Programming Robotic Agents
A Multi-Tasking Teleo-Reactive Approach

Keith Clark
Imperial College and University of Queensland
Peter Robinson
University of Queensland
Nilsson’s
Teleo- Reactive (TR) procedures

- Inspired by control theory ideas
  - continuous monitoring using sensors
  - persistent actuator responses
- Sequences of Guard $\rightarrow$ Action rules clustered into procedures
  - Action can a procedure call or set of concurrent robotic actions
  - Recursive calls
  - Earliest rule with true Guard fired, in each proc.
  - Action to achieve Guard of prior rule in proc., or nil if goal achieved
  - Reconsideration of all fired rules on percepts update
  - TR programmed tasks robust and opportunistic
- Guards can query relations defined using Prolog style rules
  - Give higher level interpretation of rapidly changing percept facts
  - May use declarative model of environment
  - Situation dependent sense data interpretation
From reason based to sensor reactive behaviour

$p(\ldots)\{\n\begin{align*}
H_1 & \rightarrow \text{nil} \quad \% H_1 \text{ goal of } p \\
H_2 & \rightarrow A_1, A_2 \\
H_3 & \rightarrow q(\ldots) \\
H_i & \rightarrow q'(\ldots) \\
\end{align*}\}

$q(\ldots)\{\n\begin{align*}
I_1 & \rightarrow \text{nil} \\
I_2 & \rightarrow r(\ldots) \\
\end{align*}\}

$r(\ldots)\{\n\begin{align*}
P & \rightarrow A_3 \\
\end{align*}\}$

Switches between diff. inter. level procs. as high level inferred beliefs change

Invokes a diff. low level proc. as inter. level inferred beliefs change
Introductory example:
bottle collecting robotic agent

- Enclosed space with green bottles scattered about and a drop area painted red
- No other objects in the space
- Task is to find, grab and deliver a bottle to the red drop area
- And repeat behavior when bottle removed
Top level control program

thing ::= bottle | drop  % enumerative type
dir ::= left | right | centre  % enumerative type
durative forward(num), turn(dir,num)  % modifiable cont. actions
discrete open_gripper, close_gripper  % ballistic actions
percept see(thing,dir,num), holding, gripper_open  % percept types

collect_bottle{
  have_delivered
  see(drop,_,0) & holding
  holding
  true
}

have_delivered <= see(drop,_,0) & next_to_bottle(_) & gripper_open

next_to_bottle: dir
next_to_bottle(Dir) <= see(bottle,Dir,Dist) & Dist≤2
Auxiliary procedures

get_bottle{
  holding
  next_to_bottle(centre) & gripper_open
  next_to_bottle(Dir) & gripper_open
  next_to_bottle
  see(bottle,_, Dist) & Dist>2
  true
}

deliver_bottle{
  see(drop,_,0)
  see(drop,_,Dist) & Dist>0
  true
}

approach: thing

approach(Th){

  see(Th, center,Dist) ~> forward(calcSpeed(Th,Dist))
  see(Th, Dir, Dist) ~> forward(calcSpeed(Th,Dist)), turn(Dir,0.2)
}

% goal to hold a bottle
~> nil

% only active whilst holding true
~> close_gripper
~> turn(Dir, 0.1)
~> open_gripper
~> approach(bottle)
~> turn(left, 0.5)
calcSpeed function

calcSpeed: thing, int -> num

calcSpeed(_,Dist) :: Dist>100 -> 3  \quad % constant speed of 3

calcSpeed(_,Dist) :: Dist>50 -> Dist/100+2  \quad % decreasing speed

calcSpeed(bottle,Dist) :: Dist≤50 -> Dist/50+0.1  \quad % slowing almost to a stop

calcSpeed(drop,Dist) :: Dist≤50 -> Dist/50+1  \quad % slowing to speed 1
Single Task TeleoR

- **Types + modes** ensure correctly typed and grounded actions

- **Extra actions**
  - $A_1 \mid T_1; A_2 \mid T_2; \ldots$ sequences of time limited durative actions
  - $A$ wait $T$ repeat $n$ repeated action if expected effect not observed
  - remember, forget … *BeliefStore* updates
  - Mess to Ag message sending to other agents

- **Extra rule forms** inhibiting firing of other rules, e.g.
  - $G$ while $C$ $\rightarrow$ $A$ inhibits *re-achieving* of $G$, while $C$
  - $G$ until $U$ $\rightarrow$ $A$ inhibits firing of earlier proc. rules until $U$
    allows *over-achieving* of goal of action $A$
  - $G$ time $T$ $\rightarrow$ $A$ inhibits firing of any other proc. rule for $T$ secs

- **Echoes of Brook’s inhibitors** but diff. motivation
  - UQ colleague Ian Hayes, semantically clean program for safety critical test case of pump/gas detector, strong *until*

- **Formal state transition operational semantics**
Example use of: *timed sequence*, *while* rule, *until* rule, *repeat* action

\begin{verbatim}
collect_bottle{
    have_delivered while time 4  ~> turn(right, 0.5) | 2 ; move(1.5)
    see(drop,_,0) & holding ~> open_gripper wait 2 repeat 3
    holding ~> deliver
    true ~> get_bottle
}
\end{verbatim}

dir ::= left | right | centre | dead_centre

\begin{verbatim}
approach(Th){
    see(Th, center,Dist) ~> forward(calcSpeed(Th,Dist))
    see(Th, Dir, Dist) until see(Th, dead_centre, _) ~> 
        forward(calcSpeed(Th,Dist)), turn(Dir,0.2)
}
\end{verbatim}
Three Thread Agent Architecture

BeliefStore

Percepts Handler
Message Handler
ag@host

Atomic Updates

Atomic Evaluation of Rule Guards

TeleoR Evaluator

Sensor data

Incoming Messages

Outgoing Messages

Action control messages

pedro

Other Ags

10
Communicating Bottle Collectors

- Joint Goal to Collect **Total** Bottles
- Tell the other agent new count whenever updated
- Can only see there is a robot in front, not its direction
- Stop when see robot near and communicate relative direction of seen robot (left, centre, right)
- Communicated relative directions used to determine collision avoidance move if both stop
Co-operating bottle collector

int count:=0
int other_count:=0 % other_count updated by agent’s message
% handler on receipt of a collected message
message::= collected(int) | stopped(dir) % reserved type name
belief stopped_received(dir) % beliefs remembered by mess. handler

coop_collect_bottles: int, handle
coop_collect_bottles(Total,OthrAg){
    #count + #other_count ≥ Total  ~> nil
    have_delivered while time 4 ~> 
        turn(right,0.5)|2 ; move_avoiding(OthrAg,1.5) ++
        count:=+ 1 ; collected(#count) to OthrAg
    see(drop,_,0) & holding  ~> open_gripper wait 2 repeat 3
    holding  ~> deliver(OthrAg)
    true  ~> get_bottle(OthrAg)
}
Simple avoidance

move_avoiding: handle, num
move_avoiding(OthrAg,FS){
   near(robot,Dir) ~> stop_avoid(OthrAg,Dir) ++
                   stopped(Dir) to OthrAg

   true ~> move(FS)
}

stop_avoid: handle, dir
stop_avoid(OthrAg,Dir){
   stopped_received(centre) & Dir=centre ~> turn(right,0.2)
   ...
   % other robot has also been stopped
   ...
   near(robot,NDir) & Dir≠Ndir ~> % other robot moving across path
                               stop_avoid (OthrAg,NDir) ++
                               stopped(Ndir ) to OthrAg

   near(robot,NDir) ~> nil % otherwise just stay stopped
}

Multi-tasking extensions of TR

• Single robotic resource
  • Task atomic procedures achieving stable sub-goals
  • Must obtain access to robotic device before entry
  • Release device on exit for whatever reason
• Multiple robotic resources
  • resource meta-type
  • resource names now parameters of task atomic procedures and primitive actions
  • tasks wait in queue for all resources they need for first (and all possible inner calls) to a task atomic procedure
  • resources released on exit of the first call
  • tasks may jump the queue when no resource they need is needed by a task in front
  • parallel use of independent resources
Multi-tasking architecture

BeliefStore
Rules for relations and functions

TR procedures

Message Handler
All incoming messages

Percepts Handler
Dynamic Facts

task1
Control Actions for different robotic resources

task2
Outgoing messages

Sensor data

TR evaluation Threads
Multi-resources tower building
Type Declarations

block ::= 1..15  % range type of 15 integers
table ::= table1 | shared | table2  % enumerated type
place ::= table || block  % union type
resource arm, table  % special class of types

durative pickup(block,table), putdown(block,place,table)

percept on(block,place), holding(arm.block)  % type decl for percepts

Dynamic percept beliefs:
  on(1,table1) on(2,3) holding(arm1,5) …
clear: !place

clear(T) \leq\text{type}(T,\text{table})
clear(B) \leq \text{not}\ on(_,B)

stack: ![\text{block}], \text{table}

stack([B],Tbl) \leq \text{on}(B,\ Tbl)
stack([B1,B2,..Bs],Tbl) \leq \text{on}(B1, B2) \& \text{stack}([B2,..Bs],\ Tbl)

tower: ![\text{block}], \text{table}

tower([B,..Bs],Tbl) \leq \text{clear}(B) \& \text{stack}([B,..Bs],\ Tbl)

located: ![\text{block}], \text{table}

located(B,Tbl) \leq \text{on}(B,Tbl)
located(B,Tbl) \leq \text{on}(B,B') \& \text{located}(B',\ Tbl)

canReach: ![\text{arm}], \text{table}

canReach(_,\text{shared})
canReach(arm1,table1)
canReach(arm2,table2)
makeTower procedure - tests inferable beliefs

task_start makeTower

makeTower: [block], table

makeTower(Bs, Tbl) {
  % Tbl is where tower must be built, % always table1 or table2. Bs a list of block labels
tower(Bs, Tbl) ~> nil

stack(Bs, Tbl) & Bs=[B,..] ~> unpile(B, Tbl)

Bs=[B] ~> moveAcrossToTable(B, Tbl)

Bs=[B, B’,..Bs’] & tower([B’,..Bs’], Tbl) ~> moveAcrossToBlock(B, B’, Tbl)

% Actions of above three rules should all achieve tower(Bs, Tbl)

Bs=[B,..RestOfBs] ~> makeTower(RestOfBs, Tbl)
}

% Last rule action should achieve guard of rule above
moveAcrossToTable – will use both arms if need be

moveAcrossToTable: block, table
moveAcrossToTable(B,Tbl){
  on(B,Tbl) ~> nil
  canReach(Arm,Tbl) & located(B,Tbl’) & canReach(Arm,Tbl’) ~> moveToTable(B,Tbl’,Tbl,Arm)
  located(B,Tbl’) & canReach(Arm’,Tbl’) ~> moveToTable(B,Tbl’,shared,Arm’)
}
% above rule for when B not on Tbl or shared
% will make use of the other arm for move to shared

task_atomic moveToTable(block, table, table, arm)
  % makeTower task must have acquired needed table and arm
  % resources before moveToTable procedure is entered
Future Possible Extensions

• ROS interface (in progress)
• MQTT interface from QuLog (in addition to Pedro)
• Parallel calls to TeleoR procedures as a rule action
• TeleoR plans with ?Goal achieve rule actions
• Non-deterministic plan call selection rules
  
  ?Goal :: BSQuery --> PlanCall
  +Fact :: BSQuery --> PlanCall
  -Fact :: BSQuery --> PlanCall

  as in BDI architectures such as AgentSpeak and Jason
• Plan failures with backtracking to ?Goal actions
• Conditional Timed Action Sequence Procedures
• Task priorities
• Uncertain reasoning
• Cost/benefit choice of rule to fire
• Collaborations welcomed
More info and software

Clark & Robinson, *Programming Robotic Agents*,
to be published by Springer early 2015
Draft chapters from k.clark@imperial.ac.uk on request

Clark & Robinson, *QuLog: Engineering Agent Applications*,
to be published by Springer middle 2015
Draft chapters from k.clark@imperial.ac.uk on request

TeleoR and QuLog Software via http://staff.itee.uq.edu.au/pjr/
late 2014, earlier from pjr@itee.uq.edu.au on request