UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Humanoid Robotics
Summer term 2018
May 16, 2018

Nils Rottmann
rottmann@rob.uni-luebeck.de

Exercise sheet 4 - Planning and Learning

Please prepare the following exercises for the upcoming tutorial.

Task 1: Path Planning

a) The original RRT algorithm has been proposed in [LaValle, 1998]. The main idea is to build a tree of feasible trajectories, rooted at the initial state $\boldsymbol{x}_{\text{init}}$. The algorithm starts with a graph which includes a single vertex. With each iteration a new vertex is created. Therefore, a random point $\boldsymbol{x}_{\text{rand}} \in \mathcal{X}_{\text{free}}$ from the search space is sampled. The nearest vertex $\boldsymbol{x}_{\text{nearest}}$ to $\boldsymbol{x}_{\text{rand}}$ from the graph $\mathcal{V}$ is used in order to generate a new point $\boldsymbol{x}_{\text{new}}$ using a steering algorithm. If the connection between $\boldsymbol{x}_{\text{nearest}}$ and $\boldsymbol{x}_{\text{new}}$ is free from obstacles, the new point $\boldsymbol{x}_{\text{new}}$ is added to the graph $\mathcal{V}$ and an edge $(\boldsymbol{x}_{\text{new}}, \boldsymbol{v})$ to the set of edges $\mathcal{E}$. The algorithm is stopped as soon as a predefined number of iteration steps n is reached. The shortest path from the initial state to the goal region is generated by starting at a node in the goal region and go backwards along the tree until the initial state is reached. In algorithm 1 an outline of the RRT algorithm is given. The search space $\mathcal{X}$ is divided into an obstacle-free space $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ and an obstacle space $\mathcal{X}_{\text{obstacle}} \subseteq \mathcal{X}$. A connection between two points is feasible if a straight line between the two do not intersect with space of obstacles $\mathcal{X}_{\text{obstacle}}$. In Figure 1 an example is given, where the algorithm is used to find a trajectory from the initial position $\boldsymbol{x}_{\text{init}} = [0 \quad 0]^\top$ to the goal position $\boldsymbol{x}_{\text{goal}} = [1.2 \quad 1.2]^\top$. The blue crosses are the randomly generated points $\boldsymbol{x}_{\text{rand}}$, the black lines represent the tree structure and the red path is the path with lowest cost from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$. The initial and goal states are marked with red circles, where the initial state is located in the bottom left corner.

b) In [Karaman and Frazzoli, 2011] the RRT* algorithm has been proposed. Both algorithms, RRT and RRT*, are probabilistically complete. Probabilistically completeness means that, if a solution exists, the probability that a solution is produced approaches 1 if sufficient time is spent. The main difference between RRT and RRT* is, that RRT* is not only probabilistically complete but also asymptotically optimal. Hence, the cost of the calculated solution converges to the optimum. In order to achieve this goal, RRT* uses rewiring of the tree in every iteration step. Therefore, a set $\mathcal{X}_{\text{near}}$ from $\mathcal{V}$ with vertices near $\boldsymbol{x}_{\text{new}}$ is generated. The radius defining if a vertex is near another can be dependent of the cardinality of $\mathcal{V}$. In order to generate an optimal path, an additive cost function is used and all points in $\mathcal{X}_{\text{near}}$ and $\boldsymbol{x}_{\text{nearest}}$ are checked in order to find the connection with the lowest cost as long as such a connection is obstacle free. The lowest cost connection $\{(\boldsymbol{x}_{\text{min}}, \boldsymbol{x}_{\text{new}})\}$ is added to the edge set $\mathcal{E}$. If there is a feasible connection from

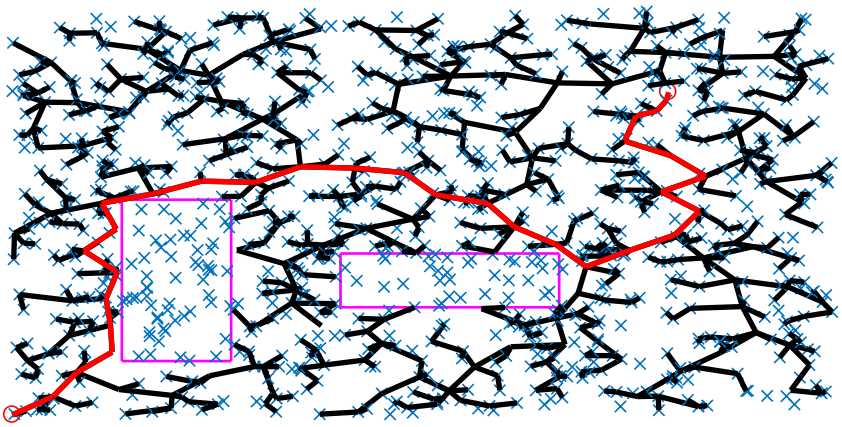

Figure 1 Rapidly-exploring Random Tree example. The branches of the tree are depicted in black and the final trajectory in red. Obstacles are represented in magenta.

Algorithm 1 RRT, adjusted from [Karaman and Frazzoli, 2011]

$\mathcal{V} \leftarrow \{\boldsymbol{x}_{\text{init}}\}; \mathcal{E} \leftarrow \emptyset$
for $i = 1, \ldots, n$ **do**
  $\boldsymbol{x}_{\text{rand}} \leftarrow \text{SampleRandom}$
  $\boldsymbol{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{V}, \boldsymbol{x}_{\text{rand}})$
  $\boldsymbol{x}_{\text{new}} \leftarrow \text{Steer}(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{rand}})$
  if $\text{ObstacleFree}(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{new}})$ **then**
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\boldsymbol{x}_{\text{new}}\}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{new}})\}$
  end if
end for
return $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

$\boldsymbol{x}_{\text{new}}$ to $\boldsymbol{x}_{\text{near}} \in \mathcal{X}_{\text{near}}$ which produces a lower cost path as the cost to reach $\boldsymbol{x}_{\text{near}}$ at this time, the edge $\{(\boldsymbol{x}_{\text{parent}}, \boldsymbol{x}_{\text{near}})\}$, which connects the point with the tree, is deleted and a new edge $\{(\boldsymbol{x}_{\text{new}}, \boldsymbol{x}_{\text{near}})\}$ added. In algorithm 2 the RRT* algorithm is presented and in Figure 2 an example is shown. The algorithm is used to find a trajectory from the initial position $\boldsymbol{x}_{\text{init}} = [0 \quad 0]^\top$ to the goal position $\boldsymbol{x}_{\text{goal}} = [1.2 \quad 1.2]^\top$. The green lines are representing the rewiring.

c) In this section the STOMP algorithm [Kalakrishnan et al., 2011] is reviewed. Here, the algorithm for a 1-dimensional trajectory is derived. For higher dimensions the derivation is similar. Considering a 1-dimensional discretized trajectory given by the vector $\boldsymbol{\theta} \in \mathbb{R}^N$, where N is the number of waypoints. The initial waypoint as well as the goal waypoint are given and kept fixed. The optimization problem is given by

$$\min_{\hat{\boldsymbol{\theta}}} \mathbb{E}\left[\sum_{i=1}^{N} q(\hat{\boldsymbol{\theta}}_i) + \frac{1}{2}\hat{\boldsymbol{\theta}}^\top \boldsymbol{R}\hat{\boldsymbol{\theta}}\right], \tag{1}$$

where $\hat{\boldsymbol{\theta}} = \mathcal{N}(\boldsymbol{\theta}, \boldsymbol{\Sigma})$ is a noisy parameter vector and $q(\hat{\boldsymbol{\theta}}_i)$ the state dependent cost. The matrix $\boldsymbol{R}$ is a positive semi-definite matrix which represents the control costs. Hence, let $\boldsymbol{A}$ be a finite difference matrix then $\boldsymbol{\theta}^\top \boldsymbol{R}\boldsymbol{\theta}$ with $\boldsymbol{R} = \boldsymbol{A}^\top \boldsymbol{A}$ leads to the sum of squared accelerations

$$\ddot{\boldsymbol{\theta}}^\top \ddot{\boldsymbol{\theta}} = \boldsymbol{\theta}^\top (\boldsymbol{A}^\top \boldsymbol{A})\boldsymbol{\theta}. \tag{2}$$

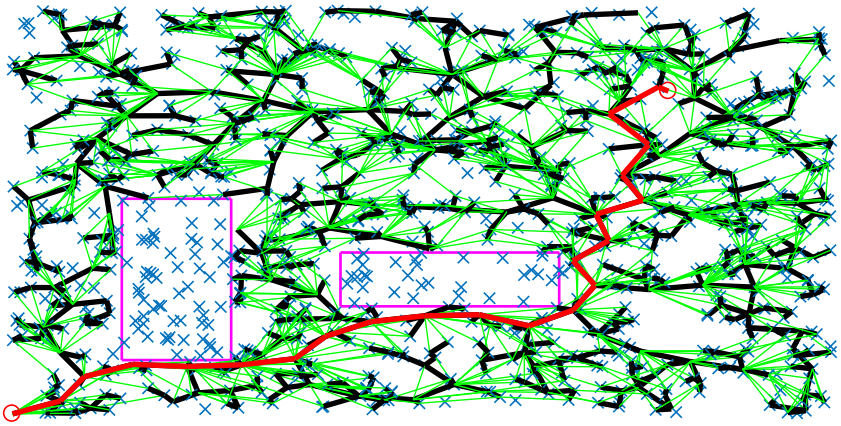

Figure 2 RRT* example. The rewiring is depicted using green lines.

Algorithm 2 RRT*, adjusted from [Karaman and Frazzoli, 2011]

$\mathcal{V} \leftarrow \{\boldsymbol{x}_{\text{init}}\}; \mathcal{E} \leftarrow \emptyset$
for $i = 1, \ldots, n$ **do**
  $\boldsymbol{x}_{\text{rand}} \leftarrow \text{SampleRandom}$
  $\boldsymbol{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{V}, \boldsymbol{x}_{\text{rand}})$
  $\boldsymbol{x}_{\text{new}} \leftarrow \text{Steer}(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{rand}})$
  if $\text{ObstacleFree}(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{new}})$ **then**
    $\mathcal{X}_{\text{near}} \leftarrow \text{Near}(\mathcal{V}, \boldsymbol{x}_{\text{new}}, r(\text{card}(\mathcal{V})))$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\boldsymbol{x}_{\text{new}}\}$
    $\boldsymbol{x}_{\text{min}} \leftarrow \boldsymbol{x}_{\text{nearest}}$
    $c_{\text{min}} \leftarrow \text{Cost}(\boldsymbol{x}_{\text{nearest}}) + c(\text{Line}(\boldsymbol{x}_{\text{nearest}}, \boldsymbol{x}_{\text{new}}))$
    for all $\boldsymbol{x}_{\text{near}} \in \mathcal{X}_{\text{near}}$ **do**
      if $\left(\begin{array}{l} \text{ObstacleFree}(\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{new}}) \quad \wedge \\ (\text{Cost}(\boldsymbol{x}_{\text{near}}) + c\left(\text{Line}(\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{new}})\right)) < c_{\text{min}} \end{array} \right)$ **then**
        $\boldsymbol{x}_{\text{min}} \leftarrow \boldsymbol{x}_{\text{near}}$
        $c_{\text{min}} \leftarrow \text{Cost}(\boldsymbol{x}_{\text{near}}) + c(\text{Line}(\boldsymbol{x}_{\text{near}}, \boldsymbol{x}_{\text{new}}))$
      end if
    end for
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\boldsymbol{x}_{\text{min}}, \boldsymbol{x}_{\text{new}})\}$
    for all $\boldsymbol{x}_{\text{near}} \in \mathcal{X}_{\text{near}}$ **do**
      if $\left(\begin{array}{l} \text{ObstacleFree}(\boldsymbol{x}_{\text{new}}, \boldsymbol{x}_{\text{near}}) \quad \wedge \\ (\text{Cost}(\boldsymbol{x}_{\text{new}}) + c(\text{Line}(\boldsymbol{x}_{\text{new}}, \boldsymbol{x}_{\text{near}}))) < \text{Cost}(\boldsymbol{x}_{\text{near}}) \end{array} \right)$ **then**
        $\boldsymbol{x}_{\text{parent}} \leftarrow \text{Parent}(\boldsymbol{x}_{\text{near}})$
      end if
      $\mathcal{E} \leftarrow (\mathcal{E} \setminus \{(\boldsymbol{x}_{\text{parent}}, \boldsymbol{x}_{\text{near}})\}) \cup \{(\boldsymbol{x}_{\text{new}}, \boldsymbol{x}_{\text{near}})\}$
    end for
  end if
end for
return $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

In order to find the optimum of the optimization problem, the derivative of (1) is set to zero:

$$\nabla_{\hat{\boldsymbol{\theta}}} \left(\mathbb{E} \left[\sum_{i=1}^{N} q(\hat{\boldsymbol{\theta}}_i) + \frac{1}{2} \hat{\boldsymbol{\theta}}^T \boldsymbol{R} \hat{\boldsymbol{\theta}} \right] \right) = 0. \tag{3}$$

This leads to the expectation of the waypoints

$$\mathbb{E}(\hat{\boldsymbol{\theta}}) = -\boldsymbol{R}^{-1} \mathbb{E} \left(\nabla_{\hat{\boldsymbol{\theta}}} \left[\sum_{i=1}^{N} q(\hat{\boldsymbol{\theta}}_i) \right] \right) = -\boldsymbol{R}^{-1} \delta \hat{\boldsymbol{\theta}}_{\text{G}}. \tag{4}$$

In order to solve (4), STOMP proposes an estimated gradient

$$\delta \hat{\boldsymbol{\theta}}_{\text{G}} = \int \delta \boldsymbol{\theta} \, \mathrm{d}\boldsymbol{P}. \tag{5}$$

In comparison to other approaches [Ratliff et al., 2009], which use the analytical gradient of the cost function to calculate an iterative gradient update, the STOMP approach allows the utilization of non-differentiable or non-smooth cost functions. With (5) the expectation of $\delta\boldsymbol{\theta}$, which is the noise in $\hat{\boldsymbol{\theta}}$, under the probability metric $\boldsymbol{P}$, can be calculated. The probability metric is defined as $\boldsymbol{P} = \exp\left(-\frac{1}{\lambda}\boldsymbol{S}(\hat{\boldsymbol{\theta}})\right)$ with

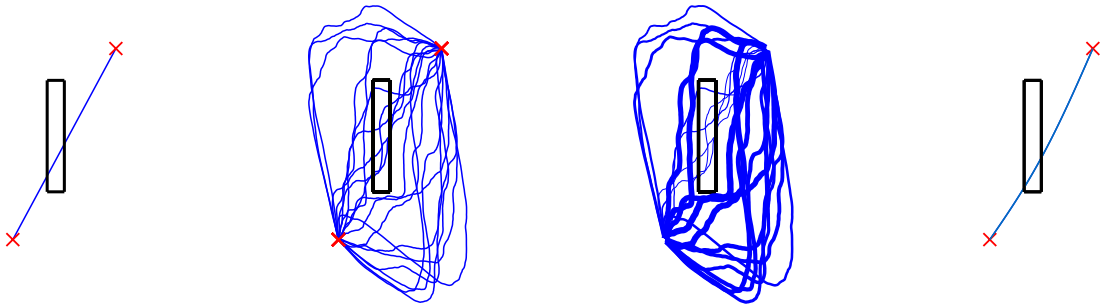

Figure 3 STOMP Procedure: Figure (a) shows the initial trajectory. In (b), the randomly sampled trajectories are depicted. Figure (c) shows the probabilities of the sampled trajectories. The higher the probability of a trajectory, the thicker the trajectory is drawn. In (d) the updated trajectory is presented.

$\boldsymbol{S}(\hat{\boldsymbol{\theta}}) = \left[\sum_{i=1}^{N} q(\hat{\boldsymbol{\theta}}_i)\right]$ being the state dependent cost defined on the trajectory. Considering the statements above, the stochastic gradient can be written as

$$\delta\hat{\boldsymbol{\theta}}_G = \int \exp\left(-\frac{1}{\lambda}\boldsymbol{S}(\boldsymbol{\theta})\right) \delta\boldsymbol{\theta}\ \mathrm{d}\delta(\boldsymbol{\theta}). \tag{6}$$

Using the given probability metric with $\boldsymbol{S}(\hat{\boldsymbol{\theta}})$ being the cost of a path, paths with higher costs get a lower probability. Hence, they have a lower contribution to the update of the initial trajectory as paths with lower costs. In Figure 3 the STOMP procedure is exemplarily shown. The first figure shows the initial trajectory, a linear path from the start to the goal point, where the rectangular represents an obstacle, which should not be passed by the trajectory. The second picture shows the randomly sampled trajectories and the third depicts the different probabilities of the sampled trajectories. The higher the probability of a trajectory, the thicker the trajectory is drawn. The last picture shows the new generated trajectory. The algorithm starts again using this trajectory as new initial one. As exploration strategy for the algorithm, samples with variance $\boldsymbol{\Sigma} = \boldsymbol{R}^{-1}$ are used. This ensures low control costs and smoothness. In Figure 4 thirty random samples drawn from a zero mean normal distribution $\boldsymbol{\epsilon} = \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{R}^{-1}\right)$ are shown. In algorithm 3 pseudo-code for the reviewed STOMP method is presented.

Task 2: Reinforcement Learning

a) The general Idea of Reinforcement Learning is to find a policy $\pi(\boldsymbol{u}|\boldsymbol{x}, \boldsymbol{\theta})$ which minimizes a cost function $\boldsymbol{R}(\boldsymbol{\tau})$ for a trajectory $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \dots)$. Here $\boldsymbol{u}$ are the control inputs into the system, $\boldsymbol{x}$ are the states of the system and $\boldsymbol{\theta}$ are the policy parameters. In Figure 5 a general overview over the subareas in Reinforcement Learning is shown.

b) From exercise 0 we know that the differential equation for the pendulum can be written as

$$\ddot{\varphi} + \omega^2 \sin(\varphi) = u; \qquad \omega = \sqrt{\frac{g}{L}} \tag{7}$$

for $m = 1$. With the given restriction of $u_{\max} = 5$ we can show that can not swing up the pendulum at once. Therefore we simulate the system with $u = u_{\max}, \forall t$ which leads to a result as depicted in Figure 6.

Algorithm 3 STOMP, based on [Kalakrishnan et al., 2011]

- **Require**
  - Start and goal positions x_0 and x_N
  - An initial 1-D discretized trajectory vector $\boldsymbol{\theta}$
  - A state-dependent cost function $q(\boldsymbol{\theta}_i)$
- Precompute
  - $\boldsymbol{A}$: A central finite difference matrix
  - $\boldsymbol{R}^{-1} = (\boldsymbol{A}^\top \boldsymbol{A})^{-1}$
  - $\boldsymbol{M} = \boldsymbol{R}^{-1}$, each column scaled such that the maximum element is $1/N$

procedure STOMP(Repeat until convergence of trajectory cost $Q(\boldsymbol{\theta})$)
  for $k = 1, \ldots, K$ **do**
    $\boldsymbol{\epsilon}_k = \mathcal{N}(0, \boldsymbol{R}^{-1})$
    $\hat{\boldsymbol{\theta}}_k = \boldsymbol{\theta} + \boldsymbol{\epsilon}_k$
    for $i = 2, \ldots, N-1$ **do**
      $S\left(\hat{\boldsymbol{\theta}}_{k,i}\right) = q(\hat{\boldsymbol{\theta}}_{k,i})$
    end for
  end for
  for $k = 1, \ldots, K$ **do**
    for $i = 2, \ldots, N-1$ **do**
      $P(\hat{\boldsymbol{\theta}}_{k,i}) = e^{-\frac{1}{\lambda} S(\hat{\boldsymbol{\theta}}_{k,i})}$
    end for
    $P(\hat{\boldsymbol{\theta}}_k) = \sum_{i=1}^{N-1} (P(\hat{\boldsymbol{\theta}}_{k,i}))$
  end for
  $P(\hat{\boldsymbol{\theta}}_k) \leftarrow \frac{P(\hat{\boldsymbol{\theta}}_k)}{\sum_{k=1}^{K} \hat{\boldsymbol{\theta}}_k)}$
  $\delta\hat{\boldsymbol{\theta}} = \sum_{k=1}^{K} P(\hat{\boldsymbol{\theta}}_k)\boldsymbol{\epsilon}_k$
  $\delta\boldsymbol{\theta} = \boldsymbol{M}\delta\hat{\boldsymbol{\theta}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$
  $Q(\boldsymbol{\theta}) = \sum_{i=1}^{N} q(\boldsymbol{\theta}_i) + \frac{1}{2}\boldsymbol{\theta}^\top \boldsymbol{R}\boldsymbol{\theta}$
end procedure

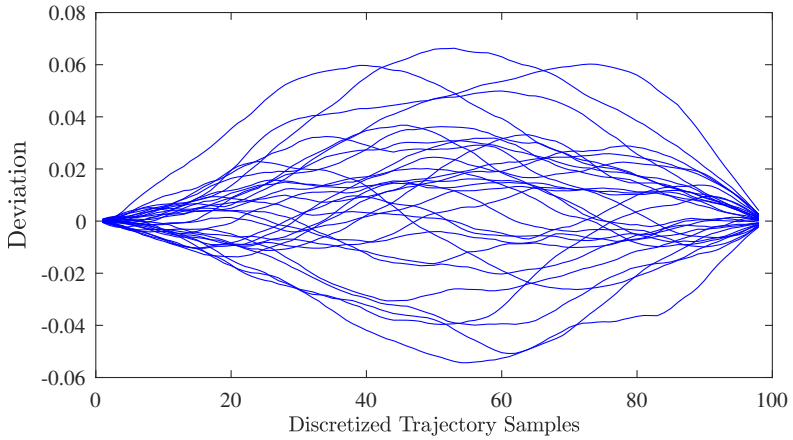

Figure 4 Thirty random samples of the noise drawn from a zero mean normal distribution $\boldsymbol{\epsilon} = \mathcal{N}\left(\mathbf{0}, \boldsymbol{R}^{-1}\right)$.

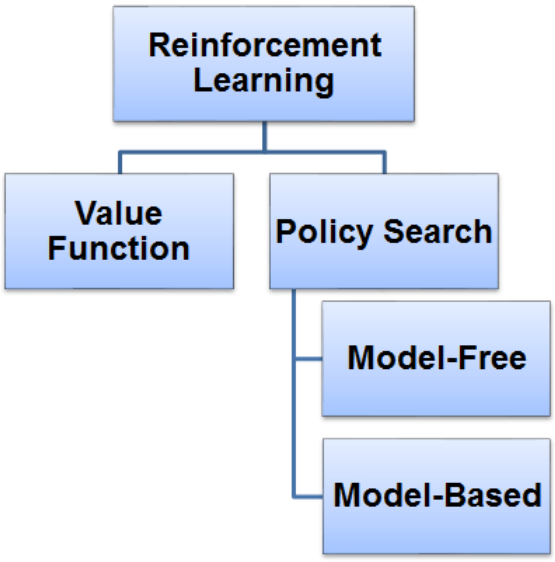

Figure 5 Reinforcement Learning

Thus we have to swing up the pendulum first to one and then to the other side in order to get enough momentum. Therefore we will use the following parametrized policy for the input signals $\boldsymbol{u}$

$$u(t) = \begin{cases} K_{p1}(g_1 - \varphi) - K_{d1}\dot{\varphi} & t \leq d_1 \\ K_{p2}(g_{\text{end}} - \varphi) - K_{d2}\dot{\varphi} & t > d_1 \end{cases} . \tag{8}$$

Here $g_{\text{end}} = \pi$ is the end position at time T. The parameters which we have to determine are g_1, the via-point, K_{p1}, K_{d1}, K_{p2} and K_{d2} are the gains for the PD-controller and d_1 is the time at when we change from steering to the via-point into steering to the end position. In the following we will write

$$\boldsymbol{\theta} = [K_{p1}, K_{d1}, K_{p2}, K_{d2}, d_1, g_1]^T. \tag{9}$$

In order to find convenient parameters we will define a cost function which has to be optimized. The intrinsic costs are defined as

$$L(\tau; \boldsymbol{\theta}) = \sum_{t=0}^{T} l(\varphi_t, u_t, t; \boldsymbol{\theta}) + c_p(\varphi_t, u_t) \tag{10}$$

with

$$l(\varphi_t, u_t, t; \boldsymbol{\theta}) = \frac{1}{N}(\varphi_t - g_i)^2. \tag{11}$$

Here N is the number of evaluations. We have decomposed our movement into two phases as described above such that g_1 and is active as long $t \leq d_1$ and otherwise g_{end} is active. We consider the second term in 10 to be zero. Finally we require extrinsic costs which evaluate the overall task performance. We set them to

$$C = 100(x_T - g_{\text{end}}). \tag{12}$$

Thus, we have to reach the desired end state, otherwise our costs will explode. We can now start to find convenient parameters for our policy. Therefore we have to find the minimum for our costs. We will use Covariance Matrix Adaptation Evolution Strategy in order to search for a minimum. Matlab code is

available at `https://www.lri.fr/~hansen/cmaes_inmatlab.html#matlab`. As a optimization result we get

$$\boldsymbol{\theta} = [K_{p1}, K_{d1}, K_{p2}, K_{d2}, d_1, g_1]^T = [6.13, -5.95, 33.83, 6.62, 1.74, -0.22]^T \tag{13}$$

as parameters. In Figure 7 the simulation result is shown.

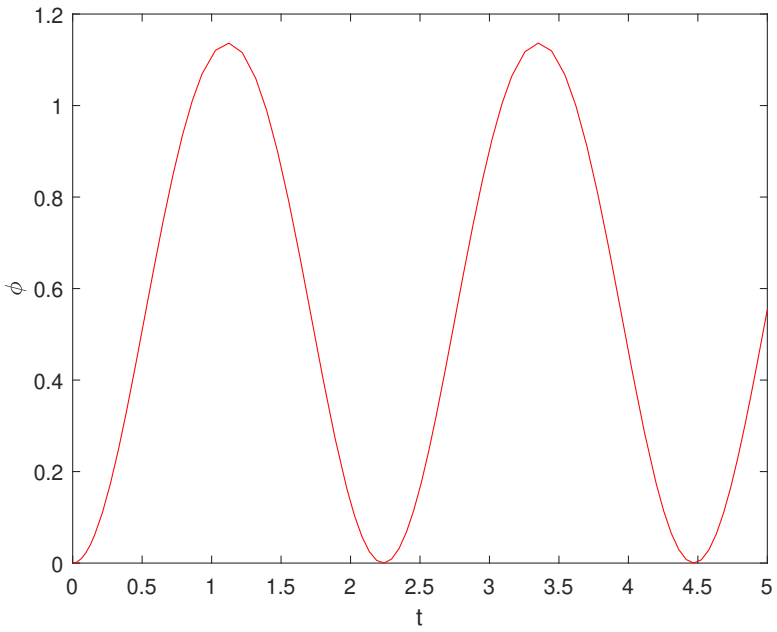

Figure 6 Pendulum with $u = u_{\max}$

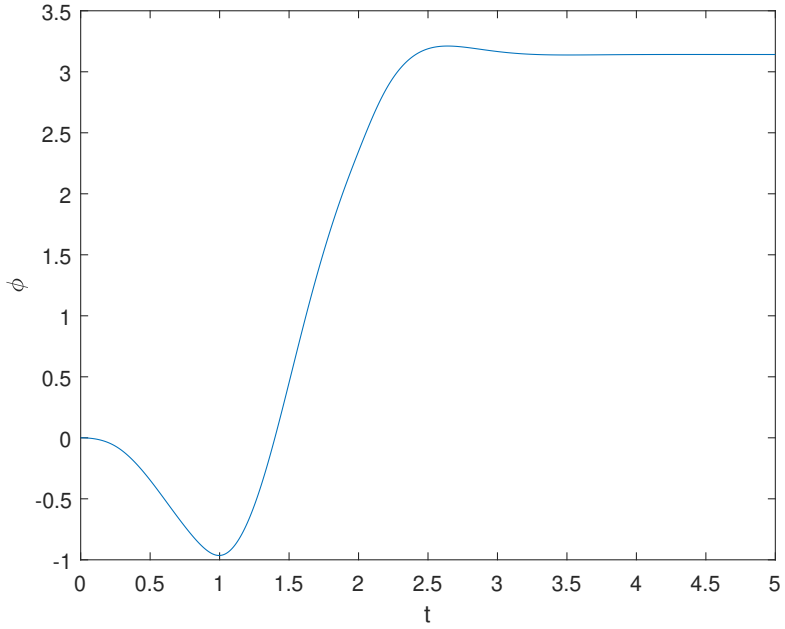

Figure 7 Up-Swing with via-point

Literatur

[Kalakrishnan et al., 2011] Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). STOMP: Stochastic Trajectory Optimization for Motion Planning. In *International Conference on Robotics and Automation (ICRA), 2011*, pages 4569–4574. IEEE.

[Karaman and Frazzoli, 2011] Karaman, S. and Frazzoli, E. (2011). Sampling-based Algorithms for Optimal Motion Planning. *The International Journal of Robotics Research*, 30(7):846–894.

[LaValle, 1998] LaValle, S. M. (1998). Rapidly-exploring Random Trees: A new Tool for Path Planning.

[Ratliff et al., 2009] Ratliff, N., Zucker, M., Bagnell, J. A., and Srinivasa, S. (2009). CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In *International Conference on Robotics and Automation (ICRA), 2009*, pages 489–494. IEEE.