Task Control Architecture: Structured Control for Autonomous Robots

What is TCA?

“TCA is a high-level robot operating system that provides an integrated set of commonly needed mechanisms to support
• distributed communications,
• task decomposition,
• resource management
• execution monitoring
• and error recovery.”
What is Task Control?

“The term task control refers to the problem of coordinating perceptual, planning and execution components of a robot system to achieve a given set of goals.

[...]

While the Task Control Architecture does not itself provide behaviors for particular tasks, it does provide designers with control constructs for developing such behaviors and software utilities for implementing the necessary control decisions.”
Deliberation and Reactivity

- deliberation
- feed-forward
- nominal

“structured control”

- reaction
- feed-back
- exceptional
Deliberation and Reactivity

TCA takes the essence of the deliberative approach to be the top-down decomposition of tasks into subtasks, specifying current and future activities and constraints (temporal, resource) among them.

Reactivity means that the system detects and makes appropriate responses to changes in its environment.

Deliberation is more concerned with the way subtasks interact, and reactivity is more concerned with the "here and now."

Note
"reactive" does not necessarily imply "reflexive" (direct connections between sensors and effectors)
TCA Modules

A robot system built using TCA consists of
• task-specific modules that communicate by sending messages
• via a general purpose, reusable central control module.

The task modules perform all robot-dependent information processing, while the central control module is responsible for routing messages and maintaining task control information.
**TCA Constructs**

The task modules use the TCA **control constructs** to specify information such as
- how to decompose tasks,
- when subtasks should be planned and executed,
- when and how to monitor the environment,
- and how to react to exceptional situations (by terminating tasks, adding new subtasks, reordering tasks, etc).

TCA utilizes this control information in turn to **schedule and coordinate the actions of the modules** and to respond to change in a context-sensitive fashion.
TCA Constructs

Note

Most of TCA control constructs can be incrementally added with little or no change to existing systems.

In particular,
• modules can be added that perform new tasks,
• temporal constraints can be modified to increase concurrency,
• new monitors and exception handlers can be easily added to existing tasks and
• new implementations can replace existing modules without changing communication protocols

re-usability!!!
Example: Ambler

Ambler is a six-legged robot designed for planetary exploration, that autonomously traverses rugged terrain.

The Ambler system uses TCA to integrate
• real-time control,
• 3D perception
• planning algorithms
• monitoring and error recovery procedures.

Deliberative aspects of TCA are used to plan safe and energy efficient moves based on kinematic, pragmatic and terrain constraints.

The reactive components are used to detect and handle deviations arising from sensor and actuator uncertainty.
Related Work

- NASREM
- BB1
- TCA
- RAP
- SSA
Task Control Architecture

TCA provides a comprehensive set of control constructs for developing deliberative and reactive robot behaviors.

The control constructs include support for:

1) Distributed inter-process communication
2) Task decomposition and temporal constraints between subtasks
3) Resource allocation and management
4) Execution monitoring
5) Exception handling
Ambler walking system

Modules of Ambler
TCA Messages

TCA provides several classes of messages, each with somewhat different semantics:

1) **Inform messages** are one-way communications used to asynchronously pass information from one module to another.

2) **Query messages** are two-way returning a reply to the sending module. Query messages are blocking, pending receipt of the reply.

3) **Goal messages** are used to decompose tasks into subtasks. They are non-blocking, to enable planning and plan execution to occur concurrently.

4) **Command messages** are similar to goal messages, but they represent executable actions of the robot system.

5) **Monitor messages** are used to set up execution monitors.

6) **Exception messages** are used for handling exceptional situations.
Running a TCA System

To run the system,
• one invokes the central control process, indicating how many modules will be running and where to log the message traffic;
• each robot-specific module is then started;
• the modules communicate with the central control module using a library of function calls;
• modules first connect to the central control module, and register, which messages they handle, their formats, and other relevant information, such as the monitors and exception handlers used by the system;
• modules then call a function that waits to receive messages and dispatches them to the appropriate handlers.

Note
Modules can run on different CPUs.
Example: Ambler

Task tree for autonomous walking
Example: Ambler

1. Walking is initiated by entering a series of waypoints at a graphical user interface.

2. When a route is determined, the user interface module sends a series of "Traverse Arc" goal messages, one for each waypoint.

3. The central control forwards the first "Traverse Arc" message to the gait planner module, queuing the remaining ones.

4. Within the gait planner, TCA decodes the message data and invokes the appropriate message handling procedure.

5. This procedure issues a query message, requesting the Ambler's current position. TCA forwards the request to the controller module and routes the response back to the gait planner.

6. The gait planner calculates an arc that passes from the current location to the goal location and issues a "Take Steps" goal message, passing the current position and desired arc to follow.

7. When the message handler completes, the central control is notified that the gait planner is no longer busy.
Example: Ambler

1. Walking is initiated by entering a series of waypoints at a graphical user interface.

2. When a route is determined, the user interface module sends a series of "Traverse Arc" goal messages, one for each waypoint.

3. The central control forwards the first "Traverse Arc" message to the gait planner module, queuing the remaining ones.

4. Within the gait planner, TCA decodes the message data and invokes the appropriate message handling procedure.

5. This procedure issues a query message, requesting the Ambler's current position. TCA forwards the request to the controller module and routes the response back to the gait planner.

6. The gait planner calculates an arc that passes from the current location to the goal location and issues a "Take Steps" goal message, passing the current position and desired arc to follow.

7. When the message handler completes, the central control is notified that the gait planner is no longer busy.
**Example: Ambler**

1. Walking is initiated by entering a series of waypoints at a graphical user interface.

2. When a route is determined, the user interface module sends a series of "Traverse Arc" goal messages, one for each waypoint.

3. The central control forwards the first "Traverse Arc" message to the gait planner module, queuing the remaining ones.

4. Within the gait planner, TCA decodes the message data and invokes the appropriate message handling procedure.

5. This procedure issues a query message, requesting the Ambler's current position. TCA forwards the request to the controller module and routes the response back to the gait planner.

6. The gait planner calculates an arc that passes from the current location to the goal location and issues a "Take Steps" goal message, passing the current position and desired arc to follow.

7. When the message handler completes, the central control is notified that the gait planner is no longer busy.
Example: Ambler

1. Walking is initiated by entering a series of waypoints at a graphical user interface.

2. When a route is determined, the user interface module sends a series of "Traverse Arc" goal messages, one for each waypoint.

3. The central control forwards the first "Traverse Arc" message to the gait planner module, queuing the remaining ones.

4. Within the gait planner, TCA decodes the message data and invokes the appropriate message handling procedure.

5. This procedure issues a query message, requesting the Ambler's current position. TCA forwards the request to the controller module and routes the response back to the gait planner.

6. The gait planner calculates an arc that passes from the current location to the goal location and issues a "Take Steps" goal message, passing the current position and desired arc to follow.

7. When the message handler completes, the central control is notified that the gait planner is no longer busy.
Example: Ambler

1. Walking is initiated by entering a series of waypoints at a graphical user interface.

2. When a route is determined, the user interface module sends a series of "Traverse Arc" goal messages, one for each waypoint.

3. The central control forwards the first "Traverse Arc" message to the gait planner module, queuing the remaining ones.

4. Within the gait planner, TCA decodes the message data and invokes the appropriate message handling procedure.

5. This procedure issues a query message, requesting the Ambler's current position. TCA forwards the request to the controller module and routes the response back to the gait planner.

6. The gait planner calculates an arc that passes from the current location to the goal location and issues a "Take Steps" goal message, passing the current position and desired arc to follow.

7. When the message handler completes, the central control is notified that the gait planner is no longer busy.
Example: Ambler (cont.)

8. When the gait planner is free, TCA forwards it the "Take Steps" message invoking the appropriate procedure.

9. This handler first checks whether the robot is at the goal location. If so, it returns immediately; if not, it calculates a body move that maximizes forward progress.

10. It then issues a query to determine the best place to move the leg. The footfall planner handles this message by querying for an elevation map (handled by the local terrain mapper) and computing the desired information.

11. Once the calculations are completed, the gait planner issues a "Place Leg" goal message, a "Move Body" command, a monitor to ensure that the move was executed correctly and another "Take Steps" message.

12. TCA immediately forwards the "Place Leg" message to the leg recovery planner, which obtains an elevation map and uses it to compute an energy-efficient path for the leg to follow.

13. The leg recovery planner then issues a "Move Leg" command, which is routed to the controller.

14. Concurrently, TCA sends the pending "Take Steps" message to the gait planner, which begins to plan out the next step.
Example: Ambler (cont.)

15. After the controller finishes handling the "Move Leg" command, it is forwarded the "Move Body" command.

16. When the body move completes, the central control activates the monitor, sending a query to see if the desired move was achieved.

17. If not, an exception is raised and the gait planner is invoked to replan from the current position.

18. Otherwise, TCA waits for the next "Move Leg" message to be issued, and forwards that to the controller, beginning another walking cycle.
Example: Ambler (cont.)

The Ambler modules instruct TCA to perform several other tasks during walking:

1. The scanner interface module indicates that it should be notified after each "Move Body" message completes, in order to acquire a new laser range image.

2. The error recovery module instructs TCA to invoke a stability monitor prior to executing each leg and body move; if the planned move is found to be potentially unstable, the monitor aborts the walking task and the system comes to a graceful halt.
Mechanisms for Deliberation: Task Trees

At the heart of TCA is a hierarchical representation of task/subtask relationships: task trees

- has goal messages as non-terminal nodes, and executable command and monitor messages at the leaves

- TCA constructs task trees automatically: whenever a goal, command, or monitor message is issued the central control creates a node and adds it as a child of the node that issued the message.
Mechanisms for Deliberation: Constraints

Temporal constraints between nodes are used to schedule task planning and execution: messages are queued until their temporal constraints are satisfied.

• **sequential-achievement constraint** between two nodes implies that all command and monitor messages under the first node (leaves of its subtree) must be handled before any of those under the second node.

  Example:
  sequential-achievement constraint between the "Traverse Arc" nodes indicates that arcs must be traversed in order.

• **delay-planning constraint** implies that a goal message should not be handled (decomposed into sub goals) until the previous task has been completely achieved.

This combination of hierarchical task decomposition and temporal constraints form TCA's representation of plans.
Mechanisms for Deliberation: Resources

• Another deliberative method for controlling tasks is to explicitly manage the robot’s physical and computational resources.

• A TCA resource is a set of message handling procedures and a capacity.

• TCA ensures that a resource's capacity is never exceeded, queuing messages if necessary until the resource becomes available.

• By default, TCA groups all message handlers registered by a module into a resource of unit capacity.
  
  Example: Ambler's gait planner handles the "Traverse Arc" and "Take Steps" messages, among others.

• TCA ensures that a module will receive only one of these messages at a time. Modules may also define additional resources, enabling them to handle multiple messages concurrently.
Mechanisms for Reactivity: Monitors

TCA provides several constructs for monitoring changes in the Environment: TCA monitor is a message that performs some action when a specified condition is triggered.

- A point monitor, which tests its condition just once, is useful for determining whether tasks have been executed according to plan.

**Example:**
Point monitor compares actual Ambler position against planned position. If difference exceeds a given threshold (e.g., if the robot slipped), the monitor’s action is triggered to replan subsequent steps.
Mechanisms for Reactivity: Monitors

- Polling and interrupt-driven monitors are used to detect unexpected changes, operate concurrently with planned actions.

- Both types of monitors test their conditions repeatedly, continuing either for a given duration or until a specified event occurs.

- For polling monitors, the central control module issues the condition message at a fixed frequency.

- For interrupt-driven monitors, TCA informs modules when to set up new monitors and when to cancel them.

- The modules have responsibility for informing the central control whenever the monitor’s condition holds, at which point TCA issues the associated action message.
Mechanisms for Reactivity: Exceptions

- Message handlers can also detect execution errors or plan failures directly, which they signal by raising exceptions. Example: a controller module may sense that a motor is stuck, or a planning module may not be able to find a clear path to a goal.

- Exception messages can be associated with task tree nodes to deal with such conditions.

- Exception handling is structured hierarchically: when an exception is raised, TCA searches up the task tree to find and dispatch a message designated to handle that exception.

- If message handler determines it cannot actually fix the problem, it reissues the exception and the search continues up the tree.

- If the root node is reached, TCA simply terminates the task.
Concurrent planning and execution in TCA

- overconstrain system in initial phase, then slowly relax constraints until desired performance is obtained.

- removal of temporal constraints may allow concurrent planning and execution

- however, removal of temporal constraints may cause problems: planning often moves quicker than execution -> planner quickly gets several steps ahead

- To prevent this, a delay-planning constraint is added between one "Take Steps" node and the node directly preceding its parent -> one step look-ahead planning
Conclusions

- no clear “recipe” or guidelines for how to structure and decompose message handlers (into task nets)
- no formal method for analyzing an establishing constraints
- mixture of middleware (communication infrastructure) and control architecture
- immediate use for others / generalization ?
+ exception handling mechanisms
+ verified in a number of field tests