Reactive Scheduling of Computational Resources in Control Systems

Hodai Goldman

Ben-Gurion University of the Negev
Department of Computer Science

January 1, 2018

1. Overview
2. Automata-based Scheduling
   - Motivation
   - Component-based Architecture
   - Component definition
3. Simulation with Kalman
4. Experiment with real-life case-study
   - Case study
   - Vision Component
   - Simplifying the Kalman Filter with Complementary Filter
   - Test and Results
5. Conclusion
   - Conclusion
   - Related Work

Overview

Contributions

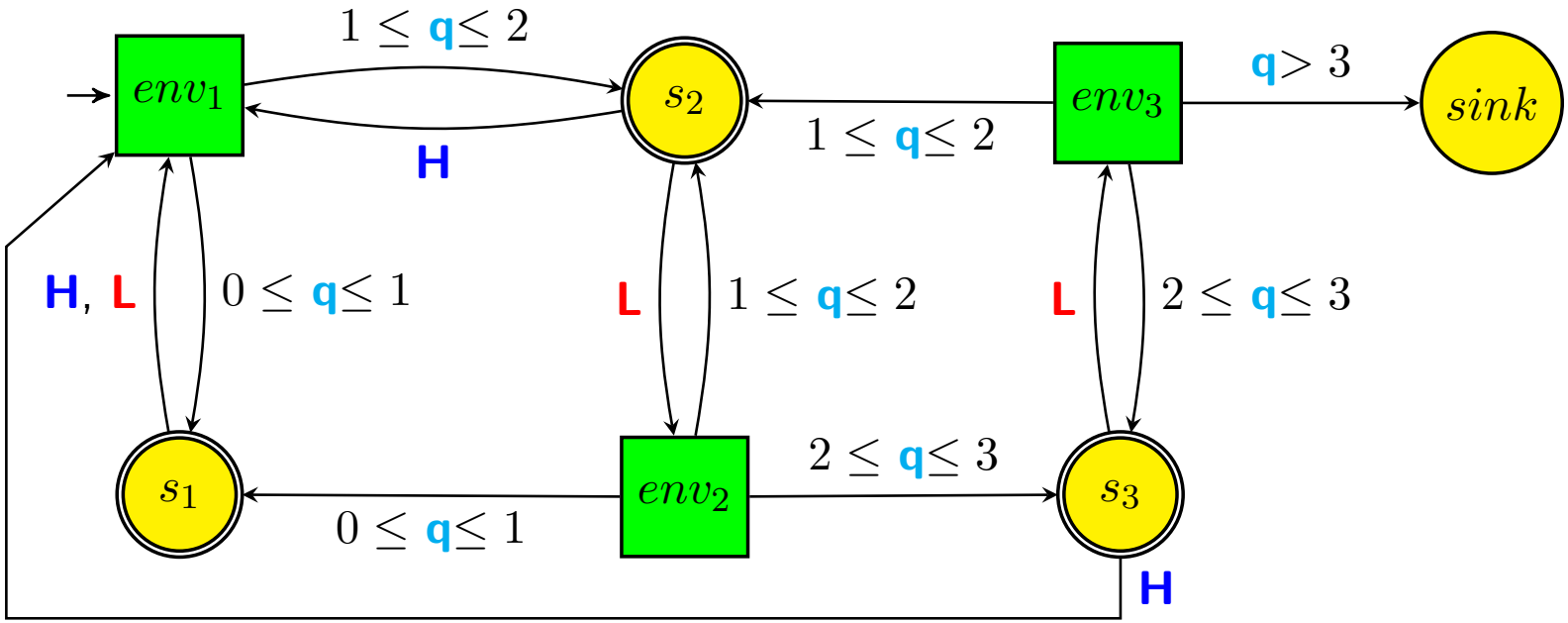
- Development of control and scheduling co-design framework
- **Reactive** scheduling (adaptation)
- **composable** interface (based on automata theory)
- Development of scheduling technique based on **Kalman filter**

Achievements of this thesis

- Extend the work of **RTComposer**
- Proof of concept with **simulation**
- Proof of concept with **real-life case-study**
- **Bridge** between control and software engineering

Example

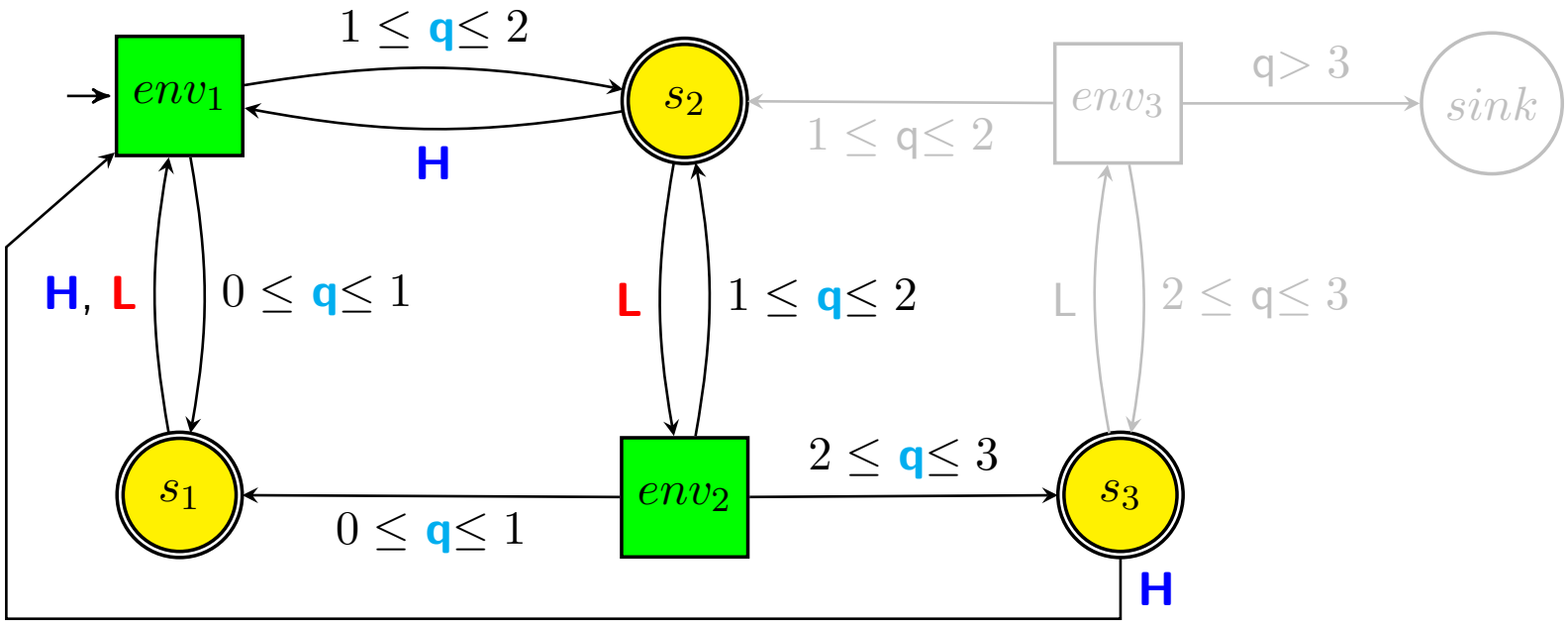
A component that "plays" a game with its environment



- A sensor with two modes: High quality, Low quality.
- After H, the estimation quality (q) is best, after L it may degrade.

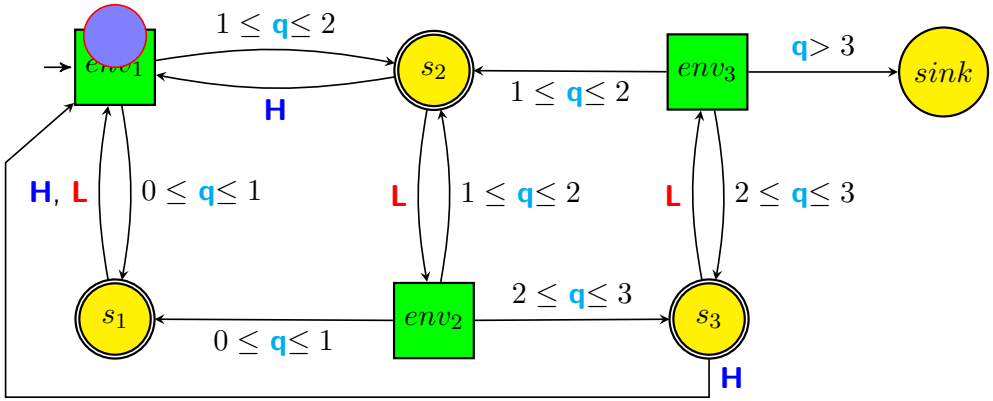
Example

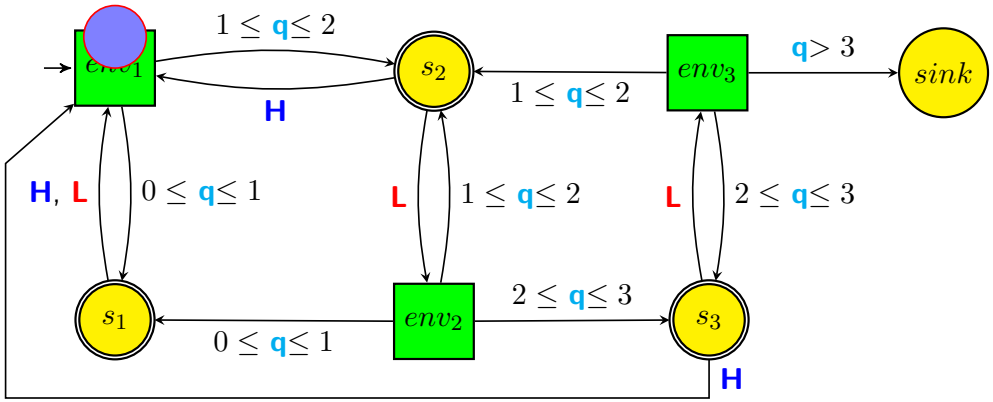
A solution of the game



- We must not schedule **H** when in s_3

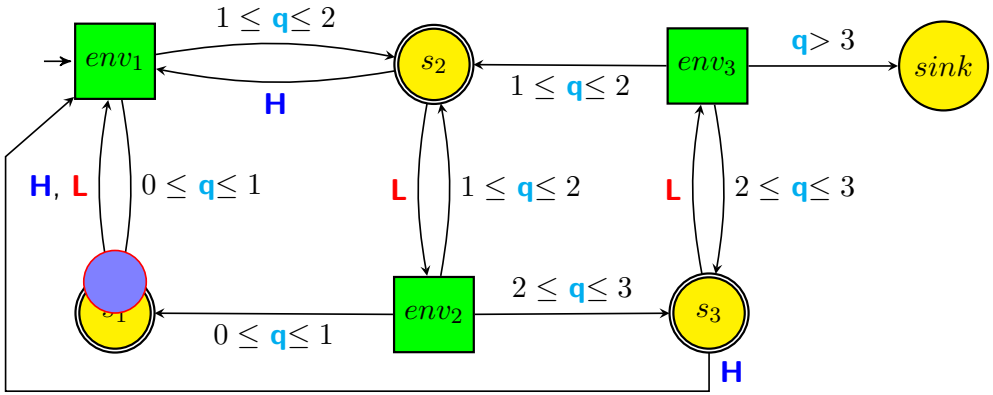
Example
Simultaneous Games (composition)

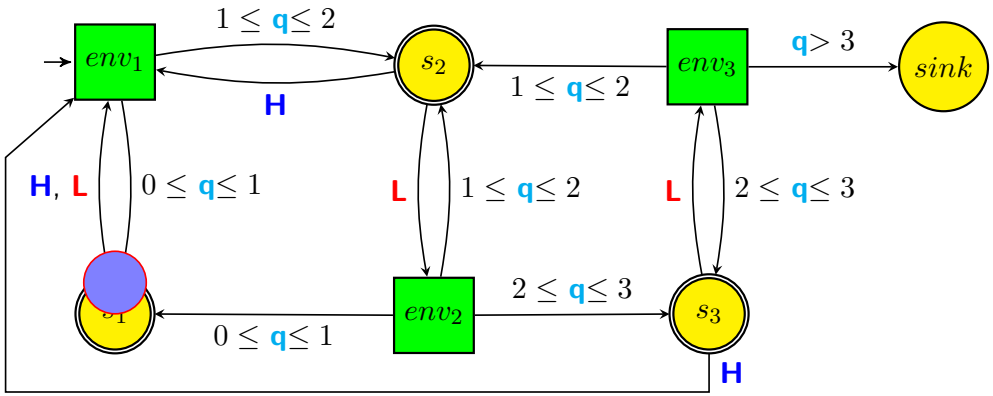




Only one **H** per slot allowed

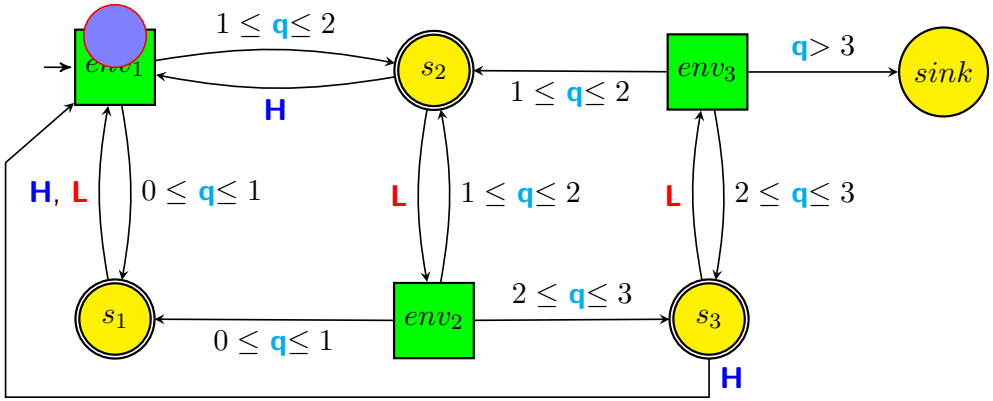
Example
Simultaneous Games (composition)

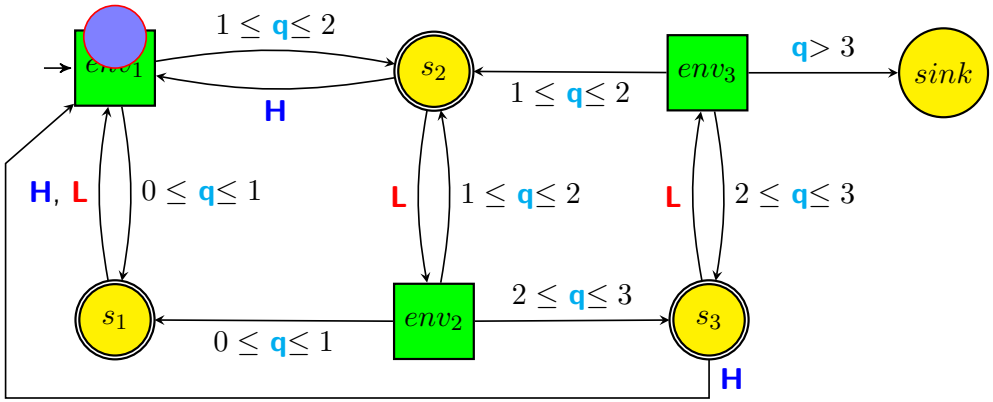




Only one **H** per slot allowed

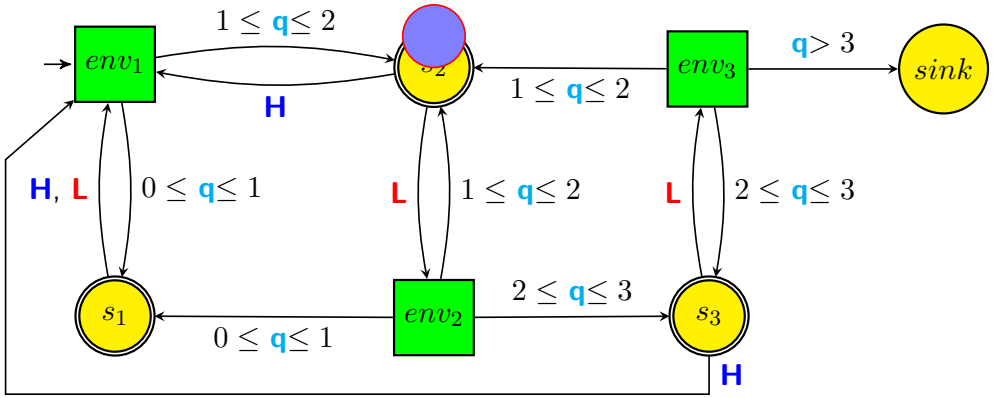
Example
Simultaneous Games (composition)

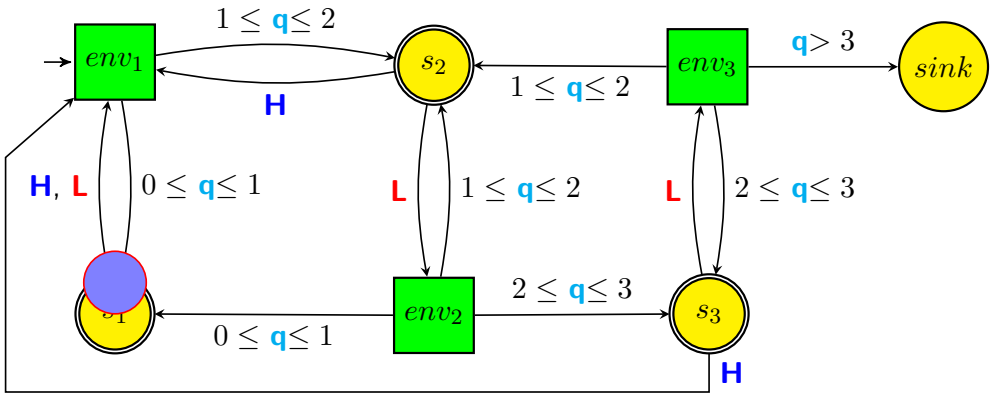




Only one **H** per slot allowed

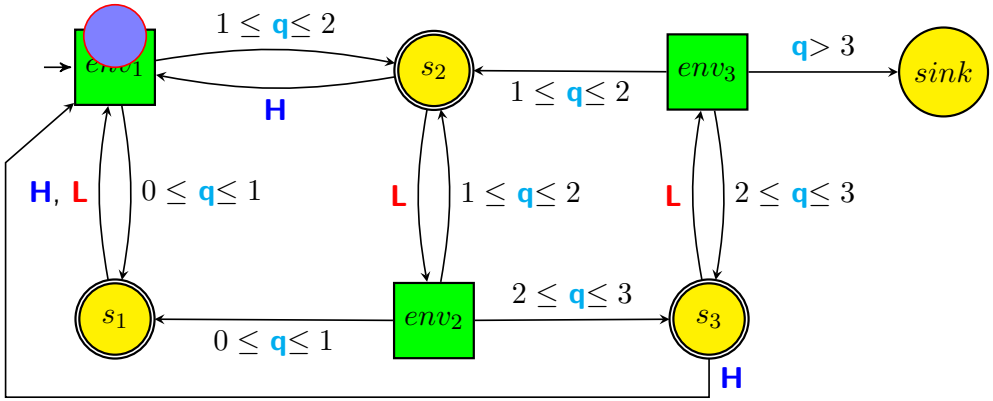
Example
Simultaneous Games (composition)

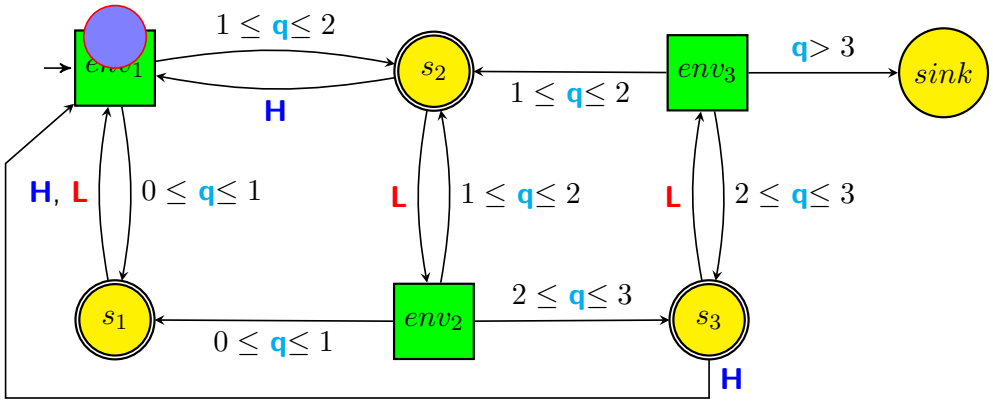




Only one **H** per slot allowed

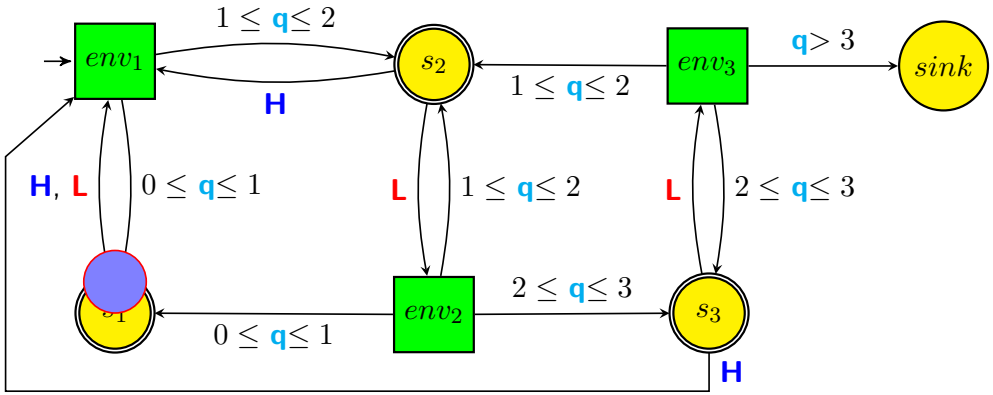
Example
Simultaneous Games (composition)

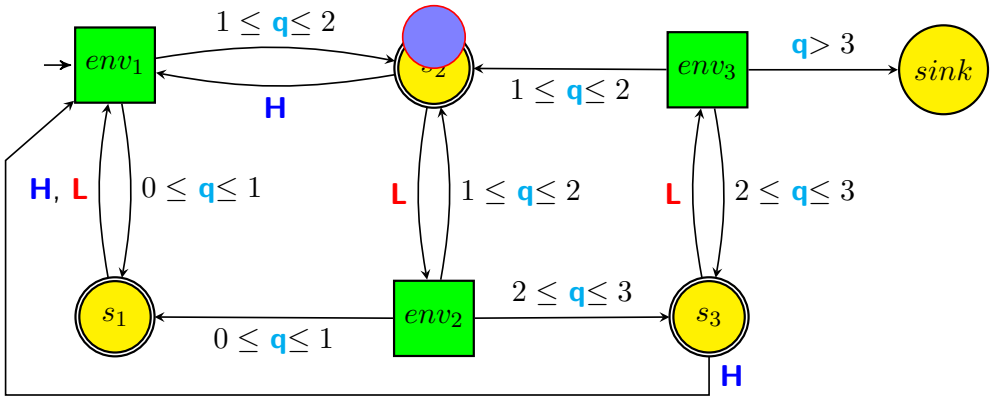




Only one **H** per slot allowed

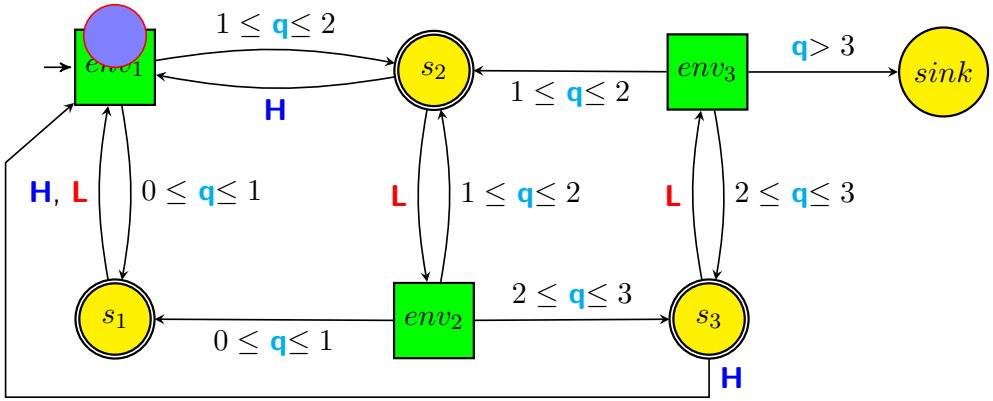
Example
Simultaneous Games (composition)

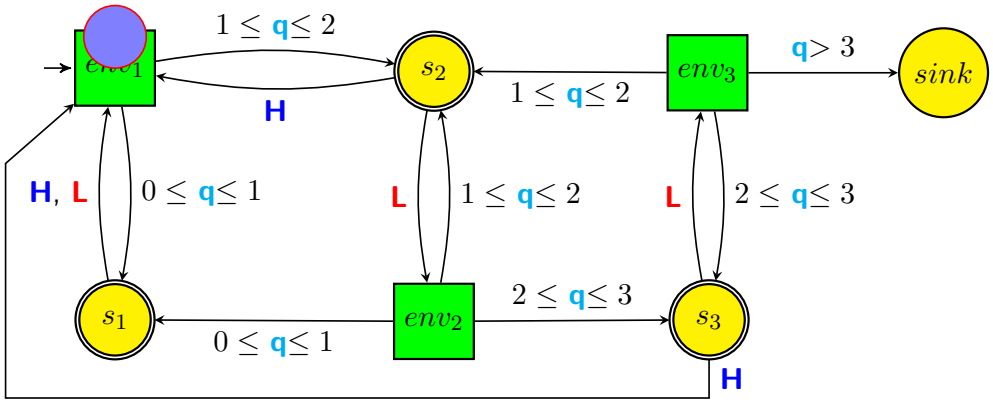




Only one **H** per slot allowed

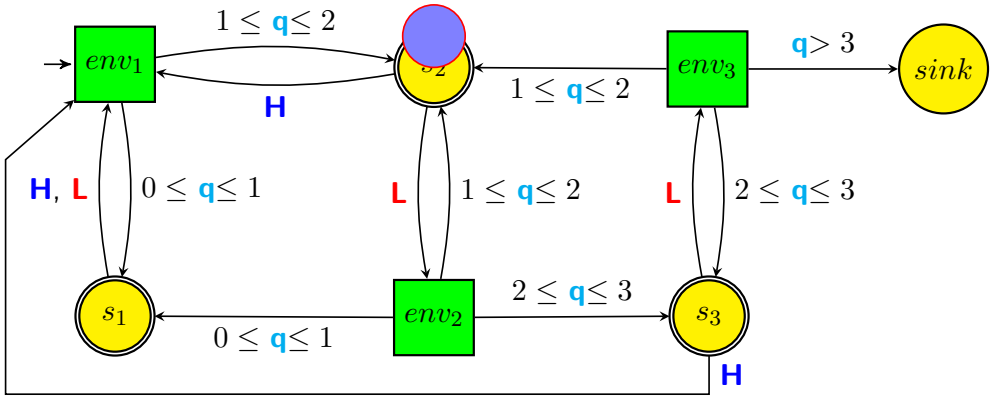
Example
Simultaneous Games (composition)

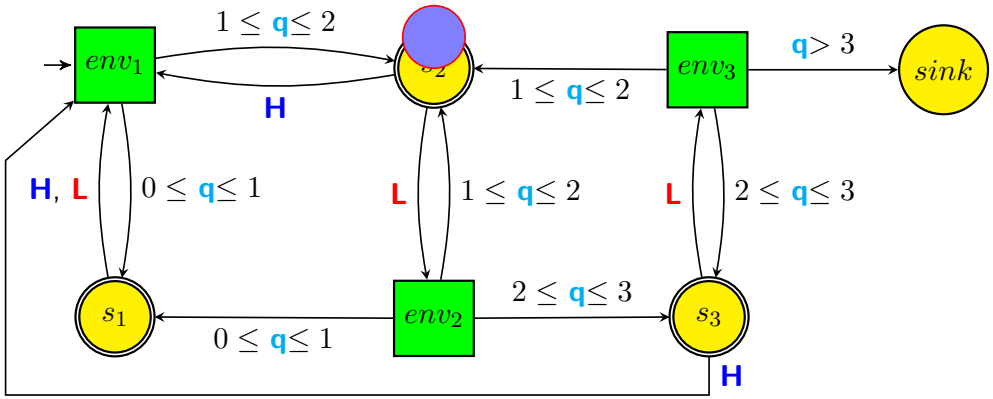




Only one **H** per slot allowed

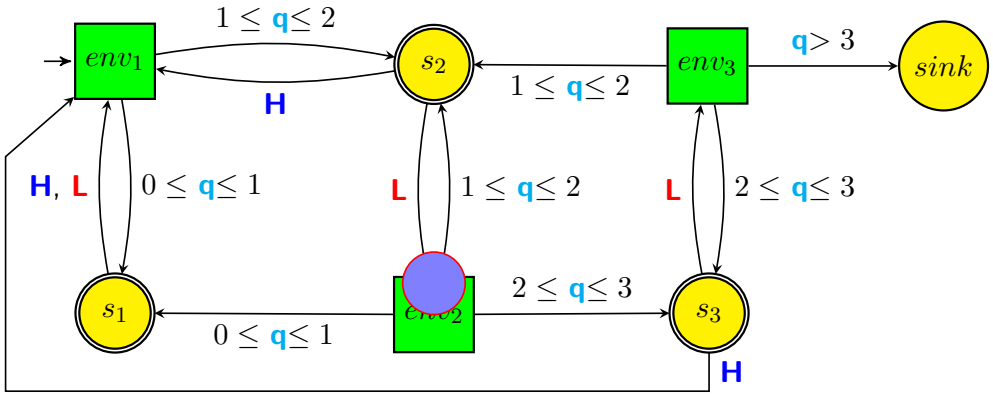
Example
Simultaneous Games (composition)

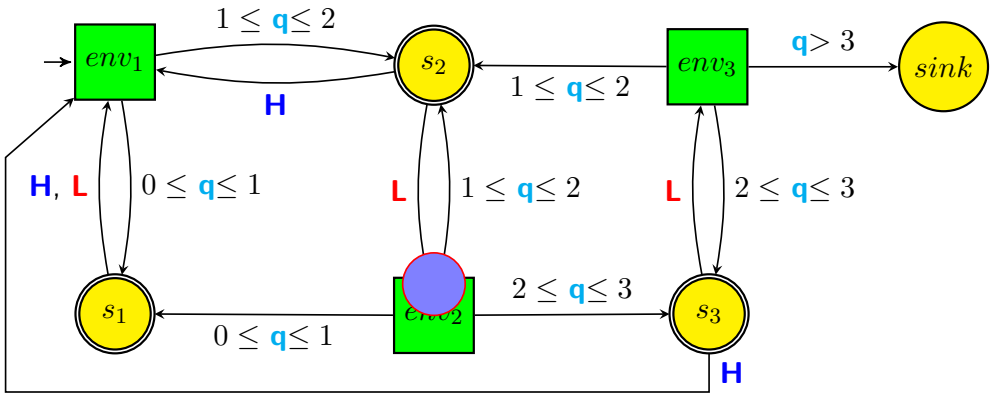




Only one **H** per slot allowed

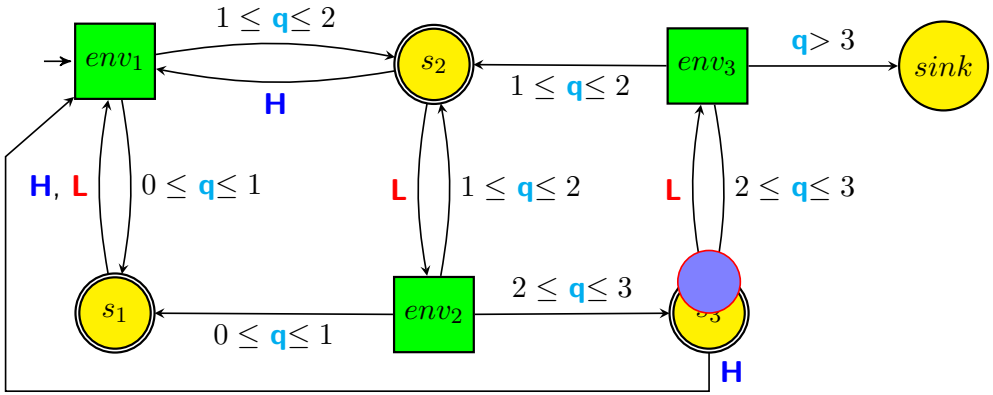
Example
Simultaneous Games (composition)

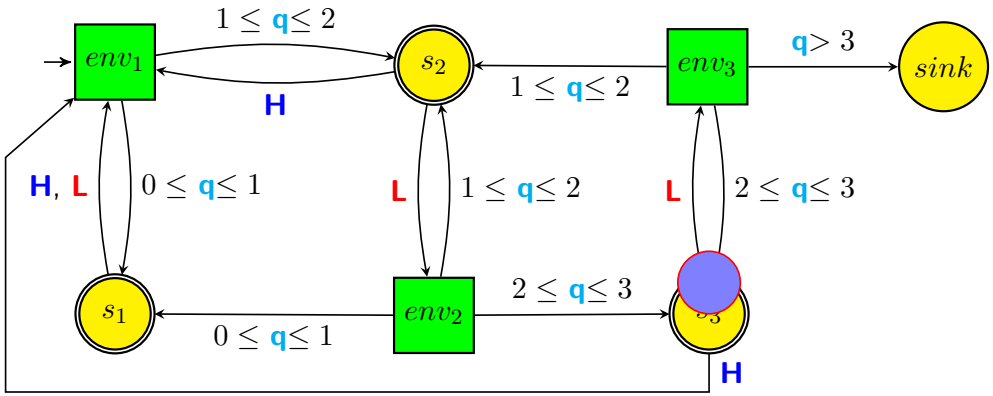




Only one **H** per slot allowed

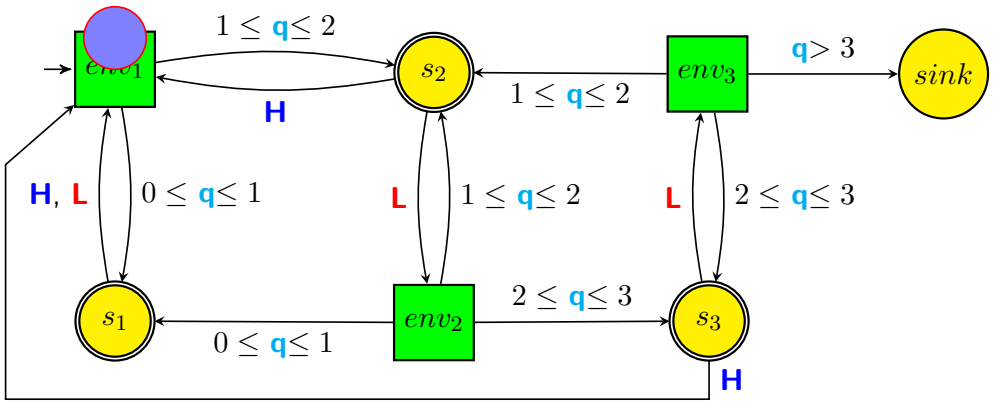
Example
Simultaneous Games (composition)

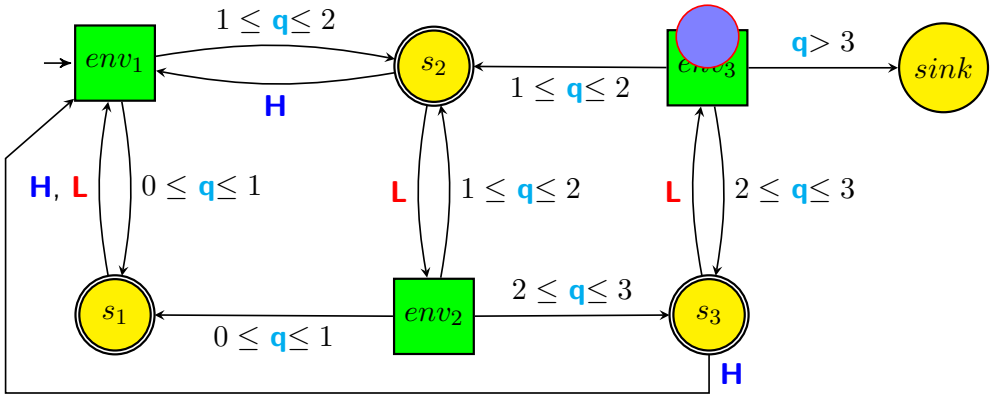




Only one **H** per slot allowed

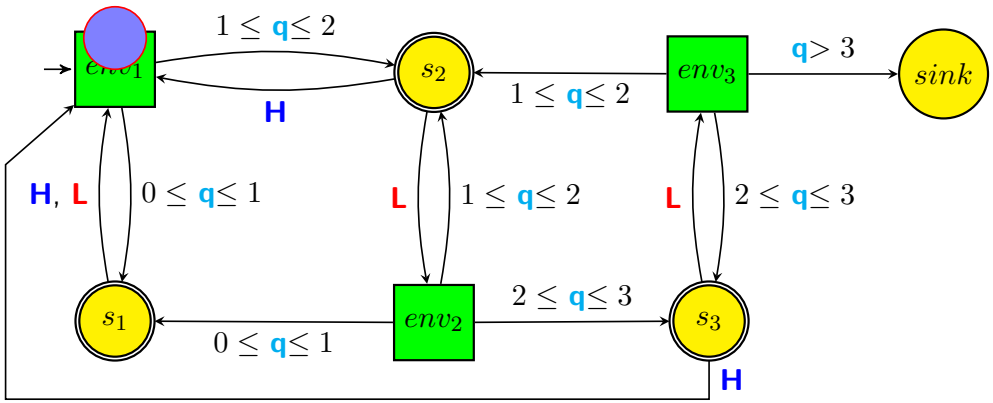
Example
Simultaneous Games (composition)



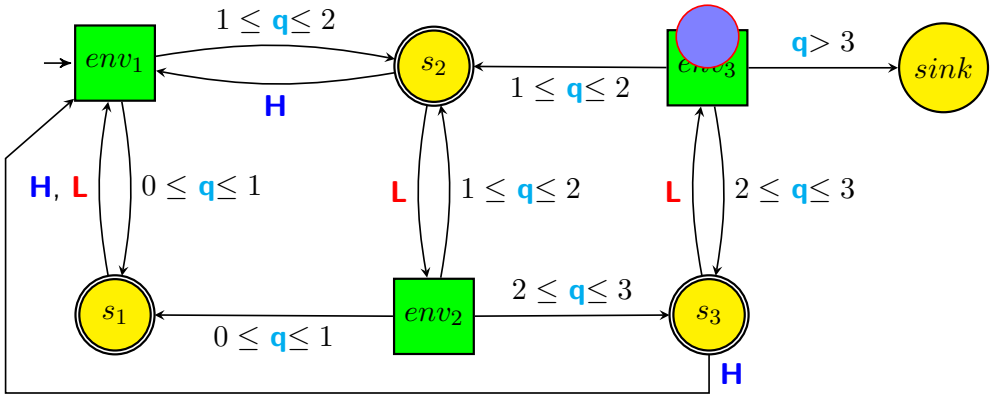


Only one **H** per slot allowed

Example
Simultaneous Games (composition)

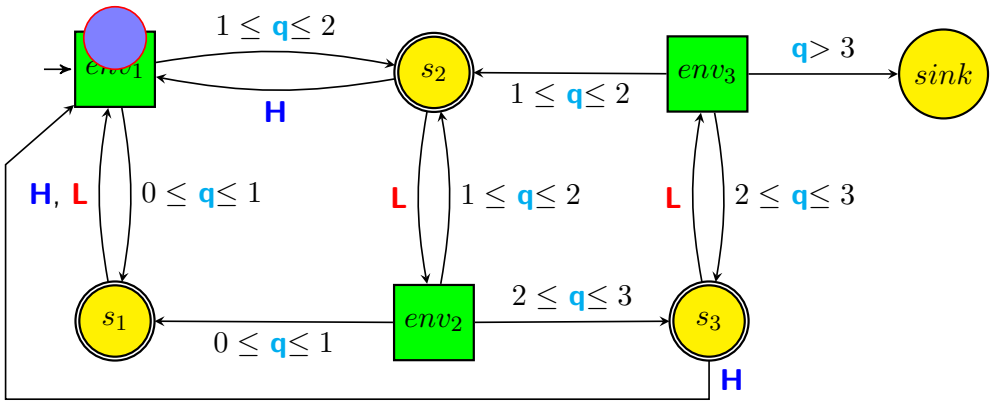


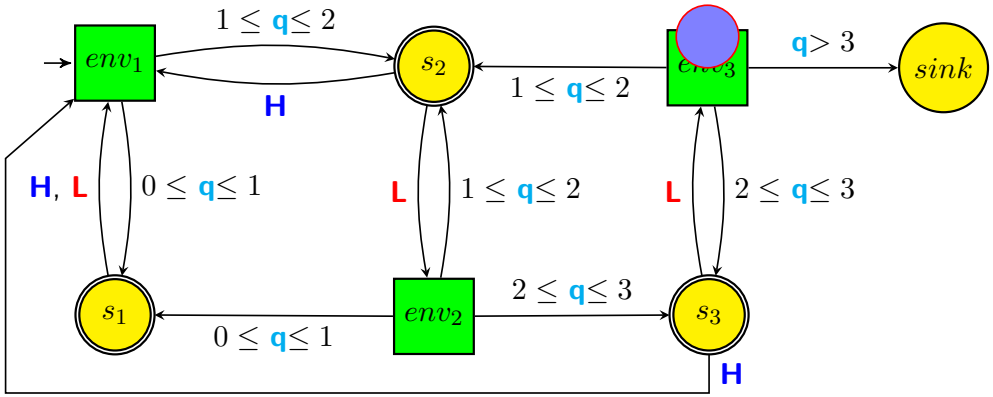
We cannot allow both components to be on env_2



Only one **H** per slot allowed

Example
Simultaneous Games (composition)



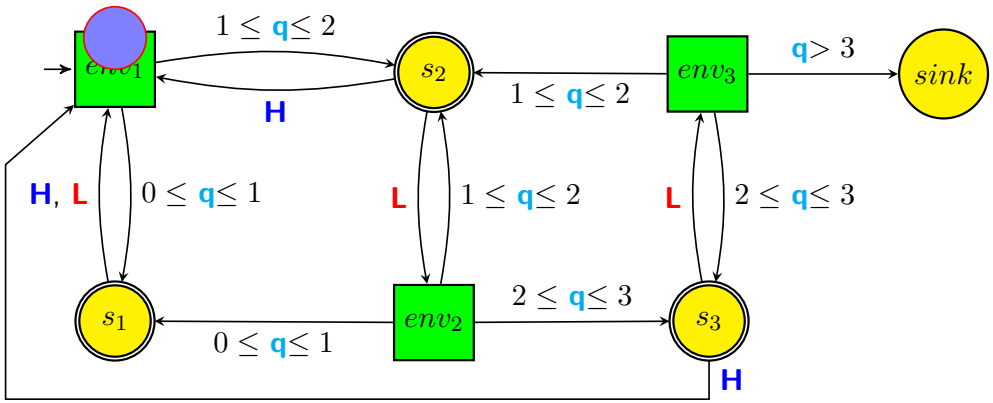


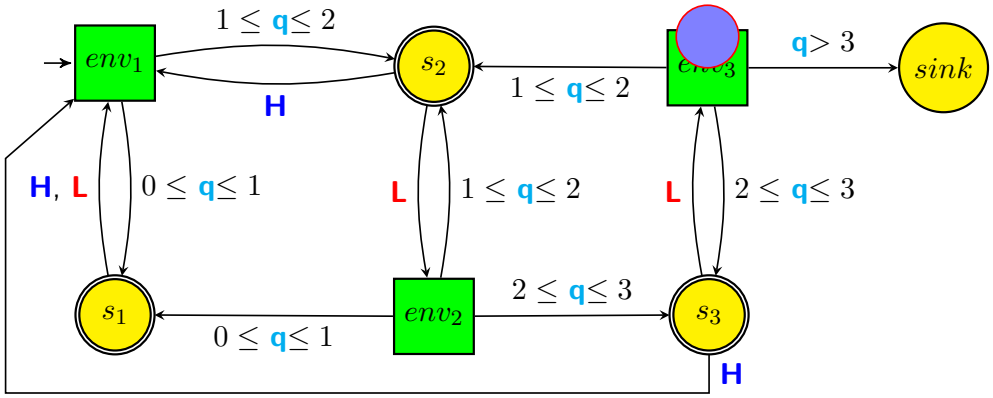
We cannot allow both components to be on env_2

If both components are at s_2, at least one **H** must be scheduled

Only one **H** per slot allowed

Example
Simultaneous Games (composition)





We cannot allow both components to be on env_2

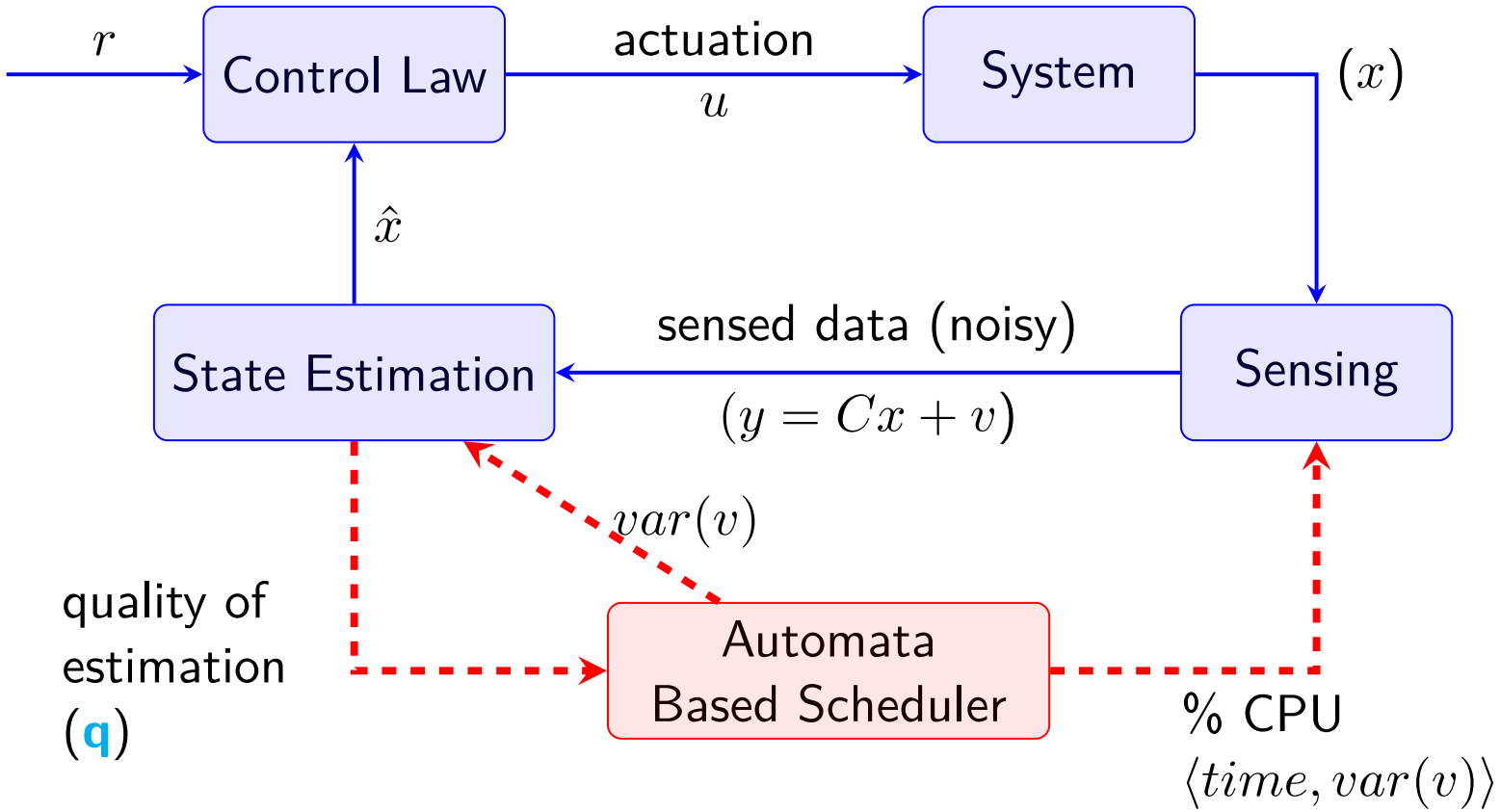
If both components are at s_2, at least one **H** must be scheduled

Adding a third component makes the system unschedulable!

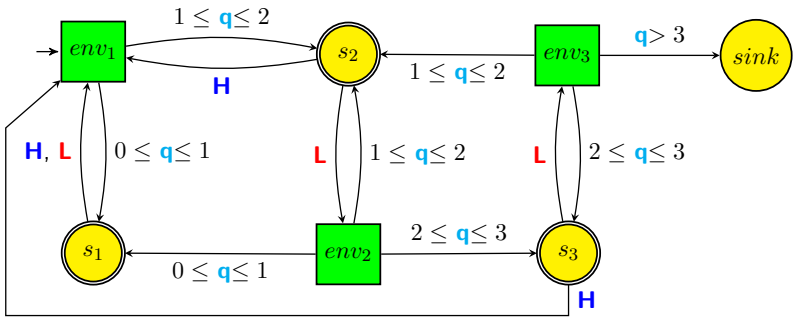
Only one **H** per slot allowed

The Proposed Architecture

The scheduler is part of the control loop



A Component in the System




```
High () {
    ...
}
Low () {
    ...
}
```

Scheduling Büchi Game

- **Alternating turns**
- Scheduler alphabet is $\Sigma_{schd} = 2^T$ (in the example $T = \{H, L\}$)
- Environment alphabet is $\Sigma_{env} = \mathbb{R}^n$ (*scheduler feedback variables*)
- There is an transition for any **possible** environmental outcome
- The **scheduler feedback variables** can be any environment-depended value
- Environment player plays first

The Proposed Architecture

Automata-Based Specification Interface

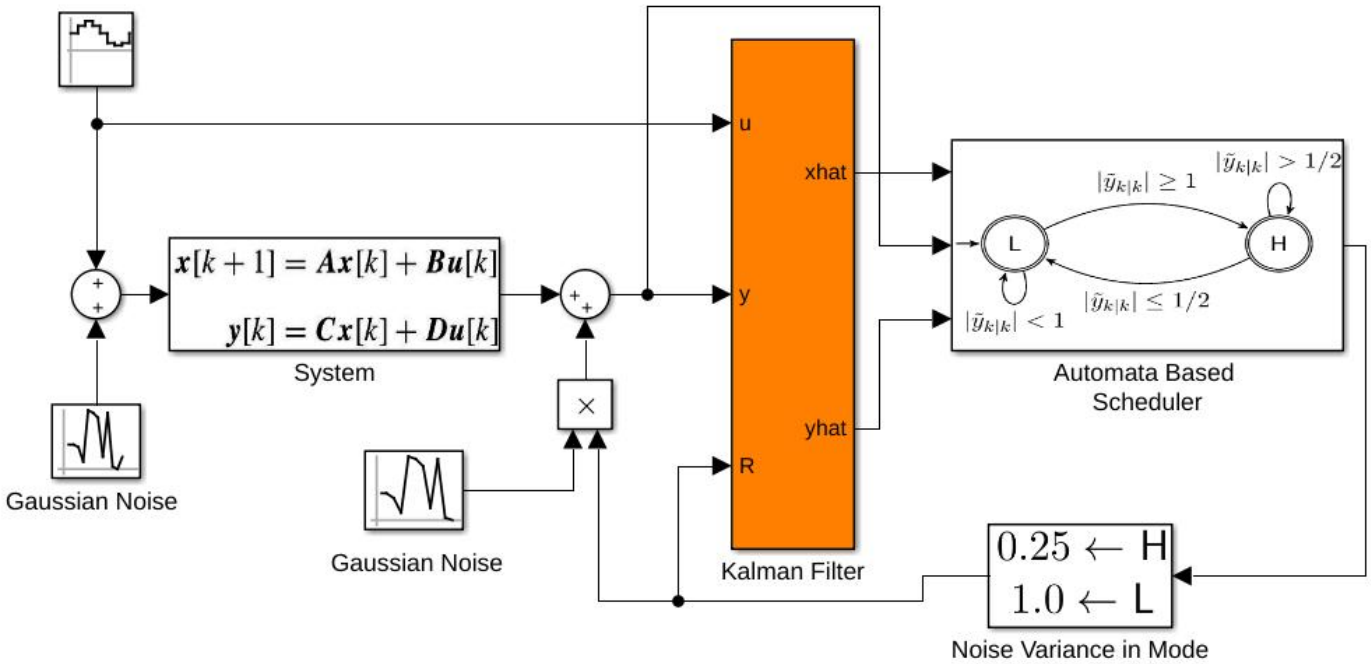
Automata advantages

- **Lite:** minimal resource consumption at run-time
- **Expressive**
- **Composable**
- **Automata theory:** allows for tools such as *GOAL*

Graphical tool for Omega-Automata and Logics

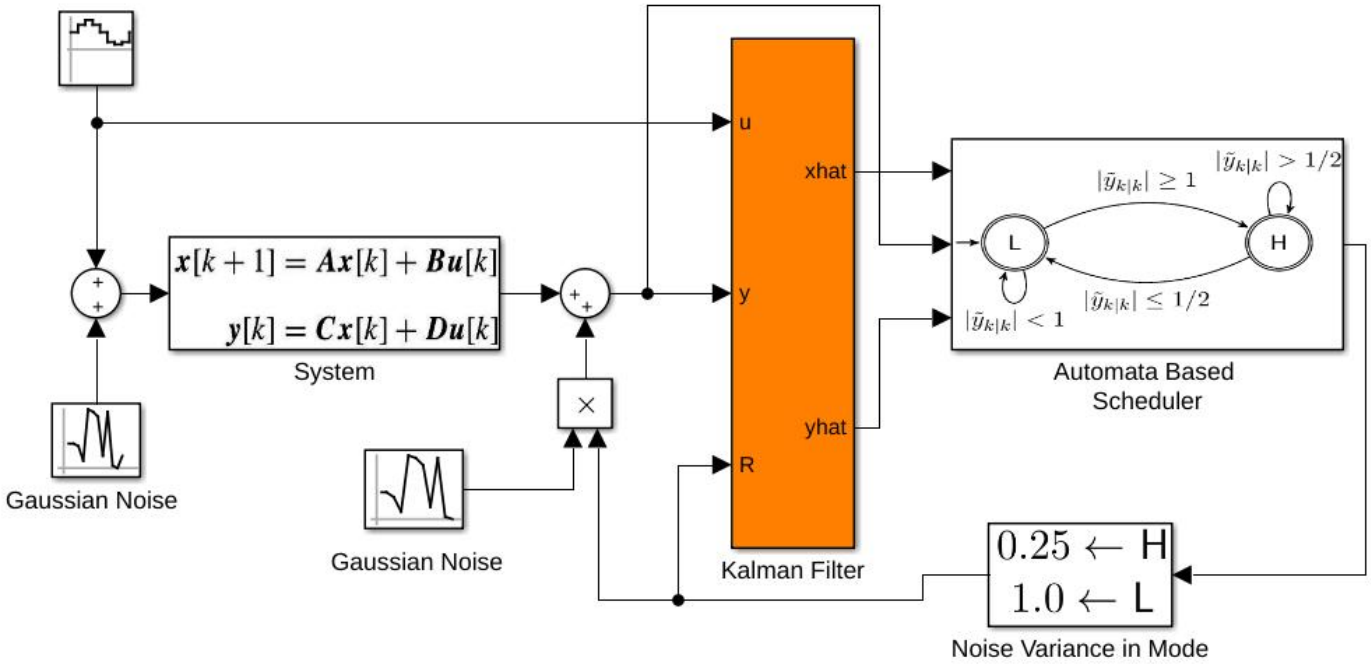
Simulation

Schedule sensing-tasks based on the estimation quality



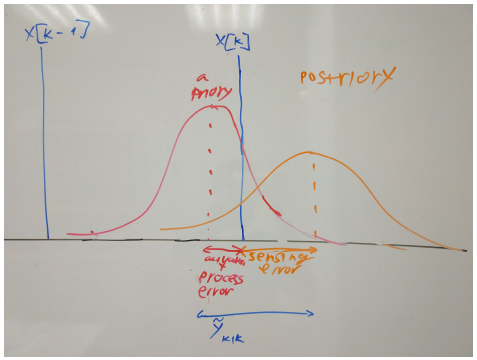
Simulation

Schedule sensing-tasks based on the estimation quality



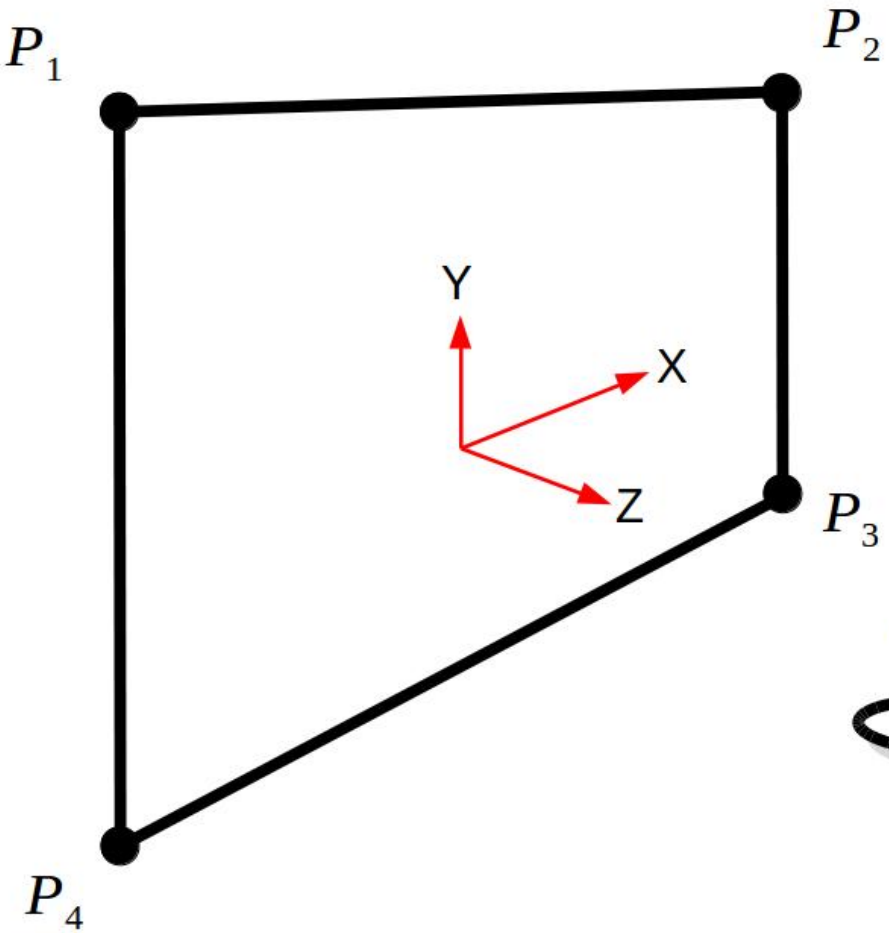
	High	Low	Aut. Based
%CPU	85	10	46
mean of $\lvert x - \hat{x} \rvert$	0.97	1.24	1.08

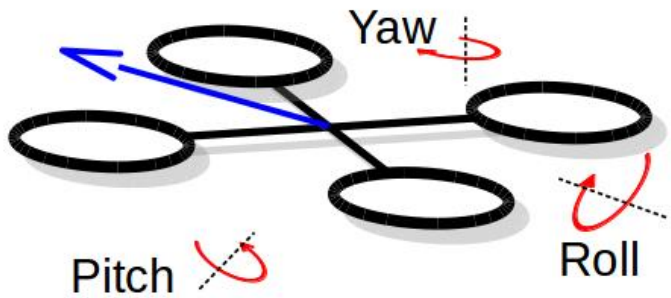
Integration with Kalman filter



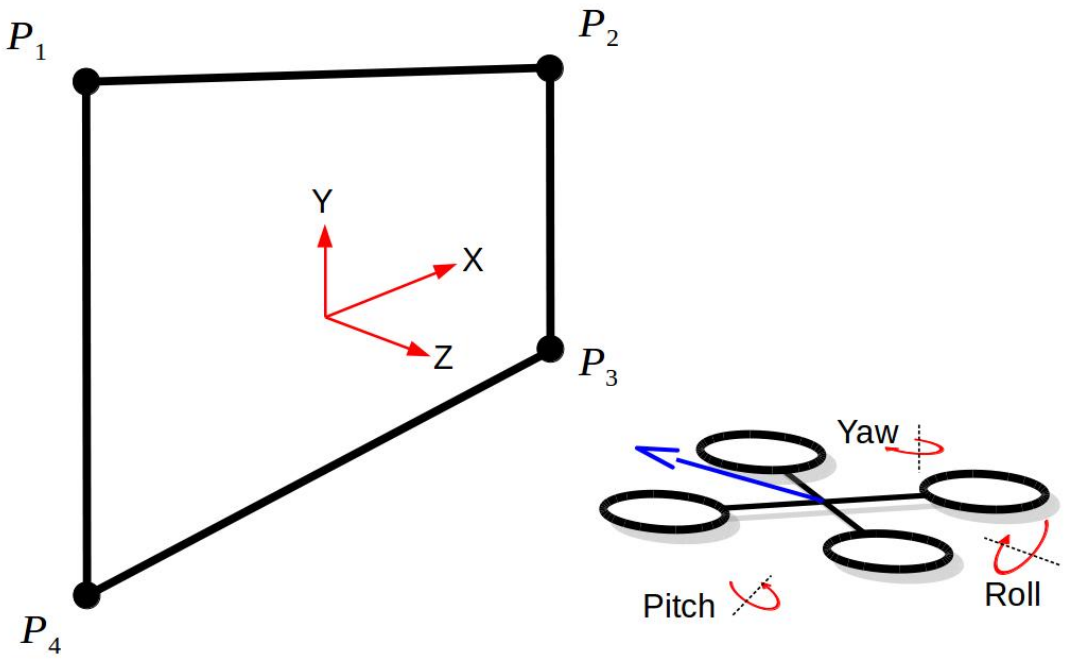
- Process & actuation noise affect the *a-priory* estimation
- Measurement error affect only the measurement
- The difference between them estimate the over-all errors

Mission Definition





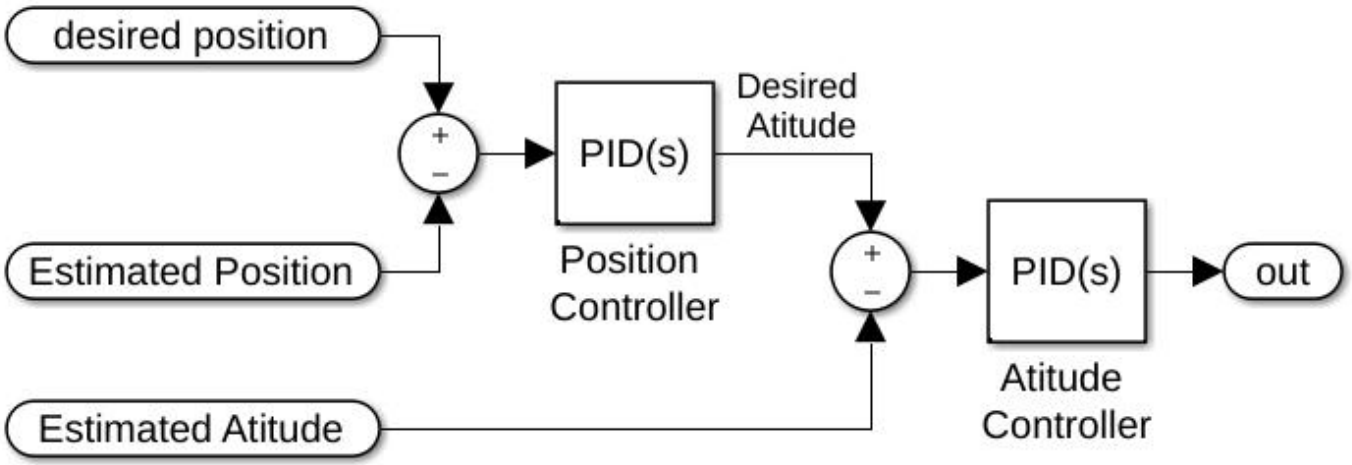
Control Objectives



Control & Scheduling Objectives

- Minimize deviation in the x axis
- Minimize CPU usage of the image processing task

Controller Design



Attitude and position controller

- The **vision** component estimate the position
- Position controller output a desired angle (proportional to acceleration)
- We used a standard PID controllers

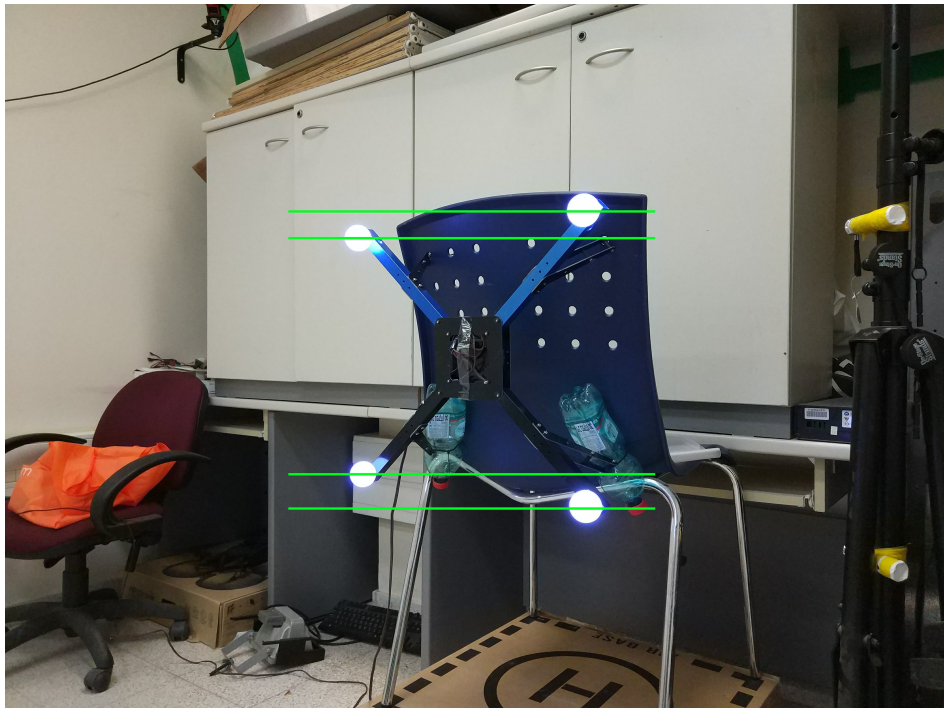
Vision Component

Image Processing Algorithm

1. Find the window corners (brute force search)
2. Calculate the drone position

Calculate the Drone Position

Vertical Difference



$$V_d = \frac{((y_1 - y_4) - (y_2 - y_3))}{((y_1 - y_4) + (y_2 - y_3))}$$

$$sz = ((y_1 - y_4) - (y_2 - y_3)) + ((y_1 - y_4) + (y_2 - y_3))$$

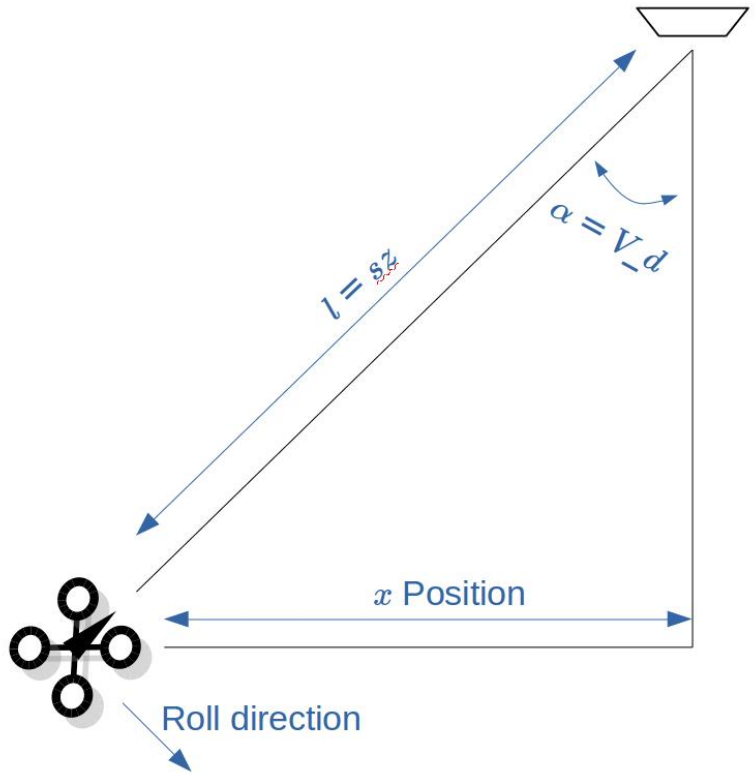
Calculate the Drone Position

Center of Mass

$$S_x = \frac{x_1+x_2+x_3+x_4}{4}$$

Calculate the Drone Position

Aproximate x Position



$$x = l \cdot \sin(V_d) \approx l \cdot V_d$$

The Filter

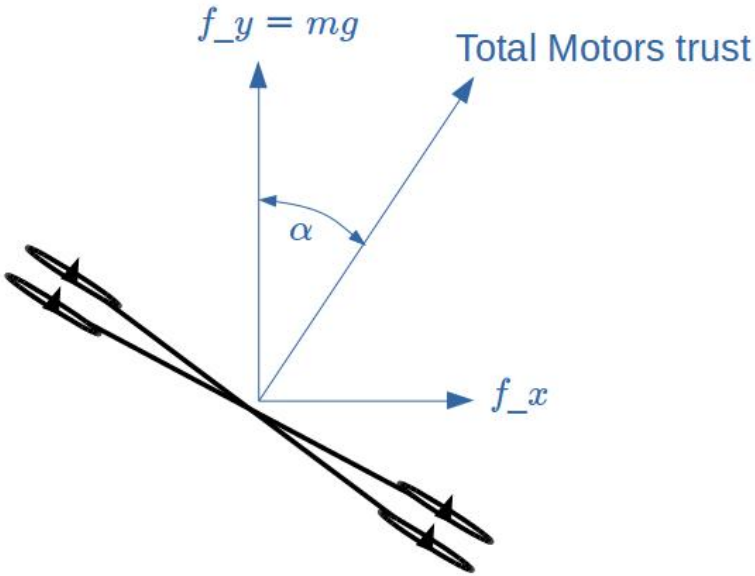
Kalman filter constrains

- The system is not linear
- The process and actuation noises distribution are unknown and unstable
- Kalman filter adds complexity in the code

Filtering in two steps

1. Predicts - with a linearized model
2. Update - with the vision and other sensors

The Linearized Model



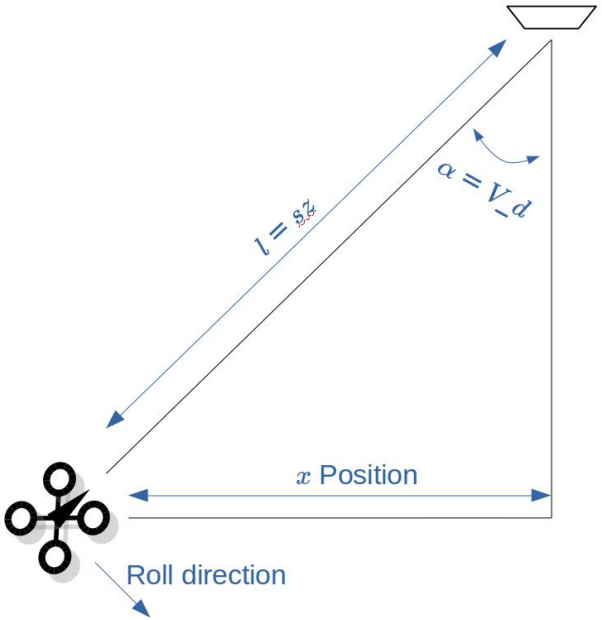
$$A = \begin{pmatrix} 1 & dt & 0 \\ 0 & 1 & dt \\ 0 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

Basic equations of motion on x axis

- **Position**: $\bar{r}_x[k+1] = r_x[k] + dt \cdot v_x[k]$
- **Velocity**: $\bar{v}_x[k+1] = v_x[k] + dt \cdot a_x[k]$
- **Acceleration**: $\bar{a}_x[k+1] = \Sigma F_x/m \approx roll \cdot g$

The Measurement vector



$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/g \end{pmatrix}$$

Measurement vector

- **Position**: from vision algorithm
- **Velocity**: $\frac{\partial r_x}{\partial t}$
- **Acceleration**: roll angle from the AHRS of APM

The Update Step

$$x[k] = K \cdot \bar{x}[k] + (1 - K) \cdot C^{-1} \cdot y[k]$$

$$x_r = K_r \cdot \bar{x_r}[k] + (1 - K_r) \cdot y_r[k]$$
$$x_v = K_v \cdot \bar{x_v}[k] + (1 - K_v) \cdot y_v[k]$$
$$x_a = \bar{x_a}$$

Overall noise estimation

$$\tilde{y}_{k|k} = \bar{x_r}[k] - y_r[k]$$

Experiment Setup

this slide is needed?

Vision Mode

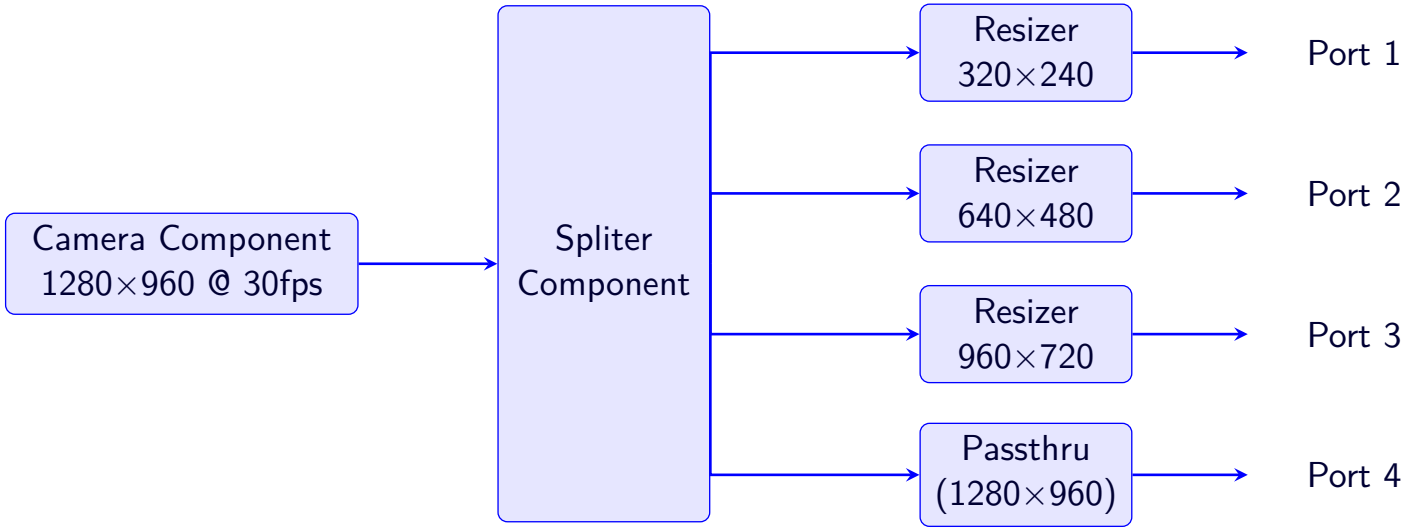


Image resolution switching

- Change camera resolution in run time adds large **delay**
- Use **hardware** (GPU resizer) for fast mode switch

Constant Vision Mode

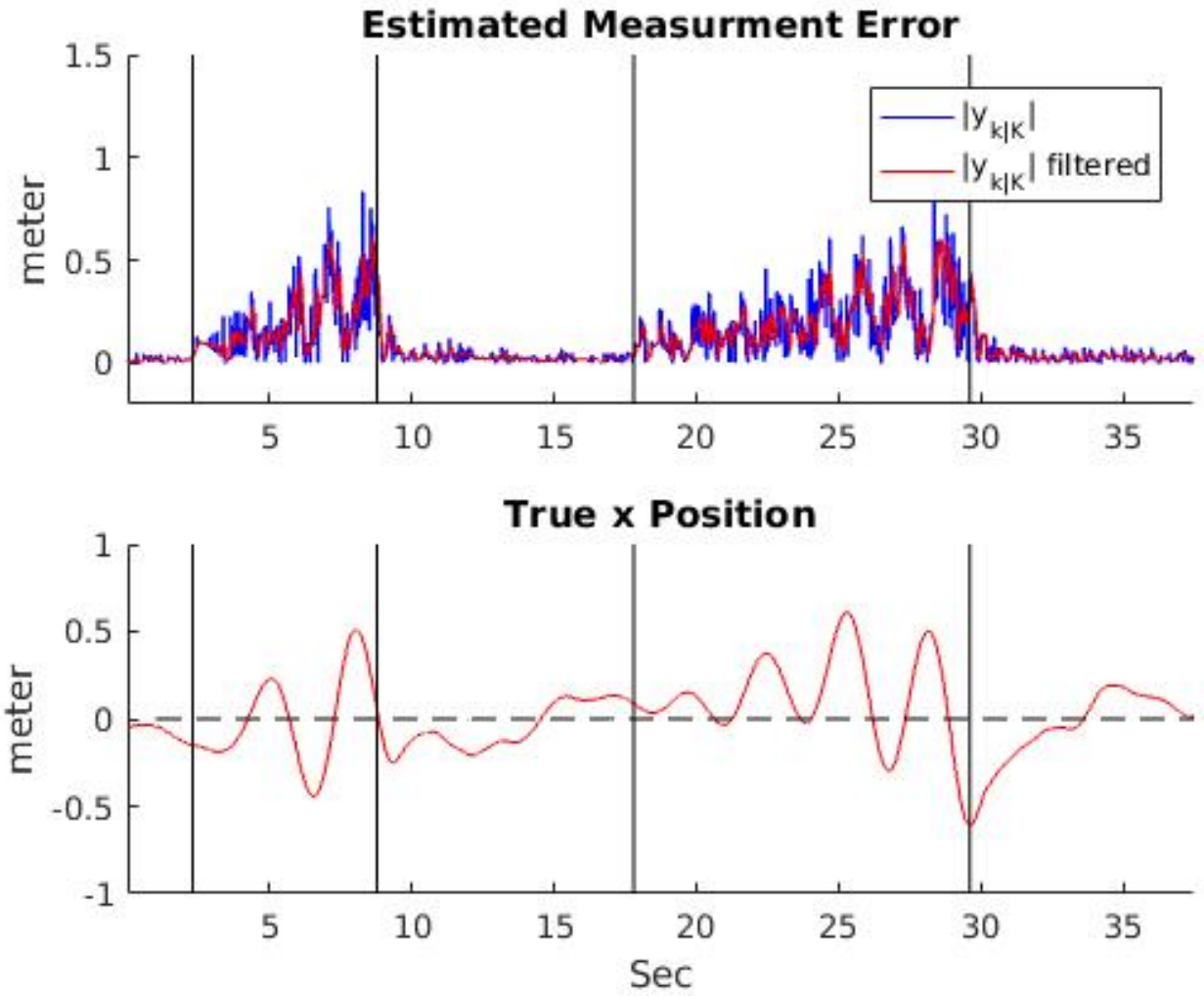
Always low quality mode

- 240p resolution
- mean error of 30cm
- 2.1% CPU usage

Always high quality mode

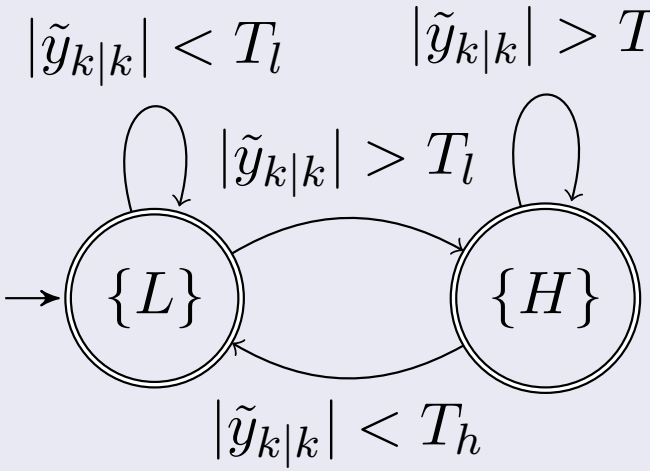
- 960p resolution
- mean error of 9.5cm
- 30% CPU usage

Manual Mode flight

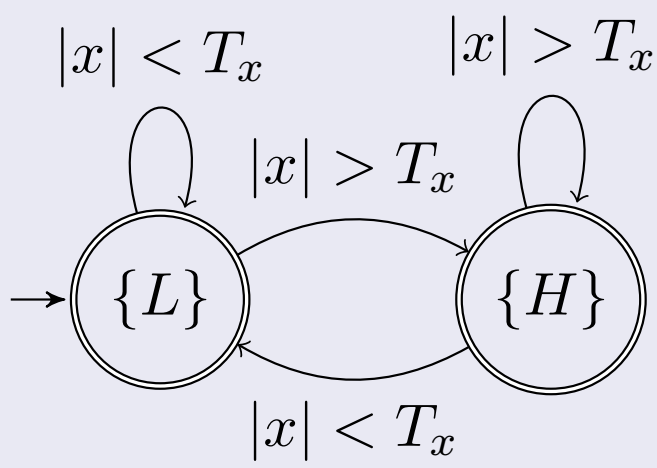


Reactive Schedulers

A_{err} - estimation error based

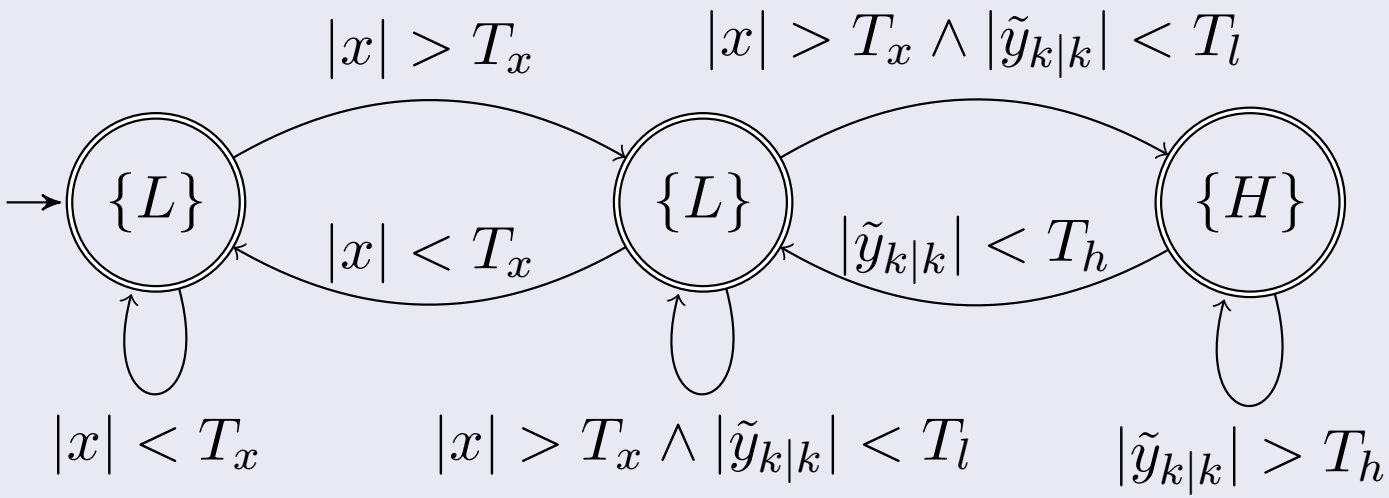


A_x - position based



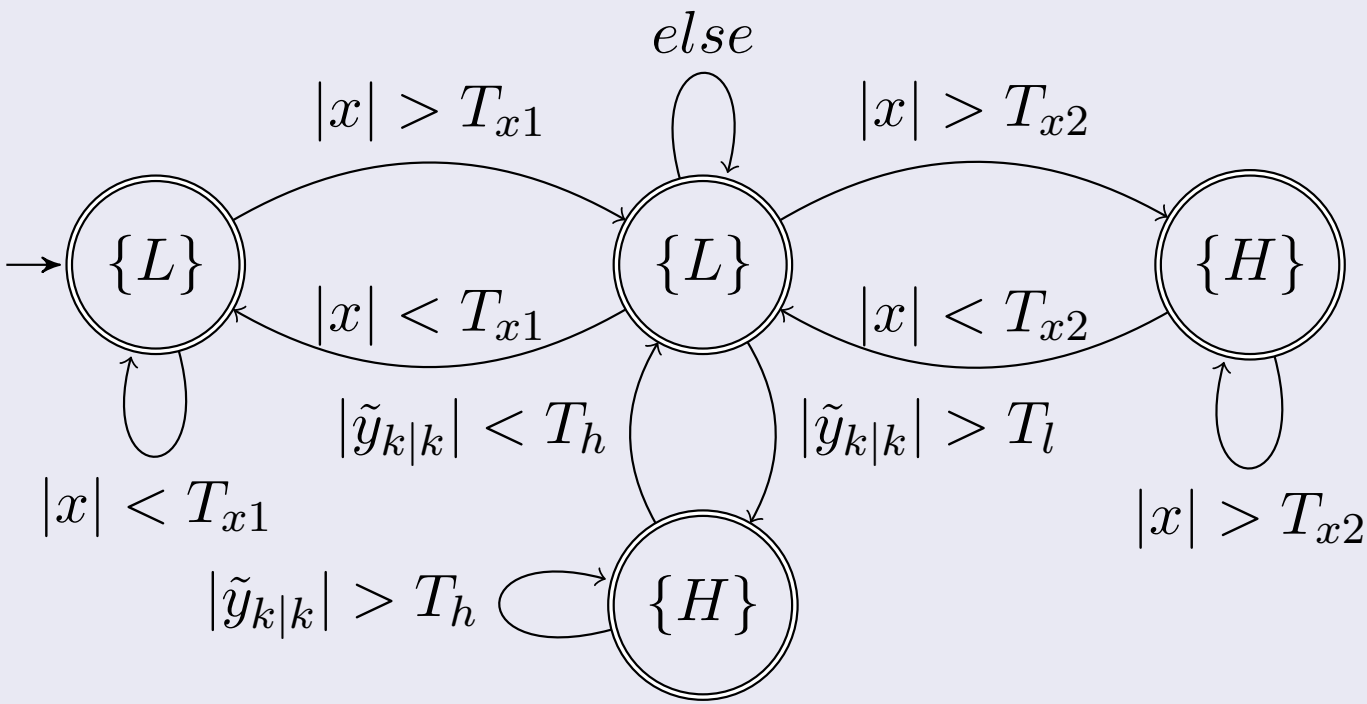
Reactive Schedulers

A_{Comp1}



Reactive Schedulers

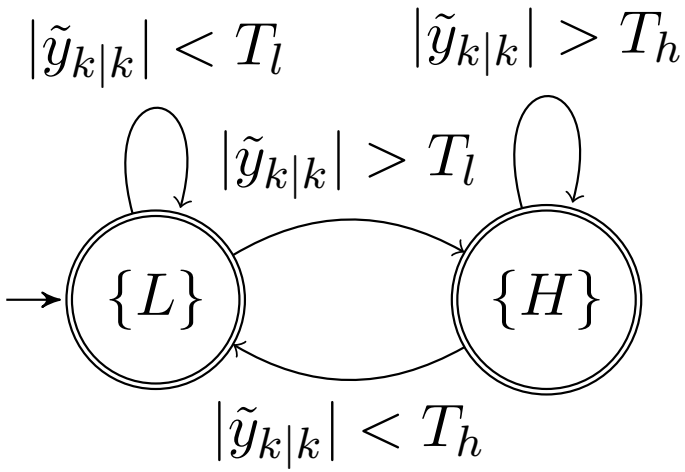
A_{Comp2}



Results

Schedule	**% CPU**	**mean($\lvert x \rvert$) (cm)**
Only High	30.9%	9.5
Only Low	2.1%	30.0
A_x $(T_x = 10)$	**16.6%**	**10.9**
A_x $(T_x = 20)$	14.0%	14.1
A_x $(T_x = 30)$	8.9%	17.4
A_{err} $(T_l = 10, T_h = 20)$	10.3%	14.9
A_{err} $(T_l = 10, T_h = 15)$	**11.7%**	**11.3**
A_{comp1} $(T_x = 10\ ,\ T_l = 10\ ,\ T_h = 15)$	**8.8%**	**12.9**
A_{comp2} $(T_{x1} = 10\ ,\ T_{x2} = 30\ ,\ T_l = 10\ ,\ T_h = 15)$	10.4%	12.7

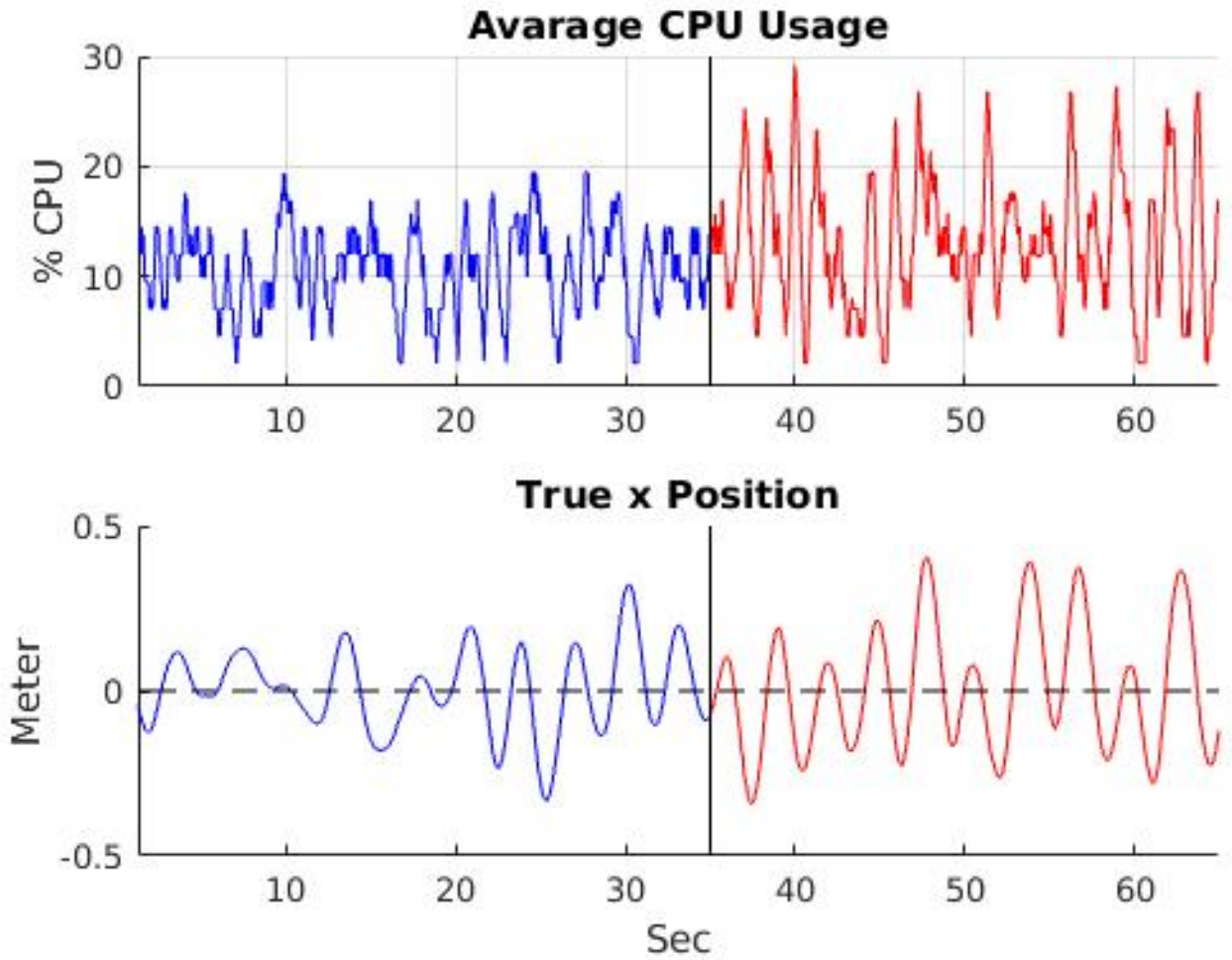
Adaptiveness Results



$$[T_l = 10, T_h = 15]$$

conditions	**% CPU**	**mean($\lvert x \rvert$) (cm)**
Fan off	11.7%	11.3
Fan on	13.2%	11.8

Adaptive Results



instead of with Related Work

review of similar papers: A table with few papers