Robotics and Autonomous Systems 91 (2017) 234–246

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

A model-based approach to robot kinematics and control using discrete factor graphs with belief propagation

Indar Sugiarto [a,b,*], Jörg Conradt [a]

[a] Neuroscientific System Theory, Technische Universität München, Germany
[b] Department of Electrical Engineering, Petra Christian University, Indonesia

HIGHLIGHTS

- Neurally inspired factor graphs for generic kinematic modeling of robot platforms.
- Performance evaluation on a real robot arm and an omnidirectional mobile robot.
- Demonstration of a dynamic factor graph for imitation learning.

ARTICLE INFO

Article history:
Received 17 March 2016
Received in revised form 22 November 2016
Accepted 4 January 2017
Available online 4 February 2017

Keywords:
Model-based approach
Factor graph
Robot kinematics

ABSTRACT

Much of recent researches in robotics have shifted the focus from traditionally-specific industrial tasks to investigations of new types of robots with alternative ways of controlling them. In this paper, we describe the development of a generic method based on factor graphs to model robot kinematics. We focused on the kinematics aspect of robot control because it provides a fast and systematic solution for the robot agent to move in a dynamic environment. We developed neurally-inspired factor graph models that can be applied on two different robotic systems: a mobile platform and a robotic arm. We also demonstrated that we can extend the static model of the robotic arm into a dynamic model useful for imitating natural movements of a human hand. We tested our methods in a simulation environment as well as in scenarios involving real robots. The experimental results proved the flexibility of our proposed methods in terms of remodeling and learning, which enabled the modeled robot to perform reliably during the execution of given tasks.

1. Introduction

Historically, robot engineering and control inherits two aspects from physics: kinematics and dynamics. The dynamic aspect is concerned with the robot motion as a function of the applied forces and torques, whereas the kinematic aspect is concerned with the relation between the overall movement and the structure of robotic system. Both aspects play important but different roles for motion planning and coordination [1–3].

In this paper, we focused on the kinematic aspect in order to evaluate the applicability of higher-level robot control and motion planning. The kinematic approach provides a fast and systematic solution for the robot agent to move in an environment with many objects; hence, it is favorable in an obstacle avoidance scenario [4–7]. It can also be extended into a dynamic modeling because once we have solved the kinematic tasks, we can construct continuous joint paths that can be followed by a dynamic controller as its input reference.

Conventional methods in kinematics usually involve some fine-tuning of parameters. Especially in a simple robot system with frequently changing tasks, this procedure will be cumbersome. To address this issue, we propose to use a model-based learning. Model-based approach attempts to model the structure of the system, and then based on that model, the learning mechanism chooses the most appropriate parameter based on the observed data. The idea is to teach the robot to generate motions based on human experience. Humans can easily change from task to task without too much effort, revealing that they have a long history of adapting model-based learning features. Using this perspective, future autonomous robots can be programmed with cognitive capabilities based on the models that rely reliably on the information perceived by the robot [8].

There is much interest in biologically inspired robotics because almost all biological entities can move robustly and adaptively in dynamic environments [8–10]. The complex motion of biological entities such as humans is composed of many small coordinated

* Correspondence to: Department of Electrical Engineering, Petra Christian University, Jl. Siwalankerto 121-131, Surabaya, 60236 Indonesia
E-mail addresses: indi@petra.ac.id (I. Sugiarto), conradt@tum.de (J. Conradt).

http://dx.doi.org/10.1016/j.robot.2017.01.003
0921-8890/

motion primitives [11–13], which are governed by motor pattern and sensorimotor generator [14]. It is also believed that the brain produces the optimal policy to generate proper actions from the current states based on its internal model, and uses the model-based learning paradigm to maintain that model in a robust manner [15–17].

We followed the model-based approach and developed a robot control strategy based on a learned model from experimental data. Our model-based approach toward robot control can be optimized for applications that involve a custom model for each new robot configuration. Two types of basic robotic systems were investigated: a mobile robot platform and a robotic arm (manipulator) platform. We strive to find a unified control strategy based on a generic model for those robots in order to make it practical to be embedded into a higher level control system with cognitive capability.

We propose to use factor graphs for implementing our approaches and we demonstrate that they can be extended into complex settings for future autonomous robotic applications. A Factor graph could be an excellent tool for processing information on different levels of abstraction: from low-level sensory processing to reasoning with high-level goals based on some cognitive architectures [18–22]. This is possible due to factor graph's capability in unifying inference mechanisms of both directed and undirected graphical models that have been developed in various systems. Using this framework, we could formulate the central problem in robotics as a coherence reasoning task of perception, decision-making, planning, and control. In this direction, the contribution of this paper can be summarized as follows.

1. We develop neurally inspired factor graphs with a generic structure for kinematic modeling of different robot platforms.
2. We present an extension of a tree-structure factor graph into a chain-structure factor graph that exemplifies the generality of a factor graph for modeling a kinematic chain.
3. We demonstrate that our dynamic factor graph can generalize sequential motion useful for imitating learning. Although the imitation learning is not our main contribution, in this paper we demonstrate that our model has a generality feature that can be used conveniently to accommodate imitation learning in a system that is built based on a graphical model.

This paper is organized as follows. In Section 2, we review the robot kinematic principle and its common implementations. In addition, we briefly review our factor graph framework that utilizes population coding principles. Next, we introduce our model in Section 3 along with the motivation for using a model-based approach. Section 4 describes our experimental setup for harnessing our model in a real robotic scenario. We provide the evaluation of our model, followed by a thorough discussion of the obtained experimental results in Section 5. Finally, Section 6 summarizes the work and provides a glimpse of possible future extensions.

2. Review of robot kinematics modeling, factor graphs, and related works

Kinematics has been used for decades by robot engineers working in industry as well as scientists developing new ideas based on cognitive intelligence. In this spectrum, the study of kinematics can be classified into two aspects: analytical kinematics that uses well-defined formulas to solve kinematic problems, and computational kinematics that uses learning paradigms in a practical methodology to find an approximation of the optimal solution. In this section, we provide a review on these aspects and how we extend them into a model-based approach. Also, the basic concept of factor graphs that will be used for kinematic modeling is presented.

2.1. Solutions to kinematics

In the most basic form, kinematics involves mapping from the object's internal states to its pose in a task space, which can be written as the following expression:

$$\mathbf{x} = \mathcal{F}(\mathbf{q}) \tag{1}$$

where $\mathcal{F}$ is a mapping function whose parameters are assumed to be known in order to relate the robot's state ($\mathbf{q}$) to the robot's pose in a task space ($\mathbf{x}$). The state $\mathbf{q}$ is a vector of joint variables; in a manipulator, it corresponds to the joints between links, while in a holonomic wheeled mobile robot, it corresponds to the angular pose of the wheels. The task variable $\mathbf{x}$ is the pose of the end-effector in a manipulator, or the pose of the mobile robot's body in its own coordinate system. However, the expression (1) might not be directly applicable in all situation, for example in the case of non-holonomic robot.

In addition to (1), we need to define the robot action in terms of its velocity, simply expressed as: $\dot{x} = \frac{\partial \mathcal{F}}{\partial q}\dot{q}$. It can be expressed in terms of a Jacobian matrix as:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}. \tag{2}$$

In order to use it in a control scenario, the inverse kinematics control problem is formulated, where a joint space trajectory $\mathbf{q}(t)$ is computed such that $\mathcal{F}(\mathbf{q}(t)) = \mathbf{x}(t)$ is satisfied given a trajectory in the task space $\mathbf{x}(t)$. The simplest solution of this inverse kinematics problem involves the joint velocities as a product of a pseudoinverse of a Jacobian matrix as:

$$\dot{\mathbf{q}} = \tilde{\mathbf{J}}(\mathbf{q})\dot{\mathbf{x}} \tag{3}$$

where $\tilde{\mathbf{J}}$ is the pseudoinverse of a Jacobian matrix. Readers who are interested more on this Jacobian-based solution are referred to [6,23].

For certain robots, the Jacobian matrix may be no longer a square matrix. Thus, it cannot be directly inverted and will require the computation of a pseudo Jacobian matrix, which is computationally expensive and is subject to numerical instabilities.

Even though Jacobian-based approaches can produce excellent performance, such analytical approaches may be impractical in some situation. For example, in our robotic setup where the main controller is a simple low-power microcontroller (see Fig. 1), the analytical approach will be unjustified due to its high computational cost. Hence, we developed hardware-friendly approaches that utilize some learning paradigms. The kinematic solution learned by the robot agent, can be used further in a more complex setting in a mimetic fashion.

2.2. From kinematics to model-based approach

The "computational" kinematics attempts to solve the problems based on an approximation paradigm applied to available robot data. It covers many methods developed in the domain of computational intelligence, such as neural networks [24–26], fuzzy logic [27,28], heuristic optimization [29–31], and probabilistic reasoning [32,33].

In this paper, we present an appealing approach of learning kinematic models based on a probabilistic reasoning with intuitive graphical embodiment. During learning phases, the robot agents were fed with reliable data that represent possible robot configurations in the solution space. We extended this basic learning mechanism into a model-based approach which requires a predefined model before the model can deliberately learn the optimal policy, integrates the learning on the basis of past experiences and the planning for future actions [34].

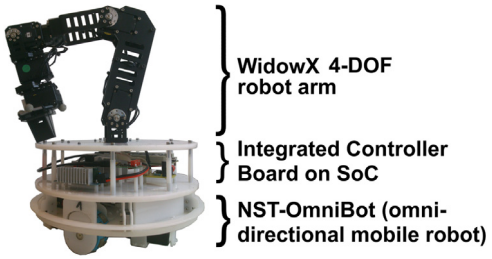
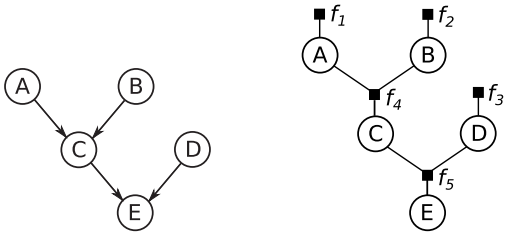


Fig. 1. The mobile manipulator used in our work. The omnidirectional mobile platform is driven by a microcontroller LPC2103 (single core and no floating point unit) running at 64 MHz, whereas the 4-DOF robot arm is driven by a low cost SoC (Zynq-XC702 from Xilinx) running at 100 MHz.



(a) Bayesian network representation. (b) Factor graph representation.

Fig. 3. Transforming a Bayesian network into a factor graph. Here the factor graph represents a structured factorization of the conditional probability.

In our work, we are interested in exploring the model-based approach because it offers several major advantages, including the opportunity to create highly tailored models for specific scenarios and the rapid prototyping and comparison of many alternative models. We developed a generic factor graph-based kinematic model that uses data-driven learning approach for two different robot platforms: an omni-directional mobile robot and a robotic arm (i.e., a manipulator). These two robot platforms are integrated to produce a mobile manipulator (see Fig. 1).

The kinematic models of these robots serve as the low level control of individual motions known as motion primitives. Different methods for learning motion primitives have been developed by many other researchers using algorithms such as Gaussian Mixture Models and Regression (GMM/GMR) [35], Hidden Markov Model (HMM) [36], Locally Weighted Projection Regression (LWPR) [37], and Dynamic Movement Primitives (DMP) [34]. In our work, we turned our attention to the paradigm of programming-by-demonstration (PbD) which is commonly known as "imitation learning". In this paradigm, the robot is expected to develop and improve its skill by exploiting statistical regularities across multiple demonstrations provided by the human teacher. Fig. 2 shows the basic principle of this paradigm.

The most intriguing challenge in this paradigm is how the model generalizes the skill so that it can also be applied in different contexts [37,38]. Skill learning itself can be developed either at a symbolic level or a trajectory level. Here we are interested in the trajectory-based approach because it allows us to concentrate on a lower level control without having too much distraction from higher level skill development problems. In this paper, we used a regression technique similar to the GMR [35], but we created the model entirely in a factor graph and used its inference mechanism to learn the trajectory.

2.3. Discrete factor graph framework

A factor graph is a probabilistic graph that unifies the directed and undirected model, and provides a convenient way for performing inference in order to compute the marginal probabilities of variables involved in the graph. Fig. 3(b) shows an example of a factor graph that originates from a directed graph (Fig. 3(a)). In such a transformation, a factor node in the factor graph captures the statistical relationship between variables such as their joint probability and/or conditional probability.

A factor node f_j is connected to the variable node X_i if and only if X_i is the argument of f_j. Given a factor graph $\mathbf{G} = (\mathbf{X}, \mathbf{F})$, the joint probability of all variables is a product of all factorization by factor nodes in the graph:

$$p(\mathbf{X}) = \frac{1}{Z} \prod_j f_j(X_i) \tag{4}$$

where $Z = \sum_X \prod_i f_i(X_i)$ is a partition function that normalize probability distribution.

The inference task in a factor graph can be performed using a message-passing mechanism which is called belief propagation. In a belief propagation, each node in the graph generates a message that will be updated consecutively from the previous value of the neighboring messages [39]. The messages are passed along the edges of the graph according to a set of message-passing rules. Two types of messages are transmitted within the factor graph: the message sent by a variable node to a factor node (denoted as $\mu_{x\to f}(x)$) and the message sent by a factor node to a variable node (denoted as $\mu_{f\to x}(x)$). These messages are computed according to the following equations:

$$\mu_{x\to f}(x) = \prod_{h\in n(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{5}$$

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} \left(f(x) \prod_{y\in n(f)\setminus\{x\}} \mu_{y\to f}(y) \right) \tag{6}$$

where $\sum_{\sim\{x\}}$ is the "not-sum" or *summary* indicating that the marginal probability over a set of variables is calculated without including the specific variable x in that set. For example using Fig. 3(b), since f_5 is a function of three variables C, D, and E, then the "summary for C" is denoted by

$$\sum_{\sim\{C\}} f_5(C, D, E) = \sum_D \sum_E f_5(C, D, E).$$

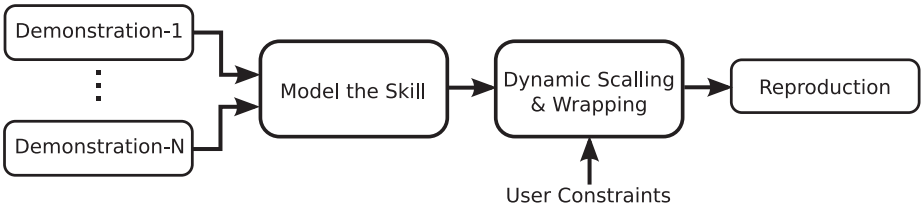


Fig. 2. The conceptual principle of programming by demonstration.

The symbol $n(x)$ means "neighbor of x", and the expression $h \in n(x) \setminus \{f\}$ denotes a region that covers a set of nodes x without including f.

The message-passing is started on every leaf-node by computing and propagating messages to its neighbors. The message propagation continues until all nodes receive the messages in both directions:

$$b_i(X_i) = \prod_{n(X_i)} \mu_{f(X_i)\to(X_i)}(X_i) \tag{7}$$

$$b_F(X_i) = f_F(X_i) \prod_{n(F)} \mu_{X_i\to f(X_i)}(X_i) \tag{8}$$

where $b_i(X_i)$ refers to the "belief" over a set of variable nodes X_i, and $b_F(X_i)$ refers to the "belief" over a factor node F (with internal function f_F) that is connected to X_i. In this circumstance, X_i refers to a region in the graph; whereas x refers to individual variable node within that region as expressed in (5) and (6).

The consistency of the belief propagation is established when the propagated messages to the variable agree in terms of the marginal of the factor over the corresponding variable:

$$b_F(X_i) = b_i(X_i). \tag{9}$$

In the termination step, the posterior probability of a variable $p(x)$ can be computed as the product of all messages directed toward x. This means that since the message passed on any given edge is equal to the product of all but one of these messages, $p(x)$ can be computed as the product of the two messages that were passed (in the opposite direction) over any single edge incident on x.

The reasoning mechanism of a factor graph relies on its inference procedure to obtain the posterior probability of certain nodes. This inference procedure does not require external computation other than the belief propagation itself.

2.4. Related works

Our previous work used factor graphs to model an omnidirectional mobile robot in a simulation environment [40]. In this paper, we describe an extension of such models to work with a real robot as well as to work with a different robot platform.

Our approach has some similarities to the Bayesian network setting proposed by Sturm et al. [32]. In their approach, the connectivity of the rigid parts that constitute the object is modeled, including the articulation model of the individual link; thus, it is only applicable to a manipulator type robot. In contrast, our approach offers a more generic solution that can be applied to another type of robots, such as mobile robots.

Another biomimetic approach for solving an inverse kinematics problem was proposed by Artemiadis [33]. Although their method was inspired by human's upper limb model, it was not a complete probabilistic graphical model since they used the Bayesian network only for describing the dependencies among the human joint angles, and then used an objective function in standard Jacobian inverse kinematics to finally solve the problem.

Regarding our dynamic factor graph, it can be considered as a heuristic approach similar to the iterative method. The basic idea of the iterative method is to approach the final state from a starting state by continuously examining the optimality expressed by the reduced cost function during each step.

An action-sequence approach depends on the initial posture, and it attempts to exploit the geometrical property of the kinematic chain. One particular example is known as cyclic coordinate descent (CCD) [41]. The CCD method iteratively minimizes errors by evaluating one joint variable at a time, starting from the end effector inward toward the manipulator base until a convergence point is obtained. The difference of this CCD method to ours is that, CCD can only provide a feasible posture if manipulator constraints for restricting motions are incorporated. Whereas in our method, the robot constraints can be learned from data.

Our dynamic factor graph will be used extensively for imitation learning. Although there is no consensus on which method performs the best when dealing with generalization problems at the trajectory level, most practical PbDs are usually performed using either statistical modeling (such as GMM and HMM) or dynamic-system-based modeling (such as DMP). In its original method, GMR tries to find the robot's joints trajectory by solving (3) using Lagrange optimization iteratively that will give the final solution:

$$\dot{\mathbf{q}} = (\mathbf{W}^q + \mathbf{J}^T\mathbf{W}^x\mathbf{J})^{-1}(\mathbf{W}^q\epsilon_q + \mathbf{J}^T\mathbf{W}^x\epsilon_x)$$

where $\mathbf{W}$ is the normalized sum of covariance matrices with respect to robot's joints $\mathbf{q}$ and the end effector's pose $\mathbf{x}$, whereas ϵ_q and ϵ_x are discrepancies of the current joints and pose trajectories to the desired ones. Here, $\epsilon_q = q^d(t) - q(t-1)$ and $\epsilon_x = x^d(t) - x(t-1)$, where the desired trajectories q^d and x^d can be estimated continuously by regression technique. Using this formulation, the joint trajectories that will be followed by the robot are computed as $q(t) = q(t-1) + \dot{q}(t)$.

Our factor graph model offers an alternative solution to this trajectory generalization by utilizing filtering-and-smoothing through forward–backward message-passing operation in belief propagation on factor graphs (i.e., similar to the Baum–Welch algorithm used in the hidden Markov model [42]).

3. Robot kinematics modeling in factor graphs

In this section, we describe our method for developing robot models along with its parameter learning mechanism.

3.1. Neurally inspired factor graphs

In order to use a factor graph in a real technical system, especially using digital hardware, its parameters need to be discretized. In this framework, a discrete variable X is a measurable function $X : \Omega \to S$ from the finite/countable sample space Ω to another measurable finite state space S. Thus, learning the parameters is basically a task of approximating a probability mass function (PMF). In our work, we used the expectation maximization algorithm (EM) for learning the parameter of a network.

In the belief propagation for EM, the network will propagate messages iteratively during which the network's parameters are regularly updated. Each iteration consists of two steps: the expectation update for the log likelihood given the old parameters and the observed data, and the maximization procedure to update the parameters. Normally, the expectation update of the log likelihood is computed as follows:

$$\begin{aligned} E[\log p(\mathbf{X})|\mathbf{Y},\theta] &= \sum_x p(\mathbf{X}|\mathbf{Y},\theta)\log p(\mathbf{X}|\theta) \\ &= E\left[\log \prod_i \left(\frac{1}{Z}\prod_a f(X^i)\right)|\mathbf{Y},\theta\right]. \end{aligned}$$

Here f denotes an internal function of a factor node in a factor graph and $X^i = x$ indicates a specific variable configuration (i.e. state) for this function. Hence, $f(X^i = x)$ corresponds to a single parameter of that function.

The maximization step is performed by setting the partial derivative with respect to $f(X^i = x)$ equal to zero that leads to the following final update step:

$$f^{t+1}(X^i = x) = f^t(X^i = x) \cdot \frac{\langle p(X^i = x \mid Y^i, \theta)\rangle_i}{p(X^i = x)}.$$

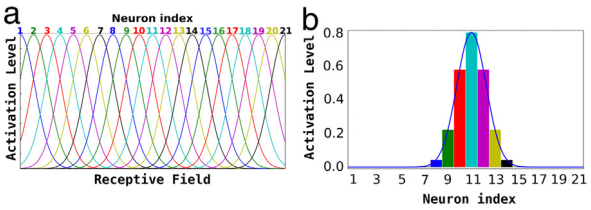


Fig. 4. (a) The Gaussian tuning curves in a homogeneous population comprised of 21 neurons. (b) The measured activation levels from all neurons are combined to produce the overall probability distribution.

The EM update rule above was derived under the assumption that the partition function Z would only be subject to small changes during an update.

The discretization of continuous values in our factor graphs uses a population coding technique [43]. This method was inspired by the idea of message encoding from a population of neurons in the central nervous system. Such a neuron population will react in synchrony after the stimulus [44,45] to produce a resemblance to a certain multinomial distribution. The precision of the population can scale exponentially with the number of neurons [46,47].

In our work, we used Gaussian tuning curves for the population coding. The combined activation levels from all neurons then shapes the overall distribution of the corresponding population. Fig. 4 shows the tuning curves and the resulting population code. To determine how many neurons are needed to properly encode a value, we use Bayesian Information Criterion (BIC) metric that is computed using the following formula:

$$\mathcal{B} = -\mathcal{L} + \frac{\rho(N_n)}{2}\log(N_p)$$

where $\mathcal{B}$ is the BIC score that will be computed iteratively for all possible N_n-number of neurons in a population, $\mathcal{L}$ is the log-likelihood of our model, N_p is the number of points in the D-dimensional training dataset, $\rho(N_n)$ is the impact factor introduced by the selected number of neurons N_n that will be computed as: $\rho(N_n) = (N_n - 1) + N_n * (D + 0.5 * D * (D + 1))$. In our experiment, we found that the minimum number of neurons N_n to produce a satisfactory result is 12 [40].

Having the minimum number of neurons, however, is not enough in order to produce accurate results. If the distribution of the values is not uniform, then the tuning curves should be adjusted to fit the distribution of the data. In our work, we employed a self-organizing-map (SOM) technique to fine tune the curves so that they could cover the data distribution properly.

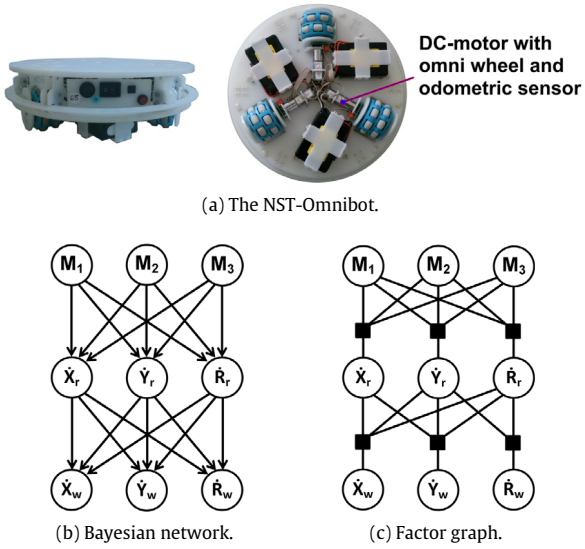
3.2. Kinematic model of a mobile robot

The first part of our hybrid robotic system shown in Fig. 1 is the mobile platform. It is a three wheels omni-directional mobile robot (see Fig. 5(a)). To perform the robot motion, we have to control the velocity of each wheel. The robot velocity in the world coordinate system is determined by the robot's wheels velocity using the following relations:

$$\mathbf{v^w} = \mathbf{G}\mathbf{v^r} \tag{10}$$

$$\text{where } \mathbf{v^r} = \mathbf{K}\dot{\mathbf{q}} = \begin{bmatrix} \frac{\sqrt{3}}{3} & \frac{-\sqrt{3}}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{-2}{3} \\ \frac{1}{3L} & \frac{1}{3L} & \frac{1}{3L} \end{bmatrix} \dot{\mathbf{q}} \tag{11}$$

$$\text{and } \mathbf{G} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$



(a) The NST-Omnibot.

(b) Bayesian network.

(c) Factor graph.

Fig. 5. The NST-Omnibot: a three-wheels omni-directional mobile robot developed at the research group "Neuroscientific System Theory" (NST) in Technische Universität München. (a) It has three independent DC-motors with an internal PID controller and a potentiometer as an odometric sensor. (b) A Bayesian network model for the kinematics of the mobile robot. (c) The factor graph version of the model in (b).

where $\mathbf{v^w}$ is the robot velocity in the world coordinate system, $\mathbf{v^r}$ is the robot velocity in the robot-self coordinate system, $\dot{\mathbf{q}}$ is the vector of the wheels' velocities, and $\mathbf{G}$ is the coordinate transformation matrix which takes robot poses α as its argument. Readers who are interested in deriving $\mathbf{K}$ and $\mathbf{G}$ for such a mobile platform are referred to [1].

Computing kinematics using the above formulas has at least two drawbacks. First, it relies heavily on deterministic sensor values, which in reality will be easily disturbed by noises; hence, it is only good for simulations. Second, during the real implementation, the robot motion will be affected by some physical uncertainty such as internal electromechanical traction as well as friction between the wheel and the floor that introduces a drift due to wheel slip. In order to take such physical uncertainty into consideration, Eqs. (10)–(12) should be modified such that:

$$\begin{aligned} \mathbf{v^w} &= \zeta_{\mathbf{L}}\mathbf{G}\mathbf{v^r} \\ \mathbf{v^r} &= \zeta_{\mathbf{M}}\mathbf{K}\dot{\mathbf{q}} \end{aligned} \tag{13}$$

where $\zeta_{\mathbf{L}}$ is an accumulated time-dependent factor related with electromechanical characteristics of the motor and $\zeta_{\mathbf{M}}$ is an accumulated constant factor related with wheel's slip.

Our factor graph offers a comprehensive way to model (13). Thus, it does not only overcome standard kinematic limitations, but also make it more adaptive to environmental changes. If we consider formula (11) and (13) respectively, we can easily understand that each wheel contributes independently to the global robot motion. We captured this insight into our robot model that can be trained to map the Cartesian robot motion into the velocity of the three wheels of the robot. The robot shown in Fig. 5(a) can receive wheel driving commands as integer values within range $[-500, 500]$ rpm, which will be used by the internal PID controller of the robot's motor to drive the wheel. First, we built a model that captures this kinematic relation as a Bayesian network shown in Fig. 5(b). Then we transformed the model into a factor graph shown in Fig. 5(c).

A set of nodes $\{M_1 \cdots M_3\}$ shown in Fig. 5(c) represent wheel velocities, a set of nodes $\{\dot{X}_r, \dot{Y}_r, \dot{R}_r\}$ represent the robot velocity in the robot coordinate system, and a set of nodes $\{\dot{X}_w, \dot{Y}_w, \dot{R}_w\}$

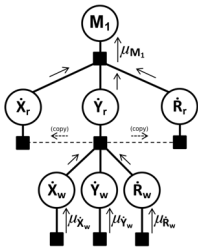


Fig. 6. An inverse kinematics model as a result of decoupling the network shown in Fig. 5c by exploiting independences given the observed variables. μ_{M_1}, $\mu_{\dot{X}_w}$, $\mu_{\dot{Y}_w}$, and $\mu_{\dot{R}_w}$ are messages representing "beliefs" about variable's values propagating through the network. Thin arrows indicate the flow of messages during belief propagation to produce final output at the "root" node (M_1).

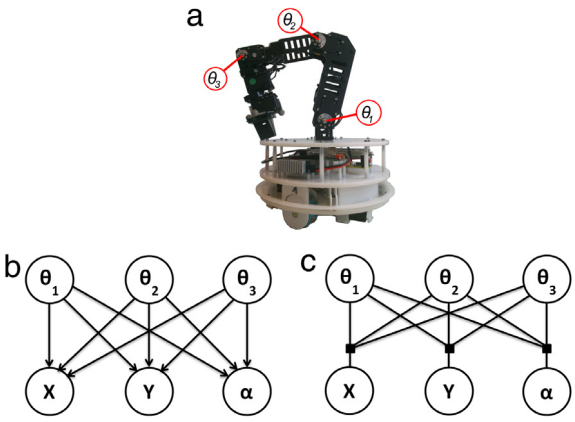


Fig. 7. (a) The robotic arm on top of the mobile robot along with its joint's labels. (b) A Bayesian network model for the kinematics of the robotic arm. Here, $\theta_1, \theta_2, \theta_3$ are the joints of the robot; X, Y is the Cartesian position of the actuator relative to the robotic arm base, and α is the actuator pose. (c) The factor graph version of (b).

represent the robot velocity in the world coordinate system. We can decouple the network into three separate models because it satisfies a constrained condition due to the independence between scope variables of the factor. This will result in similar models for both forward and inverse kinematics but with different scopes of the factor nodes as shown in Fig. 6.

After designing the model structure, the next step is to determine the parameters of the model by learning them from data. To generate data for our model, we followed the idea of motor babbling [6]. Motor babbling is a process of repeatedly performing a random motor command for a short duration in which the robot continuously observes its motion; i.e., the robot relates its internal states with its environment.

3.3. Kinematic model of a manipulator

The second part of our hybrid robotic system shown in Fig. 1 is the robotic arm. Each servo of the robotic arm shown in Fig. 7(a) has its own PID controller. Here, we developed the kinematic model of the robotic arm using the same method as the model for the mobile robot and modified the variables accordingly. The resulting model is depicted in Fig. 7(c). This kinematic model basically performs a mapping function from the joint space to the task space, and it considers only the likelihood problem instead of the posterior problem (i.e., there is no direct link between joint variables (θs)), which implies that the model does not involve the interlink-dynamic between robot's joints. For our robot shown in Fig. 7(a), there will be two solutions for the inverse kinematics with the input value X, Y, and α referred to as the "elbow-up" and "elbow-down" configurations. One practical solution for our robot is to constrain the value of θ_2 to be always in the "elbow-down" position. This is the most efficient solution for the robot because in most circumstances the arm will always be in a curved-down position. This is also true because the second orientation of the gripper will be determined solely by the pose of the mobile base. With this constraint, we can use the same network shown in Fig. 7(c) to perform the inverse kinematics.

3.4. Kinematic chain using dynamic factor graph

A factor graph is a flexible tool that can be used also for handling dynamic situations. In addition to static networks that exploit full constrained robot configuration, we proposed an extension for solving the inverse kinematics problem using a dynamic factor graph. This extension is based on a mimetic approach that comes from an interpretation of how a human arm usually moves during the task of reaching and placing an object. When a human moves his hand to grasp an object, he first makes some initial posture estimations of his hand and later adjusts his forearm and upper arm until the object is reached. With this insight, we developed a graphical model for the robot kinematics using a Markov chain model shown in Fig. 8. The state of the robot cannot be measured directly and the robot must maintain its own "beliefs" about itself and its environment. In belief propagation setting, those beliefs, which are represented by (9), can be computed as posterior probability distribution over state variables conditioned on the available data. The belief over a state variable x at step k conditioned on all past measurements $y_{1:k}$ and all past controls $u_{1:k}$ is expressed as:

$$b(x_k) = p(x_k \mid y_{1:k}, u_{1:k}). \tag{14}$$

We assume that the states are complete; i.e. the knowledge of past states, measurements, or control inputs does not carry additional information that are relevant with the determination of the current state. It means that we can remove the current measurement y_t from Eq. (14) which yields:

$$\tilde{y}_k = p(y_k \mid x_k, y_{1:k-1}, u_{1:k}) = p(y_k \mid x_k). \tag{15}$$

Hence, Eq. (14) can be re-written as:

$$\begin{aligned} \tilde{b}(x_k) &= p(x_k \mid y_{1:k}, u_{1:k}) \\ &= \eta p(y_k \mid x_k) p(x_k \mid y_{1:k-1}, u_{1:k}). \end{aligned} \tag{16}$$

Eq. (16) has a recursive form where the term $p(x_k \mid y_{1:k-1}, u_{1:k})$ is actually the prior belief similar to Eq. (14) before incorporating the new measurement y_k. The recursive form of the belief distribution now becomes:

$$b(x_k) = \eta \cdot \tilde{y}_k \cdot \tilde{b}(x_k) \tag{17}$$

where η is the normalizer constant that follows the probabilistic law enforcing that the maximum value is 1. This will be applied to the network shown in Fig. 8. It is carried out by iterating between the forward and backward phases until convergence as described below.

The forward phase is started by sending the desired actuator's pose (Z value) and the current actuator's joints (f_{θ_1} value) to node θ_1. This node then produces a message that contains the prior belief about θ_1, given the desired pose, to node θ_2. Node θ_2 then produces a message to θ_3 incorporating all prior beliefs about θ_1 and θ_2 given the desired actuator's pose. The backward phase begins by generating a message from node θ_3, which will be propagated and modulated toward θ_2 and θ_1. The posterior beliefs about θ_1, θ_2, and θ_3 are obtained by multiplying the incoming messages to the corresponding node. These will become the new estimated joint values that will re-enter the network through f_{θ_1}, f_{θ_2}, and f_{θ_3}. The network iterates until all joint values converge into steady values.

The overall process for computing the posterior belief of each joint is shown in Algorithm 1. Here, lines 3–8 are computed using (16), whereas lines 9–11 are computed using the updated belief (17). The Kullback–Leibler divergence (D_{KL}) is used to measure the divergence level of the newly learned parameters using the standard formula:

$$D_{KL}(b_{\theta_n} \parallel \theta_n) = \sum_i b_{\theta_n}(i) \ln \frac{b_{\theta_n}(i)}{\theta_n(i)}$$

where i = neuron index in the population code.

Algorithm 1 Overall Posterior Belief Inference

1: Initialization: $f_{\theta_n} \leftarrow \theta_n$
2: Initiate iteration

Process-1: Forward Phase
3: Compute $\tilde{b}^f_{\theta_1}$ from Z and f_{θ_1}
4: Compute $\tilde{b}^f_{\theta_2}$ from Z, f_{θ_2}, and $\tilde{b}^f_{\theta_1}$
5: Compute $\tilde{b}^f_{\theta_3}$ from Z, f_{θ_3}, and $\tilde{b}^f_{\theta_2}$

Process-2: Backward Phase
6: Compute $\tilde{b}^b_{\theta_3}$ from Z and f_{θ_3}
7: Compute $\tilde{b}^b_{\theta_2}$ from Z, f_{θ_2} and $\tilde{b}^b_{\theta_3}$
8: Compute $\tilde{b}^b_{\theta_1}$ from Z, f_{θ_1} and $\tilde{b}^b_{\theta_2}$

Process-3: Update Beliefs
9: Compute b_{θ_1} from Z and $\tilde{b}^b_{\theta_1}$
10: Compute b_{θ_2} from Z, $\tilde{b}^f_{\theta_1}$ and $\tilde{b}^b_{\theta_3}$
11: Compute b_{θ_3} from Z and $\tilde{b}^f_{\theta_3}$

Process-4: Evaluate Convergence
12: **for all** θ **do**
13: Compute KL-divergence $D_{KL}(\theta)$
14: **if** $D_{KL}(\theta) < D_{threshold}$ **then**
15: $\theta_n = b_{\theta_n}$
16: Stop iteration;
17: **else**
18: Repeat from Process-1;
19: **end if**
20: **end for**

return Θ

4. Experimental results

4.1. Mobile robot model performance

We evaluated our mobile robot model performance in a simulation environment and in a real experiment using the NST-Omnibot, which is a three wheels omni-directional mobile robot (see Fig. 5(a)).

4.1.1. Evaluation in a simulation environment

It is interesting to study robot behaviors based on our factor graph model in a simulation environment because we have full control on noises that are artificially generated and introduced to the model. Hence, this simulation model might reveal important information such as how robust the model operates in the presence of noise. We are also interested to see the effect of the Gaussian tuning curves in our population codes for discretizing continuous values. The challenge of using Gaussian distributions for tuning curves in a population code is how to properly define the variance values of such distributions. For this purpose, we generated simulation data based on the robot kinematics model expressed in Eq. (11).

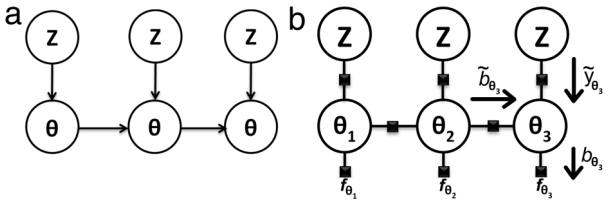


Fig. 8. (a) A Markov chain network for modeling inverse kinematics. Here, variable Z represents the pose of the robot actuator (i.e., the gripper) that can be decomposed into X, Y and α. (b) The factor graph version where each factor represents the conditional distribution associated with an edge in the model (a). The factors f_{θ_1}, f_{θ_2}, and f_{θ_3} are the estimated actuator's joints.

During the simulation, we generated commands for robot wheels and computed the resulting robot velocities. This "motor babbling" strategy was repeated several times until we collected enough data. Afterward, we fed the collected data to the network shown in Fig. 5(c) and let the network performed the inference. Fig. 9(a) shows the generated random commands for the wheel-1 (shown in red plot), as well as the estimated motor commands (shown in blue plot) from the inference result by the factor graph model shown in Fig. 6. This example result shows that the predicted commands were quite close to the originally generated commands.

When we generated the dataset, we also introduced some levels of white noise in order to evaluate the robustness of our model in the presence of noise. In our experiment, the decreasing rate in the performance of the inverse kinematics model was slightly higher than that of the forward kinematics model. With the presence of noise up to 15%, the performance was still good and acceptable. With additional noises of more than 15%, the performance became deteriorated. This information told us that in a real subsequent implementation, the model was capable of handling a Gaussian noise with zero mean and a variance of 1.38. Beyond that, our model might not be able to produce correct control signals for the robot. We were also interested in investigating the influence of the variable's cardinality (i.e., the number of neurons in the population code) to the performance of the system.

After we re-run the inference with several cardinalities, we found that it was sufficient to use 15 states for variables cardinality. The result is depicted in Fig. 9(b), which shows the correlation between the original/generated motor commands and their corresponding estimation. We can see that there is a slight variance along the ideally diagonal line (measured as normalized root-mean-squared error or NRMSE). The diagonal line is a baseline result that was computed deterministically using (10)–(12) without any learning mechanism. The graph shown in Fig. 9(b) was produced by using 15 states for variables' cardinality. This variance could be reduced by increasing the number of neurons in the population codes. For example, by using 50 states for variable's cardinality, much higher precision result was produced (shown in Fig. 9(c)). Nevertheless, the linearity of the curve was maintained well within the valid input range. In general, similar results were obtained for the other wheels of the robot.

4.1.2. Experiment on NST-Omnibot

We evaluated our model further in a real scenario. We performed an experiment by using a real robot NST-Omnibot (Fig. 5(a)). In this experiment, we used a camera tracking system to localize the robot position in a planar space and to calculate its moving speed. The camera tracking system provided information about the position of the robot in the world coordinate system. We needed to transform this absolute position value of the robot into

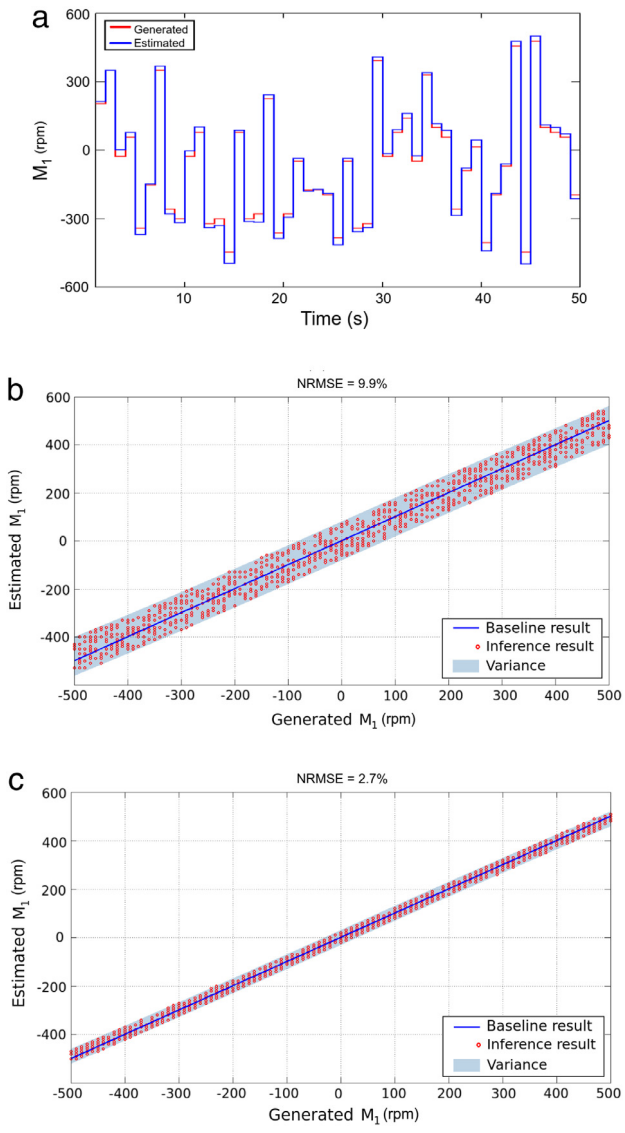


Fig. 9. Plot of estimated motor commands given the original wheels commands in the inverse kinematics case: (a) plotted in time sequence, (b) presented as a correlation plot for 15 states in variable's cardinality, and (c) also as a correlation plot for 50 states in variables' cardinality. In (b) and (c), the baseline result were obtained by using exact formula (10)–(12) on artificially-generated simulation data. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

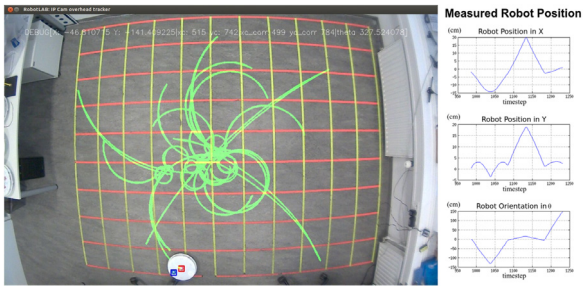


Fig. 10. The robot velocity data was generated using a motor babbling scenario.

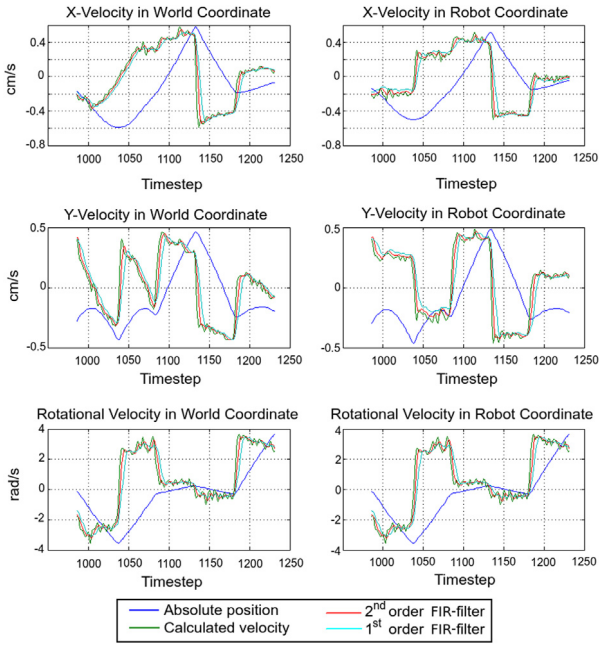


Fig. 11. Preparing the data before feeding them into the network. The raw data from the camera tracking system was very noisy. In order to reduce the noise effect so that its variance fell below 1.38, the data went through the filtering process using FIR filters. FIR filters were used because of their linear phase characteristic and their simple architecture for hardware implementation.

the robot velocity value. Fig. 10 shows how the robot generated the data and recorded the trajectory.

In the transformation, the data were filtered to reduce the noise in the camera recording data. Filtering was done on the data that conveyed information about the pose (position and orientation) of the robot. From our previous experiment using simulation data, we knew that it is better for our model to have streams of input data with noise's variance (by assuming it is a Gaussian noise) less than 1.38%. After filtering, the Cartesian velocities as well as the rotational velocity of the robot were calculated. The commands were changed every two seconds during the experiment to give us enough sampling points where the filter's delay could be neglected. Fig. 11 shows the data preparation from the camera tracking system.

The coordinate transformation matrix expressed in (12) was used to obtain the robot velocity in the robot-self coordinate system. Within its coordinate system, the velocity of the robot can be mapped properly into the velocity of the wheels. During one iteration that lasted for two seconds, only 50% of the data portion in the middle of the period was considered because both ends of the data-stream within that period contained a transitional fluctuation between successive iterations. Fig. 12 shows this process.

After pre-processing data, we fed them into the network shown in Fig. 5 for the inverse kinematics inference (i.e., computing the robot's command given the desired velocity $(\dot{X}, \dot{Y}, \dot{R})$). The result is depicted in Fig. 13, which was produced by using 15 states for the population codes. It shows a similar result, as expected, to the simulation version shown in Fig. 9(b). However, the variance is a bit larger (shown as higher NRMSE) due to various left-over noises and uncertainties in the experiment.

4.2. Manipulator model performance

As in the previous kinematic model of the mobile robot, we evaluate our manipulator model based on the inference mechanism in the inverse kinematic model. For our robot shown in Fig. 7(a), the value of θ_2 was constrained in the "elbow-down" position. This is a preferable solution for the robot because in most situations, especially in a pick-and-place scenario, the arm will

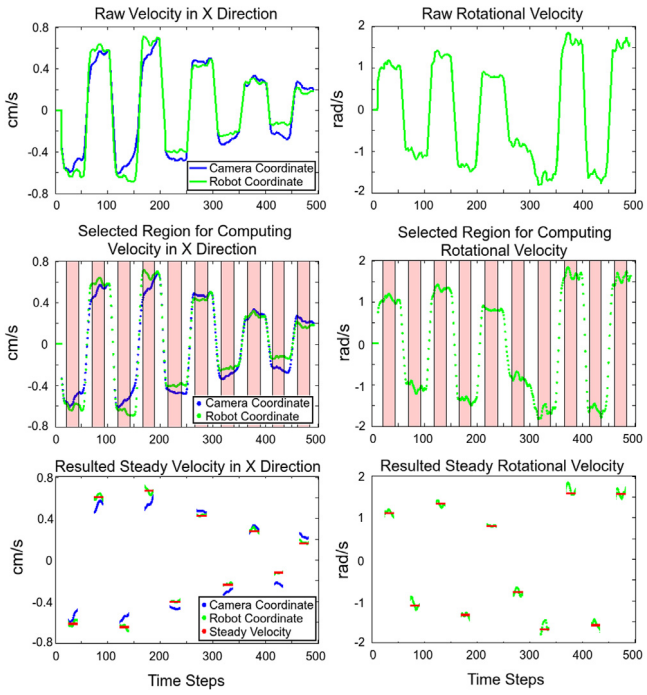


Fig. 12. The camera tracking system provides information about the robot's pose in the world coordinate system. In order to work with our model, the data needed to be transformed into the robot-self coordinate system.

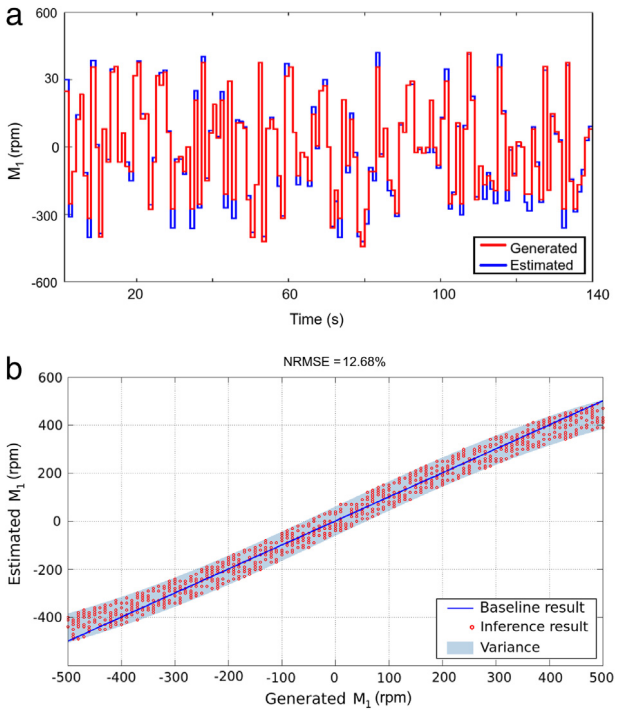
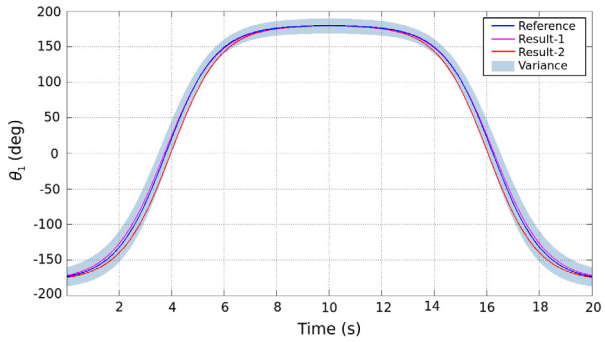
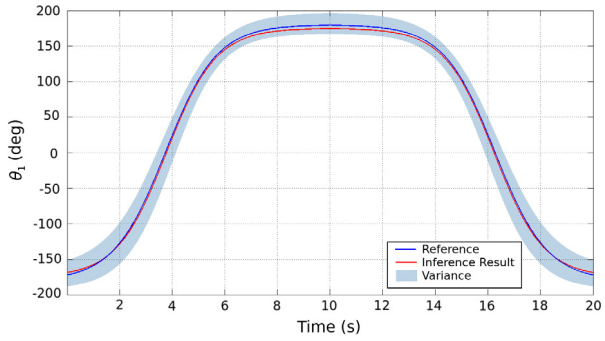


Fig. 13. Plot of generated motor commands given the desired robot velocities in the inverse kinematic case: (a) plotted in time sequence, (b) presented in a correlation plot.

almost always be in a curved-down position. In such a scenario, the second orientation of the gripper will be determined solely by the pose of the mobile base (see Fig. 1). With this constraint, we could use the network shown in Fig. 7(c) to perform the inverse kinematics.



(a) Using fully constrained model of Fig. 7(c).



(b) Using Markov chain model of Fig. 8(b).

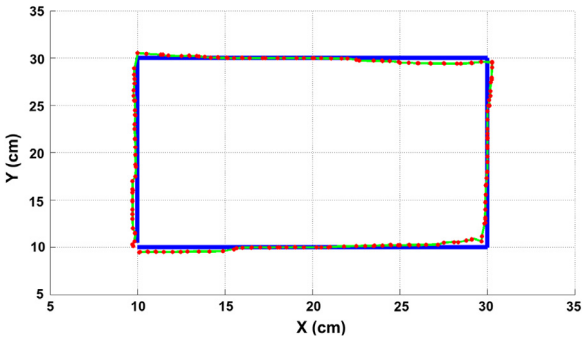
Fig. 14. The inverse kinematics results for θ_1. For the other joints, the results were very similar.

In the experiment, we sent joints' angle to the robot and measured the gripper pose. In the collected dataset, those joints' angles were referred to as the reference points. After several experiments, also using "motor babbling" scenario, we fed the collected data to the network and let the network learned the parameters for its factor nodes using MLE. Once the network completed the training phase, we performed reasoning by sending desired gripper poses to the network and ask the network to estimate the corresponding joints' angles.

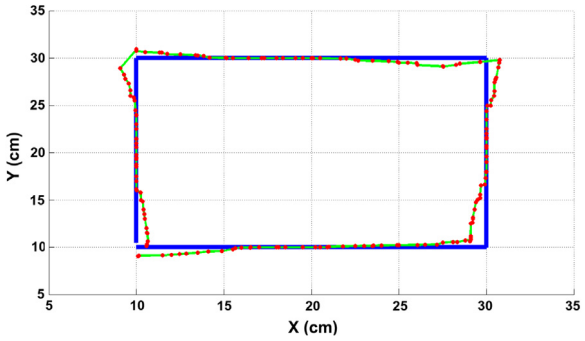
Fig. 14 shows the result of the inverse kinematics of the robotic arm using our two models: the fully constrained model and the Markov chain model. The results were quite similar with reference points with a small variance along the line due to the varying number of states used in the discretization of the variables.

Fig. 14(a) was produced using the fully constrained model (see Fig. 7(c)). In this graph, Result-1 was produced by using 50 states for variables' cardinality, whereas Result-2 as produced by using 30 states for variables' cardinality. The accuracy of Result-1 as measured in NRMSE is 2.9%, which is quite similar to the accuracy in Fig. 9(c) that was produced by the model with the same number of states. The accuracy of Result-2 was measured at 5.7%, which clearly indicates that Result-1 has a better approximation than Result-2. Lowering variables' cardinality down to 15 yielded wider variance as shown in the graph.

For the Markov chain model (see Fig. 8(a)), we found that the results were better than the fully constrained model even though they used a lower number of states. However, they were more sensitive to variance values of the tuning curves that effected the overall variation on the inference results. Fig. 14(b) depicts the inference result of the Markov chain model using only 25 states. The accuracy of this model is quantified using NRMSE and it was calculated at 3.3%. This result is quite similar to Result-1 in Fig. 14(a) with the difference in NRMSE value at about 0.4%, but with half number of neurons.



(a) Result in simulation.



(b) Result using NST-Omnibot.

Fig. 15. (a) The inference result of kinematic models in a simulation environment. The outputs of the inverse kinematics network shown in Fig. 8(b) were sent to the forward kinematics network shown in Fig. 7(c). The blue line is the trajectory for the robot to follow and the red dots connected by the green line is the resulting trajectory. (b) The experiment was conducted using the real robot. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

From this experiment, we were interested to test further the performance of the Markov chain model in a complete scenario. We created a simple task in which the robot followed a rectangular trajectory. We tested the network by using both simulation data and the real robotic arm. The rectangular trajectory occupied a planar space within the range [10, 30] cm on both directions (X and Y), sampled every 0.5 cm with the actuator orientation kept constant at 0°. In the simulation environment, those sampled points were fed into the inverse kinematics network, and the computed $f_{\theta_1}, f_{\theta_2}$, as well as f_{θ_3} were fed into the forward kinematics network shown in Fig. 7(c). Using the real robot, the same process was repeated for calculating the inverse kinematics, but the resulting $f_{\theta_1}, f_{\theta_2}$, and f_{θ_3} were sent directly to the robot. Fig. 15 shows the results of this experiment.

The result demonstrates that the model was able to reproduce the rectangular trajectory, although the shape was a bit distorted especially near the corner paths. However, this impreciseness is merely due to technical problems and noises in the robot hardware for data communication.

The test depicted in Fig. 15 demonstrates an inverse kinematics control based on a learned model. We extended this model-based inverse kinematics solution to an imitation (or PbD) scenario. In order to use the PbD paradigm for generating the skill, we needed to provide several demonstrations from which the trajectory of the movement could be learned. Each demonstration trajectory was fed into the network and the parameter of the network was updated accordingly. As an example for a complex trajectory, we "guided" the robotic arm to pick up an object from one position and then place it on another position. This scenario is depicted in Fig. 17.

In the scenario shown in Fig. 17, we simplified the task by excluding the grasping task during the demonstrations (i.e., the object was manually placed at the gripper). The left side in Fig. 16 shows the trajectories recorded during the training. The right side in Fig. 16 shows the trajectory result from the regression network over several demonstrations for each joint of the robotic arm. It shows only a snapshot of the teaching of the robot where we held the robotic arm and then extended and retracted the arm to create a trajectory. We can see from the picture that the paths were constrained at both ends of the motion. These represent regularities that the model can extract and exploit to produce a new skill for another task (e.g., by using a scaling procedure). After several demonstrations, we performed the regression on the trajectories. The estimated trajectory was then sent to the robot's controller. The snapshots of this run after learning the trajectory is depicted in Fig. 18. Although this was a simple scenario, it demonstrated one important aspect: the dynamic factor graph model (shown in Fig. 8) can be extended into a regression model to perform the imitation learning. It also demonstrated that the model can generalize over the variations in joint angles. This was done by projecting the collected trajectories onto a latent space and by estimating the resulting trajectory in a regression fashion.

5. Discussion

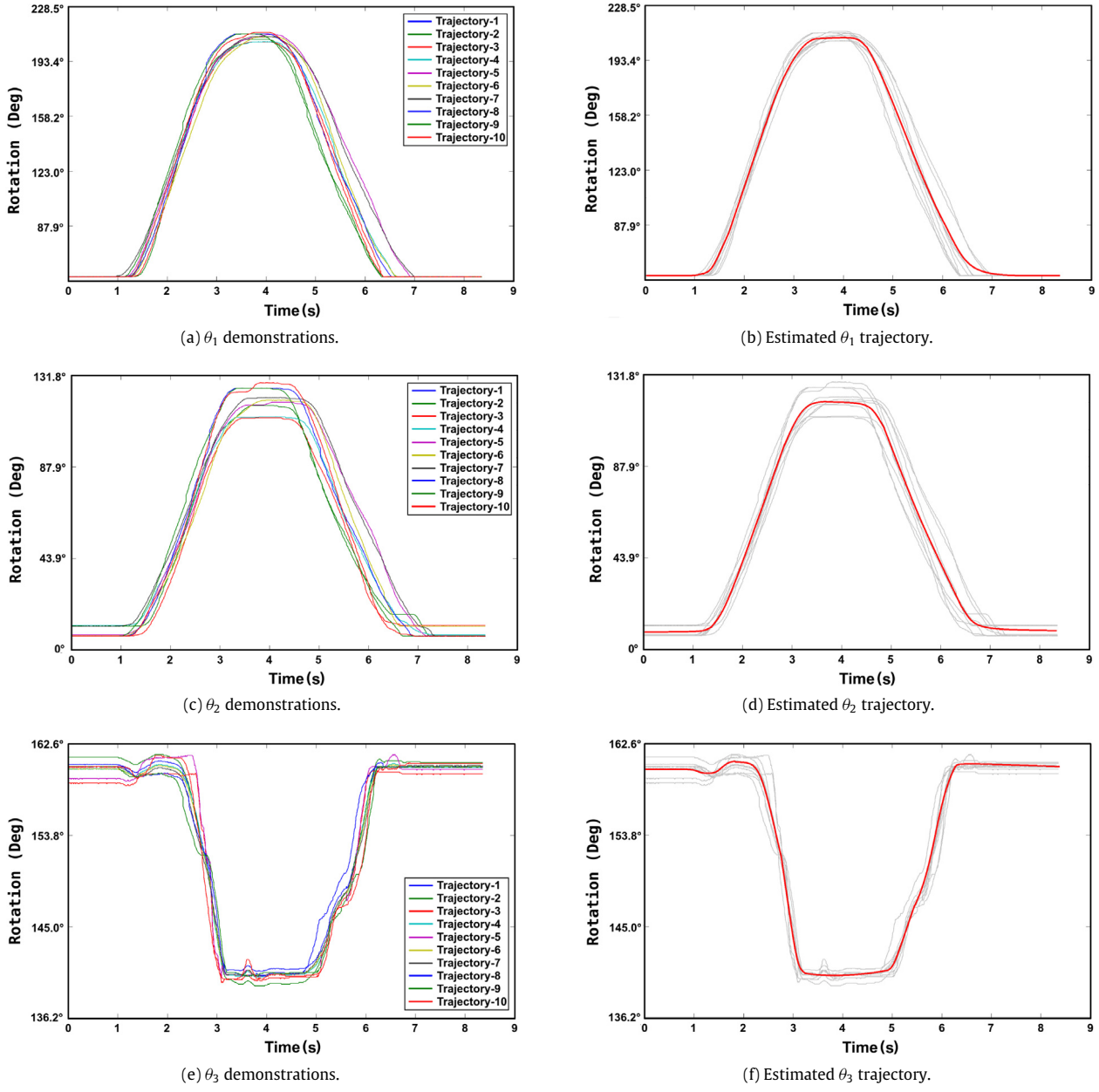
In the previous section, we described our experiments on factor graphs to model different abstraction levels for two types of fundamental robotic systems: a mobile robot and a robotic arm (i.e., a manipulator). The models represent the kinematic aspect of the robots. The first model is a generic one that represents an N-to-N mapping network for transforming the expected robot velocity to/from each wheel velocity. It can be used for both forward and inverse kinematics computation. We tested the network in a scenario where a top-head camera tracking system was used for acquiring the robot position. The inverse kinematics model then predicted the expected wheel velocity given the robot velocity. The results show that the model produced good results (see Figs. 9 and 13).

The first model presents one important remark: it can be used to learn any mapping function without deriving its exact mathematical formulas. This approach confers benefits on us especially when working with a high order dynamic system or in a dynamic environment. Hence, our model can be used in a more complex scenario that extends the basic functionality of the mobile platform, such as in a synchronous localization and mapping (SLAM) scenario.

As a generic model, our first factor graph model, which was initially developed for the mobile robot, is also applicable for the robotic arm. The physical constraint of our robot made it possible to produce a satisfactory result because it could be modeled in a fully constrained configuration. Conceptually, the other solution could also be computed but it would not be beneficial to our robot.

We also presented the second model which was based on a mimetic approach. Basically, the second model represents a dynamic network. To our knowledge, the factor graph itself does not have the capability to deal with the dynamic behavior of the system. Hence, we extended the static model into a dynamic one by unrolling it several time steps and used the powerful inference mechanism of the factor graph to perform queries on the network (e.g., filtering and smoothing).

We applied the second model on the robot to follow a simple trajectory (see Fig. 15). The result showed that it was quite reliable in a simulation environment, where the noise level was maintained low and there was no jittery effect on the data communication of the robot. When we applied the model on a real robotic arm, we observed degradation in the quality of the result though the rectangular shape that the robot had to follow can be obviously seen. At this point, it can be seen that the model has a limitation, such



(a) θ_1 demonstrations.

(b) Estimated θ_1 trajectory.

(c) θ_2 demonstrations.

(d) Estimated θ_2 trajectory.

(e) θ_3 demonstrations.

(f) Estimated θ_3 trajectory.

Fig. 16. Learning robot trajectory from several demonstrations for each joint of the robotic arm (θ_1, θ_2, and θ_3 correspond to the joints of the robotic arm shown in Fig. 7(a)).

as missing active controllers, for the stability of the robotic arm. Nevertheless, our experiment was proposed to give an intuitive example of how to use a factor graph in such a complex scenario.

The rectangular trajectory was given to the robot and the inverse kinematics model of the robot inferred it to produce the robot's joints configuration. We extended this experiment into a mimetic task in which we taught the robot by means of several demonstrations, and the robot tried to estimate a generalized trajectory. Using the regression capability of the factor graph network, the robot was able to acquire a new skill from the generalized trajectories. The demonstration of this imitation fashion (Figs. 17 and 18), proved that the dynamic factor graph can be extended into a regression model to perform the imitation learning.

In general, the factor graph models basically provided a means to build motion primitives that could be used to build a more complex robotic system. This is similar to the complex motion of biological entities, such as humans, where motion is composed of

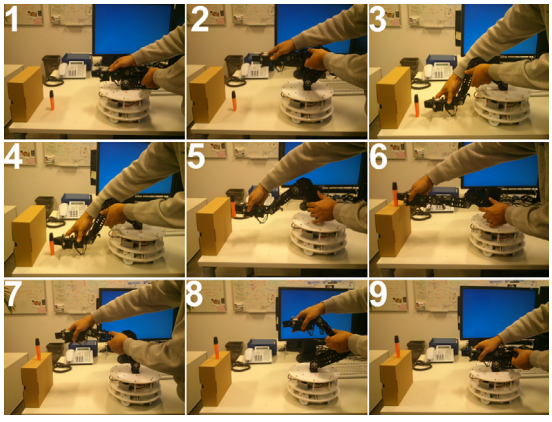


Fig. 17. Guiding the robotic arm to follow a trajectory.

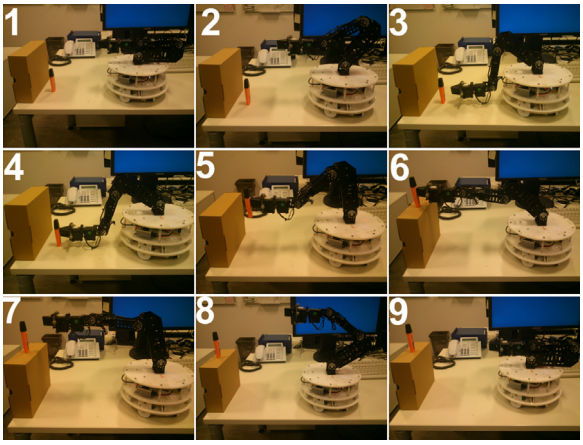


Fig. 18. Robotic arm executes the trajectory it learned before.

many small coordinated motion primitives. We provided an example of how to use these motion primitives in an imitation scenario. Conceptually, the robot can be taught with any possible task, but it requires a slight modification on the robot model if the task has several constraints that need to be addressed simultaneously.

Finally, there are two advance topics that are not yet explored in this paper. The first is regarding the generalization of a new skill. Once a robot has learned the skill (i.e., the generalized trajectory of the given task), it can use the standard scaling procedure to apply different contexts such as different start and goal positions. However, this scaling mechanism is a static procedure and will not take into account the transitional dynamic between two successive learned skills [13]. A better way to accommodate the dynamic behavior of skills is by using the dynamic imitation paradigm [34]. We believe that our dynamic factor graph model is suited for this paradigm.

The second is regarding the unified model of the mobile manipulator shown in Fig. 1 that combines both robotic platforms. Developing a unified factor graph model for such a hybrid mobile manipulator is very challenging since it involves cyclic networks that entail more examination and further exploration. It is well-known that such cyclic networks are ill-posed and there is no consensus on how to achieve convergences in such networks [48–50]. Furthermore, such a model for the mobile manipulator should consider the scenario and/or the terrain where the robot will be used, and apply appropriate mechanism for autonomous localization such as SLAM.

Together, the development of a dynamic imitation learning using factor graphs as well as the integration of SLAM algorithm into a unified generic model for the hybrid robot require a deep and thorough discussion. Hence, we take the liberty for addressing these two advance topics in our future work.

6. Conclusion

This paper presents our work on a model-based approach using biologically plausible factor graphs that can be used to build two different kinematic models for robots in a generic fashion. We proposed to use the model-based approach because we argued that it can be used to create efficient tailored models in robotic scenarios that incorporate rapid prototyping through learning mechanisms. The learning phase in our proposed method produced an uncomplicated solution by exploring the valid solution space. This was accomplished by feeding reliable data from robot experiments that represented all possible robot configurations. This model-based learning enhanced robot's movement reliability because it represented a mechanism that was always physically correct.

Our model-based approach to robot's kinematics and control can be optimized for applications that involve custom models for each new robot configuration. We demonstrated that our method is a generic one that can be extended into more complex setups for any conceptually autonomous robotic application. Our factor graph networks worked very well with two fundamental robot models, and exemplified the flexibility and generality of a dynamic factor graph. We also demonstrated that our method is applicable for mimetic scenarios that embody our understanding of natural movements of humans body. Hence, in this paper, we have laid a foundation of a method that can be extended further with online learning mechanisms to produce a more adaptive intelligent system for addressing future challenges in the robotic domain.

Acknowledgments

This work was supported by DAAD (Deutscher Akademischer Austauschdienst e.V.) under the grant A/10/76323 and by NST (Neuroscientific System Theory) at Technische Universität München.

References

[1] J. Craig, Introduction To Robotics: Mechanics and Control, third ed., Prentice Hall, 2004.
[2] Z. Zhu, J. Li, Z. Gan, H. Zhang, Kinematic and dynamic modelling for real-time control of Tau parallel robot, Mech. Mach. Theory 40 (9) (2005) 1051–1067.
[3] Z. Bi, S. Lang, Kinematic and dynamic models of a tripod system with a passive leg, IEEE/ASME Trans. Mechatronics 11 (1) (2006) 108–111.
[4] R. Williams, J. Wu, Dynamic obstacle avoidance for an omnidirectional mobile robot, J. Robot. (2010).
[5] F. Matsuno, K. Suenaga, Experimental study on control of redundant 3-D snake robot based on a kinematic model, in: Adaptive Motion of Animals and Machines, Springer Tokyo, 2006.
[6] A. D'Souza, S. Vijayakumar, S. Schaal, Learning inverse kinematics, in: IEEE International Conference on Intelligent Robots and Systems, IEEE/RSJ, Maui, Hawaii, USA, 2001, pp. 298–303.
[7] S. Dubowsky, E. Papadopoulos, The kinematics, dynamics, and control of free-flying and free-floating space robotic systems, IEEE Trans. Robot. Autom. 9 (5) (1993) 531–543.
[8] S. Schaal, Is imitation learning the route to humanoid robots? Trends Cogn. Sci. 3 (1999) 233–242.
[9] M. Lungarella, G. Metta, R. Pfeifer, G. Sandini, Developmental robotics: A survey, Connect. Sci. 15 (4) (2003) 151–190.
[10] M. Spenko, G. Haynes, J. Saunders, M. Cutkosky, A. Rizzi, R. Full, D. Koditschek, Biologically inspired climbing with a hexapedal robot, J. Field Robot. 25 (4–5) (2008) 223–242.
[11] F. Mussa-Ivaldi, E. Bizzi, Motor learning through the combination of primitives, Philos. Trans. R. Soc. Lond. Ser. B Biol. Sci. 355 (1404) (2000) 1755–1769.
[12] E. Bizzi, V. Cheung, A. d'Avella, P. Saltiel, M. Tresch, Combining modules for movement, Brain Res. Rev. 57 (1) (2008) 125–133.
[13] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal, Dynamical movement primitives: learning attractor models for motor behaviors, Neural Comput. 25 (2) (2013) 328–373.
[14] H. Burgess, M. Granato, Modulation of locomotor activity in larval zebrafish during light adaptation, J. Exp. Biol. 210 (14) (2007) 2526–2539.
[15] M. Ito, Mechanisms of motor learning in the cerebellum, Brain Res. 886 (1–2) (2000) 237–245.
[16] M. McDannald, Y. Takahashi, N. Lopatina, B. Pietras, J. Jones, G. Schoenbaum, Model-based learning and the contribution of the orbitofrontal cortex to the model-free world, Eur. J. Neurosci. 35 (7) (2012) 991–996.
[17] S. Lee, S. Shimojo, J. O'Doherty, Neural computations underlying arbitration between model-based and model-free learning, Neuron 81 (3) (2014) 687–699.
[18] H.-A. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, F. Kschischang, The factor graph approach to model-based signal processing, Proc. IEEE 95 (6) (2007) 1295–1322.
[19] M. Toussaint, C. Goerick, A bayesian view on motor control and planning, in: From Motor Learning To Interaction Learning in Robots, Springer, Berlin, 2010.
[20] C. Bishop, Model-based machine learning, Phil. Trans. R. Soc. A 371 (0120222) (2013).
[21] O. Sener, A. Saxena, rCRF: Recursive belief estimation over CRFs in RGB-D activity videos, in: Robotics Science and Systems (RSS), 2015.

[22] A. Saxena, A. Jain, O. Sener, A. Jami, D. Misra, H. Koppula, Robo brain: Large-scale knowledge engine for robots, in: International Symposium on Robotics Research, ISRR, 2015.
[23] B. Siciliano, Kinematic control of redundant robot manipulators: A tutorial, J. Intell. Robot. Syst. 3 (3) (1990) 201–212.
[24] E. Oyama, A. Agah, K. MacDorman, T. Maeda, S. Tachi, A modular neural network architecture for inverse kinematics model learning, Neurocomputing 38–40 (2001) 797–805.
[25] R. Köker, C. Öz, T. Cakar, H. Ekiz, A study of neural network based inverse kinematics solution for a three-joint robot, Robot. Auton. Syst. 49 (3–4) (2004) 227–234, Patterns and Autonomous Control.
[26] A. Ghasemi, M. Eghtesad, M. Farid, Neural network solution for forward kinematics problem of cable robots, J. Intell. Robot. Syst. 60 (2) (2010) 201–215.
[27] S.-W. Kim, J.J. Lee, M. Sugisaka, Inverse kinematics solution based on fuzzy logic for redundant manipulators, in: IEEE/RSJ International Conference on Intelligent Robots and Systems '93, IROS'93, vol. 2, 1993, pp. 904–910.
[28] J. Jih-Gau, Fuzzy neural network approaches for robotic gait synthesis, IEEE Trans. Syst. Man Cybern. B 30 (4) (2000) 594–601.
[29] U. Beyer, F. Śmieja, A heuristic approach to the inverse differential kinematics problem, Int. J. Intell. Robot. Syst. 18 (4) (1997) 309–327.
[30] M. Ayyildiz, K. Cetinkaya, Comparison of four different heuristic optimization algorithms for the inverse kinematics solution of a real 4-dof serial robot manipulator, Neural Comput. Appl. (2015) 1–12.
[31] N. Rokbani, A. Casals, A. Alimi, IK-FA, a new heuristic inverse kinematics solver using firefly algorithm, in: A.T. Azar, S. Vaidyanathan (Eds.), Computational Intelligence Applications in Modeling and Control, in: Studies in Computational Intelligence, vol. 575, Springer International Publishing, 2015, pp. 369–395.
[32] J. Sturm, C. Stachniss, W. Burgard, A probabilistic framework for learning kinematic models of articulated objects, J. Artif. Intell. Res. 41 (2011) 477–526.
[33] P. Artemiadis, P. Katsiaris, K. Kyriakopoulos, A biomimetic approach to inverse kinematics for a redundant robot arm, Auton. Robots 29 (3–4) (2010) 293–308.
[34] E. Rückert, G. Neumann, M. Toussaint, W. Maass, Learned graphical models for probabilistic planning provide a new class of movement primitives, Frontiers Comput. Neurosci. 6 (97) (2013).
[35] S. Calinon, F. Guenter, A. Billard, On learning, representing, and generalizing a task in a humanoid robot, IEEE Trans. Syst. Man Cybern. B 37 (2) (2007) 286–298.
[36] J. Aleotti, S. Caselli, Robust trajectory learning and approximation for robot programming by demonstrations, Robot. Auton. Syst. 54 (5) (2006) 409–413.
[37] S. Schaal, Dynamic movement primitives - A framework for motor control in humans and humanoid robotics, in: Adaptive Motion of Animals and Machines, Springer Tokyo, 2006.
[38] K. Mülling, J. Kober, O. Krömer, J. Peters, Learning to select and generalize striking movements in robot table tennis, Int. J. Robot. Res. 32 (3) (2013) 280–298.
[39] F. Kschischang, B. Frey, H.-A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Trans. Inf. Theory 47 (2) (2001) 498–519.
[40] I. Sugiarto, J. Conradt, Discrete belief propagation network using population coding and factor graph for kinematic control of a mobile robot, in: IEEE International Conference on Computational Intelligence and Cybernetics, CYBERNETICSCOM, Yogyakarta, Indonesia, 2013, pp. 136–140.
[41] T. Li-Chun, C. Chih, A combined optimization method for solving the inverse kinematics problem of mechanical manipulators, IEEE Trans. Robot. Autom. 7 (4) (1991) 489–499.
[42] Z. Ghahramani, An introduction to hidden markov models and bayesian networks, Int. J. Pattern Recognit. Artif. Intell. 15 (1) (2001) 9–42.
[43] I. Sugiarto, P. Maier, J. Conradt, Reasoning With Discrete Factor Graph, in: IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems, ROBIONETICS, IEEE, 2013, pp. 170–175.
[44] W. Gerstner, W. Kistler, Spiking Neuron Models: Single Neurons, Populations, Plasticity, Cambridge University Press, 2002.
[45] S. Wu, S. Amari, H. Nakahara, Population coding and decoding in a neural field: a computational study, Neural Comput. 14 (5) (2002) 999–1026.
[46] A. Mathis, A. Herz, M. Stemmler, Resolution of nested neuronal representations can be exponential in the number of neurons, Phys. Rev. Lett. 109 (2012) 018103.
[47] A. Deneve, P. Latham, A. Pouget, Efficient computation and cue integration with noisy population codes, Nature Neurosci. 4 (8) (2001) 826–831.
[48] A. Ihler, W.J. Fischer III, A. Willsky, Loopy belief propagation: convergence and effects of message errors, J. Mach. Learn. Res. 6 (2005) 905–936.
[49] C. Yanover, Y. Weiss, Finding the M most probable configurations using loopy belief propagation, in: Advances in Neural Information Processing Systems 16, MIT Press, 2004, pp. 289–296.
[50] J. Mooij, H. Kappen, Sufficient conditions for convergence of loopy belief propagation, in: The Twenty-First Conference on Uncertainty in Artificial Intelligence, UAI2005, Edinburg, Scotland, 2005.

Indar Sugiarto is a lecturer in the Department of Electrical Engineering at Petra Christian University. He holds B.Sc. in Electrical Engineering from Institut Teknologi Sepuluh Nopember, Indonesia, M.Sc. degrees in Information and Automation Engineering from Universität Bremen, Germany, and Ph.D. in Electrical and Computer Engineering from Technische Universität München, Germany. His main research interests are computational intelligence, bayesian machine learning, reconfigurable computing, and robotics.

Jörg Conradt is a Junior Professor at the Technische Universität München in the Faculty of Electrical Engineering and Information Technology, Institute of Automation and Control Engineering. The laboratory is affiliated with TUM's Competence Center on NeuroEngineering and the Munich Bernstein Center for Computational Neuroscience. He holds an M.S. degree in Computer Science/Robotics from the University of Southern California, a Diploma in Computer Engineering from TU Berlin and a Ph.D. in Physics/Neuroscience from ETH Zurich. His research group on Neuroscientific System Theory(http://www.nst.ei.tum.de) investigates key principles by which information processing in brains works, and applies those to real-world interacting technical systems.