VirtualArena: An Object-Oriented MATLAB Toolkit for Control System Design and Simulation

Andrea Alessandretti (speaker)[1]
A. Pedro Aguiar[1]
Colin N. Jones[2]

Universidade do Porto
FEUP Faculdade de Engenharia

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

1 Faculty of Engineering, University of Porto (FEUP), Porto, Portugal
2 Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Objective:

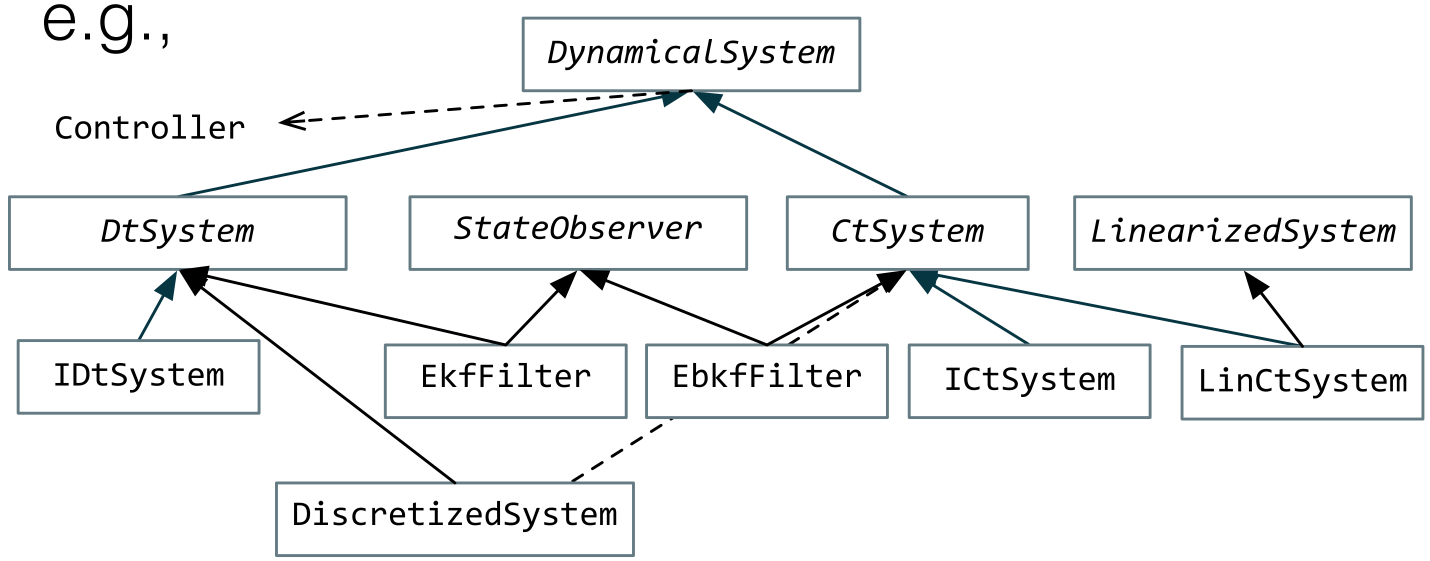
Matlab-based simulation of complex single-agent/multi-agent scenarios

(e.g., cooperative/distributed control of a network of vehicles)

Main challenge:

Increase of complexity → Need of modular/scalable code with reusable and adaptable components

e.g.,



VA: main focus on control-theoretical structure!

Toolkit requirements

What:

- Standard simulation-oriented functionalities
  (e.g., log management, numerical integrators, plots management…)
- Library of control-oriented components
  (e.g., time-varying of comm. networks, Extended Kalman filter, Model Predictive Control solvers,…)

How:

- Extendable architecture
- Collaborative design
- Easy to disseminate component

Case study:

Control design for a wheeled robot

- Vehicle model definition
- State-feedback controller definition
- Extended Kalman Filter design for output feedback
- Design Model Predictive Control law

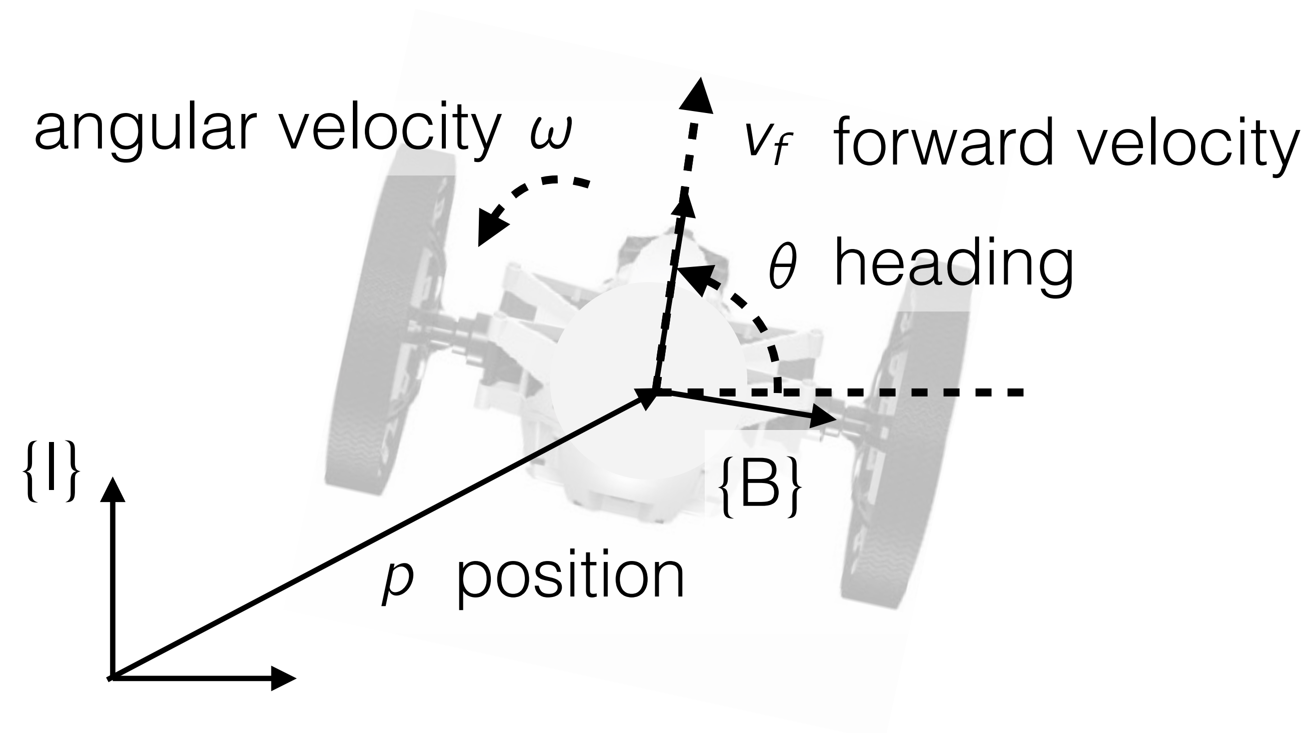
Vehicle model

State vector

$$x = \begin{pmatrix} p \\ \theta \end{pmatrix} \in \mathbb{R}^3$$

Input vector

$$u = \begin{pmatrix} v_f \\ \omega \end{pmatrix} \in \mathbb{R}^2$$



State equation

$$\dot{x} = \begin{pmatrix} R(\theta) \begin{pmatrix} v_f \\ 0 \end{pmatrix} \\ \omega \end{pmatrix} = \begin{pmatrix} v_f \cos(\theta) \\ v_f \sin(\theta) \\ \omega \end{pmatrix}$$

```
sys = ICtSystem(...
    'StateEquation', @(t,x,u,varargin) [
    u(1)*cos(x(3));
    u(1)*sin(x(3));
    u(2)],...
    'nx',3,'nu',2 ...
);
```

State-feedback controller > controller definition

Control point

$$c(t) := p(t) + R(t)\epsilon$$

Control objective

$$t \to \infty \implies c(t) \to c_d(t)$$

controller parameters

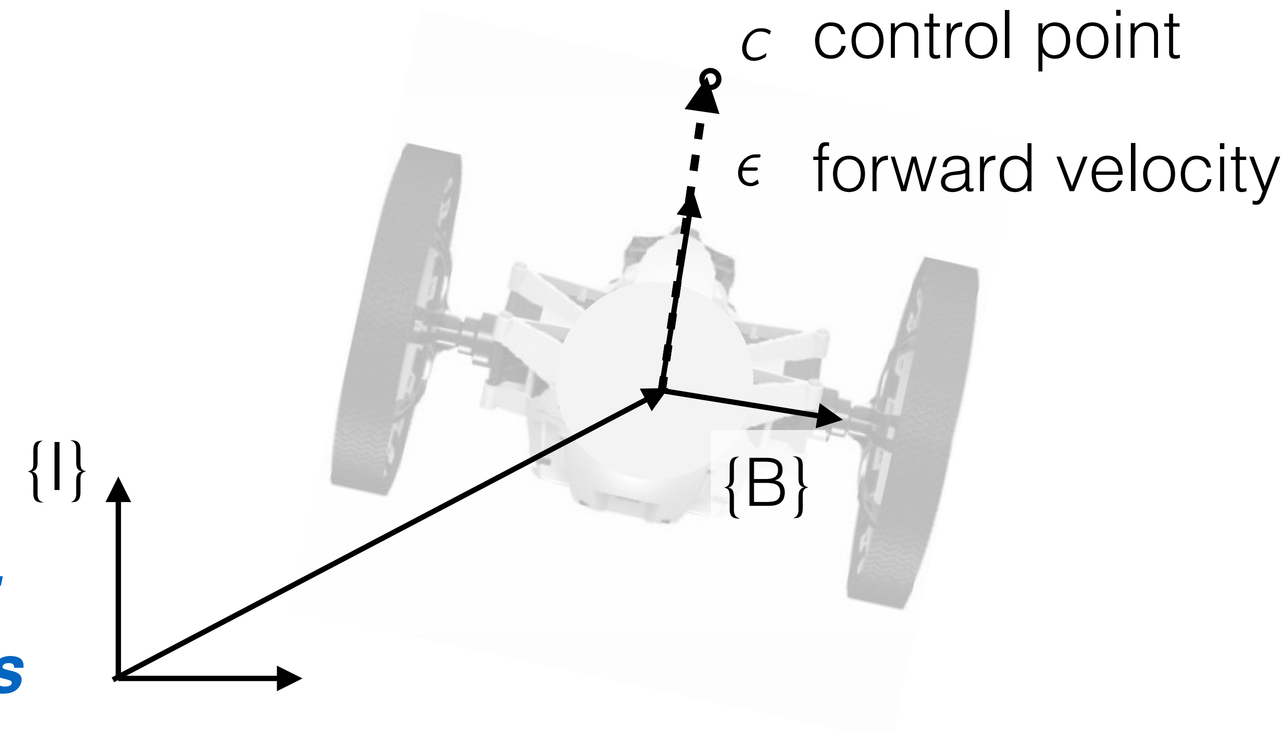
Control law

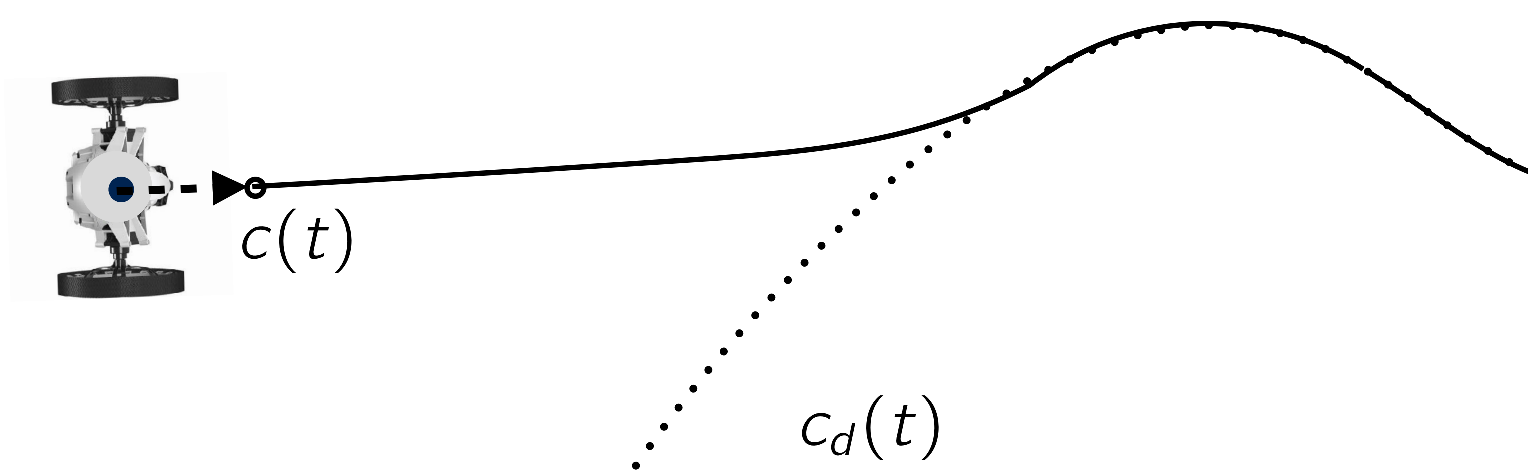
$$u(t) = \Delta^{-1}(R(t)'\dot{c}_d(t) - Ke(t))$$

$$e(t) = R(t)'(c(t) - c_d(t))$$

$$\Delta = \begin{pmatrix} 1, & -\epsilon_2 \\ 0, & \epsilon_1 \end{pmatrix}$$ full rank

controller assumptions





State-feedback controller > controller definition

in controller file

```
methods
function obj = TrackingController (cdes,cdesDot,K,epsilon)

   obj = obj@Controller();

    obj.cdes = cdes; obj.cdesDot = cdesDot;
    obj.K = K; obj.epsilon = epsilon;

    Delta   = [[1;0],[-epsilon(2);epsilon(1)]];

    if not(rank(Delta)==size(Delta,1))
        error('Assumption violated');
    end

    obj.invDelta = inv(Delta);

   obj.R = @(x)[cos(x(3)),-sin(x(3)); sin(x(3)),cos(x(3))];
end
function u = computeInput(obj,t,x)
    e = obj.computeError(t,x);
    u = -obj.invDelta*(obj.K*e-obj.R(x)'*obj.cdesDot(t));
end
function e = computeError(obj,t,x)
    p = x(1:2);
    e = obj.R(x)'*(p-obj.cdes(t))+obj.epsilon;
end
```

controller parameters

assumption validation

pre-computations for code optimization

object controller creation

control law

controller related functions

State-feedback controller > simulation

in controller file

```
methods
function obj = TrackingController (cdes,cdesDot,K,epsilon)

   obj = obj@Controller();

    obj.cdes = cdes; obj.cdesDot = cdesDot;
    obj.K = K; obj.epsilon = epsilon;

    Delta   = [[1;0],[-epsilon(2);epsilon(1)]];

    if not(rank(Delta)==size(Delta,1))
        error(‘Assumption violated');
    end

    obj.invDelta = inv(Delta);

   obj.R = @(x)[cos(x(3)),-sin(x(3)); sin(x(3)),cos(x(3))];
end
function u = computeInput(obj,t,x)
    e = obj.computeError(t,x);
    u = -obj.invDelta*(obj.K*e-obj.R(x)'*obj.cdesDot(t));
end
function e = computeError(obj,t,x)
    p = x(1:2);
    e = obj.R(x)'*(p-obj.cdes(t))+obj.epsilon;
end
```

controller parameters

assumption validation

pre-computations for code optimization

object controller creation

control law

controller related functions

Simulation

```
sys.initialCondition = ...
{[15;15;-pi/2],-[15;15;-pi/2],...
 [15;-15;pi],[-15;15;-pi/2]};

sys.controller = TrackingController(...
    @(t) 10*[sin(0.1*t); cos(0.1*t)] , ... % c
    @(t)    [cos(0.1*t);-sin(0.1*t)] , ... % cDot
    eye(2)                           , ... % K
    [1;0] ); ... % epsilon


va = VirtualArena(sys,...
    'StoppingCriteria'   , @(t,sysList)t>70,...
    'DiscretizationStep', dt,…
    'StepPlotFunction'   , @plotFunction);

log = va.run();
```

- assumption validation
- code optimization

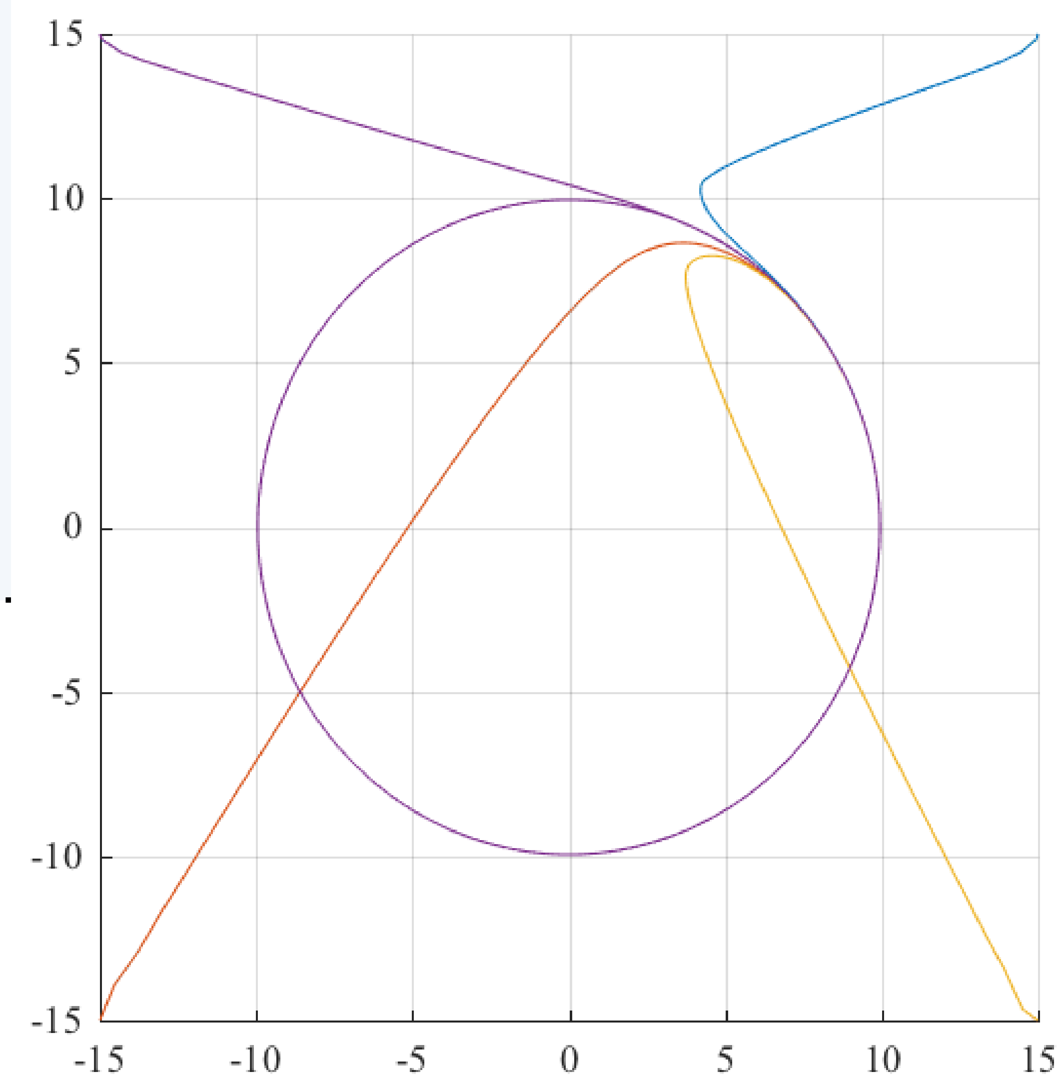
- simulation control

- Modular/self-contained design using pre-defined VA interfaces
- Easy to share without knowledge of implementation details
- VA routines (e.g., automatic system discretion, log management, simulation from multiple initial conditions, simulation management…)

Simulation

```
sys.initialCondition = ...
{[15;15;-pi/2],-[15;15;-pi/2],...
 [15;-15;pi],[-15;15;-pi/2]};

sys.controller = TrackingController(...
    @(t) 10*[sin(0.1*t); cos(0.1*t)] , ... % c
    @(t)     [cos(0.1*t);-sin(0.1*t)] , ... % cDot
    eye(2)                              , ... % K
    [1;0] ); ... % epsilon


va = VirtualArena(sys,...
    'StoppingCriteria'   , @(t,sysList)t>70,..
    'DiscretizationStep', dt,…
    'StepPlotFunction'   , @plotFunction);

log = va.run();
```

```
>> log{1}{1}
ans =
    inputTrajectory: [2x7002 double]
    stateTrajectory: [3x7002 double]
               time: [1x7002 double]
```



Output feedback > definition

Output equation

$$y(t) = p(t) \quad \textit{e.g., G.P.S.}$$

State feedback controller

Need of state observer

```
...
sys = ICtSystem(…
    ...
    'OutputEquation',@(t,x,varargin)x(1:2),...
    'ny',2,...
    ...
);
...

realSystem = ICtSystem( ... );
```

Add this line to original system

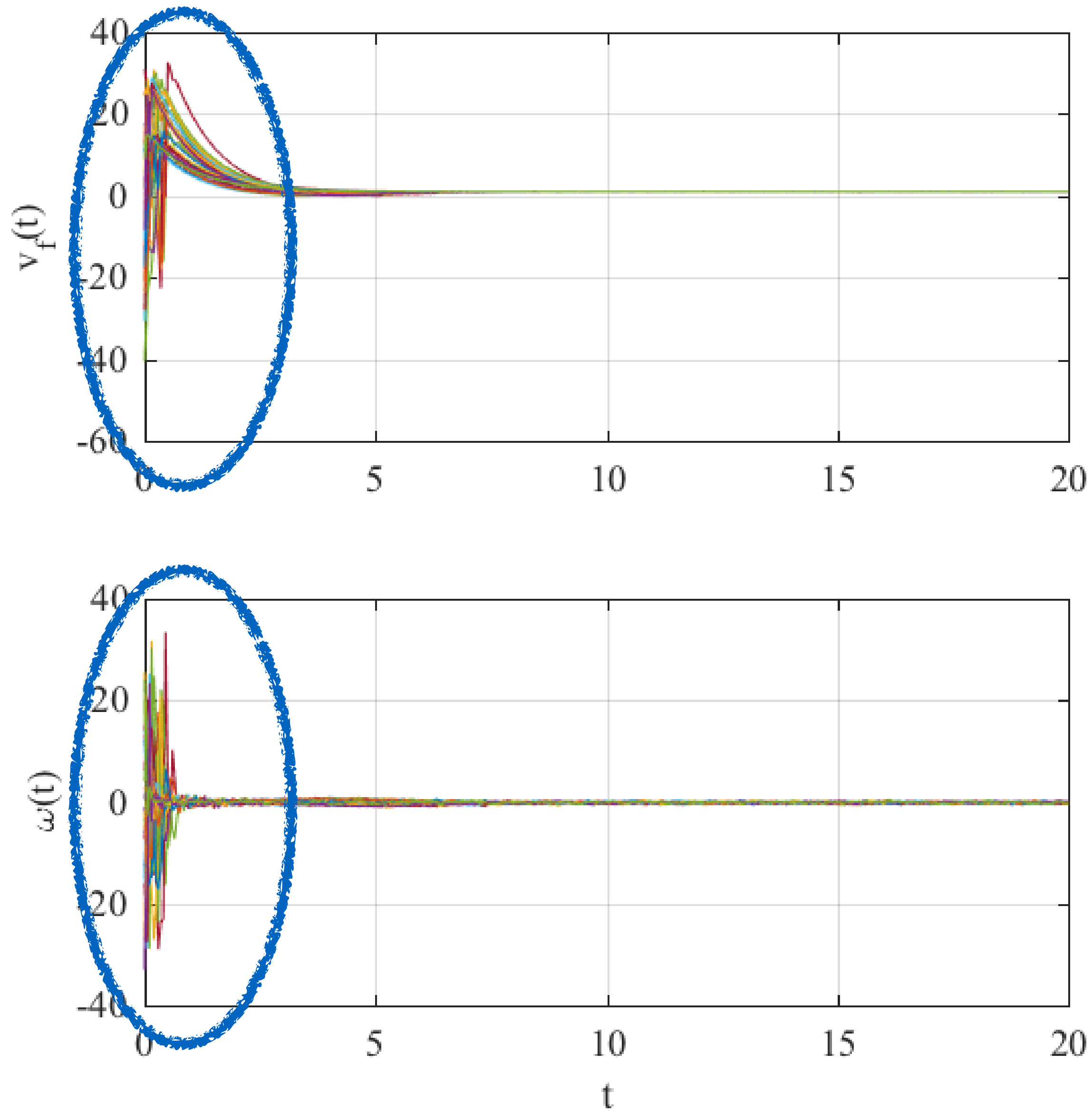
More realistic model (e.g., sys with additive noise)

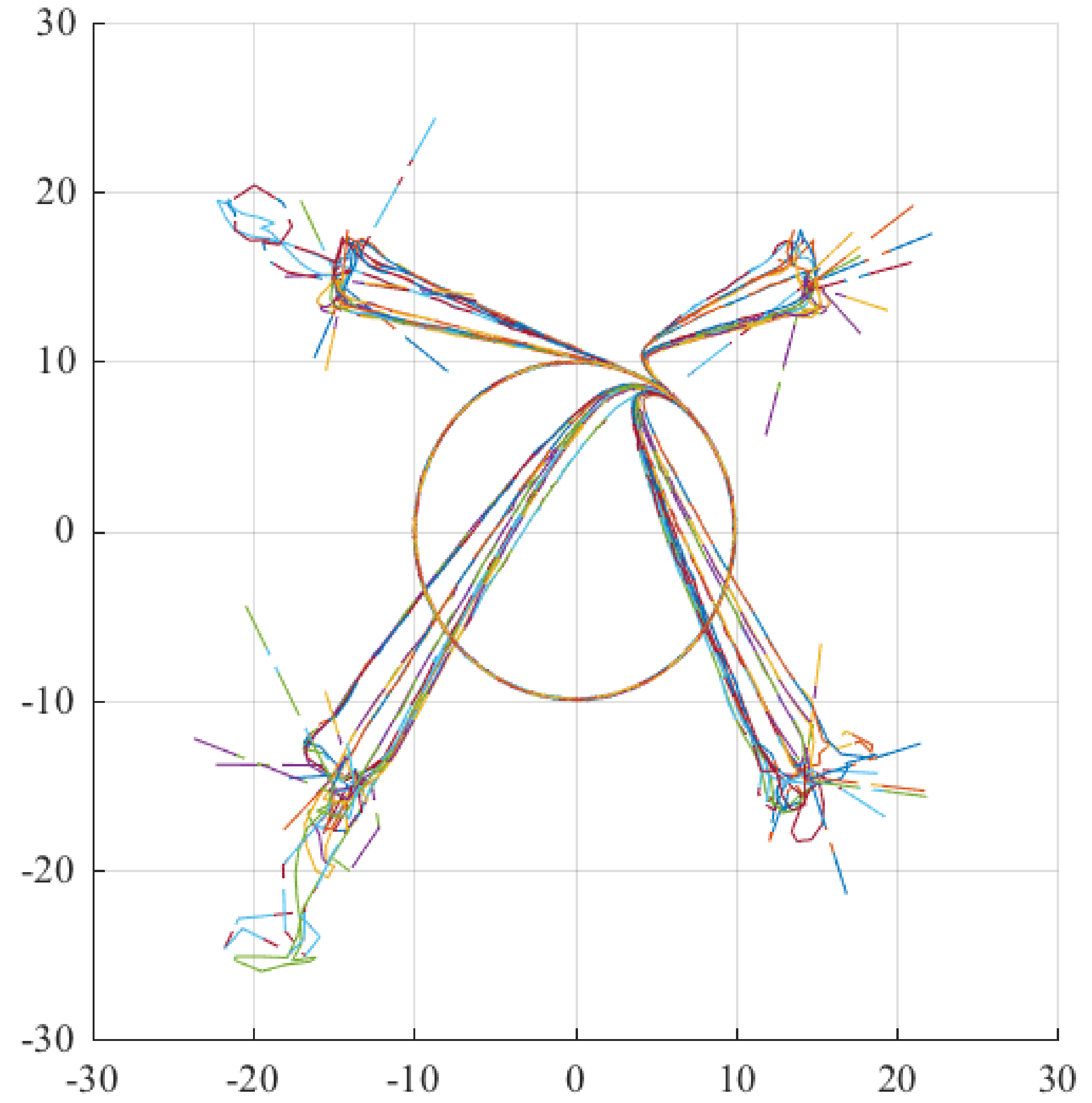
```
realSystem.stateObserver = EkfFilter(...
    DiscretizedSystem(sys,dt),...
    'StateNoiseMatrix' , dt*Q,...
    'OutputNoiseMatrix', R,...
    'InitialCondition' , x0Filter);

% Initial conditions
...
realSystem.controller =...
  TrackingController
...
va = VirtualArena(realSystem,...
```

automatic (customizable):

- discretization routines
- linearization routines
- measurement management

Output feedback > definition





High control input

Iterate on control design (e.g., use Model Predictive Control)

Model Predictive Control

Model Predictive Control

Performance index
(control objective)

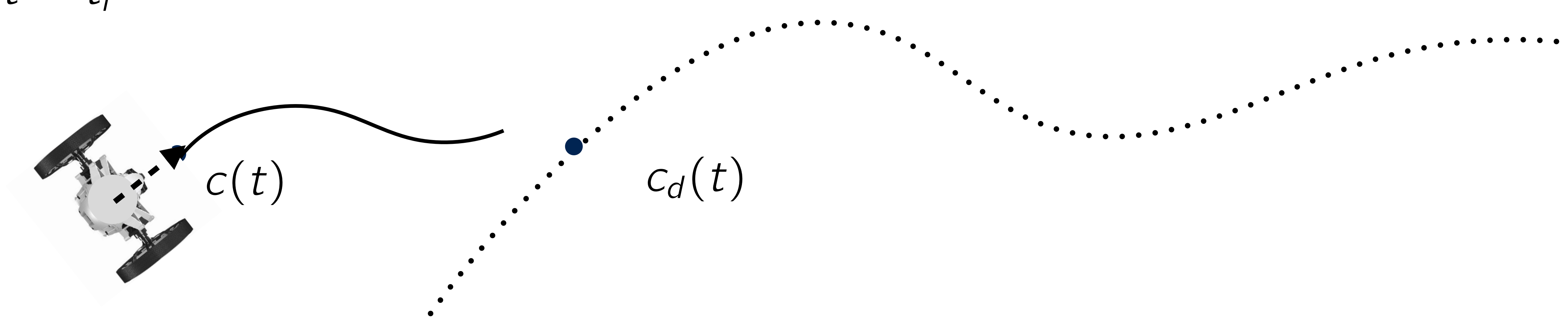
$$J_T(t, z, \bar{\boldsymbol{u}}) := \underbrace{\int_t^{t+T} l(\tau, \bar{x}(\tau), \bar{u}(\tau))d\tau}_{\text{Stage cost}} + \underbrace{m(t+T, \bar{x}(t+T))}_{\text{Terminal cost}}$$

Tracking error	$e(t) = R(t)'(c(t) - c_d(t))$
Stage cost	$l(t, x(t), u(t)) = \|e(t)\|^2$
Terminal cost	$m(t, x(t)) = 0.333\|e(t)\|^3$

$$(\bar{x}(\tau), \bar{u}(\tau)) \in \overset{\text{Obstacles}}{\mathcal{X}(\tau)} \times \overset{\text{Actuator limits}}{\mathcal{U}(\tau)}$$

System constraints

$\mathcal{T} = \{t_0, t_1, ...\}$
$t = t_i$



1. Compute the optimal finite horizon prediction.

Model Predictive Control

Model Predictive Control

Performance index
(control objective)

$$J_T(t, z, \bar{\boldsymbol{u}}) := \int_t^{t+T} \underbrace{l(\tau, \bar{x}(\tau), \bar{u}(\tau))d\tau}_{\text{Stage cost}} + \underbrace{m(t+T, \bar{x}(t+T))}_{\text{Terminal cost}}$$

Tracking error $e(t) = R(t)'(c(t) - c_d(t))$

Stage cost $l(t, x(t), u(t)) = \|e(t)\|^2$

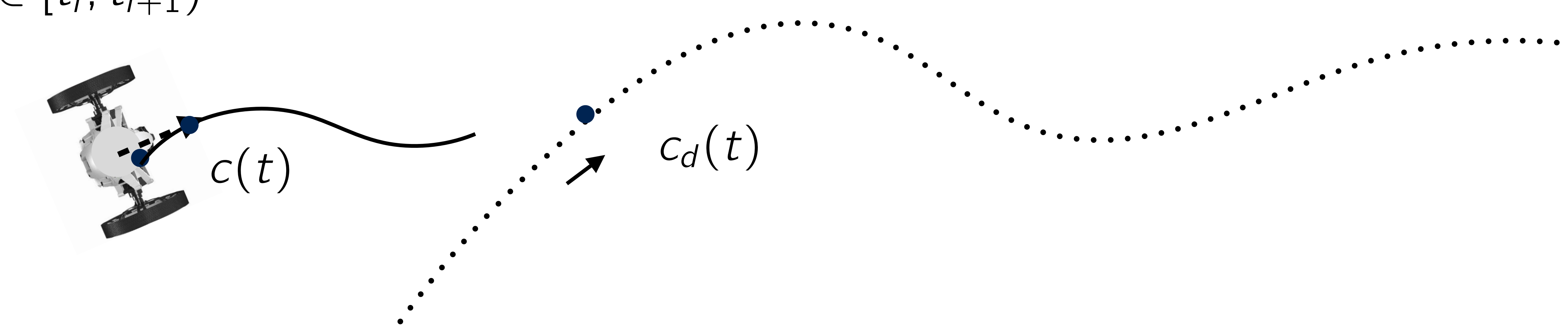
Terminal cost $m(t, x(t)) = 0.333\|e(t)\|^3$

Obstacles *Actuator limits*

$$(\bar{x}(\tau), \bar{u}(\tau)) \in \mathcal{X}(\tau) \times \mathcal{U}(\tau)$$

System constraints

$\mathcal{T} = \{t_0, t_1, \ldots\}$
$t \in [t_i, t_{i+1})$



2. Apply part of the optimal input to the system.

Model Predictive Control

Model Predictive Control

Performance index
(control objective)

$$J_T(t, z, \bar{\boldsymbol{u}}) := \underbrace{\int_t^{t+T} l(\tau, \bar{x}(\tau), \bar{u}(\tau))d\tau}_{\text{Stage cost}} + \underbrace{m(t+T, \bar{x}(t+T))}_{\text{Terminal cost}}$$

Tracking error $e(t) = R(t)'(c(t) - c_d(t))$

Stage cost $l(t, x(t), u(t)) = \|e(t)\|^2$

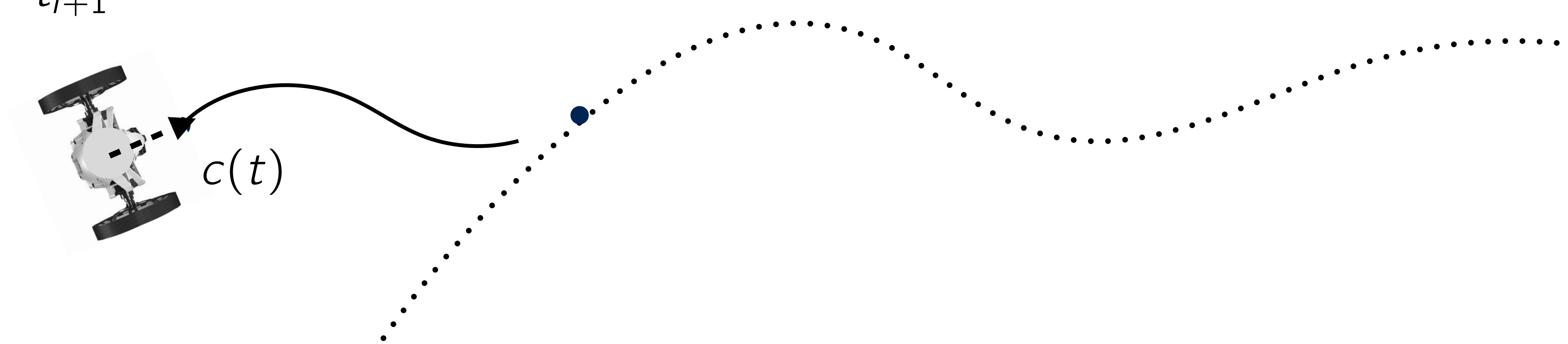
Terminal cost $m(t, x(t)) = 0.333\|e(t)\|^3$

$$(\bar{x}(\tau), \bar{u}(\tau)) \in \overset{\text{Obstacles}}{\mathcal{X}(\tau)} \times \overset{\text{Actuator limits}}{\mathcal{U}(\tau)}$$

System constraints

$\mathcal{T} = \{t_0, t_1, \ldots\}$

$t = t_{i+1}$



3. Iterate.

Model Predictive Control

Model Predictive Control

Performance index
(control objective)

$$J_T(t, z, \bar{\boldsymbol{u}}) := \int_t^{t+T} \underbrace{l(\tau, \bar{x}(\tau), \bar{u}(\tau))}_{\text{Stage cost}} d\tau + \underbrace{m(t+T, \bar{x}(t+T))}_{\text{Terminal cost}}$$

Tracking error $e(t) = R(t)'(c(t) - c_d(t))$

Stage cost $l(t, x(t), u(t)) = \|e(t)\|^2$

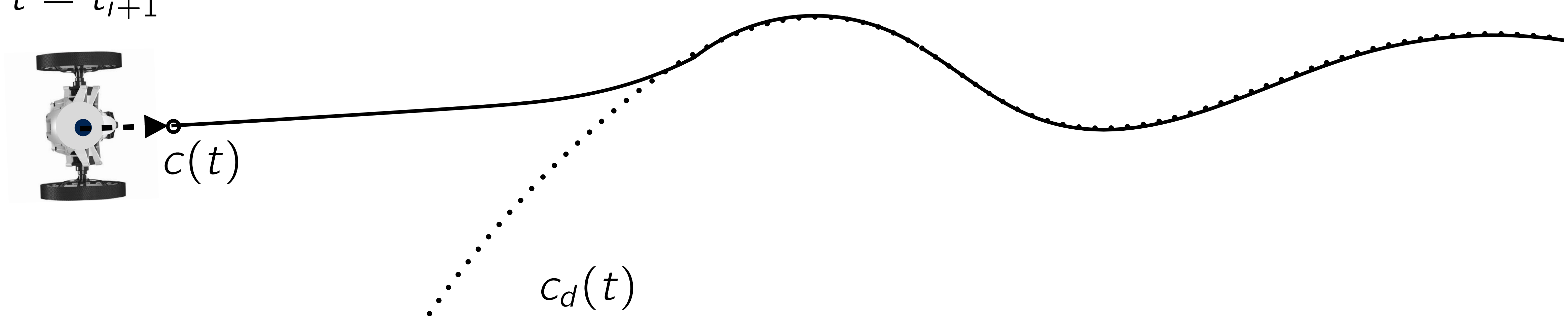
Terminal cost $m(t, x(t)) = 0.333\|e(t)\|^3$

$$(\bar{x}(\tau), \bar{u}(\tau)) \in \overset{\text{Obstacles}}{\mathcal{X}(\tau)} \times \overset{\text{Actuator limits}}{\mathcal{U}(\tau)}$$

System constraints

$\mathcal{T} = \{t_0, t_1, \ldots\}$

$t = t_{i+1}$



3. Iterate.

Model Predictive Control

Model Predictive Control

Performance index
(control objective)

$$J_T(t, z, \bar{\boldsymbol{u}}) := \int_t^{t+T} l(\tau, \bar{x}(\tau), \bar{u}(\tau))d\tau + m(t+T, \bar{x}(t+T))$$

Stage cost — Terminal cost

Tracking error $e(t) = R(t)'(c(t) - c_d(t))$

Stage cost $l(t, x(t), u(t)) = \|e(t)\|^2$

Terminal cost $m(t, x(t)) = 0.333\|e(t)\|^3$

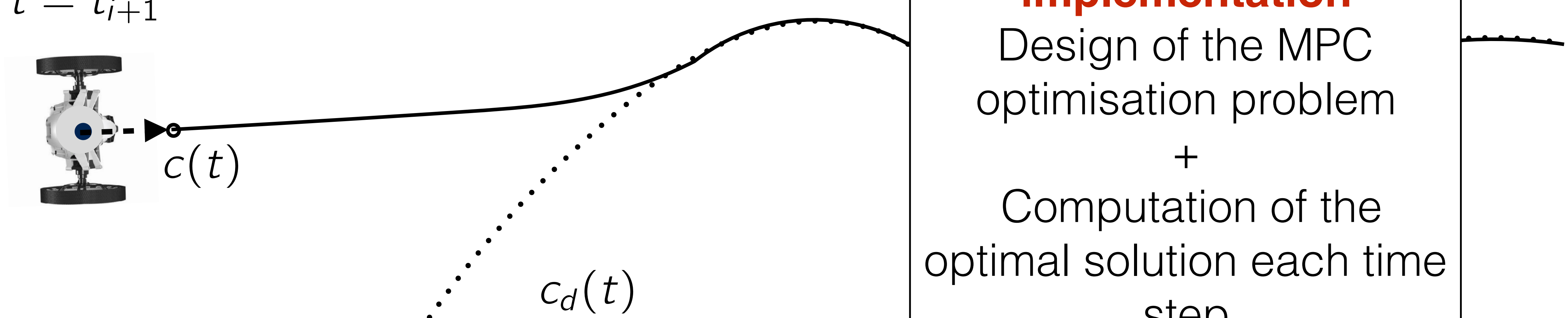
Obstacles — *Actuator limits*

$$(\bar{x}(\tau), \bar{u}(\tau)) \in \mathcal{X}(\tau) \times \mathcal{U}(\tau)$$

System constraints

$\mathcal{T} = \{t_0, t_1, \ldots\}$

$t = t_{i+1}$



Challenging implementation
Design of the MPC optimisation problem
+
Computation of the optimal solution each time step
(+ warm starting, …)

Model Predictive Control

Performance index *(control objective)*

$$J_T(t,z,\bar{\boldsymbol{u}}) := \int_t^{t+T} l(\tau,\bar{x}(\tau),\bar{u}(\tau))d\tau + m(t+T,\bar{x}(t+T))$$

Stage cost *Terminal cost*

Open-loop MPC problem

$$J_T^*(t,z) = \min_{\bar{\boldsymbol{u}}\in\mathcal{PC}(t,t+T)} J_T(t,z,\bar{\boldsymbol{u}})$$

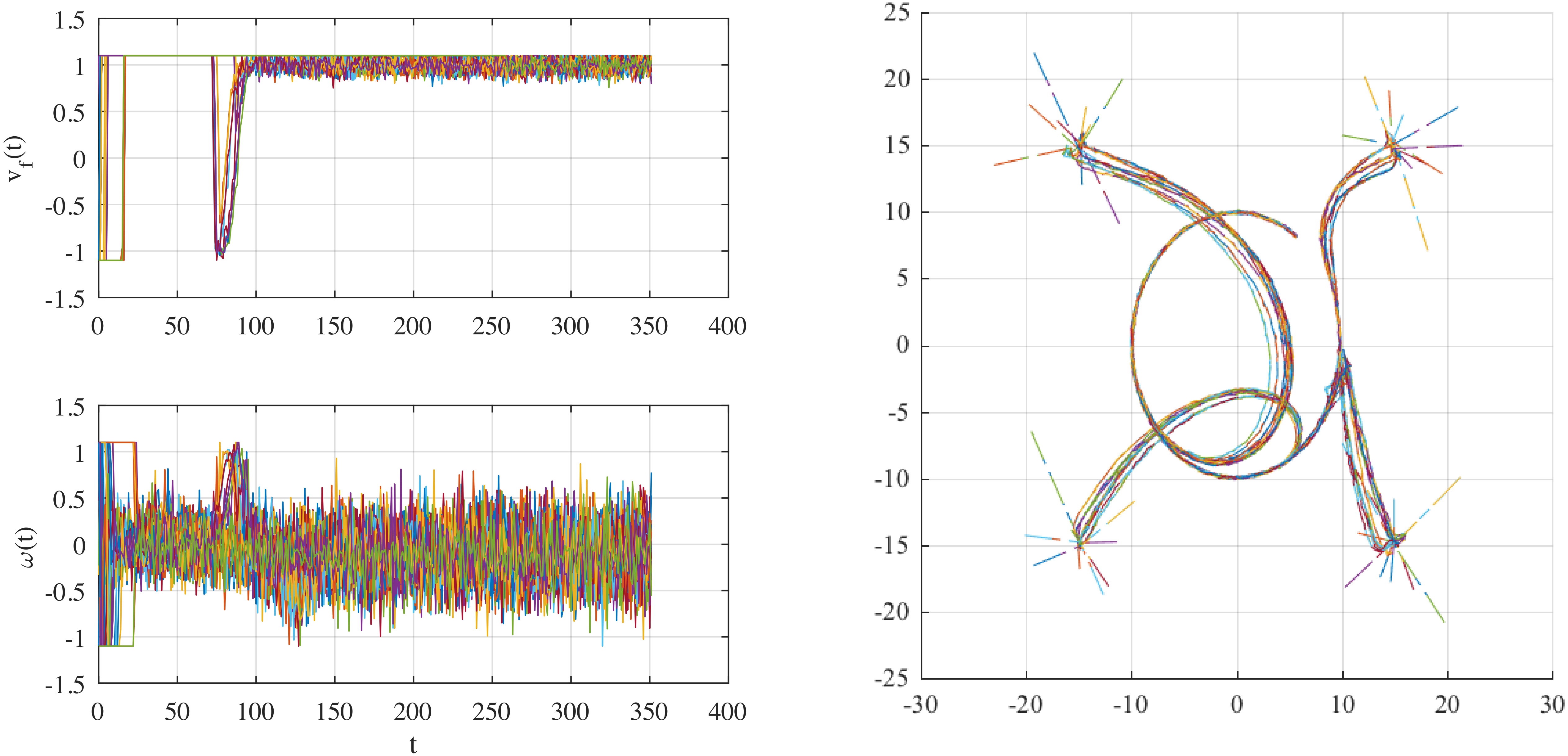
$$\begin{aligned} \text{s.t.} \quad & \dot{\bar{x}} = f(\tau,\bar{x},\bar{u}) \\ & (\bar{x}(\tau),\bar{u}(\tau)) \in \mathcal{X}(\tau)\times\mathcal{U}(\tau) \\ & \bar{x}(t) = z \\ & \bar{x}(t+T) \in \mathcal{X}_{aux}(t+T) \end{aligned}$$

```
mpcOp = ICtMpcOp( ...
 'System'              , sys,...
 'HorizonLength'       , 2*dt,...
 'StageConstraints'    , ...
    BoxSet( -[1.1;1.1],4:5,[1.1;1.1],4:5,5),...
 'StageCost'           , ...
    @(t,x,u,varargin) e(t,x)'* e(t,x),...
 'TerminalCost'        , ...
    @(t,x,varargin) 0.3*(e(t,x)'*e(t,x))^(3/2)...
);
```

```
dtMpcOp      = DiscretizedMpcOp(mpcOp,dt);
dtRealSystem = DiscretizedSystem(realSystem,dt);

dtRealSystem.controller = MpcController(...
    'MpcOp'      , dtMpcOp ,...
    'MpcOpSolver', FminconMpcOpSolver('MpcOp',dtMpcOp)...
);
```

- MPC discretisation
- MPC solver

Model Predictive Control



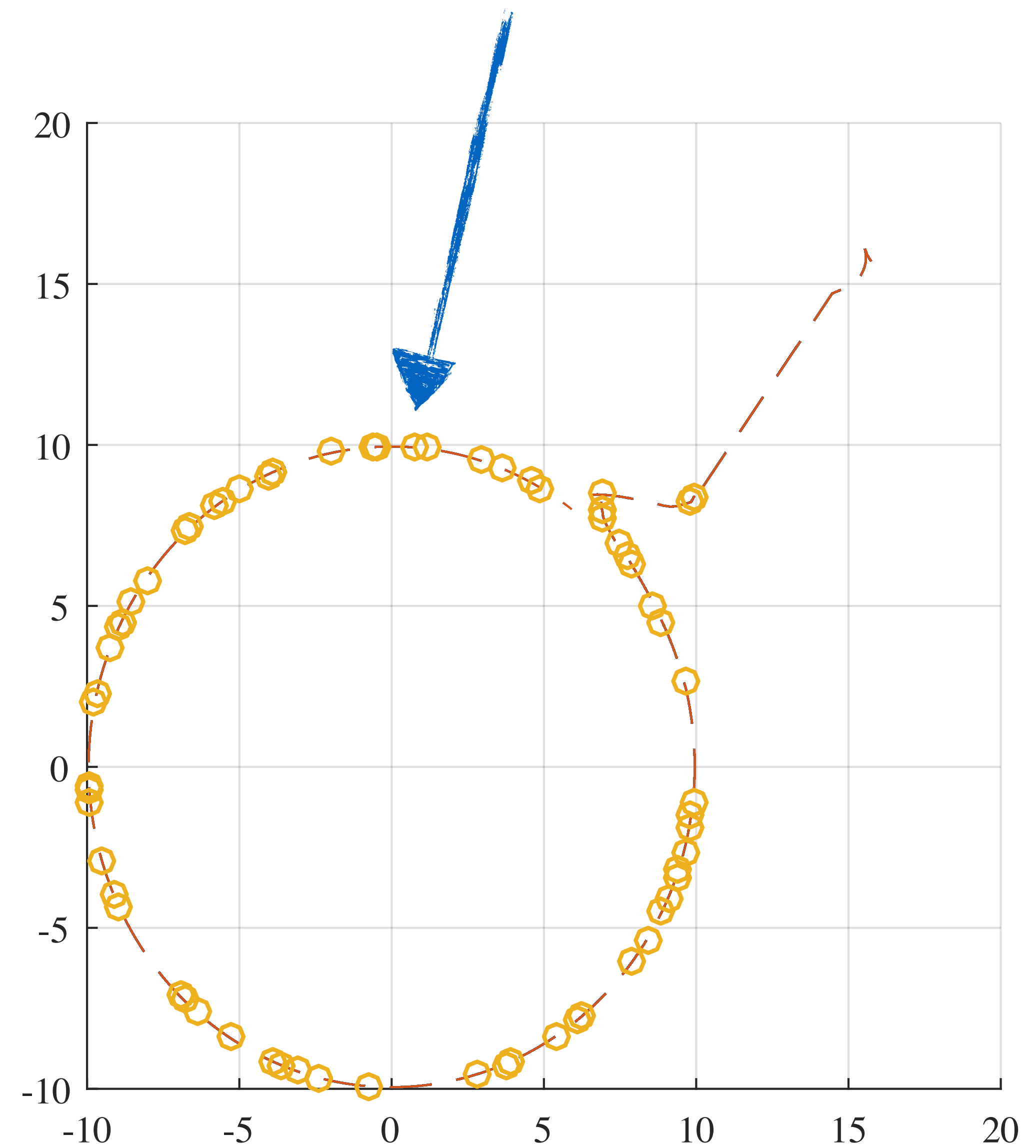
Real system > networked control

```
realSystem = ex04RemoteUnicycle(...
     'RemoteIp','127.0.0.1',...
     'RemotePort',20001,...
     'LocalIp' ,'127.0.0.1',...
     'LocalPort',20002);
```

UDP Link
(e.g. Wifi, Phones, ...)

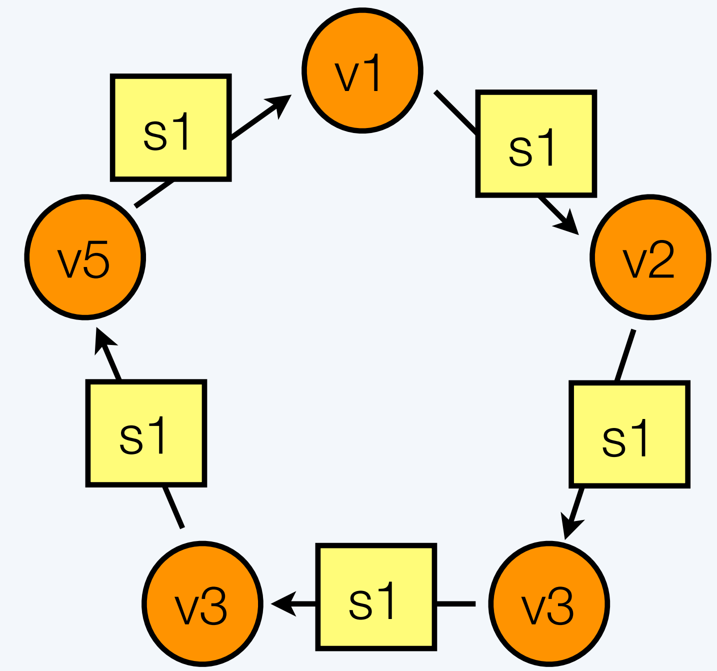
```
#Python loop
u = readFromSystem()
sendToServo(u)
x = readSensors()
sendToSystem(x)
```

Simulating communication loss



Multi-agent systems > consensus example

```
N = 5;
for i = 1:N
    v{i} = ICtSystem('StateEquation',...
        @(t,x,u,varargin)u,'nx',1,'nu',1);
    v{i}.controller = ex05BasicConsensusController();
    v{i}.initialCondition = i;
end
```



Multi-agent system

Network topology

```
%% Network
A = zeros(N); A(1,4)= 1;A(2:N,1:N-1) = eye(N-1);% Adjacency matrix - loop

s1 = IAgentSensor(@(t,agentId,agent,sensedAgentIds,sensedAgents)sensedAgent.x);
```

Network measurements

```
a = VirtualArena(v,...
    'StoppingCriteria'   , @(t,as)t>10,...
    'SensorsNetwork'     , {s1,@(t) A},...
    'DiscretizationStep', 0.1);

ret = a.run();
```

ex05BasicConsensusController

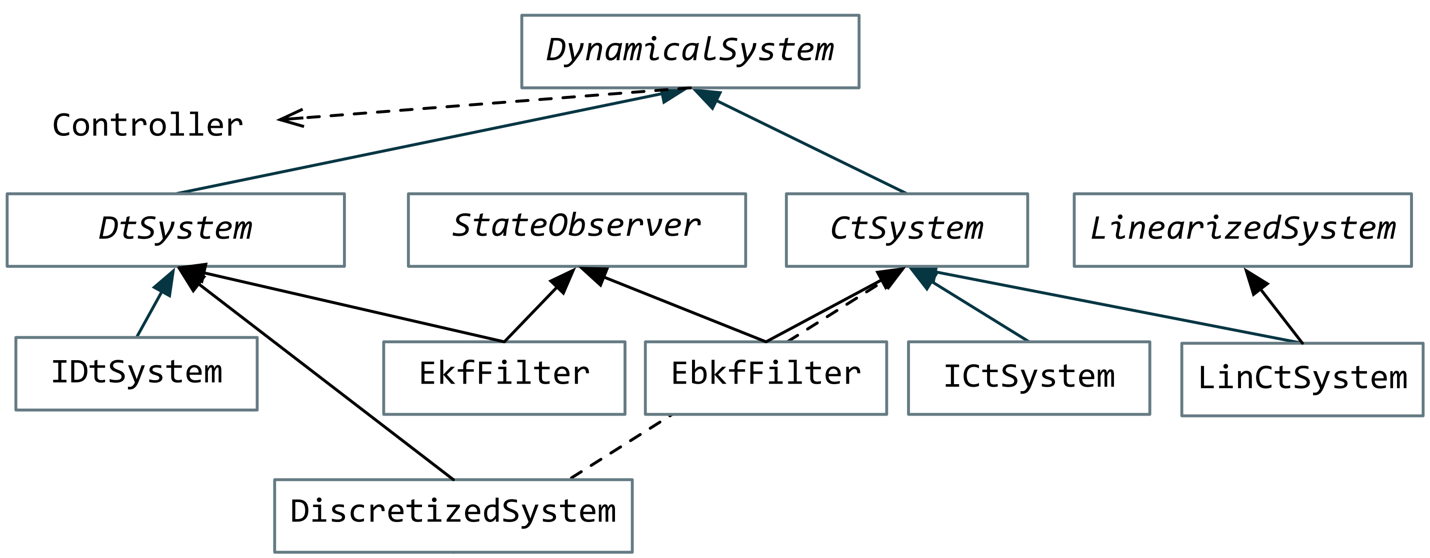
```
function u = computeInput(obj,t,x,readings)
    nNeigh = length(readings{1});
    u = 0;
    for i =1:nNeigh
        u = u+(readings{1}{i} - x)/nNeigh;
    end
end
```

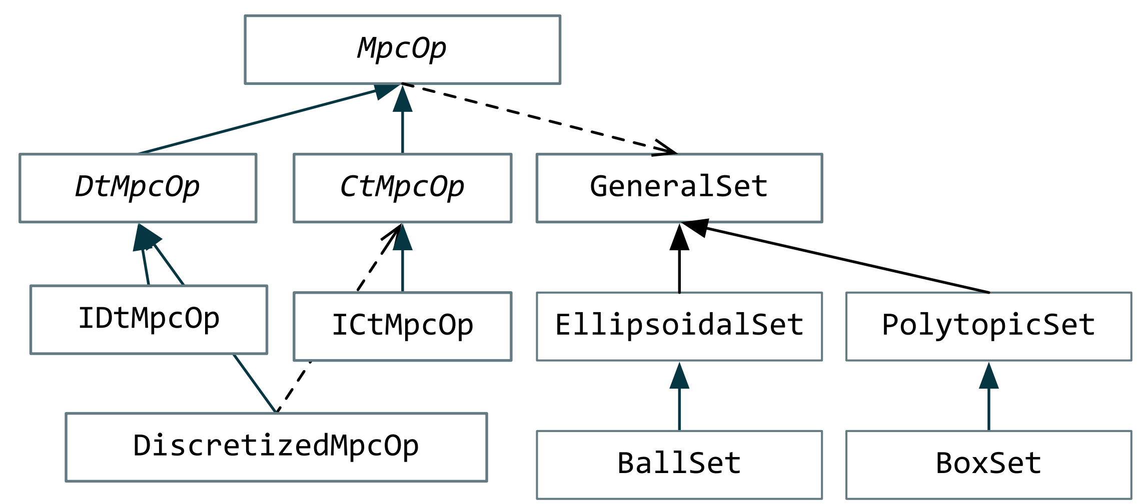
Key benefits and main features

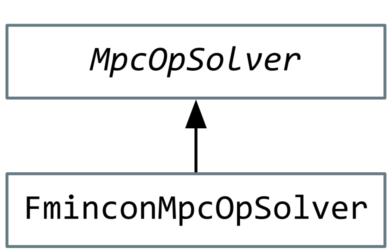
Modularity, reusability, and maintenance

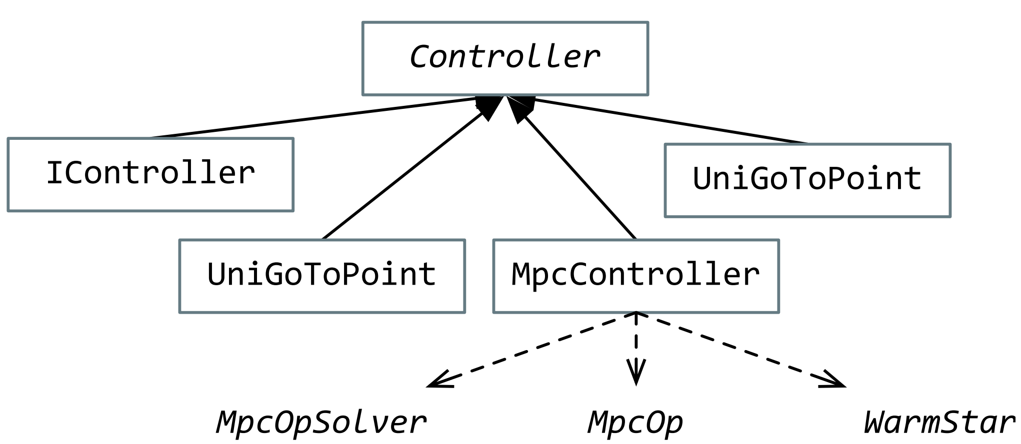
Pre-defined interfaces
(**object classes**)

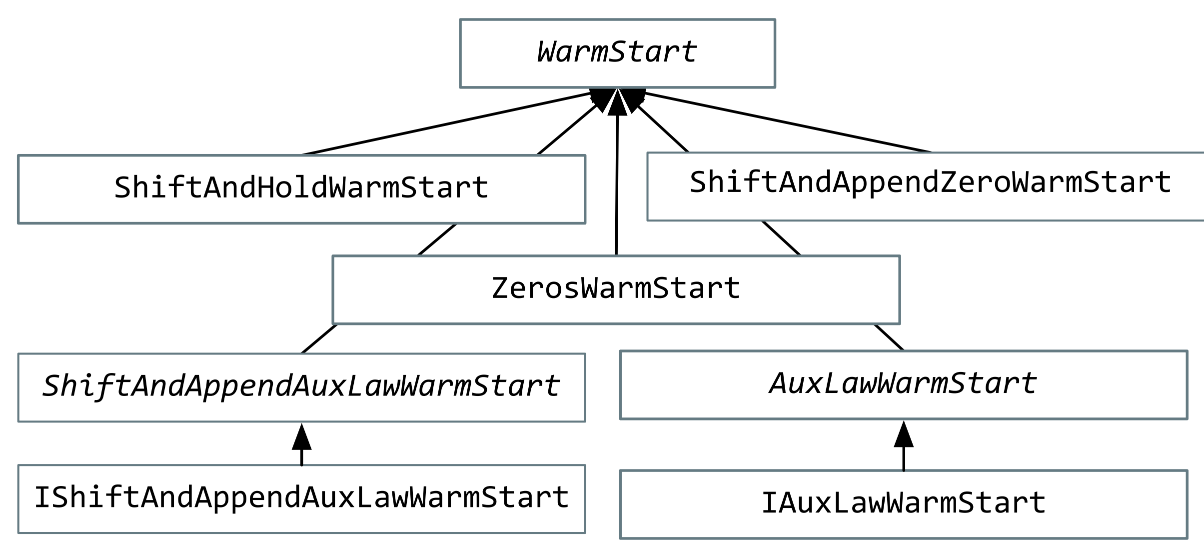
Components that are:
- self-contained
- interchangeable
- sharable











Key benefits and main features

Dissemination and extensibility

Main obstacles for dissemination of new technologies:

- theoretical background requirements (*understanding*)
- advanced control strategies (*implementation*)

```
vaInstall(URL)
```

Ready-to-use functionalities

Simulation: Time discretization methods, logging management system, multiple simulations, simultaneous simulation of a network of multiple vehicles

System definition and manipulation: discretization, linearized, (computation of the jacobian matrices via Symbolic MATLAB or via sampling)

State estimation: Automatic generation of Extended Kalman Filter, Extended Kalman-Bucy Filter, support for custom observers

Model predictive control: Definition of continuous-time and discrete-time MPC optimization problem, definition of abstract class for MPC solver and warm-start strategies, discrete-time MPC solver using fmincon

Motion control of underactuated vehicle: Different representations of attitude using quaternions and rotation matrices available in UnderactuatedVehicle

~~Conclusion~~ Getting started

1) Install VA from: github.com/andreaalessandretti/VirtualArena
2) Look at the examples on /examples
3) Implement your components
4) Share
5) Contribute to VirtualArena!

or simply copy and paste this code on Matlab

```
urlwrite('https://github.com/andreaalessandretti/VirtualArena/archive/master.zip','master.zip');
unzip('master.zip');
movefile('VirtualArena-master','VirtualArena');
cd VirtualArena/;
addPathsOfVirtualarena;
```