Getting Started With QKS

Prof. Federico Mari
Computer Science Department

MCLab
http://mclab.di.uniroma1.it/

SAPIENZA
UNIVERSITÀ DI ROMA

Ph.D. Programme "Automatica, Bioengineering and Operations Research"
Course on "Hybrid systems: Computation and Control"

September 12–22, 2018
Department of Computer, Control, and Management Engineering
Sapienza University of Rome (Italy)

Software

QKS developed in C
Linux binary distribution

https://bitbucket.org/mclab/qks/

Software

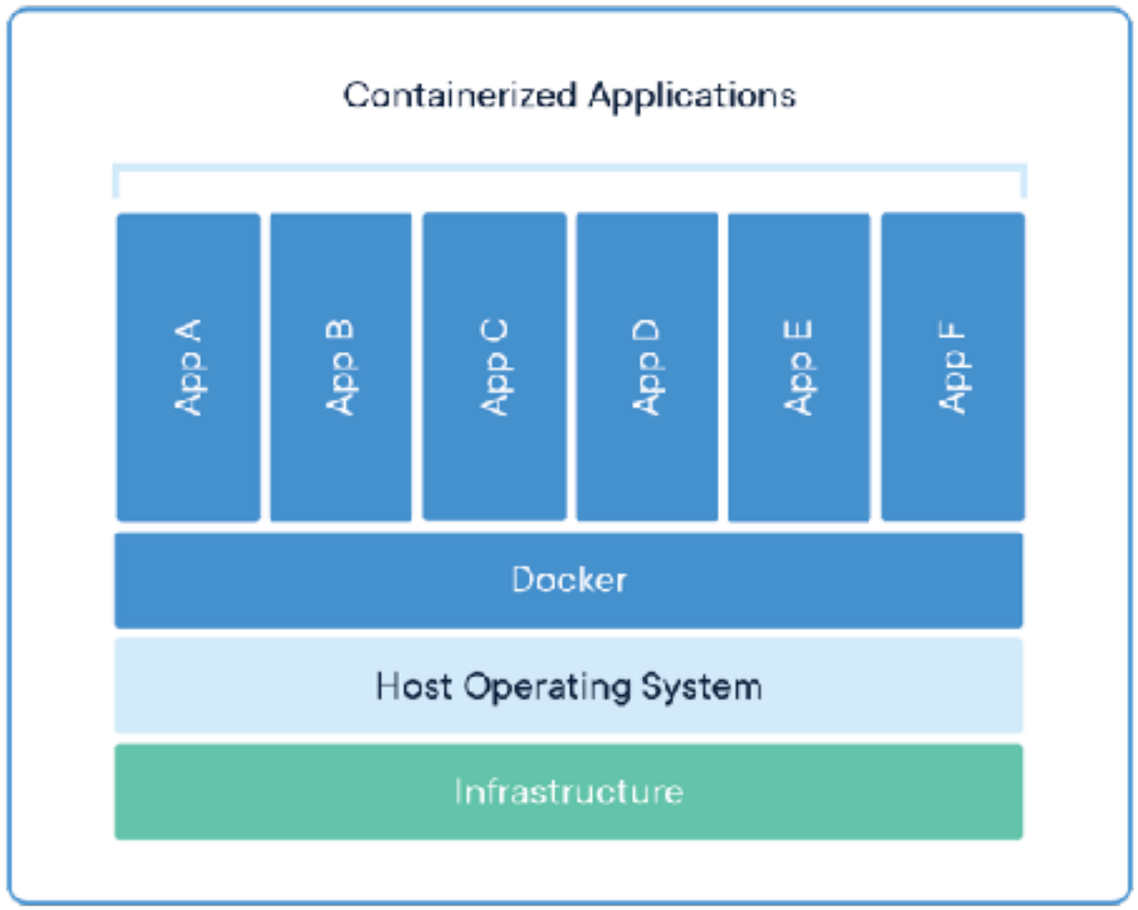
QKS Demo for learning purposes Docker container

mclab/qks

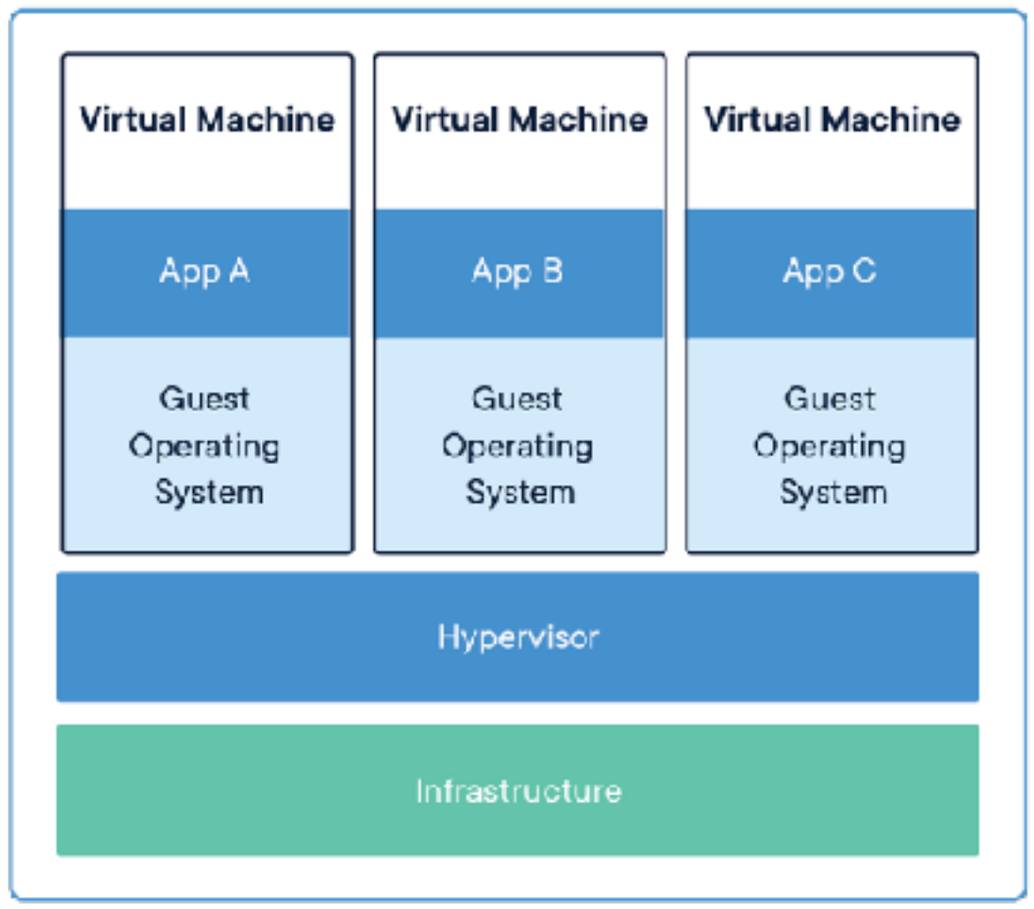
What is a Docker Container

A standardized unit of software

Containers



Virtual Machines



Credits: docker.com

Docker QKS Demo

- Download **QKS docker image**:

```
$ docker pull mclab/qks-demo
```

- Run it, this will create a **docker container**:

```
$ docker run --rm -it -v $PWD:/home/qks/work mclab/qks-demo
```

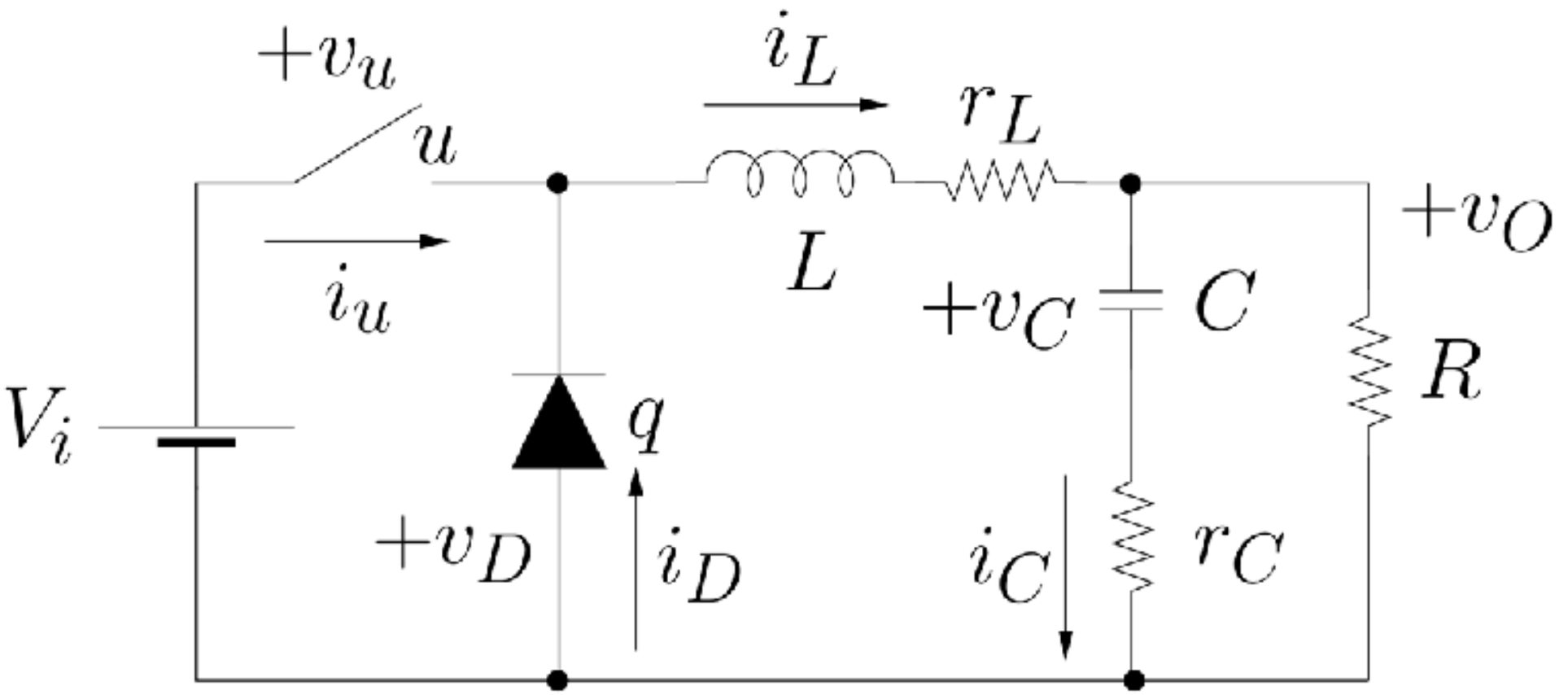
- Current working directory **`$PWD`** will be mapped to path **`/home/qks/work`** inside container

- The working directory inside container after starting it is **`/home/qks`**

Credits: Vadim Alimguzhin, Post-doc at MCLab

Modelling with QKS

Buck DC-DC converter



`/home/qks/examples/buck.m`

Modelling with QKS: Syntactic Sugar

- Constants

```
Const
  T: 1e-6; // sampling time
  ...
  lb_iL: -4; // lower bound for state variable iL
  ub_iL: 4;  // upper bound for state variable iL
```

- Types

```
Types
  iL_interval: [lb_iL, ub_iL]; // real valued range
  boolean: 0 .. 1; // integer valued range
  ...
```

Modelling with QKS: Variables

- State (`X` component of DTLHS tuple)

```
Statevars // state variables must have real types
  iL: iL_interval;
  v0: v0_interval;
```

- Input (`U` component of DTLHS tuple)

```
Inputvars // input variables must have integer types
  u: boolean;
```

- Output

```
Outputvars // output variables must have integer types
  yiL: lb_yiL .. ub_yiL;
  yv0: lb_yv0 .. ub_yv0;
```

Modelling with QKS: Predicates

- Transition relation (`N` component of DTLHS tuple)

```
Trans
  Exists
    // auxiliary variables
    q: boolean,
    iD: [-1000, 1000],
    ...
  {
    iL' = (1 + T * a_11) * iL + (T * a_12) * v0 + (T * a_13) * vD;
    ...
  };
```

- Desired controllable region (`I` component of DTLHS tuple)

```
ControllableRegion
  init_lb_iL <= iL & iL <= init_ub_iL;
  init_lb_v0 <= v0 & v0 <= init_ub_v0;
```

- Goal region (`G` component of DTLHS tuple)

```
Goal
  goal_lb_iL <= iL & iL <= goal_ub_iL;
  goal_v0_ref - goal_v0_tol <= v0 & v0 <= goal_v0_ref + goal_v0_tol;
```

Modelling with QKS: Quantization

- Observation relation

Observability introduced in DTLHS formulation for the on-the-fly algorithm

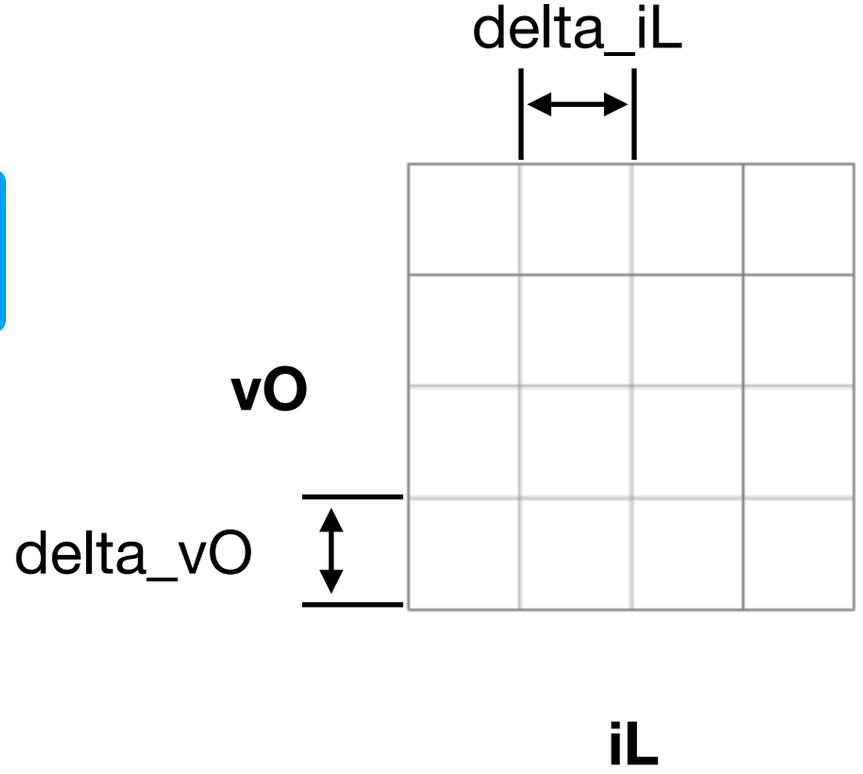
```
Const
  ...
  // AD conversion with 8 bits
  lb_yiL: 0;
  ub_yiL: 255; // 2^8 - 1
  lb_yv0: 0;
  ub_yv0: 255; // 2^8 - 1
  delta_iL : (ub_iL - lb_iL) / (ub_yiL - lb_yiL + 1);
  delta_v0 : (ub_v0 - lb_v0) / (ub_yv0 - lb_yv0 + 1);
```

Start with few bits (7, 8, …)

```
…

Observation
  Exists
    diL: [0, delta_iL],
    dv0: [0, delta_v0]
  {
    iL = lb_iL + delta_iL * yiL + diL;
    v0 = lb_v0 + delta_v0 * yv0 + dv0;
  };
```

quantization



Running QKS

```
$ cd work
$ mkdir input output
$ cp ../examples/buck.m input/model.m
$ qks -d input -dk output -v 2 # original algorithm
$ qks -d input -dk output -v 2 -np 4 # parallel algorithm with 4 processes
$ qks -d input -dk output -v 2 -non_mgo # small controller
$ qks -d input -dk output -v 2 -on_the_fly # on-the-fly synthesis
```

-v sets the verbosity level

Input directory with file model.m.
Different file names with -I non_std_name.m

Output directory

Options for different QKS variants.
Default is the original algorithm

QKS output

- Controller OBDD:

  `ctrl.blif`

- Controller C code:

  `ctrl.h`
  `ctrl.c`

- Gnuplot file generating controllable region picture:

  `ctrl_reg.strong_ctrl.gnuplot`

Note: QKS for generating small software will produce ctrl.macro.[ch] instead

Simulating Buck DC-DC Converter

```
$ cp -r ../sim/buck sim
$ cd sim
$ cp ../output/ctrl.[ch] Resources/Include/
$ omc run.mos # evolution of variables will be put into ClosedLoop_res.csv
```

Open Modelica Compiler
Closed loop model: closed_loop.mo
Plant model: plant.mo

Simulate both strong and weak controllers

Now You Try

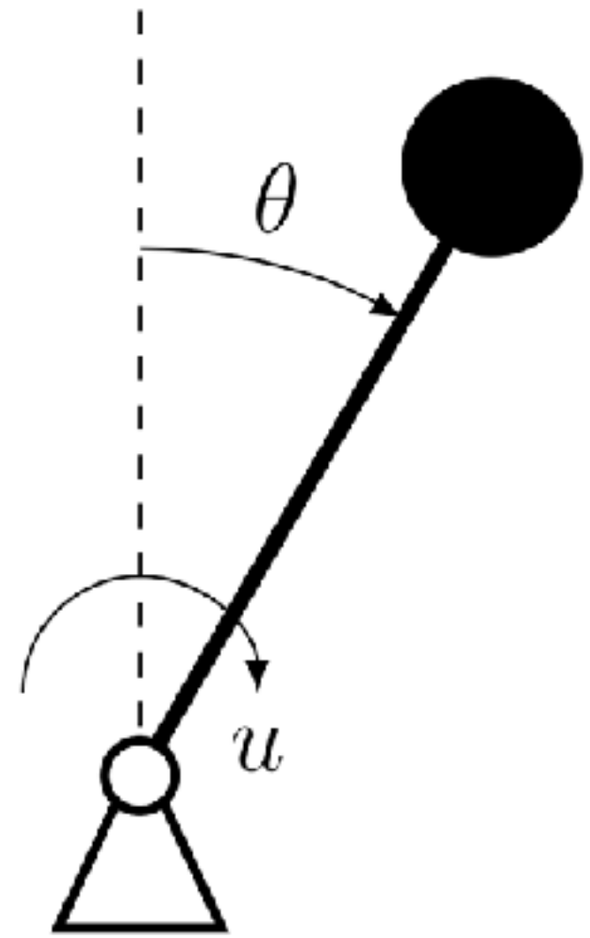
Too big: computation time very large
Too small: quantization precision very low

- Try to change number of **quantization bits** and **sampling time** to find different controllers

Too small: control software may not have time to execute
Too big: control software could not succeed in controlling the closed loop

Modelling non-linear systems

Inverted Pendulum with fixed pivot



`/home/qks/examples/pendulum.[mc]`

Modelling non-linear systems

- Additional section with non-linear functions declaration

```
Functions
  sin : 1 : (2 * pi); // function of 1 argument and period 2 * pi
```

If nonlinear function is non periodic it is “nonlin : 1;”

- Additional C input file with function definitions together with corresponding definitions of Lipschitz constant estimators.

```
~~double sin(double x) {...}~~
double sin_lipschitz(double a, double b) { return 1; }
```

No need to define sin since it is a standard C function

- Declared functions can be used in predicate definitions

```
x2' = x2 + T * ((g / l) * sin(x1, 0.1) -
      (mu / (m*l*l)) * x2 + (1 / (m*l*l)) *  mult_u * u);

// the second “argument” of sin is maximum allowed linearisation error
```

More derails in https://bitbucket.org/mclab/linearizer-benchmark

lin4qks | QKS

Tool chain: first linearize then use QKS on linearized model

lin4qks linearizes for QKS

QKS non-linear model

```
$ cd work
$ mkdir input output
$ lin4qks -i ../examples/pendulum.m -f ../examples/pendulum.c -o input/model.m -m 100 -v 2
$ qks -d input -dk output -v 2
```

C definition of non-linear and Lipschitz functions

QKS input linearized model

Number of sampling points

Simulating Inverted Pendulum

```
$ cp -r ../sim/pendulum sim
$ cd sim
$ cp ../output/ctrl.[ch] Resources/Include/
$ omc run.mos # evolution of variables will be put into ClosedLoop_res.csv
```

Simulate both strong and weak controllers

Now You Try

- Try to change the **linearization error**
- Try to change the **Lipschitz estimator**
- Try to change the number of **sampling points for lin4qks**
- Try to change the **pendulum model** (e.g., torquing force multiplier, …). This requires also the same variation in the Modelica model for simulations.

Real Experiments

- Docker image is just a demo
- For running experiments there is a **linux binary distribution of QKS** available on BitBucket and on MCLab website (software)

https://bitbucket.org/mclab/qks/

http://mclab.di.uniroma1.it/