

# Concurrent and Real-Time Task Management for Self-Reconfigurable Robots



Harris Chiu & Wei-Min Shen

Polymorphic Robotics Lab, USC-ISI

<http://www.isi.edu/robots>

Sponsors: NASA and ARO

# Outline

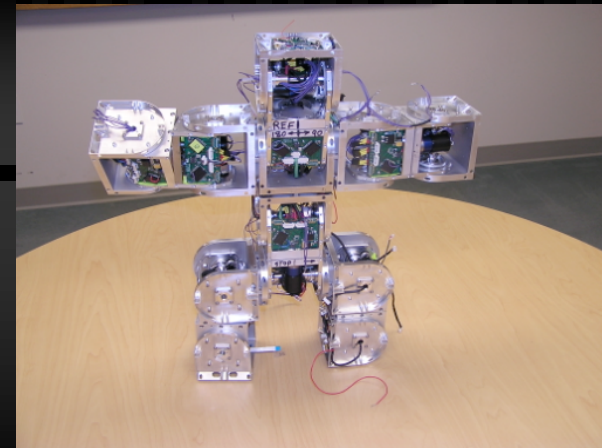


- ✓ Task management in Self-Reconfigurable Robot (SRR)
- ✓ Related work (a quick survey)
- ✓ Our approach
  - ✓ Real-Time task management based on a Real-Time OS
- ✓ Current Status and Live Demo
- ✓ Conclusions and future work

# Task Management in SRR

- ✓ Tasks running on Self-Reconfigurable Robots

- ✓ Sensors
- ✓ Actuators
- ✓ Communications
- ✓ Behaviors



- ✓ Limited number of micro controllers

# Time Constraints of tasks in SRR

- ✓ Along *Temporal* dimension
  - ✓ Tasks running on same module must
    - ✓ finish before its deadline
    - ✓ run concurrently
    - ✓ be scheduled in real-time
    - ✓ synchronize with each other

# Difficulties of Programming

- ✓ Interleaving tasks is difficult or impossible

- ✓ Task A: 

- ✓ Task B: 

- ✓ A sequential program is hard to write and debug, especially when x and y have no common denominators

- ✓ Many possible combinations of tasks

- ✓ {A,B}, {A,C}, {B,C}, .....

- ✓ Impossible if knowledge must be gained at run time

- ✓ The starting time of a task

- ✓ The duration of a task

# Difficulties of Programming (2)

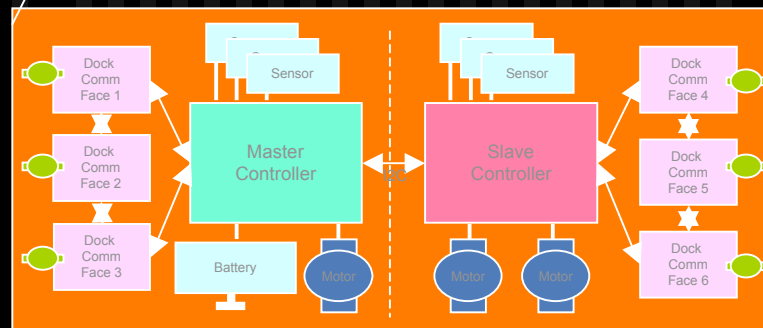
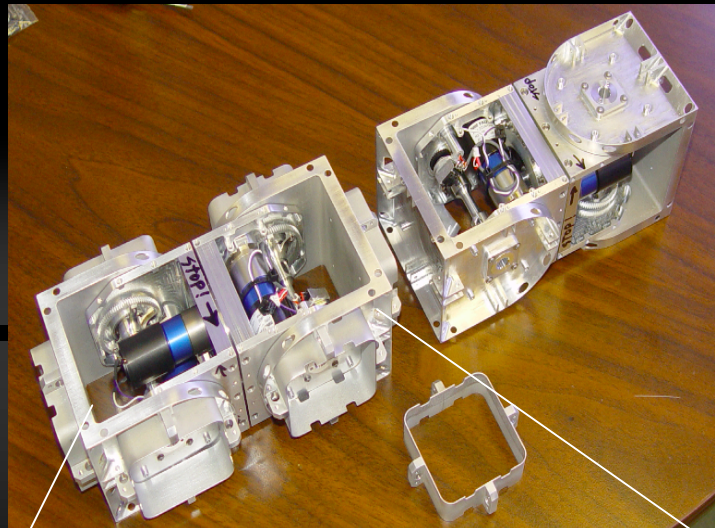


- ✓ Difficult to manage the timing if more tasks are added
- ✓ Knowledge of timing of accessing underlying hardware is required

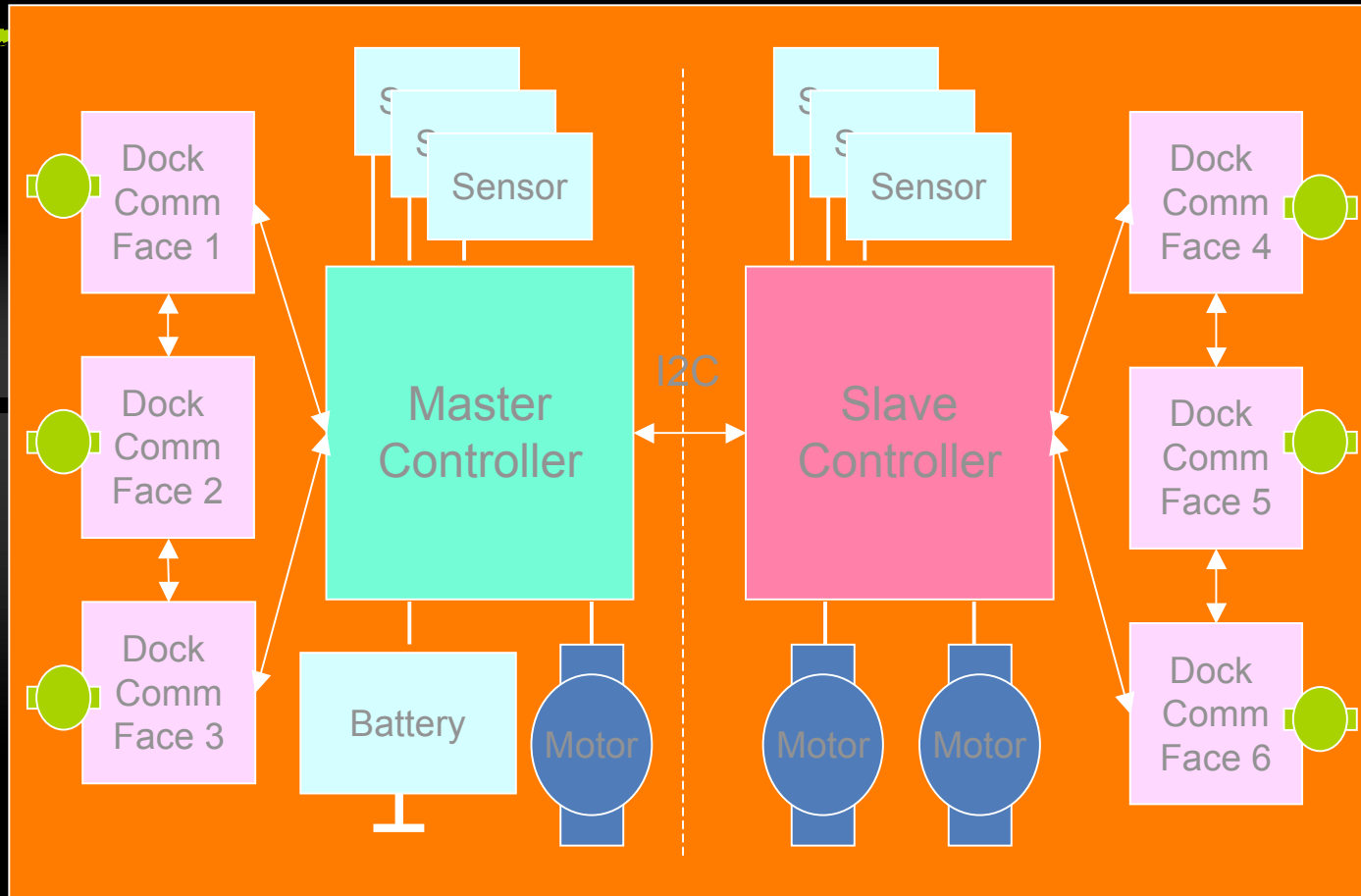
# Space Constraints in SRR

- ✓ A SRR also has constraints along the *Spatial* dimension
  - ✓ Actions must be selected based on how modules are connected
    - ✓ Starting time of actuators varies
  - ✓ Control must be *totally distributed* and *scalable*
    - ✓ No global identifiers for modules !
    - ✓ Dynamic topology change affects the starting time and the duration of tasks
- ✓ A controller of a SRR must satisfy both!

# SuperBot: Configurations and Modules



# Devices in a SuperBot Module



# Real-Time Tasks in SuperBot

- ✓ Communication tasks (9)
  - ✓ 6 inter-module (Infrared) and 1 intra-module (I2C)
- ✓ Actuator tasks (9)
  - ✓ 3 DOF plus 6 tiny motors for dock connectors
- ✓ Sensor tasks (19)
  - ✓ 9 for position sensing, 6 for IR proximity, 1 for voltage, 1 for current, 1 for RF, 1 for accelerometer
- ✓ Behavior tasks (1 or more)
- ✓ Total tasks (36 or more)

# Related Works



- ✓ Embedded Real-Time OS
  - ✓ QNX, LynxOS/BlueCat, TinyOS, XMK, NutOS, FreeRTOS, AvrX (Barello 2002)
- ✓ Applications to SRR
  - ✓ Massively distributed control net (MDCN) based on CANbus [Zhang, et al. 2002]
  - ✓ CANbus -- Limited address space, need module IDs

# Our Approach

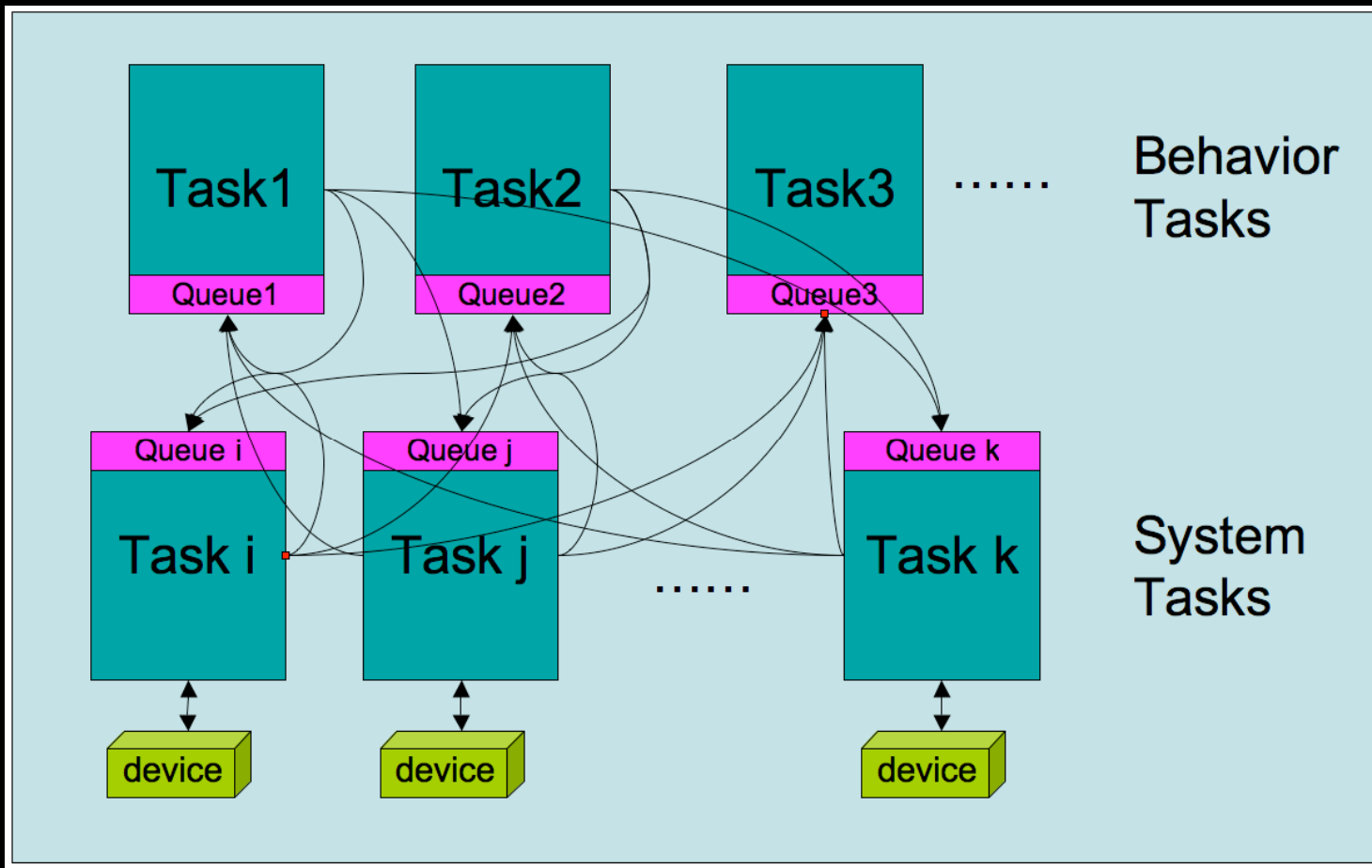


- ✓ ID-Free Concurrent Controller (IFCON)
  - ✓ Based on AvrX kernel
    - ✓ Tasks, Semaphores, Timers, Queues, Stepper, FIFO synchronization, small size (1.5kb)
  - ✓ Hierarchical task structures
    - ✓ System level (stable encapsulation of hardware)
    - ✓ Behavior level (applications via APIs)

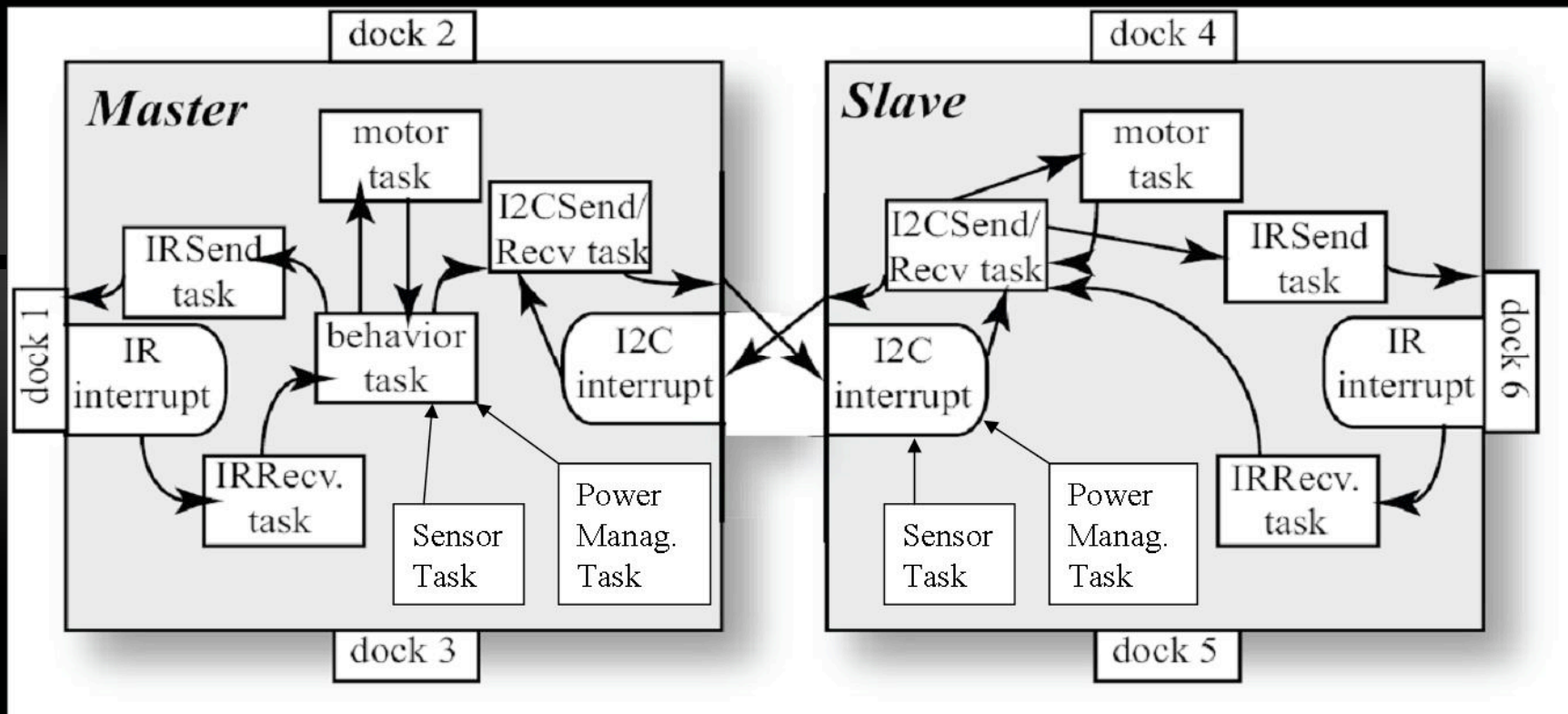
# Real-Time Task Management

- ✓ Hides time-critical issues from the users (behavior programmers)
  - ✓ Easy addition/deletion of time-critical behaviors
  - ✓ Adaptive to hardware changes
    - ✓ Sensors, actuators, microcontrollers, devices, ...
- ✓ Straightforward software development
  - ✓ Intuitive to organize into tasks for users
- ✓ Increased robustness
  - ✓ Modularized software components
  - ✓ Sensing, actuation and communication
- ✓ Easy to debug and maintain

# The IFCON Architecture



# System Tasks in a Module



# API for Behaviors (1)

## ✓ API for actuators

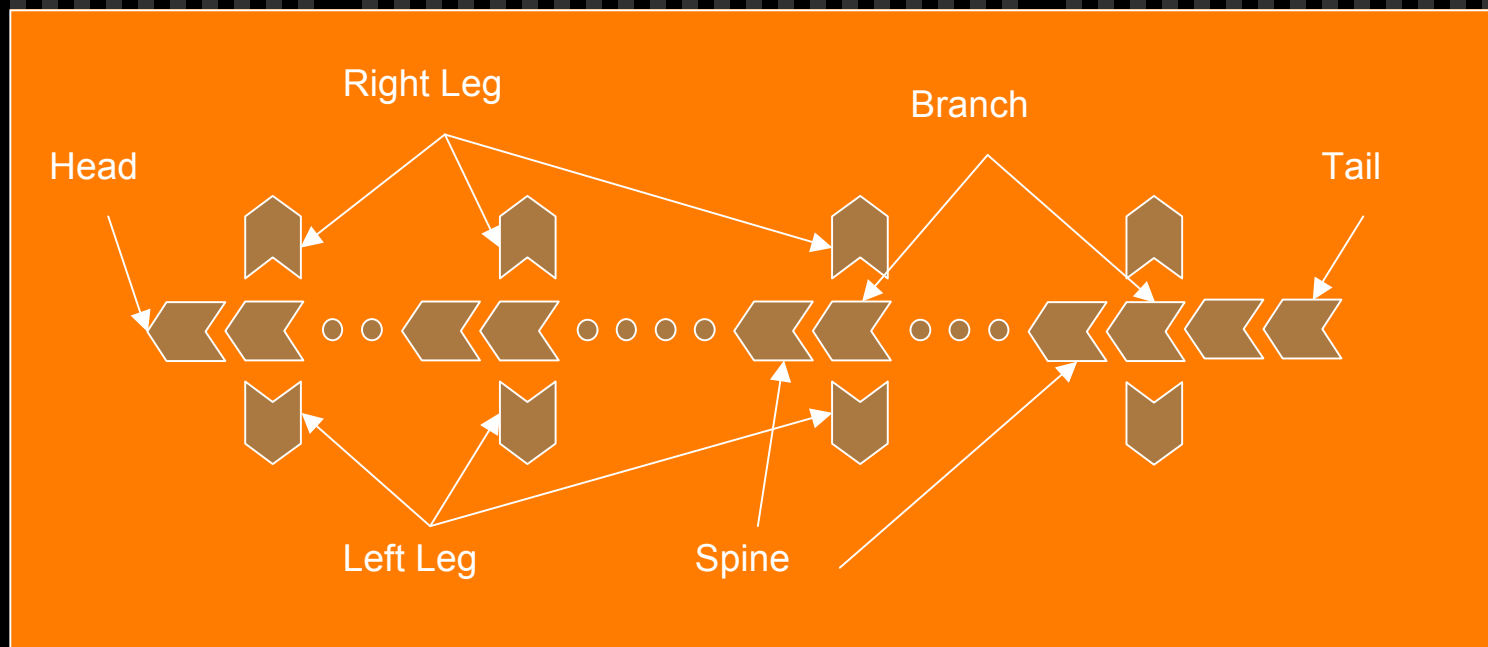
- ✓ enableMotor(motorID) //enable motor to be controlled
- ✓ disableMotor(motorID) //disable motor to be controlled
- ✓ setPID(motorID, PIDValue) //set PID gains for motor control
- ✓ getPIDValue(motorID) //get the PID gains from a motor
- ✓ getMotorStatus(motorID) //get the status of the motor
- ✓ getRadioStatus(pin) // get RF communication status
- ✓ getMotorPosition(motorid) //get the current Motor angle
- ✓ setMotorPosition(motorid, motorAngle) //set the motor angle
- ✓ enableLED(id) // switch LED on
- ✓ disableLED(id) // switch LED off

# API for Behaviors (2)

- ✓ API for communication with other modules
  - ✓ sendIRMessage(dockID, message) // send through a dock
  - ✓ enableIRReceiver(dockID, receiverID) // enable a receiver
  - ✓ disableIRReceiver(dockID, receiverID) // disable a receiver
- ✓ API for sensors
  - ✓ senseAcceleration() // a read from accelerometer
  - ✓ senseVoltage() //for getting voltage values
  - ✓ senseCurrent() //for getting current values
  - ✓ senseProximity(dockID) //get proximity sensor value
- ✓ API for communication with other behaviours in the module
  - ✓ sendTaskMessage(BehaviorTaskID, message) // send message
  - ✓ getTaskMessage(BehaviorID, message) // receive message

# Behavior Task (Example)

- ✓ To control a centipede using Behavior Tasks
  - ✓ The challenge: the # of legs and their positions may not be known in advance and can change dynamically



# Behavior Tasks for Centipedes

## *The AC\_SEND Task:*

Repeatedly probe neighbours to update the local topology.

## *The AC\_RECV Task:*

Receive probes and update the local topology.

## *The HORMONE Task:*

For each received hormone message:

Select and execute the proper local actions based on

- (a) the local topology,
- (b) the local sensor inputs,
- (c) the local state/timer information,
- (d) the received hormone message;

Propagate the hormone message to other connectors.

TABLE 1: THE ACTION RULES FOR A CENTIPEDE.

Module Type	Local Timer	Received Hormone Data	Perform Action	Send Hormone
Head	0		Straight	[CP, A, {l,r,b}]
Head	0.5*Period		Straight	[CP, B, {l,r,b}]
Spine		A	Straight	[CP, B, {b}]
Spine		B	Straight	[CP, B, {b}]
Branch		A	Straight	[CP, B, {l,r,b}]
Branch		B	Straight	[CP, A, {l,r,b}]
Left Leg		A	Swing	
Right Leg		A	Holding	
Left Leg		B	Holding	
Right Leg		B	Swing	

# Current Status



- ✓ System tasks
  - ✓ Implemented and tested
- ✓ API for behavior tasks
  - ✓ Mostly implemented and tested
- ✓ Behavior tasks
  - ✓ Prototype demonstrations
- ✓ Live demonstrations for individual modules

# Conclusions and Future work

- ✓ Concurrent and real-time task management satisfies both *Temporal* and *Spatial* dimension constraints for self-reconfigurable robots.
- ✓ Implementation and demonstration
  - ✓ IFCON for Id-free real-time distributed control
  - ✓ System tasks and their corresponding API
  - ✓ Behavior tasks
- ✓ Future work
  - ✓ Complex behaviors for different configurations on top of the architecture
  - ✓ Self-reconfigurations between configurations