# Optimal Control Mesh

Peter Taraba

*Abstract*—**We present a novel algorithm for optimal control of nonlinear systems, which creates control function over a mesh on a region of interest. The algorithm presented in this paper is an alternative to a set-oriented approach and subdivision algorithm for optimal control. The main contribution of this paper is error estimation for a found solution. We show on two dimensional and three dimensional problems, that this new algorithm is faster than a subdivision algorithm. In comparison with a set-oriented approach, the new algorithm keeps the same advantages as the subdivision algorithm which include a smaller memory foot-print of the final solution, no need for discretization and knowledge when to stop increasing the mesh size.**

*Keywords.* **Optimal control, mesh, ordinary differential equations**

## I. Introduction

Finding an optimal control for a broad range of problems is not a simple task. There are currently many methods which try to tackle this problem using a range of solutions. The closest ones to the algorithm suggested in this paper are a set-oriented approach described in [1] and in [2], and a subdivision algorithm for optimal control [3]. In addition to these, there is [4] where authors of the paper partition a set of state space into simplicital cones and provide a piecewise affine control law which ensures feasibility and stability, but is also optimal with respect to LQR problems. For more references on this topic, see [3].

The main advantage of the subdivision algorithm for optimal control over a set-oriented approach is the ability to estimate when to stop increasing the mesh size and smaller foot-print of the final solution, because with the subdivision algorithm, solutions were found even for coarse divisions of state space. Moreover, a set-oriented approach has a need for sufficient partitioning (adaptive structure), which does not necessarily improve the quality of the final solution. The last advantage of the subdivision algorithm over a set-oriented approach is no need to convert from continuous to discrete model. With optimal control mesh, we keep the advantages subdivision algorithm had, and the computations needed for finding a solution are significantly faster than with subdivision algorithm. Even for three dimensional problems, the computations are faster than computations of two dimensional problems with the subdivision algorithm. In addition, this new algorithm introduces error estimation which can be used as an indicator when to stop increasing the mesh size.

Section II of this paper describes the control problem we seek to address. Section III introduces the algorithm for finding optimal control mesh. Section IV describes a problem that the algorithm can encounter on the border and a solution how to solve this problem. Section V shows extensions of the algorithm for robust applications and problems where only the subset of state space vector $x$ is controlled. Sections VI and VII show results for two dimensional problems (inverted pendulum and DC to DC converter [5]) and section VIII shows results for a three dimensional problem (CRS system [6]). Finally, conclusions are made in section IX.

Peter Taraba is with Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Slovakia. Address: ICII (URPI) FEI STU, Ilkovicova 3, 812 19 Bratislava, Slovakia (e-mail: taraba.peter@gmail.com)

## II. Problem Formulation

Consider the problem of optimal stabilization of the continuous-time control system:

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

where $f : X \times U \to \mathbb{R}^N$ is continuous, one time differentiable, and it is assumed to be locally asymptotically controllable to the desired value $\check{x} \in X$, $x \in X \subset \mathbb{R}^N$ is the state of the system, $X$ is a region of interest, $u \in U \subset \mathbb{R}^M$ is the control input, $U$ is the compact region of admissible controls.

The goal is to construct an approximate optimal feedback $\hat{u} : x \to U$, such that time to converge to desired value $\check{x}$ will be minimal for any given point $x \in X$. This is similar to the energy function in [1]. The algorithm described in this paper creates a mesh evenly distributed on the region of interest $X$ and tries to assign to every point on the mesh its energy, which is the time needed to converge to the desired value, its optimal control value and error estimation for the energy. The algorithm starts with assigning $+\infty$ as energy for every point on the mesh, then assigns small energy values to the points closest to the desired value based on approximately how much time is needed to get to the desired value, and finally, the algorithm tries to spread the points which have finite energy further.

## III. Algorithm Description

The algorithm suggested in this paper is based on creating a mesh over region of interest $X$. As the mesh is getting smaller, function $f$ can be better approximated by linear dependency locally on the mesh because of Taylor's theorem. Hence we are considering only functions $f$ which are one time differentiable and continuous on the whole region of interest. We also assume the one time differentiability and continuity about the energy function on the region of interest, so that we can make an estimation of energies in the step 7 of the algorithm.

The main principle used in this algorithm is spreading through its neighbors, which are already able to converge to the desired value. Initially, there is a set $S$ containing points around the desired value, then set $C \subset S$, which consists of points which for a certain control value $u$ are directed closer to the desired value $\check{x}$. All the neighbor points of $C$ will be included in set $I$, which is an active set of points which might have the ability to improve their energy function as one of their neighbors has improved its energy value. Besides that

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

we also have set $F$ which keeps track of the best energy value $E$ and the best control value $u$ for each point.

Steps of the algorithm:

1) This step of the algorithm creates a mesh such that distances between points are equal. If the region of interest is $X = [x_1^{min}, x_1^{max}] \times [x_2^{min}, x_2^{max}] \times \ldots \times [x_N^{min}, x_N^{max}]$, then point on the mesh is defined as

$$p(i_1, i_2, \ldots, i_n) = \begin{pmatrix} x_1(i_1) \\ x_2(i_2) \\ \vdots \\ x_N(i_N) \end{pmatrix}$$

$$= \begin{pmatrix} x_1^{min} + \frac{i_1-1}{M_1-1}(x_1^{max} - x_1^{min}) \\ x_2^{min} + \frac{i_2-1}{M_2-1}(x_2^{max} - x_2^{min}) \\ \vdots \\ x_N^{min} + \frac{i_N-1}{M_N-1}(x_N^{max} - x_N^{min}) \end{pmatrix},$$

where $M_1, M_2, \ldots, M_N$ are numbers of points on the mesh for different coordinates and $i_j \in \{1, \ldots, M_j\}$.

2) In this step, a finite set $U$ of possible control values is created. The set $U$ will be used in several steps of the algorithm.

3) All the points created in step 1 will be added to the set $F$ which, in addition to the position of the point, also holds other information - optimal control value (which is not set initially) and the best energy value of the point, which is initially set as positive infinity as the worst case scenario (point is not able to convert to the desired value).

$$F = \{(p(i_1, \ldots, i_N), u(i_1, \ldots, i_N), E(i_1, \ldots, i_N), \varepsilon(i_1, \ldots, i_N)) : i_j \in \{1, \ldots, M_j\} \quad \forall j \in \{1, \ldots, N\}\},$$

where initially $E(i_1, \ldots, i_N) = +\infty$ and $u(i_1, \ldots, i_N)$ is not set. $\varepsilon(i_1, \ldots, i_N)$ is the error estimation of the energy value $E(i_1, \ldots, i_N)$, set initially also to $+\infty$. The way the error is estimated is described in section 5.

4) This step of the algorithm finds set $S$, which consists of the points closest to the desired value $\check{x}_j$. First, the algorithm finds $L$ indexes $i_j$ which are closest to the desired value for each coordinate

$$Z_j = \{i_j(1), \ldots, i_j(L) : |x_j(i_j(1)) - \check{x}_j| \leq \ldots \leq |x_j(i_j(L)) - \check{x}_j| \leq |x_j(k) - \check{x}_j| \; \forall k \in \{1, \ldots, M_j\} / \{i_j(1), \ldots, i_j(L)\}\}.$$

Then algorithm creates set $S$ consisting of points closest to desired value through all the combinations $Z_j$.

$$S = \{p(k_1, k_2, \ldots, k_N) \; : \; k_j \in Z_j \quad \forall j \in \{1, \ldots, N\}.$$

See fig. 1 for details ($L$ is chosen 2 in the figure). For two dimensional problems, $S$ will contain $L^2$ points, for $N$ dimensional problems, the set $S$ will contain $L^N$ points.

5) This step of the algorithm tests if $\exists u \in U$ for point $p \in S$ such that the point can get closer to the desired value than it originally was. In case it is feasible, we will add this point to controllable set $C$ of points which converge to the desired value $\check{x}$ and update point's energy and control value in set $F$.
If $p$ is the tested point and $f(p, u)$ is its direction for a control value $u \in U$ then the point can get closer to the desired value in case $\exists t > 0$ for cost function $J(p, u, t) = (p + t\, f(p, u) - \check{x})^T (p + t\, f(p, u) - \check{x})$ such that $J(t) < J(0)$. Optimal time $\hat{t}$ can be computed as

$$\hat{t}(p, u) = \frac{f^T(p, u)(p - \check{x})}{f^T(p, u) f(p, u)}.$$

The energy value for this point can be estimated as

$$E(p, u) \approx \hat{t}(p, u)$$

Point $p$ will be associated with optimal control value $\hat{u}(p) = \arg\min_{u \in U} E(p, u)$ and energy value $E(p, \hat{u}(p))$ in case $\hat{t}(p, \hat{u}(p)) > 0$.

$$C = \{(p, \hat{u}(p), E(p, \hat{u}(p))) : p \in S \;\wedge\; \hat{t}(p, \hat{u}(p)) > 0\}$$

See fig. 2, which displays this process for a two dimensional problem.

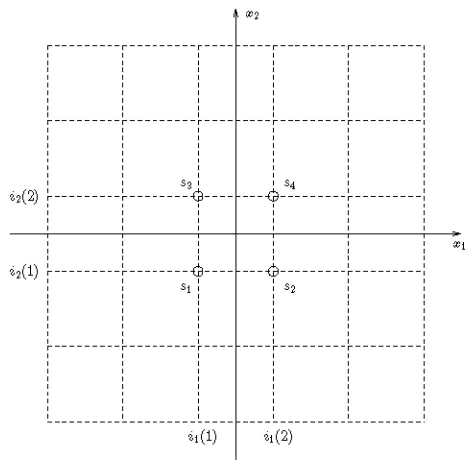

Fig. 1. Mesh over the region of interest with initial points $S = \{s_1, s_2, s_3, s_4\}$, and indexes closest to the desired value for each coordinate $Z_1 = \{i_1(1), i_1(2)\}$ and $Z_2 = \{i_2(1), i_2(2)\}$.

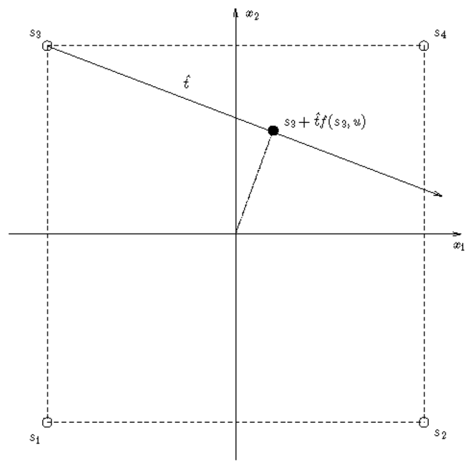

Fig. 2. Testing if point $s_3$ can point closer to the desired value with new estimation of energy with value $E(p, u) \approx \hat{t}(p, u)$.

For all the points in $C$ we update their energy values and optimal control values also in set $F$.

6) In this step, we find the initial set $I$ consisting of points which might have the ability to improve their energy value as points around them decreased their energy values. This will include all the points surrounding points in $C$. Let's define surrounding set of a point as

$$\begin{aligned} Sur(p(i_1, i_2, \dots, i_N)) = \{ & p(j_1, j_2, \dots, j_N) : \\ & |j_k - i_k| \le 1 \text{ for } \forall k \in \{1, ..., M_k\}\} \\ & / \{p(i_1, i_2, \dots, i_N)\}. \end{aligned}$$

In set $I$ in addition to remembering which points have potential to improve their energy value, we also remember the energy value of the neighbor, which was changed. The reasoning behind this is to start spreading through points with lower energy values first. This has a huge impact on the performance of the algorithm in comparison with random order of points we adapt.

$$\begin{aligned} I = \{ & (E(i_1, \dots, i_N), p(j_1, \dots, j_n)) : \\ & p(i_1, \dots, i_N) \in C \wedge \\ & \quad p(j_1, \dots, j_n) \in Sur(p(i_1, \dots, i_N))\} \end{aligned}$$

Duplicates of a point $p$ in set $I$ are not allowed and the algorithm remembers only the lowest energy value due to which point $p$ can be improved.

7) Point $p$ is selected from the set $I$ with the lowest energy associated with it and this point is also removed from this set. For all the values $u \in U$, compute the time needed to get to the point between surrounding points of point $p$.

$$\tilde{t}(p, u) = \min_{i \in \{1, \dots, N\}} \left| \frac{(x_i^{max} - x_i^{min})/(M_i - 1)}{f_i(p, u)} \right|$$

See fig. 3 for details. The new estimate of energy value

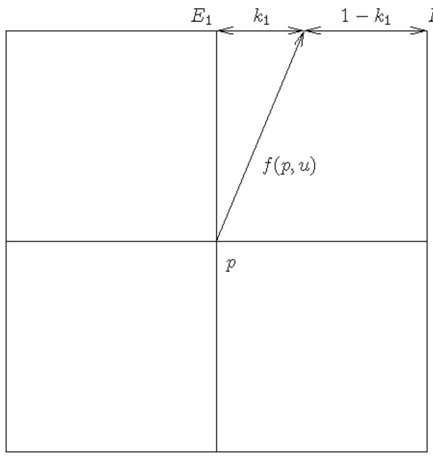

Fig. 3. Testing $u$ for point $p$. Direction $f(p,u)$ points to a point in between of other two points on the mesh $(p + \tilde{t}(p,u)f(p,u))$, whose energy values can be used to approximate new energy value of $p$.

for point $p$ and control value $u$ then can be written as

$$\tilde{E}(p, u) = \tilde{t}(p, u) + k_1 E_1 + (1 - k_1) E_2,$$

where $E_1$ and $E_2$ are energies associated with points in set $F$ and $k_1$ and $1 - k_1$ are relative distances towards these points from $(p + \tilde{t}(p, u) f(p, u))$. Value $k_1$ in different coordinates is computed following way:

$$d_i = \frac{p_i + \tilde{t}(p, u) f_i(p, u) - x_i^{min}}{x_i^{max} - x_i^{min}} (M_i - 1)$$

$$k_i = \lceil d_i \rceil - d_i$$

The new optimal control value for point $p$ will be

$$\hat{u}(p) = \arg\min_{u \in U} \tilde{E}(p, u)$$

In case the new energy $\tilde{E}(p, \hat{u}(p))$ is smaller than the energy associated with point $p$ in set $F$, we do the following:

- Update the energy and also control value for the point $p$ in set $F$ with $\hat{u}(p)$ and $\tilde{E}(p, \hat{u}(p))$
- Add surrounding points $Sur(p)$ of point $p$ to set $I$, as these points might also improve their energy values and optimal control values as their neighbor information changed. In set $I$, associate these surrounding points with energy $\tilde{E}(p, \hat{u}(p))$ through which they might get improved. If these points already exist in $I$, update the energy associated with them only in case the value $\tilde{E}(p, \hat{u}(p))$ is lower than the one already associated with them.

8) If set $I$ (set of points which might potentially improve their energy) is empty, the algorithm is done and the final solution is the set $F$. Otherwise go back to step 7.

**Remark III.1.** *This is just an implementation detail we use for set I. The algorithm uses a list of points sorted by the energy values of their neighbor which recently updated its energy value and also a hash-table of points to the same energy value. This is done in case a point we need to add to I is already there and we just need to update the value of energy it is associated with in I. The combination of sorted list and hash-table significantly improves the algorithm performance.*

**Remark III.2.** *Estimation of energy values for three dimensional problems is shown in fig. 4.*

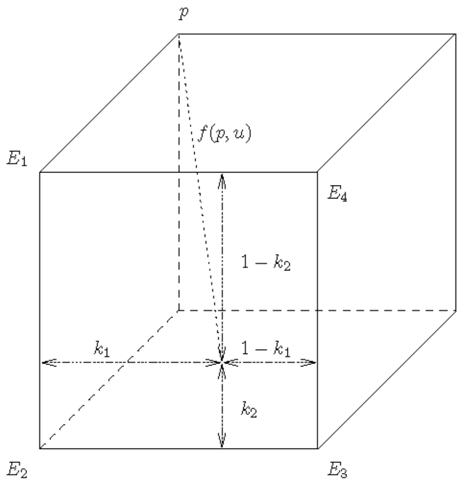

Fig. 4. Testing $u$ for point $p$. Direction $f(p,u)$ points to a point in between of other four points on the mesh $(p + \tilde{t}(p,u)f(p,u))$, whose energy values can be used to approximate a new energy value of $p$. This is now displayed for a three dimensional problem.

$$\tilde{E}(p,u) = \tilde{t}(p,u) + k_1\Big(k_2E_2 + (1-k_2)E_1\Big) + (1-k_1)\Big(k_2E_3 + (1-k_2)E_4\Big).$$

*Similarly, this can be done even for more than three dimensional problems. One can use recursion to simplify the implementation of this part of the algorithm.*

## IV. Points Pointing Only Out of Bounds

In some cases (for example an inverted pendulum), one point on the border points only outside of region of interest and hence it's energy value cannot ever be updated. This would further cause other points around it not to reach any other energy value than $\infty$. In fig. 5 we display what happens for an inverted pendulum.

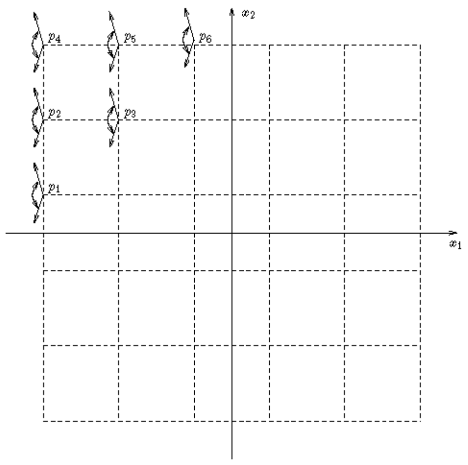

Fig. 5. Because $p_1$ points only outside of the region of interest, its energy value can't be ever updated and hence even point $p_3$ cannot ever update its value and finally because of $p_3$, point $p_6$ can't update its value either. This is happening due to the structure of the mesh rather than because of the example we run the algorithm on, as point $p_6$ can easily convert to the desired value through the region of interest for an inverted pendulum.

Due to this disadvantage of the algorithm, we approximate energy values outside of the box. See fig. 6 for details and following approximation of energy outside the box:

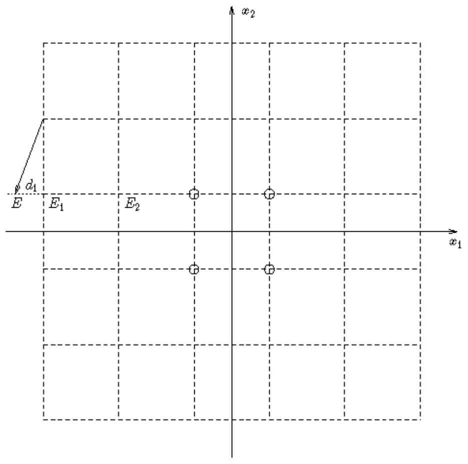

Fig. 6. Estimation of energy $E$ outside of the box based on values $E_1 < \infty$ and $E_2 < \infty$ inside of the box.

$$E \approx E_1 + (E_1 - E_2)(-d_1),$$

but only in case if $0 \geq d_1 \geq -1$.

## V. Error Estimation

When computing energy value estimations, there are two sources of error. One comes from estimation $\tilde{t}(p,u)$ ($\varepsilon_t$), the second one from estimations of $k_1$ ($\varepsilon_E$) as both approximations count on constant behavior of vector $f(p,u)$. Due to this, we can estimate the error as the following:

$$\tilde{E}(p,u) = (\tilde{t}(p,u) \pm \varepsilon_t) + ((k_1 \pm z_1)(E_1 \pm \varepsilon_1) + (1 - k_1 \mp z_1)(E_2 \pm \varepsilon_2),$$

where $\varepsilon_1$ is the estimated error associated with the same point as energy $E_1$, the same holds for $\varepsilon_2$. The first error can be estimated as

$$\varepsilon_t = \frac{x_m^{max} - x_m^{min}}{M_m - 1}\left|\frac{1}{f_m(p,u)} - \frac{1}{f_m(p + \tilde{t}(p,u)f(p,u),u)}\right|,$$

where

$$m = \arg\min_{i\in\{1,\ldots,N\}}\left|\frac{(x_i^{max} - x_i^{min})/(M_i - 1)}{f_i(p,u)}\right|.$$

The second part of the error comes from the estimation of energy between the points and can be estimated as follows:

$$\varepsilon_E = |z_1(E_1 - E_2)| + k_1\varepsilon_1 + (1-k_1)\varepsilon_2,$$

where

$$z_i = \left|\frac{\tilde{t}(p,u)(f_i(p,u) - f_i(p + \tilde{t}(p,u)f(p,u),u))}{x_i^{max} - x_i^{min}}(M_i - 1)\right|$$

for i-th coordinate.

## VI. Extensions of the algorithm

For certain problems, such as the DC to DC converter example, the desired value can be set not around origin 0, but around a value we try to converge to. Also, only one of the state space variables might be optimized and hence the initial set $S$ in the algorithm might include more points. This is displayed in fig. 7.

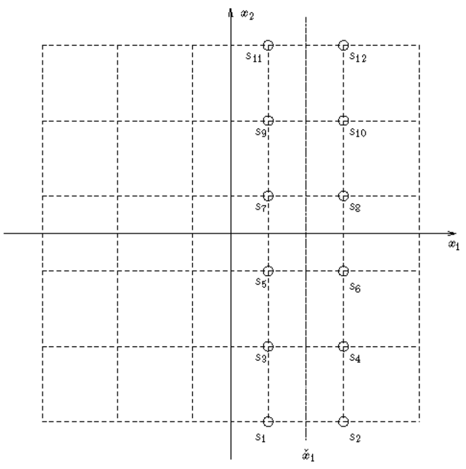

Fig. 7. This is how we extend set $S = \{s_1, \ldots, s_{12}\}$ for the algorithm if we optimize only over variable $x_1$ with the desired value $\check{x}_1$.

This means, that if the algorithm is not supposed to optimize over variable $x_2$, then $Z_2 = \{1, \ldots, M_2\}$, containing all the indexes in its coordinate.

For a robust problem, where function $f$ is dependent on a set of parameters $q$, function $f(p,u,q)$ is dependent on $q$ and

TABLE I
RESULTS (MAXIMUM ESTIMATION ERROR FOR A POINT ON THE MESH AND COMPUTATION TIME IN SECONDS) FOR THE INVERTED PENDULUM FOR DIFFERENT MESH SIZES.

| Mesh Size | 50x50 | 75x75 | 100x100 |
|---|---|---|---|
| $\varepsilon_{max}$ | 0.1341 | 0.0887 | 0.0711 |
| Computation time | 6s | 13s | 22s |

TABLE II
RESULTS (MAXIMUM ESTIMATION ERROR FOR A POINT ON THE MESH AND COMPUTATION TIME IN SECONDS) FOR THE DC TO DC CONVERTER FOR DIFFERENT MESH SIZES.

| Mesh Size | 100x100 | 150x150 | 200x200 |
|---|---|---|---|
| $\varepsilon_{max}$ | 0.4748 | 0.3130 | 0.2457 |
| Computation time | 6s | 14s | 29s |

we have several sets of parameters $Q$, we estimate energy the following way:

$$\tilde{E}(p,u) = \sum_{q\in Q}(\tilde{t}(p,u,q) + k_1(q)E_1 + (1-k_1(q))E_2).$$

## VII. EXAMPLE 1

We use a single inverted pendulum to demonstrate the algorithm on a two-dimensional control problem. For simulating such a system, we use the following simplified equations:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \sin(x_1) + u$$

We use control boundaries $U = [-3,3]$ (100 evenly distributed control values are used) and the region of interest $X = [-1,1] \times [-1,1]$. Results summary is in table I. Trajectories of 6x6 points one can see in fig. 9.

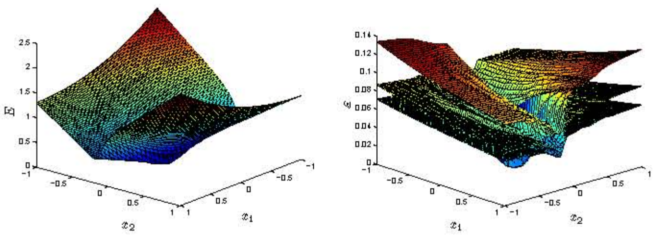

Fig. 8. Inverted Pendulum. On the left, energy function $E(i_1, i_2)$ for a mesh 100x100. On the right, error estimation for meshes 100x100, 75x75, 50x50 from bottom to top. Error estimation decreases with increasing mesh size.

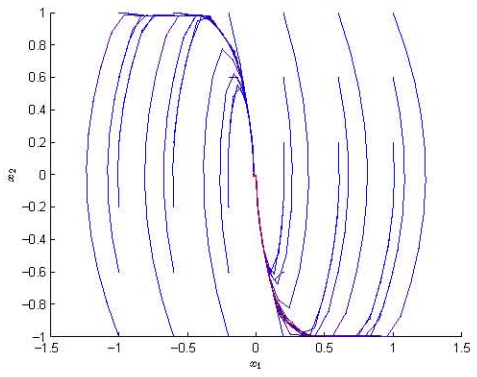

Fig. 9. Inverted Pendulum. On the region of interest 6x6 points were chosen and their convergence to origin can be seen. The trajectories are more blue towards time $t = 0$, and more red towards time $t = 3$.

## VIII. EXAMPLE 2

The second example is DC to DC converter

$$\dot{x}_1 = 0.25(x_2 - I_L)$$

$$\dot{x}_2 = -x_1 - x_2 + u$$

and the region of interest is $X = [-1,1] \times [-1,1]$ with control $U = [-1,1]$ (11 evenly distributed control values on this interval is used). A robust solution is found using three different possible loads $I_L \in \{-0.2, 0.1, 0.3\}$. Result summary is in table II. Trajectories of 6x6 points are displayed in fig 11.

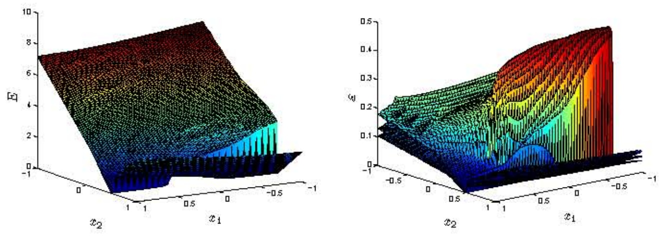

Fig. 10. DC to DC converter. On the left, energy function $E(i_1, i_2)$ for a mesh 200x200. On the right, error estimation for meshes 100x100, 150x150, 200x200 from bottom to top. Error estimation decreases with increasing mesh size.

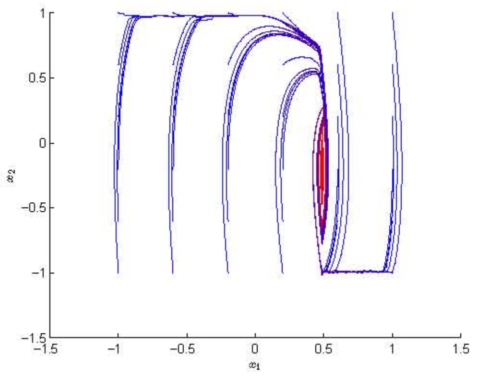

Fig. 11. DC to DC converter. On the region of interest 6x6 points were chosen and their convergence to desired value can be seen. The trajectories are more blue towards time $t = 0$, and more red towards time $t = 10$.

## IX. EXAMPLE 3

The last example is a three dimensional example in order to show that the algorithm is easy to use even on higher dimensional problems and that computations can be done in a very short time. It is convexed Reeds-Shepp (CRS) model.

$$\dot{x}_1 = u\cos(x_3)$$

$$\dot{x}_2 = u\sin(x_3)$$

$$\dot{x}_3 = v$$

The table III shows the summary results, where the error estimation decreases with higher mesh size and also shows

TABLE III
RESULTS (MAXIMUM ESTIMATION ERROR FOR A POINT ON THE MESH AND COMPUTATION TIME IN SECONDS) FOR THE CSR MODEL FOR DIFFERENT MESH SIZES.

| Mesh Size | 50x50x50 | 75x75x75 | 100x100x100 |
|---|---|---|---|
| $\varepsilon_{max}$ | 0.1751 | 0.1084 | 0.0859 |
| Computation time | 21s | 69s | 159s |

computation times in seconds on a single core machine. In fig. 12 is displayed energy level and error estimation for a chosen $x_3$ close to desired value $\tilde{x}_3$, which is middle of region of interest. In fig. 13 are displayed trajectories for 4x4x4 points converging to origin.

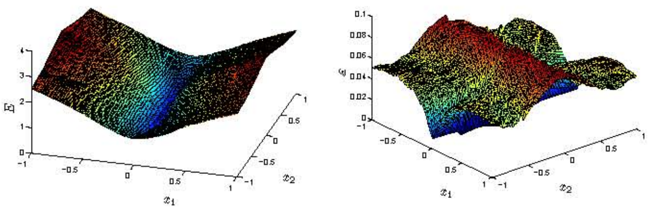

Fig. 12. CSR model. On the left, energy function $E(i_1, i_2, 50)$ for a mesh 100x100x100 with chosen $x_3$ in the middle of region of interest. On the right, error estimation for mesh 100x100x100 with chosen $x_3$ in the middle of region of interest.

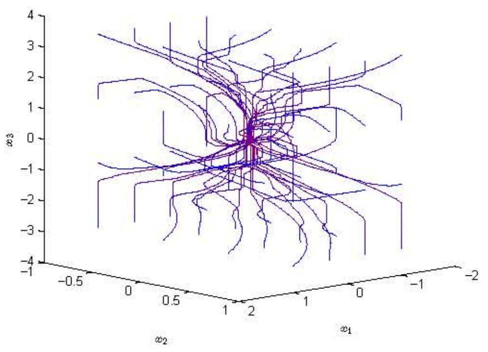

Fig. 13. CSR model. On the region of interest 4x4x4 points were chosen and their convergence to origin can be seen. The trajectories are more blue towards time $t = 0$, and more red towards time $t = 5$.

## X. CONCLUSION

We have shown a new approach for finding optimal control on a mesh, which is similar to a set-oriented approach and subdivision algorithm for optimal control. Computation times, shown on two and three dimensional examples, are better than computation times for the subdivision algorithm for optimal control. We have also shown error estimations for different mesh sizes, which adds value to the algorithm described in this paper in comparison with the subdivision algorithm for optimal control. The optimal control mesh algorithm improves upon the set-oriented approach by having mesh consisting of points evenly distributed, while the set-oriented approach needs to adapt mesh in certain regions, which makes the memory foot-print of a solution larger. The software for the algorithm described in this paper can be downloaded from [7].

## REFERENCES

[1] L. Grüne, O. Junge, A set oriented approach to optimal feedback stabilization, Systems and Control Letters, 54(2):169-180, 2005
[2] L. Grüne, O. Junge, Optimal stabilization of hybrid systems using a set oriented approach, Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems, Japan, pp. 2089-2093, 2006
[3] P. Taraba, Subdivion Algorithm for Optimal Control, Wiley, International Journal on Robust and Nonlinear Control, 2012, doi: 10.1002/rnc.2801
[4] Alberto Bemporad, Manfred Morari, Vivek Dua, Efstratios N. Pistikopoulos, The explicit linear quadratic regulator for constrained systems, Automatica, 38, 2002, 3-20
[5] B. Lincoln, A. Rantzer, Relaxing dynamic programming, IEEE Transactions On Automatic Control, Vol. 51, Issue 8, August 2006
[6] H.J. Sussmann, G. Tang, Shortest paths for the Reeds-Shepp Car: A worked out example of the use of geometric techniques in nonlinear optimal control, Technical Report No. SYNCON 91-10, Department of Mathematics, Rutgers University
[7] P. Taraba, Optirol, Available: http://www.optirol.com