Reactive Scheduling of Computational Resources in Control Systems

Hodai Goldman

Ben-Gurion University of the Negev
Department of Computer Science

January 1, 2018

Overview - *TODO: clean this slide*

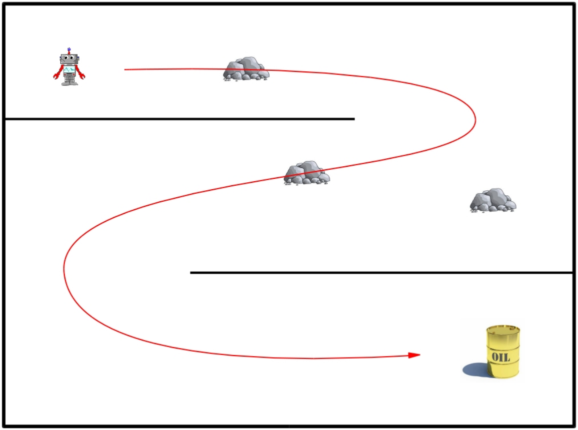
Contributions

- Development of control and scheduling co-design framework
- **Reactive** scheduling (environment condition adaptation)
- **Independent**, **adaptive**, and **composable** interface (Based on automata theory)
- *What we do better?*
- Prepare the ground for automata-based **scheduling tool**
- Development of scheduling technique based on **Kalman filter**

Achievements of this thesis

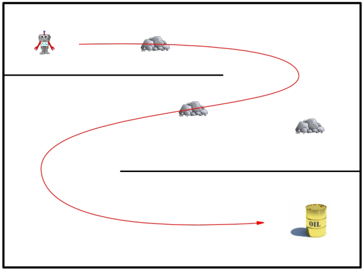
- Continue the work of **RTComposer**
- Proof of concept with **simulation**
- Proof of concept with **real-life case-study**
- **Bridge** the gaps between control and software engineering

An control problem example



Robot navigation

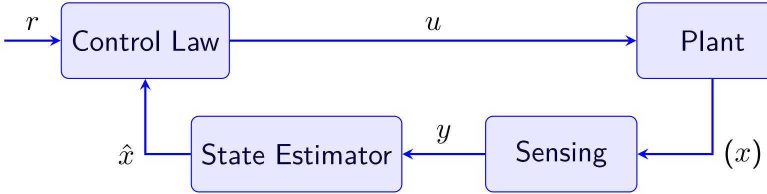
An control problem example



The Objectives

- The robot need to reach the target point **fast** and **safely**
- The robot have on-board camera for **obstacle-avoidance**
- The robot use GPS for general **navigating**

The Traditional Solution



+

Constant time steps + periodic tasks

time steps

figure+

Task	Period	Deadline
Check for obstacles	10ms	1.5ms
Check GPS position	10ms	0.5ms
Control Law	2ms	0ms
...		

The Main Software Design Problems

Task	Period	Deadline
Check for obstacles	10ms	1.5ms
Check GPS position	10ms	0.5ms
Control Law	2ms	0ms
...		

The design problems from our point of view

- **All the tasks are highly coupled**: *any change or addition of some task require to consider all other tasks requirements*
- **Static and inefficient scheduling**: *the table is defined for the worst case* *talk about related work on this direction*
- **No consideration of the environmental conditions**: it is a cyber-physical system after all

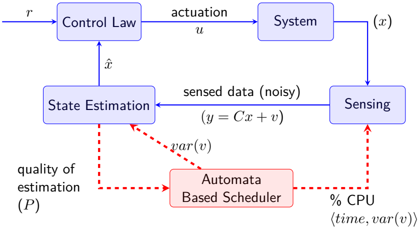
The Goal

In this thesis we design an **reactive** scheduling framework for real-time systems

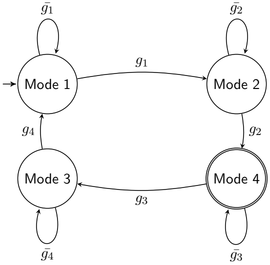
Required features:

- **Independent** and **composable** requirements
- **Control objective based** requirement interface
- Environment **adoptive** scheduler

The Proposed Architecture



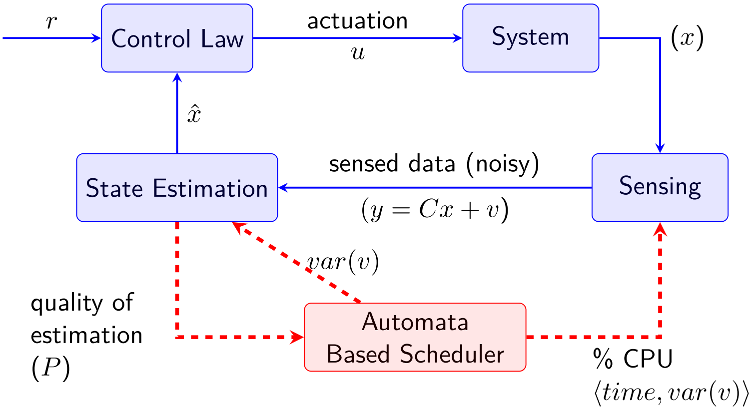
$+$



System Design

The Proposed Architecture

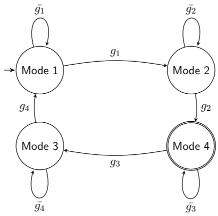
Explain that the scheduler is involve in the control loops



Automata-Based Specification Interface

The Proposed Architecture

maybe add a word about RTcomposer and GameComposer

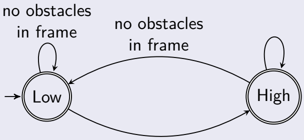


Why Automata

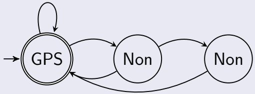
- **Lite:** minimal resource consumption at run-time
- **Composable**: easy to compose independent components
- **Automata theory built in:** allows for tools such *GOAL*
- **Expressiveness**

Example of Guarded Automata
The Proposed Architecture

Obstacle avoidance component

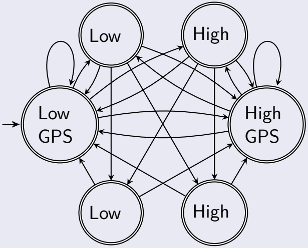


GPS navigation component



$\Rightarrow$

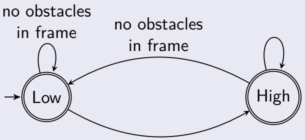
Composed guarded automata



Simplifying the Guarded Automata

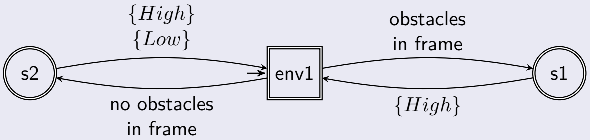
The Proposed Architecture

Mode-based guarded automata (for good intuition)



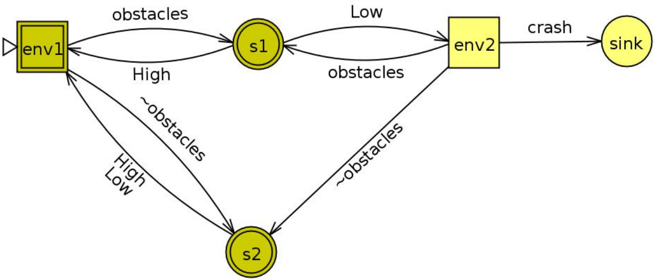
$\Downarrow$

The automata in practice (best match ω-word theory)

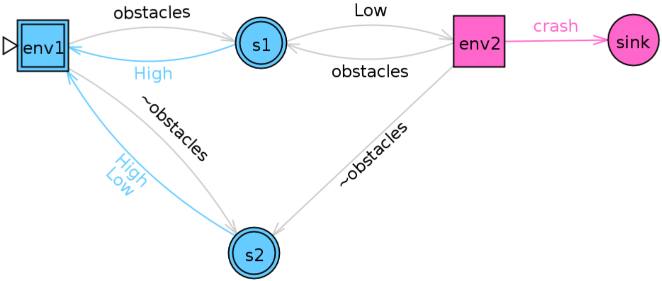


Q: How to create the guarded automata? By wining Büchi games

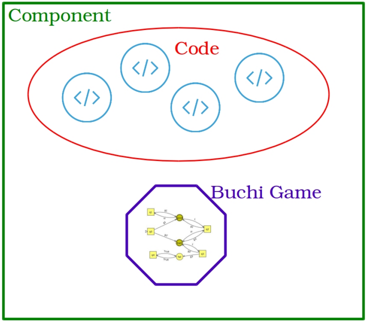
Büchi game remainder



Büchi game remainder



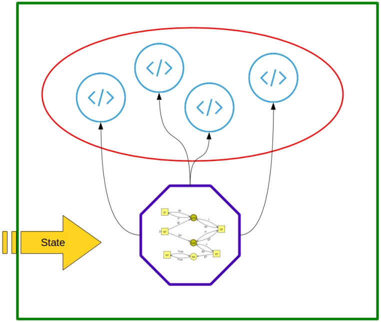
A Component in the System



Component Definition ($\langle T, G \rangle$)

- A set of subroutines (functions code)
- A Generalize Büchi Game

A Component in the System

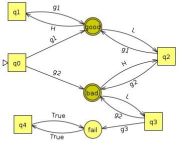


The Büchi game ($G = \langle A, \langle P_{sched}, P_{env} \rangle \rangle$)

- Is played in turns by the **environment** and the **scheduler**
- Represent the **interaction** between the scheduler and the environment reaction

Scheduling Büchi Game

A Component in the System



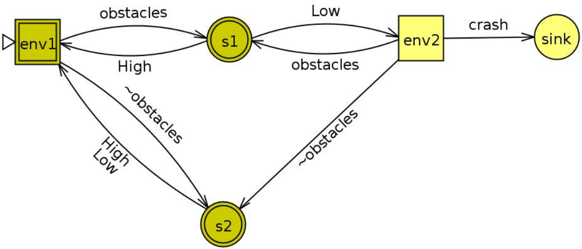
Scheduling Büchi Game

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

Example - Büchi Game

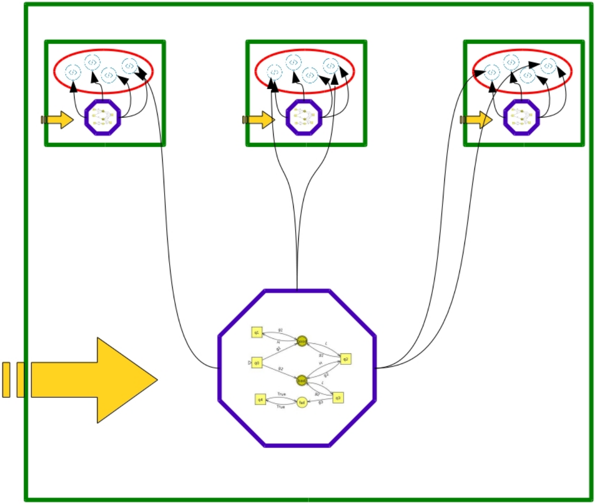
A Component in the System

The Büchi Game of the obstacles avoidance component:

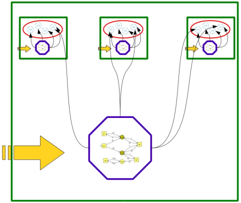


- The objectives of the component is to avoid obstacles
- The scheduler **win** $\Leftrightarrow$ the corresponding word $\omega \in \mathcal{L}(A) \Leftrightarrow$ the component achieved his **objectives**

Component Composition



Component Composition



Requirements

- A game ($G = \langle A, \langle P_s, P_e \rangle \rangle$) correspond to all the components
- The game of Component is $G_i = \langle A_i, \langle P_s^i, P_e^i \rangle \rangle$
- $\omega \in \mathcal{L}(A) \Leftrightarrow \forall i : \omega(i) \in \mathcal{L}(A_i)$

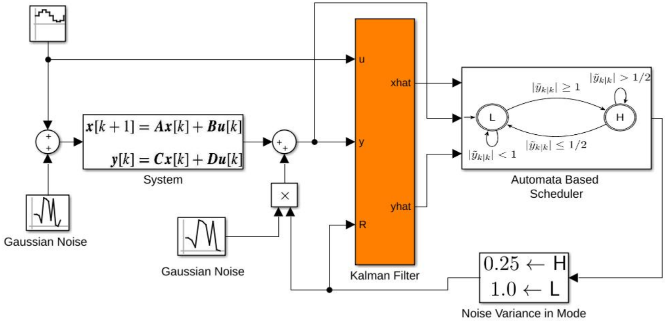
TODO: how to present the composition details?

TODO: show the resource component

TODO: show the scheduler work: 1. find wining strategy 2. simultaneously walk through the strategy automata

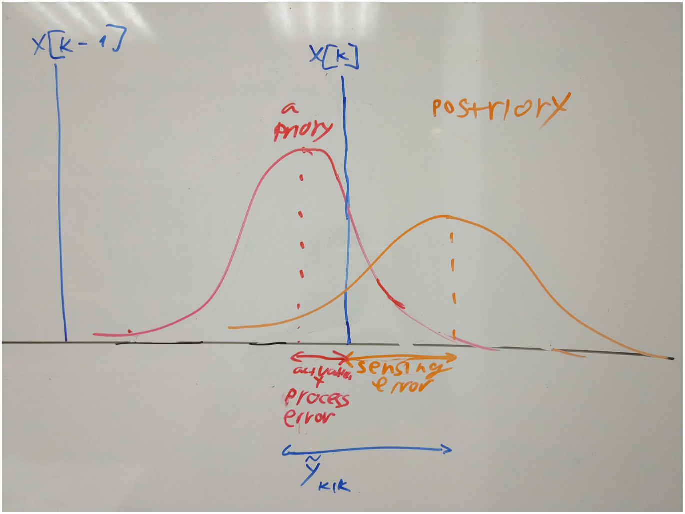
Simulation

Schedule sensing-tasks based on the estimation quality



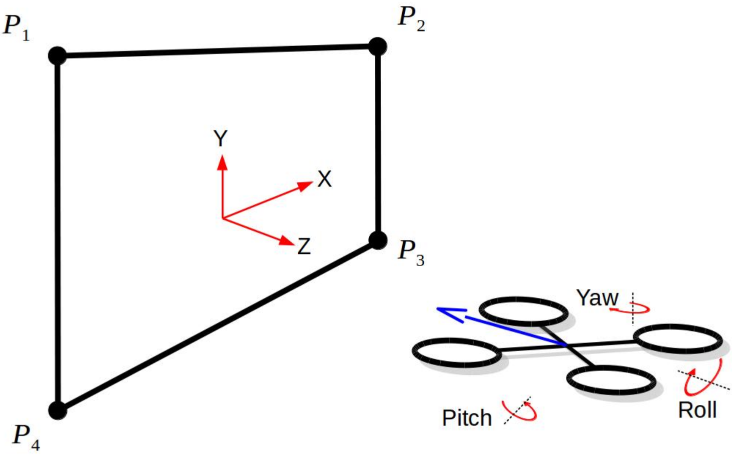
	High	Low	Aut. Based
%CPU	85	10	46
mean of $\lvert x - \hat{x} \rvert$	0.97	1.24	1.08

Integration with Kalman

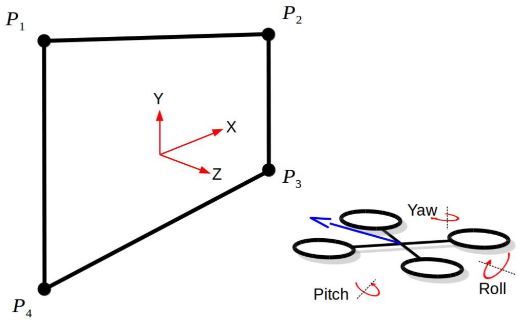


Mission Definition

Explain the window motivation



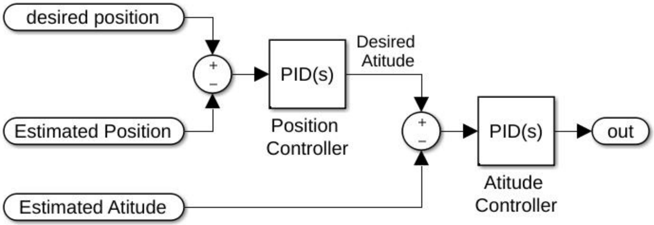
Concrete Control Objectives



Control & Scheduling Objectives

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

Traditional Controller Design



Attitude and position controller

- **vision** component estimate the x-position
- Position controller output a desired roll angle
- Attitude controller is a traditional attitude controller

Vision Component

front picture

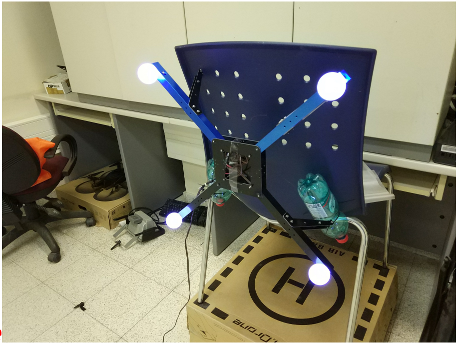


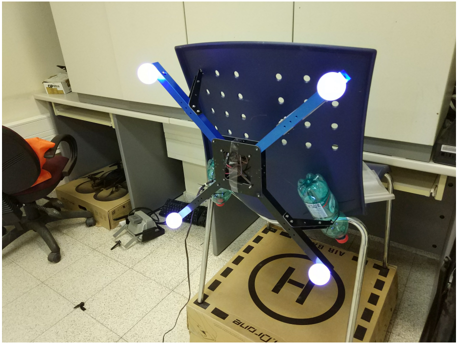
Image Processing Algorithm

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

Calculate the Drone Position

Vertical Difference

side picture

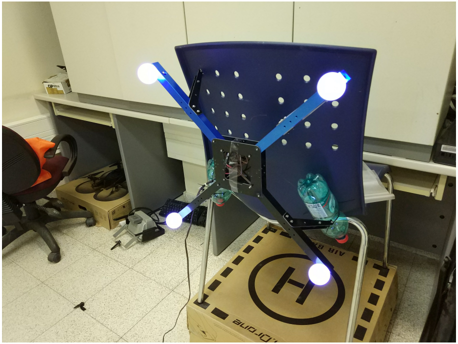


$$V_d = \frac{((y_1 - y_4) - (y_2 - y_3))}{((y_1 - y_4) + (y_2 - y_3))}$$

Calculate the Drone Position

Center of Mass

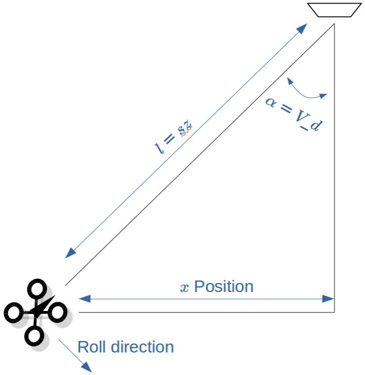
shifted picture



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

Calculate the Drone Position

Aproximate x Position



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

Two Step Filter

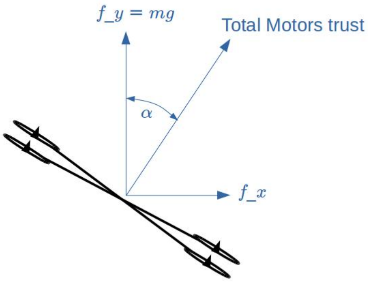
Why not Kalman filter

- It's a Non-linear system
- The process noise distribution is unknown, and unstable
- Kalman filter adds complexity in the code

Two step filter

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

The Linearized Model



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

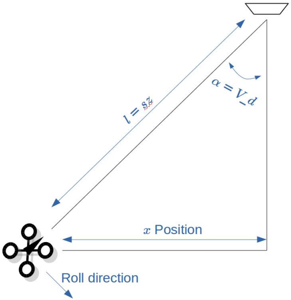
$$B = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

Basic equations of motion on x axis

Assume stable hover:

- **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

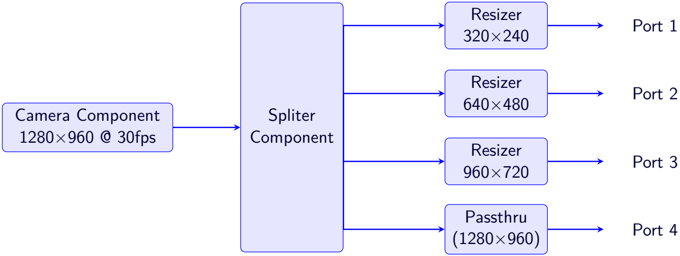


Image resolution switching

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

Constant Vision Mode

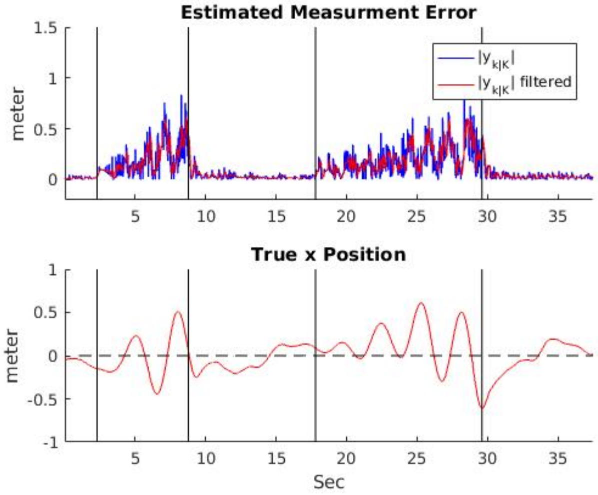
Always low quality mode

- 240p resolution
- mean error tolerance of 30cm (*not really stable*)
- 2.1% CPU usage

Always high quality mode

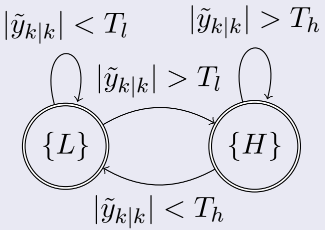
- 960p resolution
- mean error tolerance of 9.5cm
- 30% CPU usage

Manual Mode flight

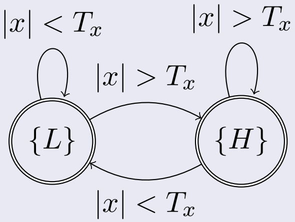


Reactive Schedulers

A_{err}

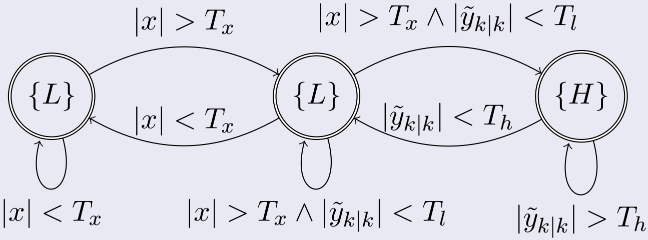


A_x



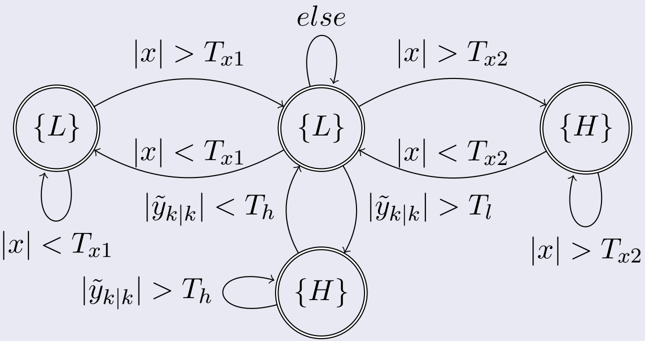
Reactive Schedulers

A_{Comp1}



Reactive Schedulers

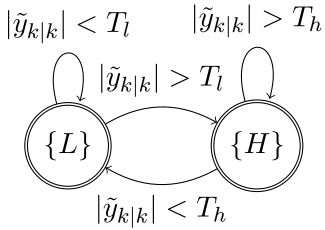
A_{Comp2}



Results

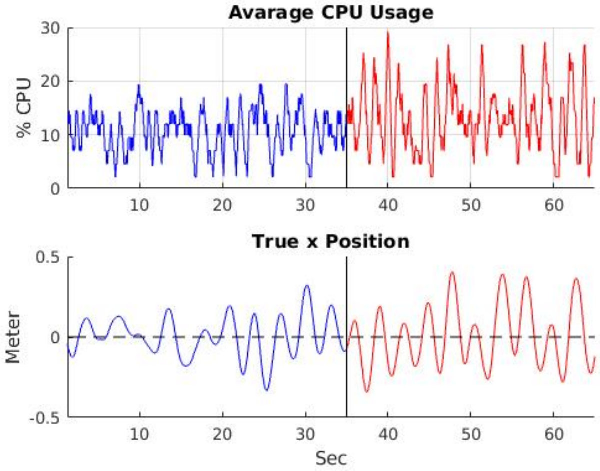
Schedule	**% CPU**	**mean($\|x\|$) (cm)**
Only High	30.9%	9.5
Only Low	2.1%	30.0
A_x $(T_x = 10)$	16.6%	10.9
A_x $(T_x = 20)$	14.0%	14.1
A_x $(T_x = 30)$	8.9%	17.4
A_{err} $(T_l = 10, T_h = 20)$	10.3%	14.9
A_{err} $(T_l = 10, T_h = 15)$	11.7%	11.3
A_{comp1} $(T_x = 10 \ , \ T_l = 10 \ , \ T_h = 15)$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

Adaptive Results



conditions	**% CPU**	**mean($|x|$) (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

Thanks