Multi-agent snake-like motion with reactive obstacle avoidance

G.I. Birbilis[a], N.A. Aspragathos[a]

[a] *Department of Mechanical Engineering & Aeronautics, University of Patras, Panepistimioupoli Rion, Greece*

Abstract

A geometric constraint satisfaction approach to serial linkage (chain) motion and reactive obstacle avoidance, based on a multi-agent architecture, is presented. Application of this architecture to achieve real-time motion planning for serial manipulators and mobile robot snake-line swarm formations is suggested. A hierarchy of "Master – Slave" relationships is used, with the event of an autonomous motion of a controller agent propagating to its two neighbouring ones in the formation chain and progressively further on towards the two endpoints of the chain. At each propagation step, a constraint preservation mechanism enforces the respecting of minimum and maximum distance constraints between pairs of consecutive chain agents. The emergent behaviour of the multi-agent system equals to having the moving part of the chain push or pull the two subparts of the chain it connects. To cater for fixed base manipulators, and to allow replaning in case some slave part of the chain formation can't adapt to its master's motion, being trapped in some obstacles or malfunctioning, the notion of a "Master – Vetoable slave" relationship is used, where a slave part can object (veto) to the motion of its master part.

Keywords: Reactive snake-like motion, Redundant manipulators, Swarms, Obstacle avoidance, Geometric constraint satisfaction, Multi-Agent systems, Master - Vetoable slave relationship, Push-Pull-Rotate behaviour

1. Introduction

In this paper we consider the usage of a geometric constraint satisfaction approach to calculate a virtual serial linkage (chain) motion, which could be a serial manipulator or a mobile robot snake-line swarm formation, with its environment possibly containing unknown stationary or moving obstacles. It is supposed that each part of the chain receives sensor feedback concerning its proximity to obstacles, in order to react accordingly.

Many approaches to the problem of motion planning in a time varying environment have been proposed, most of which are thoroughly examined in Hwang and Ahuja [1]. The case of unknown moving obstacles existing in the environment greatly favours local planning methods, since global re-planning would be too expensive, especially in the case of many obstacles or when having to plan for a system with many degrees of freedom (highly redundant), as pointed out by Chen and Hwang [2], and Challou et al. [3].

The more degrees of freedom (DOFs) a system has, the higher its flexibility is, so we mostly value an approach that easily scales up to highly redundant systems, while still supporting those with less DOFs. A top-down centralized approach would have increasing complexity and cost as the number of DOFs rises, so a bottom-up, modular approach is suggested, modelled as a *multi-agent system*, using the multi-agent system design concepts presented by Liu [4].

In the mobile robotic swarms [5] field, Dorigo et al. [6], while evolving self-organizing behaviours for their “swarm-bot”, experimented with eight robots connected by flexible links to create a snake formation and 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, because swarm members tend to maintain their direction of movement, the system displays an ability to go through narrow passages, eventually deforming its shape according to the configuration of the obstacles.

In the robotic manipulators’ field, almost all the approaches based on a multi-agent platform to address the motion planning problem are doing motion planning on each subpart of the manipulator structure and combine the solutions of those sub-problems into a solution for the global problem. In most of the cases, composition is implemented in real-time, via interaction (cooperation or contention) of the respective agents that control the various parts of the manipulator. Overgaard et al. [7], proposed a multi-agent system comprised of both joint and link agents, to control a 25-DOF snakelike robot in an environment with obstacles which are modelled using an artificial potential field, pioneered by Khatib [8]. Bohner and Lüppen [9], applied a similar concept on a 7-DOF robot, considering only the robot joints as agents. They are doing both sensor-data integration and motion planning on a per-agent basis, thus decomposing the problem and reducing its complexity to the sum of its subproblems.

In our previous work [10], we proposed a multi-agent system where each software agent controls a specific part of a planar manipulator’s joint-link chain, and agents interact with each other towards the adaptation of the manipulator configuration to external events and changing situations in real-time. We presented a geometric constraint satisfaction approach that reduces the motion planning of a manipulator to motion planning of a single part of it, for instance the manipulator tooltip, having the rest of the manipulator chain parts react and adapt or object (veto) to the moving part’s motion. In this paper, we slightly revise the agent interactions and implement the proposed architecture in both the 2D and 3D space, for either physical chain linkages (manipulators) or virtual ones (swarms of mobile robots), presenting new findings on the potential applications of this approach.

2. Problem definition

We’re considering the case of a 3D serial chain structure where every two consecutive chain elements have to maintain a given minimum and maximum distance constraint between them, and every three consecutive elements have to maintain a minimum and maximum angle constraint.

The chain can be a serial manipulator which can have both rotational and translational joints, in any combination. The base of the manipulator may be fixed or able to move around either freely or on some predefined path. Alternatively, 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 with a speed comparable to the manipulator joint motors (or to the mobile robots in the 2nd case) speed. The case of moving obstacles includes the case of other entities (mobile robots, manipulators, humans) also acting in the same environment, with which no communication channel or common protocol exists for coordination. No a-priori knowledge of obstacles is assumed and 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. [11] 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 our multi-agent system model, we control the chain’s motion in a way that mimics the motion of a chain of *rods*, interconnected at their endpoints with *connectors* (potentially constrained universal joints). The chain’s two endpoints are connectors, while some rods could 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 kept. Given a high number of rods and respective connectors, this model behaves like a snake crawling amidst obstacles.

3.1. Mapping kinematic to conceptual constraints

In the case of a manipulator chain, the base connector (that links to the environment) is used to cater for a potentially mobile (free or partially/totally constrained) manipulator base. A rotational joint and its outgoing link map directly to the first connector of a fixed-length rod, whereas a translational joint and both its incoming and outgoing links are represented by a connector and an expandable rod. Each connector agent has a position property and each rod agent has a length property, such that the length bounds of a rod are respected by the positions of the connector agents placed at the rod's two endpoints. The values of joint parameters can be calculated geometrically from the connector positions at any time, using the mapping shown in Fig. 1.

In the case of a mobile robot swarm formation, the rods represent distance constraints between pairs of consecutive robots, while the connectors represent angular constraints between triplets of consecutive robots in the chain formation. Mapping between conceptual and kinematic model in swarms robot chains is done the same as described above for manipulator chains.

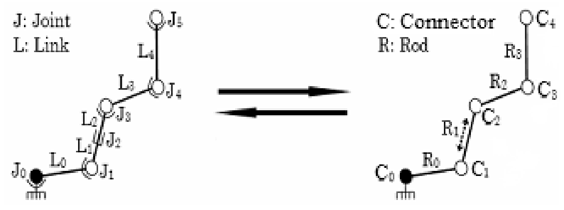

Fig. 1. Kinematic to/from Conceptual model mapping.

3.2 Inter-agent relationships

We define a *Master – Vetoable slave relationship* between two entities, a "master" and a "slave", to be such that the slave entity listens for changes of the 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 still 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 the first entity is the master of the second entity, the second entity is the master of the third entity and so on.

3.3 Change event propagation

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

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 for tails the head and the tail of the original chain respectively, as shown at the right side of Fig. 2.

To cater for fixed base manipulators, and 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. At the right side of Fig. 2, a manipulator with a fixed base is shown, where the connector agent at the manipulator's fixed base always objects to any relocation of it.

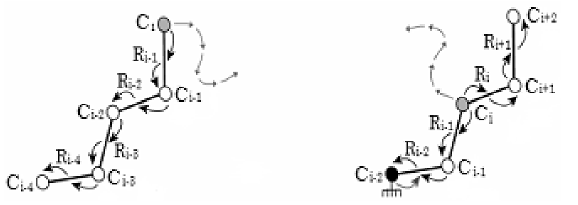

Fig. 2. Change event and veto propagation.

3.4. Reactive agent ruleset

Each slave entity follows a predefined *rule set* that defines how it reacts to the 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 we employ 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 that connects 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. 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, as illustrated in Figs. 3-4. The pushing and pulling actions of a master connector on its slave connector neighbours are obtained via the respective slave rods that connect the slave connectors to their master.

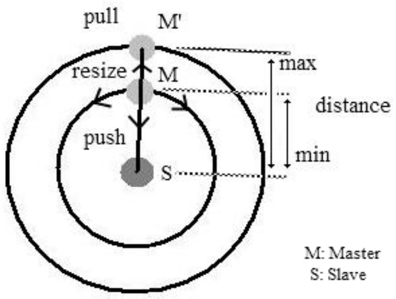

Fig. 3. Push-Pull-Resize behaviour.

3.5. Reaction to sensed obstacles

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 at section 3.4. 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, as shown in Fig. 4. We name this environmental constraint preservation behaviour as *Push-Pull-Rotate*.

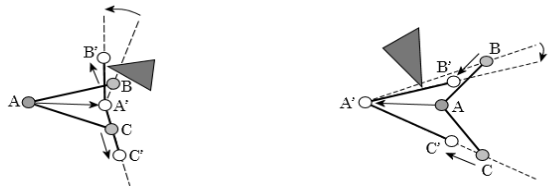

Fig. 4. Push-Pull-Rotate behaviour.

3.6. Navigation

For navigation of the motion initiating agent (controller) we extended a wall following algorithm with target seeking and corridor passage behaviours, having the following 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, as shown in Fig. 6.

2) Then, based on the sensor readings above, the mover's position adjustment is calculated. To achieve a minimum and maximum distance from obstacles (we experimented in simulation with 0.7 and 0.9 meters respectively), we define the mover's reactive behaviour using a stack of subsumption layers [12]. The lower subsumptive layers potentially replace the behaviour exposed by the higher layers, with the collision avoidance being the lowest behavioural layer.

2.1) KEEP-MOVING: The highest layer dictates the start and continuiment 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 characterised by 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: 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 the obstacles.

2.4) FAR-LEFT: 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. We avoid overriding the previous layer's behaviour 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 to not exchange much information between subsumptive layers or read the right sensor again at this layer and thus override the previous layer's behaviour and 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 and to avoid an eminent collision the mover must turn to the left.

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

Table 1
Subsumptive layers[a]

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

[a]vr: right (virtual) wheel speed, vl: left (virtual) wheel speed, square brackets denote optional trigger optimization.

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 (be it real or virtual wheels depending on the problem) 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 Pascal and Logo scriptable testbed application for 3D simulations has been developed using the Object Pascal programming language and RemObjects PascalScript scripting engine, combined with an implementation of a Logo to PascalScript transcoding compiler. The proposed multi-agent architecture has been implemented as a Pascal language script in this environment, shown in Fig. 5, achieving snake-like realtime motion at numbers of 450 and more agents.

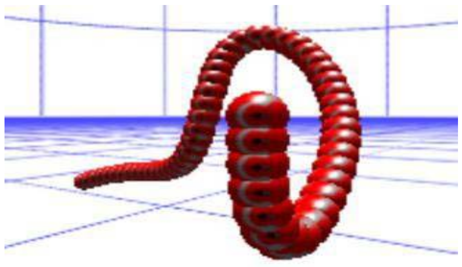

Fig. 5. Snake-like manipulator motion in the 3D space.

A 2D version of the same architecture has been implemented at the MobotSim mobile robots simulator for a swarm of mobile robots using SAX Basic script, where obstacle avoidance in narrow curved corridors has been achieved for a wandering controller robot equipped with a laser scanner and 15 slave robots following it in a snake like formation using simpler close-collision detection sensors, as shown in Fig. 6.

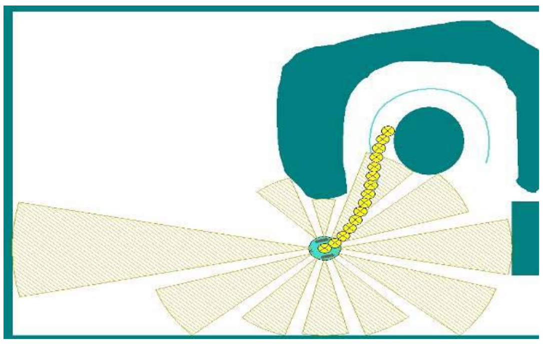

Fig. 6. Swarm evasion from highly curved corridor.

5. Future work

We plan to implement collision detection at the 3D simulator as well, where given the modular design we used, minimal changes should only be required to the implementation of the constraint preservation rules, having the rest of the multi-agent system and event propagation design kept intact.

Also, we wish to investigate further and measure the performance of this system in acquiring solutions for the inverse kinematics problem of highly redundant manipulators and physical or virtual (e.g. snake-like moving swarms) reconfigurable chain linkages in general (see Fig. 7). Currently the 2D simulation has been observed to automatically provide an inverse kinematics solution and reconstruction of a chain that had been manually broken into scattered or even to fewer parts during a simulation pause, with the reconstructed solution seeming to respect the relative formation of non-broken parts at the two ends of the chain where possible.

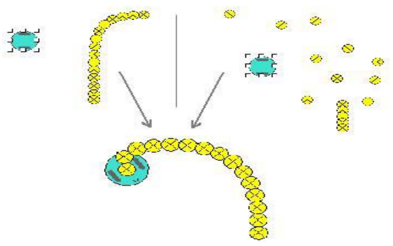

Fig. 7. Solution of the inverse kinematics problem.

6. Conclusions

Our architecture reduces the problem of path planning for chain formations to that of planning for only a master part of the chain (usually the tool-tip or leading mobile robot). The other chain parts adjust in real-time to the master's motion and at the same time react to avoid sensed obstacles. Best application should be at highly redundant or reconfigurable manipulators and swarms with high number of robots trying to move through narrow corridors. At such settings it can scale up efficiently, since an agent only needs to sense obstacles locally and interact with its two neighbouring agents in the control chain. This architecture isn't concerned with the situation where a veto from a slave agent back-propagates to the original motion initiating agent. In that case, the motion originator should replan its motion if it's an autonomous mover, or let the veto propagate further to a planner module or human operator that is controlling it, since the rest of the chain can't follow its motion.

Acknowledgements

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

References

[1] Hwang, Y., and Ahuja, N. Gross Motion Planning – A Survey. ACM Computing Surveys, no 3, vol 24 (1992) 219-291.

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

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

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

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

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

[7] Overgaard, L., Petersen, H., and Perram, J. Reactive Motion Planning: A Multi-agent Approach. Applied Artificial Intelligence, no 10 (1996) 35-51.

[8] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotic Research, no. 5 (1986) 90-98.

[9] Bohner, P., and Lüppen, R. Reactive Multi-Agent Based Control of Redundant Manipulators. Proceedings of the 1997 IEEE Int. Conf. on Robotics and Automation, Albuquerque, New Mexico (1997).

[10] Birbilis, G. and Aspragathos N. In: Lenarcic J. and Galletti C. (Eds.) On Advances in Robot Kinematics. Kluwer Academic, Netherlands, 2004, pp 441-448.

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

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