ARDL - A Library for Adaptive Robotic Dynamics Learning

Joshua Smith JOSHUA.SMITH@ED.AC.UK and **Michael Mistry** MMISTRY@INF.ED.AC.UK
School of Informatics, University of Edinburgh, UK

Editors: A. Jadbabaie, J. Lygeros, G. J. Pappas, P. A. Parrilo, B. Recht, C. J. Tomlin, M. N. Zeilinger

Abstract

Dynamics learning and adaptive control algorithms have received a lack of support from robot dynamics libraries over the years. Only a few existing libraries like Pinocchio implement the standard regressor for basic model learning. In this work we introduce an open-source dynamics library specifically designed to provide support for dynamics learning and online adaptive control algorithms. Alongside established kinematics and dynamics computations, our new dynamics library provides computation for the standard, the Slotine-Li and the filtered regressor matrices found in adaptive control algorithms. We demonstrate the library through several existing adaptive control algorithms, alongside a new online simultaneous Semi-Parametric model using a Radial Basis Function Neural Network augmented with a newly derived consistency transform.

1. Introduction

Robot dynamics modelling involves learning the forces that occur on a robot platform given a state. Such models are used in control and planning frameworks when we are able to drive the robot through torque signals. Often these controllers are limited in performance due the inaccuracy of the dynamics model, which is especially an issue on real platforms. Even if we can accurately ascertain inertial parameters from CAD estimation, often there are irregularities between manufactured platforms, unidentifiable effects from friction or wires, and the effects of wear and tear. By learning the model directly from data we can rectify these issues on individual platforms.

Robot dynamics model learning typically falls into one of three categories: Parametric, Non-Parametric and Semi-Parametric. Parametric models require specific reformulations of rigid body dynamics to isolate relevant inertial parameters of each body. These models have the advantage of being state generalisable over the entire state space, however, they suffer in accuracy from unmodelled forces. Alternatively, Non-Parametric models are implemented using pure machine learning techniques which treat the problem as a function mapping from the input to the outputs. They tend to have good accuracy (given good data) as they do not rely on a specific formulation based on physics. However, they do suffer from a lack generalisation when the input state is not close to the data initially trained on. Semi-Parametric models are a hybrid approach that combines the best attributes of both. They can exhibit better accuracy and improved generalisation, however, training of Semi-Parametric models tend to be much more complicated in order to achieve these benefits.

Adaptive dynamics model learning approaches, Middletone and Goodwin (1986); Slotine and Li (1989); Pucci et al. (2015); Sanner and Slotine (1995), effectively combine the task of learning the dynamics with providing control to the robot. Adaptive algorithms require the capability of running online within a control loop and often have to deal with long-term learning and noise from the sensors. Research in adaptive control have heavily revolved around passive stability of both the learning and control. Slotine and Li (1989) is an example of a Parametric adaptive control solution with passive stability that guarantees position, velocity and parameter convergence. In order to provide a stability proof, the controller requires the Slotine-Li regressor which maintains the skew-symmetry between the time derivative of the Inertia matrix and the Coriolis matrix, $x^T(\dot{\mathbf{M}}-2\mathbf{C})x = 0$ where x is any vector. The parameter convergence is guaranteed through the use of the filtered regressor, used to avoid using noisy joint acceleration estimation and measurements.

To implement these models and algorithms we require the ability to compute the various dynamic and regressor matrices, which is functionality typically reserved for dynamics libraries. Three examples that are popular within the robotics community are RBDL, IDyntree, and Pinocchio. RBDL, Felis (2017), is one of the most mature dynamics libraries implementing the spatial algorithms in Featherstone (2008). Due to the design of the software, RBDL is efficient in calculating various dynamics quantities using O(n) algorithms. IDyntree, Nori et al. (2015), follows a similar route to RBDL and more specifically acts as a floating base extension to the Kinematics and Dynamics Library (KDL) introduced with Orocos. It takes advantage of the same spatial notation as with RBDL and is heavily used with the iCub robot platform. Pinocchio is a more recently introduced dynamics library, Carpentier et al. (2015–2019, 2019), and has been designed to take advantage of spatial notation with Lie group representations as in RBDL. Pinocchio also implements several programming patterns to increase the efficiency of the generated programs. The two major additions they introduce are the use of the CRTP pattern which aims to solely implement static polymorphism rather than dynamic polymorphism, and the use of autodiff libraries, allowing users to automatically compute the desired derivatives of various quantities without having to manually construct them.

From all of these libraries only IDyntree and Pinocchio implement any of the regressors and they are limited to, at the time of writing, computing the standard regressor. Whilst these can be used for the algorithms, the data structures behind the inertial parameters tend to be more difficult to update due to being defined as constants or being difficult to isolate without good knowledge of the software. The lack of support means that many papers on adaptive dynamics control often have closed source custom software, which is normally not readily accessible and increases work for researchers implementing the algorithm or re-implementing for verification purposes. Some examples of the lack of open-source software are in the likes of Middletone and Goodwin (1986); Slotine and Li (1989); Yuan and Yuan (1995); Cheah et al. (2006); Wensing and Slotine (2018).

As such the contribution of this paper is introducing a new dynamics library, ARDL[1], focused on supporting adaptive learning algorithms for dynamics. ARDL mainly focuses on providing calculations for all the regressor matrices, allowing clear access to the inertial parameters to allow updating, and maintaining performance throughout. We will demonstrate the library in action with several preexisting adaptive control algorithms.

Furthermore, the concept of Semi-Parametric consistency was introduced in Smith and Mistry (2020) for online learning of the dynamics model using a Semi-Parametric model. We specifically identified a local adaptation that when applied with a Gaussian Mixture Model in a Semi-Parametric framework allows online simultaneous updates. In this work we further emphasise the generality of the consistency adaptation by presenting an online simultaneous Semi-Parametric Radial Basis Function Neural Network model using the ARDL library.

1. https://github.com/smithjoshua001/ARDL

2. Theoretical Background

Parametric models, Atkeson et al. (1985); Slotine (1985), are based on the classical articulated rigid body dynamics equation to model the forces of the platform in question.

$$\mathbf{M}(q)\ddot{q} + \mathbf{C}(q,\dot{q})\dot{q} + g(q) = \tau \quad (1) \qquad\qquad \mathbf{Y}(q,\dot{q},\ddot{q})\pi = \tau \quad (2)$$

Where q, $\dot{q}$, $\ddot{q}$, τ are the joint positions, velocities, accelerations and joint torques of the robot. $\mathbf{M}(q)$, $\mathbf{C}(q,\dot{q})$ and $g(q)$ are the inertia matrix, coriolis matrix, and gravity vector respectively. π is the inertial parameters of each link. $\mathbf{Y}(q,\dot{q},\ddot{q})$ is the standard regressor matrix, which is a reformulation of (1).

Atkeson et al. (1985) solves for the inertial parameters, through standard least squares, by stacking multiple observations through the regressor matrix with the stacked torque measurements. Slotine and Weiping Li (1987) provides a stable direct adaptive controller that ensures that the position and velocity errors are driven towards zero in a passively stable fashion. The passivity of the system is ensured through the aforementioned Slotine-Li regressor, ensuring the skew-symmetric property of $x^T(\dot{\mathbf{M}} - 2\mathbf{C})x = 0$, with a sliding mode controller.

In Slotine and Li (1989) they present an composite adaptive controller that is an extension to the direct adaptive controller by adding a term with the torque error to ensure convergence of the inertial parameters. The convergence is achieved using a first order filtered regressor to eliminate the need to use the noisy joint acceleration signals.

Non-Parametric models, Vijayakumar and Schaal (2000); Nguyen-Tuong et al. (2009), typically follow a more Machine Learning approach. They define the model as a function mapping from the robot state to the joint torques.

$$f(q,\dot{q},\ddot{q}) = \tau \tag{3}$$

Radial Basis Function Neural Networks are a class of linearly parameterized neural networks that can be used as a Non-Parametric model, Sanner and Slotine (1992). Specifically they rely on a set of basis functions that span the space of the inputs and associated weights which when combined provide a linear sum of these basis functions which can approximate the desired function. An interesting property of the RBFNN though is they can universally approximate a smooth non-linear function over a compact set, Park and Sandberg (1991).

Typically RBFNNs have an output computation as:

$$f(x) = \mathbf{W}^*\Theta(x) + \epsilon \tag{4}$$

$$\Theta(x) = \begin{bmatrix}\theta_0(x) & \theta_1(x) & \dots & \theta_k(x)\end{bmatrix}^T \tag{5}$$

$$\theta_\bullet(x) = e^{-\beta_\bullet(x-\mu_\bullet)^T(x-\mu_\bullet)} \tag{6}$$

$$\min_{\mathbf{W}} |\tau - \mathbf{W}\Theta(x)| \tag{7}$$

where $\mathbf{W}$ are the weights of the RBFNN network with $\bullet^*$ and $\hat{\bullet}$ indicating the optimal and estimated value of $\bullet$ respectively. $\mu_\bullet$ and $\beta_\bullet$ are the center and coefficient of the basis function in question. ϵ is the approximation error of the RBFNN. x is the input vector for the basis functions.

Given a function $f(x)$ we can define a gradient based update equation for the weight matrix of the RBFNN:

$$\dot{\hat{\mathbf{W}}}^T = \mathbf{P}\Theta^T(x)(f(x) - \hat{\mathbf{W}}\Theta(x)) \qquad (8)$$

$$\dot{\mathbf{P}} = -\frac{\mathbf{P}\Theta(x)\Theta^T(x)\mathbf{P}^T}{1 + \Theta^T(x)\mathbf{P}\Theta(x)} \qquad (9)$$

$\mathbf{P}$ is a positive definite matrix initially user selected, in this paper we initialized to a diagonal matrix with positive coefficients. $\mathbf{P}$ is updated according to (9).

For the purpose of learning the dynamics of a robot we can set x as the state of the robot, $[q, \dot{q}, \ddot{q}]$, and have $f(x) = \tau$. This should drive the estimated weight matrix to the optimal weight matrix and minimise the error between the true and estimated torque (7).

Semi-Parametric models, Nguyen-Tuong et al. (2010); Traversaro et al. (2017); Romeres et al. (2016); Smith and Mistry (2020), combine both the Parametric and Non-Parametric models together to try and take advantage of the benefits of both models. The generalisation from the Parametric model with the accuracy of the Non-Parametric model.

$$\mathbf{Y}(q, \dot{q}, \ddot{q})\pi + f(q, \dot{q}, \ddot{q}) = \tau \qquad (10)$$

Our previous work, Smith and Mistry (2020), uses a Semi-Parametric model based on the Composite Adaptive Controller and a Gaussian Mixture Model (GMM). We also introduced the concept of consistency in a semi-parametric model that simultaneously learns both components. The consistency transform, (12), was introduced to solve the inconsistency that arises in the Non-Parametric target function, (11), when the inertial parameters are updated. The exact formulation of the consistency transform for the GMM can be found in Smith and Mistry (2020).

$$f(q, \dot{q}, \ddot{q}) = \tau - \mathbf{Y}(q, \dot{q}, \ddot{q})\hat{\pi} \qquad (11)$$

$$\Delta f(q, \dot{q}, \ddot{q}) = -\mathbf{Y}(q, \dot{q}, \ddot{q})\Delta\hat{\pi} \qquad (12)$$

3. Main Features

To implement most of the discussed controllers we need access to all the regressors, and the state derivatives. As mentioned in the introduction, current dynamics libraries do not implement the required matrices. To solve this we introduce the open-source ARDL library. ARDL has been designed to implement these regressors and to support adaptive control algorithms.

For the Parametric dynamics we have used work from Garofalo et al. (2013) as a basis for computing all required dynamics matrices, vectors and regressors. Whilst not as efficient as the O(n) algorithms introduced in Featherstone (2008), the equations do provide the necessary foundation for the Slotine-Li regressor, and with some adaptation the filtered regressor.

Though the main focus of the library is not the re-implementation of various Non-Parameteric models, we have provided an iterative Gaussian Mixture Model implementation based on Pinto and Engel (2017). We provide explicit support for computing the required regressors and state-derivatives for applying Semi-Parametric consistency transform for the Gaussian Mixture Model, Smith and Mistry (2020), and as will be shown later the consistency transform for a Semi-Parametric RBFNN model.

To ensure efficient computation, all the equations have been implemented using the Eigen Library Guennebaud et al. (2010), which is a very mature and optimized matrix/vector library commonly used throughout robotics. The inheritance hierarchy and polymorphism has been implemented using the curiously recurring template pattern (CRTP) to force evaluation at compile time rather than run time.

3.1. Online Simultaneous Semi-Parametric Radial Basis Function Neural Network

In addition to the ARDL library we further emphasise the simultaneous consistency adaptation framework. We achieve this by not only replicating the Gaussian Mixture Model Semi-Parametric model from Smith and Mistry (2020), but also applying the framework to a Semi-Parametric model using a Radial Basis Function Neural Network (RBFNN).

3.1.1. SEMI-PARAMETRIC CONSISTENCY ADAPTATION

To use the RBFNN in a semi-parametric setup we can follow a similar approach to Smith and Mistry (2020). This means that instead of learning τ the RBFNN will learn the residual error from the parametric model, (11). When the parametric model updates, the residual error will change accordingly modifying the target function and the optimal weights, creating an inconsistency. This can cause convergence and performance issues, however, due to the structured nature of the parametric model we can compute the change in the non-parametric model and correct the inconsistency. This is achieved generally through (11) and (12).

In the case of RBFNN, we can identify how the optimal weight matrix should change with respect to the inertial parameters through (13), where the inverse is elementwise through $\Theta(x)$.

$$\frac{\partial \mathbf{W}^*}{\partial \pi} = -\mathbf{Y}\Theta^{-1}(x) \tag{13}$$

Knowing how the optimal weights change though is not particularly useful as the optimal weights are not typically known before training. Instead the focus should be on computing the change in the estimated weight matrix with respect to the inertial parameters. This can be achieved by taking the partial derivative of (8), with f(x) replaced with the residual error from the parametric model, and updating the partial derivative alongside updating the weight matrix normally.

$$\frac{\partial \dot{\hat{\mathbf{W}}}^T}{\partial \pi} = \mathbf{P}\Theta^T(x)\left(-\mathbf{Y} - \frac{\partial \hat{\mathbf{W}}}{\partial \pi}\Theta(x)\right) \tag{14}$$

When substituted into (12), we obtain how a change in the inertial parameters will change the estimated weights of the network. The update to the weights will keep the network consistent with the new target function whilst reusing the previous data.

4. Results

There are some other libraries that support the computation of some of the regressors needed for adaptive algorithms, primarily iDynTree and Pinocchio. Both these libraries only support computation of the standard regressor matrix.

We will compare the computation time of the regressor matrix between the ARDL and Pinocchio libraries. We did not compare against IDynTree due to the regressor generator refusing to compile in the latest version. The compilation errors are most likely due to the fact that the code is legacy and not maintained within the project. We shall use a variable length Kuka LWR IV arm as the test

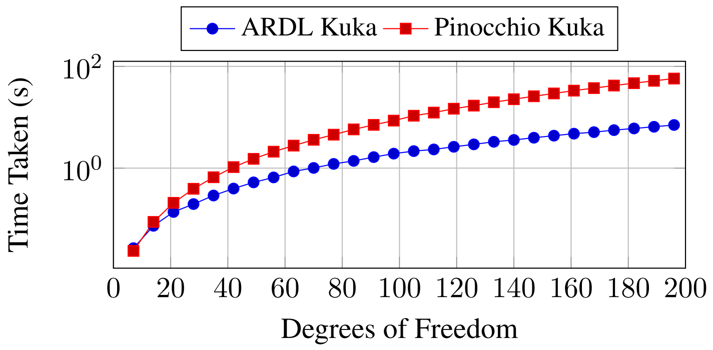


Figure 1: Time taken to compute 5000 standard regressor matrices for a variable length stacked Kuka Lwr IV manipulator.

model, which we generate by chaining multiple arms together via a fixed joint. The variable length aspect allows us to judge the scalability of the regressor computation.

The test was performed on Xubuntu 18.04 using a i7-7700HQ, set with a performance governor at 3.5Ghz, with 32GB 2133Mhz DDR4 RAM. All compiled with Clang 11 using the flags "-O3 -march=native".

To isolate small changes in timing due to scheduling on the CPU we compute the standard regressor of the model 5000 times at random configurations. We can see the time taken for each model computing 5000 random regressors in figure 1. It is clear at low dimensionality the differences between the two libraries are negligible. As the dimensionality of the model are increased the performance diverges up to about a factor of 10, in favour of ARDL, at 200 degrees of freedom. The most likely main factor of the difference between the two libraries arises through the algorithms used to generate the regressor.

4.1. Simulated Examples

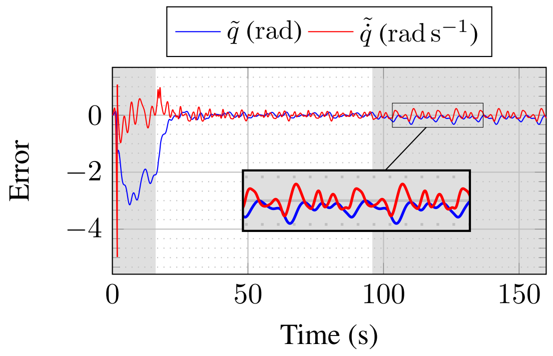
To demonstrate the effectiveness of this library several examples of adaptive model learning have been included. These include the Direct Adaptive Controller from Slotine and Weiping Li (1987), the Composite Adaptive Controller from Slotine and Li (1989), and the Online Simultaneous Semi-Parametric Model introduced in Smith and Mistry (2020).

We implement all the controllers using ARDL and execute them on a Mujoco simulation of a Kuka LWR 4 manipulator. An example of the control and learning functions of the Composite Adaptive Controller using ARDL is given in listing 1. Using these proven examples we can verify the correctness of the various quantities used in their calculations. The trajectory for all examples are the same with all models initialised to zero. We show only the results for joint 1 of the simulated robot due to space restrictions and due to the fact the majority of the forces act on this joint.

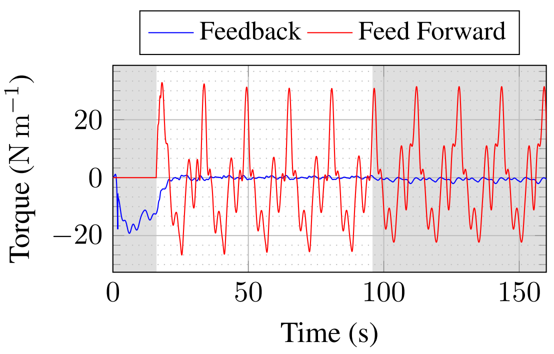
4.1.1. DIRECT ADAPTIVE CONTROL

Figures 2(*a*) and 2(*b*) show the position and velocity errors, and the feedback and feed forward torques of the controller respectively. The greyed out portions of the graphs indicate areas where no learning is performed. These areas are to demonstrate the initial model performance and the final model performance.

From figure 2(*a*) we can see that the position and velocity errors are reduced to a range of -0.2 and 0.2. More importantly we can see that the feedback torque in figure 2(*b*) are minimised only



(*a*) Position and Velocity errors of joint 1

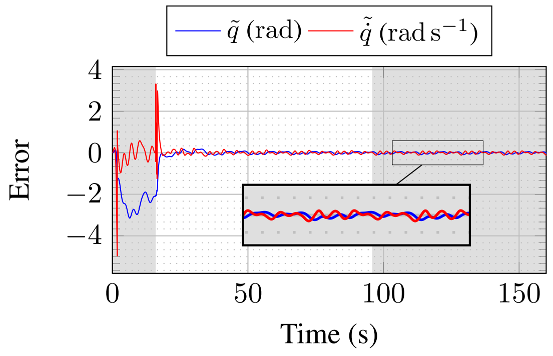


(*b*) Feed forward and Feedback contributions, of joint 1

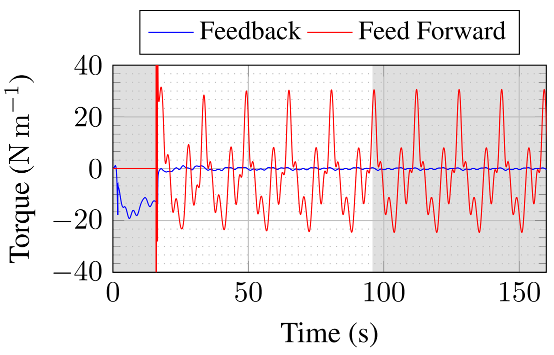
Figure 2: Direct Adaptive Controller - The learning is performed between 16 and 96 seconds, beforehand is only controlled via the feedback gains of the controller.

fluctuating slightly. The fluctuation is increased slightly once the training is finished due to the fact that the inertial parameters aren't guaranteed to converge and in this case they do not.

4.1.2. COMPOSITE ADAPTIVE CONTROL



(*a*) Position and Velocity errors of joint 1

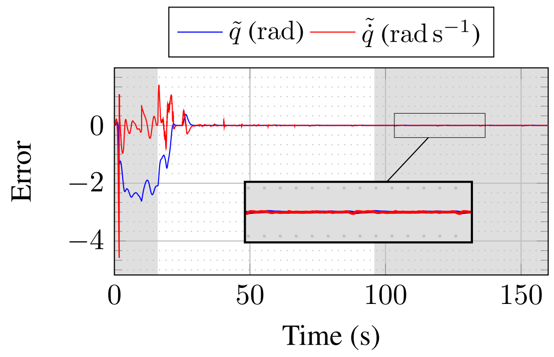


(*b*) Feed forward and Feedback contributions of joint 1

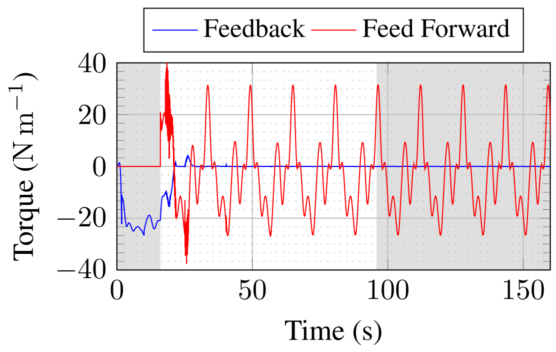
Figure 3: Composite Adaptive Controller - The learning is performed between 16 and 96 seconds, beforehand is only controlled via the feedback gains of the controller.

Figures 3(*a*) and 3(*b*) show the same as figures 2(*a*) and 2(*b*) respectively. The position and velocity errors as before are reduced significantly to between -0.1 and 0.1. The same is said for the feedback torque. A notable difference between the direct and composite approaches is that after the learning is finished the feedback and feed forward torques do not increase or change much in pattern. This can be due to the fact that the inertial parameters converge to a more stable result.

4.1.3. ONLINE SIMULTANEOUS SEMI PARAMETRIC GMM CONTROL



(*a*) Position and Velocity errors of joint 1

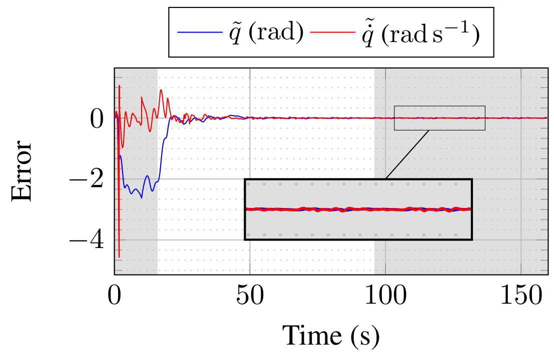


(*b*) Feed forward and Feedback contributions of joint 1

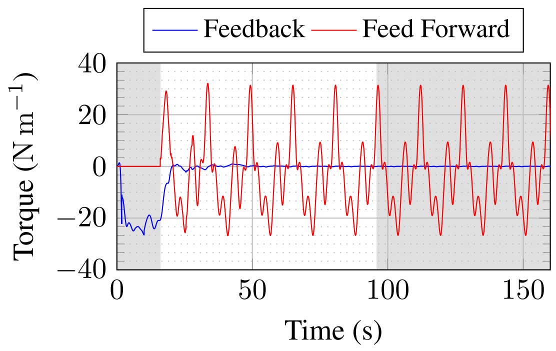
Figure 4: Semi-Parametric controller in Smith and Mistry (2020) - The learning is performed between 16 and 96 seconds, beforehand is only controlled via the feedback gains of the controller.

Implementing the model and controller in Smith and Mistry (2020) we can plot similar results as before. Figure 4(*a*) shows that the errors are reduced significantly and are practically negligible. Figure 4(*b*) shows that the feedback torque are also minimised more than in the case of the direct and composite adaptive controllers. This is achieved due to the fact the non-parametric Gaussian Mixture Model can model the friction forces.

4.1.4. ONLINE SIMULTANEOUS SEMI-PARAMETRIC RBFNN



(*a*) Position and Velocity errors of joint 1



(*b*) Feed forward and Feedback contributions of joint 1

Figure 5: Semi-Parametric RBFNN controller - The learning is performed between 16 and 96 seconds, beforehand is only controlled via the feedback gains of the controller.

We also implemented the RBFNN version of the online simultaneous semi-parametric control described in section 3.1. We implement the RBFNN using CUDA on a Nvidia GTX 1070 GPU, using the cuBLAS and cuTensor libraries provided by Nvidia. We use a total of 2048 basis functions randomly spread over the input space, $(q, \dot{q}, \ddot{q})$, of the robot using sobol noise from the cuRand library.

The initial $\hat{\mathbf{W}}$ is set to zero with the partial derivative with respect to π set to zero as well, as no data has been introduced so the weights will not change with π.

When repeating the same experiments as before we can plot the results in figures 5(*a*) and 5(*b*). The results are very similar to figures 4(*a*) and 4(*b*) where the errors for position, velocity and torque are minimized to a larger degree than in the direct and composite controllers.

5. Conclusion

We have introduced the Adaptive Robotic Dynamics Learning library in this work. This library is designed to be run-time efficient capable of running an online adaptive control algorithm. This was achieved through the use of efficient formulations of the required matrices, with minimal computation where structure can be exploited.

We have demonstrated the ability of the library through the implementation of several examples of adaptive control. Each example demonstrates the correct and expected behaviour of the relevant controller.

An online simultaneous semi-parametric RBFNN algorithm has been introduced to emphasise the consistency framework along with integration with the ARDL library. We introduce the relevant transform of the estimated weights of the network and run the controller on the mujoco simulated setup. We show similar performance to the GMM version introduced in Smith and Mistry (2020) with errors in position, velocity, and torque reduced significantly from the Parametric models.

Currently this library is designed and limited to work on a single fixed chain, however, the next step of development will be to extend this to tree-like robots and floating-base. The extension should be a trivial extension due to the use of spatial algebra and the generic formulation of the joints as in Garofalo et al. (2013). We can also look to expanding our focus into how this work can also fit into other robotics fields requiring the regressors such as model-based reinforcement learning, Morimoto et al. (2003); Abbeel et al. (2006).

Listing 1: Composite adaptive controller example

```
void control(const Eigen::VectorXd& q, const Eigen::VectorXd& qd,
const Eigen::VectorXd& q_des,const Eigen::VectorXd& qd_des,
const Eigen::VectorXd& qdd_des, Eigen::VectorXd& tau){
    //compute sliding mode variables
    Eigen::VectorXd qd_r = qd_des - Lambda.asDiagonal()*(q-q_des);
    Eigen::VectorXd qdd_r = qdd_des - Lambda.asDiagonal()*(qd-qd_des);
    Eigen::VectorXd s = qd - qd_r;
    //update the internal data of the robot chain
    c->updateChain(q,qd);
    //update the internal matrices in the chain
    c->updateMatrices();
    //compute the poses of each link
    fk->getBodyPoses(poses);
    //compute the jacobians of each joint
    fk->getJacobians(poses,jacs);
    //compute the velocity of each link
    fk->getBodyVelocities(velocities,jacs);
    //compute the time derivative of each jacobian
    fk->getJacobianDots(poses,velocities,jacs,jacDots);
    //compute the Slotine-Li regressor
    dyn->calcSlotineLiRegressor(qd_r,qdd_r,poses,velocities,
        jacs,jacDots,Y);
    //Project the regressor to use the base parameters
    tau = Y*dyn->getRegressorProjector()*pi;
    //Add feedback control into the control signal
    tau -= Damping.asDiagonal()*s;
}

void learn(const Eigen::VectorXd& q,const Eigen::VectorXd& qd,
const Eigen::VectorXd& qdd,const Eigen::VectorXd& measured){
    if(!init){
      //Initialise the filters for the filtered regressor and torque filter
      dyn->initFilteredSlotineLiRegressor(jacs, jacDots, velocities, poses,
          qd, filt, W);
      tau_filter->setBuffer(measured);
      init = true;
    }else{
      //Update the filtered regressor and measured torque filter
      dyn->computeFilteredSlotineLiRegressor(jacs, jacDots, velocities, poses,
          qd, filt, W);
      tau_filter->compute(measured);
    }
    //Compute forgetting factor from Slotine and Li (1989)
    lambda =lambda0*(1.0-(Rho.inverse().colwise().lpNorm<1>().maxCoeff())/k0);
    //Add current regressor excitation to the weight matrix Rho to make sure
    //the norm does not exceed a value of k1
    if(Rho.inverse().colwise().lpNorm<1>().maxCoeff()>k1){
      Rho += (W*dyn->getRegressorProjector()).transpose()*
          (W*dyn->getRegressorProjector())*0.001;
    }else{
      Rho += (-lambda0*Rho +((W*dyn->getRegressorProjector()).transpose()*
          (W*dyn->getRegressorProjector())))*0.001;
    }
    //update the inertial parameters using (14) in Slotine and Li (1989)
    pi -= 0.001*(Rho).inverse() * ((Y*dyn->getRegressorProjector()).transpose()
        *s+(W*dyn->getRegressorProjector()).transpose()
        *R.asDiagonal()*(W*dyn->getRegressorProjector()*pi
        -tau_filter->getResult()));
}
```

Acknowledgments

This work has been supported by the following grants: EPSRC UK RAI Hub ORCA (EP/R026173/1) and NCNR (EPR02572X/1), CogIMon project in the EU Horizon 2020 (ICT-23-2014), THING project in the EU Horizon 2020 (ICT-2017-1), and by grant EP/L016834/1 for the University of Edinburgh RAS CDT from EPSRC.

References

Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8, 2006.

Chirstopher G. Atkeson, Chae H. An, and John M. Hollerbach. Rigid body load identification for manipulators. In *1985 24th IEEE Conference on Decision and Control*, volume 24, 1985. doi: 10.1109/CDC.1985.268649.

Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio, 2015–2019.

Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *Proceedings of the 2019 IEEE/SICE International Symposium on System Integration, SII 2019*, pages 614–619. Institute of Electrical and Electronics Engineers Inc., 4 2019. ISBN 9781538636152. doi: 10.1109/SII.2019.8700380.

C. C. Cheah, C. Liu, and Jean-Jacques Slotine. Adaptive Jacobian Tracking Control of Robots With Uncertainties in Kinematic, Dynamic and Actuator Models. *IEEE Transactions on Automatic Control*, 51(6):1024–1029, 6 2006. ISSN 0018-9286. doi: 10.1109/TAC.2006.876943. URL `http://ieeexplore.ieee.org/document/1643374/`.

Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer US, Boston, MA, 2008. ISBN 978-0-387-74314-1. doi: 10.1007/978-1-4899-7560-7.

Martin L. Felis. RBDL: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 41(2):495–511, 2 2017. ISSN 0929-5593. doi: 10.1007/s10514-016-9574-0. URL `http://link.springer.com/10.1007/s10514-016-9574-0`.

Gianluca Garofalo, Christian Ott, Alin Albu-Schäffer, and Alin Albu-Schaffer. On the closed form computation of the dynamic matrices and their differentiations. *IEEE International Conference on Intelligent Robots and Systems*, pages 2364–2369, 11 2013. ISSN 21530858. doi: 10.1109/IROS.2013.6696688. URL `http://ieeexplore.ieee.org/document/6696688/`.

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

R. Middletone and G. Goodwin. Adaptive computed torque control for rigid link manipulators. In *1986 25th IEEE Conference on Decision and Control*, pages 68–73. IEEE, 12 1986. doi: 10.1109/CDC.1986.267156. URL `http://ieeexplore.ieee.org/document/4048708/`.

J. Morimoto, G. Zeglin, and C. G. Atkeson. Minimax differential dynamic programming: application to a biped walking robot. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 2, pages 1927–1932 vol.2, 2003. doi: 10.1109/IROS.2003.1248926.

Duy Nguyen-Tuong, Matthias Seeger, and Jan Peters. Model Learning with Local Gaussian Process Regression. *Advanced Robotics*, 23(15):2015–2034, 1 2009. ISSN 0169-1864. doi: 10.1163/016918609X12529286896877.

Duy Nguyen-Tuong, Jan Peters, Duy Nguyen-Tuong, and Jan Peters. Using model knowledge for learning inverse dynamics. In *2010 IEEE International Conference on Robotics and Automation*, pages 2677–2682. IEEE, 5 2010. ISBN 978-1-4244-5038-1. doi: 10.1109/ROBOT.2010.5509858. URL `http://ieeexplore.ieee.org/document/5509858/`.

Francesco Nori, Silvio Traversaro, Jorhabib Eljaik, Francesco Romano, Andrea Del Prete, and Daniele Pucci. iCub whole-body control through force regulation on rigid non-coplanar contacts. *Frontiers in Robotics and AI*, 2(March):1–18, 3 2015. ISSN 2296-9144. doi: 10.3389/frobt.2015.00006. URL http://www.frontiersin.org/Humanoid_Robotics/10.3389/frobt.2015.00006/abstract.

J. Park and I. W. Sandberg. Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation*, 3(2):246–257, 6 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.2.246.

Rafael Pinto and Paulo Engel. Scalable and Incremental Learning of Gaussian Mixture Models. 1 2017. URL http://arxiv.org/abs/1701.03940.

Daniele Pucci, Francesco Romano, and Francesco Nori. Collocated Adaptive Control of Underactuated Mechanical Systems. *IEEE Transactions on Robotics*, 31(6):1527–1536, 12 2015. ISSN 1552-3098. doi: 10.1109/TRO.2015.2481282. URL http://ieeexplore.ieee.org/document/7299688/.

Diego Romeres, Mattia Zorzi, Raffaello Camoriano, and Alessandro Chiuso. Online semi-parametric learning for inverse dynamics modeling. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 2945–2950. IEEE, 12 2016. ISBN 978-1-5090-1837-6. doi: 10.1109/CDC.2016.7798708. URL http://ieeexplore.ieee.org/document/7798708/.

R.M. Sanner and J.-J.E. Slotine. Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks*, 3(6):837–863, 1992. ISSN 10459227. doi: 10.1109/72.165588. URL http://ieeexplore.ieee.org/document/165588/.

Robert M. Sanner and Jean-Jacques E. Slotine. Stable Adaptive Control of Robot Manipulators Using Neural Networks. *Neural Computation*, 7(4):753–790, 1995. ISSN 0899-7667. doi: 10.1162/neco.1995.7.4.753.

Jean-Jacques E. Slotine. The Robust Control of Robot Manipulators. *The International Journal of Robotics Research*, 4(2):49–64, 6 1985. ISSN 0278-3649. doi: 10.1177/027836498500400205. URL http://journals.sagepub.com/doi/10.1177/027836498500400205.

Jean-Jacques E. Slotine and Weiping Li. Composite adaptive control of robot manipulators. *Automatica*, 25(4):509–519, 7 1989. ISSN 0005-1098. doi: 10.1016/0005-1098(89)90094-0. URL https://www.sciencedirect.com/science/article/pii/0005109889900940.

Jean-Jacques E. Slotine and Weiping Weiping Li. On the Adaptive Control of Robot Manipulators. *The International Journal of Robotics Research*, 6(3):49–59, 1987. ISSN 0278-3649. doi: 10.1177/027836498700600303. URL http://journals.sagepub.com/doi/10.1177/027836498700600303.

Joshua Smith and Michael Mistry. Online Simultaneous Semi-Parametric Dynamics Model Learning. *IEEE Robotics and Automation Letters*, 5(2):2039–2046, 2020. ISSN 23773766. doi: 10.1109/LRA.2020.2970987.

Silvio Traversaro, Istituto Italiano, D I Tecnologia, Supervisor Dott, and Francesco Nori. *Modelling , Estimation and Identification of Humanoid Robots Dynamics*. PhD thesis, 2017.

Sethu Vijayakumar and Stefan Schaal. Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 1079–1086, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.4151.

P. M. Wensing and J. Slotine. Cooperative adaptive control for cloud-based robotics. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6401–6408, 2018. doi: 10.1109/ICRA.2018.8460856.

Jing Yuan and Bing Yuan. Recursive computation of the Slotine-Li regressor. In *Proceedings of 1995 American Control Conference - ACC'95*, volume 3, pages 2327–2331. American Autom Control Council, 1995. ISBN 0-7803-2445-5. doi: 10.1109/ACC.1995.531387. URL http://ieeexplore.ieee.org/document/531387/.