UNIVERSITY AT ALBANY
State

Sony's Aibo

Tekkotsu Basics
Behaviors & Events

Robotics Seminar CSI445/660
Spring 2009, MW 4:15-5:35

Instructors: Tomek Strzalkowski,
Nick Webb & Michael Ferguson
University at Albany, SUNY

UNIVERSITY AT ALBANY
State University of New York

BIG WARNING!!!!

- Lab machines are currently running Aibo-R plus Tekkotsu 4.0
- By mid-semester we will have Chiara robots, and will be upgrading 2 of our 3 machines to Tekkotsu 4.1 (1 machine will stay 4.0 for people wanting to use Aibos for final projects)
- SSH support will be available locally (connect to local wireless router...)
- All of this is SUBJECT TO CHANGE!!1!ONE!

UNIVERSITY AT ALBANY
State University of New York

Installing Tekkotsu

- Choose an operating system
  - Using Linux is highly preferred (Ubuntu 8.04)
    - Remember sudo apt-get install flex texinfo
  - Windows requires Cygwin
  - Chiara platform requires Ubuntu 8.04/Fedora 9
- For AIBO: install Open-R
  - Available from Tekkotsu download page
- Install newest Sun Java JDK from:
  - http://java.sun.com/
- And install V4.1 from CVS (or 4.0 for Aibo...)
  - See tekkotsu.org for details...

Creating a Project

- This is your development workspace.
- Needs to be copied locally for each developer or group.
  - cp –R /usr/local/Tekkotsu/project ~/yournamehere
- All development source should be contained here
- Allows customization of certain settings
  - ~/yournamehere/ms/config contains
    - .tm (threshold map) and .col (color) files for color segmentation
    - tekkotsu.xml project configuration file that allows the setting of lighting conditions, segmentation files, camera settings, and various monitoring ports to be used when *your project* is compiled and run.
  - Take a look at the series of StartupBehavior_Setup….
    - Starting from StartupBehavior.cc , this is the boot sequence for the framework
    - These are a bunch of special behaviors that run on startup to handle things like initializing the vision, menus, file control, etc.

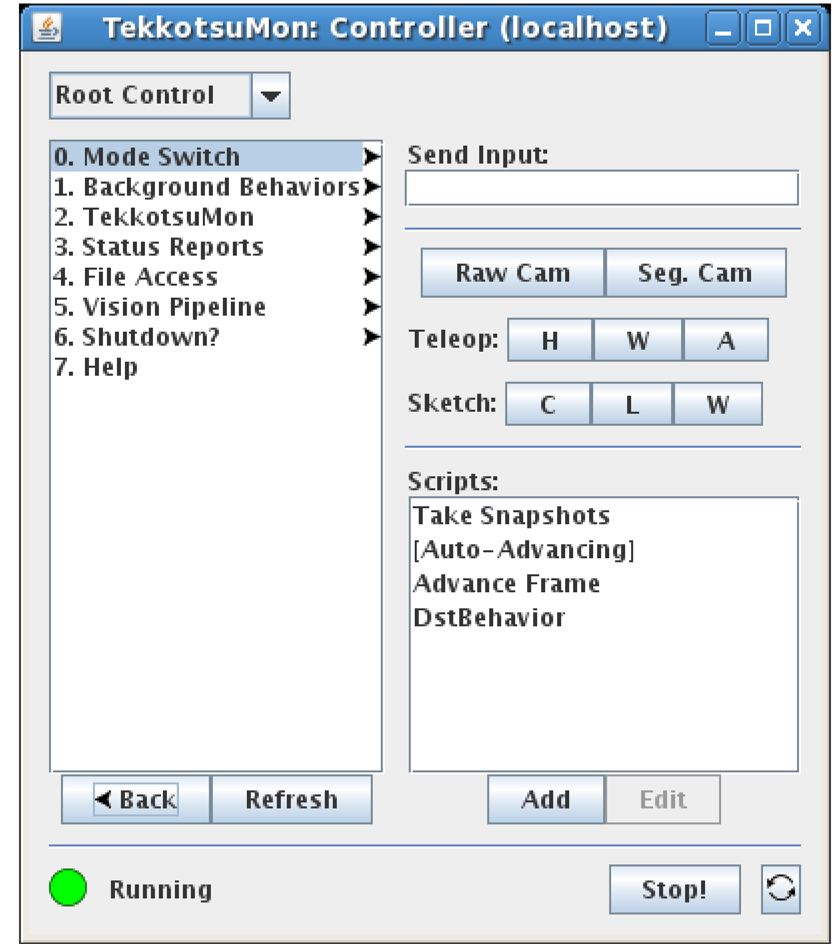
Behaviors (new style)

- Behaviors are classes defined in .h files:
- Add them to the ControllergGUI "Mode Switch" menu by calling MENUITEM in

  ~/project/UserBehaviors.h
- Double click on the "Mode Switch" menu item to instantiate Instantiated by BehaviorSwitchControl() when requested by the ControllerGUI
- When you stop a behavior (double click on the menu item), the instance is deleted

***NOTE**: To see stdout, use*

`telnet IP 59000`



Behaviors (old style)

- Add behaviors to your project
  - StartupBehavior_SetupModeSwitch.cc
    - addItem(new BehaviorSwitchControl<TestBehavior>("Test Behavior",bg,false));
  - BehaviorSwitchControl is C++ template:
    - Requires AddReference, RemoveReference, DoStart, DoStop
  - Three arguments are:
    - String of name for behavior
    - Reference to behavior group (bg for background)
    - Boolean on whether to retain behavior image in memory after DoStop is called

UNIVERSITY AT ALBANY
State University of New York

Behaviors Cont'd

One way to define a behavior would be to say that a behavior is how a robot creates action from perception as shown in the diagram below:

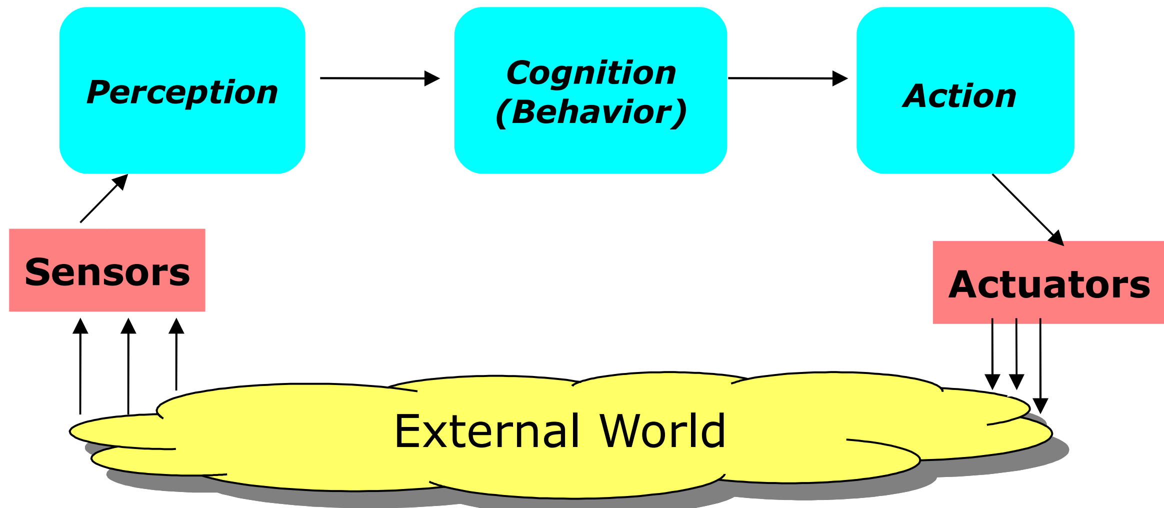


Figure 1

And now, a Hello World example:

UNIVERSITY AT ALBANY
State University of New York

```
# include “Behaviors/BehaviorBase.h”

class HelloWorld : public BehaviorBase{

        // Constructor
        HelloWorld() : Behavior(“HelloWorld”) {}

        // Called when behavior is activated
        virtual void DoStart(){
                cout << “Hello World” << endl;
        }

        // Called when behavior is stopped
        virtual void DoStop(){
                cout << “Goodbye World” << endl;
        }

        // Processes events
        virual void processEvent(const EventBase &event){
                // do stuff!
        }
}
```

***NOTE**: Use template in project/templates/behavior.h*

Makefile

- Now that we've seen the full dev process in action, let's examine the Makefile to see how everything gets put together
- Compiled code needs to be loaded onto a Sony MemoryStick
  - Always mount /mnt/memstick (or /cygdrive/*memstickdriveletter*)
- Be sure you have your memstick path correct!
  - Environment.conf
- Make options
  - make newstick
    - Initializes stick for use (use the 1st time using a new memory stick)
    - Make sure /open-r/system/conf/* have write permissions enabled
    - Make sure /open-r/mw/conf/* have write permissions enabled
  - make install
    - Compile & link with full upload to memstick
  - make update
    - Compile & link. Uses rsync to determine what to upload to memstick
  - make
    - Compile & Link
- Never ever, Ever, EVER format the memory sticks!!!
  - Pink AIBO Programming Memory Sticks won't work after a format!!!
  - Windows "Quick Format" is ok, but NO low-level formats!!!

Perception

- Sensors convert the world into electric signals our computer can understand.
- We then throw events when our sensors detect things of interest. We can subscribe to these events:

```
erouter->addListener(this, generatorEGID);
```

- And, we can access values
  - Through the event (typically for vision)
  - Or from the "World State"
- We will return to see sensors in more detail...

UNIVERSITY AT ALBANY
State University of New York

Events

- Events are subclasses of *EventBase*
- Three essential components:
  - Generator ID: what kind of event is this?
    - buttonEGID, visionEGID, timerEGID,...
  - Source ID: which sensor/actuator
    - ChiaraInfo::GreenButOffset
    - ERS7Info:HeadButOffset
  - Type ID, which must be one of
    - activateETID, statusETID, deactivateETID

UNIVERSITY AT ALBANY
State University of New York

Subscribing to Events

addListener(*listener, generator, source, type*)

```
#include “EventRouter.h”
virtual void DoStart() {
     erouter->addListener(this,
                           EventBase::buttonEGID,
                           RobotInfo::GreenButOffset,
                           EventBase::activateETID);
}
```

***NOTE**: erouter – is a global pointer to the event router*

Processing Events

```
virtual void processEvent(const EventBase &event){
  switch ( event.getGeneratorID() ) {
    case EventBase::buttonEGID:
      cout << "Button press: " <<
              event->getDescription()<< endl;
      break;
    default:
      cout << "Unexpected event: "
           << event->getDescription() << endl;
  }
}
```

Or, use the World State

- Can access sensor readings, current joints locations, etc.
- IMPORTANT: updated about 30 frames per second. BUT: if you are in processing an event, the values will not change during the call to processEvent().
- sensorEGID events occur on each update
- `state` is a global pointer to the world state

WorldState members

- float outputs[NumOutputs]
  - last sensed positions of joints, for ears (or other future "boolean joints"), x<.5 is up, x>=.5 is down
- float buttons[NumButtons]
  - magnitude is pressure for some, 0/1 for others; indexes are defined in the ButtonOffset_t of the target model's namespace
- float sensors[NumSensors]
  - IR, Accel, Thermo, Power stuff; indexes are defined in SensorOffset_t of the target model's namespace
- float vel_x, vel_y, vel_a

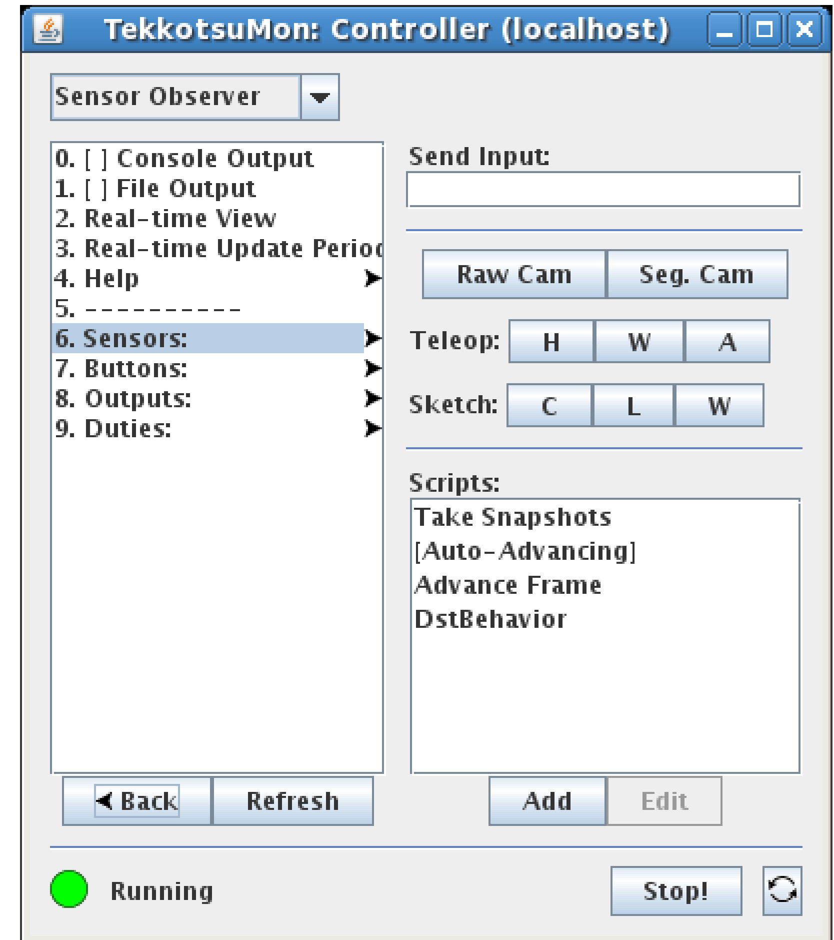
World State Examples

- To access correct values from WorldState's arrays, we need to look in the respective ModelInfo namespace to find which constant has been enumerated for our needs.
- To get current IR distance reading:
  - state->sensors[NearIRDistOffset];
- Retrieving info about a paw button
  - state->buttons[LFrPawOffset];
- Joint values
  - state->outputs[LFrLegOffset+0]
    - Recall that there are 3DOF in each leg

UNIVERSITY AT ALBANY
State University of New York

Sensor Observer

- A useful tool to monitor sensors and see what values joints take on.
- Root Control > Status Reports >Sensor Observer



UNIVERSITY AT ALBANY
State University of New York

Sensors

- Sensor Types
  - Passive sensors – capture the environment as is:
    - Vision camera, temperature sensor
  - Active sensors – emit energy and capture results
    - Sonar, IR
- Sensor noise
  - The returned value from a sensor is cluttered with data completely unrepresentative of the real world object it's perceiving
  - Some real world object is interfering with the expected value of a given sensor.

UNIVERSITY AT ALBANY
State University of New York

Handling Noise

- Either active or passive sensor types can wind up being noisy in a real application.
  - Tekkotsu handles frame-rate level noise
  - But we still may need some tools if the data just plain seems inconsistent
- We can handle noise with any of the following:
  - Thresholding
  - Hysteresis
  - Averaging
  - Learning (statistical/bayesian methods)
-

UNIVERSITY AT ALBANY
State University of New York

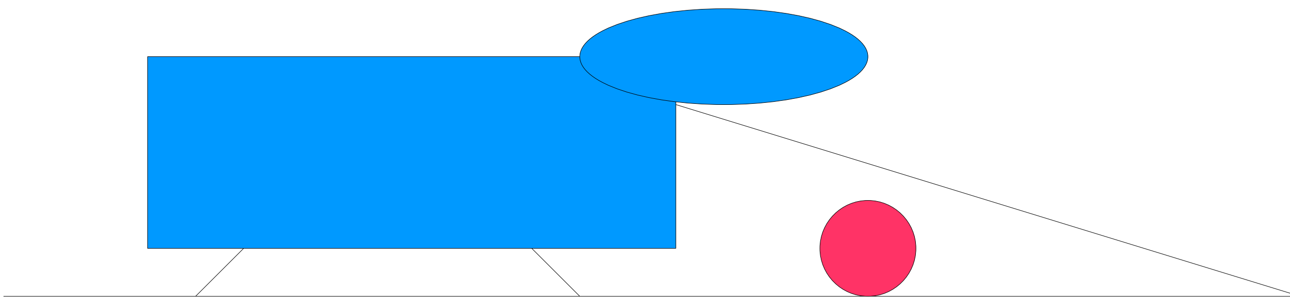
Handling Noise

- Single-value Thresholding
  - Using a single threshold returns two values and leads to transition between two states:
    - ex. if (noisy sensor < 350) -> STOP
  - Causes oscillation!
- Hysteresis
  - Still using a threshold
  - Allows for some overlap in the values between states – leads to latency
- Exponential averaging
  - Combines multiple sensor readings, use average

UNIVERSITY AT ALBANY
State University of New York

Choosing a Sensor

- Be aware of type of perception required
  - Active vs. Passive
  - IR vs Camera
  - Possible troubles:
    - Environmental Consistency?
    - Sensor location?
    - Range?



UNIVERSITY AT ALBANY
State University of New York

Summary

- Robotics = Computers + Uncertainty
- Test, test, and then test some more...