A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks

Hermann Mayer , Faustino Gomez , Daan Wierstra , Istvan Nagy , Alois Knoll & Jürgen Schmidhuber

To cite this article: Hermann Mayer , Faustino Gomez , Daan Wierstra , Istvan Nagy , Alois Knoll & Jürgen Schmidhuber (2008) A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks, Advanced Robotics, 22:13-14, 1521-1537, DOI: 10.1163/156855308X360604

To link to this article: http://dx.doi.org/10.1163/156855308X360604

Published online: 02 Apr 2012.

///VSP/// Advanced Robotics 22 (2008) 1521–1537 ADVANCED ROBOTICS www.brill.nl/ar

Full paper

A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks

Hermann Mayer [a], **Faustino Gomez** [b,*], **Daan Wierstra** [b], **Istvan Nagy** [a], **Alois Knoll** [a] **and Jürgen Schmidhuber** [a,b]

[a] Department of Embedded Systems and Robotics, Technical University Munich, 85748 Garching, Germany
[b] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), 6928 Manno-Lugano, Switzerland

Received 5 February 2008; accepted 29 April 2008

Abstract
Tying suture knots is a time-consuming task performed frequently during minimally invasive surgery (MIS). Automating this task could greatly reduce total surgery time for patients. Current solutions to this problem replay manually programmed trajectories, but a more general and robust approach is to use supervised machine learning to smooth surgeon-given training trajectories and generalize from them. Since knot tying generally requires a controller with internal memory to distinguish between identical inputs that require different actions at different points along a trajectory, it would be impossible to teach the system using traditional feedforward neural nets or support vector machines. Instead we exploit more powerful, recurrent neural networks (RNNs) with adaptive internal states. Results obtained using long short-term memory RNNs trained by the recent Evolino algorithm show that this approach can significantly increase the efficiency of suture knot tying in MIS over preprogrammed control.

Keywords
Supervised learning, recurrent neural networks, artificial evolution, minimally invasive surgery, automated knot tying

1. Introduction

Minimally invasive surgery (MIS) has become commonplace for an ever-growing number of procedures. Since MIS is performed through small incisions or ports in the patient's body, tissue trauma, recovery time and pain are reduced considerably compared to conventional 'open' surgery. While patients have profited enormously,

[*] To whom correspondence should be addressed. E-mail: tino@idsia.ch

surgeons have had to cope with reduced dexterity and perception: the instruments are long and have fewer degrees of freedom, force and tactile feedback are lost, and visual feedback is flattened to a two-dimensional image. These factors make delicate maneuvers such as knot tying very time consuming. A laparoscopically tied suture knot can take up to 3 min to complete, compared to 1 s for a manually tied knot.

Robot-assisted MIS seeks to restore the feel of normal surgery by providing the surgeon with a more intuitive and ergonomic interface. The surgeon tele-operates a slave robot that manipulates the surgical instruments from a master console that provides full 6-d.o.f. manipulation, enhanced three-dimensional (3-D) imaging and often force feedback. Robotic surgical systems such as DaVinci [1] and ZEUS [2] are in wide use today, performing a variety of abdominal, pelvic and thoracic procedures. However, despite significant advances in robot-assisted surgery, the delicate task of knot tying is still cumbersome and time consuming, in some cases taking longer than with conventional MIS [3]. Given that knot tying occurs frequently during surgery, automating this subtask would greatly reduce surgeon fatigue and total surgery time.

Building a good knot-tying controller is difficult because the 3-D trajectories of multiple instruments must be precisely controlled. There has been very little work in autonomous robotic knot tying: Kang *et al.* [4] devised a specialized stitching device, while Mayer *et al.* [5] were the first to tie a suture knot autonomously using general-purpose laparoscopic instruments. In both approaches, the controller uses a hard-wired policy, meaning that it always repeats the same prescribed motion without the possibility of generalizing to unfamiliar instrument locations. One possible way to provide more robust control is to learn the control policy from examples of correct behavior provided by the user.

The focus of this paper is on automating suture knot winding by training a recurrent neural network (RNN [6–8]) on human-generated examples. Unlike standard non-recurrent machine learning techniques such as support vector machines and feedforward neural networks, RNNs have an internal state or short-term memory which allows them to perform tasks such as knot tying where the previous states (i.e., instrument positions) need to be remembered for long periods of time in order to select future actions appropriately.

To date, the only RNN capable of using memory over sequences the length of those found in knot-tying trajectories (over 1000 datapoints) is long short-term memory (LSTM [9]). Therefore, our experiments use this powerful architecture to learn to control the movement of a real surgical manipulator to successfully tie a knot. The best results were obtained using the recent hybrid supervised/evolutionary learning framework, Evolino [10–13].

Section 2 describes the Endoscopic Partial-Autonomous Robot (EndoPAR) robotic system used in the experiments. In Section 3, we give a detailed account of the steps involved in laparoscopic knot tying. Section 4 describes the Evolino framework and in Section 5 the method is tested experimentally in the task of autonomous suture knot winding.

2. The EndoPAR System

The EndoPAR [14, 15] system is an experimental robotic surgical platform developed by the Robotics and Embedded Systems research group at the Technical University of Munich (Fig. 1). EndoPAR consists of four Mitsubishi RV-6SL robotic arms that are mounted upside-down on an aluminum gantry, providing a 20 cm × 25 cm × 40 cm workspace that is large enough for surgical procedures. Although there are four robots, it is easy to access the workspace due to the ceiling-mounted setup. Three of the arms are equipped with force-feedback instruments; the fourth holds a 3-D endoscopic stereo camera.

The position and orientation of the manipulators are controlled by two PHANToM Premium 1.5 devices from Sensable Inc. The user steers each instrument by moving a stylus pen that simulates the hand posture and feel of conventional surgical implements. The key feature of the PHANToM devices is their ability to provide force feedback to the user. EndoPAR uses a version of the PHANToM device that can display forces in all translational directions (no torque is fed back).

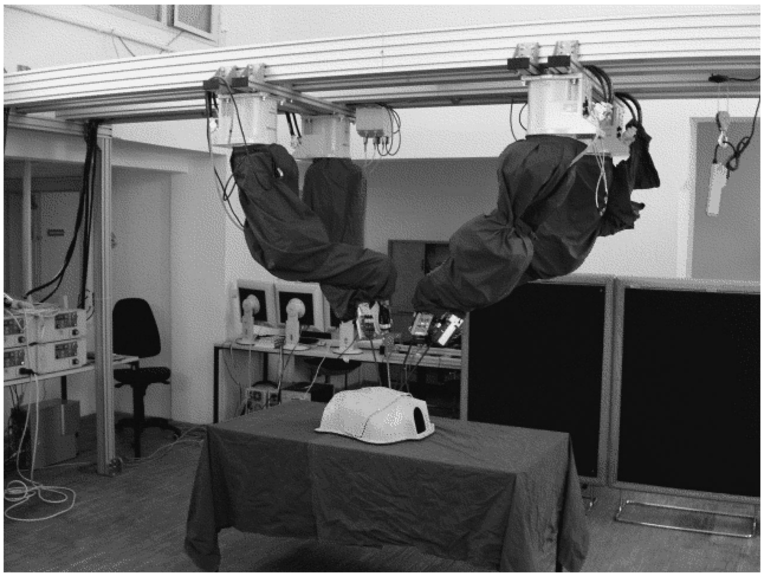

Figure 1. EndoPAR system. The four-ceiling mounted robots are shown with an artificial chest on the operating table to test tele-operated and autonomous surgical procedures. Three of the robots hold laparoscopic gripper instruments, while the fourth manipulates an endoscopic stereo camera that provides the surgeon with images from inside the operating cavity. The size of the operating area (including gantry) is approximately 2.5 m × 5.5 m × 1.5 m and the height of the operating table is approximately 1 m.

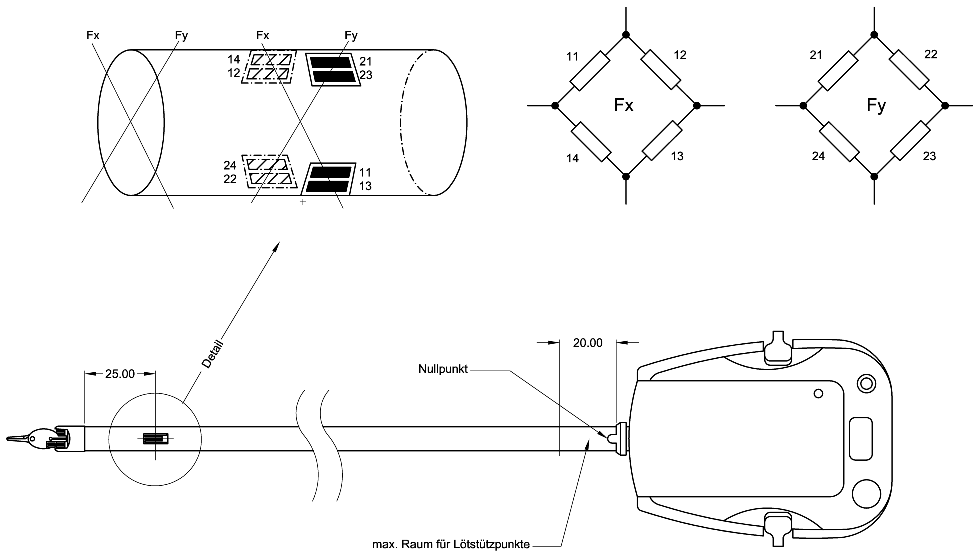

Figure 2. Force feedback. Forces are measured in the x- and y-directions (perpendicular to the shaft). The upper part shows how the strain gauge sensors are arranged along the circumference of the shaft. Each diametrically opposed pair constitutes a full bridge of four resistors dedicated to one principal axis. Sensor signals are sent back to servo motors at the input stylus so that the surgeon can sense forces occurring at the gripper.

Figure 2 shows the sensor configuration used to implement realistic force feedback in the EndoPAR system. Each instrument has four strain gauge sensors attached at the distal end of the shaft, i.e., near the gripper. The sensors are arranged in two full bridges, one for each principal axis. The signals from the sensors are amplified and transmitted *via* the CAN-bus to a PC system where they are processed and sent to small servo motors that move the stylus to convey the sensation of force to the user. Since direct sensor readings are somewhat noisy, a smoothing filter is applied in order to stabilize the results.

Force feedback makes performing MIS more comfortable, efficient, safe and precise. For knot tying, this capability is essential due to the fine control required to execute the procedure without breaking or loosing the thread [14]. As a result, the EndoPAR system provides an excellent platform with which to generate good training samples for the supervised machine learning approach explored in this paper.

3. MIS Knot Tying

Tying a suture knot laparoscopically involves coordinating the movements of three grippers through six steps. When the procedure begins, the grippers should be in the configuration depicted in Fig. 3A with the needle already having pierced the tissue (for safety, the piercing is performed manually by the surgeon). The next step (Fig. 3B) is to grasp the needle with gripper 1 and manually feed the thread to

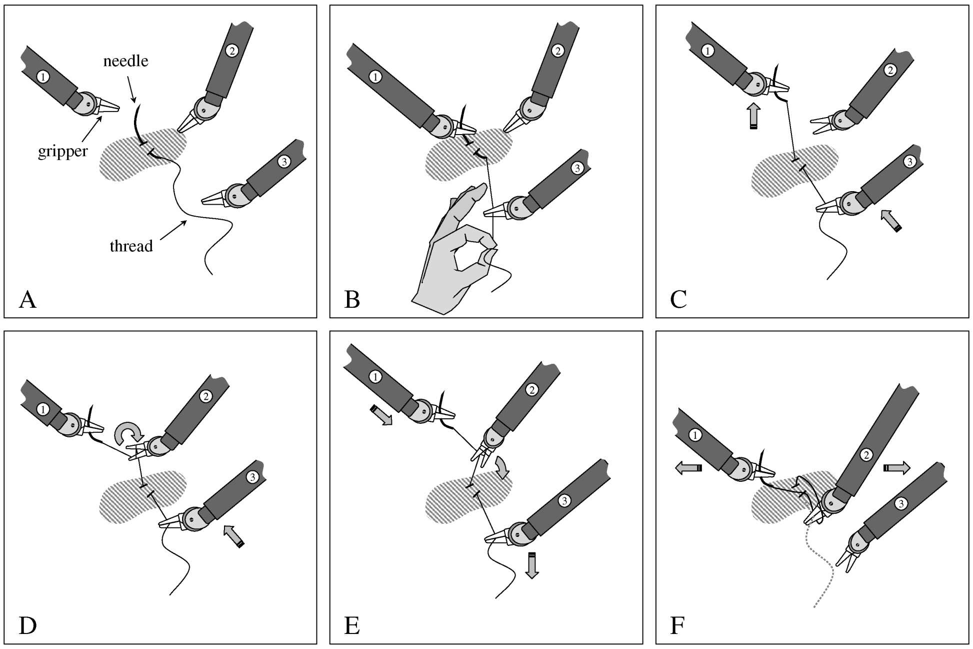

Figure 3. Minimally invasive knot tying. (A) The knot-tying procedure starts with the needle and three grippers in this configuration. (B) Gripper 1 takes the needle and the thread is fed manually to gripper 3. (C) The thread is pulled through the puncture and (D) wound around gripper 2. (E) Gripper 2 grabs the thread between the puncture and gripper 3. (F) The knot is finished by pulling the end of the thread through the loop.

gripper 3, the assistant gripper, making sure the thread is taut. Gripper 1 then pulls the thread through the puncture (Fig. 3C), while gripper 3 approaches it at the same speed so that the thread remains under tension. Meanwhile, gripper 2 is opened and moved to the position where the winding should take place.

Once gripper 2 is in position, gripper 1 makes a loop around it to produce a noose (Fig. 3D). For this step it is very important that the thread be under the right amount of tension; otherwise, the noose around gripper 2 will loosen and be lost. To maintain the desired tension, gripper 3 is moved towards the puncture to compensate for the material needed for winding. Special care must be taken to ensure that neither gripper 1 nor the needle interfere with gripper 2 or the strained thread during winding.

After completing the loop, gripper 2 can be moved to get the other end of the thread (Fig. 3E). Once again, it is critical that the thread stay under tension by having grippers 1 and 3 follow the movement of gripper 2 at an appropriate speed. In Fig. 3F, gripper 2 has grasped the end of the thread. Therefore, gripper 1 must loosen the loop such that gripper 2 can pull the thread end through the loop. Gripper 3 can now loosen its grasp, since thread tension is no longer needed. Finally, grippers 1 and 2 can pull outward (away from the puncture) in order to complete the knot.

The knot-tying procedure just described has been automated successfully by carefully programming the movement of each gripper directly [5]. Programming gripper trajectories correctly is difficult and time consuming, and, more importantly, produces behavior that is tied to specific geometric coordinates. The next section describes a method that can potentially provide a more generic solution by learning directly from human experts.

4. EVOlution of Systems With LINear Outputs (Evolino)

Evolino is a general framework for supervised sequence learning that combines neuroevolution (i.e., the evolution of neural networks) and analytical linear methods that are optimal in some sense, such as linear regression or quadratic programming. The underlying principle of Evolino is that often a linear model can account for a large number of properties of a problem. Properties that require non-linearity and recurrence are then dealt with by evolution.

The output of the network at time t, $y(t) \in \mathbb{R}^m$, is computed by the following formulas:

$$y(t) = W\phi(t) \tag{1}$$

$$\phi(t) = f(u(t), u(t-1), \ldots, u(0)), \tag{2}$$

where $\phi(t) \in \mathbb{R}^n$ is the output of a RNN $f(\cdot)$ and W is a weight matrix. Note that because the networks are recurrent, $f(\cdot)$ is indeed a function of the entire input history, $u(t), u(t-1), \ldots, u(0)$. In the case of maximum margin classification problems [16] we may compute W by quadratic programming. In what follows, however, we focus on mean squared error minimization problems and compute W by linear regression.

In order to evolve an $f(\cdot)$ that minimizes the error between y and the correct output, d, of the system being modeled, Evolino does not specify a particular evolutionary algorithm, but rather only stipulates that networks be evaluated using the following two-phase procedure.

In the first phase, a training set of sequences obtained from the system, $\{u^i, d^i\}$, $i = 1, \ldots, k$, each of length l^i, is presented to the network. For each sequence u^i, starting at time $t = 0$, each input pattern $u^i(t)$ is successively propagated through the recurrent network to produce a vector of activations $\phi^i(t)$ that is stored as a row in a $n \times \sum_i^k l^i$ matrix Φ. Associated with each $\phi^i(t)$ is a target row vector d^i in D containing the correct output values for each time step. Once all k sequences have been seen, the output weights W (the output layer in Fig. 1) are computed using linear regression from Φ to D. The column vectors in Φ (i.e., the values of each of the n outputs over the entire training set) form a non-orthogonal basis that is combined linearly by W to approximate D.

In the second phase, the training set is presented to the network again, but now the inputs are propagated through the recurrent network $f(\cdot)$ and the newly computed output connections to produce predictions $y(t)$. The error in the prediction or the residual error is then used as the fitness measure to be minimized by evolution.

Neuroevolution is normally applied to reinforcement learning tasks where correct network outputs (i.e., targets) are not known *a priori*. Here we use neuroevolution for supervised learning to circumvent the problems of gradient-based approaches. In order to obtain the precision required for time-series prediction, we do not try to evolve a network that makes predictions directly. Instead, the network outputs a set of vectors that form a basis for linear regression. The intuition is that finding a sufficiently good basis is easier than trying to find a network that models the system accurately on its own. Evolino has been shown to outperform gradient-based methods on continuous trajectory generation tasks [10]. Unlike gradient-based methods, it has the ability to escape local minima due to its evolutionary component. Moreover, it is capable of generating precise outputs by using the pseudo-inverse, which computes an optimal linear mapping. Previous work with Evolino has concentrated on comparisons with other methods in rather abstract benchmark problems, such as the Mackey–Glass time series. This paper presents the first application of Evolino to a real-world task.

In this study, Evolino is instantiated using enforced subpopulations (ESP) to evolve LSTM networks. The next sections describe ESP and LSTM, and the details of how they are combined within the Evolino framework (Fig. 4).

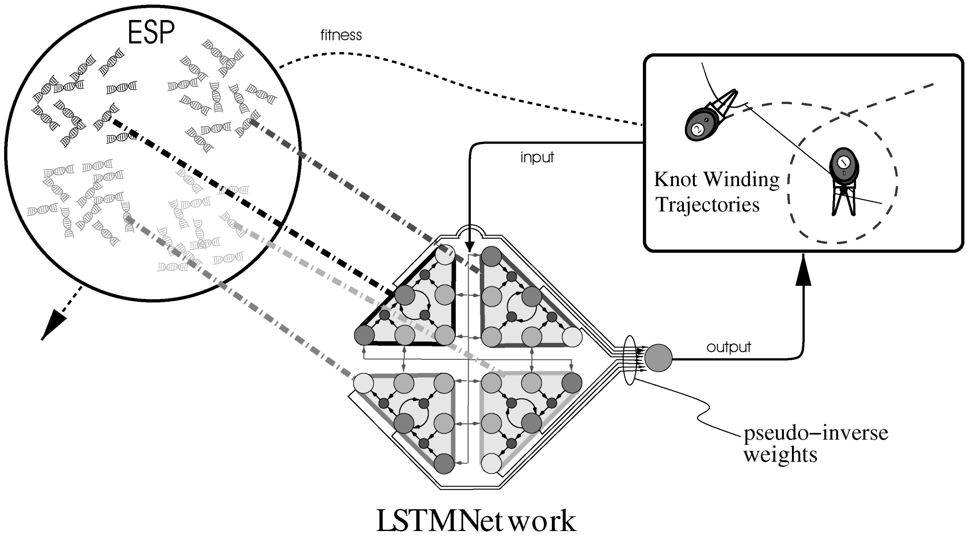

Figure 4. Evolino: three components of the Evolino implementation used in this paper: the ESP neuroevolution method, the LSTM network architecture (shown with four memory cells) and the pseudo-inverse method to compute the output weights. When a network is evaluated, it is first presented the training set to produce a sequence on network activation vectors that are used to compute the output weights using the pseudo-inverse. Then the training set is presented again, but now the activation also passes through the new connections to produce outputs. The error between the outputs and the targets is used by ESP as a fitness measure to be minimized.

4.1. ESP

ESP [17] differs from standard neuroevolution methods in that, instead of evolving complete networks, it co-evolves separate subpopulations of network components or neurons. Evolution in ESP proceeds as follows:

(i) *Initialization*. The number of hidden units H in the networks that will be evolved is specified and a subpopulation of n neuron chromosomes is created for each hidden unit. Each chromosome encodes a neuron's input, output and recurrent connection weights with a string of random real numbers.

(ii) *Evaluation*. A neuron is selected at random from each of the H subpopulations and combined to form a recurrent network. The network is evaluated on the task and awarded a fitness score. The score is added to the cumulative fitness of each neuron that participated in the network.

(iii) *Recombination*. For each subpopulation the neurons are ranked by fitness, and the top quartile is recombined using one-point crossover and mutated using Cauchy distributed noise to create new neurons that replace the lowest-ranking half of the subpopulation.

(iv) Repeat the evaluation–recombination cycle until a sufficiently fit network is found.

ESP searches the space of networks indirectly by sampling the possible networks that can be constructed from the subpopulations of neurons. Network evaluations serve to provide a fitness statistic that is used to produce better neurons that can eventually be combined to form a successful network. This cooperative coevolutionary approach is an extension to symbiotic, adaptive neuroevolution (SANE [18]), which also evolves neurons, but in a single population. By using separate subpopulations, ESP accelerates the specialization of neurons into different subfunctions needed to form good networks because members of different evolving subfunction types are prevented from mating. Subpopulations also reduce noise in the neuron fitness measure because each evolving neuron type is guaranteed to be represented in every network that is formed. This allows ESP to evolve recurrent networks, where SANE could not.

If the performance of ESP does not improve for a predetermined number of generations, a technique called burst mutation is used. The idea of burst mutation is to search the space of modifications to the best solution found so far. When burst mutation is activated, the best neuron in each subpopulation is saved, the other neurons are deleted and new neurons are created for each subpopulation by adding Cauchy distributed noise to its saved neuron. Evolution then resumes, but now searching in a neighborhood around the previous best solution. Burst mutation injects new diversity into the subpopulations and allows ESP to continue evolving after the initial subpopulations have converged.

4.2. LSTM

RNNs are a powerful class of models that can, in principle, approximate any dynamic system [19]. This means that RNNs can be used to implement arbitrary sequence-to-sequence mappings that require memory. However, training RNNs with standard gradient descent techniques is only practical when a short time window (less than 10 time steps) suffices to predict the correct system output. For longer time dependencies, the gradient vanishes as the error signal is propagated back through time so that network weights are never adjusted correctly to account for events far in the past [20].

LSTM [9, 21, 22] overcomes this problem by using specialized, linear memory cells that can maintain their activation indefinitely. Memory cells consist of a linear unit that holds the state of the cell and three gates that can open or close over time. The input gate 'protects' a neuron from its input: only when the gate is open can inputs affect the internal state of the neuron. The output gate lets the state out to other parts of the network and the forget gate enables the state to 'leak' activity when it is no longer useful.

The state of cell i is computed by:

$$s_i(t) = \mathrm{net}_i(t) g_i^{\mathrm{in}}(t) + g_i^{\mathrm{forget}}(t) s_i(t-1), \tag{3}$$

where g^{in} and g^{forget} are the activation of the input and forget gates, respectively, and net is the weighted sum of the external inputs:

$$\mathrm{net}_i(t) = h\left(\sum_j w_{ij}^{\mathrm{cell}} c_j(t-1) + \sum_k w_{ik}^{\mathrm{cell}} u_k(t)\right), \tag{4}$$

where h is usually the identity function and c_j is the output of cell j:

$$c_j(t) = \tanh(g_j^{\mathrm{out}}(t) s_j(t)), \tag{5}$$

where g^{out} is the output gate of cell j. The amount each gate g_i of memory cell i is open or closed at time t is calculated by:

$$g_i^{\mathrm{type}}(t) = \sigma\left(\sum_j w_{ij}^{\mathrm{type}} c_j(t-1) + \sum_k w_{ik}^{\mathrm{type}} u_k(t)\right), \tag{6}$$

where type can be input, output or forget and σ is the standard sigmoid function. The gates receive input from the output of other cells c_j and from the external inputs to the network.

4.3. Combining ESP With LSTM in Evolino

We apply our general Evolino framework to the LSTM architecture, using ESP for evolution and regression for computing linear mappings from hidden state to outputs. ESP coevolves subpopulations of memory cells instead of standard recurrent neurons (Fig. 4). Each chromosome is a string containing the external input weights, and the input, output and forget gate weights, for a total of $4 * (I + H)$ weights in each memory cell chromosome, where I is the number of external inputs and H is the number of memory cells in the network. There are four sets of $I + H$ weights because the three gates (6) and the cell itself (4) receive input from outside the cell and the other cells. ESP, as described in Section 4.1, normally uses crossover to recombine neurons. However, for the present Evolino variant, where fine local search is desirable, ESP uses only mutation. The top quarter of the chromosomes in each subpopulation are duplicated and the copies are mutated by adding Cauchy noise to all of their weight values.

The linear regression method used to compute the output weights (W in 2) is the Moore–Penrose pseudo-inverse method, which is both fast and optimal in the sense that it minimizes the summed squared error [23] (compare [24] for an application to feedforward RBF nets). The vector $\phi(t)$ consists of both the cell outputs, c_i (5), and their internal states, s_i (3), so that the pseudo-inverse computes two connection weights for each memory cell. We refer to the connections from internal states to the output units as 'output peephole' connections, since they peer into the interior of the cells.

For continuous function generation, backprojection (or teacher forcing in standard RNN terminology) is used where the predicted outputs are fed back as inputs in the next time step: $\phi(t) = f(u(t), y(t-1), u(t-1), \ldots, y(0), u(0))$.

During training, the correct target values are backprojected, in effect 'clamping' the network's outputs to the right values. During testing, the network backprojects its own predictions. This technique is also used by echo state networks (ESNs), but whereas ESNs do not change the backprojection connection weights, Evolino evolves them, treating them like any other input to the network. In the experiments described below, backprojection was found useful for continuous function generation tasks, but interferes to some extent with performance in the discrete context-sensitive language task.

5. Experiments in Robotic Knot Winding

Our initial experiments focus on the most critical part of suture knot tying: winding the suture loop (steps C–F in Fig. 3). While the loop is being wound by gripper 1, gripper 2 stays fixed. Therefore, networks were trained to control the movement of gripper 1.

5.1. Experimental Setup

LSTM networks were trained using a database of 25 loop trajectories generated by recording the movement of gripper 1 while a knot was being tied successfully using the PHANToM units. Each trajectory consisted of approximately 1300 gripper (x, y, z)-positions measured at every 0.1-mm displacement, $\{(x_1^j, y_1^j, z_1^j), \ldots, (x_{l_j}^j, y_{l_j}^j, z_{l_j}^j)\}$, $j = 1, \ldots, 25$, where l_j is the length of sequence j. At each step in a training sequence, the network receives the coordinates of gripper 1 through three input units (plus a bias unit) and computes the desired displacement (Δx, Δy, Δz) from the previous position through three output units.

Both gradient descent and Evolino were used in 20 experiments each to train LSTM networks with 10 memory cells. The Evolino-based networks were evolved for 60 generations with a population size of 40, yielding a total of 3580 evaluations (i.e., passes through the training set) for each experiment.

Figure 5 illustrates the procedure for training the networks. For the gradient descent approach, the LSTM networks were trained using Backpropagation Through Time [6] where the network is unfolded once for each element in the training sequence to form an l_j-layer network (for sequence j) with all layers sharing the same weights. Once the network has seen that last element in the sequence, the errors from each time step are propagated back through each layer as in standard backpropagation and then the weights are adjusted.

For Evolino-trained LSTM, each network is evaluated in two phases (see Section 4). In the first phase the activations of the network units are recorded, but no outputs are produced as, at this point, the network does not have output connections. After the entire training set has been seen, the output connections are computed using the pseudo-inverse. In the second phase, the network produces control actions that are used to calculate the fitness of the network.

The error (fitness) measure used for both methods was the sum-squared difference between the network output plus the previous gripper position and the correct (target) position for each time step, across the 25 trajectories:

$$\sum_{j=1}^{25}\sum_{t=1}^{l_j-1}(x_t^j + \Delta x_t - x_{t+1}^j)^2 + (y_t^j + \Delta y_t - y_{t+1}^j)^2 + (z_t^j + \Delta z_t - x_{t+1}^j)^2,$$

where Δx_t, Δy_t and Δz_t are the network outputs for each principal axis at time t which are added to the current position (x_t, y_t, z_t) to obtain the next position. Note that because the networks are recurrent, the output can in general depend on all of the previous inputs.

For the first 50 time steps, each training sequence is a straight line in the x-direction and the network receives the corresponding sequence entry. During this period, the network outputs are not fed back, so that the arbitrary initial state is discarded or ‘washed out’ of the network. After this washout time, the current network output and the previous input are fed back as the input for the next time step, and the network then makes predictions based on previous predictions, i.e., having no access to the training set to steer it back on course. This procedure allows the error to accumulate along the entire trajectory, so that minimizing it forces the network to produce loop trajectories autonomously, i.e., no ‘teacher’ signal.

Once a network has learned the training set, it is tested in a 3-D simulation environment to verify that the trajectories do not behave erratically or cause collisions. If the network passes this validation, it is transferred to the real robot where the procedure is executed inside the artificial rib cage and heart mockup shown in Fig. 1.

To tie the entire knot, a preprogrammed controller is used to start the knot and then the network takes over for the loop, steps C–E. During the loop, the robot fetches a new displacement vector from the network every 7 ms and adds it to the current position of gripper 1. Gripper 2 remains stationary throughout this phase and gripper 3 is moved away from the knot at a predefined rate to maintain tension on the thread. When the loop is complete, the control switches back to the program to close the knot. As in training, the winding network receives an initial 'approaching sequence' of 50 points that control the robot to start the wind and then completes the loop itself while feeding back its own outputs.

5.2. Experimental Results

Figure 6 shows the learning curve for the Evolino-trained LSTM networks. Each datapoint is the average error on the training set of the best network, measured in millimeters. By generation 40, the error has reached a level that the networks can effectively produce usable loop trajectories. The gradient-trained LSTM networks were not able to learn the trajectories, so the error for this method is not reported. This poor performance could be due to the presence of many local minima in the error surface which can trap gradient-based methods.

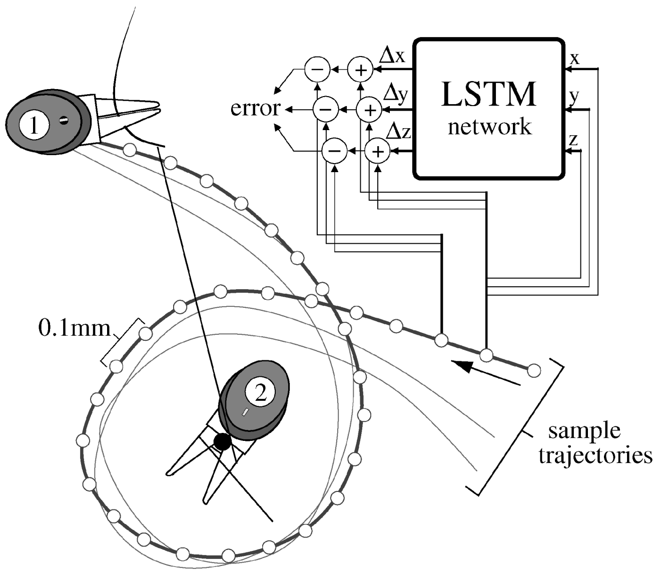

Figure 5. Training the knot-winding networks. LSTM networks are trained on a set of recordings that sample the position of gripper 1 at 0.1-mm increments during a human-controlled suture knot. The figure shows three such training sequences; the one with the thicker path shows the sample points that the network uses as input and targets. For each training sequence, the network receives the (x, y, z)-position of the gripper 1 and outputs a prediction of the distance to the next position of the gripper (i.e., the next sample in the sequence). The prediction is added to the input and compared to the correct (target) next position to produce an error signal that is used either for gradient descent learning, or as a fitness measure for Evolino, after all the training sequences have been processed.

Unlike gradient-based approaches, Evolino is an evolutionary method and, therefore, is less susceptible to local minima. All of the 20 Evolino runs produced networks that could generate smooth loop trajectories. When tested on the real robot, the networks reliably completed the loop procedure, and did so in an average of 3.4 s, a speed-up of almost 4 times over the preprogrammed loop. This speed-up in knot-winding results in a total time of 25.8 s for the entire knot compared to 33.7 s for the preprogrammed controller.

Figure 7 shows the behavior of several Evolino-trained LSTM networks from the same run at different stages of evolution. As evolution progresses, the controllers track the training trajectories more closely while smoothing them. The network in the right-hand side of Fig. 7 was produced after approximately 4.5 h of computation time (Intel Xeon CPU 2.80 GHz).

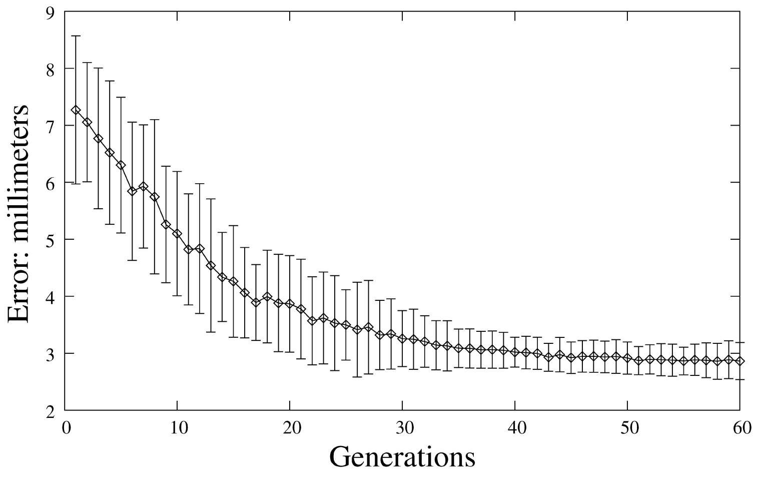

Figure 6. Evolino learning curve. The plot shows the average error on the training set measured in millimeters for the best network in each generation, averaged over 50 runs. The vertical bars indicate one standard deviation from the average.

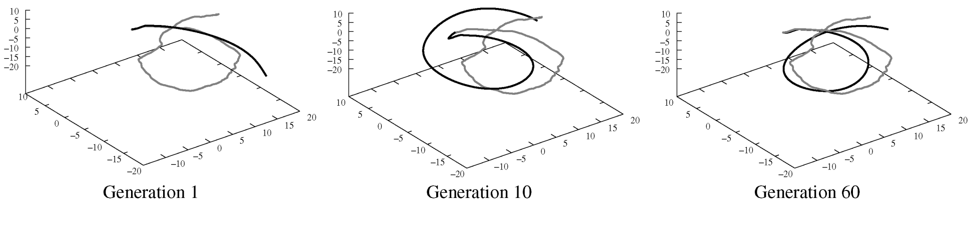

Figure 7. Evolution of loop-generating behavior. Each of the three 3-D plots shows the behavior of the best network at a different generation during the same evolutionary run. All axes are in millimeters. The dark curve is the trajectory generated by the network; the lighter curve is the target trajectory. Note that the network is being tested to reproduce the same target trajectory in each plot. The best network in the first generation tracks the target trajectory closely for the first 15-mm or so, but diverges quickly when the target turns abruptly. By generation 10, networks can form smooth loops, and by generation 60, the network tracks the target throughout the winding, forming a tight, clean loop.

To measure generalization, the evolved network was tested on new initial trajectories not seen during training. Just as in training, the network was presented with straight trajectories lasting the duration of the 50-step washout time, but now rotated up to $\pm 30^\circ$ about the x- and z-axes. Twenty such tests were performed on the physical robot where the network was able to produce clean loops for rotations of up to $\pm 20^\circ$ in both axes.

These first results show that RNNs can be used to learn from training sequences of over 1000 steps and possibly provide useful assistance to expedite MIS procedures.

6. Discussion and Future Work

An important advantage of learning directly from expert behavior is that it requires less knowledge about the system being controlled. Supervised machine learning can be used to capture and generalize expertise without requiring the often tedious and costly process of traditional controller design. The Evolino-trained LSTM networks in our experiments were able to learn from surgeons and outperform them on the real robot.

Our current approach only deals with the winding portion of the knot-tying task. Therefore, its contribution is limited by the efficiency of the other subtasks required to complete the full knot. In the future, we plan to apply the same basic approach used in this paper to other knot-tying subtasks (e.g., the thread tensioning performed by the assistant gripper and knot tightening) that are currently implemented by programmed controllers. The separate subcontrollers can then be used in sequence to execute the whole procedure.

The performance of automated MIS need not be constrained by the proficiency of available experts. While human surgeons provide the best existing control, more optimal strategies may be possible by employing reinforcement learning techniques where target trajectories are not provided, but instead some higher-level measure of performance is maximized. Approaches such as neuroevolution could be used alone or in conjunction with supervised learning to bootstrap the learning. Such an approach would first require building a simulation environment that accurately models thread physics.

7. Conclusions

This paper has explored the application of supervised machine learning to the important task of automated knot tying in MIS. LSTM neural networks were trained to produce knot-winding trajectories for a robotic surgical manipulator, based on human-generated examples of correct behavior. Initial results using the Evolino framework to train the networks are promising: the networks were able to perform the task on the real robot without access to the teaching examples. These results constitute the first successful application of supervised learning to MIS knot tying.

Acknowledgements

This research was partially funded by SNF grant 200020-107534 and the EU MindRaces project FP6 511931.

References

1. G. Guthart and J. K. Salisbury Jr, The Intuitive™ telesurgery system: overview and application, in: *Proc. Int. Conf. on Robotics and Automation*, San Francisco, CA, pp. 618–621 (2000).
2. A. Garcia-Ruiz, N. G. Smedira, F. D. Loop, J. F. Hahn, C. P. Steiner, J. H. Miller and M. Gagner, Robotic surgical instruments for dexterity enhancement in thorascopic coronary artery bypass graft, *J. Laparoendoscop. Adv. Surg. Tech.* **7**, 277–283 (1997).
3. A. Garcia-Ruiz, Manual *vs* robotically assisted laparoscopic surgery in the performance of basic manipulation and suturing tasks, *Arch. Surg.* **133**, 957–961 (1998).
4. H. Kang, Robotic assisted suturing in minimally invasive surgery, *PhD Thesis*, Rensselaer Polytechnic Institute, Troy, NY (2002).
5. H. Mayer, I. Nagy, A. Knoll, E. U. Schirmbeck and R. Bauernschmitt, The EndoPAR system for minimally invasive surgery, in: *Proc. Int. Conf. on Intelligent Robots and Systems*, Sendai, pp. 3637–3642 (2004).
6. P. Werbos, Backpropagation through time: what does it do and how to do it, *Proc. IEEE* **78**, 1550–1560 (1990).
7. A. J. Robinson and F. Fallside, The utility driven dynamic error propagation network, *Technical Report CUED/F-INFENG/TR.1*, Engineering Department, Cambridge University (1987).
8. R. J. Williams and D. Zipser, A learning algorithm for continually running fully recurrent networks, *Neural Comput.* **1**, 270–280 (1989).
9. S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9**, 1735–1780 (1997).
10. J. Schmidhuber, D. Wierstra and F. Gomez, Evolino: hybrid neuroevolution/optimal linear search for sequence learning, in: *Proc. 19th Int. Joint Conf. on Artificial Intelligence*, Edinburgh, pp. 853–858 (2005).
11. D. Wierstra, F. Gomez and J. Schmidhuber, Modeling systems with internal state using Evolino, in: *Proc. Genetic Evolutionary Computation Conf.*, New York, NY, pp. 1795–1802 (2005).
12. H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll and J. Schmidhuber, A system for robotic heart surgery that learns to tie knots using recurrent neural networks, in: *Proc. Int. Conf. on Intelligent Robotics and Systems*, Beijing, pp. 543–548 (2006).
13. J. Schmidhuber, D. Wierstra, M. Gagliolo and F. Gomez, Training recurrent neural networks by Evolino, *Neural Comput.* **19**, 757–779 (2007).
14. I. Nagy, H. Mayer, A. Knoll, E. U. Schirmbeck and R. Bauernschmitt, EndoPAR: an open evaluation system for minimally invasive robotic surgery, in: *Proc. IEEE Conf. on Mechatronics and Robotics*, Aachen, pp. 1464–1467 (2004).
15. H. Mayer, I. Nagy, A. Knoll, E. U. Braun, R. Lange and R. Bauernschmitt, Adaptive control for human-robot skilltransfer: trajectory planning based on fluid dynamics, in: *Proc. IEEE Int. Conf. on Robotics and Automation*, Rome, pp. 1800–1807 (2007).
16. V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, New York, NY (1995).
17. F. J. Gomez, Robust nonlinear control through neuroevolution, *PhD Thesis (Technical Report AI-TR-03-303)*, Department of Computer Sciences, University of Texas at Austin, Austin, TX (2003).

18. D. E. Moriarty and R. Miikkulainen, Efficient reinforcement learning through symbiotic evolution, *Mach. Learn.* **22**, 11–32 (1996).
19. H. T. Siegelmann and E. D. Sontag, Turing computability with neural nets, *Appl. Math. Lett.* **4**, 77–80 (1991).
20. S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, in: *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen (Eds), pp. 237–244. IEEE Press, New York, NY (2001).
21. F. A. Gers, J. Schmidhuber and F. Cummins, Learning to forget: continual prediction with LSTM, *Neural Comput.* **12**, 2451–2471 (2000).
22. F. A. Gers and J. Schmidhuber, LSTM recurrent networks learn simple context free and context sensitive languages, *IEEE Trans. Neural Netw.* **12**, 1333–1340 (2001).
23. R. Penrose, A generalized inverse for matrices, in: *Proc. Cambridge Philos. Soc.* **51**, 406–413 (1955).
24. E. P. Maillard and D. Gueriot, RBF neural network, basis functions and genetic algorithms, in: *Proc. IEEE Int. Conf. on Neural Networks*, Piscataway, NJ, pp. 2187–2190 (1997).

About the Authors

Daan Wierstra graduated from Utrecht University and is currently a PhD student at IDSIA. His research has focused on policy gradient and evolutionary strategy-based methods for reinforcement learning in non-Markovian environments, and for long-term dependency sequence processing.

Faustino Gomez received his PhD in Computer Science from the University of Texas at Austin, USA, in 2003. Shortly after, he held a Post-doctoral research position (2004–2006) at IDSIA, where he is now a Senior Researcher. His areas of expertise include evolutionary computation, artificial neural networks and reinforcement learning. He is also a Lecturer at the University of Lugano in the Intelligent Systems Masters program.

Hermann Mayer received his Diploma degree in Computer Science from the Technical University Munich, in 2003, and the PhD, in 2008. He has been working on the automation of surgical procedures and the employment of robots for medical applications. Currently he is working as a Research Assistant at the Robotics and Embedded Systems lab.

Istvan Nagy received both his Diploma degree in Computer Science and his PhD from the Technical University Munich. His main research topic was medical image processing in stereo-endoscopic views in the context of surgical robotics. Currently he is working for BMW on camera-based driver assistance systems.

Jürgen Schmidhuber is Professor of Cognitive Robotics at TU Munich (since 2004), Co-Director of the Swiss Institute for Artificial Intelligence IDSIA (since 1995), Professor at SUPSI (since 2003) and also adjunct Professor of Computer Science at the University of Lugano (since 2006). He obtained his Doctoral degree in Computer Science from TUM, in 1991, and his Habilitation degree, in 1993, after a Post-doctoral stay at the University of Colorado, Boulder, USA. He helped to transform IDSIA into one of the world's top 10 AI labs (the smallest!), according to the ranking of *Business Week* magazine. His numerous research grants have yielded more than 200 peer-reviewed scientific papers on topics ranging from machine learning and mathematically optimal universal AI and artificial recurrent neural networks to adaptive robotics and complexity theory, digital physics and the fine arts. He frequently gets invited to give keynote addresses at international conferences.

Alois Knoll is the Director of the research group 'Robotics and Embedded Systems'. From 2001 to 2005, he was a Member of the Board of Directors of the Fraunhofer-Institute for Autonomous Intelligent Systems, St Augustin, Bonn. His research interests include sensor-based robotics, multi-agent systems, data fusion, adaptive systems and multimedia information retrieval. In these fields, he has published over 200 technical papers and guest-edited international journals. He has been part of (and has coordinated) several large-scale national collaborative research projects. He initiated and was the Program Chairman of the First IEEE/RAS Conference on Humanoid Robots (IEEE–RAS/RSJ Humanoids 2000), he was General Chair of IEEE Humanoids 2003 and General Chair of Robotik 2008, the largest German conference on robotics. He is a member of the German Society for Computer Science (Gesellschaft für Informatik) and the IEEE.