Data-Driven Learning for Robots

Thommen George Karimpanal under the supervision of Prof. David Braun
Engineering Product Development
Singapore University of Technology and Design
Singapore
thommen_george@mymail.sutd.edu.sg

***Abstract*— This project aims to implement a model learning algorithm using data gathered in real time from a physical system. The learning algorithm will be used to learn a model of the system's dynamics and controller under changing environmental circumstances. The distinguishing feature of this learning algorithm is that it will aim to use locally updating model learning techniques as opposed to globally updating ones. The reason for this is so that the learning is done online, as and when the data is received, and learning local models would help better capture the transient dynamics and non-linearities without overfitting or underfitting. The objective used to evaluate the learning algorithm would be to have the robot drive in straight line irrespective of changing ground surfaces and asymmetric elements added to the system. The simulation and testing of this model learning algorithm would be performed on the EvoBot platform developed in SUTD's Motion, Energy and Control Laboratory.**

Keywords—Model Learning; Dynamics; Online learning; locally weighted regression

I. INTRODUCTION

Modeling of physical systems has always been a challenge, yet animals and other organisms in nature are able to accurately and efficiently learn the dynamics of their bodies within a certain span of time. This is apparent if one observes the stark difference in the manner in which a limb or any other body part is controlled in an infant and in an adult. Organisms use the learnt internal models of their bodies to control and navigate their bodies in the world around them, to react to environmental disturbances and are also able to adapt to changes in their own bodies or in the environment they are in.

In the area of robotics, mathematical models are used to predict the behaviour of a system, in order to control it. These models typically aim to predict the kinematics or dynamics of the physical system through differential equations or probabilistic methods. As the tasks assigned to robots start to involve more and more stochastically varying environmental components, arriving at a good dynamic model of the system can be very challenging. The difficulty could be attributed to several factors, including the sheer complexity of the robot linkages, as well as varying environmental conditions. For example, a good robot model need not be valid when say, the robot is required to hold some additional mass (for say, foraging applications) or when the surface roughness of the ground changes considerably (for mobile robots). Apart from this, owing to random errors and environment noise, a conventional mathematical model of a system is never correct beyond a certain limit. Parameters in the models are usually corrected and re-corrected till that limit is brought to within acceptable bounds. Even after the process of parameter fitting, the model only has a limited use. For example, once the system is taken out of the environment for which it was modeled, the model of the system's behaviour in this new environment may need to be completely reformulated. This greatly limits the usability of the system itself, as it confines the system to a certain environment, if not, to environments very similar to the one it was modeled in. For these reasons, it is essential to determine a more robust way of describing the rules that govern a system's dynamics. This project will conduct exploratory research into alternative online model learning algorithms to try and learn the dynamics of the robot. Several existing algorithms to this end may also be tested in the process.

II. LITERATURE REVIEW

Learning patterns from data in order to formulate a useful model from it is a field that is well established and has been extensively studied. The learning methods are broadly classified into supervised and unsupervised methods based on whether the learning is performed on labeled or unlabeled data. In supervised learning, one is presented with a set of input data and a corresponding set of desired output data. Although the approaches may vary, all supervised learning algorithms are focused on finding a mapping function between these two sets of data. Some of the very popular and extensively used learning algorithms are artificial neural networks, support vector machines, decision trees, etc., Artificial neural networks encompasses a number of the popular techniques that exist today for supervised learning. Some of these are multi-layer perceptrons [3,4], radial basis function networks and self organizing maps [6]. Multi-layer perceptrons use a network of nodes called neurons, which are activated as per some activation function, usually a sigmoid or a hyperbolic tangent function, the forms of which are shown below.

$$y_i = 1/(1 + e^{-v_i}) \quad \ldots(1)$$

$$y_i = \tanh(v_i) \quad \ldots(2)$$

Where y_i is the output and v_i is the input. Typically, the nodes are present in two layers, although more layers can also be used. These nodes are connected to each other by weights, which are parameters that are tuned in order to approximate the function that maps the inputs to the outputs. The learning process consists of computing the error between the desired and real output of each neuron and then updating the weights accordingly using a method called backpropagation [4].

The radial basis function networks is a simplification over the multi layer perceptrons, as it uses only a single layer, and it uses a radial basis function as its activation function. A radial basis function is one whose output depends on the distance r between its center c, and the input value x. A standard radial basis function [1,2] is shown below:

$$\emptyset(r) = \frac{e^{-r^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}} \quad \ldots(3)$$

Where σ is the variance of a gaussian function. The method of performing regression with radial basis function networks is similar to that in multi-layer perceptrons, in that the output of all the functions are weighted and summed together to obtain the global output. The weights are updated based on the network error, as in multi-layer perceptron, but using methods such as gradient descent.

Self organizing maps is another type of artificial neural network. However, it is trained using unsupervised learning in order to classify the inputs according to their similarities. A useful feature of this method is that it preserves the topological features of the input space.

One of the most popular methods for learning is one called support vector machines [7]. In this method, hyperplanes, or high dimensional planes of separation between different object classes are created in order to perform tasks such as classification and regression. The hyperplanes are chosen in such a way that the margin of separation between classes is maximized.

All these methods have been used to learn the kinematic and dynamic models of various physical systems. However, in all these methods, during the learning process, the weights or other parameters are updates globally. The distinguishing feature of this project is that it attempts to learn models by performing the parameter update locally. This method of local update may converge more quickly as the error will need to be minimized only locally, rather than globally. However, it may suffer from problems such as overfitting as some of the noise and other disturbances may be learnt, instead of being ignored during the model update in each iteration. The next section will discuss some existing as well as possible directions for advancing locally updating model learning techniques.

III. Theory

Locally updating modeling techniques are generally less extensively studied as compared to the globally updating techniques as mentioned in the previous section. In globally updating learning approaches, a new training point affects many parameters that are both near as well as far away from the query point. Also, the answer to a query depends on many parameters. Locally updating methods attempt to fit the training data only in a region around the location of the query point. Apart from local learning, algorithms may also use local selection, in which a distance function is used to determine which points are relevant to the query. Generally, all of the global learning techniques can be converted to local ones by making use of a distance function which emphasizes points close to a query more than points away from it.

Some examples of local models include nearest neighbor, weighted average and locally weighted regression. Locally weighted regression methods use Gaussian weights, but with the weights approximated by the least squares method. The main difference between radial basis function networks and locally weighted regression methods lies in the introduction of a linear model associated to each radial basis function and used to perform least squares regression. Each radial basis function defines a region of validity for the corresponding linear model. It globally approximates non-linear functions by combining local linear models.

Compared to artificial neural networks or radial basis function networks, locally weighted regression benefits from the power of linear estimation methods. But it needs to retain all training data to perform the least squares approximation, which may be infeasible for large solution spaces given memory limitations.

Another variation of the approach mentioned above is the locally weighted projection regression method, in which the input dimensionality is reduced using partial least squares. In this approach, a projection is made on the input vector to model high dimensional functions only in their most relevant directions.

There are several improved versions possible for the above methods, such as increasing the number of regions into which the input space is divided, especially in areas where there is a sudden variation of output.

The project will aim to first test out the existing versions of the local learning and perhaps include additional improvements to it. In order to test out the learning algorithms mentioned above, actual robot data will be used to train the learning algorithm. Real time implementation of the locally weighted regression algorithm is described in this work.

IV. Platform and Conventional Dynamic Model

The EvoBot is a differentially driven robot, with motors on either sides of its body. The wheels are coupled to the motors through a gearbox with a gear reduction ratio of 1:100. The speed of each motor is controlled by a pulse width modulated voltage signal. The robot's speed and heading can thus be controlled by changing the duty cycles associated with the two

PWM voltages. The heading only depends on the difference in the speeds of the two motors, and the distance between the wheels.

The robot is modeled using the bond graph technique [8]. In the model, the robot is treated as two point masses connected together, with the two masses concentrated on either side of the robot as shown in figure 1. The two halves are treated independently and the dynamics of each side is modeled. These sub-models are later combined to give the complete model of the EvoBot platform.

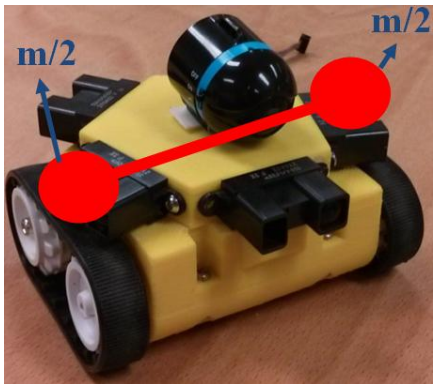

Figure 1: The EvoBot platform

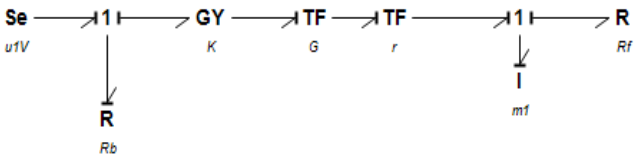

Figure 2: Model of one side of the robot

Figure 2 shows the bond graph model of one of the halves of the EvoBot platform. In the model, a pulse width modulated voltage source in series with an internal resistance drives the two motors. The motor is modeled as a gyrator element with its inductive behavior neglected. The motors in the robot also contain a gearbox which provides a speed reduction of 1:100, which is also modeled. The rotational motion of the motor is converted to translational motion using a transformer element whose modulus is equal to the radius of the wheel. Each motor is assumed to be attached to half the total mass of the robot. The motion of both 'halves' of the robot are coupled together to model the rotational motion of the robot. The model does not account for the electrical dynamics of the system, as we are primarily concerned with the mechanical dynamics. The equations derived from the bond graph are shown below:

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{2K^2}{mRG^2r^2} - \frac{2R_f}{m} & 0 & 0 \\ 0 & -\frac{2K^2}{mRG^2r^2} - \frac{2R_f}{m} & 0 \\ 1/l & -1/l & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \theta \end{bmatrix} + \begin{bmatrix} 2KV/mGrR & 0 \\ 0 & 2KV/mGrR \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Where:

v_1- Velocity of a point on the rim of the wheel/velocity of one side of the robot
v_2- Velocity of a point on the rim of the other wheel/velocity of the other side of the robot
θ– Heading/ orientation
R_b – Internal resistance of the battery (0.5Ω)
K – K value of the motors (0.005V/rad/s)
G – Gear Ratio of motors (1/100)
r – Radius of the wheels (0.02m)
l – equivalent width of the robot (0.095)
m – equivalent mass of the robot (0.26kg)
V- Battery Voltage (3.8V)
u_1- PWM ratio to first motor
u_2- PWM ratio to second motor
R_f- Translational Frictional Damping Constant (21Ns/m)

The states in the above linear model of the system are the wheel velocities and the heading/orientation. If the required output is the position in the x and y directions, these can easily be derived by performing a non linear transformation on the states $v_1, v_2$and θ as shown below:

$$\dot{x} = \left(\frac{v_1 + v_2}{2}\right)\cos\theta$$
$$\dot{y} = \left(\frac{v_1 + v_2}{2}\right)\sin\theta$$

Where x and y are the co-ordinate positions of the robot.

This model approximates the behaviour of the robot to a good degree of accuracy on a test platform, in which the surface under question is wooden. However, when the surface is changed, the model may not hold true anymore. The aim of this project is to have the EvoBot learn the model from the data it collects, irrespective of the environmental parameters. This way of learning is adaptive in nature, and is expected to give the system a certain degree of robustness. The learning is tested by evaluating the performance of the robot for a simple task such as driving in a straight line. Currently, minor fabrication defects and other factors cause some asymmetry in the robot. The model learning algorithm should be able to adapt to these intricacies and make the robot perform its given task (moving straight) irrespective of the asymmetry. In order to truly evaluate the adaptive ability of the algorithm, artificial asymmetries are introduced by covering one side of the robot's tracks with smooth tape. This causes the robot to drift towards one direction by default. The learning algorithm should be able to learn these patterns and apply the corresponding correction.

Model Learning and Results

Post processed learning:

In order to test out the working of the locally weighted regression algorithm, some test data from the EvoBot was

collected and presented as training input. The data consisted of the distance covered by the robot and the time in milliseconds. The test corresponding to this data was such that the robot was made to move in the forward direction in discontinuous bursts, with the robot moving at full speed and then stopping periodically. Figure 3 shows the results in which the output of a learnt model is compared with that of the actual robot data. As seen from the figure, the model seems to learn the data very well, perhaps even leading to a problem of overfitting the data. In the next section, the learning model is applied to a more realistic task of having the robot learn its own controller despite inherent asymmetries artificially imposed on it.

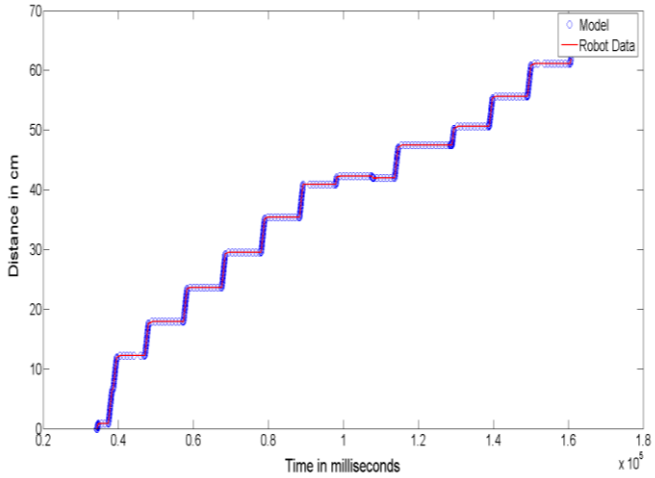

Figure 3. Figure showing the comparison of robot distance data with data from the learnt model

Real-time learning:

In order to test out the real time learning ability of the algorithm, one side of the robot was taped as mentioned in the previous sections and as shown in figure 4. The learning was tested with different wheels on either side as well, as shown in figure 5.

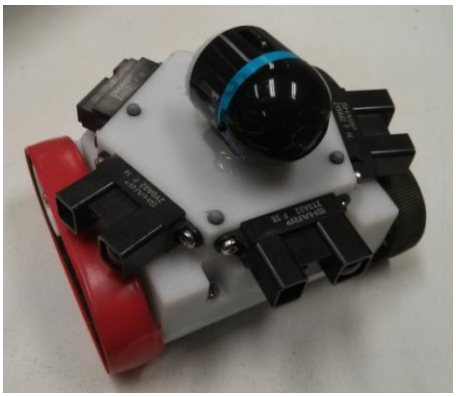

Figure 4: One side of the robot taped to introduce asymmetry

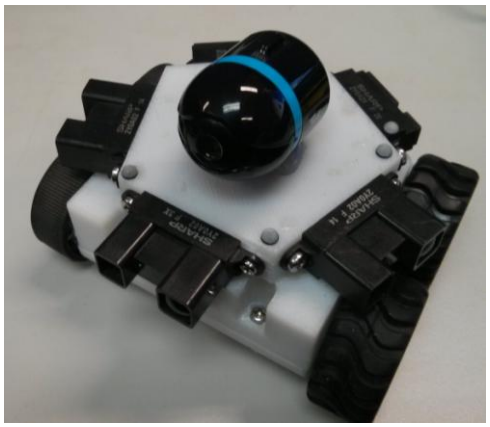

Figure 5: Different wheels used on either side of the robot to introduce asymmetries

The robot was then trained for the ability to maintain a given heading value irrespective of its internal asymmetries. The natural tendency of the robot is to drift towards one side, but the learning objective is to learn the controller so that it can correct itself to travel at the required heading.

The formulation of the problem was as follows: Since the drift or the angular velocity of the robot is dependent on the difference in the PWM duty cycles between the two sides, the left and right PWM duty cycles were parameterized as shown below:

$$u_{left} = u_{base} + \delta u$$
$$u_{right} = u_{base} - \delta u \quad \ldots(4)$$

In this way, the number of control parameters are reduced from two to just one parameter, δu. This was done to make it convenient for the learning algorithm, as now, the regression problem is reduced to a 1 dimensional one.

There were two modes of training developed; one in which the robot scanned for errors in the heading and corrected itself by training through a number of different predetermined values of δu for a predetermined length of time and storing the corresponding values of drift and δu and performing LWR on this data, and the second mode in which the robot corrected itself through gradient descent, but in the process, collected drift and δu data in order to learn a good model of the controller through LWR. The time for which a particular action was executed in this mode was also predetermined. For both modes, a system was set up for connecting the robot to a central computer through Bluetooth and for recording the robot data into a csv file. The csv file is then read to retrieve the data, and LWR is performed on this data.

Each of the above mentioned approaches have their own merits and demerits. The merits of the first approach is that the solution space is explored quite well, but it suffers from the disadvantages that the exploration of the solution space is pre-defined, and that the learning is not real time in nature, but is more like batch learning, as the robot starts learning only after it has run through its training routine.

In the second approach, the advantage is that the learning through gradient descent is truly real time, but the exploration of the solution space depends on the magnitude of initial error experienced just before learning. Since gradient descent converges relatively quickly, the number of data points explored is limited. In summary, it results in the robot learning a limited solution space, but in real time. One way in which to improve approach 2 is to insert previously collected data into the system just before it starts updating it. This could be thought of as a way to incorporate memory into the robots, with the memory being continuously updated with the learning algorithm.

Overall, the algorithm worked well for real time learning, and was able to generate a model that fit the data well. A sample output of the algorithm is shown in figure 5. It can be seen that the non-linear aspects of the system are well captured by the learnt model.

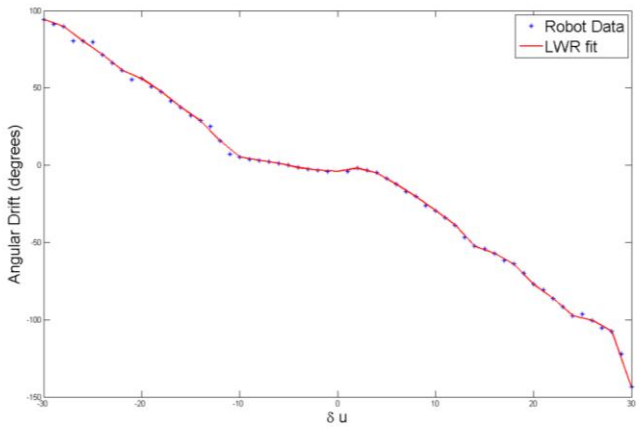

Figure 5: A plot of angular drift vs the parameter δu on data collected through approach 1.

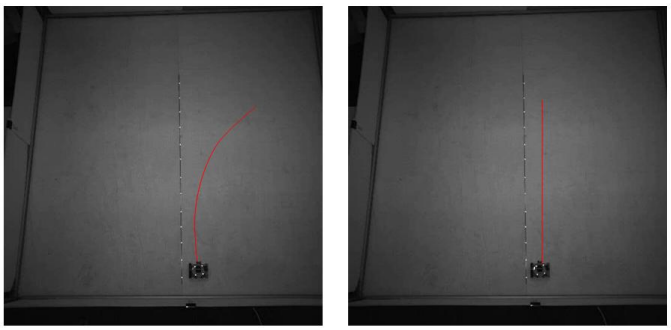

Figure 6: Paths of the robot shown before and after the learning algorithm

Problems with LWR:

Although LWR was quite efficient at learning models in real time, its performance is highly dependent on the Gaussian kernel chosen. If the Gaussian kernel has a very small standard deviation, the algorithm fails to update certain data points that are far from the query point. On the other hand, if the kernel has a high variance, the model update is smooth, but less accurate. Since increasing the variance of the kernel function unfavorably affects the results, this option is not preferred. In order to handle such cases even when the kernel has a low variance, the final learnt model was passed through a layer of linear interpolation in order for the model to be able to return reasonable values at all points in the data range. Alternatively, a non-gaussian kernel such as a sigmoid or hyperbolic tangent function could also be chosen, although its effects have not been studied in this project.

CONCLUSION

In this work, a locally weighted approach to regression is studied and implemented. The weights used are Gaussians whose variances depend on the distance of a particular point to a query point. Implementation was performed on the EvoBot platform, in order to perform the particular task of moving in a specified heading direction by learning the model of its controller. Internal asymmetries were added in order to make the learning task more challenging. The formulation of the problem was described in detail. The post-processed as well as real time approaches to the learning algorithm were described in detail and the corresponding results were presented and discussed. It was found that the learnt model of the system was adaptive in nature and performed significantly better than the conventional model of the system, which was also described. Lastly, some limitations of the LWR method were also described and potential solutions were proposed.

REFERENCES

[1] Olivier Sigaud , Camille Salaün, Vincent Padois, "On-line regression algorithms for leanring the mechanical models of robots: A survey", in Robotics and Autonomous Systems, Volume 59, Issue 12, (2011) 1115–1129

[2] Christopher G. Atkeson, Andrew W. Moore and Stefan Schaal, "Locally weighted learning" in Artificial Intelligence Review, Issue 11, pp. 11-73, 1997

[3] S. Haykin, "Neural Networks and Learning Machines", Prentice Hall, 2009

[4] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychological Review 65 (1958) 386–408.

[5] D. Nguyen-Tuong, J. Peters, Model learning for robot control: a survey, Cognitive Processing (2011) 1–22.

[6] T. Kohonen, Self-Organizing Maps, Springer, Berlin, 2001

[7] Nello Cristianini and John Shawe-Taylor, "An introduction to Support Vector Machines and Other Kernel Based Learning Methods", Cambridge University Press, 2000. ISBN 0-521-78019-5

[8] Karnopp, Dean C., Margolis, Donald L., Rosenberg, Ronald C., 1990: "System dynamics: a unified approach", Wiley, ISBN 0-471-62171-4.