Learning Intelligent Controllers for Path-Following Skills on Snake-Like Robots

Francisco Javier Marín[1], Jorge Casillas[1], Manuel Mucientes[2], Aksel Andreas Transeth[3], Sigurd Aksnes Fjerdingen[3], and Ingrid Schjølberg[3]

[1] University of Granada, Granada, Spain
[2] University of Santiago de Compostela, Santiago de Compostela, Spain
[3] SINTEF ICT Applied Cybernetics, Trondheim, Norway

Abstract. Multi-link wheeled robots provide interesting opportunities within many areas such as inspection and maintenance of pipes or vents. A key functionality in order to perform such operations, is that the robot can follow a predefined path fast and accurately. In this paper we present an algorithm to learn the path-following behavior for a set of motion primitives. These primitives could then be used by a planner in order to construct longer paths. The algorithm is divided into two steps: an example-based stage for controller learning, and a controller tuning stage, based on an objective function and simulations of the path-following process. The path-following controllers have been tested with a simulator of a multi-link robot in several complex paths, showing an excellent performance.

Keywords: Path-following, snake-like robot, multi-link mobile robot, fuzzy control.

1 Introduction

Mobile robots constitute versatile platforms for a vast range of operations. In particular, multi-link mobile robots have the potential of traversing complex structures and narrow and confined spaces, which can be either too difficult or too dangerous for people to operate in. A key functionality in order to perform such operations, is that the robot can follow a predefined path fast and accurately. Moreover, the robot should also be able to recover to the path even in the case of large deviations (e.g. after avoiding an obstacle that was placed on the path).

The field of path-following for mobile robots is vast, but much of the focus has been limited to wheeled robots [1,2,3,4,5]. These robots have restrictions in their movements, as most of them are nonholonomic, but their kinematics are not as complex as for snake-like robots and, therefore, the complexity for path-following is lower.

In this paper we present an algorithm for learning a path-following behavior for multi-link mobile robots. Unlike in [4], the multi-link robot tries to reduce heading and deviation errors jointly at each step (and not independently), adjusting velocity and turning angle properly. Also, in our approach, the learning complexity of the path-following behavior is reduced, as long paths are divided into a set of small motion primitives [6] (Fig. 1a) that can reach almost any point in the neighborhood. These primitives can be used for forward and reverse motion, but the latter have been omitted in Fig. 1a for clarity. Thus, a set controllers are learned (one for each motion primitive), instead a single one. The combination of these motion primitives allows to construct longer paths from any two points in the environment (Fig. 1b). This method favors accuracy because a single controller is dedicated to a single primitive. The learned controllers can be used with a planner that permits the coupling of the path-following behavior with an obstacle avoidance behavior, but this planner is not implemented for this paper.

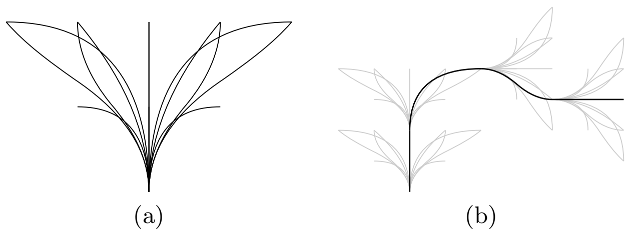

Fig. 1. a) A set of motion primitives. b) A repeated and regular pattern of motion primitives that constructs the overall motion plan (bold curve).

A nonholonomic multi-link wheeled robot called PIKo (Pipe Inspection Konda) is used as a basis for simulation trials of the algorithm presented in this paper. The simulation results show that this robot is able to follow the paths with great accuracy. Moreover, it is able to quickly move onto its path when it is initially placed with a large deviation. The presented approach can be applied to other mobile robots just replacing the kinematics model.

The kinematic model of PIKo makes the reverse motion controllability difficult, only with information of the first link. So, only controllers for forward motion have been learned. For this reason, has been assumed that in case of a dead end, the snake will replan the path so that it can turn around.

This paper is arranged as follows: Sec. 2 presents the multi-link robot used for the simulation trials, while Sec. 3 details the controller learning algorithm for the path-following behavior. Sec. 4 analyzes the obtained results and, finally Sec. 5 points out the conclusions.

2 Description of PIKo Robot

The approach presented in this paper has been validated with a snake-like robot with active wheels (Fig. 2), called PIKo (Pipe Inspection Konda). PIKo is a nonholonomic robot developed by the Norwegian research organization SINTEF [7]. The robot currently consists of five interconnected modules, each with joints with two degrees of freedom. The wheels provide for a third degree of freedom per module. Each module has a measured maximum joint moment of 11 Nm and a weight of 1.2 kg. Current sensors include angle encoders for all joints, wheel encoders, and a 3D time-of-flight (TOF) camera system. The advantages of this robot are:

- The long and articulated body of snake-like robots make them ideal for internal inspection of complex pipe structures or other confined spaces. PIKo, moreover, can move through these narrow spaces, while maintaining the direct motion of wheeled vehicles. The direct motion is a great advantage against passive wheeled robots motion because it is less complex.
- The robot has many degrees of freedom (speed, horizontal and vertical angles for each module) but its movement can simulate the n-trailer problem [9] through a set of kinematic equations (with good accuracy), reducing the number of parameters needed to control it[8].
- A PIKo simulator based on an open-source physics engine called Open Dynamics Engine [12] has been developed by SINTEF and the Norwegian University of Science and Technology (NTNU). We have used this simulator for controller tests and for the tuning stage of the learning method presented in this paper.

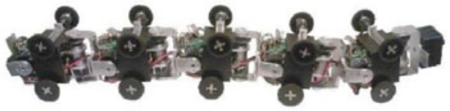
Fig. 2. The snake-like robot PIKo

3 Path-Following Learning Algorithm

A two step method is proposed to learn path-following behavior. This technique combines inductive and deductive learning in optimization methods, with a first example-based learning stage and a second tuning stage with data from a simulator, which improves the accuracy of the obtained controllers. The proposal is valid for any learning algorithm based on the optimization of an objective funcion (e.g. genetic algorithms or artificial neural networks).

The example-based learning stage is based on the algorithm proposed in [10]. It has been used in problems like wall-following or moving object following with differential steering robots. In this paper, the different steps of this algorithm have been adapted to be applied to the path-following behavior for multilink robots, from the selection of variables to the scoring function.

3.1 Example-Based Learning Stage

In this stage, a general fuzzy controller that will seed the tuning stage is learned through a set of path-following examples. Each example of the training set consists of a combination of state and action values. The examples are generated in the space of the input (state) variables, starting from the minimum value of each variable and increasing the value in a quantity p_i until the maximum value is reached. The set of examples is created combining these values for all the input variables. On the other hand, the action values are determined testing all the posible combinations of the output variables (discretized with precision p_i), and selecting those values that place the robot in the state closest to the ideal state according to a score function. These examples are used as the training dataset of an advanced algorithm that learns the fuzzy controller (database and rulebase) that best fits the data.

In order to perform this stage, we need to define: the kinematics model of the robot, the input and output variables, the universe of discourse and precision p_i for the example generation, the scoring function (*SF*) and the test function.

Kinematics Model. The kinematics model of the robot PIKo is described in [7,8]. This model is used to describe the motion of the robot for the evaluation of the examples. In order to calculate the next position of the head of the robot, the following equations are needed:

$$\theta_1(t+1) = \theta_1(t) + \frac{V_{P1}}{L_{PJ}} \cdot tan\,(\delta_1) \cdot \Delta t \tag{1}$$

$$x_1(t+1) = x_1(t) + V_{P1} \cdot sin\,(\theta_1) \cdot \Delta t,\ y_1(t+1) = y_1(t) + V_{P1} \cdot sin\,(\theta_1) \cdot \Delta t \tag{2}$$

$$\phi_2(t+1) = \phi_2(t) - \frac{V_{P1}}{L_{PJ}} \cdot \left[sin\,(\phi_2(t)) + \left(\frac{L_{JP}}{L_{PJ}} \cdot cos\,(\phi_2(t)) + 1 \right) \cdot tan\,(\delta_1) \right] \cdot \Delta t \tag{3}$$

where x_1, y_1 and θ_1 are the position and heading of the first link of the robot, ϕ_2 is the angle between the first and second links of the robot, V_{P1} and δ_1 are the linear speed and angular speed of the first link and L_{PJ} and L_{JP} are the lengths of the segment $P_i\ J_i$ and $P_i\ J_i$ being P_i the center point of the wheel shaft of link i and J_i the location of the front end of link i.

Input and Output Variables. For the path-following behavior, the deviation of the robot from the path must be minimal at each step. We can use Frenet frames to find the deviation in position and orientation of the robot head with respect to the expected path. Therefore, two of input variables will be the distance from the robot position to the closest point of the path (Δz) and the heading error ($\Delta\theta$). Instead of using the closest point heading to estimate the heading error, the closest point heading in the next step is picked. This approach has some advantages:

– The robot can control both deviation and heading at the same time.
– The recovery process is stable and soft, especially at curves.

The other two input variables are the current linear speed (v), and $\phi_2(t)$ (Eq. 3). Finally, the output variables are the linear aceleration (a) and the turning angle (δ_1).

Universe of Discourse. In order to generate the training dataset, the limits and precisions of the values of the variables have to be stablished. Some of the variables have very large universes of discourse ($\Delta z \in [-\infty, \infty]$, $\Delta\theta \in [-180^o,\ 180^o]$). For these variables, the universe of discourse must be a reduced version of the real universe, and it should contain those values of the variable that are meaningful for learning (high values of distances are not useful for learning, as for all of them the robot will execute the same action).

Taking into account the kinematic equations, and assuming that $\delta_1 \in [-20,\ 20]$, a maximum speed v (V_{P1}) of 0.2 m/s and time step $\Delta t = 0.1s$, the values for the different variables are: $\Delta z = 0.003m$, $\Delta\theta = 6^o$, and $\phi_2 = 15^o$. Although higher values for the limits are not really significant, the ranges have been extended for precision discretization. The final universes of discourse and precisions for each variable are: $\Delta z \in [-0.0042,\ 0.0042]$ (negative values are used on left deviation, positive on right), $p_{\Delta z} = 0.0007$; $\Delta\theta \in [-6,\ 6]$, $p_{\Delta\theta} = 1$; $\phi_2 \in [-15,\ 15]$, $p_{\phi_2} = 5$; $v \in [0,\ 0.2]$, $p_v = 0.1$; $a \in [-0.2,\ 0.2]$, $p_a = 0.05$; $\delta_1 \in [-20,\ 20]$, $p_{\delta_1} = 1$.

Scoring Function. An important aspect of the proposed example set generation technique is the definition of the *SF*, a function that evaluates the action of the fuzzy controller over an example. The role of *SF* is to measure the deviation of each variable from the desired value (the one associated to the ideal state). For the path-following behavior, the robot needs to reach the closest point of the desired path, but keeping a low heading error. This causes two major problems:

– If the robot is located at a point on the path, the best action is to stay in the same place, because other actions may increase heading or distance error.
– It is crucial to find a good balance between heading and distance error improvements because these two variables are hardly coupled: if we want to reduce the distance error, the heading error must be increased.

The solution for the first issue is to penalize low speeds, including the speed as a parameter with high weight on the score function. The second issue can be partially solved with dynamic weights: the weights of deviation and heading errors depend on their respective initial errors. When the initial distance error is small, its weight is lower, increasing the importance of heading and speed weights. Then, the score function is defined as:

$$SF(RB(e^l)) = \alpha_1 + \alpha_2 + \alpha_3, \tag{4}$$

where e^{l} is the l-th example and *SF* is the score of the state reached by the robot, starting at the state defined by e^{l} and applying the control action proposed by the combination of the output values of the rulebase (RB). α_1, α_2 and α_3 are computed as follows:

$$\alpha_1 = \omega_1 \cdot \frac{e^l_{\Delta z}}{max_{\Delta z}} \cdot \frac{\Delta z}{p_{\Delta z}},\ \Delta z = \sqrt{(x_{robot} - x_{path})^2 + (y_{robot} - y_{path})^2} \tag{5}$$

$$\alpha_2 = \omega_2 \cdot \frac{e^l_{\Delta\theta}}{max_\theta} \cdot \frac{\Delta\theta}{p_{\Delta\theta}}, \ \Delta\theta = |\theta_{robot} - \theta_{path}| \tag{6}$$

$$\alpha_3 = \omega_3 \cdot \frac{(max_v - v)}{p_v} \tag{7}$$

ω_1, ω_2 and ω_3 are three weights used for balancing the importance of each variable in the scoring function. These weights depend on the universe of discourse and the precision of the variables. $\frac{e^l_{\Delta z}}{max_{\Delta z}}$ is the dynamic weight for distance. $\frac{\Delta z}{p_{\Delta z}}$ determines the score of the action, divided by the precision of the variable. This makes possible the comparison of the deviations of different variables. The heading score is estimated in the same way and, finally, lower speeds are penalized on α_3, using the maximum speed (max_v) as limit. Thus, the best actions are those that set the speed closest to the maximum value and with the lowest distance and heading error. Therefore, the scoring function has to be minimized.

Test Function. After the learning process is performed, the quality of the fuzzy controllers has to be evaluated. This is done with the test function. This function simulates the path-following process in a path using the fuzzy controller that will be evaluated and the PIKo simulator. Deviation error (Δz) and current speed (v) are registered at each step until the robot reaches the final point or a maximum step limit. After that, average deviation and speed are calculated and presented together with the success flag: 1 if the robot reaches the final point, 0 in other case. Each controller is tested with several different paths.

3.2 Tuning Stage

Learning the whole fuzzy controller (database and rulebase) with examples is faster and easier than learning it with an objective function and the simulator. This is a quite theoretical learning and the generated controllers are not perfect, but they are a good starting point for the tuning stage. In this phase, the seed controller (obtained in the previous stage) is tuned based on several motion primitives, creating a set of fuzzy controllers (one for each primitive). For this task, a learning algorithm has to be used. It is run once for each motion primitive. This algorithm uses the previous generated controller rulebase and tries to improve the fuzzy database through an objective function. Also, this stage tries to improve the recovery process from large deviations. For this reason, the universe of discourse needs to be expanded for the variables Δz and $\Delta\theta$.

For this stage, we need to define: the set of motion primitives (Fig. 1a) and the objective function.

Objective Function. In the tuning stage we need to improve both deviation and recovering behaviors: the former by minimizing the distance error while maintaining a good average speed, and the latter by reducing the number of steps necessary to reach the path in a stable state (which does not generate future large deviations). The objective function (which has to be minimized) considers all this requirements:

$$ObjF = \omega_{dev} \cdot dev + \omega_{rec} \cdot rec \tag{8}$$

where ω_{dev} and ω_{rec} are the weights that determine the importance of deviation and recovering and *dev* and *rec* are the variables that measure the quality of the controller on both behaviors:

$$dev = \omega_{\Delta z} \cdot \overline{\Delta z} + \omega_v \cdot (max_v - \overline{v}) \tag{9}$$

$$rec = \omega_{\Delta z_r} \cdot \overline{\Delta z_r} + \omega_{rv} \cdot (max_{rv} - rv) \tag{10}$$

Eq. 9 uses the same parameters of the test function: average distance error ($\overline{\Delta z}$) and average speed ($\overline{v}$) during a controller full test (until the robot reaches the final point or a maximum limit of steps). These values are weighted with $\omega_{\Delta z}$ and ω_v respectively.

Eq. 10 uses the number of steps necessary to reach a point close to the desired path (rv), and the average deviation from the time instant the point was reached until the end of the controller test ($\overline{\Delta z_r}$). rv is calculated as:

$$rv = \frac{n_r}{n_{steps}} \tag{11}$$

where n_r is the number of steps needed to reach a point of the path, and n_{steps} represents the steps needed to complete the full test.

4 Results

4.1 Simulation Setup

Three different values for ω_1, ω_2 and ω_3 for the dataset generation have been tested, with five different seeds. Learned controllers have been tested on the PIKo simulator, with several paths of varying complexity. It is important to remark that these paths have not been used during training. In the first stage, the training set is only composed of a list of examples that have been chosen covering the input space with an adequate precision. Nine evaluations have been made for each controller: one without initial deviation and eight more with different initial offsets (for recovery testing), and average distance error, average speed and steps needed for recovering have been recorded (Table 1).

Controller learning and tuning have been realized with an advanced genetic fuzzy system called EGLFP [11], especially developed for fuzzy learning. We have used the following parameter values for this algorithm in both stages: 50,000 evaluations for learning and 5,000 for tuning, 50 individuals, 0.8 as crossover probability and 0.3 as mutation probability.

The set of motion primitives presented on Fig. 1 has been used in the tuning phase. It consists of 11 different primitives, so 11 different controllers have been learned. We have used a modified version of EGLFP for this task, replacing example learning with the objective function defined in Eq. 8 and data from the PIKo simulator. The weights that have been used on this phase are the following: $\omega_{dev} = 0.6$, $\omega_{rec} = 0.4$, $\omega_{\Delta z} = 0.99$, $\omega_v = 0.01$, $\omega_{\Delta z_r} = 0.95$ and $\omega_{rv} = 0.05$. Five seeds have been used for each pattern and each controller has been evaluated 9 times, like in the first stage. Average speed, steps needed for recovering and average deviation after recovering are presented at Table 2.

4.2 Path-Following

In this section we present the results of the different learned controllers, from the first and second stages, and a short study of the three weights of the score function of the first stage. Table 1 collects the results of these weights for the path-following problem with and without initial offset.

Table 1. Path-following deviation data. Deviation values are in meters and speed in m/s.

Weights (ω_1, ω_2, ω_3)	Right (2,2,90)	Left (1,2,0)	Left (1,1,90)	Right (2,2,45)
	Dev/Spd/Rec	Dev/Spd/Rec	Dev/Spd/Rec	Dev/Spd/Rec
(0.8, 0.2, 1)	0.092/0.17/0.69	0.102/0.13/0.72	0.025/0.13/0.80	0.88/0.16/0.68
(0.775, 0.225, 1)	0.022/ 0.2 /0.74	0.011/ 0.2 /0.71	0.022/ 0.2 /0.84	0.018/ 0.2 /0.69
(0.7, 0.3, 1)	0.024/ 0.2 /0.80	0.016/ 0.2 /0.77	0.022/ 0.2 /0.90	0.037/ 0.2 /0.79

Some of the tests with the weight combination of the first row have not reached the final position in the maximum number of steps, some of them were blocked at some point (first issue described in the score function subsection) and others caused for extreme heading deviation (second issue). These problems are represented in Fig. 3 (a and b). On the other hand, when the heading error has greater importance, the recovering process is slower or the robot never recovers (we can see this in the *rec* column of second and third rows). Without dynamic weights, a valid balance is never found. We can select any controller generated with the weights of the second row to be the seed of the tuning stage (Figure 3c).

Table 2 presents the deviation error, speed, and necessary steps for recovering after the tuning phase. Each motion primitive is identified with its direction, x and y displacements, and heading change. Each controller was tested with its corresponding motion primitive.

As we can see in Table 2, the proccess is very accurate: only a few millimeters of average deviation for all the patterns, with good speed. The recovering process is also improved: only a 15-30% of the total steps are needed for recovering offsets of 0.2 m (around 40% in the shortest paths). Fig. 4 shows other tests of different primitives.

Figs. 3c and 4c present the same paths with the same initial deviation. We can see that the tuning phase has greatly improved both recovery speed and deviation error.

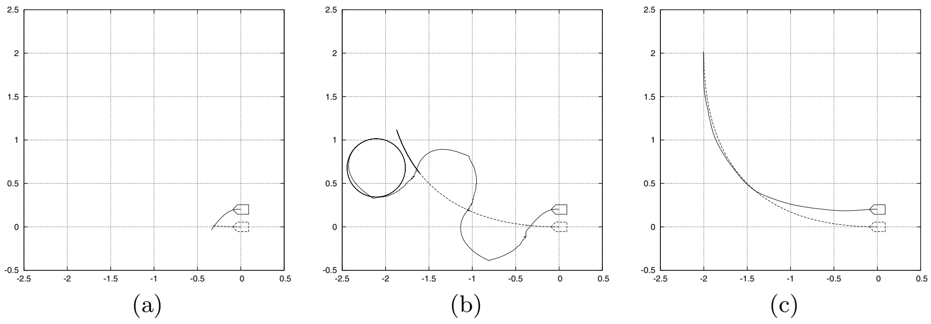

Fig. 3. Examples of controller tests. a) Blocked. b) Extreme heading error. c) One test of the controller selected as seed. The dotted curves are the ideal paths, continuous curves are the robot steps.

Table 2. Path-following deviation data after the tuning stage. Deviation values are in meters and speed in m/s.

Path	Dev/Rec/Spd	Path	Dev/Rec/Spd
Line (2,0,0)	0.002/0.24/0.13	Right (2,2,45)	0.006/0.21/0.14
Left (1,1,90)	0.009/0.41/0.16	Left (1,2,45)	0.005/0.31/0.16
Right (1,1,90)	0.007/0.36/0.15	Right (1,2,45)	0.002/0.21/0.13
Left (2,2,90)	0.001/0.16/0.14	Left (1,2,0)	0.007/0.33/0.15
Right (2,2,90)	0.002/0.16/0.13	Right (1,2,0)	0.008/0.25/0.13
Left (2,2,45)	0.004/0.20/0.14		

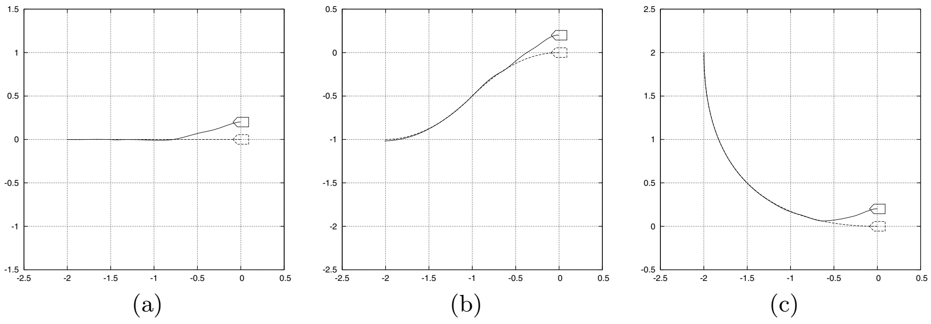

Fig. 4. Examples of controller tests. a) Line (2, 0, 0), b) Left (1, 2, 0), c) Right (2, 2, 90).

5 Conclusions

A two stage learning method for the path-following problem has been presented. It consists of a example based learning phase and a subsequent tuning phase for accuracy improvement. This process is applied to a set of motion primitives, obtaining a set of fuzzy controllers that could be combined in order to build longer paths. A tuning stage for the controllers is used to produce a very accurate path-following behavior: distance errors are reduced from various centimeters to several millimeters in this stage. This is very important for snake-like robots, as they can move in narrow spaces like pipes or vents where large path deviations must be avoided. In addition, the methodology also steers the robot quickly onto its desired path even for large initial deviations. Reverse motion controllers can also be learned with this method, but the kinematic function of the robot makes the reverse motion controllability difficult using only head information. This will be a topic for future work.

The use of motion primitives allows the robot to reach all the points of the lattice space with the combination of a few primitives. This facilitates the design of a planner, and reduces the path complexity for the learning algorithm. The areas of robot learning and multi-link robots are rapidly expanding and will eventually provide systems for autonomous inspection and maintenance operations. The results provided in this paper are steps toward such robot functionality.

References

1. Lapierre, L., Zapata, R., Lepinay, P.: Combined Path-following and Obstacle Avoidance Control of a Wheeled Robot. The International Journal of Robotics Research 26(4), 361–375 (2007)
2. Ashoorirad, M., Barzamini, R., Afshar, A., Jouzdani, J.: Model Reference Adaptive Path Following for Wheeled Mobile Robots. In: International Conference on Information and Automation (ICIA 2006), pp. 289–294 (December 2006)
3. Campani, M., Capezio, F., Rebora, A., Sgorbissa, A., Zaccaria, R.: A Minimalist Approach to Path Following among Unknown Obstacles. In: IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS 2010), pp. 3604–3610 (October 2010)
4. Liu, N.: Intelligent Path Following Method for Nonholonomic Robot Using Fuzzy Control. In: Second International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2009, pp. 282–285 (November 2009)
5. Fierro, R., Lewis, F.L.: Control of A Nonholonomic Mobile Robot Using Neural Networks. IEEE Transactions on Neural Networks 9(4), 589–600 (1998)
6. Pivtoraiko, M., Knepper, R.A., Kelly, A.: Differentially Constrained Mobile Robot Motion Planning in State Lattices. Journal of Field Robotics 26(3), 308–333 (2009)
7. Fjerdingen, S.A., Liljebäck, P., Transeth, A.A.: A Snake-Like Robot for Internal Inspection of Complex Pipe Structures. In: Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, St. Louis, USA, pp. 5665–5671 (October 2009)
8. Murugendran, B., Transeth, A.A., Fjerdingen, S.A.: Modeling and Path-Following for a Snake-Robot with Active Wheels. In: Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, St. Louis, USA, pp. 3643–3650 (October 2009)
9. Altafini, C.: Some properties of the general n-trailer. International Journal of Control 74(4), 409–424 (2001)

10. Mucientes, M., Casillas, J.: Quick Design of Fuzzy Controllers With Good Interpretability in Mobile Robotics. IEEE Transactions on Fuzzy Systems 15(4), 636–651 (2007)
11. Casillas, J.: Embedded genetic learning of highly interpretable fuzzy partitions. In: Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference (IFSA-EUSFLAT 2009), Lisbon, Portugal, pp. 1631–1636 (2009)
12. Open Dynamics Engine, `http://www.ode.org`