MACHINE LEARNING FOR MOTION PLANNERS

DANIEL BEYLKIN

Mentor: Jerrold Marsden
Co-Mentor: Marin Kobilarov

Abstract. We are concerned with designing trajectories that allow robots to maneuver in a region with obstacles. For approaches that use random sampling-based motion planning schemes, the rate of convergence to a feasible (and optimal) trajectory is critical, especially in high-dimensional state spaces. Usually such methods do not systematically incorporate information about previously computed trajectories. We investigate a method that may improve the rate of convergence by using machine learning to develop a coarse model of the cost as a function of the spatial distribution of the trajectories.

We consider a system without dynamical constraints and use locally weighted learning to develop such coarse model. We demonstrate this approach on several examples and propose a particular manner in which one could handle obstacles. We also examine potential pitfalls of the approach and develop plans for further research.

In addition, we propose a method to build dynamical models from measured data. Often, it is impossible to construct the dynamics of a system from first principles. Instead, we offer a more universal approach using sums of separable functions in high-dimensional spaces in order to learn (approximate) the dynamical model.

1. Introduction

Robotic motion and trajectory planning is one of the most fundamental, yet formidable challenges of autonomous robotic design. We would like to be able to provide a robot with a set of tasks and have it execute the commands autonomously without running into obstacles. While a seemingly simple problem given the ease with which humans deal with it on a daily basis, current algorithms have thus far been inadequate. Yet it is this precise observation that motivates a great deal of research in robotic motion planning.

In this project we are primarily interested in the computation of a realizable path for a robot to traverse (preferably in some optimal manner) with the ultimate hope of actually implementing it in a robotic system. However, in order to implement such path planners it is essential to consider the dynamics of the system including gravity and other forces. Moreover, in most situations, the robot will also have to avoid obstacles, which appear as geometric constraints on its motion. The ultimate goal is to be able to do this without complete knowledge of the dynamics and/or the physical environment and in the presence of noise and other uncertainties. However, the problem is difficult enough even without these additional complications [7].

Potential applications for such problems are vast and far too numerous to inclusively discuss each one. Instead, we mention a few as motivation. It is applicable to

autonomous motion planning for aerobots[1] and surface sampling robotic vehicles[2] necessary in solar exploration, such as NASA's eventual mission to Titan. Other applications include robotic vacuum cleaners[3] and lawn mowers[4] or humanoid robots designed for home assistance[5]. Another application is in streamlining of warehouse technologies using robots designed to store, move, and sort inventory by using inventory pods which are automatically retrieved for each order. Doing so helps minimize the amount of empty space in a warehouse providing increased capacity and efficiency[6]. Further applications include rough-terrain robotic mules[7] and robots designed to aid troops in warfare[8].

1.1. **NASA mission to Titan.** We briefly expand on NASA's eventual mission to Titan, a moon of Saturn, as it is one of the key motivations for this work. The Cassini-Huygens spacecraft mission to study Saturn revealed remarkable observations of Titan. The data sent back to Earth demonstrates the likely existence of high-latitude hydrocarbon oceans[9] and equatorial sand dunes[10]. While life is unlikely to exist on Titan, its uncanny similarities to Earth have sparked the interest and curiosity of the scientific community.

An overview of some of the most intriguing scientific questions can be found in [8, 11]. These include, but are not limited to, a study of the geochemical processes on Titan and their similarities to those found on Earth, a study of the prebiotic chemistry on Titan, and a study into the origin and evolution of Titan. Answering these questions would require greater surface exploration than is currently possible with the Cassini-Huygens mission. If NASA were to return to Titan, aerobots and Montgolfier balloons have been proposed for scientific exploration of Titan's surface [5]. Ideally the vehicle would be able to gather data for an extended period of time (hopefully greater than six months) while circumnavigating Titan with the possibility of conducting some surface sampling experiments.

While the balloon is on Titan, intervening with the flight-plan on a short timescale is impossible. In addition to a communication lag time of several hours, there will be communication blackout periods where the balloon will be operating in largely unknown environments [5]. Therefore, it is necessary for the balloon to make decisions autonomously and still be able to reach particular regions of interest on Titan in order to gather high quality data. It is hoped that by optimally exploiting the winds on Titan, the balloon would be able to autonomously navigate different regions of Saturn's moon with minimal control force or time of travel.

We are also faced with a problem of designing a sampling device that can be deployed from an aerobot and is able to sample a wide variety of terrain while traversing its vast surface. A possible option would be to use a robotic surface sampling vehicle.

[1] http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=7
[2] http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=16
[3] http://store.irobot.com/home/index.jsp
[4] http://www.friendlyrobotics.com/
[5] http://asimo.honda.com/
[6] http://www.kivasystems.com/
[7] http://www.bostondynamics.com/
[8] http://www.irobot.com/sp.cfm?pageid=109
[9] http://saturn.jpl.nasa.gov/photos/imagedetails/?imageID=2127
[10] http://saturn.jpl.nasa.gov/news/cassinifeatures/feature20080502/

1.2. **The Motion Planning Problem.** We assume that the motion of a mechanical system, such as a robot or a helicopter, is defined in state space X and constrained by a system of ODEs

$$\dot{x}(t) = f(x(t), u(t))$$

often derived from Newtonian mechanics. Given some initial and final configuration for the robot, we want to find a continuous path $x(t) : [t_0, t_f] \to X$ under the influence of control $u(t)$ (which may be underactuated - sometimes referred to as a nonholonomic system) that connects these two configuration or brings the robot sufficiently close. This problem may be thought as a variant of the classical boundary value problem. However, such formulation may not be well-suited for handling obstacles.

Most commonly these problems are recast as optimization problems, that is, we require that a solution path minimize some cost functional

$$J(x, u) = \int_{t_0}^{t_f} C(x(t), u(t))dt + V(x(t_f))$$

over possible controls $u(t)$ [10]. It is important that the resulting minimal path satisfies the system's dynamical constraints and is collision-free. We may further constrain the system by imposing various actuator limits, safety limits, etc., which would present themselves as inequalities. As a result, we have a complicated constrained optimization problem and many techniques for solving such problems are not well-suited to real-time applications necessary for implementation in a mechanical system. We concern ourselves with solution methods that may be implemented in real-time.

1.3. **Sampling-Based Motion Planners.** The overwhelming majority of solution techniques to motion planning problems are sampling-based due to differential constraints. The basic principle is that we probe the state space of the mechanical system using a sampling scheme. While we may not be able to guarantee the completeness of such schemes, many schemes use random sampling that is probabilistically complete: with enough samples the probability of finding a solution converges to one. In such schemes, it is also important to consider the rate of convergence. Our goal is to sample the system more intelligently in order to hopefully improve this.

For a more thorough discussion of sampling-based motion planners, we refer the reader to [7]. We summarize the main points here.

1.3.1. *Sampling.* While the number of states in a configuration space is infinite, we investigate it by using a finite number of samples. Thus, it is important that the sampling scheme is chosen with care, namely, we require that the algorithm guarantees a solution provided that one exists or, if it does not, terminates in finite time. Any sampling method provides us with a sequence in the set of states that can be reached in a fixed finite time (called a reachable set) and it is important that we also require that, in the limit, the sampling sequence be dense in the configuration space (excluding the obstacles). For example, for random sampling, as we increase the number of samples the probability that our sampling sequence is dense converges to one. We note that it is preferable that the samples be uniformly distributed. There are also deterministic sampling schemes which satisfy the same requirements for completeness. We note that we can choose to sample either the

state space or the control space. For now, we are not concerned with the difference between these two schemes.

The choice of sampling scheme also involves a decision on how detect which configurations cause collision with obstacles. We are not only concerned with ensuring that the sampled configurations avoid collisions but also that the path itself is collision-free. This poses an additional challenge since we cannot check every individual point along that path. In practice, we can do this by choosing a small step size and testing a sufficiently large number of points along the path. We also need to assure that inequality constraints (e.g. constraints on the velocity, actuator) are satisfied.

1.3.2. *Local Planning Method: Maneuver-Based Motion Planning.* A key aspect of sampling-based motion planners is the local planning method used to connect the samples in the configuration space. Designing such local planning methods given differential constraints is a difficult problem and a key area of research. A major issue is that in many cases we do not have physically accurate or reliable models of the dynamics, especially for complex robots. We observe that when we walk, we do so without knowledge of the precise dynamics that govern our motion and, yet, we still move without difficulty. What is particularly intriguing about this observation is that we are able to do so in spite of a very large dimension of the state space (if we choose to describe the problem in this manner). This motivates the idea that we are able to move by using the concatenation of "well-practiced" maneuvers with steady-state motions.

This method of motion planning is based on a library of motion primitives which are concatenated to form complete trajectories. It is important that during such concatenation, boundary conditions are satisfied, specifically that the final state of the first primitive coincides with the initial state of the subsequent primitive.

Because the library is built up using pre-computed trajectories, such method may be implemented in real-time. The library of motion primitives is split into two categories: trim trajectories which are steady-state motions and maneuvers which allow us to move from one trim trajectory to the next. These maneuvers may be computed experimentally which allows these methods to work even in the absence of dynamical models. We refer the reader to [6] for a more detailed discussion.

2. Machine Learning Guided Sampling

As mentioned, the rate of convergence of sampling based schemes is critical especially when sampling in a high-dimensional state space. One of our goals is to sample the system more intelligently in order to improve the rate of convergence.

The idea is to use previously computed trajectories to guide future sampling. We note that current methods do not systematically incorporate information about previously computed trajectories; ignoring this information seems wasteful. We want to infer some coarse model of the cost as a function of the spatial distribution of trajectories. This is where machine learning methods may be appropriate.

The hope is that we can use some machine learning method that will approximate the data well enough in order to allow us to extract extrema information easily. From this, we could instruct our sampling-based scheme to sample in a region nearby such location.

In our approach, we must also consider how to handle obstacles. We would like to avoid the discontinuities in our cost function that appear due to obstacles. We

explore a method of handling such obstacles by penalizing the cost associated with paths traveling through the obstacle using a smooth function. In this manner, we do not impact the location of the extrema, but avoid artificial minima that may appear inside an obstacle.

2.1. **Locally Weighted Learning.** We observe that paths lying in different regions are likely not correlated. For example, two paths on either side of a mountain likely do not contain meaningful information about each other. However, two paths on the same side of a mountain likely do. This motivates the notion of locality: correlated paths constitute the only meaningful information on the data set.

Let us consider locally weighted learning (see, e.g., [1]) as an approach. One of the key components of locally weighted learning is the choice of the distance metric. We note that in our case, the only information we have about the trajectories are some nodes along it. The distance must not depend on the choice of the nodes on the trajectory. Hence, we choose to define the distance between two trajectories as the area between them. In the future, we may even want to consider using obstacle information in the computation of the area. Hence, two paths separated by an obstacle will not be close to each other even if spatially they appear to be.

Once we have established the distance metric, we must also choose a weighting kernel. It must be maximal at zero distance and smoothly decay as the distance increases. One such option is

$$k(d) = \exp\left[-\alpha \cdot d^2\right],$$

where α is some tunable parameter. We use leave-one-out cross validation to optimize our choice of α.

We briefly summarize the locally weighted learning algorithm, however, we refer the reader to [1] for a more thorough discussion. We define $\mathbf{x}_i$ to be the i^{th} data point in the training set and $\mathbf{q}$ to be the query point. To simplify notation, let

$$w_i = \sqrt{k(d(\mathbf{x}_i, \mathbf{q}))}.$$

Let the dimension of our data be m and let us assume that our training set includes n data points. We define $\mathbf{W}$ to be the diagonal $n \times n$-matrix with elements $\mathbf{W}_{ii} = w_i$. Because we will be performing regression, in order to incorporate the constant term, we append 1 to every row in $\mathbf{x}_i^T$ (we likewise append 1 to $\mathbf{q}$). Now, let us also define $\mathbf{X}$ to be the $n \times (m+1)$-matrix where the i^{th} row is $\mathbf{x}_i^T$ and $\mathbf{y}$ to be the vector where the i^{th} element is y_i. We weight the data directly by defining

$$\mathbf{Z} = \mathbf{W}\mathbf{X}$$

and

$$\mathbf{v} = \mathbf{W}\mathbf{y}.$$

We now seek a global model that is linear in the parameters of β (note, β is an $(m+1)$-vector). We have the overdetermined system

$$\mathbf{Z}\beta = \mathbf{v}.$$

We want to determine the parameters of β that minimize the criterion

$$C = \|\mathbf{v} - \mathbf{Z}\beta\|^2.$$

This problem has a well known solution given by solving the normal equations

$$\left(\mathbf{Z}^T\mathbf{Z}\right)\beta = \mathbf{Z}^T\mathbf{v},$$

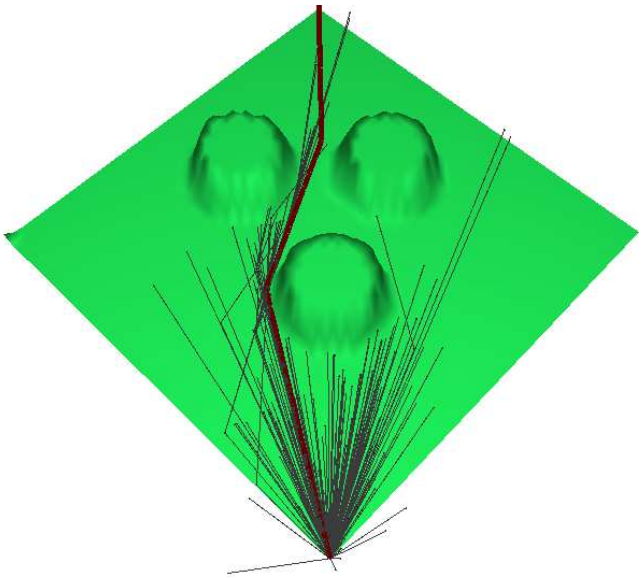

FIGURE 2.1. Demonstration of trajectory generation using a random sampling-based scheme

which provides us an estimator of the form

$$\hat{y}(\mathbf{q}) = \mathbf{q}^T \beta.$$

2.1.1. *A Proof of Concept.* We consider a particle in a plane with simple obstacles and no dynamical constraints. Figure (2.1) is a plot of the environment. There are three circular obstacles between the initial node (bottom) and the goal node (top). The algorithm iteratively adds nodes to a tree starting at the initial node. The bold line indicates the current optimal trajectory. The other trajectories connect samples in the state space, thereby forming a tree. For subsequent iterations, nodes will be connected to the graph if possible and, in some instances, the tree itself will be pruned. The specific details of the algorithm are beyond the scope of this report.

Using the same environment, we ran the algorithm twice. The first time, we collected the feasible trajectories and their associated costs into a training set. We ran the algorithm again (with more iterations in order to generate more paths) to then generate the test set.

As a proof of concept, the trajectories in Figure (2.2) are grouped based on similar costs, with green being the lowest followed by red, blue, magenta, and yellow. The bold trajectory indicates the optimal one. The reason for plotting them in this manner is to determine how the locally weighted learning scheme is able to identify regions where extrema are located. In other words, can it identify reasonably good trajectories and distinguish them from the others? Because we seek a coarse model of the trajectories, we only need to identify, spatially, where good trajectories are likely to lie.

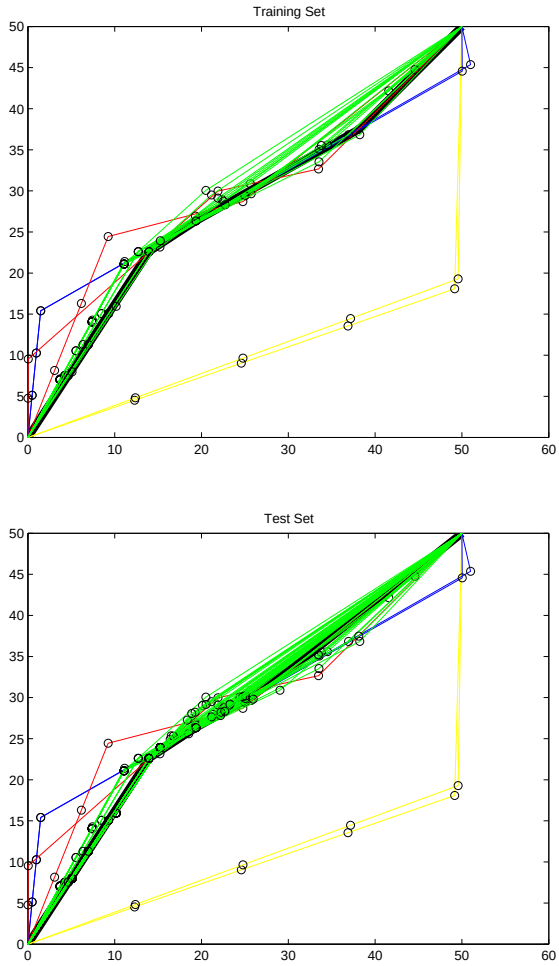

FIGURE 2.2. Plot of trajectories in the training set (top), plot of trajectories in the test set (bottom). In both cases, trajectories are grouped (i.e., colored) based on their relative costs.

Using locally weighted learning, we predict the cost of the trajectories in the test set. The result is plotted in Figure (2.3). In this case, the results are promising as the predicted costs group the trajectories in the exact same manner as in the test set. We also have a mean squared error of roughly 0.024 indicating fairly accurate results.

We caution that the results may be slightly optimistic as the trajectories in the training set and the test set are very similar. Further exploration remains one of our future goals.

2.1.2. *Implementation.* In our choice of machine learning tool, we would also like to both be able to easily extract extrema information and have it scale to systems

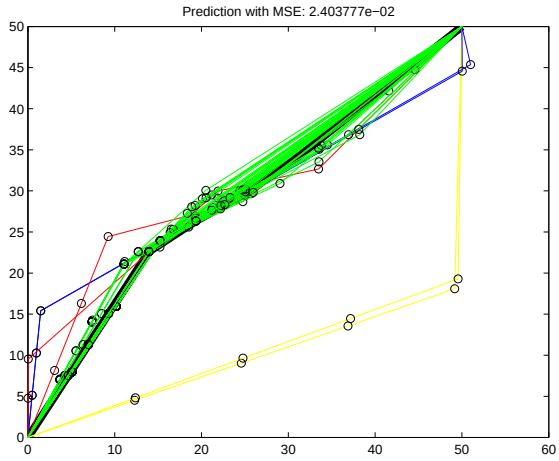

FIGURE 2.3. Plot of trajectories grouped in same manner as in Figure (2.2) using, instead, the predicted cost.

with increasingly high dimensionality as the function mapping trajectories to a cost will rapidly increase in the number of variables.

In its standard form, locally weighted learning is a so-called "lazy" learner, meaning that we must first make a query before it learns the function - notice that the weighting matrix depends entirely on the location of the query. We consider a modification of locally weighted learning that is less "lazy."

Given a set of trajectories, we exclude the optimal trajectory in the training set and use locally weighted learning to fit a hyperplane to the data point corresponding to the optimal trajectory. We also define a support region by removing all trajectories greater than one standard deviation in distance from the optimal trajectory. Using the hyperplane, we then determine a new trajectory on the boundary of the support region, which we use to guide the sampling scheme for the motion planner.

Currently, we have successfully been able to determine the new trajectory with some promising results, but integrating this with the motion planner remains a future research goal.

One drawback of using locally weighted learning is that it determines a linear approximation of the data, which may not be appropriate. Consequently, we are currently considering using higher order approximations of the data, including quadratic and multivariate Gaussian fits.

2.2. **Obstacles.** Another critical issue in modeling the trajectory costs is how to handle obstacles. Locally weighted learning and many other learning schemes are better suited to learn smooth functions whereas obstacles create discontinuities. Preliminary results indicate that it is better to handle obstacles by replacing the discontinuous cost function with an appropriately chosen smooth one. This approach offers several advantages. We do not reject trajectories and nodes simply because of obstacles in the way. Thus, we can create a larger training set in fewer iterations. Moreover, we do not simply discard nodes that may be worthwhile for later iterations. This is a particular concern in the first several iterations when the

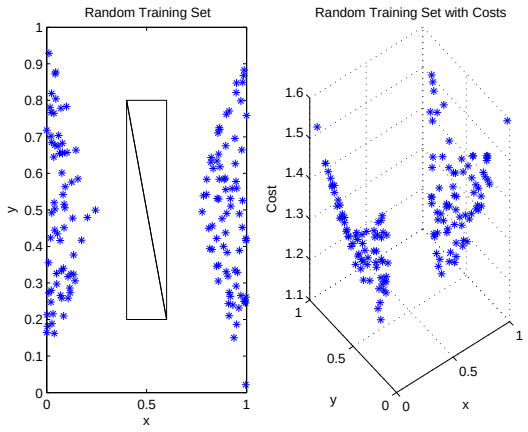

FIGURE 2.4. Training data generated by randomly sampling inside $(0,1) \times (0,1)$ region. All trajectories whose intermediate node could not linearly connect to the initial node at $(0.5, 0)$ and final node at $(0.5, 1)$ without intersecting the obstacle are disregarded. This includes all nodes inside the obstacle itself. We represent all trajectories in the training set by the location of their intermediate node.

tree of trajectories contains only a few nodes. Also, since collisions with obstacles are treated as penalties, it should help avoid artificial minima appearing inside the obstacles without impacting the extrema information outside the obstacle.

To explore this idea, we considered very simple trajectories with only one intermediate node (thus, the trajectories are functions of just the x and y position of the intermediate node). Again, the system has no dynamical constraints and in this case has only one rectangular obstacle between the initial and goal nodes. We randomly sampled the state space to develop a training set. In the first case, we rejected all nodes which could not connect to the initial and final trajectory without intersecting an obstacle. In the second case, we kept all nodes, but associated a cost that exponentially penalized intersection with an obstacle. Using these training sets, we learned the function on a uniform grid.

Figure (2.4) is a plot containing the training data for the first case. Because we do not know where extrema may lie, we are worried that artificial extrema will appear inside the obstacle. When we attempt to learn the cost for trajectories that do intersect with the obstacle, we get the results plotted in Figure (2.5). So, while we were able to do a particularly good job in the region where trajectories do not intersect with an obstacle, we note that superficial minima appear inside the obstacle. This may be problematic as we now have to restrict the region of the state space in which we search for extrema which further complicates the problem.

As we will show, we may bypass this issue by associating a penalty function for paths that intersect with an obstacle thereby associating them with finite costs.

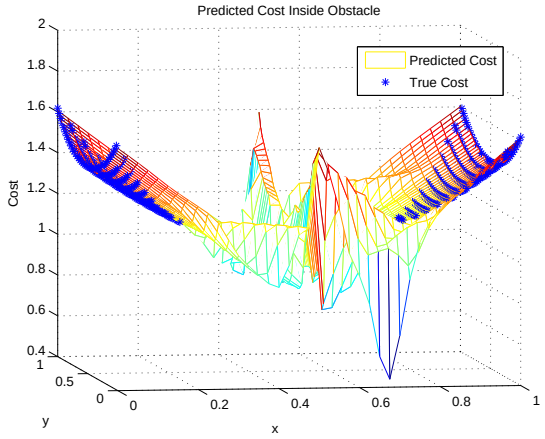

FIGURE 2.5. Using the training data shown in Figure (2.4), we predict the cost for trajectories formed my connecting the initial node at $(0.5, 0)$, the intermediate node, and a final node at $(0.5, 1)$. We generated our test set of trajectories my uniformly distributing the intermediate nodes in the space $(0, 1) \times (0, 1)$ with some nodes possibly inside the obstacle.

Figure (2.6) is a plot containing the training data for this second case. We immediately note that we have a much larger set of training data. This is important because when we consider systems in much higher dimension, it seems wasteful to simply reject paths that collide with obstacles instead of using that information in our learning scheme.

We then try to learn the cost for all trajectories whose intermediate nodes lie on a uniform grid. The results are plotted in Figure (2.7). Notice that in the extrema region, locally weighted learning still does a good job in predicting the cost. However, inside the obstacle, we do not have any artificial minima.

While this is only one example, the results seem to clearly indicate that associating a penalty for trajectories that collide with obstacles instead of discarding them is the correct strategy.

3. Dynamical Model Learning

Sampling-based schemes require some local planning method to connect the samples. While this is easy in cases where we do not have dynamical constraints, the problem becomes noticeably more difficult for more complicated systems. A common approach uses motion primitives (see, e.g., [6]), which is motivated by the idea that we are able to move by using the concatenation of "well-practiced" maneuvers with steady-state motions. These maneuvers may be computed experimentally; however, choosing the appropriate maneuvers is difficult without a physically reliable model of the dynamics, which is often the case for more complicated systems.

Although we often do not know the physical dynamics of the system, we may want to learn the dynamics experimentally. In this case, we are faced with a problem

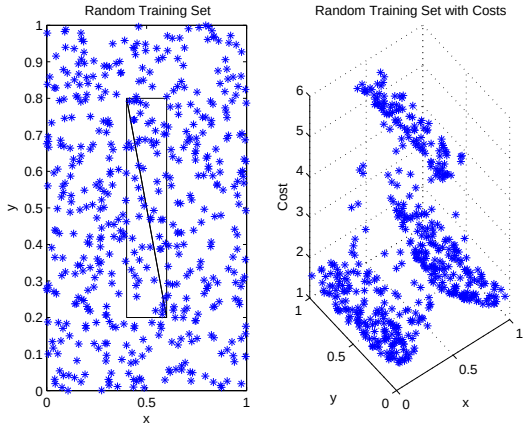

FIGURE 2.6. Training data generated by randomly sampling inside $(0,1) \times (0,1)$ region. All trajectories have initial node at $(0.5, 0)$ and final node at $(0.5, 1)$ and are represented as a function of its intermediate node. Unlike Figure (2.4), no trajectories are discarded, regardless of possible collision. Instead, we associate a penalty function for trajectories that intersect with an obstacle.

of learning functions in high dimensional spaces which is difficult. Most approaches seem to require some sort of dimensionality reduction.

We begin to explore an alternative approach where we may bypass the curse of dimensionality if we approximate the physical system well using a separated representation (see, e.g., [2, 3])

$$f(\mathbf{x}) = \sum_{l=1}^{r} s_l \prod_{i=1}^{d} f_i^l(x_i) + \mathcal{O}(\epsilon), \tag{3.1}$$

with small separation rank r and prescribed accuracy ϵ. In the classical framework where we approximate f using a single separable function, if the approximation is not good enough, there is no way to improve the accuracy. Thus, (3.1) is a natural extension of a separable representation using a sum of separable functions.

We note that many black-box approaches to system identification are based on the form in Equation (3.1). However, they differ in that they require that f_i^l are preselected from a particular basis set. In this form, the curse of dimensionality manifests itself even in simple cases. Thus, in a separated representation, the f_i^l are chosen without such constraints. The trade-off is that we switch from a linear to a nonlinear approximation.

This approach offers a greater deal of flexibility and in a few examples, the extra freedom allows one to find good approximations with small r. It may not seem immediately obvious that such representation should exist, but there is evidence that in many cases they can be found. We briefly discuss a few illustrative examples (see, e.g., [2, 3]).

Consider $\sin(x_1 + \cdots + x_d)$ which, using usual trigonometric formulas for sums of angles, would have $r = 2^{d-1}$ terms. However, numerical tests demonstrated that

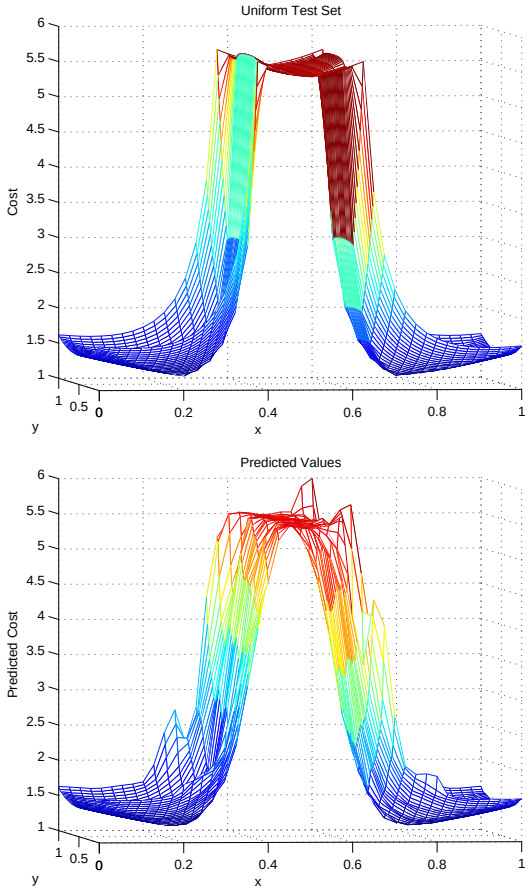

FIGURE 2.7. True penalized cost for set of trajectories with uniformly distributed intermediate nodes (top), predicted cost for these trajectories using training data shown in Figure (2.6) (bottom)

one only needs d terms, which led to a new trigonometric identity (see, e.g., [9])

$$\sin\left(\sum_{i=1}^{d} x_i\right) = \sum_{i=1}^{d} \sin(x_i) \prod_{j=1, j\neq i}^{d} \frac{\sin(x_k + \alpha_j - \alpha_i)}{\sin(\alpha_j - \alpha_i)}$$

for all choices of $\{\alpha_i\}$ such that $\sin(\alpha_j - \alpha_i) \neq 0$ for all $i \neq j$.

We highlight several key results that this examples demonstrates: the representation is not unique and there can be ill-conditioned representations (e.g., $\alpha_1 \to \alpha_2$) even when well-conditions representations are readily available. As a result, we must introduce the notion of conditioning for the representation.

If we consider the additive model $\sum_{i=1}^{d} \phi_i(x_i)$, where ϕ_i are bounded, we may naively think that it has separation rank $r = d$. However, in the limit, it actually

only has $r = 2$ terms:

$$\sum_{i=1}^{d} \phi_i(x_i) = \lim_{h \to 0} \frac{1}{2h} \left(\prod_{i=1}^{d} (1 + h\phi_i(x_i)) - \prod_{i=1}^{d} (1 - h\phi_i(x_i)) \right).$$

Finally, we note that Gaussians are also separable:

$$\exp\left[-c \left\|\mathbf{x} - \mathbf{z}\right\|^2\right] = \prod_{i=1}^{d} \exp\left[-c(x_i - z_i)^2\right].$$

3.1. **Advantages.** Common approaches to these types of problems use dimensionality reduction in order to find lower dimensional representations of complex systems with large dimensionality. However, even if we are able to represent a system with hundreds of variables using only, say, ten variables (for example, this may be accomplished using [4]), the dimension of the space may still be too large. On the other hand, an efficient "tabulation" of a multivariable function may be able to overcome these obstacles and, in principle, still work for systems with hundreds of variables.

The separated representation approach is still relatively new and may be improved. For example, currently the nonlinear approximation is accomplished using alternating least squares which is both slow and may be prone to local minima.

The current algorithm has performed reasonably well on various benchmark data sets (e.g., Friedman data set) in comparison to other currently available methods.

Our idea is to try to use the algorithm to learn the dynamics of a double pendulum as a function of its initial conditions and, perhaps, controls. Of course, machine learning using this separated representation approach is not limited to dynamical model learning. It may be used to learn solutions of boundary value problems in local neighborhoods, to learn control parameters that produce desired results, etc. It is hoped that the flexibility of the separated representation approach will provide a universal framework to handle general machine learning problems in robotics.

Acknowledgments

We would like to thank Dr. Jerrold Marsden for providing the opportunity to work on this project. We would like to thank Dr. Marin Kobilarov for his helpful advice and guidance. We would also like to thank Dr. Martin Mohlenkamp for providing the code for machine learning with sums of separable functions.

Lastly, we would like to thank the SFP office and Mr. and Mrs. Carl V. Larson for their generous financial support of the SURF program and this project.

References

[1] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally Weighted Learning. *Artificial Intelligence Review*, 11(1):11–73, 1997.

[2] Gregory Beylkin, Jochen Garcke, and Martin J. Mohlenkamp. Multivariate regression and machine learning with sums of separable functions. *SIAM J. Sci. Comput.*, 31(3):1840–1857, 2009.

[3] Gregory Beylkin and Martin J. Mohlenkamp. Algorithms for numerical analysis in high dimensions. *SIAM J. Sci. Comput.*, 26(6):2133–2159 (electronic), 2005.

[4] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Appl. Comput. Harmon. Anal.*, 21(1):5–30, 2006.

[5] Alberto Elfes, James F. Montgomery, Jeffery L. Hall, Sanjay S. Joshi, Jeffrey Payne, and Charles F. Bergh. Autonomous flight control for a Titan exploration aerobot. In *The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space - iSAIRAS*, Munich, Germany, Septermber 5-8 2005.

[6] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Maneuver-Based Motion Planning for Nonlinear Systems With Symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, 2005.

[7] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, Cambridge, 2006.

[8] Ralph D. Lorenze. Post-Cassini Exploration of Titan: Science Rationale and Mission Concepts. *Journal of the British Interplanetary Society*, 53:218–234, 2000.

[9] Martin J. Mohlenkamp and Lucas Monzón. Trigonometric identities and sums of separable functions. *Math. Intelligencer*, 27(2):65–69, 2005.

[10] Richard M. Murray. Optimization-Based Control. Draft v2.0b, January 2009.

[11] Kim Reh, Dennis Matson, and Jonathan Lunine. TSSM Architecture and Orbiter Design. Unpublished presentation, July 9 2008.