MULTI-AGENT MANIPULATOR CONTROL AND MOVING OBSTACLE AVOIDANCE

George Birbilis
Nikos Aspragathos
Mechanical Engineering & Aeronautics Department,
University of Patras

Introduction – the problem

- planning the motion of a *serial manipulator*
- workplace may contain unknown stationary or moving obstacles *(time varying environment)*
- real-time reactive/adaptive behavior
- focus on highly redundant planar manipulators

Introduction - Current approaches *(1/2)*

- many approaches in Hwang & Ahuja, 1992
- unknown moving obstacles case favors local planning (global re-planning expensive if many obstacles or highly redundant manipulator), Chen & Hwang, 1992, Challou et al., 1998
- top-down centralized approach has increasing complexity and cost when number of joints and links rises
- bottom-up, modular approach favored, modeled as a *multi-agent system*
- Motion planning on each subpart, combining local solutions into solution for manipulator path planning problem
- In most cases, composition implemented in real-time, via interaction (cooperation or contention) of agents controlling manipulator parts

Introduction - Current approaches *(2/2)*

- Overgaard et al., 1996:

  multi-agent system, both joint and link agents, control 25-DOF snakelike robot in environment with obstacles modeled using artificial potential field (Khatib, 1986)
- Bohner and Lüppen, 1997:

  7-DOF robot, only joints as agents, doing sensor-data integration and motion planning on a per-agent basis
- Zavlangas and Tzafestas, 2000:

  fuzzy logic to escape local minima
- Althoefer et al., 2001:

  neuro-fuzzy system

Introduction - a new approach

Proposed system:

- multi-agent system
- each software agent controls a specific part of the joint-link chain
- agents interact with each other to adapt the manipulator configuration to external events and changing situations in real-time

Reduces motion planning of a manipulator to motion planning of a single part of it (e.g. tool tip):

other parts react and adapt or object (veto) to the moving part's motion

The kinematic model

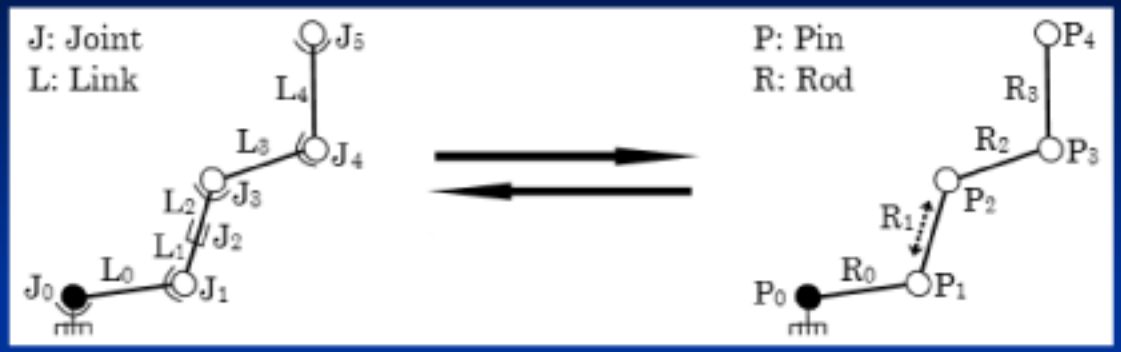
- Planar manipulator
- Can have both rotational and translational joints, any serial combination

- Manipulator base may be fixed or may move freely or on predefined path
- Base may be able to rotate around an axis perpendicular to manipulator plane

- On 2D plane that may contain obstacles
- Obstacles stationary or moving with speed comparable to joint motors speed
- Case of moving obstacles includes the case of other manipulators or humans also working in the same environment, with no communication channel or common protocol for coordination

- No a-priori knowledge of obstacles assumed
- Incoming collisions detected either using touch sensors covering the manipulator body (Bohner and Lüppen, 1997), or range sensors detecting the closest obstacle at each side of the links (Zavlangas and Tzafestas, 2000)

The conceptual model



- potentially expandable ***rods*** (of length Li in [Li_{min}, Li_{max}])
- interconnected at their endpoints using ***pins*** (rotation angle Ai in [Ai_{min},Ai_{max}])
- forming a chain

- behaves like a rope (given a high number of rods & pins)

Mapping kinematic to conceptual constraints

Kinematic model constraints mapped to conceptual model ones:

- Rotational joint : [$Pin_i Angle_{min}$, $Pin_i Angle_{max}$]
- Translational : [$Rod_i Length_{min}$, $Rod_i Length_{max}$]
- Non-moveable manipulator [$Pin_i Position$=fixed]
- Irregular geometry: e.g. simulate a 90-degrees angle shaped link using two links connected by a constrained (fixed angle) rotational joint

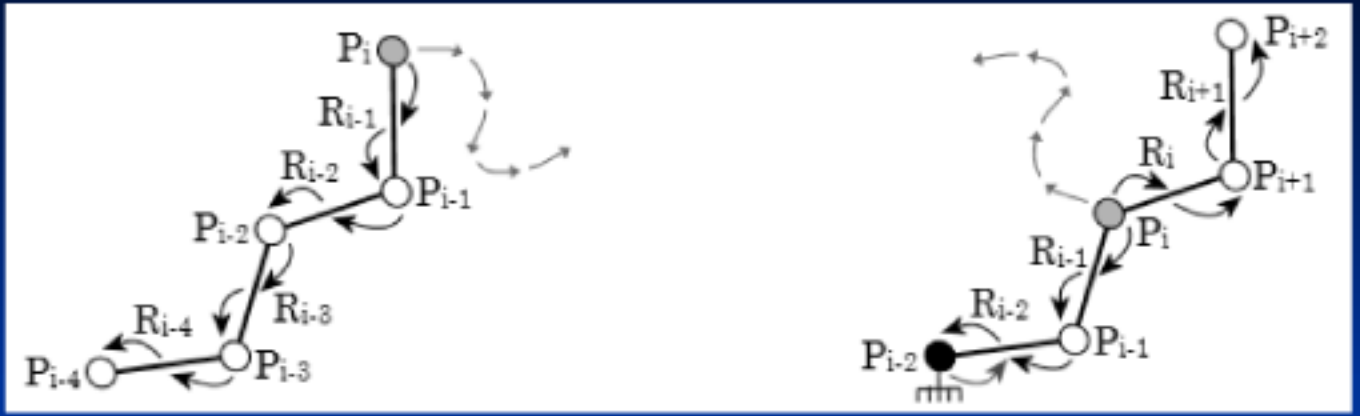
Master - Slave relationship

Master-Slave relationship:

- two entities, "master" and "slave"
- master emits change events for its state
- slave listens for changes to master's state
- slave reacts to master's state change by changing its own state

Listen & react to change also called *observer pattern*

Change event propagation



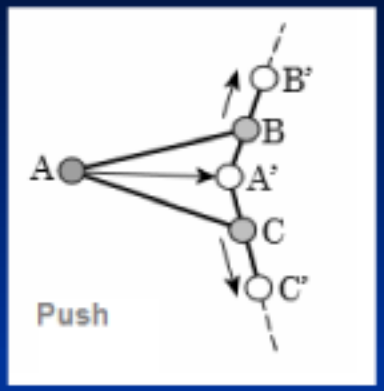
Any chain part (not just the tool tip) can initiate motion, receiving its own motion commands from planner module or human operator (via higher application layer), so chain is split by the mover part into two sub-chains:

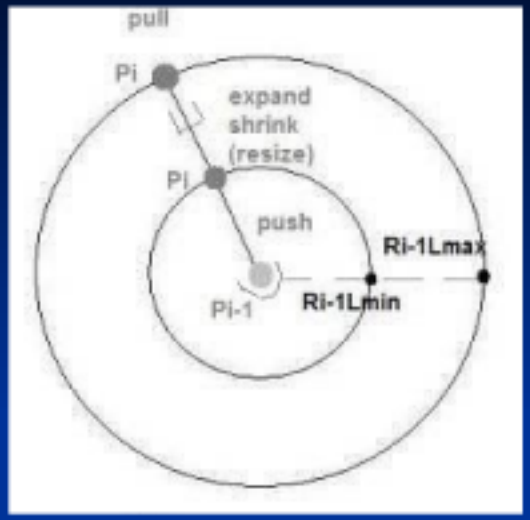
$\mathbf{P}_i$ master of slave P_{i-1}	$\mathbf{P}_i$ master of slaves P_{i-1} & P_{i+1}
P_{i-1} master of slave P_{i-2} …	P_{i+1} master of slave P_{i+2} …

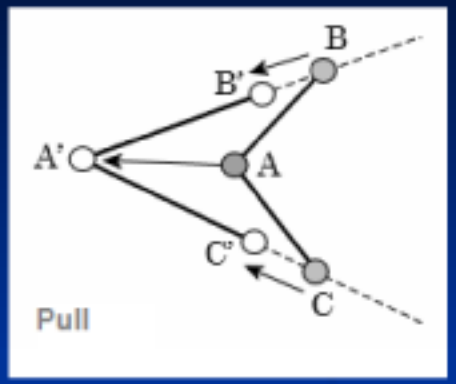
a slave can have its own slaves (a master-slave chain/hierarchy, nested relationships)

- *state change events* at the head of each Master-Slave sub-chain propagate towards the tail
- each slave tries to react and adapt to its master's state change event

React to master's state change events







Slave's predefined *rule set* = rules reacting to changes of master entity's state

Reactive rules try to preserve *constraints* imposed on slave and on the relation to its master:
- by design (internal, hardware constraints)
- decided on the fly at runtime (environmental constraints, obstacles or failure of subparts)

Two basic model constraint preservation behaviors (master Pi, slave Pi-1):
- *Push :* distance (Pi's new-pos, Pi-1's current-pos) < Ri-1Lmin (min length of rod Ri-1)
- *Pull :* distance (Pi's new-pos, Pi-1's current-pos) < Ri-1Lmax (max length of rod Ri-1)

Slave pin tries to move on the (Pi's new-pos, Pi-1's current-pos) guiding line (rod)

React to sensed obstacles

- Master pin motion:

  Reaction of slave pin (move) & slave rod (resize) to motion of master pin

- Push/Pull model constraint preservation behaviors (if slave rod can't resize enough):

  new slave pin position

- Obstacle – Rod collision may block new slave pin placement:

  Master-Slave's Rod rotates around master pin, at "contact" point/side with obstacle (new guiding-line for slave motion)

Environmental constraint preservation behaviors named *Push-Rotate* and *Pull-Rotate*

Master - Vetoable slave relationship

Master – Vetoable slave relationship:

- two entities, "master" and "slave"
- master emits change events for its state
- slave listens for changes to master's state
- slave reacts to master's state change by changing its own state
- Difference from classic Master-Slave relationship is that the slave can object to the state changing of the master (*veto*)

Listen & react+veto usually called a "*Vetoable Change Listener*" in Object Oriented Programming (OOP)

Veto back-propagation

Using Master - Vetoable slave relationship to support:

- fixed base manipulators
- replanning in case some slave part of the chain can't adapt to its master's motion because it's trapped in some obstacles or malfunctioning

A slave part of the chain forced to react and adapt to an initial motion of a master neighboring part of the chain, can object (veto) to the motion of that master part …

… if itself cannot move to adapt to the master's motion while still preserving the chain model and environmental constraints

e.g. the pin agent of a manipulator's fixed base always objects to its relocation

The software model *(1/2)*

Agent = an entity that is: *(Liu et al., 2002)*

- able to live and act in an environment
  able to sense its local environment
- driven by certain objectives
- has some reactive behaviors

Multi-agent system = a system that has:

- a set of agents
- an environment =
  - a space where the agents live
  - set of reactive rules, governing the interaction between agents and their environment (laws of agent universe)

The software model *(2/2)*

- Agent & Multi-agent system definitions:
  - map directly to the pin & rod entities defined in the conceptual model
  - straightforward implementation
- Event & veto propagation can be implemented:
  - as remote events and exceptions over a data channel at embedded platforms like Java or .NET
  - as hardware interrupts at lower level implementations

Current development & plans

- Conceptual model simulated in 3D space:
  - 3impact 3D graphics engine (www.3impact.com)
  - Multi-agent logic in Microsoft Visual C++ .NET
- Future plans:
  - Use 3impact's included Open Dynamics Engine (www.ode.org) to simulate physical model too
  - Control robot simulators and physical robots via IPC and I/O channels (RoboTalk, Serial I/O etc.)

Conclusions *(1/2)*

- Most important advantage:
  - Reduces external path-planning only to the motion of the master (usually tool-tip)
  - Other manipulator parts adjust in real-time to the master's motion, reacting to sensed obstacles
- Best applied to highly-redundant manipulators:
  - Scales up efficiently, agents interact with their immediate neighboring agents only
- Supports fixed and moving base manipulators

Conclusions *(2/2)*

If veto from slave agent back-propagates as far as the original mover agent (the rest of the manipulator chain can't adapt to its movement):

- Autonomous mover: replan its motion
- Externally controlled mover: propagate veto to planner module or human operator

Simulation

Conceptual model 3D simulator…