Tekkotsu
A Rapid Development Framework for Robotics

Masters Thesis Presentation
Ethan Tira-Thompson
Advisor: Dr. David S. Touretzky
Contents

• Part I - Design Considerations
• Part II - Implementation
• Part III - Evaluation
• Concluding Remarks
Part I
Design Considerations
Design Dimensions

• Goal: provide a modular framework for developing robot behaviors

• Six fundamental classes of primitives:

|-------------------|-----------------|-----------------|
1. User Interface

• Two principles:

  • Multiple interfaces – remote and physical
    • Model-View-Controller – Abstract input method from implementation
    • Improves portability and accessibility
  
  • Transparency
    • Provide developer with access to monitor internal state
    • Reduces development time
2. Perception

- Primitives should convert real-time streams of data to event notifications

- Three main advantages:
  - Reduce recomputation – data is only processed once, regardless of number of dependencies
  - Lower learning curve – users don’t need to know where data processing is being done, only that it is available
  - Modularity - weak linking, swap implementations
3. Control Structure

- Number of different control strategies
  - Subsumption (Brooks, 1986)
  - State machines (everybody)
  - Logical formalisms (e.g. Golog - Levesque, et.al 1997)
3. Control Structure (cont.)

• Hierarchical State Machine
  • Declarative construction
    • set parameters and connections
    • very easy to develop, especially with a graphical editor
  • Separates decisions (transitions) from actions (states)
    • Increases code reusability
4. Manipulation

• Motion primitives directly control physical machinery in real-time.

• Need to marry event driven processing with the real-time processing

• Conflicts can occur in complex behaviors
  • Serialize access?
  • Priority override?
5. Mapping

• Robots need to track current location as well as locations of objects in its environment

• No commercially available personal robotic platforms do mapping or localization
6. Memory/Learning

- Memory structures provide communication between states
- Memory of experiences is needed for learning
Advanced Features

• Navigation
  • Follow a path, perhaps performing some task along the way
  • Combines perception with manipulation

• Attentional Control
  • Decide where to direct sensor resources

• Complex Behaviors
  • Search for an object, or monitor a condition and report when it changes
Part II
Implementation
Platform Information
Tekkotsu Data Flow

- **MainObj**
  - EventRouter
  - Vision
  - Behaviors can play sounds anytime
  - System sends state information (~32ms)
  - System sends camera frames (~30fps)

- **WorldState**
  - State
  - Can access state anytime for reactive/open loop control
  - Behaviors request lock on MotionCommands to make direct function calls on them

- **MotionManager**
  - motman
  - Added to Motion Manager
  - Requests sound buffer
  - Returns sound buffer by mixing current sounds

- **MotoObj**
  - Requests joint positions (~32ms)
  - Sends new joint positions to system

- **SoundPlay**
  - Requests sound buffer (~32ms)
  - Sends 32 ms of sound to system

- **SoundManager**
  - sndman
  - Can play sounds at any time

- **MotionCommands**
  - dynamically created
  - Behaviors created by currently active Behaviors

- **TinyFTPD**
  - From Sony’s sample code, allows you to FTP files during run time.
  - Self contained

- **Key**
  - Pre-emptive Process
  - Shared Memory Region
  - Unshared Global Variable
1. User Interface
1. User Interface
1. User Interface
2. Perception

- Global EventRouter maintains mapping of listeners to event streams
  - Centralized access is easy to learn
- Behaviors can subscribe to event streams at different granularities
  - Generator
  - Generator and Source
  - Generator, Source, and Type
2. Perception (cont.)

- Available Generators:
  - sensor – new sensor readings available
    - button – button status
  - textMsg – user entered text from GUI or console
  - timer – timer has expired (not broadcast)
  - estop – emergency stop status
  - stateMachine – state transition occurred
2. Perception (cont.)

- Available Generators:
  - audio – sounds start/stop playback
  - motman – MotionCommands added/removed
  - locomotion – movement through world
  - power – battery and temperature status
  - vision generators – various stages of processing
2. Perception

System
OFbkImageVectorData
Direct function call

MMCombo
DataEvent<OFbkImageVectorData*>
visOFbkEGID
visRawCameraSID

CDTGenerator
SegmentedColorFilterBankEvent

CMVision
SegmentedColorGenerator
SegmentedColorFilterBankEvent

RLEGenerator
SegmentedColorFilterBankEvent

RegionGenerator
SegmentedColorFilterBankEvent

JPEGGenerator
FilterBankEvent (grayscale)

InterleavedYUVGenerator
FilterBankEvent

JPEGGenerator
FilterBankEvent (color)

SegCamBehavior

RawCamBehavior

BallDetectionGenerator
VisionObjectEvent
(pink)

BallDetectionGenerator
VisionObjectEvent
(blue)

BallDetectionGenerator
VisionObjectEvent
(hand)

Legend

Ellipse: Non-Behaviors

Box: Behaviors

Green: DataEvents

Yellow: FilterBankEvents

Orange: SegmentedColorFilterBankEvents

Purple: VisionObjectEvents

Wireless
3. Control Structure

- Supported Architectures:
  - Hierarchical State Machine
    - Supports “fork” of activation
  - Subsumption-like
    - Priority overrides allow motion primitives to override each other
3. Control Structure (cont.)

• Process Management
  • Each behavior can be thought of as a non-pre-emptive process
    • Low overhead
    • Must give up processor control
  • Events carry not only information, but also a token representing processor control
4. Manipulation

• Motion Manager uses priority levels to resolve conflicts between primitives
  • Ties are resolved with a weighted average

• Primitives include:
  • Walking, head pointing, LED special effects, motion script playback, PID control, direct posture control, emergency stop, tail wag
5. Mapping

- Under development
- Uses Visual Routines based processing
6. Memory/Learning
Part III

Evaluation
Comparison to Other Frameworks

- LEGO MindSTORMS
  - Standard Environment
  - Third-party firmware (legOS/leJOS)
- Sony Master Studio (AIBO)
- BeeSoft (Real World Interfaces B14/B21)
Comparison to LEGO MINDSTORMS (Standard)

http://mindstorms.lego.com/eng/inventions/invention.asp?id={D56DB42C-2C7B-4BE4-A9BE-E0AA6CA0A224}&slotN=1

http://mindstorms.lego.com/eng/inventions/invention.asp?id={EE1A945A-FF21-4361-B192-E8ADB5D38F9F}&slotN=1
Comparison to LEGO MINDSTORMS (Standard)

• Pros:
  • Graphical editor is very easy to learn
  • Large support community
  • Can also use source code – Not Quite C, Ada, Scheme
Comparison to LEGO MINDSTORMS (Standard)
Comparison to LEGO MINDSTORMS (Standard)

• Cons:
  • Very limited number of variables, no dynamic memory
  • No call stack (no recursion or functions)
  • No floating point math
Comparison to LEGO MINDSTORMS (Third-party)

• Pros:
  • Enables nearly full support for common languages (C, Java)

• Cons:
  • Still running on very limited hardware
  • Minimal toolset
Comparison to Sony Master Studio
Comparison to Sony Master Studio

• Pros:
  • Graphical state machine editor is easy to use and more powerful than Mindstorms
  • Includes primitives for on board visual and auditory processing
    • Recognition of pink ball, action cards
    • Speech recognition (static vocabulary)
  • Has a source code language – R-Code
Comparison to Sony Master Studio
Comparison to Sony Master Studio
Comparison to Sony Master Studio

• Cons:
  • Even fewer variables than MINDSTORMS
  • Very limited dynamic/real-time abilities
Comparison to BeeSoft
Comparison to BeeSoft

• Pros:
  • Designed with clustering in mind
  • Flexible hardware configurations

• Cons:
  • Little or no support included for manipulators, video, or audio processing
Comparison to BeeSoft

The BeeSoft Simulator simulates the physical robot only.
Tekkotsu

- **Pros:**
  - Extensive tools, libraries
  - Includes primitives for visual processing
  - Uses a well known language (C++)
  - Open Source

- **Cons:**
  - Requires advanced C++ knowledge (i.e. templates)
  - Large codebase
**State Machine Sample:**

```cpp
template virtual void setup() {
    StateNode *wait = start = addNode(new WaitNode("Wait", this, bandit));
    StateNode *left = addNode(new PressNode("Left", this, LFrLegOffset + KneeOffset));
    StateNode *right = addNode(new PressNode("Right", this, RFrLegOffset + KneeOffset));
    StateNode *decide = addNode(new DecideNode("Decide", this, bandit, left, right));
    StateNode *recover1 = addNode(new OutputNode("\BadPressLeft", this, std::cout, wait));
    StateNode *recover2 = addNode(new OutputNode("\BadPressRight", this, std::cout, wait));
    left -> addTransition(new SmoothCompareTrans<float>(wait, &state -> piddles[LFrLegOffset + RotatorOffset], ...));
    right -> addTransition(new SmoothCompareTrans<float>(wait, &state -> piddles[RFrLegOffset + RotatorOffset], ...));
    wait -> addTransition(new TimeOutTrans(decide, 2000));
    left -> addTransition(new TimeOutTrans(recover1, 1500));
    right -> addTransition(new TimeOutTrans(recover2, 1500));
    reexecute();
    StateNode::setup();
}
```

**State Node Sample:**

```cpp```
virtual void processEvent(const EventBase & event) {
    if (event.getGeneratorID() == EventBase::visObjEID) {
        // in case the head isn't pointing straight forward, we'll strafe
        float x = 120.0f * cos(state -> outputs[HeadOffset + PanOffset]);
        float y = 120.0f * sin(state -> outputs[HeadOffset + PanOffset]);
        float z = static_cast<const VisionObjectEvent*>(event) -> getX();
        MMAccessor<WalkMC>(walker_id) -> setTargetVelocity(x, y, z);
    }
}
```
Scoring Rubric

• List of qualities for each of the six classes of primitives

• 1 point is assigned to each framework for each applicable quality

• The six classes are equally weighted and summed to calculate the final score
Scoring Rubric

- User Interface
  - Allows both remote and hands-on control
  - Transparency of robot state
  - Can create new interfaces
  - Includes tools for robot status reports
  - Portability
Scoring Rubric

• Perception
  • Event system
  • Can do visual processing
  • Can do auditory processing
  • Can add new sensor events
  • Portability
Scoring Rubric

- Control Structures
  - Loops
  - Arrays
  - Dynamic Memory
  - Recursion
  - Modularity

- State Machine
  - Threads
  - Low Learning Curve
  - Portability
Scoring Rubric

- Manipulation
  - Canned script playback
  - Sensor access
  - Kinematics library
  - Real-time control
  - Portability
Scoring Rubric

• Mapping
  • Feasibility

• Machine Learning
  • Feasibility
Categorical Summary

![Bar chart showing categorical summary of different software tools. The categories are User Interface, Perception, Control Struct., Manipulation, Mapping, and Mem./Learning. The tools compared are MindStorms, BeeSoft, legOS/leJOS, and Tekkotsu.](image)
Final Summary

MindStorms: 1.69
legOS/leJOS: 4.87
Master Studio: 1.61
BeeSoft: 4.83
Tekkotsu: 5.89
Concluding Remarks
Future Development

- Multi-Robot Communications
- Graphical State Machine Editor/Viewer
- Applications in Education
- Training of Behaviors
- Standard Robotics Platform
Conclusions

• Robot manufacturers need to start considering a standard software platform
  • Attempts are under way, e.g. Webots

• We need more complete software libraries to propel robotics research (e.g. CMVision)

• Tekkotsu is now in use by at least a dozen universities around the world

• 109 member mailing list
Acknowledgements

• AIBO Lab: Alok Ladsariya, Thomas Stepleton, Neil Halelamien, Jordan Wales
  • Director: Professor David S Touretzky

• CORAL Group, particularly Scott Lenser, for helping to get us off the ground
  • Director: Professor Manuela Veloso

• My parents and Sylvia for keeping me sane

• This work partially funded by Sony Corp.
For Further Information

- Tekkotsu Homepage
  - http://www.tekkotsu.org
  - http://www-2.cs.cmu.edu/~tekkotsu