Minimum-time Control of the Acrobot

Gary Boone
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280
gboone@cc.gatech.edu, www.cc.gatech.edu/~gboone

Abstract—The Acrobot, a two-link arm that is unactuated at the first joint, is challenging to control because it is an underactuated, nonlinear, continuous system. We describe a direct search algorithm for finding swingup trajectories for the Acrobot. The algorithm uses a lookahead search that maximizes the Acrobot's total energy in an N-step window. Because the controls are extremal and the number of switches in the window limited, the algorithm is fast. Nevertheless, the resulting trajectories are near optimal.

I. INTRODUCTION

The Acrobot is a two-link robot arm powered at the elbow but free-swinging at the shoulder (Figure 1). Controlling it has become a challenging problem in nonlinear control[1], [2], [3]. The Acrobot is a member of a class of systems that are underactuated, having more degrees of freedom than actuators. Many systems are underactuated, including manipulator arms on diving vessels or spacecraft, non-rigid-body systems, and balancing systems such as unicycles or dynamically-stable legged robots.

Previous researchers have hand designed Acrobot controllers based on linearizing parts of the state equations. Our approach places the Acrobot in an optimal control framework that seeks minimum-time trajectories for swingup tasks. We chose the optimal control approach for two reasons. First, we are interested in the automatic design of controllers. Second, we wish to explore reinforcement learning, one method of automatic design that requires an optimal control framework[4]. This paper presents an example of a fast planner, a direct lookahead search based on the Acrobot's energy equations.

II. THE ACROBOT

The Acrobot can swing up above its pivot by applying torques at the second joint and has been compared to a gymnast swinging up above a high bar by bending at the hips. Two tasks are commonly considered. The first requires the Acrobot to swing its endpoint above a desired goal height[3]. In the results described below, the goal height is a link length above the shoulder joint. Another Acrobot task is to swing up to vertical, either into the capture region of a balancing controller[5], [2], or into a balanced position, that is, a vertical position with near-zero velocities. We will refer to these as the "height" and "handstand" tasks, respectively.

Prior discussions of the Acrobot have appeared in the robotics and control community, focusing on the design of controllers for balancing[6], [1], swingup and balance tasks[7], [8], [9], [2], [10], [11], and even walking[12], [13]. Acrobot swingup trajectories have also been generated by artificial intelligence and reinforcement learning approaches such as explanation-based learning[5] and Q-Learning[3].

Most previous Acrobot controllers have been based on linearization of the dynamics. Balancing controllers that maintain the Acrobot in the handstand position can be created by applying linear controllers in the neighborhood of the handstand position. Several researchers have applied feedback linearization to control the swingup from the hanging position, using feedback to create a linearizable system that can be controlled by well-understood linear system techniques. However, the Acrobot is not state feedback linearizable[1]. Nevertheless, an output function may be found that allows input/output linearization[6]. Other authors have used pseudo-linearization, in which higher order terms are ignored to allow linearization[7], or partial linearization, in which only the actuated degrees of freedom are linearized[2], [10], [11].

Typically, these approaches result in controllers for following trajectories. The trajectories required for
the swingup are determined by parameterized periodic trajectories [12], heuristics designed to pump energy into the system [2], [10], [11], or gradient projection methods [9]. A separate balancing controller may be required to complete the swingup and maintain balance once the Acrobot reaches the controller’s capture region.

Our work emphasizes two aspects of the problem that have received little attention. First, we seek optimal trajectories, minimizing the total swingup time. Second, we allow only bounded controls. The controller cannot apply arbitrary torques, but instead is limited to ±1 Nm. This limitation makes the Acrobot underpowered, requiring many swings to reaching a goal configuration. Our solution uses the Acrobot energy equations to provide a heuristic to guide an N-step, k-switch lookahead search. This approach does not require a balancing controller to complete the swing up to a handstand, allows the Acrobot to start from a stationary hanging position, and quickly produces near-optimal swingup trajectories.

III. Motion and Energy Equations

The Acrobot’s inverse dynamics can be written

\[ D(\theta) \ddot{\theta} + C(\dot{\theta}, \theta) + \Phi(\theta) = T, \]

where \( \theta = [\theta_1, \theta_2]^T \), and \( T = [0, T]^T \). \( D(\theta) \) is a symmetric, inertial acceleration matrix defined by

\[
d_{11} = m_1 l_{c1}^2 + m_2 (l_1^2 + l_{c2}^2 + 2l_1 l_{c2} \cos(\theta_2)) + I_1 + I_2
\]

\[
d_{22} = m_2 l_{c2}^2 + I_2
\]

\[
d_{12} = d_{21} = m_2 (l_2^2 + l_1 l_{c2} \cos(\theta_2)) + I_2.
\]

\( C(\dot{\theta}, \theta) \) is a Coriolis and centrifugal force vector

\[
c_1 = -m_2 l_1 l_{c2} \dot{\theta}_2^2 \sin(\theta_2) - 2m_2 l_1 l_{c2} \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_2)
\]

\[
c_2 = m_2 l_1 l_{c2} \dot{\theta}_2^2 \sin(\theta_2).
\]

\( \Phi(\theta) \) is a gravitational loading force vector

\[
\phi_1 = (m_1 l_{c1} + m_2 l_1) g \cos(\theta_1) + m_2 l_{c2} g \cos(\theta_1 + \theta_2)
\]

\[
\phi_2 = m_2 l_{c2} g \cos(\theta_1 + \theta_2),
\]

where \( g \) is the gravitational force. The other parameters are defined in Table I.

The Acrobot’s forward dynamics are given by

\[
\ddot{\theta} = D^{-1}(\theta) \left( T - C(\dot{\theta}, \theta) - \Phi(\theta) \right)
\]

\[
= \begin{bmatrix} 
  d_{22} & -d_{12} \\
  -d_{21} & d_{11}
\end{bmatrix} \begin{bmatrix} 
  T - C(\dot{\theta}, \theta) - \Phi(\theta) \\
\end{bmatrix}
\]

TABLE I. Acrobot Parameters. Using these parameters [3], the Acrobot requires several swings to reach the goal height or handstand position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_{c1}, l_{c2} )</td>
<td>Link lengths</td>
<td>1.0</td>
</tr>
<tr>
<td>( l_{c1}, l_{c2} )</td>
<td>Joint to mass center</td>
<td>0.5</td>
</tr>
<tr>
<td>( m_1, m_2 )</td>
<td>Link masses</td>
<td>1.0</td>
</tr>
<tr>
<td>( I_1, I_2 )</td>
<td>Link inertias</td>
<td>1.0</td>
</tr>
<tr>
<td>( \tau_{min}, \tau_{max} )</td>
<td>Torque range</td>
<td>((-1.0, 1.0))</td>
</tr>
<tr>
<td>( t_i )</td>
<td>Integration time step</td>
<td>0.05</td>
</tr>
<tr>
<td>( t_c )</td>
<td>Control time step</td>
<td>0.2</td>
</tr>
</tbody>
</table>

The energy equations for the Acrobot are given by the sum of the potential and kinetic energy for each link. They are also derived readily in the Lagrangian formulation of the dynamic equations for planar, two-link robot arms. For example, see Riven [14].

The kinetic energy of the Acrobot is given by

\[
K = \frac{1}{2} \dot{\theta}^T D(\theta) \dot{\theta}.
\]

\[
= \frac{1}{2} d_{11} \dot{\theta}_1^2 + d_{12} \dot{\theta}_1 \dot{\theta}_2 + \frac{1}{2} d_{22} \dot{\theta}_2^2
\]

\[
= \frac{1}{2} m_1 l_{c1} \dot{\theta}_1^2 + \frac{1}{2} l_1 \dot{\theta}_1 \dot{\theta}_2 + \frac{1}{2} m_2 l_{c2} \dot{\theta}_2^2
\]

\[
+ \frac{1}{2} m_2 l_{c2} (\dot{\theta}_2^2 + 2 \dot{\theta}_1 \dot{\theta}_2 + \dot{\theta}_1^2)
\]

\[
+ m_2 l_1 l_{c2} (\dot{\theta}_1^2 + \dot{\theta}_1 \dot{\theta}_2) \cos(\theta_2)
\]

\[
+ \frac{1}{2} L_2 (\dot{\theta}_2^2 + 2 \dot{\theta}_1 \dot{\theta}_2 + \dot{\theta}_1^2)
\]

The potential energy is based on the heights of the center of masses of the links,

\[
U = m_1 g l_{c1} \sin(\theta_1) + m_2 g l_1 \sin(\theta_1) + m_2 g l_{c2} \sin(\theta_1 + \theta_2).
\]

The total energy of the system is the sum of the kinetic and potential energy of the links, \( E = K + U \).

IV. Window Search

There are many ways to find optimal trajectories, including using Pontryagin’s Minimum Principle [15], gradient descent [9], dynamic programming, and direct search.
The Acrobat has several properties that make a direct search for swingup trajectories efficient. It has only one actuator; with the assumption of bang-bang control (Section VI), there are only two possible actions at every time step. Further, we assume a discrete time controller and, following Sutton[3], allow the Acrobat to apply a new control every 0.2 seconds. Nevertheless, exhaustively searching the space of all possible control sequences is impractical due to its large size. For example, our 82 step handstand swingup trajectory is one of 10^24 possible 82 step trajectories.

The Acrobat’s energy function provides a useful heuristic to guide a local trajectory search for a swingup trajectory. To swing up as quickly as possible, the Acrobat must pump energy into the system. Therefore, local searches that maximize the energy function can be used to construct a trajectory. Note, however, that once sufficient energy has been accumulated, the search objective changes to moving to the goal configuration rather than increasing energy. Therefore, Acrobat trajectories can be found in two phases. First, the Acrobat must pump enough energy into its motion to reach the neighborhood of the goal[11]. Then it must find a sequence of controls that moves the Acrobat into the goal region. These phases must be combined to produce an optimal trajectory.

Energy profiles of hand-generated swingup trajectories suggest an almost monotonically increasing energy function during the pumping phase, as shown in Figure 5. The energy content along an optimal trajectory does not increase at a constant rate, and may occasionally decrease, due to the nonlinearities of the Acrobat. Therefore, our algorithm searches with an N-step lookahead search. As shown in Figure 2, the algorithm repeatedly extends the current trajectory by constructing all possible N-step, k-switch extensions, where N is the window size and k is the number of switches allowed in the window. For example, there are 55 10-step extensions that have 2 switches. After each extension is constructed, the candidate trajectory is simulated and the highest energy achieved (Figure 3). The first step of this extension is added to the current trajectory and the process repeats. The algorithm slides an N-step window one time step per iteration, following the trajectory that maximizes the energy function within the window.

Pumping energy into the system is insufficient because the task is to raise the endpoint to a goal position; adding energy eventually leads to rapidly spinning the Acrobat. The energy function provides a mechanism for determining when to change the search objective from adding energy to moving to the goal position. As the lookahead search proceeds, a trajectory is eventually created with a maximum energy exceeding that of the goal energy. The goal energy is defined as the minimum potential energy in the goal region, as shown in Figure 4. Once the Acrobat's total energy exceeds the goal energy, the it has sufficient energy to reach the goal. Continuing to maximize the total system energy may no longer be effective. Some of the steps that the search algorithm has already committed to may be better used for moving into the goal configuration. Therefore, once the goal energy is reached, the algorithm backs up by one window length, doubles
the window size, and doubles the number of switches allowed. It continues to search all possible 2N-step extensions, but chooses the trajectory with the minimum time to the goal if the goal is reached by any trajectories.

Figure 4. Goal Energies. For the height task, the goal is the region above the goal line. For the handstand task, the goal is defined as a neighborhood around vertical and stationary for each link. The minimum energy required to reach the goal is given by the minimum potential energy within the goal region. Note that for both tasks, there are multiple goal configurations.

V. RESULTS

We ran extensive simulations of the Acrobot with the parameters shown in Table I using 4th order Runge-Kutta integration on a 200MHz MIPS R4400 processor. For the height task, the goal height was set at one link length above the shoulder joint. For the handstand task, the goal was defined as ±0.3 radians of vertical with velocities within ±0.3 radians/second for each link. We varied the window size between 2 and 30 control time steps and the number of switches between 1 and 10.

Table II shows that the fastest trajectories were found with a 10-step, 2-switch window for the height task and a 15-step, 2-switch window for the handstand task. For comparison, we ran the same tasks using Sarsa Q-Learning with CMAC function approximation and eligibility traces[3]. Q-Learning is a reinforcement learning algorithm for finding optimal policies. Eligibility traces allow previously visited states to explicitly receive discounted rewards. The CMAC function approximator allows generalization and was observed to provide better trajectories than table-based Q-Learning. We followed the parameters used in [3], but used 4th order Runge-Kutta integration and did not include the zero-torque action. The Q-Learning based algorithm is able to use generalization to converge to a good solution with only a relatively few evaluations of the Acrobot state for the height task. However, the training of the function approximator is slow and the algorithm is unable to find the 61-step trajectory. For the handstand task, the algorithm is unable to converge to a solution within several days of computing time.

Table II shows that the fastest trajectories were found with a 10-step, 2-switch window for the height task and a 15-step, 2-switch window for the handstand task. For comparison, we ran the same tasks using Sarsa Q-Learning with CMAC function approximation and eligibility traces[3]. Q-Learning is a reinforcement learning algorithm for finding optimal policies. Eligibility traces allow previously visited states to explicitly receive discounted rewards. The CMAC function approximator allows generalization and was observed to provide better trajectories than table-based Q-Learning. We followed the parameters used in [3], but used 4th order Runge-Kutta integration and did not include the zero-torque action. The Q-Learning based algorithm is able to use generalization to converge to a good solution with only a relatively few evaluations of the Acrobot state for the height task. However, the training of the function approximator is slow and the algorithm is unable to find the 61-step trajectory. For the handstand task, the algorithm is unable to converge to a solution within several days of computing time.

Figure 5 shows the best trajectory the algorithm found for the handstand task. To find this trajectory, the algorithm was run with a 15 step, 2 switch window. The energy profile of the swingup is also shown. As expected, the potential energy reaches the goal energy while the kinetic energy goes to zero in the goal region. Note that the control does not pump more energy into the Acrobot than is necessary to achieve the goal.

As the search proceeds, only a few switches are allowed within the window. This restriction limits the search based on the a priori knowledge that switches are relatively infrequent during pumping. However, it does not prevent the final trajectory from including rapid switches, as shown in Figure 5. Figure 6 shows the effects of varying the window size and the number of switches within the window. The graph shows that allowing only one switch is insufficient for effective searches. However, beyond one switch, there is little difference between searches with different numbers of switches between 2 and 10 switches. As shown in Table II, additional switches lead to many more trajectories to evaluate. Therefore, a smaller number of switches is preferred. Table II also demonstrates the efficiency of the algorithm. The total number of eval-
Figure 6. Window Size Variation. For the height tasks, the direct search produces similar results despite variations in the number of switches within the window. Searches with only one switch were unable to find the shortest trajectory, which is 61 steps for the height task. All of the other window sizes found it with a window size of 10.

VI. DISCUSSION

Controlling the Acrobot is a difficult problem because it is underactuated and nonlinear. For our studies, it is also underpowered, requiring many swings before reaching the goal. As a result, the space of admissible trajectories is large, creating difficulties with locally optimal solutions and prohibiting exhaustive searches. The Acrobot state space is four dimensional and continuous, making it difficult to represent. State-space grids, used by dynamic programming and some reinforcement learning algorithms such as Q-Learning, are expensive to store and introduce discretization effects. Finally, the handstand task has an extremely small goal region, increasing the difficulty of finding good solutions.

Two previous successful approaches to the Acrobot problem have been gradient descent and hand-designed controllers based on local linearization and heuristics. The gradient descent technique requires an initial trajectory and produces a locally optimized result. The N-step lookahead search avoids the need for an initial trajectory by using the energy heuristic. Controllers based on linearization are often only locally stable and can be difficult to tune[10]. Feedback linearization can require unbounded control[16]. The previous Acrobot controllers based on feedback linearization have required unbounded controls.

Typically, the trajectory is generated by a heuristic that pumps energy into the system by swinging the lower link in phase with the upper link[5], [2], [10], [11]. This heuristic will cause the acrobot to swing higher with each swing. However, pumping is not the right activity when the system is near the goal. For the handstand tasks, the planner must find a path that brings the Acrobot to a stop in the inverted, vertical position. But the pumping heuristic will cause the Acrobot to pump more and more energy into the system, spinning faster at the shoulder rather than slowing appropriately.

One solution is to switch to a new controller. Once the Acrobot enters the capture region of a balancing controller, the balancing controller is activated and the Acrobot balances. Unfortunately, the resulting path to vertical may not be time optimal. A more serious problem is that the pumping heuristic may not bring the Acrobot into the capture region of the controller. For some previous approaches, the extent of the capture region of the balancing controller must be known to decide when to switch to the balancing controller[7]. Further, it is possible for the combined system to be unstable[8].

The algorithm described here finds a single optimal trajectory that reaches the goal without switching to a balancing controller near the goal. To remain upright, a balancing controller is required. However, our algorithm produces trajectories that reach the goal directly and quickly; a balancing controller is not required to complete the path to the goal. This approach also reduces the need to know the extent of the capture region of the balancing controller.

Direct search is practical for finding Acrobot trajectories because there are only two possible actions at each time step. Although the Acrobot may be defined with continuous actions, the optimizing controls will drive the actuators to their extremes. This is shown
by the application of Pontryagin’s Minimum Principle [15], which states that optimal trajectories minimize the Hamiltonian, a function that combines the state equations and optimization criteria. For systems with state and criteria that are both linear in the controls, minimization of the Hamiltonian requires that the controls must be operated at their extremes. Such controllers are called bang-bang controllers. Their design is simplified by requiring only a switching function to determine the schedule of switches between minimum and maximum controls.

### Table II: Fast Swingups

The best trajectories were found with 2 switch windows, corresponding to 2 and 3 second lookaheads, respectively. For the handstand task, the Sarsa/CMAC algorithm was unable to find a trajectory. Both the window size and trajectory steps correspond to 0.2 second control time steps. The Total Evals. column lists the total number of Acrobot states evaluated during each search.

<table>
<thead>
<tr>
<th>Acrobot Task</th>
<th>Algorithm</th>
<th>Window Size</th>
<th>Window Switches</th>
<th>Traj. Steps</th>
<th>Total Evals.</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>Direct Search</td>
<td>10</td>
<td>2</td>
<td>61</td>
<td>54,332</td>
<td>11.7</td>
</tr>
<tr>
<td></td>
<td>Sarsa/CMAC</td>
<td>—</td>
<td>—</td>
<td>82</td>
<td>22,767</td>
<td>2066.8</td>
</tr>
<tr>
<td>Handstand</td>
<td>Direct Search</td>
<td>15</td>
<td>2</td>
<td>82</td>
<td>269,889</td>
<td>73.6</td>
</tr>
<tr>
<td></td>
<td>Sarsa/CMAC</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Table III: Handstand Searches

A series of 2-switch searches with varying window sizes shows the tradeoff between solution effort and quality. With a window size of 11, a good answer is found visiting about 90,000 Acrobot states. Finding a shorter trajectory requires a window size of 15 and visits almost 3 times as many states. na indicates that the search did not find a solution with less than 200 steps and was terminated. Each step corresponds to a 0.2 second control time step.

<table>
<thead>
<tr>
<th>Window Size</th>
<th>Trajectory Steps</th>
<th>Total Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>86</td>
<td>63,709</td>
</tr>
<tr>
<td>11</td>
<td>84</td>
<td>89,994</td>
</tr>
<tr>
<td>12</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>13</td>
<td>94</td>
<td>157,354</td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>199,355</td>
</tr>
<tr>
<td>15</td>
<td>82</td>
<td>269,889</td>
</tr>
<tr>
<td>16</td>
<td>88</td>
<td>259,383</td>
</tr>
<tr>
<td>17</td>
<td>112</td>
<td>434,951</td>
</tr>
</tbody>
</table>

Bang-bang control also requires that the switching function equals zero at only a finite number of points. If it remains zero on any arc, the Hamiltonian no longer defines the optimal control in terms of the state and costate variables. Such arcs are called singular arcs. Whether robot arms with multiple degrees of freedom, such as the Acrobot, have singular arcs remains an open question [17]. The Acrobot, whose forward dynamics are given in Equation 1, is linear with respect to the control input, \( T \). For minimum-time problems, the one-step performance index is 1 because the cost is the count of steps taken. Thus, minimum-time Acrobot controllers are bang-bang, provided that the optimal trajectories have no singular arcs. We assume they do not, based on our observation that the best solutions were almost always bang-bang, even when a third, null action control was allowed as defined in [3].

The greatest limitation of direct searches is their computational cost. However, because the Acrobot has only one control, and because we assume it is bang-bang for minimum-time problems, searching is practical for the Acrobot swingup problem. Our work shows that the best trajectories are found with moderate length windows and a small number of switches. Thus, the algorithm takes only a few minutes to compute trajectories that are near if not optimal. More difficult problems have larger search trees and may be solvable with additional computational resources or more efficient searches. Many control problems have natural heuristics that can be utilized in searches such as energy functions or error gradients.

Note that the algorithm can be run in real time if sufficient computational power is provided to evaluate the trajectories corresponding to the window size in less than one control time step. This approach provides a more integrated computation of path and control, like dynamic programming and reinforcement learning approaches, but avoids the calculations of complete policies, which may provide trajectories for states that are never visited. Direct search is efficient because it does not attempt to visit every state and uses the energy heuristic to bias the search toward promising paths.

Direct searches can increase controller robustness by searching online as the system’s state evolves or as the parameters of the system model change. The controlled system may not follow optimal paths perfectly due to modeling errors such as an external disturbance. Once disturbed, it may no longer be optimal to return to the previously optimal path. Online searches can
generate new plans that are optimal with respect to
the new state. Direct search can increase robustness
when system parameters are unknown or varying. In
conjunction with learning, searching can be used to
replan optimal trajectories as the model is updated by
actual experience. An example of A* search applied
to model learning for the Acrobot is described in a
companion article [18].

VII. CONCLUSION
The N-step, k-switch lookahead search works well
for the Acrobot because of its single actuator and be-
cause a useful heuristic is available. The energy func-
tion provides a rough gradient toward the goal that
can be exploited with a lookahead search. The best
trajectories are found with moderate window sizes cor-
responding to a few seconds and allowing only a small
number of switches.

REFERENCES
[1] Richard M. Murray and John Hauser, "A case study in
approximate linearization: The acrobot example", Tech-
nical Report UCB/ERL/M91/46, University of California,
Berkeley, Electronics Research Laboratory, College of En-
[2] Mark W. Spong, "The swing up control problem for the
[3] Richard S. Sutton, "Generalization in reinforcement learn-
ing: Successful examples using sparse coarse coding", in
[4] Richard S. Sutton, "Learning to predict by the methods of
temporal differences", *Machine Learning*, vol. 3, pp. 9–44,
[5] Gerald DeJong and Mark W. Spong, "Swinging up the
acrobot: An example of intelligent control", in *Proce-
dings of the American Control Conference*, pp. 2158–2162,
trollers for non-integrable systems: The acrobot example", in *Proceedings of the 1990 American Control Conference*,
[7] Daniel E. Davison and Scott A. Bortoff, "Regulation of the
acrobot", in *Proceedings of the 1995 IFAC Symposium on Nonlinear
using pseudolinearization and high-gain feedback", *Auto-
learning", in *Proceedings of the American Control Confer-
ence*, Seattle, WA, June 1996.
[10] Mark W. Spong and Laurent Praly, "Control of underactu-
ated mechanical systems using switching and saturation", in
*Proceedings of the Block Island Workshop on Control Using Logic Based Switching*, Springer-Verlag, 1996.
actuated mechanical systems", in *IFAC World Congress*,
San Francisco, CA, July 1996.
[12] Mathew D. Berkemeier and Ronald S. Fearing, "Control of
a two-link robot to achieve sliding and hopping gaits", in
*IEEE International Conference on Robotics and Automati-
experiments on an underactuated robot with application to
legged locomotion", in *IEEE International Conference on
[16] Cumhur Yalcin Kaya, *Control in the Plane and Time-
Optimal Bang-Bang Control*, PhD thesis, University of
Western Australia, 1995.
[17] Hans P. Geering, Lino Guzzella, Stephan A. R. Hepner, and
Christopher H. Onder, "Time-optimal motions of robots in
assembly tasks", *IEEE Transactions on Automatic Con-
[18] Gary N. Boone, "Efficient reinforcement learning: Model-
based acrobot control", in *IEEE International Conference