2015 IEEE 54th Annual Conference on Decision and Control (CDC)
December 15-18, 2015. Osaka, Japan

Neural Programming: Towards Adaptive Control in Cyber-Physical Systems

K. Selyunin[1], D. Ratasich[1], E. Bartocci[1], M.A. Islam[2], S.A. Smolka[2], and R. Grosu[1]

[1] Vienna University of Technology, Austria
konstantin.selyunin,denise.ratasich,ezio.bartocci,radu.grosu@tuwien.ac.at

[2] Stony Brook University, NY, USA
mdaislam,sas@cs.stonybrook.edu

***Abstract*— We introduce *Neural Programming* (NP), a novel paradigm for writing adaptive controllers for Cyber-Physical Systems (CPSs). In NP, *if* and *while* statements, whose discontinuity is responsible for frailness in CPS design and implementation, are replaced with their smooth (probabilistic) neural *nif* and *nwhile* counterparts. This allows one to write robust and adaptive CPS controllers as dynamic neural networks (DNN). Moreover, with NP, one can relate the thresholds occurring in soft decisions with a Gaussian Bayesian network (GBN). We provide a technique for learning these GBNs using available domain knowledge. We demonstrate the utility of NP on three case studies: an adaptive controller for the parallel parking of a Pioneer rover; the neural circuit for tap withdrawal in *C. elegans*; and a neural-circuit encoding of parallel parking which corresponds to a proportional controller. To the best of our knowledge, NP is the first programming paradigm linking neural networks (artificial or biological) to programs in a way that explicitly highlights a program's neural-network structure.**

I. Introduction

Recent advances in sensing, actuation, communication, and computation, along with their integration within increasingly smaller, interconnected devices, has lead to the emergence of the so-called *cyber-physical systems* (CPSs), which operate in uncertain, continuously evolving environments. Examples of CPS include smart grids, smart factories, smart transportation, and smart health care [1].

To equip CPSs with the ability to adapt and act in uncertain environments, various researchers have started to investigate whether current CPS analysis, design and implementation techniques are still adequate. Parnas, Chaudhuri and Lezama identified in a series of papers [2]–[4] the if-then-else construct as the main culprit for program frailness. In a simple decision of the form `if (x > a)`, the predicate $x > a$ acts like a step function (the bold black line in Fig. 1), with infinite plateaus to the left and right of the discontinuity point $x = a$. The nesting of if-then-else statements leads to a highly nonlinear program inducing a large number of plateaus separated by discontinuous jumps. This has important implications.

From a CPS design point of view, where one is interested to find the values of `a` for which an optimization criterion is satisfied, predicates of the form $f(x) > a$ are problematic. They render *CPS optimization intractable*. To alleviate this problem, Chaudhuri and Lezama [3] proposed to smoothen the steps by passing a Gaussian input distribution through the CPS. The authors, however, stop short of proposing a new programming paradigm, and the step-like functions in the programs to be optimized pose considerable challenges, as they cut the Gaussians in very undesirable ways.

From a CPS implementation point of view, conditional statements of the form `if (f(x) > a)` are also difficult to deal with. They render *CPSs nonadaptive*: a small change in the environment or the program itself, may lead to catastrophic consequences, as the CPS is unable to adapt. In the AI community, where steps are called *hard neurons* and sigmoid curves are called *soft neurons*, adaptation and robustness is achieved by learning a particular form of Bayesian networks with soft-neuron distributions. Such networks have recently achieved noteworthy performance, for example in recognition of sophisticated patterns [5], [6].

Having identified the if-then-else programming construct as the major source of difficulty in the design and implementation of CPSs, we propose a combined approach for developing adaptive CPS controllers using *Neural Programming* NP. In the NP paradigm, a controller (skeleton) is written representing a dynamic neural network (DNN). Additional knowledge about the thresholds in the controller's *nif* and *nwhile* conditions, the key NP constructs, is then optimally encoded as a Gaussian Bayesian network (GBN).

In contrast to deterministic-program (DP) controllers, which capture only one execution for any given input, NP controllers capture all valid executions. For example, if one has enough space between two cars, there are multiple ways to parallel park (PP), all of which are valid (i.e. result of a car being parked at a dedicated parking spot without collision). There is no reason to restrict PP to only one trajectory (a DP), like in [3], because a small perturbation (for example sliding), may lead to an invalid trajectory (e.g. trajectory that leads to collision). In NP, a small perturbation may eliminate some of the valid trajectories, but leave enough of the valid ones such that the controller can adapt. By adaptation we understand the ability to react on environment's change by eliminating trajectories over time that are no longer valid. Using NPs we are able to handle uncertainty in the model [7].

To validate our new paradigm we define three controllers for two case studies: *Parallel parking* from [8] and the *Tap Withdrawal neural circuit* of *C. elegans* [9]. In the PP case study, we show how to achieve robustness by expressing the controller as a NP, where the associated GBN helps to compensate for perturbations in the environment. We provide a technique for learning the parameters of a GBN from traces. In the second case study, the controller is *C. elegans*' neural circuit for tap withdrawal, expressed in terms of NPs. Each synaptic link of a neuron can either fire or not, which corresponds to a *nif* statement. Since NP is a general concept, it can be used as a language for expressing controllers or to model the system. In the third case study, we use knowledge gained from the *C. elegans* circuit to provide a controller for parallel parking, expressed as a neural circuit, where the voltage, current, and conductance become position (or angle), velocity, and control flow, respectively.

The main contributions of the work presented in this paper can be summarized as follows:

1) *We propose NP, a new programming paradigm* for the development of robust CPS controllers as DNNs where *if* statements are replaced by smooth *nif* statements.
2) *We demonstrate the versatility of this new paradigm* on two case studies: an adaptive parallel parking controller for a Pioneer rover (the youtube videos are available at [10]), a tap-withdrawal neural circuit for *C. elegans*, and a parallel-parking neural-circuit.

The rest of the paper is organized as follows. Section II discusses related work. Section III introduces our programming paradigm. Section IV focuses on how to learn the Gaussian Bayesian network. Section V presents our case studies, implementation platform, and experimental results. Section VI offers our concluding remarks.

II. Related Work

Although probabilistic programs, Gaussian Bayesian networks (GBN) and neural networks were considered before, the development of NPs[1], with *nif* and *nwhile* thresholds related within a GBN is, to the best of our knowledge, new. Moreover, the encoding of the C.Elegans neural circuit as an NP is new as well, and so is the encoding of an adaptive, proportional, parallel-parking controller as a neural circuit.

Probabilistic programs introduced in [11] differ from "traditional" ones by the ability to sample at random from the distribution and condition the values of variables via observations, where the result of the program is the expectation of the return value. This approach requires static analysis to give an output of the program, while in our case we concentrate on a run or an execution with given probabilistic semantics.

In [3], the authors adapted the signal and image processing technique called *Gaussian smoothing (GS)*, for program optimization. Using GS, a program could be approximated by a smooth mathematical function, which is a convolution of a denotational semantics of a program with a Gaussian function. This approximation facilitates solving the parameter synthesis problem. In [4] this idea was extended to define soundness and robustness of smooth interpretation of programs. In these works the authors do not consider any means for eliminating the re-normalization step of the probability density function (PDF) when a variable is passed through a conditional branch in the execution trace. Moreover, they stop short of proposing new, smooth control statements.

Learning Bayesian Networks comprises different tasks and problem formulations: i) Learning the structure of the network, ii) Learning the conditional probabilities for the given structure and iii) Performing querying-inference for a given Bayesian Network [12]. In [13] the authors introduce a unified method for both discrete and continuous domains to learn the parameters of Bayesian Network, using a combination of prior knowledge and statistical data.

Various formulations of a mobile parking problem were extensively studied for robots with different architectures [14]–[18]. In [18] the authors use a custom spatial configuration of the ultrasonic sensors and binaural method to perceive the environment and park the robot using predefined rules. Another approach [16] is to approximate the trajectory for the parking task with a polynomial curve, that the robot could follow with the constraints satisfied, and minimize the difference between specified trajectory and actual path. In [19] the authors try to infer a "hidden trajectory" from a series of observations, while in our setting in order to adapt we allow all admissible trajectories.

NP control and fuzzy control [20] have different ontological commitments: In NPs, statements about the world are either true or false, but the NP's partial knowledge renders its beliefs probabilistic. In fuzzy-logic control the statements about the world are fuzzy (have a continuous domain of values) whereas the knowledge about the world is total. Moreover, fuzzy logic implicitly assumes (in the definition of conjunction) that variables are independent, whereas NPs do not. In fact the dependence between variables plays a key role in learning parameters of a GBN.

Inspirations from neuroscience profoundly influence technological development and hardware design today. For instance, Intel developed *TrueNorth* [21], an architecture, which represents a cluster of neurons implemented in CMOS technology. The programming environment "Corelet" [22] for this architecture specifies an interconnection between neurons, and stays on the lower abstraction level w.r.t. to our approach, where we define operation logic as a NP.

As every controller, a NP must be aware of the internal state of the process to be able to robustly control the CPS. Sensors measure the outputs of a process, whereof the state can be estimated. The measurements are distorted by noise and the environment may be unpredictable. State estimators [23]–[25] and in particular Kalman filters [24], [26] are commonly used methods to increase the confidence of the state estimate evaluated out of raw sensor measurements.

III. Neural Programs

Traditional inequality relations (e.g. $>$, $\geq$, $\leq$, $<$) define sharp (or firm) boundaries on the satisfaction of a condition, and can therefore be seen as step functions (see Fig. 1). Using firm decisions in a program operating on Normal random variables (RVs) cuts distributions in half, resulting in unnormalized and invalid PDFs; see Fig. 2. In this figure, the upper-right plot shows what happens to the PDF of a Normal RV after passing it through a traditional conditional statement (if or while). To avoid such situations and maintain valid probability density one needs to re-normalize the PDF.

To avoid re-shaping its PDF each time an RV x is passed through a conditional, we introduce a new control structure called *neural if*, or *nif* for short. The name is coined to

[1]Starting from this section we use NP to abbreviate both a neural program and neural programming

express the key novelty of our approach: we propose to use smooth conditionals $\texttt{cdf}_{\mu,\sigma^2}(x)$ instead of firm ones.

Let $\#$ range over the set of comparison operators $\{>, \geq, \leq, <\}$. Nif statements are of the form `nif(x # a,`σ^2`) S1 else S2`, where `x # a` is a predicate and σ^2 is a variance. With every `nif` statement, we associate a boolean RV (also called `nif`) with the evaluation of `(x # a,`σ^2`)`. Specifically, if RV nif(`x # a`) evaluates to 1 with variance σ^2, statement `S1` will be taken; otherwise `S2`.

The evaluation of `nif` statements is performed in two steps: (i) Find an $\mathcal{R}$-interval I representing the confidence in which `S1` will be taken. (ii) Check if a sample from the Gaussian Distribution (GD) $\mathcal{N}(0, \sigma^2)$ falls within I. For the case where $\sigma^2 \to 0$ (no uncertainty), we require the `nif` statement to behave as a traditional `if` statement. Note that `nif` statements define a family of RVs parameterized by $\#$, a, and σ^2.

To find the I in (i), we calculate the difference `diff(x,a)` between `x` and `a` as follows, where ϵ is the smallest real number on a computer:

$$\texttt{diff(x,a)} = \begin{cases} \texttt{x - a} - \epsilon & \text{if } \# \text{ is } >, \\ \texttt{x - a} & \text{if } \# \text{ is } \geq, \\ \texttt{a - x} - \epsilon & \text{if } \# \text{ is } <, \\ \texttt{a - x} & \text{if } \# \text{ is } \leq . \end{cases}$$

Informally, our confidence is characterized by the difference: the larger the value of `diff(x,a)`, the larger the probability of executing `S1`. The probability of nif(`x # a`)$=1$ (the probability of executing `S1`) is given by $\texttt{cdf}_{0,\sigma^2}(\texttt{diff(x,a)})$ and defines the interval $[q_1; q_2]$ by calculating two symmetric quantiles q_1 and q_2 such that:

$$\int_{q1}^{q2} \texttt{pdf}_{0,\sigma^2}(\texttt{x})dx = \texttt{cdf}_{0,\sigma^2}(\texttt{diff(x,a)}). \qquad (1)$$

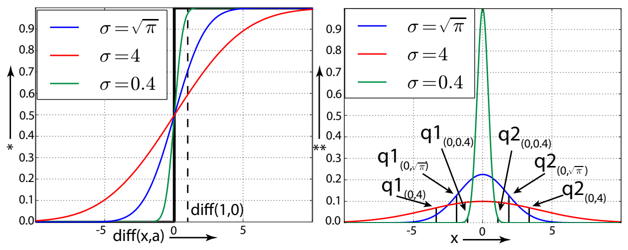
In step (ii), a random sample is taken from the distribution $\mathcal{N}(0, \sigma^2)$ and tested to see if it belongs to the confidence interval $[q_1; q_2]$. If it is within the interval, `S1` is executed; otherwise `S2` is executed. The probability to execute `S1` is influenced by the variance σ^2 (see Fig. 1(a)). The dependence is twofold: `diff(x,a)` shows how confident we are in making the decision, and σ^2 characterizes the uncertainty.

For the case $\sigma^2 \to 0$, the `nif` statement is equivalent to the `if` statement. In this case, the PDF is expressed as a Dirac function $\delta(x)$, which essentially concentrates all of the PD in a single point $x = 0$. Hence, the $\texttt{cdf}_{0,\sigma^2\to 0}(x)$ becomes a step function (the bold black line in Fig. 1). The two possible cases for evaluating `nif` statements without uncertainty are: (i) `diff(x,a)` ≥ 0 and (ii) `diff(x,a)` < 0. In the first case, the probability of executing `S1` is equal to 1; hence the interval is $(-\infty; +\infty)$ and includes every sample. For the second case, the probability of taking `S1` is 0; hence the interval cannot contain any sample.

To illustrate the evaluation of `nif` statements, consider the following example, where `x`, `a` $\in \mathcal{R}$, and $\sigma^2 \in \mathcal{R}^+$.

```
nif( x >= a, σ²) S1 else S2
```

Suppose in the current execution `x` $= 1$ and `a` $= 0$. Figure 1 illustrates how decisions are made if σ^2 is $0.4^2, \pi, 4^2$. Since `diff(x,a)` $= 1$, the probability of executing `S1` is defined by $\texttt{cdf}_{0,\sigma^2}(1)$ and for the above cases is equal to 0.994, 0.714 and 0.599, respectively. The intervals I are [-1.095;1.095], [-1.890; 1.890], and [-3.357; 3.357]. In the second step, we sample from the GDs with the corresponding σ^2 ($\mathcal{N}(0, 0.4^2)$, $\mathcal{N}(0, \pi)$, and $\mathcal{N}(0, 4^2)$), and check if the value lies within the intervals. The plot in Fig. 1(b) shows the interval I for various values of σ^2.



(a) $*$ P(nif = 1)   (b) $**$ PDF(x)

Fig. 1. (a) "Soft" (colored lines) and "hard" (bold black line) thresholds; (b) PDFs and the quantiles for $x = 1$ and $a = 0$

So far we have been concerned with execution of single samples $x \sim \mathcal{N}(\mu, \sigma^2)$ on `nif` statements. Fig. 2 illustrates what happens at the distribution level; in particular, the difference between passing a GD RV `x` $\sim \mathcal{N}(0, 0.1)$ through the statements `if(x >= 0.15)` and `nif(x >= 0.15, 0.1)`. Since the input RV x has a GD, and a GD is used to evaluate the condition, the result is a product of two GDs, which is also a GD scaled by some constant factor k. Using our approach, the GD is not cut in undesirable ways (upper-right plot in Fig. 2), and maintains its GD form after passing the `nif` statement.

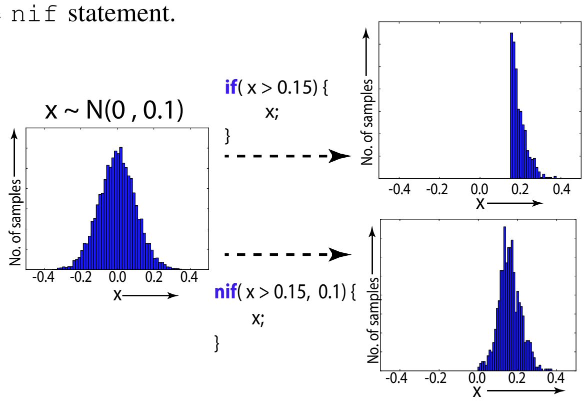


Fig. 2. Passing RVs through conditions

We can now apply "soft" thresholds to loops. The *neural while* `nwhile(` `x # a`,σ^2`)`$\{P_1\}$ statement takes a predicate `(x # a)` and a variance σ^2 and executes the program P_1 according to the following ruled: (1) Compute `diff(x # a)`, find interval I and quantiles q_1, q_2 according to Eq. 1. (2) Check if a random sample $x \sim \mathcal{N}(0, \sigma^2)$ is within the interval $[q_1; q_2]$. (3) If the sample belongs to the interval, execute P_1 and go to step (1); else exit. The `nwhile` is an extension of the traditional `while` statement that introduces uncertainty into its execution.

Since the `nif` and `nwhile` statements subsume the behavior of traditional `if` and `while` statements (the case $\sigma^2 \to 0$), we use them to *define an imperative language with probabilistic control structures*. Binary operators *bop* (e.g. addition, multiplication), unary operators *uop* (negation), and constants c are used to form expressions E. A program P is a statement S or combination of statements.

$$
\begin{aligned}
E &::= \;\; \mathtt{x}_i \mid c \mid bop(E_1, E_2) \mid uop(E_1) \\
S &::= \;\; \mathtt{skip} \mid \mathtt{x}_i := E \mid S_1; S_2 \mid \\
&\qquad \mathtt{nif}(\,\mathtt{x}_i \,\#\, c, \sigma^2\,)\; S_1 \;\mathtt{else}\; S_2 \mid \\
&\qquad \mathtt{nwhile}(\,\mathtt{x}_i \,\#\, c,\, \sigma^2)\{\; S_1 \;\}
\end{aligned}
$$

To illustrate neural programming with our language, consider the first case study involving the parallel parking of a mobile robot. The basic operations required are: Go backwards up to a point l_1, turn up to an angle α_1, go backwards up to l_2, turn up to α_2, and go backwards up to l_3. The control-program skeleton for this application can be specified as a sequence of `nwhile` statements, as shown in Listing 1.

Listing 1. Parallel parking program skeleton

```
nwhile(currentDistance < targetLocation1, sigma1){
  moving();
  currentDistance = getPose();
  }
updateTargetLocations();
nwhile(currentAngle < targetLocation2,    sigma2){
  turning();
  currentAngle = getAngle();
  }
updateTargetLocations();
nwhile(currentDistance < targetLocation3, sigma3){
  moving();
  currentDistance = getPose();
  }
updateTargetLocations();
nwhile(currentAngle < targetLocation4,    sigma4){
  turning();
  currentAngle = getAngle();
  }
updateTargetLocations();
nwhile(currentDistance < targetLocation5, sigma5){
  moving();
  currentDistance = getPose();
  }
```

The versatility of this approach is that the program skeleton is written only once and constitutes and infinite number of controllers (when the distance l_1 is smaller then the target, the next locations to be visited will be re-sampled in order to compensate for the difference). The question we next need to answer is:

What are the distances and turning angles for each action and how uncertain are we about each of them?

To find the unknown parameters in Listing 1, namely the target locations and variances, we use the learning procedure described in Section IV.

IV. Bayesian-Network Learning

Given a neural program as a skeleton, the next step is to identify the corresponding GBN and learn its parameters. We illustrate the process on the first case study, since for the second case study the network model is already given.

Parking can be seen as a sequence of moves and turns, where each action depends on the previous one: e.g. the turning angle typically depends on the previously driven distance. Due to sensor noise and imprecision, inertia and friction forces, and also many possible ways to perform a parking task starting from one initial location, we assume that the dependence between actions is probabilistic, and in particular, the RVs are distributed according to GD. We represent the dependencies between actions as the GBN in Figure 3, where l_i or α_j denotes a distance or a turning angle of the corresponding action and b_{ij} is a conditional dependence between consecutive actions.

$$\begin{tikzcd}
l_1 \arrow[r, "b_{21}"] & \alpha_1 \arrow[r, "b_{32}"] & l_2 \arrow[r, "b_{43}"] & \alpha_2 \arrow[r, "b_{54}"] & l_3
\end{tikzcd}$$

Fig. 3. Gaussian Bayesian Network for parking

In order to learn the conditional probability distributions of the GBN in Figure 3, and to fill in the `targetLocations` and the `sigmas` in Listing 1, we record trajectories of the successful parkings done by a human expert. Figure 4 shows example trajectories used during the learning phase.

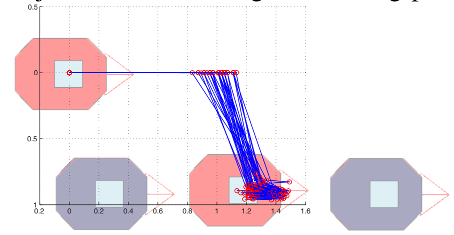


Fig. 4. Example trajectories for the parking task

We than use the fact that any GBN can be converted to a Multivariate Gaussian Distribution (MGD) [12] in our learning routine. Learning the parameters of the GBN can be divided into three steps:

1) Convert the GBN to the corresponding MGD,
2) Update the precision matrix $\mathbf{T} = \Sigma^{-1}$ of the MGD,
3) Extract σ^2s and conditional dependences from $\mathbf{T}$.

1. Conversion step. To construct MGD we need to obtain the mean vector μ and the precision matrix $\mathbf{T}$. The mean vector μ comprises the means of all the variables from the GBN. To find the symbolic form of the precision matrix, we use the recursive notation in [13], where the value of the coefficients b_i, will be learned in the update step below.

$$\mathbf{T}_{i+1} = \begin{pmatrix} \mathbf{T}_i + \frac{\mathbf{b}_{i+1}\mathbf{b}_{i+1}^T}{\sigma_{i+1}^2} & -\frac{\mathbf{b}_{i+1}}{\sigma_{i+1}^2} \\ -\frac{\mathbf{b}_{i+1}^T}{\sigma_{i+1}^2} & \frac{1}{\sigma_{i+1}^2} \end{pmatrix} \tag{2}$$

In order to apply Equation 2 we define an ordering starting with the initial node l_1, which precision matrix is $\frac{1}{\sigma_1^2}$. The vector $\mathbf{b}_i$ in Equation 2 comprises dependence coefficients for node i on all its immediate parents it in the ordering.

Since each action in the parking task depends only on the previous one , we can generalize the precision matrix for the arbitrary number of moves. For a GBN with k moves, all non-zero elements of the precision matrix $T \in \mathcal{R}^{k;k}$ can be found according to the Equation 3, where $\mathbf{T}(r, c)$ is a c-th element in a r-th row of the precision matrix with indices started from one.

$$
\begin{aligned}
\mathbf{T}(i, i-1) &= -\frac{b_{i(i-1)}}{\sigma_i^2}, \\
\mathbf{T}(i, i) &= \frac{1}{\sigma_i^2} + \frac{b_{(i+1)i}^2}{\sigma_{i+1}^2}, \\
\mathbf{T}(i, i+1) &= -\frac{b_{(i+1)i}}{\sigma_{i+1}^2},
\end{aligned}
\tag{3}
$$

2. Update step. Once we derived the symbolic form of the precision matrix , we use the training set, in order to learn the actual values of its parameters, as described in the algorithm from [12]. Each training example $\mathbf{x}^{(i)}$ corresponds to a vector of lengths and turning angles for a successful parking task. The total number of examples in the training set is M. The procedure allows us to learn iteratively and adjust the prior belief by updating the values of the mean μ and covariance matrix β of the prior, where v is a size of a training set for the prior belief, and $\alpha = v - 1$.

$$\beta = \frac{v(\alpha - n + 1)}{v+1}\mathbf{T}^{-1}, \tag{4}$$

The updated mean value μ^* incorporates prior value of the mean μ and the mean value of the new training examples $\mathbf{x}$.

$$\begin{aligned} \overline{\mathbf{x}} &= \frac{\sum_{i=1}^{M} \mathbf{x}^{(i)}}{M} \\ \mu^* &= \frac{v\mu + M\overline{\mathbf{x}}}{v+M} \end{aligned} \tag{5}$$

The size of the training set v^* is updated to its new value:

$$v^* = v + M \tag{6}$$

The updated covariance matrix β^* combines the prior matrix β with the covariance matrix of the training set $\mathbf{s}$:

$$\begin{aligned} \mathbf{s} &= \sum_{i=1}^{M} \left(x^{(i)} - \overline{\mathbf{x}}\right)\left(x^{(i)} - \overline{\mathbf{x}}\right)^T \\ \beta^* &= \beta + s + \frac{rm}{v+M}\left(x^{(i)} - \overline{\mathbf{x}}\right)\left(x^{(i)} - \overline{\mathbf{x}}\right)^T \end{aligned} \tag{7}$$

Finally, the new value of the matrix β is used to calculate the covariance matrix $(\mathbf{T}^*)^{-1}$, where $\alpha^* = \alpha + M$.

$$(\mathbf{T}^*)^{-1} = \frac{v^* + 1}{v^*(\alpha^* - n + 1)}\beta^* \tag{8}$$

3. Extraction step. The new parameters of the GBN can now be retrieved from the updated mean vector μ^* and from $(\mathbf{T}^*)^{-1}$. If new traces are available at hand, one can update the distributions by recomputing μ^* and $(\mathbf{T}^*)^{-1}$ using Equations 5-8. Unknown parameters from the program skeleton are learned from successful traces and these dependencies are used during the execution phase to sample the commands.

V. Case Studies

We illustrate our approach on the following case studies: (i) parallel parking of a Pioneer Rover and (ii) simulation of tap withdrawal neural circuit of a nematode (iii) controller for parallel parking as a neural circuit.

A. Parallel parking

We performed our experiments on a `Pioneer P3AT-SH` mobile rover from Adept MobileRobots [27] (see Figure 5). The rover uses the `Carma Devkit` from SECO with Tegra 3 ARM CPU running the Robot Operating System (ROS) on top of Ubuntu 12.04.

We use the following dymanics model of Pioneer rover for estimating the state of the robot:

$$x(k+1) = x(k) + v(k)\Delta t cos\left[\Theta(k+1)\right] + w_x\Delta t \tag{9}$$

$$y(k+1) = y(k) + v(k)\Delta t sin\left[\theta(k+1)\right] + w_y\Delta t \tag{10}$$

$$\Theta(k+1) = \Theta(k) + \omega\Delta t + w_\Theta\Delta t \tag{11}$$

$$v(k) = \frac{\omega_L(k)r_w + \omega_R(k)r_w}{2} + a\Delta t \tag{12}$$

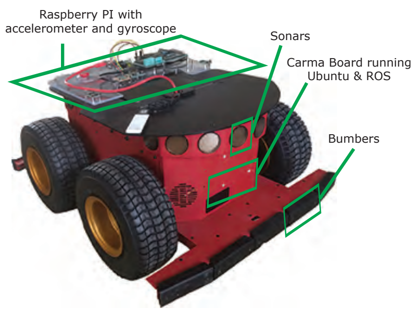

Raspberry PI with accelerometer and gyroscope
Sonars
Carma Board running Ubuntu & ROS
Bumbers

Fig. 5. Experimental platform: Pioneer Rover

$$\omega(k) = \frac{\omega_L(k)r_w - \omega_R(k)r_w}{R_b}, \tag{13}$$

where w_x, w_y and w_Θ are zero mean Gaussians taking into account noise during the observations; ω_L and ω_R are angular velocities of left and wheels respectively, r_w is a radius of a wheel and R_b is a base (0.3 m).

1) Structure of the Parking System: The parking system can be separated into several building blocks (see Figure 6). The block *Rover Interface* senses and controls the rover, that is, it establishes an interface to the hardware. The block *Sensor Fusion* takes the sensor values from the *Rover Interface* block, and provides the estimated pose of the rover to the high-level controller *Engine*. The *Engine* uses the *GBN* block to update the motion commands based on the estimated pose. Furthermore, the *Engine* maps the (higher level) motion commands to velocity commands needed by the *Rover Interface* to control the rover.

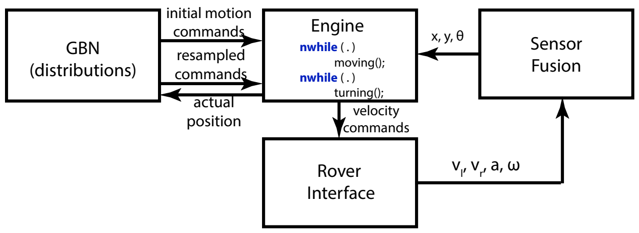


Fig. 6. Parking system architecture

a) The Gaussian Bayesian Network Block: The goal of the GBN block in Figure 6, is to generate motion commands for the Engine to execute. A motion command corresponds to a driving distance or a turning angle.

The distribution of the first move l_1 is independent from any other move and has the form $\mathcal{N}(\mu_1, \sigma_1^2)$. Starting from the second move α_1, each motion depends on the previous one: For motion number n, the distribution has the form $\mathcal{N}(\mu_n - b_{n,n-1} * x_{n-1}^{\text{sampled}}, \sigma_n^2)$. The initial command vector is obtained by sampling from l_1, and each subsequent command vector is obtained by taking into account the previous sample.

As the rover and its environment are uncertain (e.g., sensors are disturbed by noise) we use the pose provided by the sensor fusion unit to update the motion commands. Hence the motion commands are constantly adapted to take into account the actual driven distance (which could be different from the planned one due to the aforementioned uncertainty of the CPS). This allows us to incorporate the results of the sensor fusion algorithm in the updated commands.

b) The Engine Block: During the run we execute a motion command according to the semantics of the `nwhile` loop. In particular, the estimated pose is passed from the Sensor Fusion block to the Engine and compared with the target location, specified as a point on a 2-D plane. Since the rover is affected by noise its path can deviate and never come to the target location. To be able to detect and overcome this problem we estimate the scalar product of two vectors: The first one is the initial target location, and the second one is the current target location. This product becomes negative after passing the goal even on a deviating path. In an `nwhile` statement we check the distance (or angle) and detect if we should process the next command. After executing each command we resample the pose in order to take into account actual driving distance in subsequent moves.

c) The Rover Interface Block: The block *Rover Interface* implements the drivers for sensors and actuators. The *wheel velocities* are measured by encoders, already supplied within the Pioneer rover. A built-in microcontroller reads the encoders and sends their value to the `Carma Devkit`. Additionally the rover is equipped with an *inertial measurement unit (IMU)* including an accelerometer, measuring the linear acceleration, and a gyroscope, measuring the angular velocity of the rover. The `Raspberry Pi` mounted on top of the rover samples the accelerometer and gyroscope, and forwards the raw IMU measurements to the `Carma Devkit`. The rover is controlled by the incoming velocity commands.

d) The Sensor Fusion Block: The current pose is observed by sensors, which suffer from uncertainty. Measurements are distorted by noise, e.g., caused by fluctuations of the elements of the electrical circuit of the sensors. The environment may be unpredictable, e.g., the rover may slip over water or ice when parking. To overcome such problems sensor fusion techniques are applied, i.e., several sensors are combined to estimate a more accurate state. A common method is state estimation (also called *filtering*) [23], [24]. In this application, an *unscented Kalman filter (UKF)* [26] is used. This filter combines the measurements from sensors with a dynamics model describing the relations from the measured variables to the pose of the rover.

We implemented the architecture described above using Robot Operating System (ROS). ROS [28] is a meta-operating system that provides common functionality for robotic tasks including process communication, package management, and hardware abstraction. The system components (see Fig. 6) are implemented as ROS-nodes, which communicate with each other by passing messages. Our approach differs from existing approaches described in Section II in a way that the GBN block uses information about the current state of the rover provided by a sensor fusion to issue a command which takes into account the environment.

2) Parameters of the GBN: After the learning phase, we obtain the parameters of the GBN that we use in the program skeleton (Table. I). Since we track the position using the data from the sensor fusion and each movement has the experimentally learned uncertainty, we are resistive to the perturbation of the actual driving distances and angles.

–		b_{21}	0.7968	b_{32}	-0.2086	b_{43}	0.5475
σ_1^2	0.0062	σ_2^2	0.0032	σ_3^2	0.0019	σ_4^2	0.022
b_{54}	-0.0045	b_{65}	1.1920	b_{76}	-0.0968		
σ_5^2	0.0008	σ_6^2	0.0178	σ_7^2	0.0013		

TABLE I
BN VARIANCES AND COEFFICIENT DEPENDENCES.

B. Tap withdrawal circuit of C.elegans

In the second case study we apply NPs for simulation of neuron activity of *Caenorhabditis elegans* (*C. elegans*), a nematode, whose adult individual has exactly 302 neural cells. The absence of variability in the neural structure, relatively low number of cells made *C. elegans* a perfect testbed in gerontology, developmental, evolutional and neuro- biology, and, recently, in formal verification [29].

Wicks et al. in [9] presented a neural model of a tap withdrawal circuit (Fig. 7). Three mechanosensory neurons (`AVM, PLM, ALM`) can be stimulated with external current I_{stim}. The fourth sensory neuron `PVD` reacts on harsh touch or cold temperatures and is not stimulated in our case. The stimuli propagate through the network (see Fig. 7) via currents towards the cells responsible for the locomotion: `PVC` and `AVB` for the forward movement; `AVA` and `AVD` for the backward movement. `DVA` plays the role in maintenance of the overall activity of the circuit.

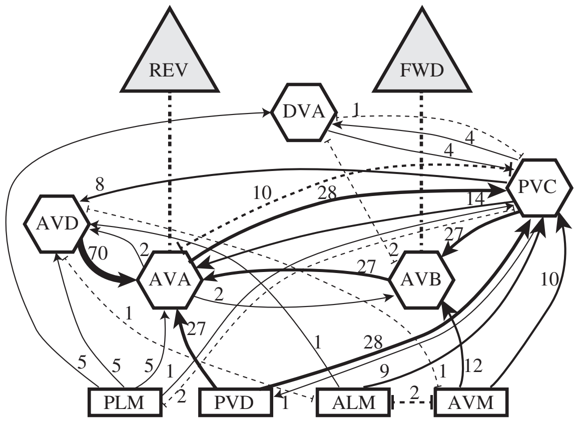


Fig. 7. Tap withdrawal circuit of *c.elegans*

The model by Wicks [9] defines membrane potentials and currents of a neuron i according to equations 14 - 17:

$$\frac{dV^{(i)}}{dt} = \frac{V_{Leak} - V^{(i)}}{R_m^{(i)} C_m^{(i)}} + \frac{\sum_{j=1}^{N} (I_{syn}^{(ij)} + I_{gap}^{(ij)}) + I_{stim}^{(i)}}{C_m^{(i)}} \quad (14)$$

$$I_{gap}^{(ij)} = w_{gap}^{(ij)} \, g_{gap}^{(ij)} \, (V_j - V_i) \quad (15)$$

$$I_{syn}^{(ij)} = w_{syn}^{(ij)} \, g_{syn}^{(ij)}(V^{(i)}) \, (E^{(ij)} - V^{(j)}) \quad (16)$$

$$g_{syn}^{(ij)}(V^{(j)}) = \frac{\bar{g}_{syn}}{1 + e^{K\left(\frac{V^{(j)} - V_{eq_j}}{V_{range}}\right)}} \quad (17)$$

Change of a membrane potential $dV^{(i)}$ of the neuron i is affected by all the currents flowing into the neuron, current potential $V^{(i)}$ and leakage potential V_{Leak} of the cell. Resistance $R_m^{(i)}$ and conductance $C_m^{(i)}$ of a membrane of each neuron characterize the strength of a dependence on voltage and current. The neurons have two types of connections:

synaptic and gap junction (respectively solid and dashed lines on Fig. 7). Gap junction current $I_{gap}^{(ij)}$ from a neuron j to a neuron i is characterized by a difference of membrane potentials, constant conductance $g_{gap}^{(ij)}$, and number of gap junctions $w_{gap}^{(ij)}$. This current characterises instantaneous resistive connection between neurons and comprises a linear combination of inputs. Synaptic current $I_{syn}^{(ij)}$ is of chemical nature, characterized by a weight $w_{syn}^{(ij)}$ or a number of synaptic connections from neuron i to neuron j, conductance $g_{syn}^{(ij)}(V^{(i)})$ and difference of potentials. Synaptic conductance $g_{syn}^{(ij)}(V^{(i)})$ has a sigmoid shape, parametrized by constant K, presynaptic voltage range V_{range}, and equilibrium point V_{eq}.

The output of Wicks' model characterizes the direction of movement, either *forward* or *backward* after applying stimulus current to mechanosensory neurons. The continuous model above shows an average behavior of the circuit. In [30] the authors claim that biological processes possess inherent stochasticity and should be modeled using appropriate tools. Given the network structure on Fig. 7 we rewrote differential equations as a neural program and captured behavior of each synaptic connection (Eq. 17) with a `nif` statement: we also claim our method better reflects the reality since a decision for each synaptic connection if it is open or closed is made probabilistically based on a semantics of a `nif` statement and takes into account inherent noise in each neuron. Our simulation produces on average the same results as a deterministic model (Table II).

The following pseudo code expresses the operation of the tap withdrawal circuit of *C.elegans* as a neural program. For simplicity the computation only for one neuron is shown, and all the neurons from Fig. 7 compute in parallel. For each time step we compute currents flowing from one neuron to another and the difference of membrane potentials. Whenever the exact number of operations is required, we exploit the limit case of `nwhile` statement with zero uncertainty. Table II shows the difference between ODE simulation and average output of a neural program as well as time for executing a simulation in MATLAB and C++. The row $*$ in Table II denotes stimulation of all three input neurons.

```
1: **nwhile** ( $t \le t_{dur}$, 0)
2:     compute $I_{gap}^{(ij)}$ using equation 15
3:     **nwhile** ( $k \le w_{syn}^{(ij)}$, 0)
4:         **nif** ($V^{(j)} \le V_{eq}$, $K/V_{range}$)
5:             $g_{syn}^{(ij)} \leftarrow g_{syn}^{(ij)} + g_{syn}$
6:     compute $I_{syn}^{(ij)}$ using equation 16
7:     compute $dV^{(i)}$ using equation 14
8:     $V^{(i)} \leftarrow V^{(i)} + dV^{(i)}$
9:     $t \leftarrow t + dt$
```

NEURON	ODE.OUT	AV.OUT	DIFF.%	t_{MATLAB}	$t_{\text{C++}}$, s
AVM	-0.0326	-0.0330	1.35	59.06	0.395
ALM	-0.0314	-0.0320	2.00	59.06	0.403
PLM	0.0807	0.0811	0.38	59.04	0.418
$\star$	-0.0272	-0.0266	2.25	60.12	0.420

TABLE II
COMPARISON OF SIMULATION RESULTS

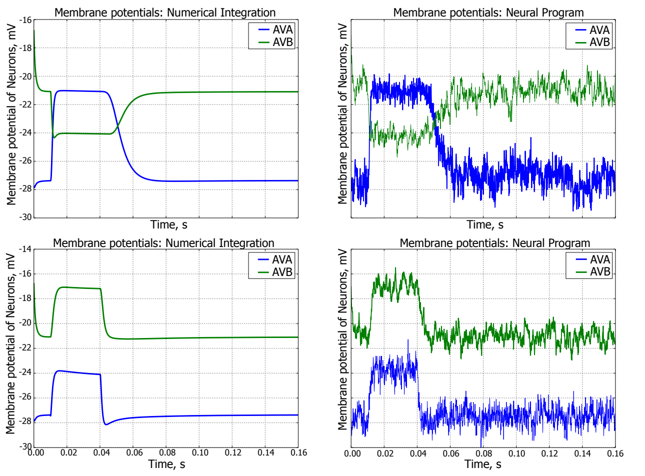


Fig. 8. Deterministic simulations of potentials of AVA and AVB neurons (left); stochastic simulations using `nif` condition (right)

C. Parallel Parking as a Neural Circuit

In this section we draw analogy between the two case studies and express an adaptive proportional controller for parallel parking as a neural circuit. We define the following variable mapping between the two case studies: (i) voltages $V^{(i)}$ from the second case study are mapped to the distances x and turning angles θ in the first case study; (ii) synaptic currents $I_{syn}^{(ij)}$ from the neural circuit are mapped to the linear v and angular velocities ω in the first case study; (iii) synaptic conductance $g_{syn}^{(ij)}$ defines a control flow in parallel parking controller; (iv) synaptic potentials $E^{(ij)}$ are mapped to the target locations (thresholds) $l^{(i)}$ and $\alpha^{(i)}$. Neurons have only synaptic connections (see Fig. 9).

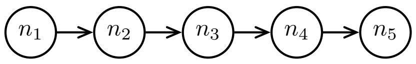

$$\begin{tikzcd} n_1 \arrow[r] & n_2 \arrow[r] & n_3 \arrow[r] & n_4 \arrow[r] & n_5 \end{tikzcd}$$

Fig. 9. Controller for parallel parking as a neural circuit

Linear and angular velocities are defined as follows:

$$\frac{dx^{(1)}}{dt} = w^{(1)}(l^{(1)} - x^{(1)}) \tag{18}$$

$$\frac{d\theta^{(i)}}{dt} = w^{(i,i-1)}\ g^{(i,i-1)}(x^{(i)})\ (\alpha^{(i)} - \theta^{(i)}) \tag{19}$$

$$\frac{dx^{(i)}}{dt} = w^{(i,i-1)}\ g^{(i,i-1)}\ (\theta^{(i)})(l^{(i)} - x^{(i)}) \tag{20}$$

$$g^{(i,i-1)}(\theta^{(i)}) = \text{nif}(\theta^{(i)} > \alpha^{(i)}, \sigma_i^2)\quad 1 \quad \text{else} \quad 0 \tag{21}$$

The circuit (see Fig. 9) acts as a proportional controller with proportionality constant $w^{(i,i-1)}$: the velocity of the rover is proportional to the difference between current and target locations (i.e. the error). A neuron i executes its corresponding control policy (Eq. 18 for the first neuron, Eq. 19 and Eq. 20 for even and odd neurons respectively). The controller operates as follows: at a starting time point the neuron n_1 is active. Each neuron represents the one motion primitive in parallel parking procedure. First neuron is active until $(x_1 > l_1, \sigma_1^2)$ evaluates to 0 according to `nif` semantics, and then $g^{(21)}$ fires the second neuron. The second neuron stays active until nif$(\theta_1 > \alpha_1, \sigma_2^2)$ evaluates to 0 and so on. A neuron i is activated by synaptic conductance $g^{(i,i-1)}$ and stays active until the rover reaches the corresponding target location up to a variance σ_i^2.

VI. Conclusions

In this paper we introduced *neural programming* (NP), a new formalism for writing robust and adaptive cyber-physical-system (CPS) controllers. Key to this formalism is: (i) the use of smooth probability distributions in conditional and loop statements, instead of their classic stepwise counterparts; and (ii) the use of Gaussian Bayesian networks for capturing the dependencies among probability distributions.

We validated the utility of NP by developing a parallel-parking CPS controller that is able to adapt to unforeseen environmental situations, such as a slippery ground or noisy actuators. No classic program has such an ability: one would have to encode all these unforeseen situations, which would lead to unintelligible code. Simulation of the tap-withdrawal response of *C. elegans* as an NP program yields the same result as the differential-equations model and allows one to directly address the stochastic nature of each synaptic connection between two neurons. We also showed analogy between case studies by expressing a proportional controller of the first problem as a neural circuit.

As future work, we plan to explore the advantages of NP in the analysis as well as the design (optimization) of CPS controllers. The desirable mathematical properties of NP programs makes them an ideal formalism for these tasks.

Acknowledgements. Research supported in part by the project HARMONIA (845631), funded by a national Austrian grant from FFG (Österreichische Forschungsförderungsgesellschaft) under the program IKT der Zukunft; Austrian National Research Network S 11405-N23 and S 11412-N23 (RiSE/SHiNE) of the Austrian Science Fund (FWF), the Doctoral Program Logical Methods in Computer Science of the Austrian Science Fund (FWF). AFOSR grant AFOSR FA9550-14-1-0261, and NSF grants CCF-0926190 and IIS-1447549.

References

[1] M. Broy and E. Geisberger, "Cyber-physical systems, driving force for innovation in mobility, health, energy and production," *Acatech: The National Academy Of Science and Engineering*, 2012.

[2] D. L. Parnas, "Software aspects of strategic defense systems," *Commun. ACM*, vol. 28, no. 12, pp. 1326–1335, Dec. 1985.

[3] S. Chaudhuri and A. Solar-Lezama, "Smooth interpretation." in *PLDI*, 2010, pp. 279–291.

[4] S. Chaudhuri and A. S. Lezama, "Smoothing a program soundly and robustly." in *CAV*, 2011, pp. 277–292.

[5] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 3642–3649.

[6] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Mar. 2010.

[7] L. Cao, C. Ma and Y. Xu, "Adaptive control theory and applications,," in *Journal of Control Science and Engineering, vol. 2012, Article ID 827353*, vol. 1, 2012, pp. 1–2.

[8] S. Chaudhuri and A. Solar-Lezama, "Smooth Interpretation: Presentation Slides," http://people.csail.mit.edu/asolar/Talks/PLDI2010Final.pptx *(Accessed 14.03.2015).*

[9] S. R. Wicks, C. J. Roehrig, and C. H. Rankin, "A dynamic network simulation of the nematode tap withdrawal circuit - Predictions concerning synaptic function using behavioral criteria." *Journal of Neuroscience*, vol. 16, pp. 4017–4031, 1996.

[10] "Parking Videos," http://youtu.be/xNOj_ARSEYs?list=PLP5Gx6r7gK2cxjKv0K2V5fBedovfo8_3y *(Accessed 12.03.2015).*

[11] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, "Probabilistic Programming," in *International Conference on Software Engineering (ICSE Future of Software Engineering).* IEEE, May 2014.

[12] R. E. Neapolitan, *Learning Bayesian Networks*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003.

[13] D. Heckerman and D. Geiger, "Learning bayesian networks: A unification for discrete and gaussian domains," in *UAI*, 1995, pp. 274–284.

[14] M.-A. Ibarra-Manzano, J.-H. De-Anda-Cuellar, C.-A. Perez-Ramirez, O.-I. Vera-Almanza, F.-J. Mendoza-Galindo, M.-A. Carbajal-Guillen, and D.-L. Almanza-Ojeda, "Intelligent algorithm for parallel self-parking assist of a mobile robot," in *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2012 IEEE Ninth*, Nov 2012, pp. 37–41.

[15] K. Jiang and L. Seneviratne, "A sensor guided autonomous parking system for nonholonomic mobile robots," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, 1999, pp. 311–316 vol.1.

[16] M. Khoshnejad and K. Demirli, "Autonomous parallel parking of a car-like mobile robot by a neuro-fuzzy behavior-based controller," in *Fuzzy Information Processing Society, 2005. NAFIPS 2005. Annual Meeting of the North American*, June 2005, pp. 814–819.

[17] N. Scicluna, E. Gatt, O. Casha, I. Grech, and J. Micallef, "Fpga-based autonomous parking of a car-like robot using fuzzy logic control," in *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, Dec 2012, pp. 229–232.

[18] T. Li, Y.-C. Yeh, J.-D. Wu, M.-Y. Hsiao, and C.-Y. Chen, "Multifunctional Intelligent Autonomous Parking Controllers for Carlike Mobile Robots," *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 5, pp. 1687–1700, May 2010.

[19] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *Int. J. Rob. Res.*, vol. 29, no. 13, pp. 1608–1639, 2010.

[20] K. M. Passino and S. Yurkovich, *Fuzzy Control*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[21] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. L. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013.* IEEE, 2013, pp. 1–10.

[22] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha, "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013.* IEEE, 2013, pp. 1–10.

[23] H. Mitchell, *Multi-Sensor Data Fusion - An Introduction*. Berlin, Heidelberg, New York: Springer, 2007.

[24] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge: MIT Press, 2006.

[25] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[26] E. Wan and R. Van der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2000, pp. 153–158.

[27] Adept MobileRobots(2013). Pioneer 3-AT. http://www.mobilerobots.com/ *(Accessed 24.02.2015).*

[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[29] M. A. Islam, R. DeFrancisco, C. Fan, R. Grosu, S. Mitra, and S. A. Smolka, "Model Checking Tap Withdrawal in C. Elegans," *ArXiv e-prints*, Mar. 2015.

[30] S. Ditlevsen and A. Samson, *Introduction to stochastic models in biology*, ser. Lecture Notes in Mathematics. Springer, 2013, pp. 3–34, 2013; 1.