# Reactive multi-agent based control of mobile robot chain formations

**George I. Birbilis*. Nikos A. Aspragathos***

**University of Patras, Mechanical Engineering & Aeronautics Dept., Greece*
*(Tel: +30-2610-997268; e-mail: birbilis@kagi.com, asprag@mech.upatras.gr)*

**Abstract:** A Multi-agent based coordination scheme for chain formations of mobile robots, employing geometric constraint satisfaction rules, is presented. Applying this architecture on mobile robot swarms, snake-like real-time motion and reactive obstacle avoidance is achieved. A "Master – Slave" relationship is used, where a controller agent's motion is propagated to its two neighbouring ones in the chain formation and progressively further on towards the two chain endpoints. At each propagation step, a constraint preservation mechanism enforces the respect of minimum and maximum (minmax) distance constraint between agent pairs, and of minmax angular constraint among triplets of consecutive chain agents. The emergent behaviour of this multi-agent system is as if a moving part of the chain pushes or pulls the two subparts it divides the chain into. To allow replaning in case a slave part can't adapt to its master's motion, being trapped in obstacles or malfunctioning, the notion of a "Vetoable slave" behaviour is used, where the slave part can object (veto) to the motion of its master part.

*Keywords:* Chain formation, Snake-like motion, Multi-Agent, Mobile Robots, Swarm, Reactive control, Master-Slave relationship, Vetoable Slave behaviour.

## 1. INTRODUCTION

In this paper, a multi-agent based coordination scheme, employing geometric constraint satisfaction rules, is applied to the problem of real-time kinematic planning and obstacle avoidance for a chain formation of multiple mobile robots (a swarm according to Bonabeau (1999)). The environment of the swarm potentially contains unknown stationary or moving obstacles, while each robot is equipped with sensors detecting its proximity to obstacles, to implement obstacle avoidance behaviour.

Various approaches to motion planning in time varying environments exist, most of which are being thoroughly examined in Hwang and Ahuja (1992). As has been pointed out by Chen and Hwang (1992), and Challou et al. (1998), the existence of unknown moving obstacles in the environment is in great favour of local planning methods, with global re-planning being very expensive, especially in the case of many obstacles, or when having to plan for highly redundant systems (with excess degrees of freedom – DOFs, in our case a multitude of mobile robots).

A modular bottom-up approach, modelled as a *multi-agent system* (see Liu (2001)) is suggested, employing local interactions that surface emergent system behaviour, able to scale up to swarms comprised of high numbers of robots, while still supporting those with less mobile robots. Using a centralized top-down approach instead would present increasing complexity and cost as the number of robots rises, due to the high dimensionality of the problem search space that occurs at that case.

Dorigo et al. (2004) presented a system comprised of several mobile robots, called "swarm-bot", and evolved self-organizing behaviours for it. While experimenting with eight robots connected by flexible links to create a snake formation, they observed that they were able to negotiate a unique direction to produce coordinated movement along it and collectively avoid walls. Given the use of flexible links, the swarm-bot tends to change its shape during coordination phases and during collision with obstacles. Also, with swarm members tending to maintain their direction of movement, the system displays the ability to travel through narrow passages, eventually deforming its shape according to the configuration of the obstacles.

In Birbilis and Aspragathos (2006), a multi-agent system is proposed where each software agent controls a specific part of a robotic manipulator's joint-link chain or a mobile robot swarm's chain formation, and the agents interact towards the adaptation of the chain's configuration to external events and varying situations in real-time. A geometric constraint satisfaction approach is employed, that reduces the kinematic planning of the system to planning only for one of its parts, with the other chain parts reacting and adapting or vetoing to the motion of that part. The proposed architecture has been simulated in 2D and 3D space, for both physical chain linkages (manipulators) and virtual ones (mobile robot swarm formations).

In this paper focus is placed upon the case of coordinating vast numbers of multiple mobile robots (swarm) that follow a leader robot using the aforementioned multi-agent system, while the reactive system behaviours are presented in an object-oriented algorithmic form (see Appendix A).

## 2. PROBLEM DEFINITION

We're considering the case of a serial chain structure where each two consecutive chain elements must maintain a given minimum and maximum distance constraint between them, and every three consecutive elements must maintain a minimum and maximum angle constraint.

The chain can represent a formation maintained by a swarm of mobile robots, with each two consecutive robots in the chain maintaining a directional communication link (e.g. short range optical or microwave), or using vision in order for a follower robot to be able to detect the motion direction of its leading robot relative to itself.

The environment may contain obstacles, either stationary or moving, however with a speed comparable to the mobile robots' speed. The case of moving obstacles also covers the case of other entities (mobile robots, manipulators, humans) acting in the same environment, when there is no communication channel or any common protocol for coordinating them too. No a-priori knowledge of obstacles is assumed, while incoming collisions can be detected at each part of the chain using either touch sensors or range sensors.

## 3. MULTI-AGENT APPROACH

Liu et al. (2002) define an agent as an entity that is able to live and act in an environment, is able to sense its local environment, is driven by certain objectives and has some reactive behaviour. A multi-agent system is defined as a system that has an environment, that is a space where the agents live, a set of reactive rules, governing the interaction between the agents and their environment (the laws of the agent universe) and of course a set of agents.

In the proposed multi-agent system model, the serial chain's motion is controlled in a way that mimics (Fig. 1) the motion of a chain of *rods*, interconnected at their endpoints with *connectors* (potentially constrained spherical joints). The chain's two endpoints are connectors too, while rods can be resizable, having a minimum and maximum length instead of a fixed one. Any part of the chain that initiates a motion "pushes" or "pulls" the other parts of the chain so that the constraints defining the chain structure are always kept. Given a high number of rods and respective connectors, this model behaves like a snake crawling amidst obstacles.

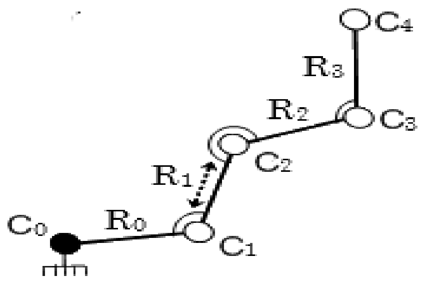

Fig. 1. A serial linkage (chain) defined using Rods (R) and Connectors (C), where mobility, distance and angular constraints among Connectors may apply.

*Constraint modelling*

Rods represent distance constraints between pairs of consecutive robots, while Connectors represent angular constraints between triplets of consecutive robots in the chain formation. Both are modelled by respective agents, which can be embedded in the robots (in situ), or interact inside a separate system (be it single computer or computational grid).

Each *Connector agent* has a *Position* property and each *Rod agent* has *MinLength* and *MaxLength* properties, such that the length bounds of a rod are respected by the positions of the connector agents at the rod's two endpoints. Connector agents also have *MinAngle* and *MaxAngle* properties, constraining the angle defined by the triplet comprised of the connector and its previous and next connectors on the chain.

The current *Length* property of a rod is calculated at any time from the positions of its two endpoint connectors, while the current *angle* property of a connector agent is easily calculated geometrically from its position and those of its two immediate neighbour connectors. Calculated properties are shown in square brackets at Table 1.

**Table 1. Rod and Connector agents and their properties**

| *Entity* | *Properties* |
|---|---|
| Rod | MinLength, MaxLength, [Length] |
| Connector | Position, MinAngle, MaxAngle, [Angle] |

*Agent interactions*

A *Master – Vetoable slave relationship* is defined between two entities, "master" and "slave", to be such that the slave entity listens for changes of master entity's state and reacts to them either by changing its own state or by objecting to its master changing state (*veto*). The slave entity of such a relationship can take part as a master in other relationships, having its own slave entities. So, given a finite number of entities, one can define an open chain of Master-Vetoable slave relationships, where first entity is master of the second entity, second entity is master of the third entity and so on.

*Event propagation*

State change events at the head of a master-slave relationship chain propagate towards its tail, since each slave tries to react and adapt to its master's state change event, changing state and thus causing its own slave to react and try itself to change state and so on (Fig. 2).

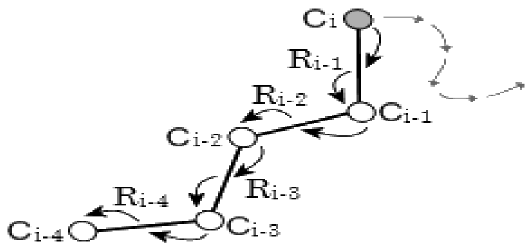

Fig. 2. Change event propagation from master to slave(s).

Any part of the chain and not just the chain endpoints can initiate the chain motion, acting autonomously to avoid some obstacle or receiving its own motion commands from a planner module or a human operator. The chain is treated as two sub-chains, both having as head the part that initiated the motion, and as tails the head and the tail of the original chain respectively (Fig. 3).

To support replaning 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 can object (veto) to the motion of its master part if it cannot move to adapt to the master's motion while still preserving the chain model and environmental constraints. Veto back-propagation practically reverses the master-slave hierarchy for the given event. A stationary (fixed) robot's connector agent will always object to any relocation of it (Fig 3).

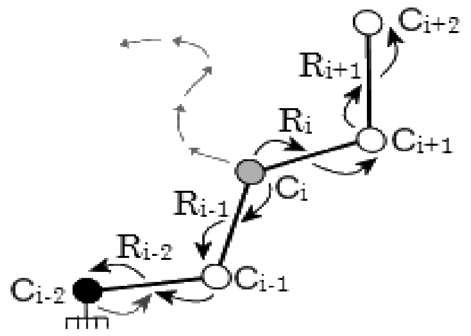

Fig. 3. Event propagation from master connector to the two sub-chains it defines, and veto event back-propagation.

*Reactive behaviours*

Each slave entity follows a predefined rule set that defines how it reacts to changes of its master entity's state. That rule set defines reactions that try to preserve some constraints imposed on the slave object and its relation to the master one, either by design (internal, hardware constraints) or decided on the fly at runtime (environmental constraints, obstacles or failure of subparts).

The reaction rules employed are exposing two basic model constraint preservation behaviours, named *Push* and *Pull*, with the former one caused when the master-slave connectors' distance exceeds the minimum length of the rod between the two connectors, and the later one caused when that maximum length is exceeded.

When a master connector is "pushing" a slave connector, an immediate neighbour of it on the chain, the effect is that the slave connector moves on the guiding line defined by the new position of the master and the old position of the slave respectively (Fig. 4-5). The slave is pushed away on the guiding line only if its distance from the new position of the master is less than the minimum length of the (potentially shrinkable) rod that connects it to the master connector.

A similar situation exists when a moving master connector is "pulling" a slave one. The slave connector is pulled on that same guiding line as before, but only if its distance from the master exceeds the maximum length of the (potentially expandable) rod that connects it to the master connector (Fig. 4-5). The pushing and pulling actions of a master connector on its slave connector neighbours are propagated via the respective slave rod agents that match the slave connectors to their master.

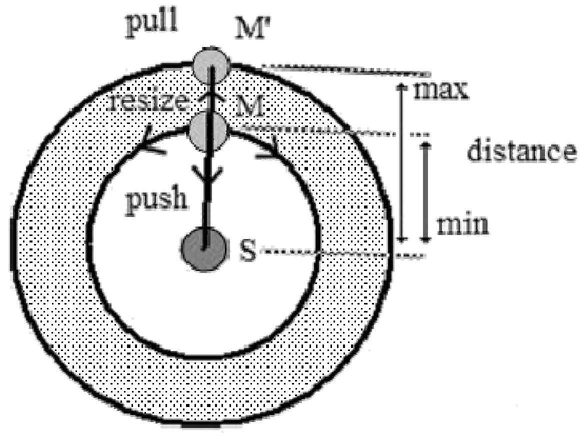

Fig. 4. "Push-Pull-Resize" behaviour and respective reaction areas defined around a slave connector (S: slave's position, M: master's old position, M': master's new position).

*Obstacle avoidance*

During the reaction of a slave connector to the motion of a master connector towards a new position in the free space, a potential obstacle collision for the rod that connects the master and slave connectors may block the realization of the new placement of the slave connector that is obtained using the push or pull behaviours described above. In such a case, the rod rotates around the master connector as if the motion of the master connector was causing a rotation (at the rod's obstacle collision point that is closest to the master) of the guiding line on which the rod would have lied if the obstacle wasn't there (Fig. 5). This environmental constraint preservation behaviour is named *Push-Pull-Rotate*.

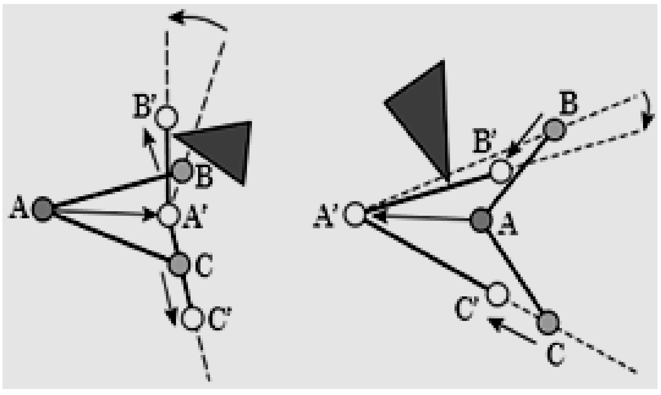

Fig. 5. "Push-Pull-Rotate" behaviour. Slave moves towards old master position until guiding line defined by new master position and slave is collision free, then moves on that line.

*Exploratory behaviour*

For navigation of the motion initiating agent (controller) a wall following algorithm has been extended with target seeking and corridor passage behaviours, using a set of discrete steps:

1) A reading of the left-front and right-front side (laser scanner) sensors is acquired to get the closest distance to visible obstacles at the left and right side of the controlling agent respectively (Fig. 7).

2) Based on the acquired sensor readings, the mover's position adjustment is calculated. To achieve a minimum and maximum distance from obstacles (experimented in simulation with 0.7 and 0.9 meters respectively), the mover's reactive behaviour is defined using a stack of subsumption layers, originally introduced by Brooks (1986). The lower subsumptive layers can override the behaviour exposed by higher layers, with the collision avoidance implemented at the lowest behavioural layer.

2.1) KEEP-MOVING: The highest layer dictates the start and preservation of motion, keeping the current mover direction. If a specific direction the mover should follow to a target is available (for example if the environment is enhanced with a force field gradient function), that direction could be used instead.

2.2) TARGET-CHECK: This layer checks for target reach in order to stop the motion, so that any target handling behaviour can then be initiated.

2.3) FAR-RIGHT: This layer checks if the right sensor gives a distance reading above the maximum distance and turns the mover to the right. Tactile pressure or IR and Sonar sensors could also be used; sensing touch or close collision with an obstacle, instead of trying to keep some predefined minimum distance from obstacles.

2.4) FAR-LEFT: This layer checks if the left sensor gives a distance reading above the maximum distance and above the right sensor reading (that was acquired at the previous layer) to turn the mover to the left. Overriding the previous layer's behaviour is avoid if the mover is very far from obstacles at both the right and the left side, for optimization reasons. In hardware based multilayer implementations one might better choose not to exchange much information between subsumptive layers, or read the right sensor again here and thus override the previous layer's behaviour to only go towards the left if the mover is far from obstacles at both the left and the right side.

2.5) CLOSE-RIGHT: A check is done if the right sensor gives a distance reading below the minimum distance, in which case the mover must turn to the left to avoid an eminent collision.

2.6) CLOSE-LEFT: At the lowest layer, an eminent collision check is done using the left sensor, turning to the right if needed. An optimization is done again here as in layer 2.4 above, which could be skipped at a hardware implementation for simplicity in layer design and to allow easier exchange of the subsumptive layers' order (the layer stack) to experiment with the emergent behaviour of the system.

**Table 2. Subsumptive layers (vr: right wheel speed, vl: left wheel speed)**

| *Priority: Layer* | *Trigger condition* | *Output* |
|---|---|---|
| 6: KEEP-MOVING | Always | vl=vr=s |
| 5: TARGET-CHECK | Target detection | vl, vr to target |
| 4: FAR-RIGHT | Leaving right wall | vl=s, vr=0 |
| 3: FAR-LEFT | Leaving left wall but not right one | vl=0, vr=s |
| 2: CLOSE-RIGHT | Wall close to right | vl=0, vr=s |
| 1: CLOSE-LEFT | Wall close to left but not to right | vl=s, vr=0 |
| 6: KEEP-MOVING | Always | vl=vr=s |

The output after the final one of the above layers is the resulting mover behaviour. At that point, regarding turning, the mover can turn left or right immediately by a prearranged number of degrees, or turn more depending on the error between the current distance and the offended distance limit (be it min or max distance that needs to be preserved), or even use the notion of a left and right wheel speed setting and thus turn gradually as time elapses.

The above algorithm enables passing through narrow corridors (even when they have high curvature) which was the main goal behind its design. Also it should work with a dynamically changing environment and moving obstacles, as long as they have a speed comparable to the mover's speed.

## 4. EXPERIMENTS

A testbed environment for 2D and 3D simulations has been developed (in Object Pascal / Delphi programming language), scriptable using PascalScript and Logo languages. The proposed multi-agent architecture has been implemented at this environment achieving real-time snake-like emergent motion for swarms of over 450 agents.

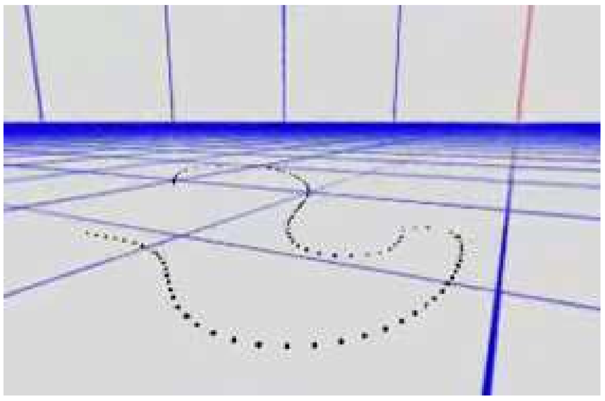

Fig. 6. Snake-like motion of a 140 mobile robots swarm.

This architecture has also been implemented (in SAX Basic language) at the 2D mobile robot simulator MobotSim for a swarm of mobile robots, where collision avoidance at narrow highly curved corridors has been achieved for a wanderer mobile robot guide equipped with a laser (SICK) sensor and 15 mobile robot "slaves" following it at a snake-like formation, using simpler close-collision detection sensors.

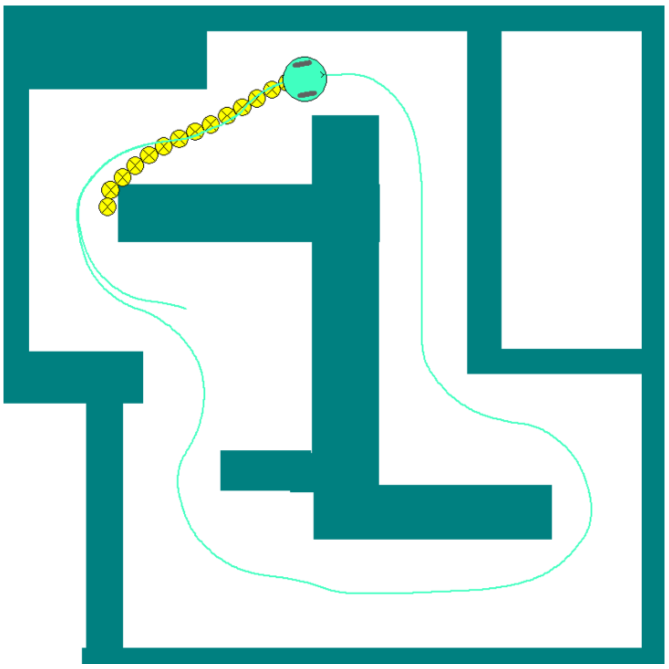

Fig. 7. Snake-like swarm formation navigating in corridors.

## 4. FUTURE WORK

It is planned to study and measure this method's results in generating solutions for the inverse kinematic problem at highly redundant manipulators and physical or virtual (snake-like swarms) reconfigurable chain linkages. It has been observed to automatically produce a solution for the inverse kinematics problem and reconstruct a chain manually broken into scattered parts or even into fewer parts during a pause of the simulation, with the provided solution seeming to respect the original relative placement of the chain parts.

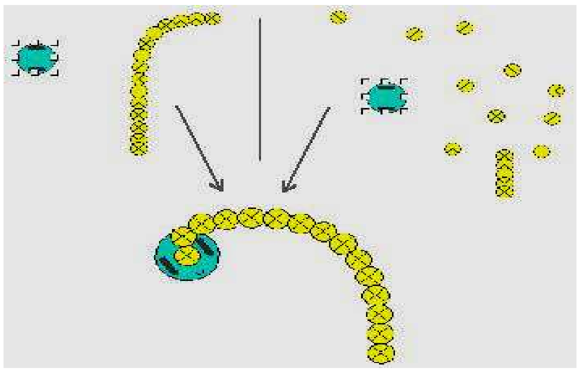

Fig. 8. Solution of the inverse kinematics problem.

It is also worth investigating the possibility of making use of the presented architecture in protein folding research (a distributed approach to protein folding simulation is presented in Snow (2002)). Proteins are the moving power of biology, however before they are able to perform their important functions, they self-configure their 3D structure (folding), a process vital and fundamental, but from many sides still unexplained. When proteins don't fold correctly, there may be serious implications, like the Alzheimer disease, mad cow disease, Parkinson and various cancers and related syndromes.

## 5. CONCLUSIONS

The proposed architecture reduces the problem of motion planning for chain formations to that of the planning for only the master part of the chain (usually the leading mobile robot). The rest of the chain parts adapt in real-time to the master's motion and at the same time they react to avoid detected obstacles.

Its best application is at swarms with high number of robots that try to move through narrow corridors. At such situations it can scale up efficiently, since an agent only needs to sense obstacles locally and interact with its two neighbouring ones at the control chain.

A direct application of the presented algorithm (see Appendix A) can also be in real-time 3D graphics. Rigging, that is using bones, joints and respective "weights" to transform surface polygons and textures of 3D models, in order to simulate articulated motion, skin, cloth, etc. has become common practice, but is still requiring lots of computational resources. Calculations are either performed offline (not in real-time) or the visual quality of the produced results are limited for the sake of a real-time implementation, due to the non-scalable classic inverse kinematics algorithms that are being employed.

It should be noted that this architecture isn't concerned with the situation where a veto from a slave agent back-propagates up to the agent that had initiated motion in the first place. In that case, the motion initiator has to replan its motion if moving autonomously, or let the veto propagate further to a planning unit or to a human operator, since the rest of the chain can't follow it.

## 6. ACKNOWLEDGMENTS

University of Patras is a partner of the EU-funded FP6 Innovative Production Machines and Systems (I*PROMS) Network of Excellence (www.iproms.org).

## REFERENCES

Birbilis, G., Aspragathos N. (2006). Multi-agent snake-like motion with reactive obstacle avoidance. *I*PROMS 2006 virtual conference*.

Bonabeau, E., M. Dorigo, and G. Theraulaz (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2, 14-23.

Chen, P., and Hwang, Y. (1992). Sandros, a motion planner with performance proportional to task difficulty. *Proceedings of IEEE Int. Conf. on Robotics and Automation*.

Challou, D., Boley, D., Gini, M., Kumar, V., Olson, C. (1998). In Kamal Gupta and Angel P. del Pobil (ed.), *Practical Motion Planning in Robotics: Current Approaches and Future Directions*.

Dorigo, M., Trianni, V., Sahin, E., Labella, T., Grossy R., Baldassarre, G., Nolfi, S., Deneubourg J-L., Mondada, F., Floreano D., Gambardella, L.M. (2004). Evolving Self-Organizing Behaviours for a Swarm-bot. *Swarm Robotics special issue of the Autonomous Robots journal*, vol. 17 (2-3), 223-245.

Hwang, Y., and Ahuja, N. (1992). Gross Motion Planning – A Survey. *ACM Computing Surveys*, vol. 24 (3), 219-291.

Liu, J. (2001). *Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation*. World Scientific, Singapore.

Liu, J., Jing, H., and Tang Y. (2002). Multi-agent oriented constraint satisfaction. *Artificial Intelligence,* no. 136, 101-144.

Snow, C.D., Zagrovic, B., Pande, V.S. (2002). The Trp Cage: Folding Kinetics and Unfolded State Topology via Molecular Dynamics Simulations. *Journal of the American Chemical Society*, vol. 124 (49).

## Appendix A. Algorithm

The architecture "push-pull-rotate with veto", is presented below in an object-oriented algorithmic form:

```
//----------------------------------------------------------------
// Move a (master) connector to the given position
//----------------------------------------------------------------

Motion (master: Connector, new_pos: Position)

   master.Position := new_pos
   Propagation(master, 0)   //propagation on chain to one…
   Propagation(master, 1)   //…and the other direction

//----------------------------------------------------------------
// Propagate the event of a master connector's motion
// to a neighbouring slave connector on the given direction
// in the chain formation
//----------------------------------------------------------------

Propagation (master: Connector, direction: integer)

local_variables:
   slave: Connector

slave := master.Slaves(direction)
If exists(slave)
   If not Adaptation(master, slave) then
      direction := -direction   //back-propagate veto
  Propagation(slave, direction)  //propagate using  recursion

//----------------------------------------------------------------
// Adapt given slave connector part position to the new
// position of the given master connector part, returning
// false if slave part failed to adapt and true is succeeded
//----------------------------------------------------------------

Adaptation (master, slave: Connector): Boolean

local_variables:
   master_pos, slave_pos, diff, pos: Position
   distance: real
   rod: Rod

master_pos := master.Position
slave_pos := slave.Position
rod := master.Rod(slave)
result := false

//ROTATE (ROD)//

Do While CollideSegment(master_pos, slave_pos, pos)
   //shake the colliding position and use that one together
   //with the new master position to define slave motion axis
   slave_pos := Sign(slave_pos - location) * Random + pos

//calculate slave motion axis (guiding line)
diff := slave_pos – master_pos
distance := diff.Length

//PULL SLAVE (CONNECTORS)//

If (distance > rod.MaxLength) then    //excess distance
   If slave.Fixed then exit    //Stationary slaves can't move
   pos := master_pos + diff * (rod.MaxLength / distance)
   If not slave.Motion(pos) then exit

//PUSH (SLAVE CONNECTORS)//

Else If (distance < rod.MinLength) then    //excess approach
   If slave.Fixed then exit    //Stationary slaves can't move
   pos := master_pos + diff * (rod.MinLength / distance)
   If not slave.Motion(pos) then exit

//RESIZE (ROD)//

Else
   result := true //slave motion isn't needed
```