Model-Based Development and Validation of Multirobot Cooperative System

Jüri Vain
Dept. of Computer Science
Tallinn University of Technology

Syllabus

- **Monday morning: (9:00 – 13.30)**
  - 9:00 – 10:30 Intro: Model-Based Development and Validation of Multirobot Cooperative System (MCS)
  - 10:30 – 12:00 MCS model construction and learning
  - 12:00 – 13:30 Model-based testing with reactive planning testers
- **Tuesday morning: (9:00 – 12.30)**
  - 9:00 – 10:30 Model checking Multirobot Cooperative Systems
  - 10:30 – 12:00 Hands-on: Distributed intruder capture protocol

Lecture #L2 : Model construction & learning Motivation

- Model construction is one of the bottlenecks in MBD
  - is time consuming
  - needs understanding of the system modelled
  - needs understanding why it is modelled
  - needs understanding how has to be modelled
- Choose the right modelling formalism that:
  - is intuitive
  - has right modelling power
  - has efficient decision procedures
  - has good tool support

→ UPTA

Model construction techniques

- Model extraction from program code
- Text analysis (used in some UML tools)
- Pattern-based model construction
- Model learning

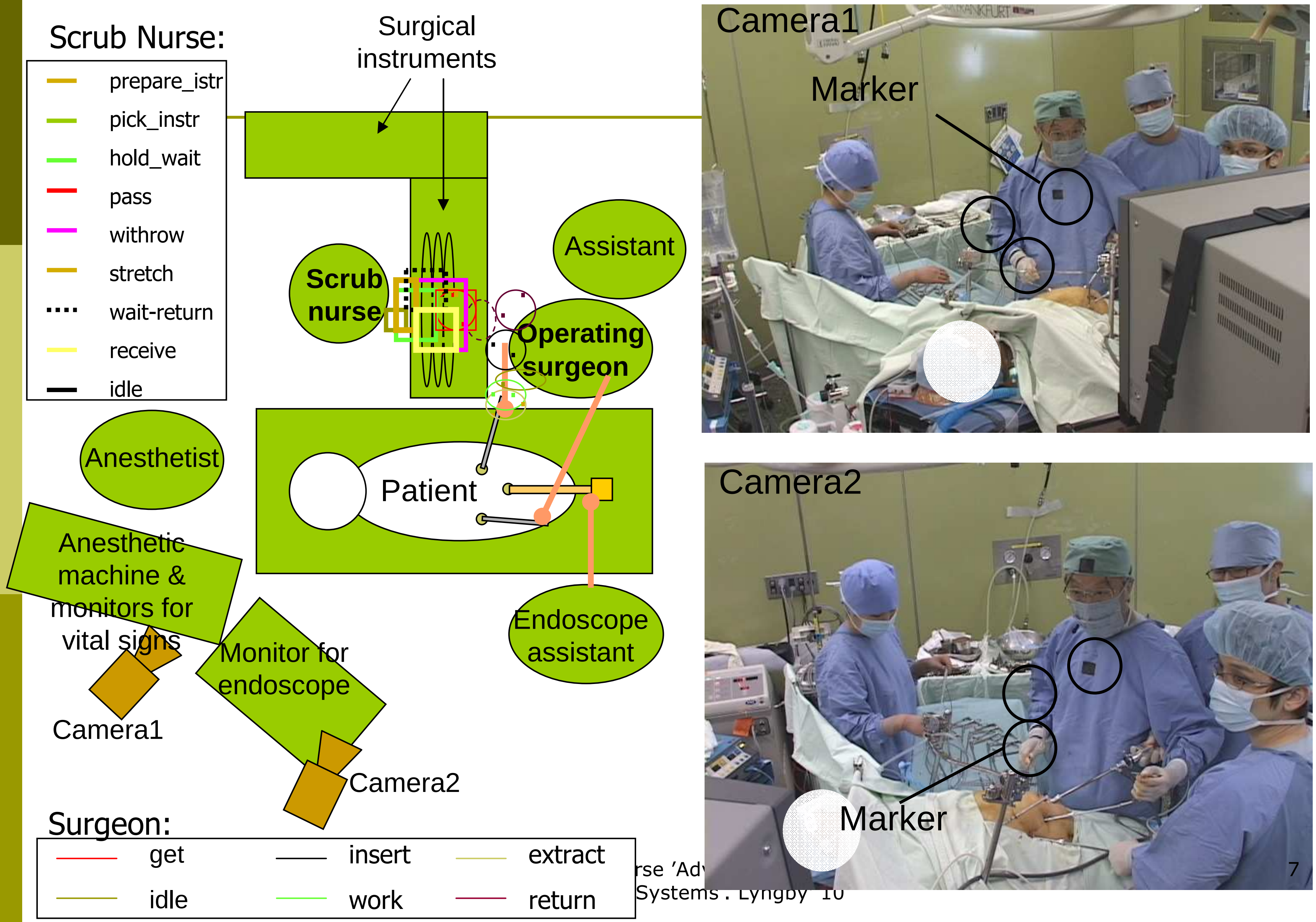
Terminology of machine learning

- Passive *vs* active
- Supervised *vs* unsupervised
- Reinforcement learning (reward guided)
- Computational structures used:
  - FSA
  - Hidden Markov Model
  - Kohonen Map
  - NN
  - Timed Automata
  - etc

Learning XTA (lecture plan)

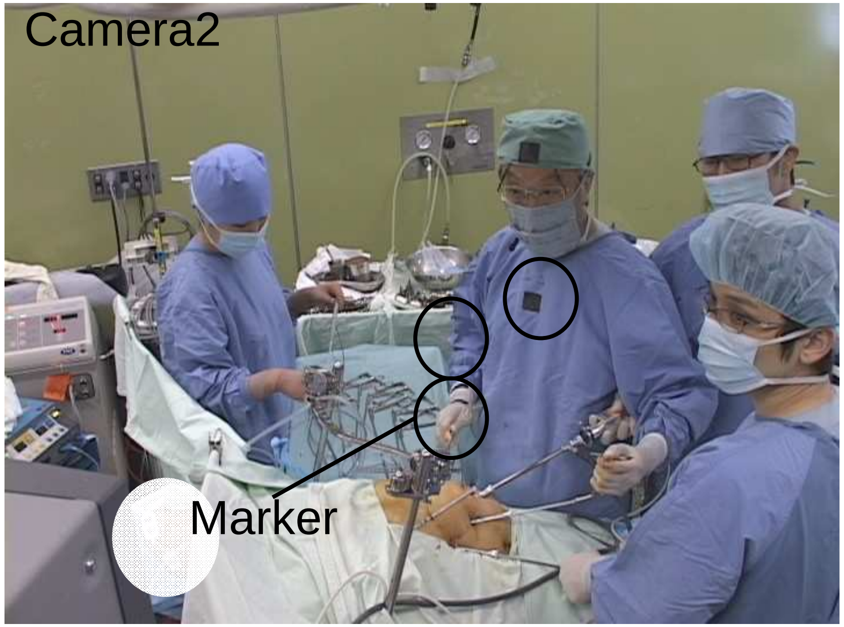
- Problem context
- Assumptions:
  - I/O observability
  - Fully observable (output determinism)
  - Generally non-deterministic models
- The learning algorithm
- Evaluating the quality of learning

Problem context: SNR scene and motion analysis



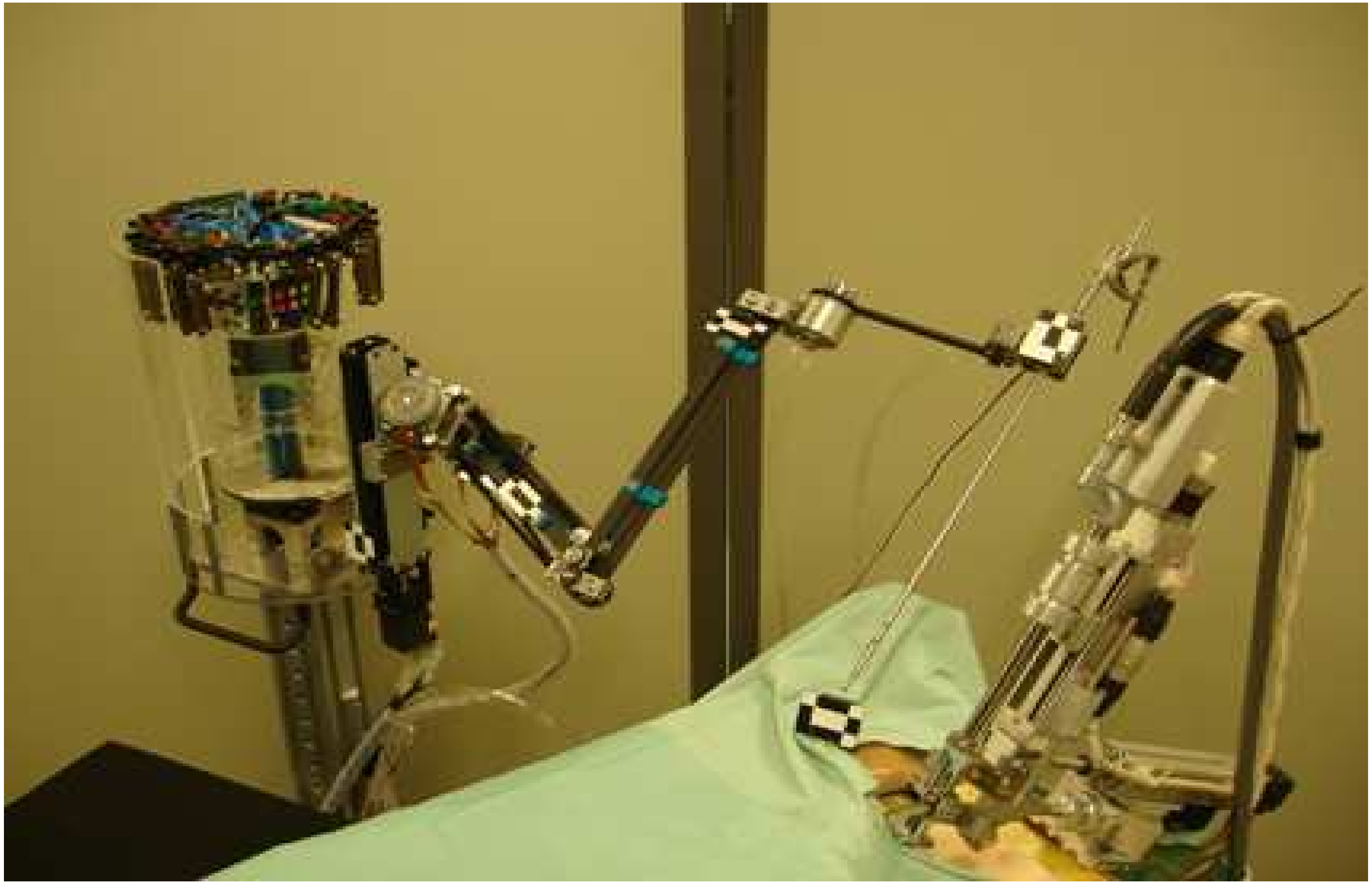
Scrub Nurse Robot (SNR): Motion analysis

Demo

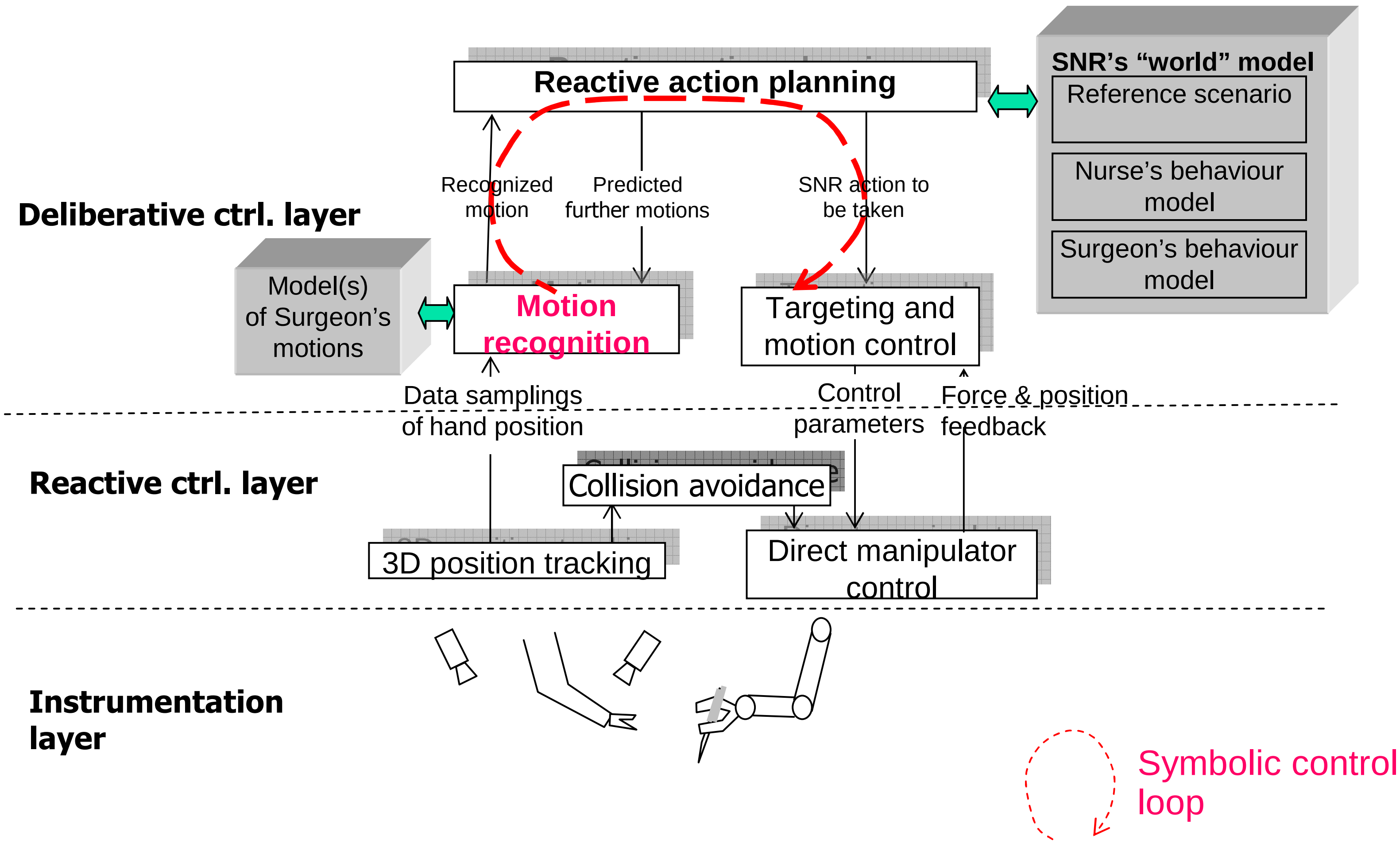


Photos from CEO on HAM, Tokyo Denki University

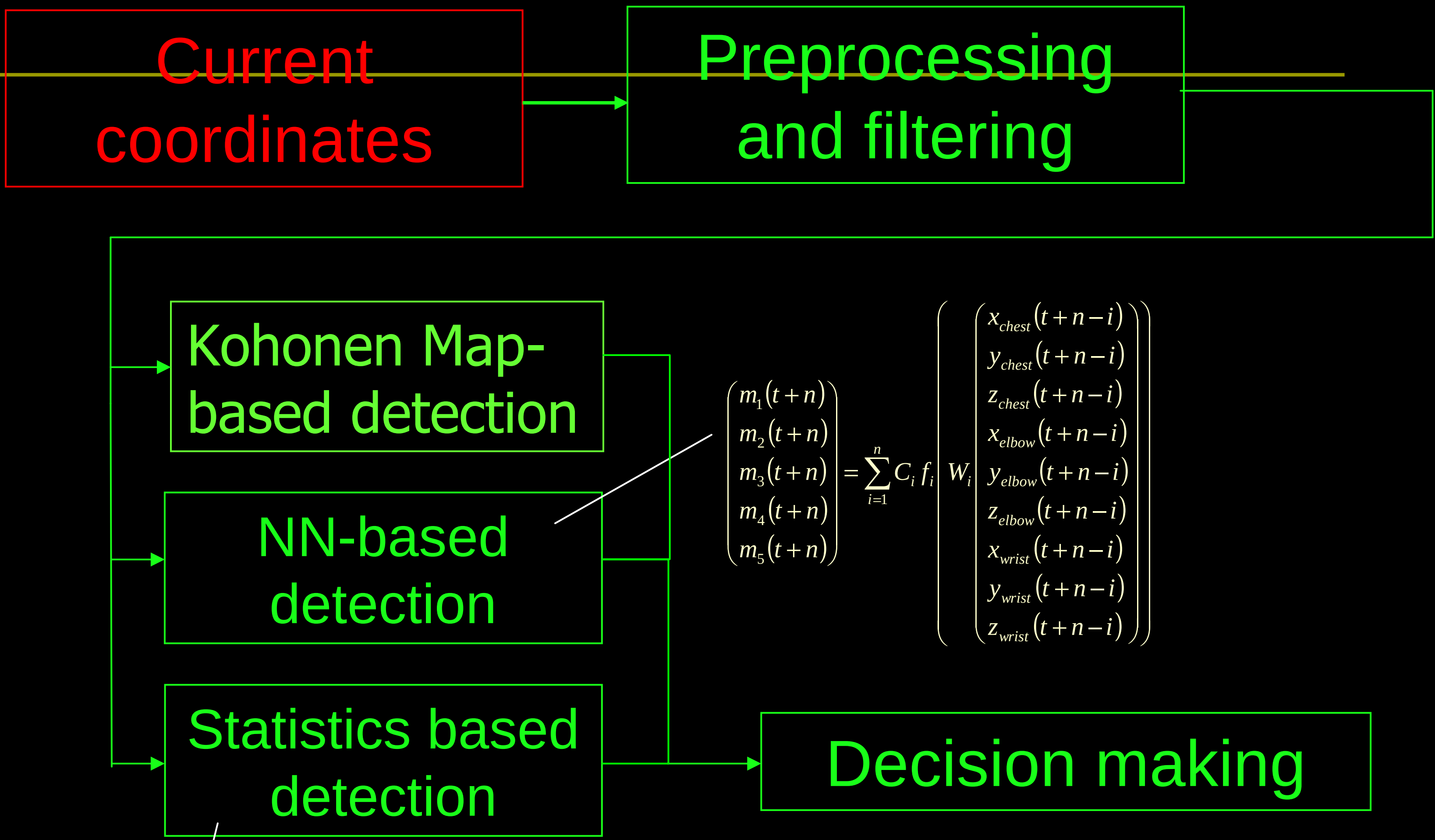
3rd generation Scrub Nurse Robot “Michael”



SNR Control Architecture



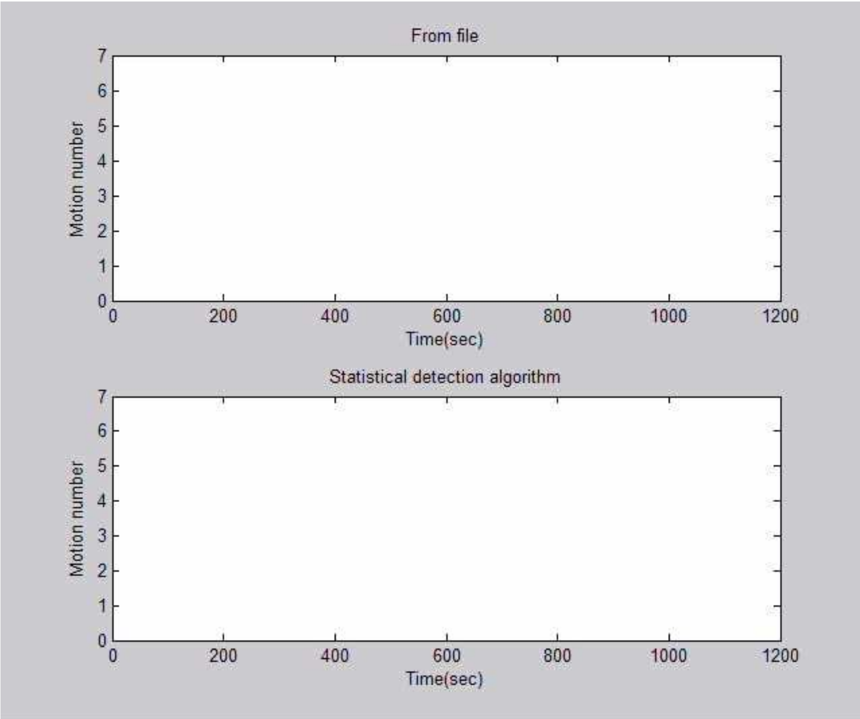
Motion recognition

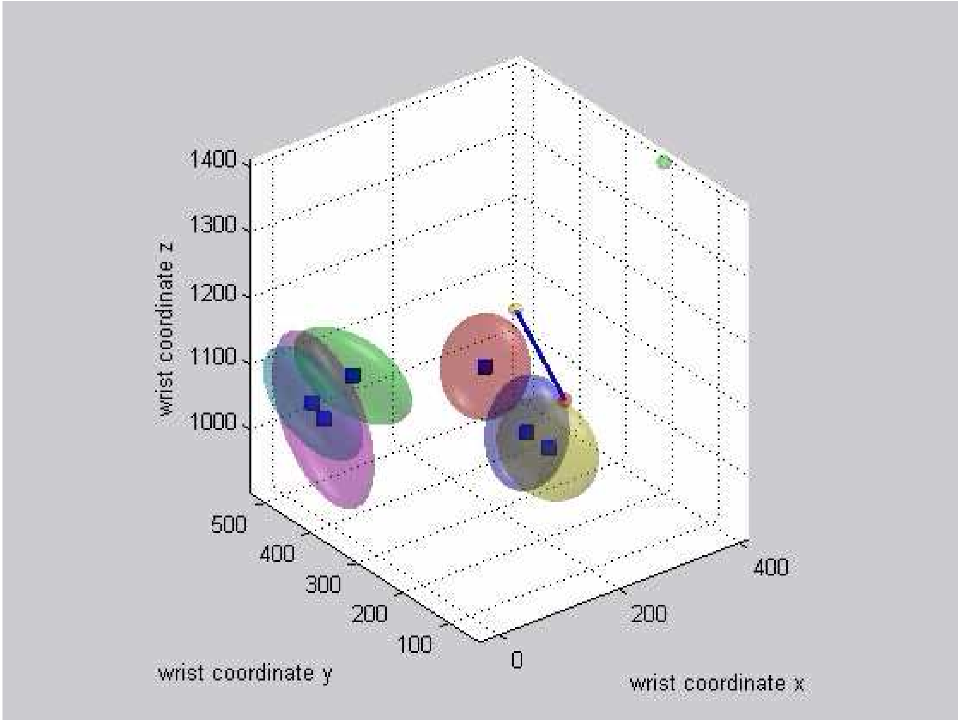


$$\begin{pmatrix} m_1(t+n) \\ m_2(t+n) \\ m_3(t+n) \\ m_4(t+n) \\ m_5(t+n) \end{pmatrix} = \sum_{i=1}^{n} C_i f_i \left(W_i \begin{pmatrix} x_{chest}(t+n-i) \\ y_{chest}(t+n-i) \\ z_{chest}(t+n-i) \\ x_{elbow}(t+n-i) \\ y_{elbow}(t+n-i) \\ z_{elbow}(t+n-i) \\ x_{wrist}(t+n-i) \\ y_{wrist}(t+n-i) \\ z_{wrist}(t+n-i) \end{pmatrix} \right)$$

$$p = \iiint_D \frac{1}{(2\pi)^{\frac{3}{2}}} |\Sigma|^{\frac{1}{2}} e^{\frac{-1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)} dxdydz$$

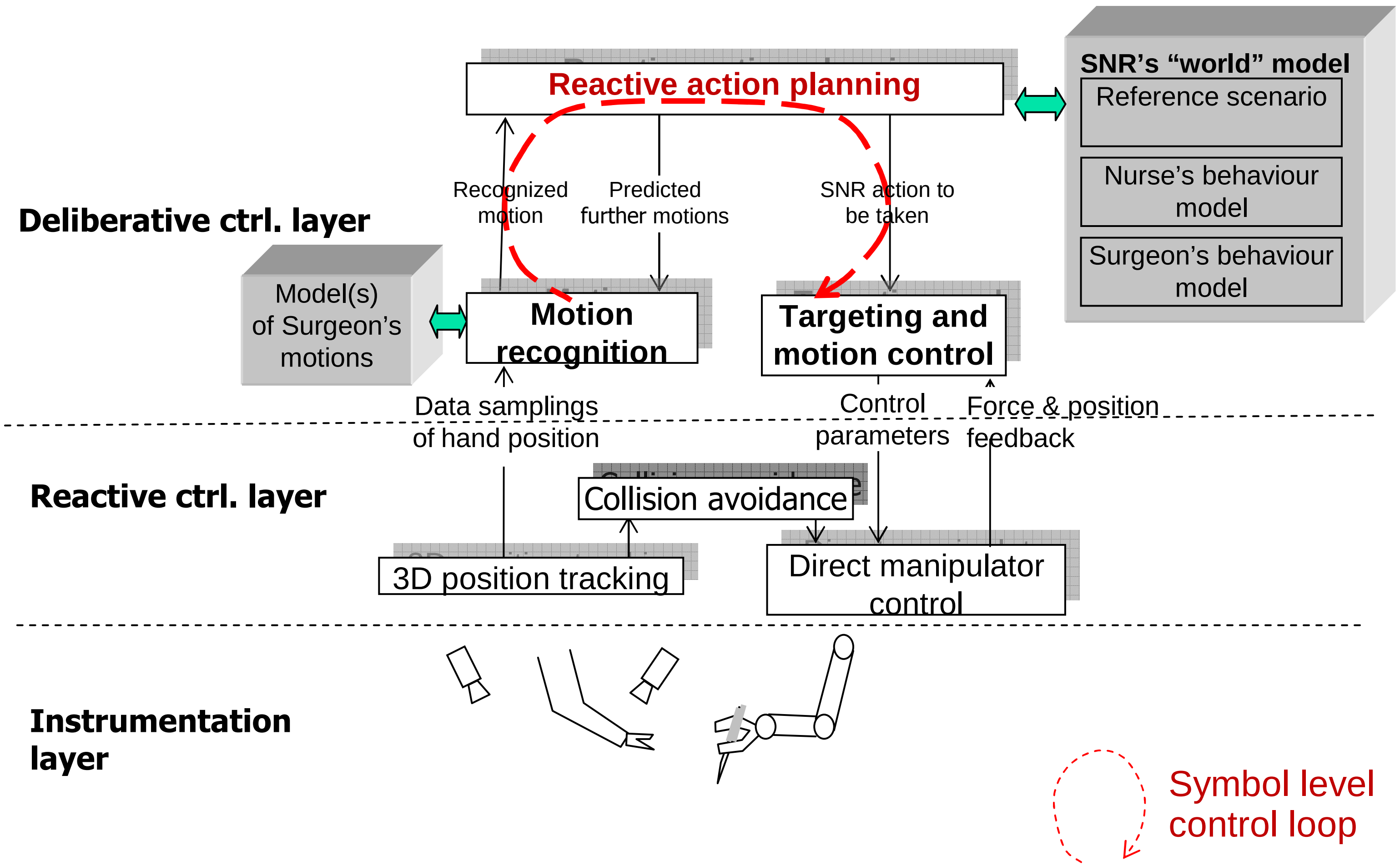
Motion recognition using statistical models



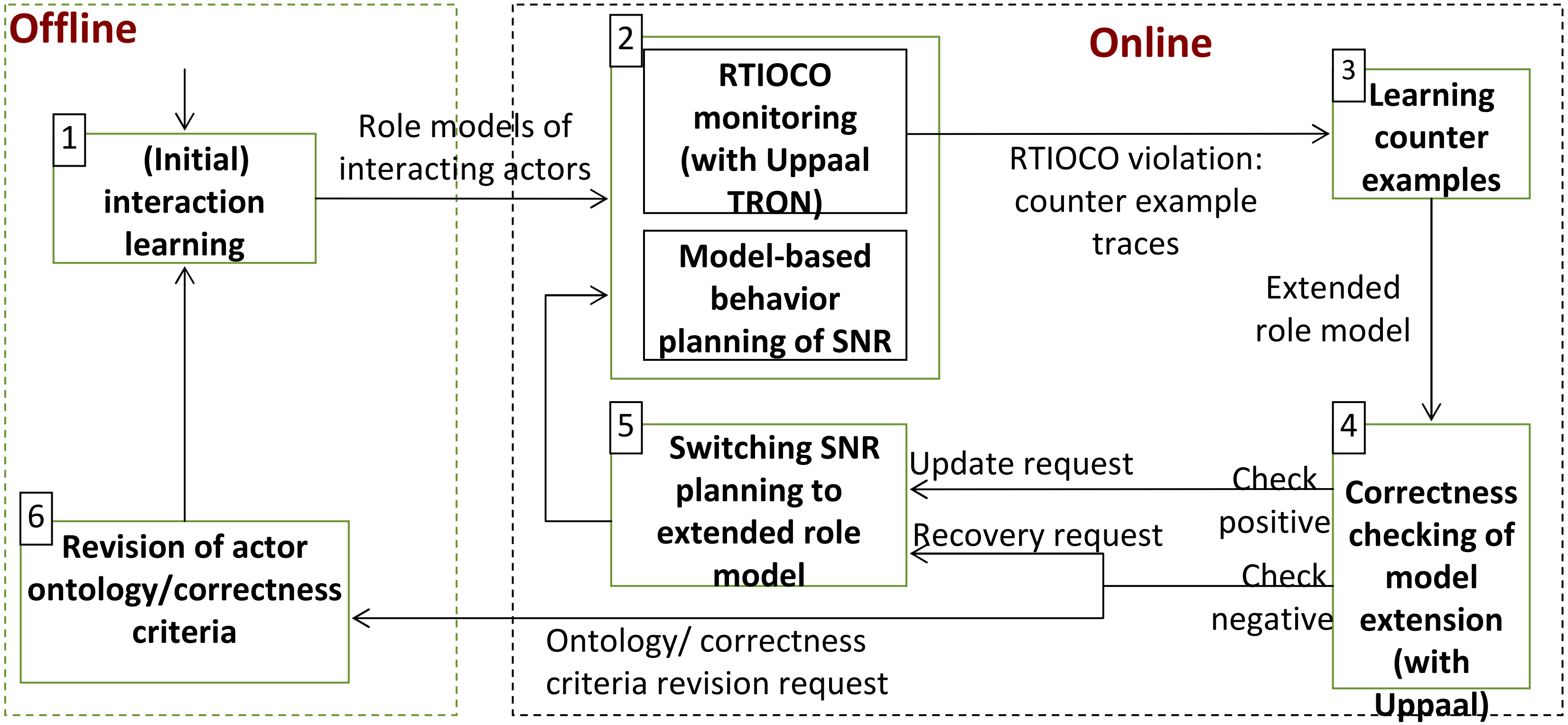


- working->extracting, - extracting->passing, - passing->waiting, - waiting->receiving, - receiving->inserting, - inserting->working

SNR Control Architecture



High-level behavior learning of SNR



(Timed) automata learning algorithm

- Input:
  - Definition of actors' Observable inputs/outputs $\boldsymbol{X_{obs}}$
  - Rescaling operator R: $\boldsymbol{X_{Obs}} \rightarrow \boldsymbol{X_{XTA}}$
    - where $\boldsymbol{X_{XTA}}$ is a model state space
  - Equivalence relation "~" defining the quotient state space $\boldsymbol{X}/_{\sim}$
  - Time-stamped sequence of observed i/o events (timed trace *TTr(Obs)*)
- Output:
  - Uppaal timed automaton *A* s.t.

$$TTr(A) \supseteq_{\mathrm{RTIOCO}} R(TTr(Obs))/_{\sim}$$

Algorithm 1: model compilation (one learning session)

- *Initialization*

$L \leftarrow \{l_0\}$	% L –	set of locations, l_0 – (auxiliary) initial location
$T \leftarrow \varnothing$	% T –	set of transitions
$k,k' \leftarrow 0,0$	% k,k'–	indexes distinguishing transitions between same location pairs
$h \leftarrow l_0$	% h –	history variable storing the id of the motion currently processed in the TTr FIFO E
$h' \leftarrow l_0$	% h' –	variable storing the id of the motion before previous
$h_{cl} \leftarrow 0$	% h_{cl} –	clock reset history
$l \leftarrow l_0$	% l –	destination location of the current switching event
$cl \leftarrow 0$	% cl –	local clock variable of the automaton being learned
$g\_cl \leftarrow \varnothing$	% g_cl -	3D matrix of clock reset intervals
$g\_x \leftarrow \varnothing$	% g_x -	4D matrix of state intervals that define state switching conditions
	% E	*TTr* FIFO, consisting of switching triples: [*target_action_ID, switching time, switching state*]

Encode new motion

Create a new eq.class

Match with existing eq.class

Extend existing eq.class

Encode motion previously observed

Create a new eq.class

```
1: while E ≠ ∅ do
2:     e ← get(E )                          % get the motion switching event record from buffer E
3:     h'← h, h ← l
4:     l ← e[1], cl ← (e[2] - h_cl), X ← e[3]
5:     if l ∉ L then                        % if the motion has never occurred before
6:         L ← L ∪ {l},
7:         T ← T ∪ {t(h,l,1)}               % add transition to that motion
8:         g_cl(h,l,1) ← [cl, cl]           % add clock reset point in time
9:         for all x_i ∈ X do
10:            g_x(h,l,1,x_i) ← [x_i, x_i]  % add state switching point
11:        end for
12:    else                                 % if switching e in existing equivalence class
13:        if ∃k∈ [1,|t(h,l,.)|], ∀x_i∈ X,: x_i ∈ g_x(h,l,k,x_i) ∧ cl ∈ g_cl(h,l,k) then
14:         goto 34
15:        else                             % if switching e extends existing equival. class
16:            if ∃k∈ [1,|t(h,l,.)|], ∀x_i∈ X:  x_i ∈ g_x(h,l,k,x_i)↨^Ri ∧ cl ∈ g_cl(h,l,k)↨^Rcl
17:            then
18:                if cl < g_cl(h,l,k)^- then g_cl(h,l,k) ← [cl, g_cl(h,l,k)^+] end if
19:                if cl > g_cl(h,l,k)^+ then g_cl(h,l,k) ← [g_cl(h,l,k)^-, cl]  end if
20:                for all x_i ∈ X  do
21:                    if x_i < g_x(h,l,k,x_i)^-   then g_x(h,l,k,x_i) ← [x_i, g_x(h,l,k,x_i)^+] end if
22:                    if x_i > g_x(h,l,k,x_i)^+  then g_x(h,l,k,x_i) ← [g_x(h,l,k,x_i)^-, x_i] end if
23:            end for
24:            else            % if switching e exceeds allowed limits of existing eqv. class
25:            k ←|t(h,l,.)| +1
26:            T ← T ∪ {t(h,l,k)}               % add new transition
27:            g_cl(h,l,k) ← [cl, cl]           % add clock reset point in time
28:            for all x_i ∈ X do
29:                g_x(h,l,k,x_i) ← [x_i, x_i]  % add state switching point
30:            end for
31:        end if
32:    end if
33:    a(h',h,k') ← a(h',h,k') ∪ X_c        % add assignment to previous transition
34: end while
```

HRI learning algorithm (3)

Finalize TA syntax formatting

for all $t(l_i,l_j,k) \in T$ **do** % compile transition guards and updates

$g(l_i,l_j,k) \leftarrow$ ‘$cl \in g\_cl(l_i,l_j,k) \wedge \bigwedge_{s \in [1,|X|]} x_i \in g\_x(l_i, l_j, k, x_s)$’

$a(l_i,l_j,k) \leftarrow$ ‘$X_c \leftarrow random(a(l_i,l_j,k)), cl \leftarrow 0$’ % assign random value in *a*

end for

for all $l_i \in L$ ***do***

$inv(l_i) \leftarrow$ ‘$\bigwedge_k g(t_{ki}) \wedge \neg \bigvee_j g(t_{ij})$’ % compile location invariants

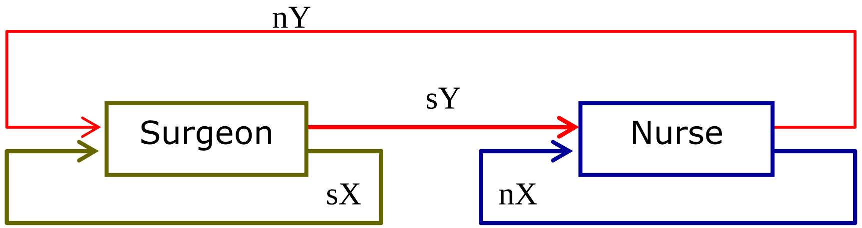
end for

Interval extension operator:

$[.,.]{\updownarrow}^R : [x^-, x^+]{\updownarrow}^R = [x^- - \delta, x^+ + \delta]$, where $\delta = R - (x^+ - x^-)$

Learning example (1)

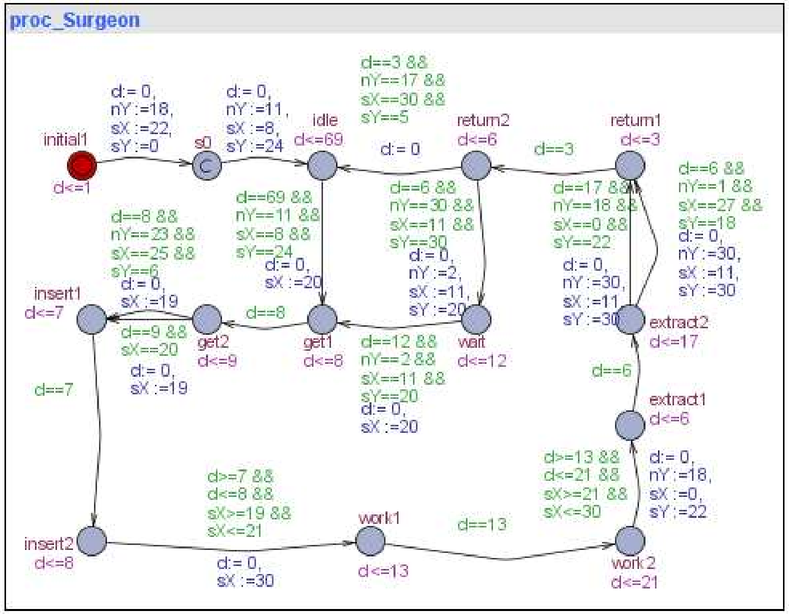
- Given
  - Observation sequence E =
  - System configuration



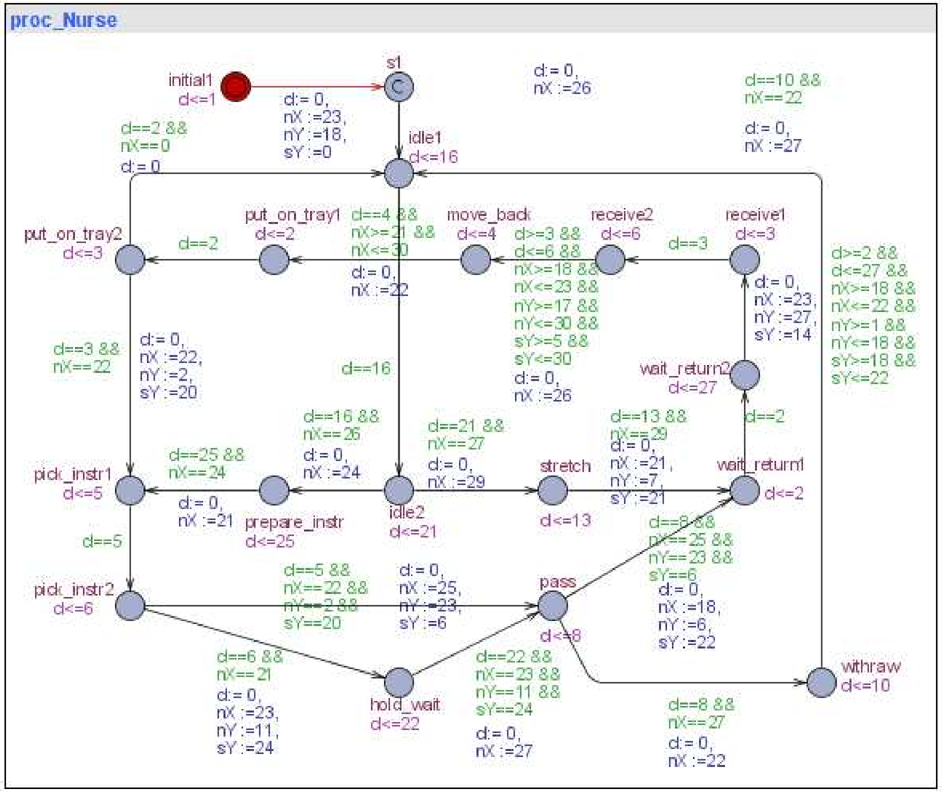
  - Rescaling operator with region [0,30] for all x_i
  - Granularity of the quotient space $\boldsymbol{X}/_\sim$: 2

	$I_2^{Surg}\|O_2^{Surg}$	$I_1^{Nurse}\|O_1^{Surg}$	$I_2^{Nurse}\|O_2^{Nurse}$	$I_1^{Surg}\|O_1^{Nurse}$	Action	
Time	sX	sY	nX	nY	Surgeon	Nurse
1	123	52	214	64	idle	idle
17	$	76	237	34	$	prepare_instr
42	$	93	222	85	$	pick_instr
48	$	57	191	55	$	hold_wait
70	81	123	212	46	get	pass
78	$	132	245	72	$	withraw
79	118	85	$	26	insert	$
86	116	73	$	85	work	$
88	$	73	202	66	$	idle
107	121	59	$	44	extract	$
109	$	77	244	88	$	stretch
122	$	86	259	35	$	wait_return
124	59	116	199	63	return	receive
130	92	139	211	93	wait	move_back
134	$	75	194	55	$	put_on_tray
137	$	104	201	33	$	pick_instr
142	92	110	201	26	get	pass
150	133	68	230	76	insert	wait_return
158	121	76	$	55	work	$
171	146	63	$	27	extract	$
177	138	105	170	22	return	receive
180	147	66	169	62	idle	move_back
184	$	124	268	90	$	put_on_tray
186	$	73	20	20	$	idle

Learning example (2)



||



Does XTA exhibit the same behavior as the traces observed?

- Question 1: How to choose the equivalence relation "~" to define a feasible quotient space?

- Question 2: How to choose the equivalence relation to compare traces *TTr(XTA)* and *TTr(Obs)*?

$$TTr(XTA) = R\,(TTr(Obs))\,/_{\sim} \quad ?$$

$$\downarrow$$

$$TTr(XTA) \supseteq_{\mathrm{RTIOCO}} R(TTr(Obs))\,/_{\sim}$$

How to choose the equivalence relation "~" do define a feasible quotient space?

- State space *granularity parameter* γ_i defines the maximum length of an interval $[x^-_i, x^+_i)$, of equivalent (regarding ~) values of x_i where $x^-_i, x^+_i \in X_i$ for all X_i (i = [1,n]).

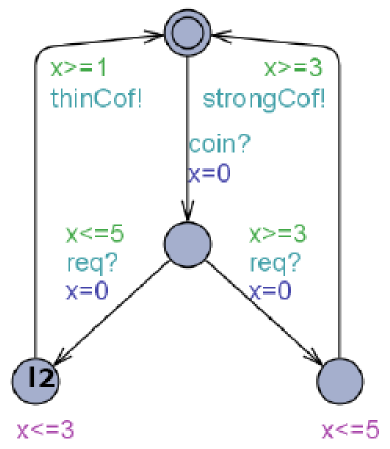
- Partitioning of *dom* X_i (i = [1,n]) to intervals (see line 16 of the algorithm) is implemented using interval expansion operator $[.,.]\updownarrow^R$:

Interval extension operator:
$[.,.]\updownarrow^R$: $[x^-, x^+]\updownarrow^R = [x^- - \delta, x^+ + \delta]$, where $\delta = R - (x^+ - x^-)$

On Question 2:

- Possible candidates:
  - Equivalence of languages?
  - Simulation relation?
  - Conformace relation?
  - ...?

Timed Conformance



- Derived from Tretman's IOCO

- Let **I, S** be timed I/O LTS, P a set of states
- **TTr**(P): the set of *timed traces* from P
  - eg.: σ = coin?.**5**.req?.**2**.thinCoffee!.**9**.coin?
- **Out**(P **after** σ) = possible ***outputs*** and ***delays*** after σ
  - eg. out ({l2,x=1}): {thinCoffee, **0...2**}

- **I rt-ioco S =def**
  - **∀σ ∈ TTr(S): Out(I after σ) ⊆ Out(S after σ)**
  - **TTr(I) ⊆ TTr(s)** *if s and I are input enabled*

- **Intuition**
  - **no illegal output is produced and**
  - **required output is produced (at right time)**

See also [Krichen&Tripakis, Khoumsi]

Conclusions

- Proposed UPTA learning method makes the on-line synthesis of model-based planning controllers for HA robots feasible.
- Other practical aspects:
  - *learning must be incremental*, i.e., knowledge about the previous observations can be re-used;
  - UPTA models can be verified - functional correctness and performance can be verified on the model before used e.g., for planner synthesis;
  - *adjustable level of abstraction* of the generated model to keep the analysis tractable.