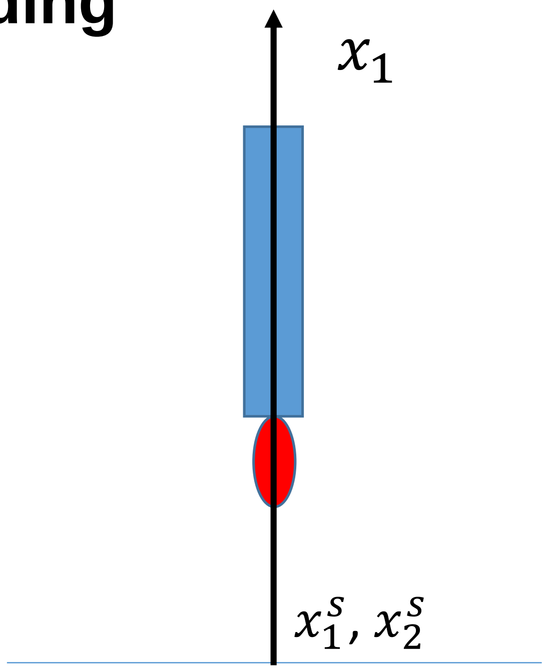
Ex. : Optimal Control of Landing

$x_1(t)$: Position

$x_2(t)$: Speed

$u(t)$: Propulsive Force

m : Mass ($m = 1$ kg)



Model Equations:

$$\dot{x}_1(t) = x_2(t)$$

$$u(t) = m\,a(t) = m\,\dot{x}_2(t)$$

$$\dot{x}_1(t) = x_2(t)$$

$$\dot{x}_2(t) = \frac{1}{m}u(t)$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

Ex. : Optimal Control of Landing

Objectives of the optimal control :

- Minimization of the error : $\boxed{(x_1^s - x_1(t)),\ (x_2^s - x_2(t))}$ **Cost @ terminal**
- Minimization of energy : $\boxed{\int_0^t u(t)dt}$ **Integration of cost rate during flight**

Problem formulation :

Cost function :

$$\min_{u(t)} \frac{1}{2}\int_0^\infty \left\{\left[x_1^S - x_1(t)\right]^2 + 2\left[x_2^S - x_2(t)\right]^2 + \left[u(t)\right]^2\right\} dt$$

Model (state) equations :

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

Initial states:

$$x_1(0) = 2; \quad x_2(0) = 1$$

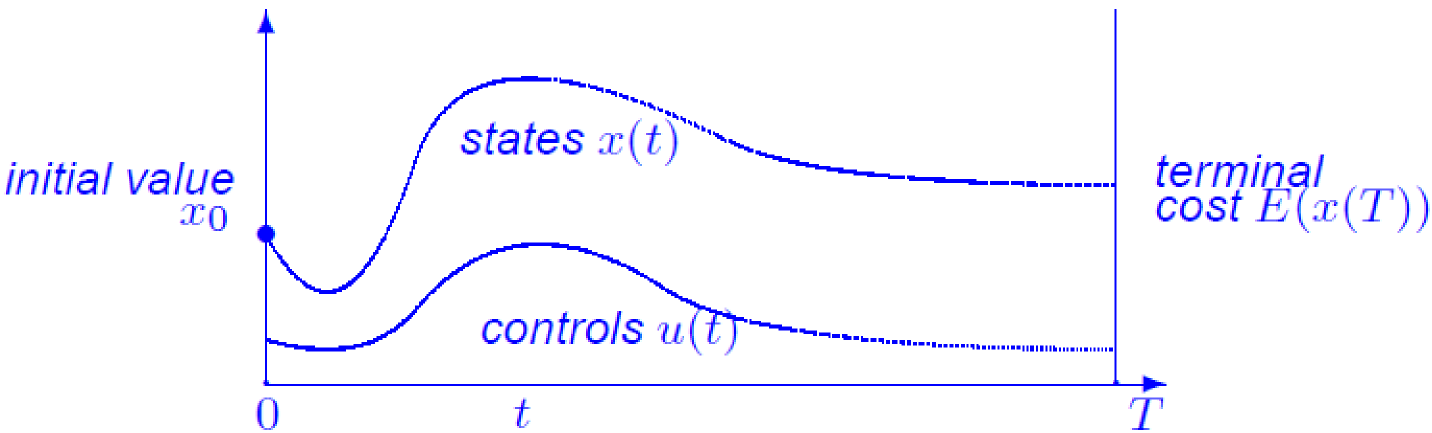
Desired final states:

$$x_1^S = 0; \quad x_2^S = 0$$

How to solve the above optimal control problem in order to achieve the desired goal ? That is, how to determine the optimal trajectories x_1^*(t), $x_2^*(t)$ that provide a minimum energy consumption $\int_0^t u(t)dt$ so that the rocket can be halted at the desired position?

Optimization of Dynamic Systems

Regard simplified optimal control problem:



$$\underset{x(\cdot),\,u(\cdot)}{\text{minimize}} \quad \int_0^T \underbrace{L(x(t),u(t))}_{\text{Cost rate @ t}}\,dt \;+\; E\,(x(T))$$

subject to

$$\begin{aligned} x(0)-x_0 &= 0, && && \text{(fixed initial value)} \\ \dot{x}(t)-f(x(t),u(t)) &= 0, && t\in[0,T]. && \text{(ODE model)} \end{aligned}$$

Euler Discretization

Introduce timestep

$$h = \frac{T}{N}$$

$$\underset{s,\,q}{\text{minimize}} \quad \sum_{i=0}^{N-1} hL(s_i, q_i) \;+\; E\,(s_N)$$

subject to

$$\begin{aligned} s_0 - x_0 &= 0, && && \text{(initial value)} \\ s_{i+1} - s_i - hf(s_i, q_i) &= 0, && i = 0, \ldots, N-1, && \text{(discretized system)} \end{aligned}$$

Dynamic Programming for Euler Scheme

Cost @ t=t_k $\quad J(x, t_k) = J_k(x)$

Cost @ t=0 $\quad J_0(x) = \sum_{k=0}^{N-1} hL(x, u, t_k) + E(u_N)$

$= hL(x, u, t_0) + \sum_{k=1}^{N-1} hL(x, u, t_k) + E(u_N)$

$= hL(x, u, t_0) + J_1(x)$

$= hL(x, u, t_0) + hL(x, u, t_1) + \cdots + hL(x, u, t_{N-1}) + J_N(x)$

Cost @ t=1 $\quad J_1(x) = hL(x, u, t_1) + J_2(x)$

$\vdots$

Cost @ t=N-1 $\quad J_{N-1}(x) = hL(x, u, t_{N-1}) + J_N(x)$

Cost @ t=N $\quad J_N(x) = E(u_N)$

So given **u**(¢) we can solve inductively backwards in time for objective **J**(**t**, **x**, **u**(¢)), starting at **t** = $\mathbf{t_N}$

➔ Called dynamic programming (DP)

Dynamic Programming for Euler Scheme

Using DP for Euler Discretized OCP yields:

Optimal control $$J_k(x) = \min_u hL(x,u) + J_{k+1}(x + hf(x,u))$$

Replacing the index k by time points $t_k = kh$ we obtain

$$J(x,t_k) = \min_u hL(x,u) + J(x + hf(x,u), t_k + h).$$

Assuming differentiability of $J(x,t)$ in (x,t) and Taylor expansion yields

$$J(x,t) = \min_u hL(x,u) + J(x,t) + h\nabla J(x,t)^T f(x,u) + h\frac{\partial J}{\partial t}(x,t) + O(h^2)$$

Hamilton-Jacobi-Bellman (HJB) Equation

Bringing all terms independent of u to the left side and dividing by $h \rightarrow 0$ yields

$$-\frac{\partial J}{\partial t}(x,t) = \min_u L(x,u) + \nabla J(x,t)^T f(x,u)$$

This is the famous Hamilton-Jacobi-Bellman equation.
We solve this partial differential equation (PDE) backwards for $t \in [0,T]$, starting at the end of the horizon with

$$J(x,T) = E(x).$$

NOTE: Optimal feedback control for state x at time t is obtained from

$$u^*(x,t) = \arg\min_u L(x,u) + \nabla J(x,t)^T f(x,u)$$

Continuous Time Optimal Control

Optimal feedback control for state x at time t is obtained from

$$u^*(x,t) = \arg\min_u L(x,u) + \nabla J(x,t)^T f(x,u)$$

optimal control policy

Hamilton-Jacobi-Bellman equation

a set of extremely challenging partial differential equations

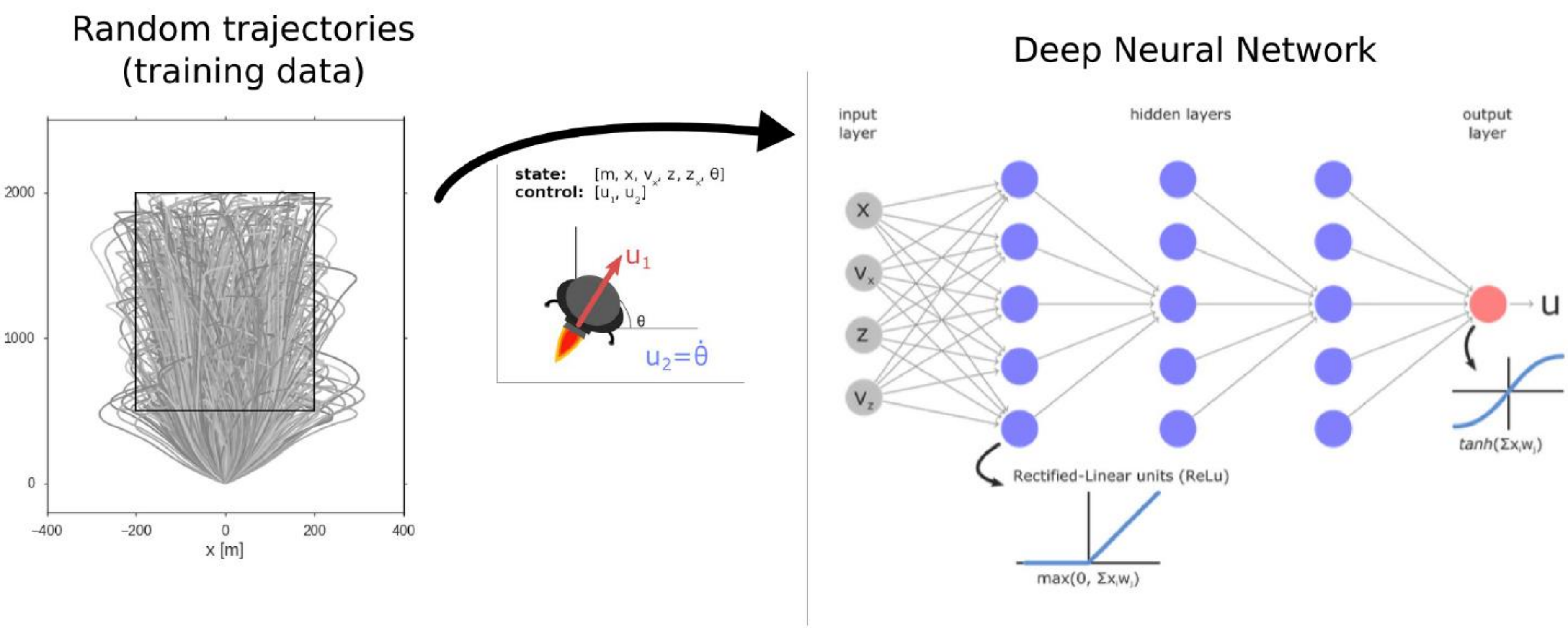
➔

Impossible to implement an Onboard Real Time Controller

Proposed Approach

Learning the optimal control policy using deep networks

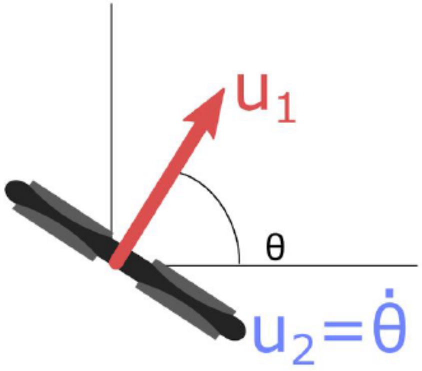
1. Pre-compute many optimal trajectories
2. Train an artificial neural network to approximate the optimal behaviour
3. Use the network to drive the spacecraft



Landing models

Multicopter
(Earth pinpoint landing)

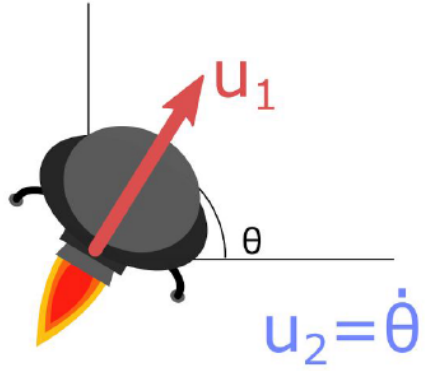
in $[x, v_x, z, z_x, \theta]$
out $[u_1, u_2]$



- Fixed mass
- Thrust + Torque
- Optimize power & time

Spacecraft I
(Moon free landing)

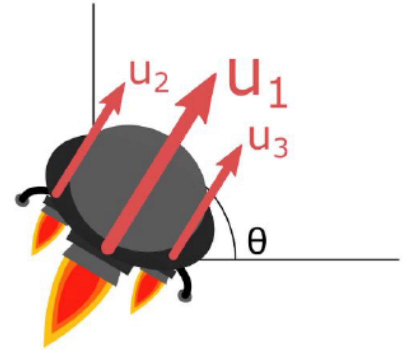
in $[m, x, v_x, z, z_x, \theta]$
out $[u_1, u_2]$



- Variable mass
- Thrust + exchange momentum wheel
- Optimize mass

Spacecraft II
(Moon free landing)

in $[m, x, v_x, z, v_z, \theta, v_\theta]$
out $[u_1, u_2, u_3]$

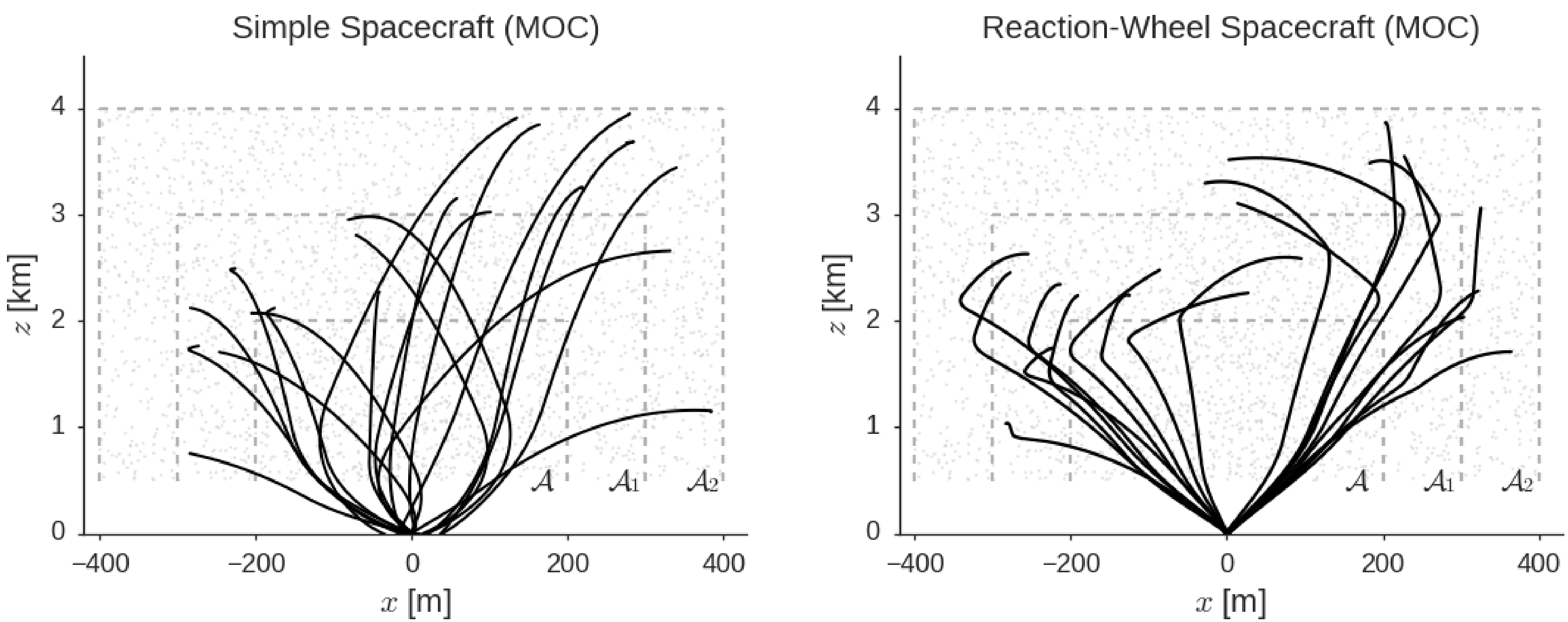


- Variable mass
- Main thrust + two lateral engines
- Optimize mass

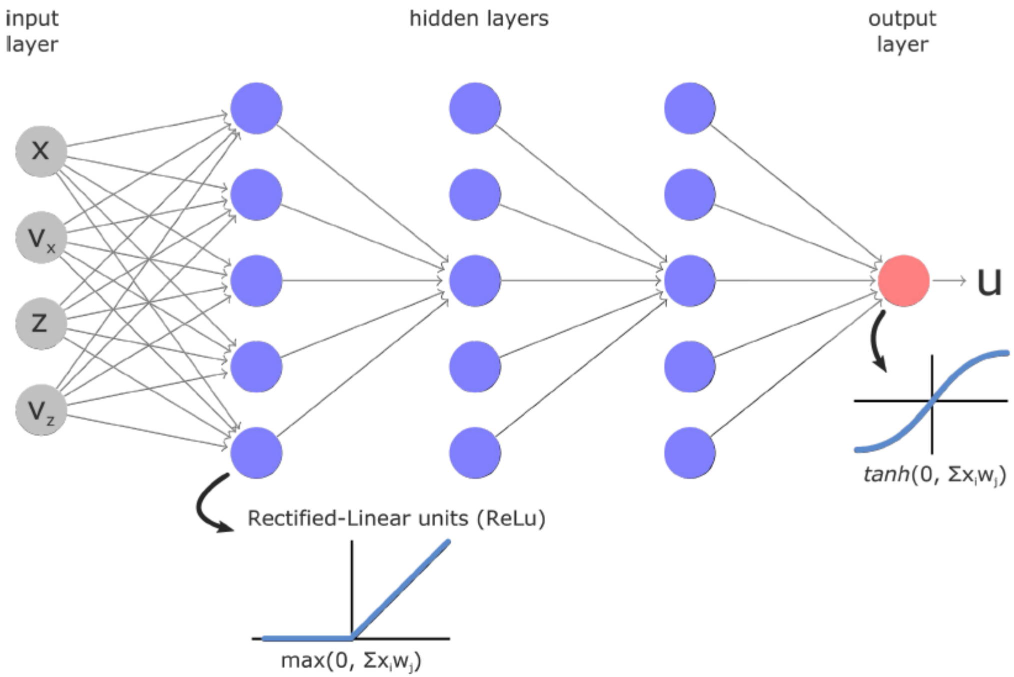
Training data generation

- The training data is generated using the **Hermite-Simpson transcription** and a **non-linear programming** (NLP) solver
- The initial state of each trajectory is randomly selected from a training area
- **150,000 trajectories** are generated for each one of the problems

(9,000,000 - 15,000,000 data points)



Approximate state-action with a DNN



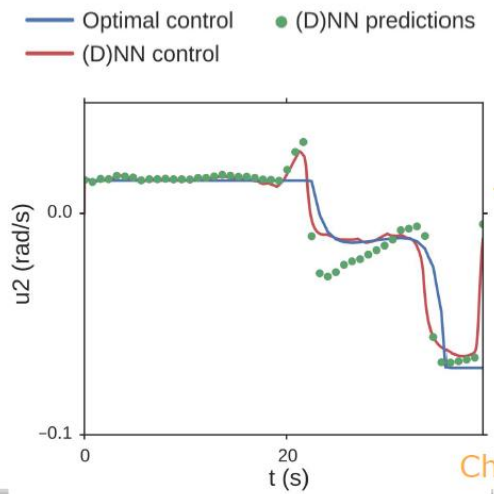
- Networks with 1 - 4 hidden layers
- **Stochastic Gradient Descent** (and momentum)

$$v_i \rightarrow v_i' = \mu v_i - \eta \frac{\partial C}{\partial w_i}$$

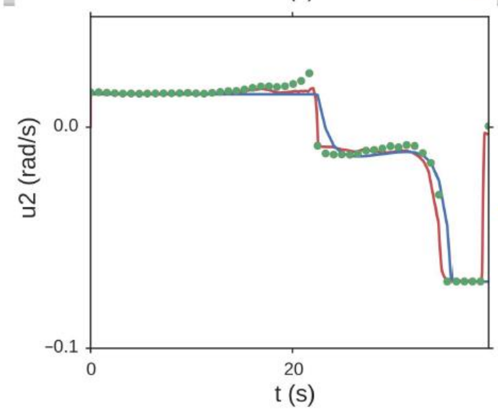
$$w_i \rightarrow w_i' = w_i + v_i'$$

- Minimize the squared loss error (C)
- We integrate over time the dynamics to get the full DNN-driven trajectory

Approximate state-action with a DNN



Challenging landing example

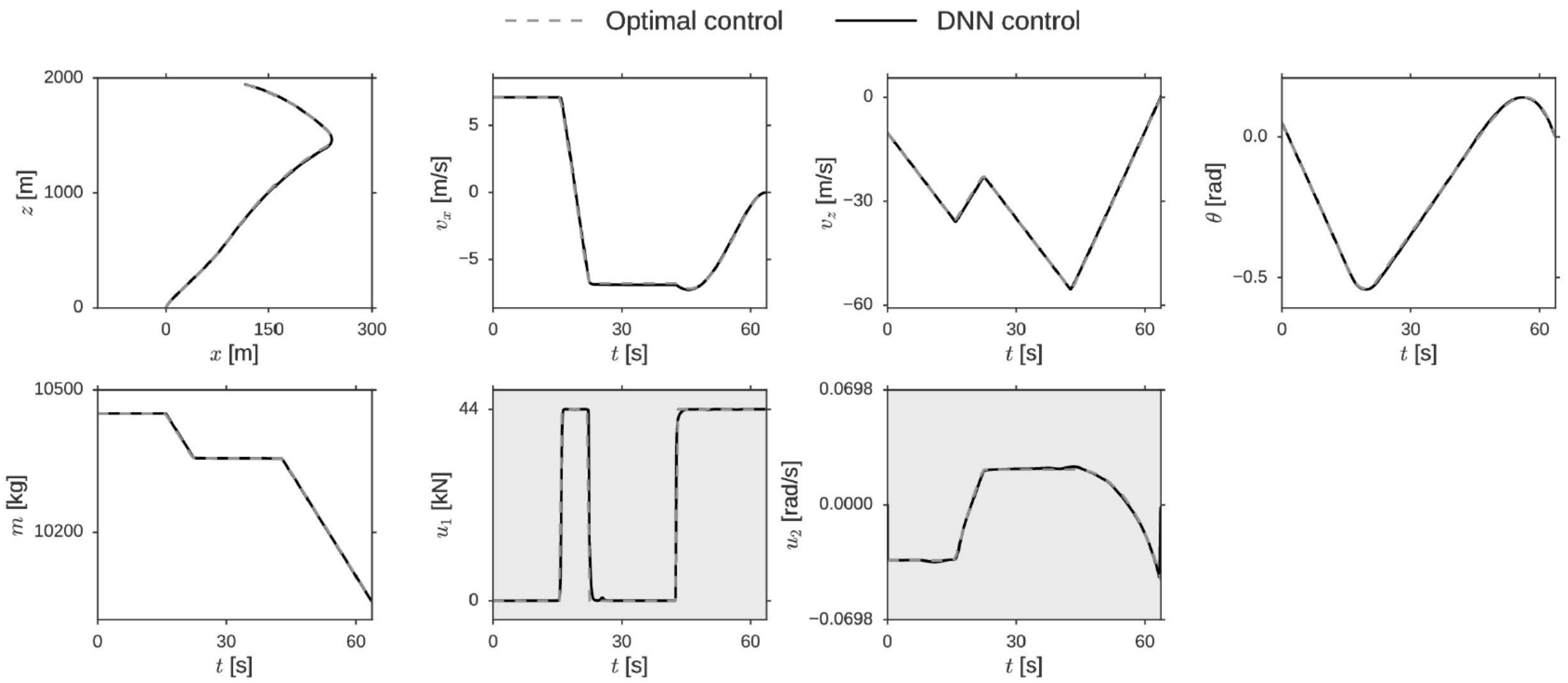


Spacecraft model 1 - Mass

	Layers	Units	#params	Train	Val.
u_2	2	256	2,048	0,00462	0,00460
	2	1,104	8,834	0,00379	0,00379
	2	2,048	16,384	0,00370	0,00371
	3	64	4,672	0,00307	0,00307
	3	128	17,536	0,00270	0,00272
	3	256	67,840	0,00263	0,00264
	4	64	8,832	0,00250	0,00252
	4	128	34,048	0,00241	0,00242
	5	64	12,992	0,00236	0,00237

- Deep networks are always better than shallow networks with the same number of parameters.

Approximate state-action with a DNN



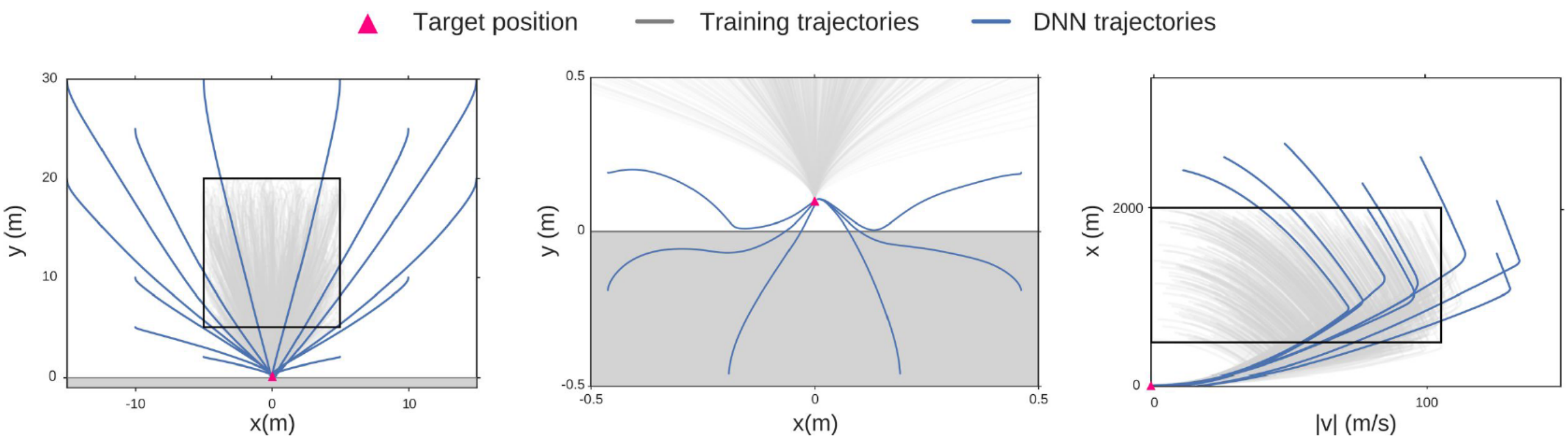
How good is it ?

Table 6 Performance of the DNN-driven trajectories.

	Success rate	Distance to target				Optimality
		r [m]	v [m/s]	θ [deg]	ω [deg/s]	
QUAD-QC	100.0%	0.014	0.027	0.36	-	1.82%
QUAD-TOC	100.0%	0.016	0.028	0.48	-	1.12%
SSC-QC	100.0%	0.40	0.052	-	-	0.24%
SSC-MOC	100.0%	2.47	0.12	-	-	0.45%
RWSC-QC	100.0%	0.29	0.044	0.20	-	0.40%
RWSC-MOC	98.3%	2.90	0.073	0.31	-	0.72%
TVR-QC	99.0%	1.10	0.066	0.06	0.0075	0.38%
TVR-MOC	95.0%	1.95	0.094	0.012	0.0054	0.33%

Very accurate results
+
The DNNs can be used as an on-board reactive system (32.000x faster than optimization methods used to generate the data)

Generalization ?



MULTICOPTER (POWER)

MULTICOPTER (POWER)

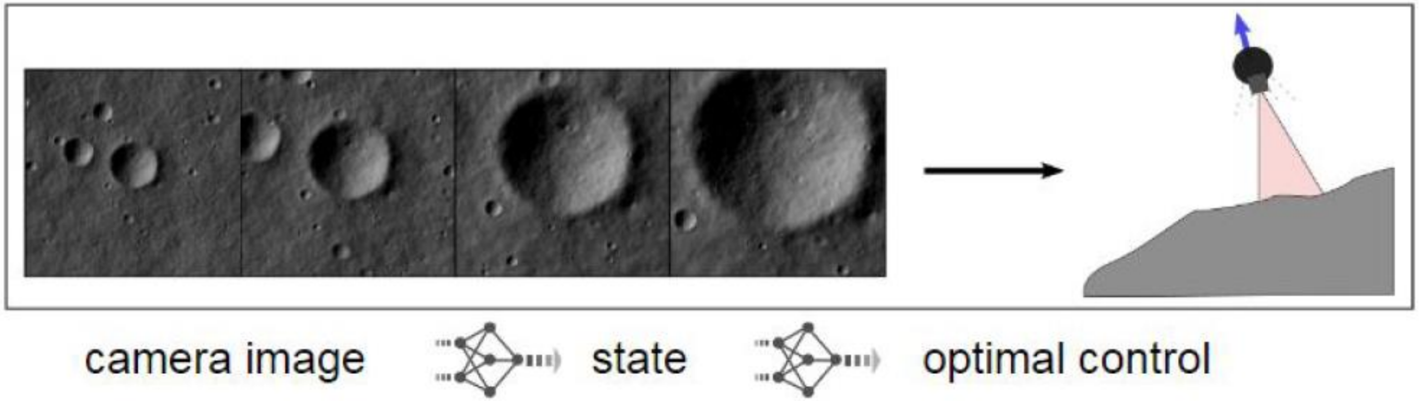
SPACECRAFT I

- Successful landings from states outside of the training initial conditions
- This suggest that an approximation to the solution of the HJB equations is learned


ADDING A CNN FOR THE PERCEPTION

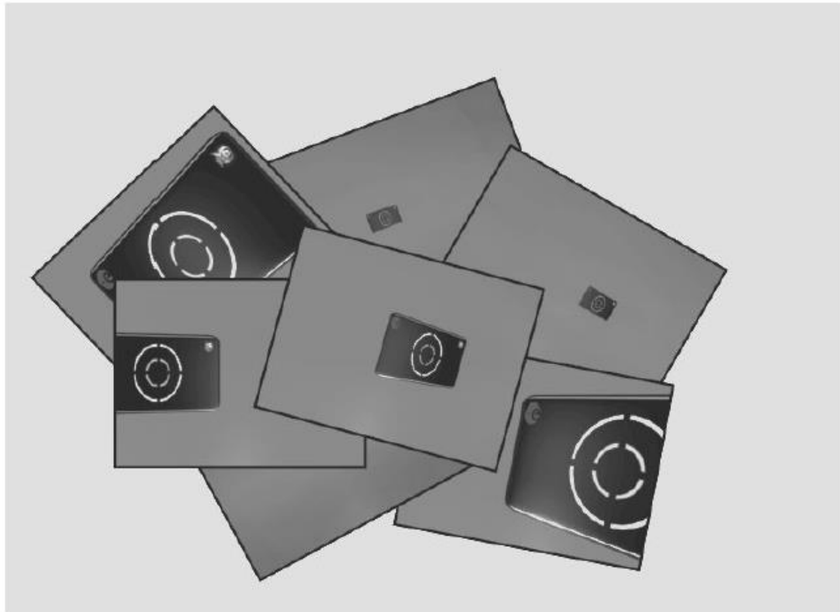
CNN for state estimation from camera

1. Train a neural network to guess the state from an on-board camera
2. Use it together with the previous DNNs to get fully automated visual landing

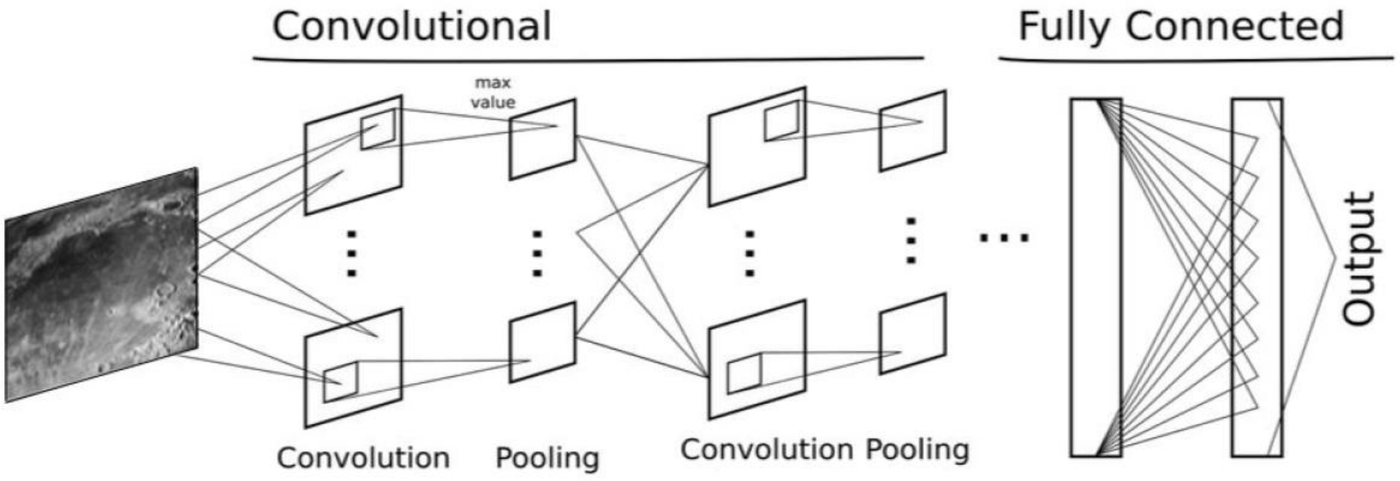


CNN for state estimation from camera

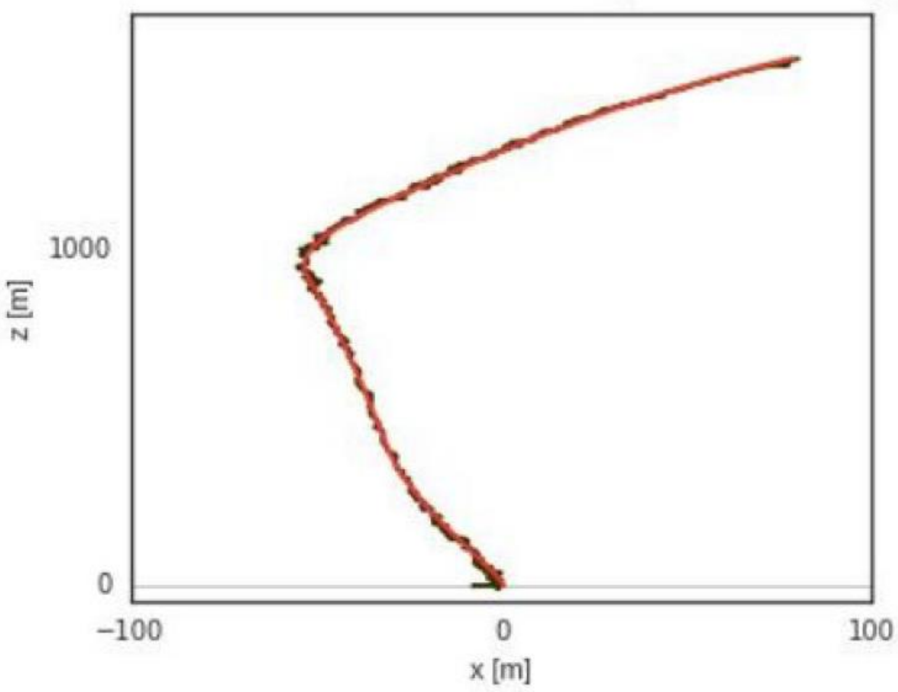
A simple setup is used: a 3D model (Blender) of a rocket landing on a sea platform (Falcon 9 inspired

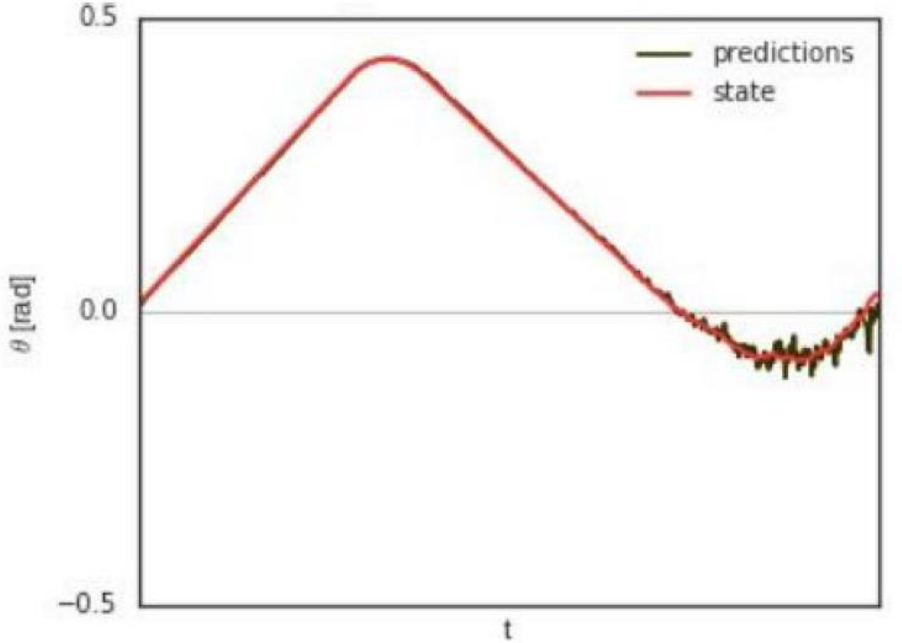


CNN for state estimation from camera

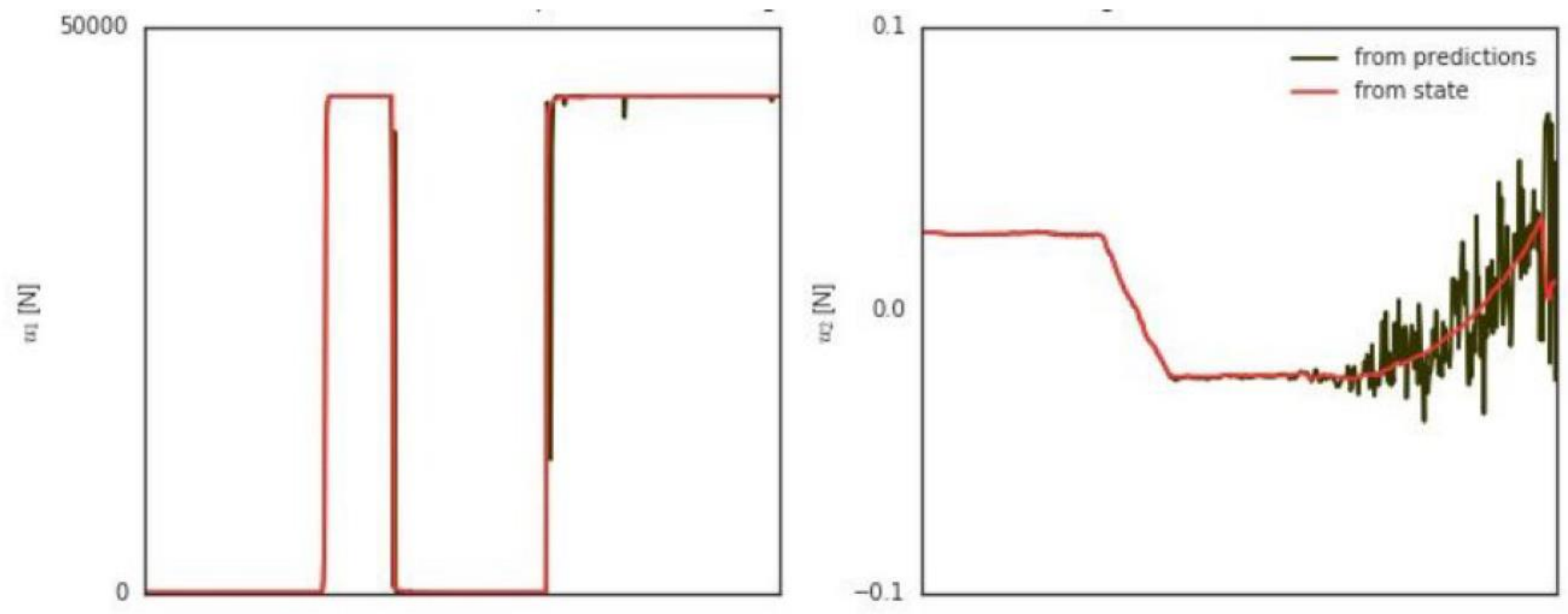


CNN for state estimation from camera





CNN for state estimation from camera



Watch the video clip

https://youtu.be/8sDldAK4O0E

Conclusions

- **Deep networks trained with large datasets** successfully approximate the optimal control even for regions out of the training data
- DNNs can be used as an **on-board reactive control system**, while current methods fail to compute optimal trajectories in an efficient way