System of Funnels Framework for Robust Non-linear Control

Rina Shvartsman

Submitted in total fulfilment of the requirements of the degree of Doctor of Philosophy

October 2015

The Department of Mechanical Engineering
The University of Melbourne

Produced on archival quality paper
Abstract

In this thesis, we propose a general hybrid control framework which can be used to implement a large class of desired behaviours for non-linear systems in a robust manner, even if the reference behaviour is mildly discontinuous. Examples for desired behaviours include: global stabilisation of some set in the state space of the system, allowing particular transitions of the state between neighbourhoods of several locally stable sets, and maintaining the state of the system in a subset of the state space.

Although applications for our general framework can be found in many other contexts, we focus our attention on motion planning problems for dynamic non-linear systems. We demonstrate how our framework can be used to achieve a globally stable and robust approximate implementation of a given motion plan, which may have discontinuities due to computational limitations.

Inspired by previous work, we generalise the notion of a funnel around a nominal trajectory having either a finite or an infinite time duration, using additional generalised notions such as entrance and outlet. A funnel can be seen as a region in the state-time space of the dynamic system to which state-time trajectories initialised from the entrance of the funnel are confined when using a local stabilising control associated with the funnel. This guarantees the arrival of trajectories to a designated outlet region in the case of funnels about nominal trajectories with a finite time duration.

We formulate proper funnel interconnection conditions under which a set of funnels can be considered a system of funnels and used in our framework. We then present a hybrid controller for the system of funnels, which switches between the local stabilising controllers such that when initiated from a particular set of initial conditions, the nominal behaviour of the dynamic system’s state with the hybrid controller is according to the desired motion. In situations when the state of the system is not outside the system of funnels due to improper initialisation, disturbances or measurement noise, we assume usage of a bootstrap control to allow recovery of trajectories to the desired motion.

The proposed hybrid controller is proved to uniformly globally asymptotically stabilise a set in the hybrid state space, according to the constructed system of funnels. The existing global stabilisation control algorithms ‘Throw-and-Catch’ and ‘LQR-Trees’ are
shown to be special cases of our general framework.

In order to demonstrate the framework on some motion planning problems, we first find nominal motions for underactuated systems using a modified bi-directional RRT algorithm. The modifications are made according to some useful heuristics which were found to improve the successfulness of the algorithm and their usefulness is explained, albeit not analytically proved.
Declaration

This is to certify that:

i the thesis comprises only my original work towards the PhD except where indicated in the Preface,

ii due acknowledgement has been made in the text to all other material used,

iii the thesis is fewer than 100 000 words in length, exclusive of tables, maps, bibliographies and appendices.

------------------
Rina Shvartsman
Acknowledgement

The writing of this thesis is a result of a four-year roller coaster of experiences and emotions. In this process, I was hugely supported by many amazing people to whom I owe much more than this short chapter of acknowledgements. It is probably an understatement to say that I lacked the confidence in my ability to get to this point, which often undermined my motivation and performance. It was the incredible people around me who pulled me out of the emotional pits shared by many PhD students. By all means, this work is thanks to them.

My supervisors, Dr Denny Oetomo, A/Prof. Ying Tan and Prof. Dragan Nešić, did absolutely everything needed during my entire candidature - way beyond what I asked for or had in my most ambitious expectations. Their input was extraordinary and inspiring. I feel extremely privileged and lucky to receive such vast support. They were there for me in the hard and the good times, encouraged and motivated me, directed me technically and mentally with a lot of vision and experience, listened to me, helped and supported me, and truly cared for my well-being and happiness. I am enormously grateful to them.

Prof. Andrew (Andy) Teel from UCSB has given me an unbelievable amount of technical guidance. I am so lucky to have been exposed to his utter professional knowledge and skills. Andy challenged the correctness of every single letter I wrote in our correspondence and at the same time guided me very precisely with a well-planned clear path. My thesis definitely began and continued shaping up much due to him. Miraculously, his presentation of complex control material removed so much of the fear I had associated with the field, and I managed to vividly engage in discussions about funnels and hybrid controllers. I am so thankful to his incredible contribution.

I would also like to thank the examiners of this thesis who kindly gave me important feedback.

A great deal of the programming work in this work has been done by Jonathon Harms and Xiaoxiao Cheng. Their hard work, dedication and support was an enormous contribution and also highly motivated me.

Many thanks go to the Department of Mechanical Engineering academics and staff: our
Head of Department, Prof. Doreen Thomas, the departmental administrative staff Jan May and Carol Barrie, Prof. Saman Halgamuge, and the members of my advisory committee: Prof. Chris Manzie and Prof. Peter Lee.

I am also highly grateful to all of my dear friends from the University of Melbourne who shared my journey and made me stronger. Being surrounded by such nice, welcoming, fun, friendly, kind and supportive people is a true gift and what has made the last four years definitely the best of life. Not only the list of friends is long, so are the lists of things they did for me. It is with great sadness that I am only able to mention no more than their names in this acknowledgement: David Watt, Shou-han Zhou, Hadi Sugianto, Michael Lei, Mahdi Pourdanial, Katie Ewing, Jonathan Eden, Justin Fong, Ben Leng, Hurmat Ul Ain, Demian Moreno, Darwin Lau, Behzad Falahati, Sina Porsa, Alireza Mohammadi, Joe Eden, Florence Leong, Hannah Ozturk, Nuray Korkmaz, Vincent Crocher, Azadeh Kian, Hossein Mokhtarzadeh, Fatemeh Malekipour, Soudabeh Mehdikhani, Loai Khalayli, Gijo Sebastian, Florence Leong and Rhys Davis. Thank you for every hug.

I am not able to find sufficient words to thank enough my greatest friend of all, Baruch (Roy) Madar, who was always there for me in all of the hard times and did absolutely everything for my success. I am so grateful for his love, support and encouragement. My family in Melbourne and in heart, Polina Tserniak and Izzy Gurevich have supported me greatly during these years. I thank you for all the light within you that you shared with me and for always welcoming me so warmly in your home.

Finally, I would like to express my loving thanks to my closest for all of their endless love, dedication and support that made my life extremely fulfilling: my husband Jonás Fuks and his family, my sister Maly and her partner Damian Jakob, my brothers Yuval and Lidor, and my parents, Inna and Haim.
2 Global Stabilisation via System-of-Funnels Framework

2.1 Hybrid Systems - Background Terms and Definitions

2.2 Definition 1: A Funnel

2.3 Definition 2: A System of Funnels

2.3.1 Properties of a System of Funnels

2.4 The Hybrid Controller Design

2.4.1 Hybrid State Variable

2.4.2 Flow Set

2.4.3 Flow Map

2.4.4 Jump Set

2.4.5 Jump Map

2.4.6 The Hybrid Closed-Loop System

2.5 Nominal Behaviour of the Hybrid Controller

2.5.1 Theorem 1

2.5.2 Proof

2.6 Incorporating Bootstrap Control

2.6.1 Recovery Policy

Bootstrap Control Assumption:

2.6.2 Theorem 2

2.6.3 Proof

2.7 The Limiting Behaviour of Solutions to the Hybrid System

2.7.1 Theorem 3

2.7.2 Proof

Lemma 1

Lemma 2

Lemma 3

Lemma 4

Lemma 5

Lemma 6

Proof of Lemma 1

Proof of Lemma 2
5 Illustrating Examples

5.1 Example 1: Two Funnels Example ......................................... 169
5.2 Example 2: Four Funnels Example ......................................... 171
5.3 Example 3: Wing Swing Motion - Task Description .................... 177
  5.3.1 Simple Pendulum Model ................................................. 178
  5.3.2 Obtaining Nominal Trajectories ...................................... 179
  5.3.3 Hybrid System Design .................................................. 181
    5.3.3.1 Funnels 2 and 3 .................................................. 183
    5.3.3.2 Funnels 1 and 4 .................................................. 188
    5.3.3.3 Bootstrap Controller ............................................. 190
    5.3.3.4 The Hybrid Controller ........................................... 192
  5.3.4 Simulated Motion ...................................................... 193
5.4 Conclusions ............................................................... 195

6 Conclusions

6.1 Summary of Contributions .................................................. 198
  6.1.1 System-of-Funnels Framework for Global Stabilisation of Nominal
        Motions .............................................................. 198
  6.1.2 A Bi-Directional RRT Algorithm for Constructing Nominal Tra-
        jectories ............................................................ 199
6.2 Suggested Future Work ..................................................... 200
  6.2.1 Hybrid Frameworks for Global Stabilisation ......................... 200
  6.2.2 Nominal Trajectory Planning using Randomised-Sampling Algo-
        rithms .............................................................. 201

Bibliography ........................................................................... 203
List of Figures

1.1 Results of Simulation using Algorithm 1.1.1 [86] ............................ 25
1.2 Results of Simulation using Algorithm 1.1.2 [86] [84] ................. 27
1.3 Results of Simulation using Algorithm 1.1.3 [86] [84] .................... 28
1.4 Results of Simulation using Algorithm 1.1.4 ............................. 30
1.5 Results of Simulation using RG-RRT ........................................ 32
1.6 A motivating illustration ....................................................... 66

2.1 Example of a graph $G$ of a system of funnels. .......................... 99
2.2 The double pendulum model ................................................. 101
2.3 The 4 equilibria and the constructed tree structure ..................... 102
2.4 A qualitative illustration of the behaviour of the system under the appli- ............................ 103
2.5 The different sets defined. .................................................... 114
2.6 LQR-Tree $\mathcal{T}(\mathbb{K}, \Upsilon)$ construction illustrated example ........ 115

3.1 A time-invariant funnel: (On the right:) time-invariant funnels $\tilde{\mathcal{F}}$ (top) and $\mathcal{F}$ (bottom) shown either in cyan or transparent and defined for an infinite time domain $[0, \infty)$, and their time-invariant entrance $\tilde{\mathcal{E}}$ (top) and $\mathcal{E}$ (bottom) shown in magenta and defined for some finite time-domain. The funnels are positioned about an equilibrium state $z_e$ (shown in red) for a two-dimensional system. (On the left:) a projection on the $\tilde{z}_1 - \tilde{z}_2$ plane (top) where $\tilde{F}, \tilde{E}$ and $\tilde{R}_e$ are defined, and a projection on the $z_1 - z_2$ plane (top) where $F, E$ and $R_e$ are defined. ............................ 120

3.2 A funnel $\mathcal{F}$ (cyan or transparent) defined about a nominal trajectory, $\varphi$: The shadowed magenta area (with its blue or transparent extrusion) shows the entrance $\mathcal{E}$ of the funnel and the green area represents the outlet, $\mathcal{O}$. Note that for all $\tau \in I$, $\rho(\tau) < \alpha(\tau)$. ............................ 123

3.3 A bi-directional funnel composed from two sub-funnels about disconnected trajectories: The forward sub-funnel, $\mathcal{F}_{\text{start}}$ and its entrance $\mathcal{E}_{\text{start}}$, are shown on the left. The backward sub-funnel, $\mathcal{F}_{\text{target}}$ and its entrance $\mathcal{E}_{\text{target}}$, are shown on the right. It can be seen that the outlet of the forward funnel, $\mathcal{O}_{\text{start}}$, is contained in the the set $\{T_{\text{start}}\} \times E_{\text{target}}(0)$, which means that the two sub-funnels are properly connected. ............................ 127
LIST OF FIGURES

3.4 A funnel $\mathcal{F}$ ................................. 129
3.5 A funnel about a limit cycle, $\mathcal{F}$, and its entrance, $\mathcal{E}$, which is inside the limit cycle’s region of attraction. ................................. 133
3.6 An infinite-time-domain funnel $\mathcal{F}$ about an invariant set and its finite-time-domain entrance $\mathcal{E}$ shown as projections on a two-dimensional state-space. The broken line represents the region of finite-time attraction about the invariant set, $\mathcal{R}_\Omega$. A typical trajectory which starts inside the entrance set and is pulled by the controller into the invariant set is also shown. ................................. 135

4.1 Two nodes, A and B, in a tree and their reachable states. It is clear in this example that it is easier to develop node A to reach the target than node B. The measure of Euclidean distance, however, does not indicate the level of difficulty to reach the target from each node. ................................. 148
4.2 A comparison between the resulting trees for the torque-limited simple pendulum with $\text{NumOfStages} = 3$. Plots (a), and its magnified view (c), refer to the modified distance function Eq. 4.1, whereas plots (b), and its magnified view (d), refer to the conventional Euclidean distance function. The rectangles represent the three nested search areas (SA) used for the three stages and the circles represent the different $B_i$. The start states and target states (which is $2\pi$-periodic) are marked with bold square and circles, respectively. The best path found is shown in solid line. ................................. 156
(a) (a) .................................................. 156
(b) (e) .................................................. 156
(c) (b) .................................................. 156
(d) (f) .................................................. 156
4.3 The magnified views of Fig. 4.2 in the vicinity of the target. The accurate distribution of nodes in the various $B_i$ for the two cost functions at each stage of the search is presented in Table 4.1. ................................. 157
(a) (a) .................................................. 157
(b) (c) .................................................. 157
(c) (b) .................................................. 157
(d) (d) .................................................. 157
4.4 Illustration of how mates are updated when a new node is created. ................................. 167
5.1 Example 1: Scenario 1 and 2 ................................. 172
LIST OF FIGURES

5.2 Example 1: Scenario 3 and 4 ................................................. 173
5.3 Top View ................................................................. 175
5.4 3D View ............................................................... 176
5.5 Top view and 3D view of the solution to Example 2 ..................... 177
5.6 Four configurations for the design of the system of funnels ............. 178
5.7 All four trajectories vs. time ............................................. 182
5.8 All four trajectories in state space ...................................... 183
5.9 Torques used to generate the bi-directional trajectories ................. 184
   (a) \( u_1 \) ................................................................. 184
   (b) \( u_2 \) ................................................................. 184
   (c) \( u_3 \) ................................................................. 184
   (d) \( u_4 \) ................................................................. 184
5.10 Bi-directional tree for finding the trajectory \( \varphi_1 \) (a) and the magnified view of the trajectory .................. 185
   (a) ................................................................. 185
   (b) ................................................................. 185
5.11 A graph of 4 funnels for wing swing motion of a pendulum .......... 186
5.12 Initial attempt to construct Funnel 2 and Funnel 3 ...................... 187
   (a) ................................................................. 187
   (b) ................................................................. 187
5.13 (a) Initial attempts of Funnels 2 and 3 plotted together (b) Funnels 2 and 3 - only start and end sections .............. 188
   (a) ................................................................. 188
   (b) ................................................................. 188
5.14 Final construction of Funnel 2 (a) and Funnel 3 (b) ..................... 189
   (a) ................................................................. 189
   (b) ................................................................. 189
5.15 Final construction of Funnels 2 and 3 - only start and end sections (and entrances) ........................................ 190
5.16 Final construction of Funnel 1 (a) and Funnel 4 (b) ..................... 191
   (a) ................................................................. 191
   (b) ................................................................. 191
5.17 Final construction of all four Funnels - only start and end sections (and entrances) ........................................ 192
5.18 A trajectory from the initial condition (30\(^\circ\), 0) - state-space plot . 194
5.19 A trajectory from the initial condition (180°, 0) - state-space plot . . . . . 195
5.20 A trajectory from the initial condition (−90°, 5) - state-space plot . . . . 196
5.21 A trajectory from the initial condition (−90°, 5) - θ vs. time . . . . . . . 196
5.22 A trajectory from the initial condition (−90°, 5) - ˙θ vs. time . . . . . . . 197
# List of Tables

4.1 Algorithm comparison for the torque-limited pendulum. . . . . . . . . . . . . . 155
4.2 Algorithm comparison for the Acrobot. . . . . . . . . . . . . . . . . . . . 155
4.3 Distances . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 163

5.1 Model Parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 179
5.2 Settings for the Simple Pendulum . . . . . . . . . . . . . . . . . . . . . 180
5.3 Settings for four trajectories . . . . . . . . . . . . . . . . . . . . . . . . 181
5.4 Results from the Simple Pendulum Pathfinder . . . . . . . . . . . . . . 181
## List of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Algorithm Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>Simple Random Sample Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Basic RRT</td>
<td>26</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Basic RRT With Constraints</td>
<td>27</td>
</tr>
<tr>
<td>1.1.4</td>
<td>Basic RRT With Dynamic Constraints</td>
<td>29</td>
</tr>
<tr>
<td>1.1.5</td>
<td>RG-RRT</td>
<td>31</td>
</tr>
<tr>
<td>1.1.6</td>
<td>RRT_CONNECT_PLANNER($q_{init}$, $q_{goal}$)</td>
<td>33</td>
</tr>
<tr>
<td>1.1.7</td>
<td>EXTEND($\mathcal{T}$, $q$)</td>
<td>33</td>
</tr>
<tr>
<td>1.1.8</td>
<td>CONNECT($\mathcal{T}$, $q$)</td>
<td>33</td>
</tr>
<tr>
<td>1.1.9</td>
<td>LQR_Based_Proximity</td>
<td>35</td>
</tr>
<tr>
<td>1.1.10</td>
<td>COST_TO_GO</td>
<td>35</td>
</tr>
<tr>
<td>1.1.11</td>
<td>RRT Body Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>1.1.12</td>
<td>Extend (unique to RRT*)</td>
<td>38</td>
</tr>
<tr>
<td>1.1.13</td>
<td>LQR - RRT*((V,E),N)</td>
<td>40</td>
</tr>
<tr>
<td>1.1.14</td>
<td>ChooseParent($X_{near}$, $x_{new}$)</td>
<td>40</td>
</tr>
<tr>
<td>1.1.15</td>
<td>Rewire((V,E), $X_{near}$, $x_{new}$)</td>
<td>40</td>
</tr>
<tr>
<td>1.1.16</td>
<td>PLANNER</td>
<td>43</td>
</tr>
<tr>
<td>1.1.17</td>
<td>SAMPLE(k)</td>
<td>43</td>
</tr>
<tr>
<td>1.1.18</td>
<td>LOCAL_BIAS</td>
<td>43</td>
</tr>
<tr>
<td>1.1.19</td>
<td>SAMPLE(k)</td>
<td>45</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Directionality-Guided-Nested-RG-RRT(Start, Target)</td>
<td>151</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Directionality-Guided-Nested-Bi-RG-RRT</td>
<td>159</td>
</tr>
<tr>
<td>4.6.2</td>
<td>AlgorithmInitialisationAndSettings()</td>
<td>160</td>
</tr>
<tr>
<td>4.6.3</td>
<td>ChooseNodeForExpansionRG-RRT</td>
<td>161</td>
</tr>
<tr>
<td>4.6.4</td>
<td>CreateNewNode</td>
<td>165</td>
</tr>
<tr>
<td>4.6.5</td>
<td>UpdateBestCouple</td>
<td>166</td>
</tr>
<tr>
<td>4.6.6</td>
<td>PrepareForNewStage</td>
<td>166</td>
</tr>
</tbody>
</table>
Nomenclature

\( \mathbb{R}^n \) The n-dimensional Euclidean space
\( \mathbb{R}_{\geq 0} \) The set of non-negative real numbers
\( \mathbb{N} \) The set of natural numbers (non-negative integers)
\( \bar{\mathbb{N}} \) The extended set of natural numbers: \( \bar{\mathbb{N}} := \mathbb{N} \cup \{\infty\} \)
\( z \) The state of a dynamic system, e.g., \( \dot{z} = f(z) \)
\( u \) The input of a control system, e.g., \( \dot{z} = f(z, u) \)
\( f : D \to I \) A single-valued mapping from the domain set \( D \) to the image set \( I \)
\( G : D \Rightarrow I \) A set-valued mapping from the domain set \( D \) to the image set \( I \)
\( S_1 \times S_2 \) A Cartesian product of two sets
\( \mathcal{F} \) A funnel
\( \mathcal{E} \) An entrance
\( \mathcal{O} \) An outlet
\( \text{dom}(f) \) The domain of the function \( f \)
\( \text{gph}(G) \) The graph of the mapping \( G \)
\( \text{cl}(S) \) The closure of the set \( S \)
\( \text{int}(S) \) The interior of the set \( S \)
\( \emptyset \) The empty set
\( \Sigma \) A set of funnels
\( \mathcal{K} \) A set of funnel indices
Introduction

In numerous applications, it is often necessary to design a particular type of behaviour for a dynamic system. Examples can be found in controlling an aircraft to perform some elaborate manoeuvres; manipulation of a bipedal robot to perform a gait of some desired average speed; or, navigation of a car from a start to a target position and orientation. Such applications normally give rise to the important problem of global or local stabilisation of desired behaviours for non-linear systems. This problem can be cast into two sub-problems:

- nominal trajectory planning (also known as motion planning);
- global or local stabilisation of the nominal trajectory (also known as tracking).

In this thesis, we propose a robust hybrid control framework which achieves a global stabilisation of a desired behaviour, along with a trajectory planning strategy which can be used to obtain the nominal trajectories which reflect the desired behaviour of the system. Before we present the background literature on motion planning and stabilisation of nominal trajectories, we first define these problems more precisely.

Consider, for example, a non-linear control system of the general form:

\[
\begin{align*}
\dot{z} &= f(z, u), \\
y &= h(z),
\end{align*}
\]

where \( f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \) is a continuous function describing the system’s dynamics with respect to time, \( h : \mathbb{R}^n \rightarrow \mathbb{R}^p \) is a continuous function describing the system’s output,
$z \in \mathbb{R}^m$ is the state of the system, $u \in \mathbb{R}^m$ is its input and $y \in \mathbb{R}^p$ is its output.

The nominal trajectory planning problem is concerned with finding an input signal over some time interval such that the output of the system will have some desired characteristics. For instance, it is often required that the output of the system starts in a defined value $y_{\text{start}}$ and terminates in a defined value $y_{\text{end}}$ within some allowable time $T_{\text{max}} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, while avoiding undesirable values in some set $Y_{\text{avoid}} \subset \mathbb{R}^p$. The nominal trajectory planning problem can be typically defined as follows (however, other definitions can be formulated as well):

\[
\text{Find } T \in \mathbb{R}_{\geq 0} \cup \{\infty\} \text{ and } u^0 : [0, T] \to \mathbb{R}^m \text{ such that:}
\]

1. $T \leq T_{\text{max}}$,
2. $y(0) = y_{\text{start}}$,
3. $y(T) = y_{\text{end}}$, and
4. $y(t) \notin Y_{\text{avoid}}$ for all $t \in [0, T]$.

Normally, all of these problems are initially addressed via the implementation of some sort of motion planning algorithm or trajectory design approach. The output of the motion planner is an input signal with a finite- or an infinite-time duration which should be supplied to the system in order to induce the planned trajectory.

Assume that some nominal input signal is generated using some trajectory planner. We now have an open-loop control law, a state trajectory and a resulting desired output trajectory, all defined on the same time interval:

\[
u^0(\cdot), \ z^0(\cdot), \ y^0(\cdot),\]

satisfying the system equations:

\[
\dot{z}^0 = f(z^0, u^0), \quad (1.3) \\
y^0 = h(z^0). \quad (1.4)
\]

The next step is to stabilize the obtained nominal trajectory or, in other words, achieve tracking of the nominal trajectory. To this end, an error system can be defined using shifted coordinates:
\[ \tilde{u} := u - u^0(t), \quad \tilde{z} := z - z^0(t), \quad \tilde{y} := y - y^0(t), \]

Substituting \( u, z \) and \( y \) in terms of the shifted coordinates to the original system in Eqs. (1.1)-(1.2) results in a system that is later referred to as error dynamics:

\[
\begin{align*}
\dot{\tilde{z}} &= f(z^0(t) + \tilde{z}, u^0(t) + \tilde{u}) - \dot{z}^0(t), \\
\tilde{y} &= h(z^0(t) + \tilde{y}) - y^0(t).
\end{align*}
\] (1.5) (1.6)

Stabilization of the nominal trajectory (or tracking) consists of finding a controller (e.g. static, dynamic or hybrid) for which the system (1.5-1.6) is stable (in the sense of Lyapunov) and in particular is such that for some set of initial conditions, the solutions of the closed-loop system satisfy

\[ \lim_{t \to \infty} |y(t) - y^0(t)| = \lim_{t \to \infty} |\tilde{y}(t)| = 0. \]

In the case of local tracking, we require this to hold only for initial conditions that are close to the nominal trajectory. On the other hand, for global tracking this should hold for all solutions of the closed-loop system.

Various tracking techniques allow, under certain assumptions on the system and the trajectory, the computation of an appropriate signal \( \tilde{u} \) which guarantees that \( \tilde{y} \) is driven to zero. This can be achieved, for instance, using a continuous-time dynamic controller of the following form:

\[
\begin{align*}
\dot{\xi} &= f_c(t, \xi, \tilde{y}) \\
\tilde{u} &= u_c(t, \xi, \tilde{y}),
\end{align*}
\] (1.7) (1.8)

where \( \xi \) is the state of the dynamic controller and \( f_c \) and \( u_c \) are the appropriately defined mappings. Finding a controller of the form (1.7), (1.8) that achieves global tracking is hard in general. On the other hand, a local tracking control for the trajectory is often more readily achievable. Consider the local linearisation\(^1\) of the error dynamics (1.5)-(1.6) about the nominal trajectory:

\(^1\)The local linearisation is valid when \( \tilde{u}, \tilde{z} \) and \( \tilde{y} \) are small.
\[
\dot{z} \approx f(z^0, u^0) - z^0 + \frac{\partial f}{\partial z} \bigg|_{z = z^0, u = u^0} (z - z^0) + \frac{\partial f}{\partial u} \bigg|_{z = z^0, u = u^0} (u - u^0) \quad (1.9)
\]
\[
\dot{\tilde{y}} \approx h(z^0, u^0) - \tilde{y}^0 + \frac{\partial h}{\partial z} \bigg|_{z = z^0, u = u^0} (z - z^0). \quad (1.10)
\]

Cancelling out the terms using Eqs. (1.3)-(1.4) gives:

\[
\dot{\tilde{z}} = A(t)\tilde{z} + B(t)\tilde{u}, \quad (1.11)
\]
\[
\tilde{y} = C(t)\tilde{z}, \quad (1.12)
\]

where

\[
A(t) := \frac{\partial f}{\partial z} \bigg|_{z = z^0, u = u^0}, \quad (1.13)
\]
\[
B(t) := \frac{\partial f}{\partial u} \bigg|_{z = z^0, u = u^0}, \quad (1.14)
\]
\[
C(t) := \frac{\partial h}{\partial z} \bigg|_{z = z^0, u = u^0}. \quad (1.15)
\]

Then, stabilization via linear techniques for the linearised system (1.11), (1.12) using an observer-controller pair (e.g. via LQR control techniques) can often\(^2\) be used to obtain a linear controller of the form (1.7), (1.8) that achieves local tracking.

As already stated, finding controllers of the form (1.7), (1.8) is in general difficult whereas this problem is easier for local tracking. It should be noted that in some cases, it is simpler to find a set of locally stabilizing controllers and compose a hybrid controller which switches between them to achieve global tracking, rather than to find a continuous-

\(^2\)This is not always achievable. See, for instance, [35].
time globally stabilising control. Such controllers take the form of hybrid systems, such as

\[
\begin{align*}
\dot{\xi} &\in F_c(\xi, \tilde{y}) & (\tilde{y}, \xi) &\in C_u \\
\xi^+ &\in G_c(\xi, \tilde{y}) & (\tilde{y}, \xi) &\in D_u \\
\tilde{u} &\in U_c(\tilde{y}, \xi), 
\end{align*}
\]

where \(F_c\) and \(G_c\) are flow and jump set-valued maps, and \(C_u\) and \(D_u\) are the flow and jump sets, and \(U_c\) is the control map.

This thesis will thoroughly present how such a hybrid controller can be constructed and used to globally stabilize a desired motion; such controller switches between a set of local controllers that stabilize various nominal trajectories or pieces of a given nominal trajectory. We also provide an approach for generating nominal trajectories, also called motion plans. In the sequel, we provide an overview of the literature that is most related to this thesis; in particular, we concentrate on the extensive literature on motion planning and then discuss several techniques on global stabilization of nominal trajectories via hybrid control that were recently proposed in the literature.

1.1 Background

1.1.1 Generation of Nominal Motion Plans

The basic problem of motion planning is concerned with computing open-loop control trajectories for dynamic systems such that some desired behaviour results for the states of the system. Motion planning is used in numerous applications, such as in walking robots, navigation, robotic manipulation, and many more. There are plenty of different motion planning algorithms, most of which are variations of some basic ones. Different algorithms are suitable for different classes of problems. In this section, we present several important motion planning algorithms which appeared in the literature in the last two decades. Some of these algorithms are presented for the sake of comparison, while others will be directly used later as a basis for constructing our own motion planning algorithm. In Chapter 3, we present the developments for our own algorithm, while in this section, we provide all the necessary background to it. Although many combinations and subclasses of motion planning strategies exist, we divide the existing strategies into three main groups:
1.1. BACKGROUND

- sampling-based motion planning algorithms;
- inverse dynamics;
- forward dynamics.

We will now discuss each of these groups and later add some more miscellaneous strategies. One should notice that a common problem of imperfections such as discontinuities of some sort of infeasibility in the resulting motion plans is apparent in all of these techniques and has to be dealt with using feedback control. A special focus will be given to motion planning for systems with non-holonomic constraints and under-actuated systems (which characterize most of the bipedal walking systems performing a dynamic gait) for which the above problem of imperfections is accentuated.

[31] provides an extensive coverage of the topic of motion planning. Motion planning algorithms can be divided into two main families: deterministic algorithms, where either the state space of the system or the input domain is sampled according to a predefined algorithm or using a grid (such as in [37] and references therein, where trajectories were found based on trying several inputs from each node over a time duration which is reduced as the algorithm progresses), and random-sampling algorithms, which usually avoid using a grid or any pre-defined expansion technique and instead, sample random states in the state space at each iteration and develop a graph which is expanded towards the currently generated sample.

1.1.1.1 Sampling Based Motion Planning

In this section we present a commonly used technique for motion planning (often called incremental motion planning, or sampling-based motion planning), whose characteristic is that it does not immediately plan trajectories for the entire motion, but instead, incrementally generates a graph of many possible trajectories in the state space (or configuration space) of the system and then picks a path along this graph to be the chosen motion plan. The nodes of the graph are states of the system, and they are connected with (usually directed) edges which are short segments of trajectories connecting one node to another. The motion planning algorithms are not ‘black-box’ solutions and therefore they have two main advantages over the other two approaches of forward and inverse dynamics which are:
1. All of the constraints on the required nature of the motion can be easily incorporated into the specific implementation of the algorithm to ensure that all of the developed trajectories comply with them.

2. The maximal required running time for roughly achieving a desired result, if it exists, can be estimated, at least after one run of the program. Moreover, the programs allow relatively easy tuning of parameters to match the exact motion planning problem at hand, that is, the programs are easily adaptable to user-specific problems.

For these reasons sampling based motion planning algorithms was selected in our work in order produce nominal trajectories.

A dominant advantage of the random-sampling methods over their deterministic counterparts is that random-sampling methods are much more suitable for high-dimensional systems, by avoiding having to exhaustively traverse the high-dimensional state-space. After unsatisfactory results were obtained using deterministic approaches for a fairly low-dimensional under-actuated system, we have decided to focus on the random-sampling approaches, which proved to be generally more successful in finding good trajectories in practice. Most of the deterministic methods have proven completeness guarantees; that is, if a feasible solution exists, it will be surely found after infinitely many iterations). The random-sampling approaches have a so-called probabilistic completeness guarantee; that is, if a feasible solution exists, it will be found with probability one (almost surely) after infinitely many iterations. In most practical cases, where the algorithms can be run only for a finite number of iterations, the probabilistic completeness guarantee serve to provide the same level of assurance as that of the deterministic approaches. The sampling based randomized algorithms however, manage to allow better search in practice for high-dimensional systems by avoiding the need to discretize the state and/or input space or to “check all possible options” exhaustively.

The randomized motion planning can be itself categorized into two distinct subgroups according to the type of graph which is constructed during the running of the algorithm: the Probabilistic Road Maps (PRM) approach creates a general connected graph in the configuration space, where as the Rapidly-Exploring Random Tree (RRT) algorithm creates a directed tree graph. The former algorithm is suitable for multi-query problems, while the latter is designed for single-query tasks. There are important modifications
1.1. BACKGROUND

to RRT which allow its efficient use for planning problems which have under-actuation or other non-holonomic constraints. Therefore, we chose to study it in relative depth in this thesis. In the rest of this subsection, we shortly discuss the PRM algorithm, the RRT algorithm and some of its important extensions.

Random Walker

The simplest of the random-sampling algorithms is a Random Walker. We simply choose a random node from the developing tree, and then a random action. We then apply this action to compute a new state and add this state to the tree. We repeat this process until we reach within a certain margin of the goal.

<table>
<thead>
<tr>
<th>Algorithm 1.1.1: Simple Random Sample Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( T \leftarrow x_0 ) ;</td>
</tr>
<tr>
<td>2 for ( n = 1 ) to ( NumberOfIterations ) do</td>
</tr>
<tr>
<td>3 Select random node, ( x_{\text{rand}} ), from the tree, ( T ) ;</td>
</tr>
<tr>
<td>4 Select random action, ( u_{\text{rand}} ), from a distribution over feasible actions ;</td>
</tr>
<tr>
<td>5 Compute the dynamics: ( x_{\text{new}} = f(x_{\text{rand}}, u_{\text{rand}}) ) ;</td>
</tr>
<tr>
<td>6 ( T \leftarrow x_{\text{new}} ) ;</td>
</tr>
<tr>
<td>7 if</td>
</tr>
<tr>
<td>8 Terminate. Solution found.</td>
</tr>
<tr>
<td>9 return ( Path )</td>
</tr>
</tbody>
</table>

While this method is probabilistically complete, without any heuristics to guide the selection of nodes scheduled for expansion, it is extremely inefficient. The results of this simulation running for 1000 iterations is shown in Figure 1.1 which is adopted from [86]. From this, it is clear that it will take a long time to reach the goal.


1.1. BACKGROUND

Probabilistic Road Maps

The method of Probabilistic Road-maps (PRM) [28, 2] is a multi-query path-planning approach which works in two separated phases: ‘learning phase’ and ‘query phase’. In the learning phase (which usually can be done once offline), a road-map of feasible trajectories in the static search space of the system is constructed (in a form of a graph). In the query phase (which can be done on-line in many cases), given a task of start and target configurations, a valid path from the given start state to the given target state in the configuration space is found using the following three stages: connecting the start state to the road-map, connecting the target state to the road-map, and finding a path along the road-map which connects the start and the target states. This method is useful for multi-query tasks, however, is limited to apply only to holonomic robots (and hence, cannot be applied for systems with underactuation or non-holonomic constraints such as car-like robots). See [28] for algorithm and examples.

Rapidly Exploring Random Trees - RRT (basic algorithm)

From the example given in Section 1.1.1.1 it is clear that we need an algorithm that expands into the unexplored space more quickly. The Rapidly Exploring Random Tree
(RRT, see [33, 32, 30, 34, 36]) achieves this with a modification to Algorithm 1.1.1. Instead of the sample being a random node on the tree, we sample a random node within the bounds that we are searching, and expand the tree towards that random sample. This is done by

1. A random sample is found, occasionally this is chosen to be the target itself. The distance between all nodes in the tree and the sample is calculated to find the nearest node, which is then chosen to be expanded.

2. A new node is created that extends the tree towards that sample up to a maximum length. If the point is further away from the chosen node than the maximum length, then the new node is chosen to be along the line between the two points and a distance of the maximum length away from the closest node. Otherwise it is the random sample itself.

This algorithm dramatically improves the search for a path between two points. This is shown in Figure 1.2. We can see that a feasible trajectory is found with a much smaller number of random nodes.

**Algorithm 1.1.2: Basic RRT**

```
1  \mathcal{T} \leftarrow x_0 ;
2  \textbf{for} n = 1 \textbf{to} NumberOfIterations \textbf{do}
3      x_{\text{rand}} \leftarrow \text{RandomState}(x_{\text{target}});
4      x_{\text{near}} \leftarrow \text{NearestNeighbour}(x_{\text{rand}}, \mathcal{T});
5      x_{\text{new}} \leftarrow \text{NewState}(x_{\text{near}}, x_{\text{rand}});
6      \mathcal{T} \leftarrow x_{\text{new}} ;
7      \textbf{if } ||x_{\text{new}} - x_{\text{target}}|| < \varepsilon \textbf{ then}
8          \text{Break. Solution Found.}
9  \textbf{return } Path
```
This algorithm also successfully deals with constraints in the system. By only allowing new nodes to be in the valid area (ie removing all nodes that violate constraints) the tree will successfully find a feasible trajectory while not violating any constraints. This is shown in Figure 1.3.

**Algorithm 1.1.3:** Basic RRT With Constraints

1. $\mathcal{T} \leftarrow x_0$ ;
2. for $n = 1$ to $NumberOfIterations$ do
   3. $x_{\text{rand}} \leftarrow \text{RandomState}(x_{\text{target}})$ ;
   4. $x_{\text{near}} \leftarrow \text{NearestNeighbour}(x_{\text{rand}}, \mathcal{T})$ ;
   5. $x_{\text{new}} \leftarrow \text{NewState}(x_{\text{near}}, x_{\text{rand}})$ ;
   6. if $x_{\text{new}} \neq \text{NULL}$ then
      7. $\mathcal{T} \leftarrow x_{\text{new}}$ ;
      8. if $||x_{\text{new}} - x_{\text{target}}|| < \varepsilon$ then
10. return Path
So far we have not considered the dynamics of the system. Namely the new node is chosen without any consideration of the physical characteristics of the system. We can incorporate this into the RRT algorithm by modifying point 2 as below:

1. A random sample is found, occasionally this is chosen to be the target itself. The distance between all nodes in the tree and the sample is calculated to find the nearest node, which is then chosen to be expanded.

2. An appropriate command torque is then chosen to be applied for a short time duration from a discrete set of torques. The torque command should be chosen in such a way that the tree is expanded towards the random sample. A forward integration of the equations of motion is then performed and the resulting state is added as a node in the tree (if all constraints are satisfied).
Algorithm 1.1.4: Basic RRT With Dynamic Constraints

1. Initialize $\mathcal{T} \leftarrow x_0$
2. for $n = 1$ to NumberOfIterations do
3.     $x_{\text{rand}} \leftarrow \text{RandomState}(x_{\text{target}})$
4.     $x_{\text{near}} \leftarrow \text{NearestNeighbour}(x_{\text{rand}}, \mathcal{T})$
5.     $u \leftarrow \text{SelectInput}(x_{\text{rand}}, x_{\text{near}})$
6.     $x_{\text{new}} \leftarrow \text{NewState}(x_{\text{near}}, u, \Delta t)$
7.     if $x_{\text{new}} \neq \text{NULL}$ then
8.         $\mathcal{T} \leftarrow x_{\text{new}}$
9.     if $||x_{\text{new}} - x_{\text{target}}|| < \varepsilon$ then
11. return Path

We can investigate the performance of this algorithm by trying to solve the swing up problem for a simple torque limited pendulum. The equation of motion is:

$$I \ddot{\theta} + b \dot{\theta} + mgl \sin(\theta) = \tau$$

with parameters $m = 1.0 \text{ kg}$, $l = 0.5 \text{ m}$, $b = 0.1$, $I = ml^2 \text{ kgm}^2$, $g = 9.8 \text{ m/s}^2$.

In this simulation the distance function is simply the Euclidean distance (wrapped to be in the range $[-\pi/2, 3\pi/2]$) and the torque is selected from a discrete set of 20 in the range $[-5, 5]$ Nm. The desired torque is chosen by simulating the system using all possible torques and choosing the one the results in the closest distance to the random sample. The timestep is 0.1s. Figure 1.4 shows the results for one particular simulation of this system. This simulation was derived from code provided in [87].
1.1. BACKGROUND

We can see that a feasible trajectory has been found using 760 nodes. This result is fairly average of the simulations. Sometimes it would be faster than this but the most common occurrence was that the tree would fail to find a solution close enough to the goal with less than 1000 nodes.

There are two ways in which this algorithm can be modified in an attempt to get better performance:

1. Modify the sampling distribution

2. Modify the distance metric used to determine ‘closeness’.

**RG-RRT**

The Reachability Guided RRT (RG-RRT, see [75, 76]) method builds on the RRT method described in Algorithm 1.1.4 by modifying the sampling distribution (line 3). When we create a node, we can calculate the set of states towards which it can be expanded in one step. These are called the reachable states. All of the reachable states are part of the reachable set $\mathcal{R}$.

RG-RRT works by rejecting samples that are closer to the tree itself than to the reachable states. This encourages the tree to reach out more rapidly and avoid a close tangle of
nodes. The logic behind this is that when a node is close to a sample it may not be the case that expanding the node will grow the tree towards the sample while there are constraints on the system.

The function $\text{NearestNeighbourRGRRT}(x_{\text{rand}}, T)$ now also returns a boolean value $\text{IsTreeClosest}$ that indicates whether or not the sample state is closer to $T$ than to $R$. If this is the case, then we need to generate a new sample and try again, until we get a sample that is valid.

A simulation was performed using the same model as the simple RRT method and the results of one run are shown in Figure 1.5. The Reachability Guided method was observed to find a solution more reliably and faster than the simple RRT. Comparing this to Figure 1.4 we can see that the tree is spread out more and has avoided growing the tree when it will form nodes that do not explore into the reachable area.
RRT Connect

RRT Connect is the name of the first bi-directional form of the RRT algorithm which originally appeared in [29]. It was originally designed for increasing the efficiency of a single-query RRT search for systems with no kinodynamic constraints.

In the RRT-connect algorithm there are mainly two stages: the first stage is applying the usual RRT algorithm for one of the trees (such that a new node is obtained) and the second stage is trying to connect the other tree to the new node of the first tree. The connection is attempted by the CONNECT function, however, it can be replaced by the EXTEND function to make an incremental stepping towards the node instead of the greedy operation of the CONNECT function. After these two stages are completed, the roles of the trees are switched and another iteration is taken.
The algorithm is presented below:

**Algorithm 1.1.6: RRT_CONNECT_PLANNER**(\(q_{\text{init}}, q_{\text{goal}}\))

1. \(T_a\).init(\(q_{\text{init}}\));
2. \(T_b\).init(\(q_{\text{goal}}\));
3. for \(k = 1\) to \(K\) do
   4. \(q_{\text{rand}} \leftarrow \text{RANDOM\_CONFIG}()\);
   5. if not (\(\text{EXTEND}(T_a, q_{\text{rand}}) = \text{Trapped}\)) then
      6. if (\(\text{CONNECT}(T_b, q_{\text{new}}) = \text{Reached}\)) then
         7. return \(\text{PATH}(T_a, T_b)\);
   8. SWAP(\(T_a, T_b\));
9. return Failure

**Algorithm 1.1.7: EXTEND**(\(T, q\))

1. \(q_{\text{near}} \leftarrow \text{NEAREST\_NEIGHBOR}(q, T)\);
2. if \(\text{NEW\_CONFIG}(q, q_{\text{near}}, q_{\text{new}})\) then
   3. \(T\).add_vertex(\(q_{\text{new}}\));
   4. \(T\).add_edge(\(q_{\text{near}}, x_{\text{new}}, u\));
   5. if \(q_{\text{new}} = q\) then
      6. return \(\text{Reached}\)
   7. else
      8. return \(\text{Advanced}\)
9. return \(\text{Trapped}\)

**Algorithm 1.1.8: CONNECT**(\(T, q\))

1. repeat
2. \(S \leftarrow \text{EXTEND}(T, q)\)
3. until not (\(S = \text{Advanced}\));
4. return \(S\)

**LQR-RRT**

The work in [17, 16] presents a non-Euclidean distance metric to be used in the RRT algorithm. The proposed metric is based on the LQR cost-to-go functional. This measure estimates the cost to reach the sample from a node in the tree for the system.
obtained of the affine linearisation of the original system’s dynamics about the sample. A low cost-to-go indicates a more natural reaching motion from the node to the sample, considering the system’s dynamics. Since the metric is based on local linearisation, it works well when the node and the sample are already relatively close. The method was combined with RG-RRT and tested on several example models, showing that a good tree-coverage is obtained mainly in simple non-constrained examples but that for more complicated examples with non-holonomic constraints, the tree-coverage is similar to that of the tree obtained by using the traditional Euclidean distance. As the main purpose was to provide more efficient and natural exploration of state-space using RRT, a more precise cost-to-go function was utilised to serve as distance pseudo-metric instead of the Euclidian distance. The new metric relies on the LQR cost-to-go function obtained by linearizing the system’s dynamics about random samples. They also use the finite-horizon (with finite time being left as a free parameter for optimization of the cost) to arrive at a control strategy that assists steering states towards samples. The logic of the algorithm is the same as the basic (unbiased) RRT, except for two main natural modifications:

- The heuristic used to find the node in the tree nearest to the random sample is based on a finite-horizon affine quadratic regulator value function.
- The input used to steer the system’s state is according to the computed finite-horizon affine quadratic regulator aimed at bringing the system nearby the random sample in a fashion which would have been optimal if the system was linear.

It should be noted that the linearisation of the system is made once at every iteration about the random sample point. This defines a cost-to-go to the random sample in a finite time from any other state in the state space. Specifically, the cost-to-go from every node in the tree is calculated within a relatively short amount of time and the node with the minimal cost-to-go to the sample is chosen for expansion. The expansion is performed by applying the calculated optimal control policy within an optimized time-duration $0 < t_{\text{opt}} < t_f$ which is a parameter used to decide which is the best time to use to get to the sample with minimal cost. Interestingly, leaving $t_{\text{opt}}$ as a free parameter does not affect the computation of $S(t)$ or $d(t)$ which are both calculated using the
maximal allowed time of execution $t_f$.

**Algorithm 1.1.9: LQR-Based Proximity**

<table>
<thead>
<tr>
<th>Input: $x, x', \text{Plant_dynamics}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $A, B, c \leftarrow \text{Linearize_Plant_dynamics at } x'$;</td>
</tr>
<tr>
<td>2 $S(t) \leftarrow \text{COST_TO_GO}(A, B, c)$;</td>
</tr>
<tr>
<td>3 $d(t) \leftarrow e^{A(t_f-t)x} + \int_0^{t_f} e^{A(t_f-\tau)}Cd\tau$;</td>
</tr>
<tr>
<td>4 return $\min_{(0&lt;t&lt;t_f)} (t + \frac{1}{2}d^T S(t)d)$</td>
</tr>
</tbody>
</table>

**Algorithm 1.1.10: COST_TO_GO**

<table>
<thead>
<tr>
<th>Input: $A, B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Calculate $P(t)$ by solving: $\dot{P}(t) = AP(t) + P(t)A^T + BR^{-1}B^T$ with final condition $P(t_f) = 0$;</td>
</tr>
<tr>
<td>2 $S(t) \leftarrow P(t)^{-1}$;</td>
</tr>
<tr>
<td>3 return $S(t)$</td>
</tr>
</tbody>
</table>

The idea of the paper was to qualitatively and quantitatively prove that a more intuitive choice of the distance metric - a one which approximates better the true cost-to-go from one state to another - will improve the fast-exploration of the state space. For this sake, no goal state was defined, and the task was to cover the state space with a tree that starts at some starting state.

In the literature, the algorithm was applied on three typical test-systems:

- The double integrator (sliding frictionless brick) - for which the new measure of distance showed a significant improvement of state space coverage

- The simple torque-limited pendulum

- The Acrobot - a significant improvement was not obtained

It is also shown for the double integrator case with the cost function defined by the duration of the motion (for which the optimal solution is analytically known) that the Voronoi diagram using LQR heuristic is qualitatively almost identical to the exact one, while the Euclidian heuristic produces a completely different Voronoi diagram.

**RRT-Based Algorithms Involving Re-Planning and Optimization**

The literature in this topic can be organized into several groups. The first group is concerned with re-planning mainly in the case of changes in environment, without con-
sidering optimality of the trajectory ([13, 97, 50]). Hence, this group is excluded from review in this report.

A second group is relying on the RRT* idea of rewiring the tree based on costs of nodes and edges of the tree [26, 27].

The third group is based on the idea of gap-reduction [7], which shapes a given trajectory (found as an initial solution using any RRT) using a local optimization technique. A fourth group employs the optimal control to find locally [88, 17] or globally [60] optimal trajectories. These trajectories are already involving closed-loop control as part of the search.

**Online RRT***

This work in [27] presents a slightly modified version of the RRT* algorithm which is implemented in an online (“anytime”) manner to a non-holonomic system (a forklift).

The paper compares usage of RRT and RRT* algorithm implementations, both having the same modification to make them more fit to online application. The modification is that as the algorithm computes the planned motion \( x(t), u(t), t \in [0, T] \), the motion is executed until some ‘committed’ time \( t_{com} < T \) and while execution is in progress the algorithm re-searches for a new plan from \( x(t_{com}) \) to the target (it also deletes all the nodes with time less then \( t_{com} \)).

The RRT* algorithm itself completely relies on the suggested algorithm in a preliminary (and thorough) paper [26].

An additional feature is pruning of some nodes based on their cost in a process named branch-and-bound in order to decrease runtime.

The algorithm in [27] is similar in character of sampling and growing of a tree to a standard RRT algorithm. There are 3 main differences:

1. Assigning costs to vertices and edges of the tree, which represent the cost to go one node to another along the directed edges of the tree. Typically, the cost of a node is the cost of its parent plus the cost of the edge connecting it to its parent. The reference cost of the root node is taken as zero. The cost used in the paper has units of time.

2. The selection of the parent of a new node. First, an initial parent candidate is chosen using the nearest node in the tree just like in the classical RRT algorithm. However, later, and if the new edge is collision-free, a selected set of nodes in the
vicinity of the newly added node is checked to find the best parent according to cost considerations. The final parent will be the one for which the cost of the path from the root node to the new node is minimal (and collision-free).

3. A rewiring procedure takes place in order to allow changes in the tree when the newly added node can serve as an alternative parent node to any of its surrounding nodes with a lower cost to get from the root to these surrounding nodes.

The last two points are similar to search algorithms with optimality element such as A*. It is important to note the the radius of the sphere within which the set of “near” nodes are selected shrinks as the tree grows (the formula is described in detail in [26]) and this allows a significant improvement in the time it takes to calculate the path. In fact, the time is not much longer than for the RRT. The algorithm is proved in [26] to result in a probabilistic convergence to the optimal solution, i.e., if executed over an infinite number of iterations, the probability to find the optimal solution (if it exists) is one
1.1. BACKGROUND

("almost surely" under a certain set of assumptions).

**Algorithm 1.1.11:** RRT Body Algorithm

```plaintext
V ← \{x_{init}\}; E ← \emptyset; i ← 0;
while i < N do
    G ← (V, E);
    x_{rand} ← Sample(i); i ← i + 1;
    (V, E) ← Extend(G, x_{rand});
```

**Algorithm 1.1.12:** Extend (unique to RRT*)

```plaintext
V' ← V; E' ← E;
x_{nearest} ← Nearest(G, x);
x_{new} ← Steer(x_{nearest}, x);
if ObstacleFree(x_{nearest}, x_{new}) then
    V' ← V' \cup \{x_{new}\};
    x_{min} ← x_{nearest};
    X_{near} ← Near(G, x_{new});
    for all the x_{near} ∈ X_{near} do
        if ObstacleFree(x_{near}, x_{new}) then
            c' ← Cost(x_{near}) + c(Line(x_{near}, x_{new}));
            if c' < Cost(x_{new}) then
                x_{min} ← x_{near};
                E' ← E' \cup \{(x_{min}, x_{new})\};
        end if
    end for
    if ObstacleFree(x_{new}, x_{nearest}) and Cost(x_{near}) > Cost(x_{new}) + c(Line(x_{new}, x_{near})) then
        x_{parent} ← Parent(x_{nearest});
        E' ← E' \{ (x_{parent}, x_{near}) \};
        E' ← E' \cup \{ (x_{new}, x_{near}) \};
    end if
end if
end for
end if
end for
return Gf = (V', E')
```

For the example of the forklift robot in [27], the paper demonstrates both on a real-world example and also in various monte-carlo simulations that the results of the RRT* implementations are more optimal and more "neat" without unnecessary curves or prolonged path-choices.
In addition, the re-planning is demonstrated to be useful as the initial plan may be less optimal, but as the search continues starting from a state which is closer to the target, it is more likely to arrive at more optimal solutions.

This algorithm is good for use only when the amount of time required for planning a valid candidate trajectory is significantly shorter then the time required to execute it. This allows enough time for online re-planning. In this paper, the system was indeed non-holonomic, but with an analytical guide to steer the car-like system from one state to another in a given amount of time. The computation of a path takes around one second, while typical driving executing the planned trajectory with a closed-loop control takes several minutes.

This privilege does not exist normally for under-actuated systems.

LQR-RRT*

The work in [60] is based on the RRT* algorithm presented in [26] but differs only in the “distance” function: LQR based instead of Euclidian. The idea to use an LQR based heuristic is taken from [17]. The stages of the algorithm are identical, but the LQR distance heuristic provides a better estimation of the cost-to-go, hence allows solving problems which fail to be solved with a Euclidian distance metric.

The LQR is designed anew at each iteration about the random sample $x_0$ and a zero nominal input $u_0 = 0$. The dynamics of the nonlinear system $\dot{x} = f(x, u)$ system are approximated about the working point $(x_0, u_0)$ to be $\dot{x} \approx A(x_0, u_0)\bar{x} + B(x_0, u_0)\bar{u}$ and the control policy is defined as $\bar{u}^* = -R^{-1}B(x_0, u_0)S$ where $S$ is the solution of the ARE used with matrix $Q$ chosen to be the identity matrix.

The distance pseudo-metric used to estimate the optimal cost-to-go from a node $v$ in the tree to the sample $x_0$ is taken as $c(v) = (v - x_0)^TS(v - x_0)$ which would have been exact for a linear system. This distance function is inaccurate in the initial formation stages of the tree, but as it covers more of the search space, it becomes more accurate and reliable then the Euclidian distance metric.

In addition, the choosing of the NEAR set of nodes and the pruning of nodes in the ‘branch-and-bound’ technique are done in a similar manner as in [26]: The NEAR set is defined as

$$LQRNear(V, x) = \left\{ v \in V : (v - x)^TS(v - x) \leq \gamma \left( \frac{\log n}{n} \right)^{1/d} \right\}$$
and the pruning is done to all the nodes with the total cost-to-go (from the root to the node) greater than the current total cost-to-go of the node nearest to the target.

**Algorithm 1.1.13: LQR - RRT*((V,E),N)**

1. for i=1,...,N do
2. \( x_{\text{rand}} \leftarrow \text{Sample} \);
3. \( x_{\text{nearest}} \leftarrow \text{LQRNearest}(V,x_{\text{rand}}) \);
4. \( x_{\text{new}} \leftarrow \text{LQRSteer}(x_{\text{nearest}},x_{\text{rand}}) \);
5. \( X_{\text{near}} \leftarrow \text{LQRNear}(V,x_{\text{new}}) \);
6. \((x_{\min},\sigma_{\min}) \leftarrow \text{ChooseParent}(X_{\text{near}},x_{\text{new}})\);
7. if CollisionFree(\(\sigma\)) then
   8. \( X \leftarrow X \cup \{x_{\text{new}}\} \);
   9. \( E \leftarrow E \cup (x_{\min},x_{\text{new}}) \);
   10. \((V,E) \leftarrow \text{Rewire } ((V,E),X_{\text{near}},x_{\text{new}})\);
8. return \(G = (V,E)\);

**Algorithm 1.1.14: ChooseParent(\(X_{\text{near}},x_{\text{new}}\))**

1. \( \text{minCost} \leftarrow \infty; x_{\min} \leftarrow \text{NULL}; \sigma_{\min} \leftarrow \text{NULL} \);
2. for \(x_{\text{near}} \in X_{\text{near}}\) do
3. \( \sigma \leftarrow \text{LQRSteer}(x_{\text{near}},x_{\text{new}}) \);
4. if Cost\(x_{\text{near}}\) + Cost\(\sigma\) < minCost then
   5. \( \text{minCost} \leftarrow \text{Cost}(x_{\text{near}}) + \text{Cost}(\sigma) \);
   6. \( x_{\min} \leftarrow x_{\text{near}}; \sigma_{\min} \leftarrow \sigma \);
7. return \((x_{\min},\sigma_{\min})\);

**Algorithm 1.1.15: Rewire((V,E),X_{\text{near}},x_{\text{new}}))**

1. for \(x_{\text{near}} \in X_{\text{near}}\) do
2. \( \sigma \leftarrow \text{LQRSteer}(x_{\text{near}},x_{\text{new}}) \);
3. if Cost\(x_{\text{new}}\) + Cost\(\sigma\) < Cost\(x_{\text{near}}\) then
   4. if CollisionFree(\(\sigma\)) then
      5. \( x_{\text{mparent}} \leftarrow \text{Parent}(x_{\text{near}}) \);
      6. \( E \leftarrow E \setminus \{x_{\text{mparent}},x_{\text{near}}\} \);
      7. \( E \leftarrow E \cup \{x_{\text{new}},x_{\text{near}}\} \);
8. return \((V,E)\);
1.1. BACKGROUND

The LQR-RRT* algorithm is applied to three different systems:

1. The torque-limited simple pendulum. The task was to bring it to the upright state in rest.

2. The Acrobot system. The task was to bring it to the upright state in rest.

3. A robotic agent in a belief space of dark-light domain. The task was to bring the agent from one position to another under uncertain-self-positioning ability.

In the first case, the authors demonstrate qualitative difference in the planned path between using $R = 1$ and $R = 50$. They also show that the cost of the LQR-RRT* converges to the optimal cost computed using dynamic programming while the plans obtained by the non-optimal algorithm LQR-RRT has a much higher cost and no convergence. The time of finding a solution is 4.8 and 22.2 seconds for the LQR-RRT and LQR-RRT* algorithms, respectively.

Similar results were found for the other systems. For the Acrobot system, the timings were: 109 vs. 136 seconds. In all cases, 5000 nodes were enough to arrive at a nearly-optimal solution for all three examples.

Sampling Heuristics for Optimal Motion Planning

A modification to the original RRT* algorithm by suggesting a different random sampling technique which is used to reduce the cost of an initially found trajectory was presented in [1]. The unmodified (original) RRT* indeed promises convergence to the optimum trajectory, however, it may take a lot of iterations to find a trajectory which is close enough to the optimum and the search may become exhaustive. To try and make the sinking to optimum cost more efficient without harming the probabilistic optimality guarantee, the approach of [1] is to bias the samples towards the surroundings of the current best path once such path is found. In addition, the effect of separate and combined node-rejection strategy was studied and a Bi-directional search for a 7-DOF system was implemented.

The algorithm in this paper relies on the EXTEND_RRT* of the RRT* algorithm (without change). However, in contrast to the original RRT* algorithms where samples are collected from a uniform distribution (possibly with a bias $0 \leq \theta \leq 1$ towards the target
in a single-directional RRT*), here this is true only until an initial trajectory (connecting the start and the target) is found. After the algorithm finds an initial trajectory, or “a seed”, the algorithm continues to search for the optimal trajectory, however its sampling is now biased differently. With a bias of $0 \leq \beta \leq 1$ and a random number $0 \leq p \leq 1$, if $p < \beta$, the local biasing is applied, otherwise, a random sample is generated as usual. The local biasing functionality is as follows: an internal node $q$ is chosen within the currently-best path and a direction $q_{\text{rand}}$ is defined between $q$ and the geometric average of its parent and child nodes along the path. Then the sample is chosen along the defined trajectory with distance from $q$ being some random value between the (design) parameters $r_{\text{min}}$ and $r_{\text{max}}$. The reasoning behind this new method of sampling is trying to sample more about existing nodes within paths in order to refine the paths. If this is not done, then naturally, external (frontier) nodes in the tree have more odds to be selected for expansion, causing the tree to grow and expand faster - “exploration” (this is sometimes desired and purposely enhanced - see [75]) but slowing the process of tweaking the chosen trajectory to reduce the cost - “exploitation”.

The node rejection mechanism is employed from the start of the algorithm, similarly to the branch-and-bound technique. A node $q$ will be rejected in case that: $\|q - q_{\text{start}}\| + \|q_{\text{target}} - q\| > c_{\text{best}}$. This condition means that a node will be either pruned from the tree when the total path length from the start to the node (in a minimum possible cost) and then from the node to the target (in a minimum possible cost) is longer than the currently available best-path. This is relevant only when the cost functional obeys the
triangular inequality.

<table>
<thead>
<tr>
<th>Algorithm 1.1.16: PLANNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $V \leftarrow q_{init}$;</td>
</tr>
<tr>
<td>2 $E \leftarrow \theta$;</td>
</tr>
<tr>
<td>3 while $i &lt; N$ do</td>
</tr>
<tr>
<td>4 \hspace{1em} $q_s \leftarrow \text{SAMPLE}(i)$;</td>
</tr>
<tr>
<td>5 \hspace{1em} if not $\text{NODE}_\text{REJECT}(q_s)$ or not $\text{PathFound}$ then</td>
</tr>
<tr>
<td>6 \hspace{2em} $\text{EXTEND}_\text{RRT}^*(V, E, q_s)$;</td>
</tr>
<tr>
<td>7 \hspace{1em} end if</td>
</tr>
<tr>
<td>8 \hspace{1em} end while</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 1.1.17: SAMPLE(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $p \leftarrow \text{RAND}(0.0, 0.1)$;</td>
</tr>
<tr>
<td>2 if $\theta &lt; p$ and not $\text{PathFound}$ then</td>
</tr>
<tr>
<td>3 \hspace{1em} return $q_{goal}$</td>
</tr>
<tr>
<td>4 \hspace{1em} else if $\beta &lt; p$ and $\text{PathFound}$ then</td>
</tr>
<tr>
<td>5 \hspace{2em} return $\text{LOCAL}_\text{BIAS}(\text{path})$</td>
</tr>
<tr>
<td>6 \hspace{1em} else</td>
</tr>
<tr>
<td>7 \hspace{2em} return $\text{RAND}_{q}$</td>
</tr>
<tr>
<td>8 \hspace{1em} end if</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 1.1.18: LOCAL_BIAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $q \leftarrow \text{RAND}_\text{NODE}(\text{path})$;</td>
</tr>
<tr>
<td>2 $q_1 \leftarrow \text{path}(q - 1)$;</td>
</tr>
<tr>
<td>3 $q_2 \leftarrow \text{path}(q + 1)$;</td>
</tr>
<tr>
<td>4 $q_{\text{tmp}} \leftarrow \frac{q_1 + q_2}{2} - q$;</td>
</tr>
<tr>
<td>5 $q_{\text{rand}} \leftarrow \frac{q_{\text{tmp}}}{|q_{\text{tmp}}|} \cdot \text{RAND}(r_{\text{min}}, r_{\text{max}})$;</td>
</tr>
<tr>
<td>6 return $(q_{\text{rand}})$</td>
</tr>
</tbody>
</table>

Unfortunately, this algorithm was not demonstrated on a kinodynamic system, let alone, system with underactuation. It was demonstrated to work better than the plain RRT* on two example systems:

1. 2D navigator (i.e., translating agent). In this case it was enough to use a uni-directional RRT* to show how the modifications affect the resulting search.

2. 7D robotic manipulator (presumably, only in configuration space). In this case, a bi-directional search was used as a uni-directional RRT* search was not efficient enough.
In both cases, a time-constraint was imposed on the algorithm so that when the time is up, the generated trees from the modified and the original algorithms are compared. The major observations were as follows:

- The node rejection is helpful mainly when the search space is sparse (i.e., not restrictive) but in any case, it does not significantly elongate the run-time, hence it was generally recommended.

- The local-biasing approach is generally useful (more in high-dimensional systems) and helps reduce the cost of the found trajectory a lot faster than the plain RRT*.

**LQR-trees**

The LQR-Trees approach [88] was introduced earlier in the context of hybrid control. The growth of the LQR tree is based, however, on the RRT algorithm in the following manner:

The algorithm starts from the target state and aims to cover the whole state space (or a subset of it which is relevant for consideration) with funnels. Initially, a region of attraction is estimated about the target state. At each iteration, a sample is chosen outside the region of attraction of the current LQR tree and a trajectory is created by connecting (approximately) the existing tree to the sample via a local trajectory generator (for instance, shooting methods, or direct collocation). Then a region of finite-time invariance (funnel) is calculated around the new trajectory and added to the
region of attraction of the whole tree.

**Algorithm 1.1.19:** SAMPLE(k)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>([A, B] \leftarrow \text{linearization of } f(x, u) \text{ around } x_G, u_G;)</td>
</tr>
<tr>
<td>2</td>
<td>([K, S] \leftarrow \text{LQR} (A, B, Q, R);)</td>
</tr>
<tr>
<td>3</td>
<td>(\rho_c \leftarrow \text{time-invariant level-set}; \quad \text{// computed as described in section 3.1.1})</td>
</tr>
<tr>
<td>4</td>
<td>(\text{T.init(}{x_G, u_G, S, K, \rho_c, \text{NULL}});)</td>
</tr>
<tr>
<td>5</td>
<td>(\text{for } k = 1 \text{ to } K \text{ do})</td>
</tr>
<tr>
<td>6</td>
<td>(x_{\text{rand}} \leftarrow \text{random sample;})</td>
</tr>
<tr>
<td>7</td>
<td>(\text{if } x_{\text{rand}} \in C_k \text{ then})</td>
</tr>
<tr>
<td>8</td>
<td>(\text{continue ;} \quad \text{// sample rejected because it is within the current})</td>
</tr>
<tr>
<td>9</td>
<td>(\text{region of state space steerable to the target})</td>
</tr>
<tr>
<td>10</td>
<td>([t, x_0(t), u_0(t)] \text{ from trajectory optimization with a &quot;final tree constraint&quot;;})</td>
</tr>
<tr>
<td>11</td>
<td>(\text{// as described in section 3.2})</td>
</tr>
<tr>
<td>12</td>
<td>(\text{if } x_0(t_f) \not\in T_k \text{ then})</td>
</tr>
<tr>
<td>13</td>
<td>(\text{continue ;} \quad \text{// ran out-of-time before any nominal trajectory})</td>
</tr>
<tr>
<td>14</td>
<td>(\text{(connecting the sample to the tree) was found})</td>
</tr>
<tr>
<td>15</td>
<td>([K(t), S(t)] \leftarrow \text{time-varying LQR} (A(t), B(t), Q, R); \quad \text{// as described in})</td>
</tr>
<tr>
<td></td>
<td>(\text{section 3.3})</td>
</tr>
<tr>
<td>16</td>
<td>(\rho_c(t) \leftarrow \text{time-varying level-set;} \quad \text{// computed as described in section})</td>
</tr>
<tr>
<td>17</td>
<td>(\text{3.3.1})</td>
</tr>
<tr>
<td>18</td>
<td>(i \leftarrow \text{pointer to branch in } T \text{ containing } x_0(t_f);)</td>
</tr>
<tr>
<td>19</td>
<td>(\text{T.add - branch(}{x_0(t), u_0(t), S(t), K(t), \rho_c(t), i};)</td>
</tr>
</tbody>
</table>

**Symmetry-Exploiting Gap Reduction**

The work in [7] presents a technique for reducing the resulting gap of motion plans obtained from RRT-like algorithms which can be applied for systems with group symmetries. The technique uses low-precision RRT solution first and then refines them with a small amount of extra computation due to the exploitation of the symmetry properties. The technique also suits PRMs and problems with differential constraints.

1.1.1.2 **Forward Dynamics**

Forward dynamics motion planning strategies, also frequently called ‘shooting methods’, are normally used when feedback linearisation is not possible for the system, such as due
1.1. BACKGROUND

to under-actuation. Therefore, there is an abundance of papers using forward dynamics motion planning strategies to create gaits for under-actuated bipedal mechanisms.

The basic idea of forward dynamics is parametrising the input of the system using functions (such as Bezier polynomials of finite order, finite sums of sinusoids with free coefficients, or piecewise linear functions with free choice of slopes) and then using a numerical constrained optimization to find a good set of parameters which result in the desired motion.

This simple idea was implemented in [80], where open-loop gaits of walking and running were found for a simplistic under-actuated bipedal mechanism. The method in [80] was to choose the stride duration first as a gait indicator and divide this duration into many short time segments. Then, the generalized forces in the actuators were assumed piecewise linear, where each linear part corresponds to a time segment. An optimization algorithm (Sequential Quadratic Programming) was employed to choose a set of parameters which characterise the piecewise linear input. At each iteration of the optimization program, a certain parameters set was chosen according to the algorithm’s decision-making protocol and then the dynamics of the system were simulated with the generated force (i.e., input signal) in order to produce a resulting trajectory. The trajectory is then examined with respect to some constraints such as periodicity and feasibility. While trying to achieve motions which meet the constraints of the problem, the optimization program also attempts to minimize some cost function such as energy expenditure during the stride. The stability of the gait was not analysed in the paper, and since the trajectories are obtain by using a numerical solver, they intrinsically contain numerical computation errors, which means that the trajectories are imperfect and require an appropriately validated stabilizing control.

Another important example of forward dynamics motion planning to achieve periodic gaits can be found in [96] and many related papers [95, 55, 9, 79, 89, 56, 94, 92, 93] based on the theory of hybrid zero dynamics. The hybrid zero dynamics method makes use of defining virtual constraints and asymptotically driving the system to satisfy these constraints, taking into account the impacts with the ground in a way which will not destroy this behaviour.

In order to define the virtual constraints, a parameter is chosen to represent the gait.
1.1. BACKGROUND

This parameter has to change monotonically in a course of a stride. Normally, this parameter can be a generalized coordinate in the modelling of the system, such as the angle between the walking surface and the line connecting the stance foot to the stance hip. Then, using partial feedback linearisation, the generalised forces in the actuators are computed to realize some desired motion of the actuated degrees of freedom, which is parametrised with respect to the monotonic parameter (instead of time) and some other free decision coefficients. This desired motion of the actuated degrees of freedom affects the behaviour of the non-actuated degrees of freedom. Here again, an optimization algorithm (usually \texttt{fmincon()} function in MATLAB) is used to alter the free decision parameters at each iteration and then integrate the systems equations of motion (a lower dimensional system is required here, since the behaviour of the actuated degrees of freedom is known) in order to inspect the behaviour of the whole system for this choice of parameters. The algorithm attempts to converge to a set of parameters which result in a desired motion for the whole system and also locally minimizes a cost function. The desired behaviour is a periodic gait which satisfies some feasibility constraints.

The use of optimization software to find a good set of parameters is, in a way, a black-box solution to the problem of finding a feasible motion plan. The optimization software is usually generic and lacks problem-specific strategies for a good updating technique for the set of parameters from one iteration to the next. Each iteration takes a relatively long time, since it requires integration of the equations of motion over the whole duration of a stride, and if the resulting motion is infeasible, it is not of much use to the designer. In addition, the software is usually initialized with an almost random set of parameters (since it is hard to predict the motion of the system by ‘hand-picked’ parameters) which then may often result in one three potential bad outcomes:

- The convergence rate of the software is too low and the program has to be stopped due to practical considerations. Note that in most cases it is not possible to estimate in advance an upper bound for how long it will take the software to converge to a good set of parameters.

- The software returns a failure output. This happens often when the software fails to find a feasible solution from the given initial condition. Since, the designer usually lacks a constructive manner of choosing initial conditions for the software, the process may require many running attempts, without much knowledge learned in each. In other words, once a run of the program fails to find a feasible trajectory,
1.1. BACKGROUND

it is hard to point on the exact reason why, estimate whether such a trajectory exists, and what should be changed from the original attempt in order to find a solution.

- The software returns a feasible solution which is obviously very far from the actual optimum. Again, when improperly initialized, the resulting local minimum can be a feasible motion which is very undesirable, such as a ‘flip-flop’ walking characterized by having the swing leg rotating above the hip.

In total, forward dynamics is a good and intuitive ‘off-the-shelf’ motion planning technique, however, it faces several crucial disadvantages which are out of the designer’s control. In addition, the resulting motion plans have some imperfections since the equality constraints are satisfied only up to some acceptable numerical tolerance. This can frequently result instability over one or more strides (in the walking motion example) and hence, a stabilizing control is required to be added to the nominally planned trajectory. In their work, [96] present very useful techniques for stabilizing control stability and verification. The dependence on the success of finding good nominal motion plans using the optimization algorithms remains the bottle neck of the method.

1.1.1.3 Inverse Dynamics

The method of inverse dynamics can always be used when the dynamic system is fully actuated, and hence feedback-linearisable. In this case, each continuous path in the state space (or, configuration space) of the system can be realized by simply computing the required inputs that will realize the motion.

This is normally impossible in the presence of under-actuation, since not every trajectory of the state of the system is realizable using the reduced number of actuators. However, the idea of inverse dynamics can still be used in a sense, by presenting a so-called virtual actuation in the place of the non-actuated degrees of freedom, then using the inverse dynamics strategy in an iterative manner to attempt finding a trajectory for which the resulting torque in the virtual actuators is as close as possible to zero.

As opposed to the method of forward dynamics motion planning, where the generalized forces are chosen first and the motion is solved for and then examined, the inverse dynamics motion planning technique [82, 12] first determines a desired trajectory plan
for all of the degrees of freedom (even if this is not a feasible plan) and then computes
the required generalized forces (both real and virtual). Optimization software are used
to iteratively generate desired trajectories in the state space (which inherently satisfy
conditions such as periodicity or final state constraints). At each iteration, the required
generalized forces are calculated algebraically. In contrast to the long numerical integra-
tion of the system’s dynamics done in the forward dynamics motion planning approach,
this computational operation is very short. The optimization program then attempts
to minimize some cost function related to both the overall usage of the virtual actuator
and other common qualitative characteristics of the motion such as energy expenditure.

In practice, the resulting motion usually still depends on some residual virtual actuation
that was not identically zeroed in the optimization process, and is therefore infeasible.
Simulating the motion with zero input in the non-actuated degrees of freedom nor-
mally results in trajectories which are quite close to the nominal planned trajectory
but without adding a proper stabilizing control, the resulting motion is prone to result
in instability. It should be noticed that due to the almost inevitable non-zero residual
virtual actuation, the planned motion is infeasible, and therefore the resulting actual
implementation does not comply with the original constraints.

1.1.1.4 Miscellaneous Motion Planning Approaches

Another motion planning technique which is used in the literature for gait planning is
based on passivity ideas [78, 77] and it takes advantage of natural gaits which can be
achieved by bipedal walkers down a small slope due to gravity only (with no actuation).
By adding actuation and energy-shaping control, it is possible to generate stable similar
gaits for these walkers on plain surfaces or even uphill slopes. However, since these
motion planning ideas are very specific to walking and require a complete new analysis
for each separate motion planning problem, we choose to not focus on it here.
In addition, some planners are based on a technique called direct collocation (see for
instance gait-design application in [10]). This approach is similar in nature to a combi-
nation of forward and inverse dynamics motion planning, however we do not dwell on it
here and simply remark that it can be used instead of our RRT algorithm to generate
motion plans in various applications.
1.1.2 Global Stabilisation of Desired Trajectories

The problem of global stabilisation of a desired behaviour can be addressed in two main approaches: using a continuous globally-applicable control law for tracking of a planned trajectory, or using a hybrid control scheme which switches between various locally-applicable control laws. The first approach may present various disadvantages such as restrictive assumptions on the system or the difficulty of computing a global tracking control law for a general system and a general nominal trajectory. Therefore, we choose to focus on the second approach, which allows a more flexible implementation of desired trajectories.

1.1.2.1 Global Tracking Techniques

Tracking (local or global stabilization) of a given nominal trajectory is a standard problem in control theory. In the case of time-invariant linear systems, global tracking of a nominal trajectory is easily achieved through the stabilization of the error coordinate system. For general nonlinear systems, we have already described the procedure for local tracking via linearization around the nominal trajectory and the use of classical techniques for stabilization of linear time-varying systems (e.g. LQR control).

The literature on global tracking of nominal trajectories for nonlinear systems is very rich and various approaches have been reported in the literature. Adaptive control techniques have been used for the global tracking of unknown linear systems [23]; adaptive techniques in general may suffer from the lack of robustness unless certain persistence of excitation conditions hold (which are in general very hard to check). Several approaches to global tracking for classes of nonlinear systems (e.g. feedback linearizable systems) can be found in [24]. Model predictive control can also be used for the purposes of global tracking of nonlinear systems [51]. Various other techniques exist but we will not present them as they are outside the scope of this thesis. Instead, in the next section we concentrate on several recently published techniques based on hybrid control that are directly related to this thesis.
1.1. BACKGROUND

1.1.2.2 Global Stabilisation via Hybrid Control

In this section we briefly present three control approaches which highly motivate the work in the thesis. The first two approaches use hybrid system formalism from [21] and obtain robustness results. The third framework is not immediately related to this formalism. However, we later show how it can be presented using it. Furthermore, we demonstrate in the next chapter how the first and the third approaches can be viewed as special cases of our general framework. Additional examples of hybrid control frameworks which are not reviewed in detail here are [19, 68].

Various modelling frameworks have been proposed to study hybrid dynamic systems and their solutions. Among these, a recent and thorough modelling framework was published in [20, 21] and it will be used throughout this thesis as a baseline for the hybrid control framework that we develop. In their work, an extensive amount of examples of hybrid systems are included as well as explanations for how their framework relates to others. Just to give a couple of examples, the work in [58, 57] uses the hybrid systems framework in [21].

The work here is based on the fundamental developments in [20, 21] which establish the ground for modelling and analysing hybrid systems. Hybrid systems are systems which have two possible distinct types of behaviour of the state of the system: flow and jump. Flow behaviour corresponds to a continuous change of the state with respect to time in the form of a solution to differential equations, whereas jump behaviour corresponds to an instantaneous change in the state of the system due to an impulsive event in the form of a solution to a difference equation. In contrast to differential or difference equations (or inclusions), hybrid systems permit the occurrence of both flows and jumps by considering the two types of equations (or inclusions) on different sets in the state space of the system.

Many dynamical systems cannot be described with either purely continuous or purely discrete models and require the combination of these in their modelling in order to account for the true set of behaviours which they can exhibit. A common example to a hybrid system is a bouncing ball. Assume gravity is acting in the negative vertical axis. In order to describe the behaviour of the ball over time, we consider its continuous dynamics once it is above the ground level or on the ground level but with a non-negative
velocity. The continuous dynamics of the ball will correspond to $F = ma$, where $m$ is the mass of the ball, $a$ is its acceleration and $F = -mg$ is the gravity force. These dynamics can be described as a two-dimensional system of first order differential equations with the state of the system corresponding to the vector $(x, \dot{x})$, i.e., the position and velocity of the ball. In addition, we consider the discrete behaviour of the ball when its height is zero and its velocity is negative. In this case, the state of the ball changes instantaneously according to an impact model such that immediately after the impact, the position of the ball is still zero and its velocity becomes positive.

Another example for a hybrid system is a walking mechanism. The dynamics can be divided into the continuous swing phase and the discrete double-support impact phase which corresponds to the impact of the swing foot with the walking surface and change of roles between the support and the swing legs. The event that triggers an instantaneous jump in this case is the zeroing of the height of the swing foot while its velocity in the vertical direction is downwards.

Another type of hybrid systems is systems with hybrid control. For instance, one may append the state of a feedback-controlled system with a timer state which can be reset. Normally, the timer state grows with a positive rate with time, however, once some condition on the total state is satisfied, the timer state can be reset and the control signal, which depends on the timer state, will be affected as a result of the jump in the timer state. Other examples for hybrid systems can be found in [20].

Consider a system with a state $x \in \mathbb{R}^n$ and a hybrid time $(t, j) \in \mathbb{N}_{\geq 0} \times \mathbb{R}_{\geq 0}$. The data of a hybrid system can be arranged into 4 entities:

- Flow map, denoted by $F$ for set-valued mappings (like in differential inclusions) and $f$ for single valued mappings (like in differential equations). The flow map defines the flow dynamics of the system when the state of the system experiences a continuous time-dependant evolution, termed flow. The rate of change of the system’s state is denoted $\dot{x}$.

- Flow set, denoted $C$. The flow set is the region in the system’s state space where the behaviour of the system is governed by the flow map.

- Jump map, denoted by $G$ for set-valued mappings (like in difference inclusions) and $g$ for single valued mappings (like in difference equations). The jump map
defines the instantaneous change of the state of the system when it undergoes a jump. The state of the system right after the jump is denoted $x^+$. 

- **Jump set**, denoted $D$. The jump set is the region in the system’s state space where jumps are allowed. If the state of the system belongs to $D$ but not to $C$, then a jump will certainly occur. If the state of the system belongs to $C \cap D$, then both flow and jump may occur (multiple solutions may exist).

The general formalism of a hybrid system is:

\[
\begin{align*}
 \dot{x} & \in F(x), \ x \in C, \text{ or } \tag{1.18} \\
 \dot{x} & = f(x), \ x \in C, \tag{1.19} \\
 x^+ & \in G(x), \ x \in D, \text{ or } \tag{1.20} \\
 x^+ & = g(x), \ x \in D \tag{1.21}
\end{align*}
\]

A hybrid system is often denoted by $\mathcal{H}$ using its data as vector entries $\mathcal{H} = (C, F, D, G)$. We present further mathematical details on hybrid systems in the next chapter. In the rest of this section, we discuss three different global tracking approaches that use hybrid controllers.

**Throw and Catch**

In [72, 83], the authors present a robust hybrid control framework which switches between three types of controllers: throw, catch and bootstrap.

The throw control is designed to steer trajectories from the neighbourhood of some equilibrium state to the neighbourhood of another equilibrium, possibly using open-loop pre-designed control policy. Each equilibrium state except for the one which is chosen to be eventually stabilized has its own defined throw control which brings trajectories from its vicinity towards another designated equilibrium. The Catch control is a locally stabilizing control designed to attract trajectories of the system towards the current equilibrium set. The Catching stabilization lasts indefinitely for the last equilibrium set, which is made globally stable. Every other catch, however, lasts a finite time duration, since the ensuing throw control commences once the trajectory arrives.
close enough to the equilibrium set, from where it can now be steered towards the next equilibrium set. The Bootstrap control is an additional component in the method which relies on the assumption that there exists a state-feedback control that can steer the state of the system to some desired neighbourhood of any of the equilibria.

In order to globally stabilize a particular equilibrium set, a high-level tree-structure connects the equilibrium sets of the system such that the root of the tree is the set which has to be globally stabilized. A pair \((p, q)\) refers to the \(q\)-th equilibrium set on the \(p\)-th branch in the tree (\(q_{\text{max}}\) is the last node on the \(p\)’th branch and for all values of \(p\) refers to the ultimate equilibrium set being globally stabilized).

An augmented state of the system can be defined as \((\tau, z, p, q)\), where \(\tau\) is a timer state used to measure which part along the current throw control has already been executed (the value of \(\tau\) is reset at each throw), \(z\) is the continuous state of the dynamic system, and \(p\) and \(q\) hold information about whether or not the bootstrap controller operates. If bootstrap controller is in operation, then both \(p\) and \(q\) have zero values, otherwise it would specify whether catch or throw control is in operation.

The flow sets of the hybrid control system are defined as regions in the state space of the augmented systems state where jumps between the various controllers are not allowed. Relevant jump sets are defined for transitions from catch to throw, from throw to catch, from throw or catch to bootstrap, and from bootstrap to throw or catch. The augmented state space is covered by flow and jump sets such that any given state cannot simultaneously belong to two different flow sets or to two different jump sets. In addition, the intersections between relevant flow and jump sets contain open non-empty regions. These hysteresis properties are shown to guarantee robustness of the hybrid control system to measurement noise and external disturbances.

The overall strategy is to cascade a trajectory starting anywhere in the state space via the neighbourhoods of other equilibria towards the final equilibrium that is to be globally stabilized. The ultimate equilibrium state is then a root of a tree of the system equilibria and the cascading is done according to the pre-defined tree structure, where initially trajectories arrive to the vicinity of one node and then continue along the branch to which that node belongs.
1.1. BACKGROUND

In a normal operation of the hybrid control system, when the initial augmented state of the system lies in any of the catch or throw flow sets, a cascaded switching between catch and throw control is done along the same branch $p$ until the last catch towards the ultimate equilibrium set which is globally stabilized.

When the system is improperly initialized, or when due to disturbances or noise, the augmented state does not behave as expected (for instance, exiting the reachability set from the last used throw-to-catch jump set using the current stabilizing controller during a catch, or exceeding the maximal allowed time of a throw), then the bootstrap control is initiated and it operates for a time duration until the augmented state can be updated (without changing $z$) to an admissible value inside any of the throw or the catch flow sets, thus allowing recovery of the normal behaviour of the system.

We give more specific details about the Throw and Catch approach at the end of the next chapter, and show in detail how it falls as a special case of our general framework. It should be noted that we use a similar recovery policy in the shape of a bootstrap control which can be turned on and off on demand, and show that within up to two jumps, normal behaviour can be resumed in the nominal hybrid control system. In addition, we are highly inspired by the hysteresis between the various jump sets which prevents chattering in the presence of measurement noise. Therefore, the construction of our flow and jump sets will be philosophically very similar.

Manoeuvre-Based Motion Planning

[71] introduced a hybrid-control framework for manoeuvre-based motion planning using pre-defined motion primitives for nonlinear systems with symmetries and proved its robustness to a variety of perturbation classes. The motion plans are based on the ones presented in [14] and are composed of trim trajectories and manoeuvres concatenated in an alternating order, such that they can be properly connected by matching displacements which are members of the Lie group with respect to which the symmetry of the dynamic system is defined. In [14], a method for motion planning for systems with symmetries by concatenating specific motion primitives (equivalence classes of trajectories in the dynamic system) properly to create motion plans from one state to another in the state space of a dynamic non-linear system with various constraints was presented. The framework in [71] defines a library of trim trajectories and manoeuvres,
1.1. BACKGROUND

where trim trajectories are relatively simple motion classes based on a constant input for which the motion can be predicted by the initial condition and the displacement which characterises it (for a known left action that preserves the symmetry condition) and manoeuvres are prescribed motions based on some possibly-time-varying input and execution time. A motion plan is an ordered set composed of these manoeuvres and trim trajectories together with the proper displacements and time durations that define how the initial (respectively, final) state of the a manoeuvre can be connected to the final (respectively, initial) state of the preceding (respectively, ensuing) trim trajectory. In a similar way to the Throw-and-Catch approach, trim trajectories are treated as attractors using a locally stabilizing feedback control to bring states in the neighbourhood of the trim trajectory towards it (in a form of a ‘catch), whereas manoeuvres are implemented in open loop (in the form of a ‘throw). In order to ensure the robustness of a motion plan, appropriate flow and jump sets are defined together with a hybrid controller which defines the flow and jump maps, in a way that ensures that starting in a certain region \( D_k \) about the initial state of the \( k \) trim trajectory brings the state to some other region from which, after implementing the \( k + 1 \) manoeuvre, the state of the dynamic system will be in the interior of the \( D_{k+1} \) from which the same property will hold. It is proven that all trajectories solving the system with perturbations in the initial condition or a system with exogenous disturbances or state measurement noise are ‘close to the nominal motion plan, with a notion of closeness defined for solutions of hybrid systems with possibly different hybrid time domains. This work can be treated as another special case of the general system-of-funnels framework by considering throwing and catching funnels about the manoeuvres and the trim trajectories, respectively, with appropriately defined entrances and outlets such that the same behaviour of the system is obtained under our hybrid controller.

LQR-Trees

The approach of LQR-Trees was presented in [85, 88]. With similarity to the Throw-and-Catch approach, the goal of constructing an LQR-Tree is to stabilize an equilibrium state. The approach combines various aspects such as motion planning and estimation of regions of attraction. For the sake of clarity of presentation, we describe the basic operation of the approach to explain how it gives rise to the construction of a hybrid control system, and later in we return to this approach to describe other aspects of it in more detail.
In the LQR-Trees approach, a tree of non-trivial trajectories is constructed in the state space of the dynamic system. The trajectories are connected to each other. The connections define parenthood relationships, that is, if the final state of one trajectory lies on another trajectory, then the other trajectory is said to be the parent of the first.

The root of the tree is the equilibrium state which has to be stabilized by the approach. Each nominal non-trivial trajectory in the tree is locally stabilized for a finite time duration using a time-varying linear quadratic regulator (TV-LQR) which attracts trajectories of the system starting inside some neighbourhood of the nominal trajectory towards it. A neighbourhood, called funnel, about each non-trivial nominal trajectory is computed numerically to ensure a property of finite-time invariance. A trajectory starting inside a funnel about some nominal trajectory with a known time duration is guaranteed to remain within that funnel for some finite amount of time when the TV-LQR control stabilizing the nominal trajectory is applied.

Another region is defined as a neighbourhood of the equilibrium state (the root of the tree) by finding an (infinite-horizon) invariant set for the system in closed loop with a time invariant linear quadratic regulator (TI-LQR).

Each nominal trajectory and its corresponding funnel are parametrised with respect to the same timer parameter. Each funnel is labelled with a unique integer label. Assume that the invariant region found using the TI-LQR about the root state is also labelled in the same manner. Together, all the funnels in the LQR-Trees approach cover the whole state space of the system or a subset of it from where the system can be initialized.

The outlet of a funnel is a region about the final state of the associated nominal trajectory to which all trajectories starting inside the funnel are steered using the associated controller. Since the nominal trajectories form a tree structure, so do their corresponding funnels. The outlet of each funnel is designed to be contained inside the funnel associated with the parent nominal trajectory to which the trajectory one is connected. The funnels associated with trajectories which are connected to the root state have outlets within the invariant set associated with the equilibrium state and its TI-LQR control.
When the system is initialized at some state, the shortest distance from this state to a nominal trajectory in the tree determines which controller will be used first and what will be the initial value for the timer along the associated funnel. The most suitable stabilizing controller is then applied to the system until the associated timer reaches the maximal duration of the trajectory. In this instant, the control is switched to the one associated with the parent trajectory and the timer is reset according to the connection state along the parent trajectory. This procedure repeats until the trajectory reaches the outlet of the funnels which corresponds to a trajectory connected to the root. The controller then switches for the last time to the TI-LQR control associated with the root state and keeps operating indefinitely.

Similar to the previously described approaches, the LQR-Trees approaches defines a switching technique that ensures a desired behaviour (in this case, the stabilization of an equilibrium state) is attained for the hybrid control system. However, in contrast to the previous approaches, it does not supply appropriate robustness mechanism for dealing with various disturbances or noise. Therefore, in the presence of unmodelled disturbances or noise, the system is at risk of having undesirable behaviours such as chattering about the boundaries of funnels. Nevertheless, we adopt many ideas from this approach in creating our system-of-funnels hybrid control framework. At the end of the next chapter, we demonstrate how this approach can be viewed as a special case of the system-of-funnels framework using some mild additional assumptions which deal with the robustness issue pointed above.

Construction of Funnels

An initial idea of using funnels can be seen in the mid-80s, when automatic motion planning and motion control in the presence of model inaccuracies and disturbances were sought after. A sub-problem often emerged in various contexts: how to practically find a set of initial conditions as well as motion plans which start from them, such that a final goal will be reached? For instance, the work in [38] discusses in detail the problem of inserting a peg into a hole (as a demonstrating example of a class of problems where some fine-motion strategy is required and modelling errors, as well as other disturbances, can be estimated). The peg-in-hole geometric problem was initially transformed into its configuration-space manifestation to create the equivalent problem of controlling a point to arrive to some desired surface. The idea was to initially calculate the pre-image
of the desired surface (of configurations), i.e., the set of all points in the configuration space from which it is possible to arrive at the surface in a straight line motion. Then, the process continues recursively to find the pre-images of the existing pre-images, until the initial condition lies within one of the found pre-images. This idea strongly relates to the idea of funnels by considering the pre-image as the entrance (or domain) of some funnel and the mapping of the pre-image as the outlet. The execution of the control algorithm is based on switching between various controllers, each of which is associated with some pre-image and goal region. This idea is also regarded as backward chaining (or, backchaining). The paper describes methods to regard uncertainties, compliant motion and friction. Finally, the control strategy gives rise to a reachability graph of velocity commands. The paper presents the idea of cascading controllers based on that pre-calculated graph of motions that reach from one pre-image to another, until the final reaching from the pre-image of the goal to the goal itself. However, the implementation of these hierarchic controllers is not directly discussed in terms of a trajectory passing through funnels. The transition from one controller to another is not directly referred to as arrival to the outlet of a funnel. Instead, some termination predicates are specified in order to trigger the next controller in the hierarchy. Nevertheless, the presented ideas are very closely related to their more modern versions, where the different controllers are each associated with a funnel.

In the work presented in [6], it is demonstrated how dexterous dynamic behaviours could be achieved in a robotic system using the same idea of backchaining. However, this time the method is presented using funnels-terminology. The set of funnels corresponds to a set of controllers that are hierarchic in such a way that a final desired steady state is achieved by sequentially evoking a series of the controllers, which are chosen depending on the initial conditions. The system which is used to demonstrate the approach is a physical paddle and ball. In addition, the paddle has a beam which is regarded as an obstacle. The objective is to make the ball reach a resting state in a particular position on the paddle through several bounces, without hitting the beam. In order to carry out the task, a pallet (set) of controllers (‘jugglers’) is formulated, some of which are used for palming, while others, for catching. Each controller is attributed to a funnel, Φ. The domain of the funnel, denoted \(D(\Phi)\) represents the set of states which are known to be “safely” steered to a goal set of the funnel, denoted \(G(\Phi)\). The funnels were computed by empirically testing the behaviour of the system from various initial condition, and afterwards establishing conservative estimates of positively-invariant regions in state-
space for various families of controllers. The state-space (or, a large portion of it) was partitioned into cells such that if an initial condition belongs to some cell, it belongs to some domain of a funnel. A funnel $\Phi_i$ is said to be prepared by another funnel, $\Phi_j$, (denoted $\Phi_i \succeq \Phi_j$) if $G(\Phi_i) \subset D(\Phi_j)$. The set of all funnels is structured in a tree graph here as well, so that switching from the controller of funnel $i$ to that of funnel $j$ occurs when the state of the system arrives at the goal of funnel $i$, which is a subset of the domain of funnel $j$. Here, too, the root funnel ensures arrival of the ball to the designated point and remaining there at rest. In addition, all of the funnels are intentionally locally designed to guarantee that trajectories passing through them will not be at prohibited areas which arise due to the existence of the beam. Since the initial construction of the set of funnels does not necessarily produce a tree-structured graph, but a graph that can contain cycles, a breadth-first search is performed in order to create a unique chain of funnels that leads to the ultimate goal state from each initial condition. This idea, albeit with a more analytically precise construction of the invariant funnel regions was later reported in the literature, as will be reviewed in the following.

In the recent years there has been a significant development in the ability to solve sum of squares (SOS) programs by transforming them into problems of finding semi-definite matrices and solving them very efficiently using convex optimization [59, 61]. For instance, SOSTOOLS [66, 67] is a MATLAB toolbox which allows convenient representation of symbolic polynomials, definition and solution of SOS programs.

This development immediately affected the ability to compute inner estimations of regions of attraction, or various invariant or finite-time-invariant sets. This is because it allowed finding Lyapunov functions for which the satisfaction of positive definiteness condition (required for ensuring various stability properties) was ensured by a more restrictive condition of a function being a sum-of-squares polynomial (or a negation of it, respectively). In other words, in order to find a Lyapunov function for some application, it is possible to construct it as a polynomial with free coefficients and define a SOS program which can find the coefficients such that some linear equality and/or inequality constraints are met. One important result of this is that we can fins a region in which the Lyapunov function is surely positive definite and its derivative is surely negative definite (or, smaller than some desired value), which, in turn, guarantees some desired behaviour for the system.
1.1. BACKGROUND

Due to this ability in estimation of regions of attraction, some research has been done to compute funnels [90] as (possibly) time-varying objects containing a region of the state space such that some invariance property holds. For instance, a funnel is defined in [90] as follows: Given an $n$-dimensional dynamic system of the form $\dot{x} = f(t,x)$, a set $\mathcal{F} \subset [t_0, t_f] \times \mathbb{R}^n$ is a funnel, if for each $(\tau, x_\tau) \in \mathcal{F}$, the solution $x(t)$ to $\dot{x} = f(t,x)$ initialized with $x(\tau) = x_\tau$ exists on $[t_0, t_f]$ and for each $t \in [t_0, t_f]$ we have $(t, x(t)) \in \mathcal{F}$.

A detailed explanation of computation of such funnels was given in the work of LQR-Trees [85, 88], where funnels were computed about nominal trajectories stabilized via TV-LQR control, ensuring that if the system is initialised ‘in a funnel’, the resulting motion will bring the state of the system to the outlet of this funnel, which is inside another funnel, and so on. Using the switching of control in the appropriated instances of transitions form one funnel to the next, a trajectory of the system flows like a stream of water inside funnels until arriving to the final basin of attraction about some locally stabilized equilibrium state. The main importance of the work in [88] is the clear presentation of the construction of a set of funnels which are connected to each other in a way that guarantees a desired behaviour of trajectories for a non-linear control system.

Many implementations of the LQR-Tree have recently appeared in the literature. We give here several examples of these implementations.

A gain-scheduled (GS) control is used in [39] to transition between local TV-LQR controllers. In this work, a GS-Bi-RRT is developed such that LQR control is applied to attract trajectories towards the obtained paths of trajectories and the algorithm is stopped once a near-connection is obtained. The goal state (which is the root of the backward tree) is an equilibrium about which a region of attraction is estimated and serves as an infinite-time-domain funnel. At each step of the algorithm, after adding a node to the forward tree, a tracking attempt is made to steer the state of the system from the new node towards the goal equilibrium via a reference trajectory passing from nearest node in the backward tree to the goal via a path connecting it to the goal along the backwards tree. The algorithm is stopped once the resulting trajectory terminates at the estimated region of attraction of the goal, thus, reducing the need for the two trees to be very closely connected.

The work reported in [43, 40, 44, 4] uses funnels for an on-line motion planning under
1.1. BACKGROUND

uncertainties in the environment, the model or state readings. The planning is done using a pre-computed library of stabilized trajectories and their respective funnels. Using the library plans and funnels, updates can be done to consider bounded disturbances. The safest plans are then executed. The approach is demonstrated by simulation of a UAV with parametric uncertainty flying among obstacles.

In [42], a large funnel was computed, to cover as much volume in state space as possible, that can be used to steer the state of a physical under-actuated and torque-limited double pendulum (Acrobot) to the upright stable equilibrium. The LQR-Trees approach was used in [70] to provide a large guaranteed region of attraction to an equilibrium for an under-actuated non-linear mechanical system with two degrees of freedom. It is also used in [52, 54, 18], where the motion of a perching UAV with 7 degrees of freedom is demonstrated. Based on the similar ideas as LQR-trees, [53] introduce adaptive controllers which are robust to parameter uncertainty and guarantee proper behaviour around nominal trajectories for three different under-actuated systems with uncertainties.

Additional similar work can be found in [5] (LQG-approach for motion planning). [3] use funnels (trajectory enclosures) to deal with uncertainties in parameters and initial conditions. [69] use an alternative for the LQR-Trees approach which is identical except for a replacement of the algebraic verification of funnels using SOS optimization with simulation-based approximations of funnels which results in an easier and faster implementation. A high-level control algorithm similar to LQR-Trees was implemented in [11] (with funnels called reach-tubes) to ensure that a robot performs various tasks in the configuration space. Humanoid push-recovery control was implemented in [91] using SOS programming.

In [25], a physical compass gait walker is designed, constructed and controlled in several different strategies. The first strategy uses virtual constraints and feedback linearisation. The second strategy uses an implementation of LQR-Tree algorithm while re-parametrisering the dynamics to the form of transversal dynamics state and transversal surface parameter. The re-parametrisation is done in order to properly address the impact with the ground while developing the LQR-Tree (by aligning the impact surface with a transversal surface). In addition, the transversal stabilisation allows to properly stabilise the periodic walking of the compass gait walker. Other related work is done in
Another important type of funnels is related to limit cycles and hybrid limit cycles (i.e., periodic orbits containing both smooth state flows as well as one or more impulsive state transitions). The construction of this type of funnels was presented mathematically in [46] and several examples for walking presented in [48]. Some related papers on this topic (including examples and motivation from a robotic bipedal walking perspective) are [73, 47, 74, 15, 49].

In [46], a technique is presented to compute the transverse dynamics of a system in the vicinity of a periodic orbit of some period $T > 0$ that it can exhibit. Computing the transverse dynamics is done by first choosing some mapping that maps the $n$-dimensional state $x$ of the dynamical system into its transverse dynamics decomposition $(\tau, x_\perp)$, where $\tau \in [0, T]$ represents a transverse section (a plain which is transversal to the periodic orbit such that the family of these plains for all $\tau \in [0, T]$ contains the periodic orbit) and $x_\perp$ is an $(n-1)$-dimensional state restricted to the transversal section $\tau$ and describes the position of the original state $x$ with respect to to the intersection point between the transversal section and the periodic orbit. In order to study the stability of the periodic orbit, it is possible to compute the behaviour of the transverse dynamics and obtain their linear approximation about the periodic orbit. Exponential stability of the transversal state $x_\perp$ translates to exponential orbital stability of the periodic orbit.

In order to calculate inner estimations of the region of attraction, or funnel, about the periodic orbit, it is possible to use SOS programming for the approximated transversal dynamics. In [46], it is mathematically shown how such funnel can be computed for both a continuous and hybrid periodic orbit (after ensuring that it is indeed exponentially stable) and in addition, a stabilizing controller is formulated in order to stabilize a particular periodic orbit which is not naturally stable.

1.2 Contributions and Thesis Structure

This thesis has two main contributions:

- A general “system of funnels” framework for global stabilization of nominal tra-
1.2. CONTRIBUTIONS AND THESIS STRUCTURE

Trajectories for general nonlinear systems; this framework covers all three hybrid techniques that we overview (throw and catch, manoeuvre-based motion planning and LQR trees) and, in addition, it can be suited to numerous other applications. Some examples of such applications can be: designing various periodic gaits for a robot (e.g., walking in different speeds and directions, running, etc...) and, in addition, appropriately coordinating switches between the different gaits (e.g., in order to transition from walking to running); generating well-synchronized repetitive behaviours between several mechanically-decoupled subsystems; performing complicated manoeuvres of air-crafts with a degree of robustness; and so on. Our results are rigorously proved and they are tailored to provide nominal robustness of the stability property.

- A modified RRT* algorithm that contains several modifications to the existing algorithm; the introduced modifications were necessary to deal with a range of problems considered in this thesis as the original algorithm was unable to compute the nominal trajectories.

As a result, we have obtained a practical method for the generation of nominal trajectories and rigorous, general and flexible framework for global stabilization of nominal trajectories via the system of funnels framework. In the rest of this section, we list our contributions in each chapter.

Chapter 2:
The term ‘funnel’ has been previously used to define such regions of attractivity about trajectories or distinct equilibria states. In this work, we provide a more general definition for a funnel, accompanied by general definitions for the entrance to a funnel and the outlet of a funnel. The separation between the funnel and its entrance plays an important role in ensuring the robustness of the hybrid controller to measurement noise and bounded disturbances, such that undesirable chattering or instability do not occur when the state of the hybrid system is near the boundary of a funnel. These definitions form the basis for our robust hybrid control framework.

In order to orchestrate the different funnel-related local control laws appropriately, we employ the field of hybrid systems to create a framework, called ‘system-of-funnels’ which is based on the execution of a hybrid control policy that ensures a stable and robust implementation of the overall desired motion, irrespective of the discontinuities in the nominal trajectory. This is done by executing an appropriate switching policy between local controllers which correspond to the different funnels in the system.
1.2. CONTRIBUTIONS AND THESIS STRUCTURE

Note that the system of funnels should be defined with respect to a directed graph which represents the desired interconnections between the different funnels. The vertices of the graph correspond to the different funnels in the system and its connecting edges are defined such that, roughly speaking, the trajectories exiting one funnel through its outlet will enter the entrance of the next funnel, in a manner that realizes the desired motion. In this manner, we can cascade several parts of the desired motion in a row with some robustness, regardless of the imperfect nominal trajectories. This is shown in the illustration of transition from forward to backward walking depicted in Fig. 1.6.

In the illustration in Fig. 1.6, we can see projections of eight funnels (onto the state space of a system) connected such that there are two possible periodic behaviours for the state of the system and a transition behaviour from one periodic orbit to another, which can occur once at most. The graph that suits this system-of-funnels is shown on the bottom. The drawn connections between the funnels intuitively illustrate that the outlet of one is in the entrance of the funnel to which it connects.\(^3\) There are two funnels which correspond to the impact with the walking surface and therefore they have a zero time domain (i.e., only jumps in the state of the dynamic system are allowed).\(^4\) Note that the nominal trajectories drawn may in practice have discontinuities.

In order to cater for situations when the trajectories of the hybrid system exit funnels, we present an assumption of a bootstrap controller which is responsible for the recovery of the system to a normal ‘in-funnel’ mode.

We also show via three theorems that the behaviour of the hybrid system approaches a continuous trajectory\(^5\) when the hybrid time\(^6\) goes to infinity.

Chapter 3:
The different segments of the planned trajectories can sometimes be globally tracked

\(^3\)Our framework allows much more elaborate connections which are not shown here for clarity.

\(^4\)We will not deal with this sort of jump dynamics in the presented framework, however, our framework can be naturally generalised to include such funnels that take into account jumps in the system’s dynamics.

\(^5\)This can be, for instance, a single state, a periodic orbit, or a chaotic motion constrained to a set in state space.

\(^6\)i.e., chronological time and number of jumps. An appropriate definition is given later on in Section 2.1.
under some assumptions on the system and the trajectories, however, in many applications, global tracking methods are not applicable. Instead, it is often possible to have a local tracking control that steers only nearby states towards the nominal trajectory. In addition, some recent techniques allow computation of regions about the nominal trajectories which satisfy some finite-time invariance properties under the local tracking control. These regions can be viewed as funnels with a finite time-domain in the state-time space of the dynamic system. Analogously, when a stabilizing control is used to asymptotically stabilise an equilibrium state, one can consider the resulting region of attraction (or its conservative estimate) as the projection of a funnel with an infinite time-domain onto the state space.

In chapter 3 we present - both analytically and pictorially - how several such funnel objects can be defined for various types of nominal trajectories in accordance with the system-of-funnels framework established in chapter 2. In order to make the text more self-contained, we include an explanation of how funnels can be numerically computed, however, this is not a contribution.

Chapter 4:
Often, it is required to plan the motion for systems characterized by non-holonomic constraints or under-actuation. In these cases, it is usually impossible to design trajectories which are continuous from the start state to the target state. Instead, the output of trajectory planners is mostly either discontinuous trajectories, or trajectories which do not exactly arrive at the target state. In most practical applications, it is sufficient to have the system performing a close trajectory provided some stability is retained.

The problem of imperfect (e.g., discontinuous) trajectories can be addressed in two approaches. One approach is focused on improving the trajectory planning algorithms themselves so that they become more efficient, accurate and suitable for systems with non-holonomic constraints or under-actuation. Another approach is focused on the feedback control of the resulting trajectories attempting to overcome the challenges of discontinuities.

In this thesis, we use both of these approaches consequently: first, we suggest better motion planning algorithms and then we use the resulting discontinuous trajectories to design feedback control which guarantees satisfactory robust behaviour of the dynamic system regardless of the imperfections in the nominal trajectory.

The existing trajectory planning algorithm which was found to be the most suitable for solving non-local problems (i.e., the start and target states are significantly apart from one another) for non-linear systems with non-holonomic constraints or under-actuation is based on the Rapidly-exploring Random Tree (RRT) algorithm. Although the most basic form of the RRT algorithm suits high order systems well due to its random exploration of the state space, it is not very applicable for systems with under-actuation.

We present several significant modifications to the RRT algorithm such that the modified algorithm is much more suitable for the above systems. In order to plan a nominal trajectory for a particular task, several desired states in the state space can be chosen initially, and then the motion planning algorithm can be applied between every two consecutive states to design nominal trajectories.

Chapter 5:
We present 3 illustrative examples to demonstrate the use of the suggested system-of-funnels framework: first, two mathematical examples to illustrate some of the concepts.
1.2. CONTRIBUTIONS AND THESIS STRUCTURE

Then, a more nature-inspired example to illustrate how the framework can be employed to design trajectories and globally control them.

Chapter 6:
This chapter concludes the thesis and suggests ideas for future work.
Global Stabilisation via System-of-Funnels Framework

In this chapter, we present the main contribution of this thesis, which is system-of-funnels hybrid control framework which can be used for robust and global stabilisation of desired behaviours for a dynamic system. The idea is that a desired behaviour is composed of one or several segments and that each segment can have some local dynamics and local control. These local behaviours give rise to a geometrical object termed funnel.

The structure of this chapter is as follows: We first present in Section 2.1 several general definitions for hybrid systems which are required as background on which we base the presentation of the rest of the chapter. In Section 2.2, we provide formal definitions for the basic building blocks of the framework: A funnel, An entrance, An outlet. In Section 2.3 we formally define the fundamental conditions for a set of funnels to be termed a system of funnels. This term reflects a connected structure of funnels which is designed in a manner that prevents from certain trajectories to ‘leak’ outside of the funnels in the system. We then show in Section 2.4 how a hybrid controller can be constructed for a given system of funnels and then in Section 2.5 we include a theorem that explains how this construction ensures that trajectories of the control system starting within entrances remain in the system of funnels.

Section 2.6 introduces a bootstrap control assumption which is used to make the system recover to its nominal behaviour (i.e., trajectories are inside funnels). This bootstrap control is incorporated into the hybrid control scheme. We explain in a second theorem the behaviour of trajectories in the overall hybrid system. A third theorem showing a
global stability property of the controlled system is presented in Section 2.7. Finally, we describe in Section 2.8 how our general system-of-funnels framework, when used in specific settings, can generate the same behaviours of trajectories as in two existing approaches in the literature, namely, LQR-Trees and Throw-and-Catch.

### 2.1 Hybrid Systems - Background Terms and Definitions

A hybrid system is often denoted by $\mathcal{H}$ using its data as vector entries $\mathcal{H} = (C, F, D, G)$.

In order to enable work with solutions of hybrid systems, it is useful to define the notion of hybrid time: A subset of $\mathbb{R}_{\geq 0} \times \mathbb{N}$ is called a hybrid time domain [20] if it is the union of infinitely many intervals of the form $[t_i, t_{i+1}] \times \{i\}$, $0 = t_0 \leq t_1 \leq t_2 \leq \ldots$, or of finitely many such intervals, with the last one possibly of the form $[t_i, t_{i+1}] \times \{i\}$, $[t_i, \infty) \times \{i\}$.

A hybrid system is initialised when $t = 0$ and $i = 0$. A hybrid time $(t, i)$ means: after $t$ time units and $i$ jumps from the initial state. A solution to a hybrid system is defined over a hybrid time domain. It is absolutely continuous\(^\dagger\) when $t$ belongs to the interior of intervals as above and it may be discontinuous on the boundaries of these intervals.

A hybrid arc is a function $x : \text{dom}(x) \rightarrow \mathbb{R}^n$, where $\text{dom}(x)$ is a hybrid time domain and, for each fixed $i$, $t \rightarrow x(t, i)$ is a locally absolutely continuous function on the interval $I_i = \{t \mid (t, i) \in \text{dom}(x)\}$. The hybrid arc $x$ is a solution to the hybrid system $\mathcal{H}$ if $x(0, 0) \in C \cup D$ and the following conditions are satisfied:

1. For each $i \in \mathbb{N}$ such that $I_i$ has a non-empty interior:

\[
\begin{align*}
\dot{x}(t, i) &\in F(x(t, i)) \text{ for almost all } t \in I_i \quad (2.1) \\
x(t, i) &\in C \text{ for all } t \in [\min I_i, \sup I_i]. \quad (2.2)
\end{align*}
\]

\(^\dagger\)Let $I \subset \mathbb{R}$ be a compact interval. A function $f : I \rightarrow \mathbb{R}$ is termed absolutely continuous on $I$ if, for any $\varepsilon > 0$ there exists $\delta > 0$ such that any sequence of pairwise disjoint sub-intervals $(x_k, y_k) \subset I$ satisfying $\sum_k (y_k - x_k) < \delta$ also satisfies $\sum_k |f(y_k) - f(x_k)| < \varepsilon$. 
2. For each \((t,i) \in \text{dom}(x)\) such that \((t,i+1) \in \text{dom}(x)\):

\[
x(t,i+1) \in G(x(t,i)),
\]

(2.3)

\[
x(t,i) \in D.
\]

(2.4)

Let \(\text{dom}(x) \subseteq \mathbb{R}_{\geq 0} \times \mathbb{N}\) be a hybrid time domain for a solution \(x : \text{dom}(x) \to \mathbb{R}^n\) to a hybrid system and let \((t,i) \in \text{dom}(x)\) be a parametrization of \(x\), that is, \(x(t,i)\) is the value of the hybrid state after \(t\) time units and \(i\) jumps.

A solution \(x\) to a hybrid system is termed maximal if it cannot be extended, that is, the hybrid system has no solution \(x'\) such that \(\text{dom}(x)\) is a proper truncation\(^2\) of \(\text{dom}(x')\) and \(x'(t,i) = x(t,i)\) for all \((t,i) \in \text{dom}(x)\). Denote by \(S_H(S)\) the set of all maximal solutions \(x\) for the system \(H\) that start from \(S\), that is \(x(0,0) \in S\). Complete solutions are maximal solutions for which the domain is unbounded.

We include here the definitions for outer semi-continuity and local boundedness from [20], which will be used later to define basic assumptions on the data of hybrid systems. For a hybrid system, satisfying the basic assumptions holds guarantees for the behaviour of solutions to the system.

A set-valued mapping \(F : \mathbb{R}^n \rightrightarrows \mathbb{R}^n\) is outer semi-continuous if its graph \(\{ (x,y) \mid x \in \mathbb{R}^n, y \in F(x) \}\) \(\subseteq \mathbb{R}^{2n}\) is closed.

The mapping \(G\) is locally bounded on a set \(C\) if, for each compact set \(K \subset C\), \(F(K)\) is bounded.

A continuous function \(f : C \to \mathbb{R}^n\), where \(C\) is closed, can be viewed as a set-valued mapping whose values on \(C\) consist of one point and are the empty set outside \(C\). Then, \(f\) is locally bounded on \(C\) and is outer semi-continuous.

The following assumptions on the data \((C,F,D,G)\) of the hybrid system are termed Basic Assumptions (also taken from [20]) and they are sufficient for obtaining desirable

\(^2\)This means that there exists some hybrid time \((t',i') \in \text{dom}(x')\), such that for all \(t + i < t' + i'\), if \((t,i) \in \text{dom}(x')\) then \((t,i) \in \text{dom}(x)\) and that for all \(t + i > t' + i'\), \((t,i) \not\in \text{dom}(x)\), but there exists some \((t,i) \in \text{dom}(x')\) such that \(t + i > t' + i'\).
properties for solutions to systems which satisfy them:

1. $C$ and $D$ are closed sets in $\mathbb{R}^n$.

2. $F : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ is an outer semi-continuous set-valued mapping, locally bounded on $C$, and such that $F(x)$ is non-empty and convex for each $x \in C$.

3. $G : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ is an outer semi-continuous set-valued mapping, locally bounded on $D$, and such that $G(x)$ is non-empty for each $x \in D$.

Since a solution can be defined only when it belongs to $C \cup D$, which may often not include the whole state space, it may happen that solutions cannot be extended beyond a certain hybrid time. It still makes sense to define notions of stability and asymptotic stability for hybrid systems without requiring that solutions are defined on an unbounded hybrid time domain. We define below some important notions related to the behaviour of solutions of hybrid systems: (The definitions are taken directly from [20] and [21].)

If a hybrid system satisfies the basic assumptions, then it is nominally well-posed (see Definition 6.2 and Theorem 6.8 in [21]).

**Stability:** a compact set $A$ is stable for the hybrid system $H$ if for each $\varepsilon > 0$ there exists $\delta > 0$ such that $|x(0,0)|_A \leq \delta$ implies $|x(t,i)|_A \leq \varepsilon$ for all solutions $x$ to $H$ and all $(t,i) \in \text{dom}(x)$. (The notion $|x|_A = \min\{|x - y| \mid y \in A\}$ indicates the distance of the vector $x$ to the set $A$.)

**Pre-attractivity:** A compact set $A$ is pre-attractive if there exists a neighbourhood of $A$ from which each solution is bounded and the complete solutions converge to $A$, that is, $|x(t,i)|_A \to 0$ as $t + i \to \infty$, where $(t,i) \in \text{dom}(x)$. Note that it is not required that maximal solutions are complete, but only bounded.

**Pre-asymptotic stability:** A compact set $A$ is pre-asymptotically stable is it is stable and pre-attractive. The basin of pre-attraction is defined as the set of all points in $\mathbb{R}^n$ from which each solution is bounded and the complete solutions converge to $A$. If the basin of pre-attraction is $\mathbb{R}^n$, then the set $A$ is globally pre-asymptotically stable.
2.2. DEFINITION 1: A FUNNEL

Pre-forward completeness: Given a set $S \subset \mathbb{R}^n$, a hybrid system $\mathcal{H}$ on $\mathbb{R}^n$ is pre-forward complete from $S$ if every $x \in \mathcal{S}_\mathcal{H}(S)$ is either bounded or complete.

$\omega$-limit set of a solution $x$: Let $x : \text{dom}(x) \to \mathbb{R}^n$ be a solution of a hybrid system $\mathcal{H}$. The $\omega$-limit set of $x$, denoted $\omega(x)$, is the set of all accumulation points of $x$, that is, it is the set of all points $x_\omega \in \mathbb{R}^n$ for which there exists a sequence $\{(t_p, i_p)\}_{p=1}^\infty \subseteq \text{dom}(x)$ with $\lim_{p \to \infty} t_p + i_p = \infty$ and $\lim_{p \to \infty} x(t_p, i_p) = x_\omega$.

$\omega$-limit set of a set $S$: Let $S \subset \mathbb{R}^n$ be a set. The $\omega$-limit set of $S$, denoted $\omega(S)$, is the set of all points $x_\omega \in \mathbb{R}^n$ for which there exists a sequence of maximal solutions to the hybrid system $\mathcal{H}$, $\{x_p\}_{p=1}^\infty \subset \mathcal{S}_\mathcal{H}(S)$, and a sequence $\{(t_p, i_p)\}_{p=1}^\infty$ such that for each $p$, $(t_p, i_p) \in \text{dom}(x_p)$, $\lim_{p \to \infty} t_p + i_p = \infty$ and $\lim_{p \to \infty} x_p(t_p, i_p) = x_\omega$.

Using the above terminology, we will later be able to present the system-of-funnels framework in the remaining of the chapter.

2.2 Definition 1: A Funnel

In this section, we generalize the definition of a funnel which appears in [90]. Our definition includes three basic objects which are used in the system-of-funnels framework: a funnel, an entrance and an outlet. The most important aspects of the generalisation are the separation of the funnel from its entrance (that is, each of these terms are defined separately, where as a funnel and an entrance are assumed the same object in [90]) and the generalisation of the outlet definition. The separate treatment of a funnel and its entrance allows a robust design of the hybrid control system and the generalised outlet definition allows more possibilities of switching from one funnel to another.

Consider the non-linear dynamic control system:

$$\dot{z} = f(z, u), \quad (2.5)$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is continuous.

Consider a set-valued mapping $F : \mathbb{R} \Rightarrow \mathbb{R}^n$ and define $\text{dom}(F) := \{\tau \mid F(\tau) \neq \emptyset\}$ as its domain.
Denote by $\mathcal{F}$ the graph of $F$, i.e., $\mathcal{F} := \text{gph}(F) := \{(\tau, z) \mid z \in F(\tau)\}$.

$\mathcal{F}$ (or $F$) is called a funnel if:

(a) $\text{dom}(F) = [0, \infty)$ or $\text{dom}(F) = [0, T]$ for some $T > 0$, and

(b) There exist a function $u : \mathcal{F} \to \mathbb{R}^m$ such that $\tau \to u(\tau, z)$ is piecewise continuous and $z \to u(\tau, z)$ is continuous, and a set $\mathcal{E} \subset \mathcal{F}$ such that for each $(\tau_0, z_0) \in \mathcal{E}$, each solution of (2.5) with $u = u(\tau, z)$ and $\dot{\tau} = 1$ starting at $(\tau_0, z_0)$ at time $t = 0$ does not exhibit finite escape time and satisfies $z(t) \in \text{int}(F(\tau(t)))$ for all $t$ where the solution is defined.\(^3\)

If $F$ is outer semi-continuous (osc), we have that $\mathcal{F}$ is a closed set and we say that the funnel is closed.

We refer to such set $\mathcal{E}$ as the entrance of the funnel $\mathcal{F}$, and to such function $u$ as the control policy associated with the funnel $\mathcal{F}$.

Let $\mathcal{O}$ be a closed subset of $\mathcal{F}$ and define a set-valued mapping $O : \mathcal{E} \rightrightarrows \text{dom}(F)$ such that for each $\xi_0 := (\tau_0, z_0) \in \mathcal{E}$, $O(\xi_0)$ is the set of all values of $\tau$ in $\text{dom}(F)$ for which a trajectory $\xi$ that starts from $\xi_0$ and flows in the funnel belongs to $\mathcal{O}$.

$\mathcal{O}$ is termed the outlet of a funnel $\mathcal{F}$ if it satisfies the following conditions:

(a) If $\text{dom}(F) = [0, \infty]$, then $\mathcal{O} := \emptyset$.

(b) If, otherwise, $\text{dom}(F) = [0, T]$, then, for each $\xi_0 \in \mathcal{E}$, at least one of the following two conditions is satisfied:

- $T \in O(\xi_0)$.
- $O(\xi_0)$ contains an open interval $I$ (a connected subset of $\text{dom}(F)$), such that $\inf I \geq \tau_0$.

Roughly speaking, an outlet is significant only for funnels with a finite temporal depth and must meet the condition that any trajectory $\xi$ that starts in the entrance of the funnel will eventually reach its outlet such that it will not be able to keep flowing in the set $\mathcal{F} \setminus \mathcal{O}$ either at all or for some non-zero amount of time.

\(^3\) $z(t)$ is defined for all $t$ such that $\tau(t) = \tau_0 + t \in \text{dom}(F)$. 

2.3 Definition 2: A System of Funnels

We define below a system of funnels with respect to a set of edges, which holds information about possible connections between funnels.

Define a set of funnel indices as $\mathbb{K} := \{1, ..., K\}$, where $K \in \mathbb{N} \setminus \{0\}$. Let $\Upsilon \subseteq \mathbb{K} \times \mathbb{K}$ be a set of directed edges. Let $\Sigma := \{\mathcal{F}_k\}_{k \in \mathbb{K}}$ be a set of $K$ funnels, such that each funnel $\mathcal{F}_k \in \Sigma$, $k \in \mathbb{K}$, is generated via the mapping $F_k : \mathbb{R} \rightarrow \mathbb{R}^n$, and let $\mathcal{E}_k$, $\mathcal{O}_k$ and $u_k$ be the entrance, the outlet and the control policy associated with $\mathcal{F}_k$, respectively.

A set of funnels $\Sigma$ is called a system of funnels relative to the set of edges $\Upsilon$, if for each $k \in \mathbb{K}$, the condition $(\tau, z) \in \mathcal{O}_k$ implies the existence of $j$ and $r$ such that $(k, j) \in \Upsilon$ and $(r, z) \in \mathcal{E}_j$.

2.3.1 Properties of a System of Funnels

Let $\Sigma$ be a system of funnels. For every $k \in \mathbb{K}$, define $J_k := \{j \mid (k, j) \in \Upsilon\}$, that is, the set of indices of funnels connected to $\mathcal{F}_k$ via outgoing edges from $\Upsilon$.

$\Sigma$ is said to be locally bounded if for each $k \in \mathbb{K}$ and for each compact set $S_k \subset \mathcal{O}_k$, the set $G_k^S := \{(\tau, r, z, j) \mid (\tau, z) \in S_k, (r, z) \in \mathcal{E}_j, j \in J_k\}$ is bounded.

$\Sigma$ is said to be closed if for each $k \in \mathbb{K}$ the set $G_k^O := \{(\tau, r, z, j) \mid (\tau, z) \in \mathcal{O}_k, (r, z) \in \mathcal{E}_j, j \in J_k\}$ is closed.

2.4 The Hybrid Controller Design

Consider a system of funnels, $\Sigma$, defined with respect to a set of edges $\Upsilon$. We define the following hybrid controller with respect to $\Sigma$.

2.4.1 Hybrid State Variable

We use $\xi = (\tau, z) \in \mathbb{R} \times \mathbb{R}^n$ as a hybrid state variable, where $\tau$ is a state of the dynamic, hybrid controller and can be viewed as the current temporal depth inside a funnel (whose
domain is the set of all admissible temporal depths), and \( z \) is the state of the dynamic system (2.5).

The aim of the controller is to ensure that \( \xi \) flows inside funnels or switches from one funnel to another once reaching outlets of funnels.

In addition, for a given system of funnels \( \Sigma \), we define the hybrid state \( x = (\xi, k) \in \mathbb{R}^{n+1} \times \mathbb{R} \), where \( k \) is the index of a funnel \( F_k \in \Sigma \).

### 2.4.2 Flow Set

First, we wish to define subsets of \( \mathbb{R}^{n+1} \) where \( \xi = (\tau, z) \) changes continuously, that is, flows. Flowing is allowed when the hybrid state \( x = (\tau, z, k) \) is such that \( \xi \in F_k \) and \( k \in K \). Therefore, for each \( k \in K \), we define \( \tilde{C}_k \subseteq \mathbb{R}^{n+1} \) as:

\[
\tilde{C}_k := \text{cl}(F_k \setminus O_k).
\]  

Finally, we define the set \( C \subseteq \mathbb{R}^{n+2} \) as:

\[
C := \bigcup_{k \in K} \tilde{C}_k \times \{k\}.
\]  

The set \( C \) contains the subset of \( \mathbb{R}^{n+2} \) where \( x = (\tau, z, k) \) should flow continuously, hence it is called the flow set.

### 2.4.3 Flow Map

When \( \xi \in \tilde{C}_k \), that is, when \( x \in C \), the hybrid trajectory’s dynamics evolve according to the flow map \( f_k : \mathcal{F}_k \rightarrow \mathbb{R}^{n+2} \):

\[
\dot{x} = (\dot{\tau}, \dot{z}, \dot{k}) = f_k(\xi),
\]  

where in general we have:

\[
f_k(\xi) := (1, f(z, u_k(\tau, z)), 0).
\]
However, in the special case (which is used later in Theorem 3) when

- the domain of the funnel is infinite (that is \( \text{dom}(F_k) = [0, \infty) \)), and
- both the funnel, \( F_k \), and the control policy associated with it, \( u_k \), are independent of \( \tau \) (that is, time invariant),

we use different dynamics for \( \tau \):

\[
f_k(\xi) := (-\tau, f(z, u_k(z)), 0).
\]  

The use of \( \dot{\tau} = -\tau \) in Eq. (2.10) is merely technical. It does not affect the dynamics of any other state variable but \( \tau \) and is chosen to prevent the linear unbounded growth of \( \tau \) which happens when \( \dot{\tau} - 1 \). The bounded dynamics of \( \tau \) in 2.10 assist us in proving a stability property later in Section 2.7.

### 2.4.4 Jump Set

We wish to enable a switch of the funnel index \( k \) and a reset of \( \tau \) when the state \((\tau, z, k)\) is such that \((\tau, z)\) has reached the outlet \( O_k \).

For each \( k \in \mathbb{K} \) we define a set \( \tilde{D}_k \subseteq \mathbb{R}^{n+1} \) as:

\[
\tilde{D}_k := O_k.
\]  

(2.11)

Note that \( \tilde{D}_k \) is closed when \( F_k \) is osc.

Also define the set \( D \subseteq \mathbb{R}^{n+2} \) as:

\[
D := \bigcup_{k \in \mathbb{K}} \tilde{D}_k \times \{k\}.
\]  

(2.12)

The set \( D \) is called the jump set as it indicates that the trajectory of the hybrid state \((\tau, z, k)\) is such that the values of \( k \) and \( \tau \) may jump.

### 2.4.5 Jump Map

For each \((k, j) \in \Upsilon\), we define a set-valued reset map \( R_{k,j} : \mathbb{R} \times \mathbb{R}^n \rightrightarrows \mathbb{R} \) which specifies the admissible new values for the temporal depth state after a jump from funnel \( F_k \) to
funnel $\mathcal{F}_j$ has occurred:

$$R_{k,j}(\tau, z) := \begin{cases} \{ r \in \mathbb{R}_{\geq 0} \mid (r, z) \in \mathcal{E}_j \}, & k \in \mathbb{K}, \ (\tau, z) \in \mathcal{O}_k, \ j \in J_k, \\ \emptyset, & \text{otherwise}. \end{cases} \quad (2.13)$$

The jump map $G_k : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^{n+2}$ for a given value of $k$ is then the union of all possible reset maps for $k$, augmented by the respective values of $z$ and $j$:

$$G_k(\tau, z) := \bigcup_{j \in J_k} (R_{k,j}(\tau, z), z, j). \quad (2.14)$$

Note that $(R_{k,j}(\tau, z), z, j)$ is empty when $R_{k,j}(\tau, z)$ is empty. Also note that $G_k(\xi)$ is not empty for $\xi \in \hat{D}_k$ due to the definition of a system of funnels, since we have that $\xi \in \hat{D}_k = \mathcal{O}_k$ implies the existence of $j$ and $r$ such that $j \in J_k$ and $(r, z) \in \mathcal{E}_j$.

If $\Sigma$ is closed, then we obtain that $\text{gph}(G_k)$ is closed and therefore $G_k$ is osc. Moreover, $G_k$ is locally bounded if $\Sigma$ is locally bounded.

### 2.4.6 The Hybrid Closed-Loop System

Our design of the hybrid control strategy leads to the following hybrid system:

$$\mathcal{H}_{cl} : \begin{cases} \dot{x} = f_k(\xi), & x \in C, \\ x^+ \in G_k(\xi), & x \in D. \end{cases} \quad (2.15)$$

Note that $f_k$ takes one of the forms (2.9) or (2.10), depending on whether or not the funnel $\mathcal{F}_k$ has the properties required for equation (2.10) to hold.

### 2.5 Nominal Behaviour of the Hybrid Controller

#### 2.5.1 Theorem 1

Consider a closed system of funnels $\Sigma$. If the hybrid system (2.15) is initialized at a point $x(0, 0) = (\tau_0, z_0, k_0) \in C \cup D$ such that $(\tau_0, z_0) \in \mathcal{E}_{k_0}$, then all of its maximal solutions $x$
are complete (i.e., \( \text{dom}(x) \) is unbounded) and satisfy \( z(t, i) \in \text{int}(F_{k(t,i)}(\tau(t, i))) \) for all \((t, i) \in \text{dom}(x)\).

### 2.5.2 Proof

1. First note that whenever \( k_0 \) is such that \( \text{length}(\text{dom}(F_{k_0})) = \infty \) and \((\tau_0, z_0) \in \mathcal{E}_{k_0} \subseteq \mathcal{F}_{k_0} \), since there are no finite escape times, all solutions \( z(t) \) to the dynamics \((2.5)\) with \( \dot{\tau} = 1 \) initiated at \( z(0) = z_0, \tau(0) = \tau_0 \) are defined over the unbounded time domain \([0, \infty)\). Note that, in this case, jumps are not possible since \( \tilde{D}_{k_0} = \mathcal{O}_{k_0} = \emptyset \) when \( \text{dom}(F_{k_0}) = [0, \infty) \). If the flow map is defined by equation \((2.9)\), then the dynamics of \( \tau(t) \) are defined by \( \dot{\tau} = 1 \) and hence \( \tau(t) = \tau_0 + t \) is defined over the infinite time domain \([0, \infty)\). If, alternatively, the flow map is defined by equation \((2.10)\), then the dynamics of \( \tau(t) \) are defined by \( \dot{\tau} = -\tau \) and hence \( \tau(t) = \tau_0 e^{-t} \) is defined over the infinite time domain \([0, \infty)\) as well. Note that in the latter case, applicability of equation \((2.10)\) implies that \( F_{k_0} \) and \( u_{k_0} \) are independent of \( \tau \), and therefore, all solutions \( z(t) \) to \((2.5)\) with \( z(0) = z_0 \) are independent of \( \tau \) and defined over \([0, \infty)\). Since there are no jumps, \( k_0 \) is not changed. Therefore, the hybrid time domain of solutions \( x(t, i) \) with \( x(0, 0) = (\tau_0, z_0, k_0) \) is \( \text{dom}(x) = [0, \infty) \times \{0\} \). In addition, according to property (b) in the definition of a funnel, we have that \( z(t, i) \in \text{int}(F_{k(t,i)}(\tau(t, i))) \) for all \((t, i) \in \text{dom}(x)\).

2. Next, note that whenever \( k_0 \) is such that \( T_{k_0} := \text{length}(\text{dom}(F_{k_0})) < \infty \), the solutions \( z(\tau) \) to the dynamics \((2.5)\) with \( \dot{\tau} = 1 \) and \( z(\tau_0) = z_0 \) are defined over \( \text{dom}(F_{k_0}) \setminus [0, \tau_0] = [\tau_0, T_{k_0}] \) and satisfy \( z(\tau) \in \text{int}(F(\tau)) \) for all \( \tau \in \text{dom}(F_{k_0}) \setminus [0, \tau_0] = [\tau_0, T_{k_0}] \) (by the definition of a funnel). Therefore, when \( \tau \in [\tau_0, T_{k_0}] \), we have that \( \xi \in \tilde{C}_{k_0} \) (see definition in Eq. \((2.6)\)) and hence \( k_0 \) remains unchanged and \( \dot{\tau} = 1 \) (by the definition of the flow map \((2.9)\)). Consequently, we eventually (when \( \tau = T_{k_0} \)) have \( \xi \in \mathcal{O}_{k_0} = \tilde{D}_{k_0} \) which implies that \( x \) is on the boundary of \( C \) and the vector field of \( \dot{x} \) points outside \( C \), hence, flow cannot continue\(^4\).

3. When \( \xi \in \tilde{D}_{k_0} \), we have \( \xi = (T_{k_0}, z) \in \mathcal{O}_{k_0} \). According to the definition of a system of funnels, the latter condition guarantees the existence of \( k^+ \) and \( \tau^+ \) such that \((k_0, k^+) \in \Upsilon \) and \((\tau^+, z) \in \mathcal{E}_{k^+} \). Therefore, \( G_{k_0}(\xi) \) is not empty. By the definition of the hybrid closed-loop system, each solution \( x \) will undergo a jump when \( x = \)

\(^4\)This is because when \( \xi = (\tau, z) \in \mathcal{O}_{k_0} \), we have that \( \tau \) has reached its maximal value inside the funnel \( \mathcal{F}_{k_0}, T_{k_0} \), and since the flow map imposes flow rate of \( \dot{\tau} = 1 \) for the temporal depth state, further flow would move the trajectory out of \( \text{gph}(F_{k_0}) = \tilde{C}_{k_0} \), thus, out of \( C \).
(T_{k_0}, z, k_0) ∈ D and will be reset to a new value \( x^+ := (\tau^+, z^+, k^+) \) according to the definition of the jump map (2.14). The jump map is defined such that \( \xi^+ := (\tau^+, z^+) \in E_{k^+} \subset F_{k^+} \). Hence, after a solution undergoes a jump, we have that \( \xi^+ \in \tilde{C}_{k^+} \) and \( x^+ \in C \).

4. After a jump occurs, the trajectory satisfies either steps 1 or 2 of the proof with a new \( k = k_0 \leftarrow k^+ \), \( \tau = \tau_0 \leftarrow \tau^+ \) and \( z = z_0 \leftarrow z^+ \) such that \((\tau_0, z_0) \in E_{k_0} \). Satisfying step 1 will always guarantee the result of the theorem and will finish the proof, whereas satisfying step 2 will lead to step 3, which will again result in either steps 1 or 2 to follow. If at any stage step 1 is taken (i.e., for some \((t_0, i_0) \in \text{dom}(x)\) we have \((\tau(t_0, i_0), z(t_0, i_0)) \in E_{k_0(t_0, i_0)} \) and \(\text{dom}(F_{k_0}) = [0, \infty)\)), the theorem holds (and \([t_0, \infty) \times \{i_0\} \subseteq \text{dom}(x)\)). If not, we will have steps 2 and 3 occurring repeatedly and indefinitely. At each recurrence, the trajectory will exist and stay for a finite time duration \(^5 T_{k_0} - \tau_0 \) inside \( \tilde{C}_{k_0} \), before the following jump occurs. Hence, the trajectory will not stop and will remain inside \( \Sigma \) indefinitely.

Finally, we conclude that all maximal solutions are complete.

### 2.6 Incorporating Bootstrap Control

#### 2.6.1 Recovery Policy

Let \( \Sigma \) be a system of funnels as defined in Section 2.3. Under the assumption that

\[
\xi(0, 0) \in E_{k(0, 0)}, \quad (2.16)
\]

Theorem 1 guarantees that all maximal solutions \( x = (\xi, k) \) to (2.15) are complete and satisfy \( \xi(t, i) \in F_{k(t, i)} \) for all \((t, i) \in \text{dom}(x)\).

In order to extend the result of Theorem 1, we wish to treat some cases when the assumption (2.16) does not hold. Consider the following two cases:

1. \( \xi(0, 0) \notin F_{k(0, 0)} \). In this case, neither flow nor jump are possible with system (2.15) and therefore solutions cannot be extended.

\(^5\)Note that for some solution \( x(t, i) \) this finite time duration can decrease to zero fast enough in the limit of \( i \to \infty \) so that \( t \) can be bounded, but in this case the solutions will still be complete as \( i \) grows unbounded.
2.6. INCORPORATING BOOTSTRAP CONTROL

2. $\xi(0,0) \in \mathcal{F}_{k(0,0)} \setminus \mathcal{E}_{k(0,0)}$ and at some $t' \in \left[0, \text{length(dom}(\mathbf{F}_{k(0,0)}))\right]$ we have that $z(t',0)$ intersects with the boundary of $\mathbf{F}_{k(0,0)}(t')$ and cannot flow or jump any more under the dynamics governed by (2.15).

In both of these two cases, solutions cannot be extended according to system (2.15). Therefore, we wish to allow the execution of a recovery policy once $x \in \text{cl}(\{x | \xi \not\in \mathcal{F}_k, \ k \in \mathbb{K}\})$ because this condition may imply that it is impossible for the solution to be extended under the dynamics of system (2.15). Note that for $k \in \mathbb{K}$, the set $\text{cl}(\{x | \xi \not\in \mathcal{F}_k, \ k \in \mathbb{K}\})$ includes both the case $\xi \not\in \mathcal{F}_k$ as in 1 and the case $\xi \in \partial \mathcal{F}_k$ (where $\partial$ stands for boundary) which is implied by the scenario that arises in 2.

The aim of the recovery policy is to ensure that:

- maximal solutions are complete, even if the initial condition does not satisfy (2.16); and that

- if the recovery policy is executed, then solutions will satisfy the condition $\xi(t,i) \in \mathcal{E}_{k(t,i)}$ for some finite $t \in \mathbb{R}_{\geq 0}$ and $i \in \{0, 1, 2\}$.

For each $k \in \mathbb{K}$, define the mapping $\mathbf{E}_k : \mathbb{R} \to \mathbb{R}^n$ such that $\mathcal{E}_k$ is its graph, i.e., $\mathcal{E}_k = \text{gph}(\mathbf{E}_k)$. Define the domain of each $\mathbf{E}_k$ as $\text{dom}(\mathbf{E}_k) := \{\tau | \mathbf{E}_k(\tau) \neq \emptyset\}$. We allow switching back to “in-funnel” control when $z$ is such that there exist $k' \in \mathbb{K}$ and $\tau' \in \text{dom}(\mathbf{E}_{k'})$ such that $z \in \mathbf{E}_{k'}(\tau')$.

Bootstrap Control Assumption:

We assume the existence of a continuous function $u_0 : \mathbb{R}^n \to \mathbb{R}^m$, with the property that every solution $z$ of (1) with $u = u_0(z)$ initiated at $z(0) = z_0 \in \mathbb{R}^n$ satisfies $z(T_0) \in \text{int}(\mathbf{E}_{k'}(\tau'))$ for some finite (trajectory dependent) $T_0 \in \mathbb{R}_{\geq 0}$, $k' \in \mathbb{K}$ and $\tau' \in \text{dom}(\mathbf{E}_{k'})$. We refer to such $u_0$ as a bootstrap control.
Remark: It may be possible in many cases to limit the set of entrances allowed for the bootstrap controller (to steer trajectories into) to only a subset of the entire set of entrances. For instance, we may replace $k' \in \mathcal{K}$ in the bootstrap assumption above with $k' \in \mathcal{K}'$, where $\mathcal{K}' \subset \mathcal{K}$. If the bootstrap control assumption is satisfied for $\mathcal{K}'$, then it is trivially satisfied for $\mathcal{K}$. This limitation can be helpful for some practical designs of systems of funnels, especially when it is difficult to continuously check whether the state is inside any entrance in the system but easy to check whether it is inside the subset of entrances with indices from $\mathcal{K}'$.

Although the bootstrap control assumption may look too restrictive, it should be noticed that in many cases a bootstrap controller can be easily constructed. Consider, for instance, some mechanical system with a non-negligible friction. Using a zero bootstrap controller, that is, not actuating the mechanical system altogether, can often bring it to a certain resting configuration. The design of the system of funnels can be made to contain a funnel (or a sequence of funnels) which is related to this naturally-stable equilibrium of the mechanical system. In other cases, we may have a different bootstrap control law that depends on the initial condition in the state space of the system such that all of the different bootstrap control laws together achieve the behaviour assumed here. We restrict ourselves to the analysis of the system of funnels with a single general bootstrap controller, bearing in mind that some generalisations can be made to the bootstrap control assumption.

We use a logic state variable $\ell \in \{0, 1\}$ to indicate whether or not the bootstrap control is being used. Note that $\ell = 0$ when the bootstrap is turned off.

The recovery policy is comprised of up to three stages (depending on the initial state) which are:

1. Once a condition $x \in \text{cl}(\{x \mid \xi \not\in \mathcal{F}_k, k \in \mathcal{K}\})$ (such as in cases 1 and 2 above) is detected while $\ell = 1$ (i.e., bootstrap control is turned off), the hybrid controller is allowed to switch $\ell$ to 0 without changing $x$ (which would correspond to one jump). In the case that the trajectory resulting from the dynamics $\dot{\xi} = (1, f(z, u_k(\xi)))$ or $\dot{\xi} = (-\tau, f(z, u_k(z)))$ with $\xi \in \mathcal{F}_k$ is forced outside of the flow set before reaching $\mathcal{O}_k$, then this jump would become mandatory.

2. While $\ell = 0$, the bootstrap control is applied continuously until the $z$ component of
2.6. INCORPORATING BOOTSTRAP CONTROL

the state reaches some value \( z' \) for which there exist \( \tau' \) and \( k' \) such that \( z' \in E_{k'}(\tau') \).

3. Once such \( \tau' \) and \( k' \) exist for some value of \( z' \) while \( \ell = 0 \), the hybrid controller sets \( \ell, \tau \) and \( k \) to 1, \( \tau' \), and \( k' \), respectively. This corresponds to another jump, after which the assumption of Theorem 1 is satisfied.

We define a set-valued mapping \( G_0 : \mathbb{R}^{n+1} \Rightarrow \mathbb{R}^{n+2} \) as:

\[
G_0(\xi) := \{(r, \zeta, j) \mid (r, \zeta) \in E_j, \ z = \zeta, \ j \in K\}, \tag{2.17}
\]

where \( \xi = (\tau, z) \).

We make the following two assumptions:

1. All \( E_k, k \in K \), are closed sets. Therefore, \( G_0 \) is osc.

2. For each compact set \( Z \in \mathbb{R}^n \), the set \( \{(r, z, j) \mid z \in Z, \ (r, z) \in E_j, \ j \in K\} \) is bounded. Therefore \( G_0 \) is locally bounded. This assumption can be relaxed by, for instance, introducing a morphism \( \hat{G}_0 : \mathbb{R}^{n+2} \rightarrow \mathbb{R}^{n+2} \) which assigns for each set \( G_0(\xi) \) a bounded set \( \hat{G}_0(G_0(\xi)) \subseteq G_0(\xi) \) and using \( \hat{G}_0(G_0(\xi)) \) instead of \( G_0(\xi) \).

We define three jump sets:

1. \( D_0 \), corresponding to the conditions required to switch \( u_0 \) off.

2. \( D_1 \), corresponding to the conditions required to switch \( u_0 \) on.

3. \( D_2 \), corresponding to the conditions when \( x = (\xi, k) \) is such that \( \xi \in O_k, \ k \in K \), while \( \ell = 1 \).

These sets are defined as follows:

\[
D_0 := \{x \mid G_0(\xi) \neq \emptyset\} \times \{0\}, \tag{2.18}
\]

\[
D_1 := \text{cl}((\{x \mid \xi \notin F_k, \ k \in K\}) \times \{1\}), \tag{2.19}
\]

\[
D_2 := \{x \mid \xi \in O_k, \ k \in K\} \times \{1\} = D \times \{1\}. \tag{2.20}
\]

We also define two flow sets:

\[\text{Note that the mapping } G_0 \text{ is independent of its } \tau \text{ argument. It appears as a dummy argument for a clearer presentation later on.}\]
2.6. INCORPORATING BOOTSTRAP CONTROL

1. $C_0$, corresponding to the region in which the trajectory flows with bootstrap control and $\ell = 0$.

2. $C_1$, corresponding to the region in which the trajectory flows “in funnels”, i.e., such that $\xi \in \mathcal{F}_k$, $k \in \mathbb{K}$ and $\ell = 1$.

These sets are defined as follows:

$$C_0 := \text{cl}(\{x \mid G_0(\xi) = \emptyset\}) \times \{0\}; \quad (2.21)$$

$$C_1 := \{x \mid \xi \in \mathcal{F}_k, k \in \mathbb{K}\} \times \{1\} = C \times \{1\}. \quad (2.22)$$

With these definitions above, we can now redefine the closed-loop hybrid control system as:

$$H_{cl} : \begin{cases} 
\begin{pmatrix} \dot{x} \\ \dot{\ell} \end{pmatrix} = \begin{pmatrix} f_{k\ell}(\xi) \\ 0 \end{pmatrix}, & (x, \ell) \in C_0 \cup C_1, \\
\begin{pmatrix} x^+ \\ \ell^+ \end{pmatrix} = \begin{pmatrix} x \\ 0 \end{pmatrix}, & (x, \ell) \in D_1, \\
\begin{pmatrix} x^+ \\ \ell^+ \end{pmatrix} \in \begin{pmatrix} G_{k\ell}(\xi) \\ 1 \end{pmatrix}, & (x, \ell) \in D_0 \cup D_2,
\end{cases} \quad (2.23)$$

where $k\ell$ (the index of $f_{k\ell}$ and $G_{k\ell}$) is the product of $k \in \mathbb{K}$ and $\ell \in \{0, 1\}$ and $f_0 := (1, f(z, u_0(\tau, z)), 0)$.

Note that when $\ell = 1$, $f_k$ takes one of the forms (2.9) or (2.10), depending on whether or not the funnel $\mathcal{F}_k$ has the properties required for equation (2.10) to hold.

Note also that there is a non-empty intersection between $D_1$ and $D_2$ (corresponding to $\ell = 1$ and $\xi \in \partial\mathcal{O}_k$), so that when $(x, \ell)$ belongs to this intersection, any of the jumps in the second and third lines of (2.23) can take place.
2.6. INCORPORATING BOOTSTRAP CONTROL

2.6.2 Theorem 2

Every solution \((x, \ell)\) to the hybrid system (2.23) starting at \((x_0, \ell_0) := (\tau_0, z_0, k_0, \ell_0) = (\tau_0, z_0, k_0, 0) \in \mathbb{R}^{n+1} \times K \times \{0, 1\}\) is complete, and for each such solution there exist \(t' \in \mathbb{R}_{\geq 0}\) and \(i' \in \{0, 1, 2\}\) such that \((x, \ell)\) is guaranteed to satisfy \((\tau(t, i), z(t, i)) \in F_{k(t, i)}\) with \(k(t, i) \in K\) and \(\ell = 1\) for all \((t, i) \in \{t, i\} \in \text{dom}(x)\) satisfying \(t \geq t'\) and \(i \geq i'\).

2.6.3 Proof

Since \(\ell_0 \in \{0, 1\}\), we demonstrate that the result of the theorem is obtained for each allowable value of \(\ell_0\) separately:

1. Assume that \(\ell_0 = 0\) (that is, bootstrap control is switched on). Note that \(\mathbb{R}^{n+1} \times K \times \{0\} \subseteq D_0 \cup C_0\). Note also that \(D_0 \cap C_0 = \partial D_0 = \partial C_0\), where \(\partial\) stands for boundary. Therefore, only the following three cases are possible for \((x_0, \ell_0)\):

   a. \((x_0, \ell_0) = (x_0, 0) \in \text{int}(D_0)\).
   b. \((x_0, \ell_0) = (x_0, 0) \in \text{int}(C_0)\).
   c. \((x_0, \ell_0) = (x_0, 0) \in D_0 \cap C_0 = \partial D_0 = \partial C_0\).

We describe in 1A, 1B and 1C (see below) the scenarios which will occur in each of the cases 1a, 1b and 1c, respectively, and show that all three lead to the result of the theorem for some values of \(t'\) and \(i'\).

A. This scenario corresponds to the situation when bootstrap control should be switched off and “in-funnel” control should be started. Notice that when \((x, \ell) \in \text{int}(D_0)\), flow is not possible since this condition implies \((x, \ell) \not\in C_0 \cup C_1\). As a result, \((x_0, \ell_0)\) will instantly undergo a jump to \((x_0, 1, \ell(0, 1)) \in (G_0(\xi), 1)\). Due to the definition of \(G_0(\xi)\) (which is not empty in this case, see Eq. (2.18)), \((x_0, 1)\) will satisfy \(\xi(0, 1) \in E_{k(0, 1)}\). Moreover, \(\ell(0, 1) = 1\) and the hybrid system (2.23) dictates a behaviour of \(\dot{\ell} = 0\) and \(\dot{x} = f_k\) (since \((x, l) \in C_1\) after the jump). As a result, Theorem 1 applies. Hence, in this case we get that after the jump, the behaviour of \(x\) in system (2.23) is identical to the behaviour of \(x\) in system (2.15) with the assumption of Theorem 1 being satisfied. Notice that since \(\xi\) will not reach the boundary of \(F_k\) after the jump (by application
2.6. INCORPORATING BOOTSTRAP CONTROL

of Theorem 1), the condition \((x, \ell) \in D_0\) cannot be met again, hence \(\ell\) will remain 1, that is, bootstrap control will not operate again. Therefore, we conclude that \(z(t, i) \in \text{int}(F_{k(t,i)}(\tau(t,i)))\) (and therefore \((\tau(t, i), z(t, i)) \in \mathcal{F}_{k(t,i)}\)) with \(k(t, i) \in \mathbb{K}\) for all \((t, i) \in \{(t, i) \in \text{dom}(x) \mid t \geq 0, i \geq 1\}\).

B. This scenario corresponds to the continuous operation of the bootstrap control. Notice that when \((x, \ell) \in \text{int}(C_0)\), jumps are not possible, since this condition implies \((x, \ell) \notin D_0 \cup D_1 \cup D_2\). As a result, \((x, \ell)\) will flow according to the flow map \((\dot{x}, \dot{\ell}) = (f_0, 0)\) for some finite amount of time (which depends on the trajectory), \(T_0\), until \(x\) will be such that \(x \in E_{k'}(\tau')\) for some \(k' \in \mathbb{K}\) and \(\tau' \in \text{dom}(E_{k'})\) and that \((x, \ell)\) cannot continue flowing in \(C_0\) any longer. Therefore \((x(T_0, 0), \ell(T_0, 0))\) will belong to \(D_0\). (Since \(\ell\) remains 0, \((x(T_0, 0), \ell(T_0, 0)) \notin C_1\). This will happen due to the assumption of the bootstrap controller. Consequently, a jump will occur as in scenario 1A but at time \(T_0\). Thus, similarly, we obtain that \((\tau(t, i), z(t, i)) \in \mathcal{F}_{k(t,i)}\) with \(k(t, i) \in \mathbb{K}\) for all \((t, i) \in \{(t, i) \in \text{dom}(x) \mid t \geq T_0, i \geq 1\}\).

C. This scenario allows occurrence of both scenarios described in 1A and 1B (i.e., flow and jump). The trajectory of \((x, \ell)\) can either keep flowing in \(C_0\) for up to some \(T_0 \in \mathbb{R}_{\geq 0}\) time units before it cannot keep flowing in \(C_0\) any more (due to the bootstrap controller assumption) and it necessarily enters \(D_0\) (when a jump as in scenario 1A occurs), or, it can experience a jump prior to \(T_0\), provided that \((x, \ell) \in D_0 \cap C_0\) at that time. With similarity to scenarios 1A and 1B above, we conclude that here, too, the result of the theorem is satisfied for some \(t' = T_0\) and \(i' = 1\), where \(T_0\) is the maximal amount of time that the bootstrap control can operate before the jump.

2. Now assume that \(\ell_0 = 1\) (that is, bootstrap control is switched off). Note that \(\mathbb{R}^{n+1} \times \mathbb{K} \times \{1\} \subseteq D_1 \cup C_1\). Note also that \(D_1 \cap C_1 = \partial D_1 = \partial C_1\). Therefore, again, only the following three cases are possible for \((x_0, \ell_0)\):

- a. \((x_0, \ell_0) = (x_0, 1) \in \text{int}(D_1)\).
- b. \((x_0, \ell_0) = (x_0, 1) \in \text{int}(C_1)\).
- c. \((x_0, \ell_0) = (x_0, 1) \in D_1 \cap C_1 = \partial D_1 = \partial C_1\).

We describe in 2A, 2B and 2C (see below) the scenarios which will occur in each of the cases 2a, 2b and 2c, respectively, and show that all three lead to the result
of the theorem for some values of \( t' \) and \( i' \).

A. This scenario corresponds to the situation when bootstrap control should be switched on since \( \xi \) is not in \( F_k \). Notice that when \( (x, \ell) \in \text{int}(D_1) \), flow is not possible since this condition implies \( (x, \ell) \not\in C_0 \cup C_1 \). As a result, \( (x, \ell) \) will instantly undergo a jump to \( (x(0,1), \ell(0,1)) \in D_0 \cup C_0 \) since \( \ell \) is switched to zero according to the jump map. Therefore, \( (x(0,1), \ell(0,1)) \) will behave according to the scenarios described in 1A to 1C. Hence, the result of the theorem will be guaranteed for \( t' = T_0 \) and \( i' = 2 \), where \( T_0 \) is the time it would take the bootstrap controller which operates after the first jump to bring \( z \) into \( E_{k'}(\tau') \) for some \( k' \in K \) and \( \tau' \in \text{dom}(E_{k'}) \) so that further flow of \( (x, \ell) \) in \( C_0 \) is not possible.

B. This scenario corresponds to the situation when “in-funnel” control is operating, i.e., \( \xi \) flows inside \( F_{k_0} \) with \( k_0 \in K \). Notice that when \( (x, \ell) \in \text{int}(C_1) \), jumps are not possible, since this condition implies \( (x, \ell) \not\in D_0 \cup D_1 \cup D_2 \). As a result, \( (x, \ell) \) flows in \( C_1 \) according to the flow map \((\dot{x}, \dot{\ell}) = (f_{k_0}, 0)\). This mandates that \( \tau \) grows with a rate of 1, unless the flow map \( f_{k_0} \) is given by (2.10), in which case, the dynamics of \( z \) are independent of \( \tau \). Consequently, three possibilities arise:

i. \( \xi \) remains indefinitely in \( F_{k_0} \) (assuming that \( \text{dom}(F_{k_0}) = [0, \infty) \)). This trivially guarantees the result of the theorem for \( t' = 0 \) and \( i' = 0 \).

ii. \( \xi \) escapes \( F_{k_0} \) (or \( \xi \in \partial F_{k_0} \)) when \( \tau = \tau_{D_1} < T_{k_0} := \text{length}(\text{dom}(F_{k_0})) \), where \( \tau_{D_1} \) is the value of \( \tau \) when \( (x, \ell) \) arrives at the boundary of \( D_1 \) right before exiting \( C_1 \). Therefore, we get \((x(t_1,0), \ell(t_1,0)) \in D_1 \), where \( t_1 := \tau_{D_1} - \tau_0 \), if the flow map (2.9) is valid, or \( t_1 := \ln(\tau_0/\tau_{D_1}) \), if the flow map (2.10) is valid. Since \( \tau_0 \) and \( \tau_{D_1} \) are finite, \( t_1 \) is finite. As a result, a jump will occur like in scenario 2A such that \((x(t_1,1), \ell(t_1,1)) \in D_0 \cup C_0 \). Therefore, the result of the theorem will be satisfied for \( t' = t_1 + T_0 \) and \( i' = 2 \), where \( T_0 \) is the time it would take the bootstrap controller which operates after the first jump to bring \( z \) into \( E_{k'}(\tau') \) for some \( k' \in K \) and \( \tau' \in \text{dom}(E_{k'}) \) so that further flow of \( (x, \ell) \) in \( C_0 \) is not possible.

iii. \( \xi \) arrives at \( O_{k_0} \) when \( \tau = T_{k_0} \). In this case, we obtain \((x(t_2,0), \ell(t_2,0)) \in D_2 \), where \( t_2 := T_{k_0} - \tau_0 \). As a result (since flow cannot be continued), \((x(t_2,0), \ell(t_2,0)) \) will undergo a jump. If \( \xi(t_2,0) \in \text{int}(O_{k_0}) \), then \( x \) will
jump according to the jump map $G_{k_0}$. Due to the definition of $G_k$ in (2.14), $x(t_2, 1)$ will satisfy $(\tau(t_2, 1), z(t_2, 1)) \in E_{k(t_2, 1)}$. Since $\ell$ remains 1 in the jump, the behaviour of $x$ in the system (2.23) for $t \geq t_2$ will be essentially the same as in system (2.15). Since Theorem 1 holds in this case (taking $t_2$ as the initial time), we can guarantee that $z(t, i) \in \text{int}(F_k(t, i))$ for all $(t, i) \in \{ (t, i) \in \text{dom}(x) \mid t \geq t_2, i \geq 1 \}$. This case corresponds to a “funnel transition” and guarantees the result of the theorem for $t' = t_2$ and $i' = 1$. If, however, $\xi(t_2, 0) \in \partial K_k$, then we have that $(x(t_2, 0), \ell(t_2, 0)) \in D_1 \cap D_2$. In this case, there would be a jump which is either of the “funnel-transition” type as before, or, alternatively, according to the jump map $(x, 0)$ which will trigger the bootstrap control. This will result in $(x(t_2, 1), \ell(t_2, 1)) \in D_0 \cup C_0$ and with similarity to scenario 2A we obtain the result of the theorem for $t' = t_2 + T_0$ and $i' = 2$, where $T_0$ is the time it would take the bootstrap controller which operates after the first jump to bring $z$ into $E_{k'}(\tau')$ for some $k' \in K$ and $\tau' \in \text{dom}(E_{k'})$ so that further flow of $(x, \ell)$ in $C_0$ is not possible.

C. This scenario refers to the case when $\xi$ flows on the boundary of some funnel and therefore the hybrid controller allows occurrence of both scenarios described in 2A and 2B (i.e., flow and jump). The trajectory of $(x, \ell)$ can keep flowing in $C_1$ either indefinitely (which will trivially guarantee the result of the theorem like in scenario 2(B)i for $t' = 0$ and $i' = 0$) or for a finite amount of time before it either hits $D_2$ (and a jump as in scenario 2(B)iii occurs), or, jumps as in scenario 2A, provided that $(x, \ell) \in D_1 \cap C_1$ at that time. With similarity to scenarios 2A and 2B above, we conclude that here, too, the result of the theorem is satisfied for some $t'$ and $i'$ according to the relevant case from 2A and 2B, as described above.

3. We showed in 1 and 2 above that the result of the theorem is satisfied for $(x_0, \ell_0) \in \mathbb{R}^{n+1} \times K \times \{0\}$ and $(x_0, \ell_0) \in \mathbb{R}^{n+1} \times K \times \{1\}$, respectively, by providing the values (or ranges) of $t'$ and $i'$ for which $x(t, i)$ is such that $(\tau(t, i), z(t, i)) \in F_{k(t, i)}$ and $\ell(t, i) = 1$ for all $t \geq t'$ and $i \geq i'$ for each of the sub-cases that can occur. Therefore we conclude that for all $x_0 \in \mathbb{R}^{n+1} \times K$ and all $\ell_0 \in \{0, 1\}$ the result of the theorem will hold.
2.7 The Limiting Behaviour of Solutions to the Hybrid System

The theorem developed in this section is useful for proving various uniformity and robustness properties.

2.7.1 Theorem 3

Consider the hybrid system (2.23) defined for a closed system of funnels \( \Sigma \). Assume that the following conditions apply for \( \Sigma \):

1. The index set \( K \) is finite, i.e., \( K < \infty \).

2. The entrance set for each funnel, \( E_k \) \( (k \in K) \), is compact.

3. For each \( k \in K \) such that the funnel \( F_k \) has an infinite temporal depth (i.e., \( \text{dom}(F_k) = [0, \infty) \)), we have that:
   (a) The domain \( \text{dom}(E_k) \) is of the form \([0, T_E]\), where \( T_E \in \mathbb{R}_{\geq 0} \).
   (b) The controller \( u_k \), the funnel \( F_k \) and its entrance \( E_k \) are time invariant, that is \( u_k = u_k(z) \) and the mappings \( F_k \) and \( E_k \) are constant on their domains.\(^7\)

   For simplicity, we define: \( F_k := F_k(\tau) \) for all \( \tau \in \text{dom}(F_k) \) and \( E_k := E_k(\tau) \) for all \( \tau \in \text{dom}(E_k) \).

   (c) For the system:
      \[ \dot{z} = f(z, u_k(z)), \quad z \in F_k, \]  
      (2.24)

   there are no finite escape times and, for each compact set \( S_k \subset F_k \) there exists a time \( T \geq 0 \) so that each solution starting in \( S_k \) either terminates within time \( T \) or else, reaches \( E_k \) within time \( T \).

4. The bootstrap control \( u_0(z) \) is such that every solution \( z \) to (2.5) with \( u = u_0(z) \) eventually reaches the set \( \mathcal{U} := \bigcup_{k \in K} \bigcup_{\tau \in \text{dom}(E_k)} \text{int}(E_k(\tau)) \) in finite time.\(^8\)

Then, there exists a compact set \( \mathcal{X} \subset \mathbb{R}^{n+1} \times K \times \{1\} \) such that the \( \omega \)-limit set of any compact set of initial conditions \( S \subset \mathbb{R}^{n+1} \times K \times \{1,2\} \) is a subset of \( \mathcal{X} \), that is, \( \omega(S) \subset \mathcal{X} \), and is therefore uniformly globally asymptotically stable (UGAS).

---

\(^7\)Note that in this case, equation (2.10) describes the flow map.

\(^8\)This assumption is equivalent to the original bootstrap assumption.
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

2.7.2 Proof

A brief general intuition:
Based on Theorem 2, we see that due to the bootstrap control, which can operate for a limited hybrid time duration, maximal trajectories are complete and are guaranteed to arrive and stay inside the system of funnels for all future time. The assumptions of Theorem 3 lead to the conclusion that complete trajectories always end up arriving at some entrance set of some funnel in the system at least once without the bootstrap control being used any more. Later we show that the infinite-horizon reachable sets from entrance sets of either finite- or infinite-temporal-depth funnels are compact. Since the number of funnels is finite, we arrive to the conclusion that the union of all of these reachable sets forms a compact set which contains the \( \omega \)-limit set of any compact set of initial conditions.

The outline of the proof is based on the following six lemmas, which are proved later:

**Lemma 1**
For each compact set \( S \subset \mathbb{R}^{n+1} \times \mathbb{K} \times \{1, 2\} \), there exists \( T_h \in \mathbb{R}_{\geq 0} \) such that for each solution \((x, \ell)\) of (2.23) starting in \( S \), there exists \((t', i')\) \( \in \text{dom}((x, \ell)) \) satisfying \( t' + i' \leq T_h \) and also:

1. \( \ell(t, i) = 1 \) for all \((t, i) \in \text{dom}((x, \ell))\) that satisfy \( t + i \geq t' + i' \),
2. \( \xi(t', i') \in \mathcal{E}_{k(t', i')} \);
3. For each jump of \( \xi \) at hybrid time \((t, i) \in \text{dom}((x, \ell))\) such that \( t + i \geq t' + i' \), immediately after the jump we have \( \xi(t, i + 1) \in \mathcal{E}_{k(t, i + 1)} \).

**Lemma 2**
The infinite-horizon reachable set \( \mathcal{X}_1 \) of all trajectories \((x, \ell)\) of (2.23) starting with the conditions \( \ell = 1 \) and \( \xi \in \mathcal{E}_k \), where \( k \) refers to any funnel \( \mathcal{F}_k \) with an infinite temporal depth, is compact.

**Lemma 3**
The infinite-horizon reachable set \( \mathcal{X}_2 \) of all trajectories \((x, \ell)\) of (2.23) starting with the conditions \( \ell = 1 \) and \( \xi \in \mathcal{E}_k \), where \( k \) refers to any funnel \( \mathcal{F}_k \) with a finite temporal...
depth, and such that \( k \) never takes a value that refers to a funnel with an infinite temporal depth, is compact.

**Lemma 4**

\[ X := X_1 \cup X_2 \] is the infinite horizon reachable set of all trajectories \((x, \ell)\) of (2.23) starting with the conditions \( \ell = 1 \) and \( \xi \in \mathcal{E}_k \), where \( k \in \mathbb{K} \) and it is compact.

**Lemma 5**

Since each trajectory \( \xi \) arrives within some finite hybrid time, which is uniformly bounded by \( T_h \) from any compact set of initial conditions of \((x, \ell)\), \( S \subset \mathbb{R}^{n+1} \times \mathbb{K} \times \{1, 2\} \), at some entrance \( \mathcal{E}_k \) with \( k \in \mathbb{K} \) and with \( \ell = 1 \) at that time, then we have that for each such compact set of initial conditions \( S \), the \( \omega \)-limit set of the set \( S \), \( \omega(S) \), is a subset of \( X \): \( \omega(S) \subseteq X \). Note that \( X \) is independent of \( S \).

**Lemma 6**

Since we can arbitrarily choose a compact set \( S_c \subset \mathbb{R}^{n+3} \), we can always choose it to be large enough such that its interior contains the compact set \( X \), that is, \( \omega(S_c) \subseteq X \subset \text{int}(S_c) \). According to Corollary 7.7 in [21], this guarantees UGAS of \( \omega(S_c) \). Note that for any initial condition in \( S_c \) that is not in \( \mathbb{R}^{n+1} \times \mathbb{K} \times \{1, 2\} \) there are no solutions.

**Proof of Lemma 1**

This item shows that trajectories of \( \xi \) arrive at entrances of funnels such that the future behaviour of \((\xi, k)\) is dictated by the hybrid system (2.15) so that Theorem 1 is applicable, while \( \ell \) is set to 1.

Below we discuss all the possible qualitative behaviours of trajectories starting from the set \( S \) and show that for all of them items 1(a) to 1(c) of the proof are satisfied.

A. If \( \ell(0, 0) = 0 \), then the bootstrap control operates, according to the hybrid system \( H_{cl} \) in (2.23). Assumption 4 of Theorem 3 implies that the set \( U \) is globally recurrent (see definition in [81]) for the bootstrap dynamics, i.e., equation (2.5) with \( u = u_0(z) \), since the set \( U \) is reached in finite time from every initial condition \( z_0 \in \mathbb{R}^n \). Note that the set \( U \) is open (since it is the interior of a compact set). In addition, we have that Standing Assumption 1 in [81] holds, since we consider here the continuous
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

system $\dot{z} = f(z, u_0(z))$ defined for all $z \in \mathbb{R}^n$ and $\mathbb{R}^n$ is a closed set. As a result, it follows from Proposition 1 in [81] that $\mathcal{U}$ is uniformly globally recurrent for the bootstrap dynamics considered. This means that for each compact set $S_z \subset \mathbb{R}^n$, there exists $T_0 \in \mathbb{R}_{\geq 0}$ such that for every solution $z(t)$ of $\dot{z} = f(z, u_0(z))$ that starts in $S_z$, there exists time $t', 0 \leq t' \leq T_0$, such that $z(t') \in \mathcal{U}$. Therefore, the hybrid system in (2.23) ensures that solutions $(x(t), \ell(t, i))$ starting with $\ell(0, 0) = 0$ from a compact set $S \subset \mathbb{R} \times S_z \times \mathbb{K} \times \{0, 1\}$ arrive at the jump set $D_0$ within up to $T_0$ time units, after which further flow in $C_0$ is not possible (due to the definition of the flow set $C_0$ and the set $\mathcal{U}$). Consequently, $(x, \ell)$ will undergo a jump (according to the jump map $G_0$) such that $\xi(t', 1) \in \mathcal{E}_{k(t', 1)}$ and $\ell(t', 1) = 1$, where $t' \in [0, T_0]$ is the time instant when the jump occurs. After the jump, $x$ behaves according to dynamics of system (2.15) with the condition of Theorem 1 being satisfied at the hybrid time $(t', 1)$. Therefore, by application of Theorem 1, for all $(t, i) \in \text{dom}((x, \ell))$ satisfying $t + i \geq t' + 1$ we have that $\xi(t, i)$ will not reach the boundary of $\mathcal{F}_{k(t, i)}$, which ensures that $(x(t, i), \ell(t, i))$ will not reach $D_1$, thus $\ell(t, i)$ will remain 1. From the definition of the jump map of $G_k$ in (2.15), we also get that any consequent jump due to $(x(t, i), \ell(t, i)) \in D_2$ for some $(t, i) \in \text{dom}((x, \ell))$ satisfying $t + i \geq t' + 1$ will result in $\xi(t, i + 1) \in \mathcal{E}_{k(t, i+1)}$. If the set $S$ contains states with $\ell = 0$, define $T_A := T_0 + 1$ (where $T_0$ is the upper bound of $t'$ as described above, which depends on the choice of $S_z$). Otherwise, let $T_A := 0$. Obviously, $T_A$ is finite. For all trajectories of $(x, \ell)$ starting in $S$ with $\ell(0, 0) = 0$, we showed here that statement 1 of the proof is satisfied for $T_h = T_A$.

B. If $\ell(0, 0) = 1$ and $\xi(0, 0) \notin \mathcal{F}_{k(0, 0)}$, then, according to the jump map in (2.23), a jump will take place immediately and $\ell(0, 1)$ will be set to 0. With similarity to the arguments of item A above, we infer that bootstrap control will be invoked and that for each solution $(x, \ell)$ of (2.23) starting inside the compact set $S$, there exists a time $t' \in [0, T_0]$ such that $z(t') \in \mathcal{U}$ ($T_0$ is defined as in item A above). Therefore, a second jump will take place, ensuring that $\xi(t'+2) \in \mathcal{E}_{k(t'+2)}$ and that $\ell(t'+2) = 1$. If the set $S$ contains states with $\ell = 1$ and $\xi \notin \mathcal{F}_k$, define $T_B := T_0 + 2$. Otherwise, let $T_B := 0$. For all trajectories of $(x, \ell)$ starting in $S$ with $\ell(0, 0) = 1$ and $\xi(0, 0) \notin \mathcal{F}_{k(0, 0)}$, we showed here that statement 1 of the proof is satisfied for $T_h = T_B$.

C. Define $\mathbb{K}^\infty := \{k \in \mathbb{K} \mid \text{dom}(\mathcal{F}_k(\tau)) \text{ is infinite}\}$ and define $\mathbb{K}^T := \mathbb{K} \setminus \mathbb{K}^\infty$. If $\ell(0, 0) = 1$, $\xi(0, 0) \in \mathcal{F}_{k(0, 0)}$ and $k(0, 0) \in \mathbb{K}^T$ (that is, $\xi(0, 0)$ is inside a funnel with a finite time domain) then $\xi$ cannot flow inside $\mathcal{F}_k$ for more than $t_1 := T_k - \tau(0, 0)$.
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

time units (recall that $T_k = \text{length}(\text{dom}(\mathbf{F}_k))$). Within this time, which is uniformly
bounded for all funnels by the finite time $\max_{k \in \mathbb{K}^T} T_k$ since $K$ is finite, $\xi$ either reaches
the outlet $O_{k(0,0)}$ or exits $\mathcal{F}_{k(0,0)}$ through its boundary. If $\xi$ reaches the boundary
of $\mathbf{F}_k$ at some time instant $t_2 \in [0, t_1]$, then a jump may occur (since $(x, \ell) \in D_1$ in
this case) such that $x(t_2, 1) = x(t_2, 0)$ and $\ell(t_2, 1) = 0$, according to the dynamics of
system (2.23). Note that if $\xi(t_2, 0)$ is such that it cannot keep flowing inside $\mathbf{F}_{k(0,0)}$, then
this jump is mandatory. Consider a hybrid system $\mathcal{H}$ with state $x = (\tau, z, k)$, flow set
$\bigcup_{k \in \mathbb{K}^T} \mathcal{F}_k \times \{k\}$, flow map $\dot{x} = (1, f(z, u_k(\xi)), 0)$, and an empty jump set.
This system is pre-forward complete (see definition 6.12 in [21]) due to the lack of
finite escape times and the finite duration of the flow of $\tau$ (each $\mathcal{F}_k$ with $k \in \mathbb{K}^T$ is
bounded in the $\tau$ direction), and therefore also $x$ (according to the definition of a
funnel), until solutions stop as a result of exiting the flow set, that is, all maximal
solutions of $x$ for this system are bounded. Consider any compact set $S_z \subset \mathbb{R}^n$ and define
$S := [0, \max_{k \in \mathbb{K}^T} T_k] \times S_z \times \mathbb{K}^T$. Note that $S_z$ is compact. In order to
show that the reachable set in finite time form $S_x$ for the system $\mathcal{H}$ is confined to a
compact set, we wish to use Lemma 6.16(b) of [21]. However, Lemma 6.16 requires
$\mathcal{H}$ to be nominally well-posed (see Definition 6.2 in [21]). For a hybrid system to
be nominally well-posed it is sufficient (according to Theorem 6.8 in [21]) that its
data satisfies the Basic Assumptions. These assumptions are given in Assumption
6.5 in [21] and in Section 2.1 above, and in particular, they require, among other
conditions, that the flow map is outer semi-continuous. Since the flow map of $\mathcal{H}$ is
only piecewise continuous, it is not (necessarily) outer semi-continuous. In order to
resolve this issue, we can use a Krasovskii regularization (see Definition 4.13 in [21])
of $\mathcal{H}$, $\hat{\mathcal{H}}$, for which the flow and jump sets are unchanged (assuming that all funnels
are closed) and the flow map is $(1, \hat{\mathcal{f}}, 0)$, where $\hat{\mathcal{f}}$ is a set-valued mapping with the
property that for each discontinuity point $\tau_d$ of the mapping $\tau \rightarrow f(z, u_k(\tau, z))$, $\hat{\mathcal{f}}$
takes all the values in the segment $[\lim_{\tau \searrow \tau_d} f, \lim_{\tau \nearrow \tau_d} f]$ (for each value of $z$), and for
all $\tau$ such that $f$ is continuous, $\hat{\mathcal{f}}$ is identical to $f$. The Krasovskii regularization $\hat{\mathcal{H}}$
does satisfy the basic conditions since the flow map is locally bounded. Due to the
absence of finite escape times for $\mathcal{H}$ and the fact that $f$ is piecewise continuous in $\tau$, and
the boundedness of the flow set in the $\tau$ direction, we get that all solutions of
$\hat{\mathcal{H}}$ are also bounded and therefore $\hat{\mathcal{H}}$ is pre-forward complete. According to Lemma
6.16(b) in [21], the reachable set in time $\max_{k \in \mathbb{K}^T} T_k$ of solutions $x(t)$ to $\hat{\mathcal{H}}$ starting
from $S_x$ is compact. This reachable set contains, in particular, all the states $x$ for
which $z(0) \in S_z$ and $\xi \in \partial \mathcal{F}_k$ (for some time instant $t_2$). Denote by $\mathcal{R}(S_z)$ the union

93
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

of all the $z$ values of these states. $\mathcal{R}(S_z)$ is compact. As a result from the bootstrap control assumption, since $\mathcal{R}(S_z)$ is compact, there exists a time $T_0 \in \mathbb{R}_{\geq 0}$ such that for each solution $z(t)$ of the bootstrap dynamics $\dot{z} = f(z, u_0(z))$ there exists $t' \leq T_0$ such that $z(t') \in \mathcal{U}$. Then, with similarity to the arguments in item A above, we arrive at the conclusion that all trajectories $(x, \ell)$ of the system (2.23) starting in $S \cap \mathbb{R} \times S_z \times \mathbb{K}^T \times \{1\}$ with $\xi$ arriving at $\partial \mathcal{F}_k$ such that a jump takes place at some time instant $t_2 \in [0, t_1]$ so that $\ell(t_2, 1) = 0$, we have that after some additional time $t' \in [0, T_0]$, another jump will happen, and this time we will have that $\ell(t_2 + t', 2) = 1$ as well as $\xi(t_2 + t', 2) \in \mathcal{E}_{k(t_2 + t', 2)}$ (according to the jump map $G_0$). After the second jump, $\ell$ remains 1 due to the same reasoning as in item A above. If $\xi$ does not exit $\mathcal{F}_{k(0,0)}$, it arrives into its outlet, $\mathcal{O}_{k(0,0)}$, and a jump occurs such that $\xi(t_1, 1) \in \mathcal{E}_{k(t_1,1)}$ and $\ell(t_1, 1) = 1$ (according to the dynamics of the hybrid system (2.23) for the case of $(x, \ell) \in D_2$). In both of the two described scenarios, we eventually get that for each trajectory $(x, \ell)$ of (2.23) starting from $S$ such that $z \in S_z, \xi \in \mathcal{F}_k$, $k \in \mathbb{K}^T$ and $\ell = 1$, there exists a hybrid time $(t', i')$ such that $t' + i' \leq \max_{k \in \mathbb{K}^T} T_k + T_0 + 2$ and for all $(t, i) \in \text{dom}((x, \ell))$ such that $t + i \geq t' + i'$ we have that $(x, \ell)$ satisfies $\xi \in \mathcal{E}_k$ with $\ell = 1$. If such states exist in $S$, then define $T_C := \max_{k \in \mathbb{K}^T} T_k + T_0 + 2$. If not, let $T_C := 0$. We have shown that for all such trajectories, statement 1 of the proof is satisfied for $T_h = T_C$.

D. If $\ell(0,0) = 1$, $\xi(0,0) \in \mathcal{F}_{k(0,0)}$ and $k(0,0) \in \mathbb{K}^\infty$, then using similar arguments as in item C above, and using assumptions 1 and 3(c) of Theorem 3, we obtain that for the given compact set $S$, there exists a uniform bound $T \in \mathbb{R}_{\geq 0}$ (which is the maximum of all bounds from the relevant funnels (with an infinite time domain)) that contain $\xi(0,0)$ while $\ell(0,0) = 1$ such that for each trajectory of $(x, \ell)$ (with the described initial conditions), there exists a time $t_1 \leq T$ such that $\xi(t_1, 0)$ either arrives at $\mathcal{E}_{k(0,0)}$, or exit the funnel $\mathcal{F}_{k(0,0)}$. If the $\xi$ trajectory leaves the funnel at $t_1$, then using the similar arguments as in items A and B above, we get that there exists another uniform bound $T_0$ such that for each trajectory $(x, \ell) \in S$ for which a jump from $D_1$ to $C_0 \cup D_0$ occurred at $t_1$, there exists a time $t_2, 0 \leq t_2 \leq T_0$, so that $\xi(t_1 + t_2, 2) \in \mathcal{E}_{k(t_1 + t_2, 2)}$ and $\ell(t_1 + t_2, 2) = 1$. In both of these scenarios, we obtain that there exists $(t', i') \in \text{dom}((x, \ell))$ such that $t' + i' \leq T_0 + T + 2$ and for all $(t, i) \in \text{dom}((x, \ell))$ with $t + i \geq t' + i'$, the properties 1(a)-1(c) of statement 1 of the proof are satisfied. Again, we define $T_D := T_0 + T + 2$ if the set $S$ contains states $(x, \ell)$ such that $\ell = 1$, $\xi \in \mathcal{F}_k$ and $k \in \mathbb{K}^\infty$, or otherwise, we set $T_D := 0$. For the
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

considered states in S, statement 1 of the proof is satisfied for $T_h = T_D$.

E. Now, define $T_h = \max(T_A, T_B, T_C, T_D)$. Note that for a given compact set of initial conditions, $S$, $T_h$ is finite due to the explanations of items A-D above. Steps A-D above discuss all possible qualitative behaviours of trajectories $(x, \ell)$ starting in $S$ and behaving according to the dynamics (2.23). In all of these cases, we have that there exists $(t', i') \in \text{dom}((x, \ell))$ so that $t' + i' \leq T_h$ such that the properties 1(a)-1(c) of statement 1 of the proof are satisfied for all $(t, i) \in \text{dom}((x, \ell))$ with $t + i \geq t' + i'$.

Proof of Lemma 2

Trajectories of $\xi$ starting with $(\tau(0, 0), z(0, 0)) \in \mathcal{E}_{k(0,0)}$, $\ell(0,0) = 1$ and $k \in K^\infty$ remain inside $\text{int}(F_k(\tau))$ and there would be no jumps. Due to the exponentially stable dynamics of $\tau$, $\dot{\tau} = -\tau$, we get that $\tau$ decays to 0 from any initial condition $\tau_0 := \tau(0, 0)$, and the infinite-horizon reachable set of $\tau$ is is $[0, \tau_0] \subseteq [0, T_\infty]$.

There are no finite escape times of $z$ and according to assumption 3(c) of Theorem 3, there exists a finite time $T \in \mathbb{R}_{\geq 0}$ by which each trajectory $z$ which starts in the compact set $E_k$ returns to $E_k$.

Define by $Z^T_k$ the reachable set of $z$ by time $T$ from $E_k$, that is:

$$Z^T_k := \{z(t) \mid z \text{ is a solution to } \dot{z} = f(z, u_k(z)), \; z(0) \in E_k, \; t \leq T\}.$$  

$Z^T_k$ is compact according to Lemma 6.16 (b) in [21] (Note that by the definition of a funnel and the assumptions of Theorem 3, we have that every $z$ trajectory that starts from $E_k$ is complete and therefore $E_k$ is pre-forward complete).

Then, every trajectory $z(t)$ that starts in $E_k$ will remain in $Z^T_k$ for all future time (i.e., as $t \to \infty$). In order to see this point, define a sequence of times $0 \leq t_0 \leq t_1 \leq t_2 \leq \ldots$ with $\lim_{j \to \infty} t_j = \infty$, such that $t_{j+1} - t_j \leq T$ and $z(t_j, i) \in E_k$ for all $j \in \mathbb{N}$. This is possible according to assumption 3.(c). Then, for each time segment $[t_j, t_{j+1}]$, we have that $z(i) \in Z^T_k$ by the definition of $Z^T_k$.

Since there are no jumps and according to the flow map $C_1$, clearly, $k$ and $\ell$ remain constant. Therefore, we conclude that the infinite-horizon reachable set of trajectories $(x, \ell)$ starting such that $\xi \in \mathcal{E}_k$ and $\ell = 1$ is $[0, T_\infty] \times Z^T_k \times \{k\} \times \{1\}$.
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

Define $X_1 := \bigcup_{k \in K} [0, T_k] \times Z^T \times \{k\} \times \{1\}$. All trajectories $(x, \ell)$ initiated with $\xi(0,0) \in \mathcal{E}_{k(0,0)}$, $k(0,0) \in K^\infty$ and $\ell(0,0) = 1$ will never leave the set $X_1$.

Proof of Lemma 3

Consider trajectories of $(x, \ell)$ starting with $\xi(0,0) \in \mathcal{E}_{k(0,0)}$, $k(0,0) \in K^\infty$ and with $\ell(0,0) = 1$.

By the definition of a funnel with a finite time domain, such trajectories are guaranteed to reach the set $O_k \times \{k\} \times \{1\}$ in finite time, which is less than or equal to $T_k$, the length of the funnel’s time domain.

The definition of a system of funnels ensures that for each $z$ such that $(T_k, z) \in O_k$, there exists $j \in J_k$ and $r \in \mathbb{R}_{\geq 0}$ so that $(r, z) \in \mathcal{E}_j$. Recall that the jump map $G_k$ ensures that each state $\xi(t,i) \in O_k$ of the considered trajectories will jump to some entrance $\mathcal{E}_j$, where $j \in J_k$. Since $\bigcup_{j \in K} \mathcal{E}_j$ is compact (each $\mathcal{E}_j$ is compact and $K < \infty$), it follows that each $O_k$ is compact.

In order to show confinement to a compact set of the reachable set of the considered trajectories, we proceed in a similar way as in item B of the Explanation of item 1 in the proof by considering a Krasovskii regularization of the dynamics in a finite-time-domain funnel.

Let $\widehat{f}_k$ be the Krasovskii regularization of the mapping $(\tau, z) \to f(z, u_k(\tau, z))$. $^9$ We define by $\mathcal{R}^{T_k}_k$ the reachable set from $\mathcal{E}_k$ in time $T_k$ of the system:

$$\dot{\tau} = 1,$$
$$\dot{z} \in \widehat{f}_k,$$
$$(\tau, z) \in \mathcal{F}_k.$$

Due to the absence of finite escape times (which follows from the behaviour without the regularization as described above, and the fact that $u_k$ is piecewise continuous in $\tau$), the fact that $\mathcal{F}_k$ is bounded in the $\tau$ direction, and the fact that $\bigcup_{j \in K} \mathcal{E}_j$ is compact, we get that $\mathcal{E}_k$ is pre-forward complete for this system (since each maximal trajectory of $(\tau, z)$ starting from $\mathcal{E}_k$ is bounded). As a result, according to Lemma 6.16 (b) in [21],

$^9$Since $\tau \to u_k$ is piecewise continuous, so is $\tau \to f$ and hence $f$ needs to be regularized.
we obtain that $\mathcal{R}_k^{T_k}$ is compact.
Effectively, $\mathcal{R}_k^{T_k}$ defines the set of all trajectories $\xi$ which are initiated inside $\mathcal{E}_k$ with $k \in \mathbb{K}^T$ until they reach the outlet $O_k$.

Define $\mathcal{X}_2 := \bigcup_{k \in \mathbb{K}^T} \mathcal{R}_k^{T_k} \times \{k\} \times \{1\}$. since $K < \infty$, we have that $\mathcal{X}_2$ is compact.
We can conclude that each trajectory of $(x(t,i),\ell(t,i))$ with $\ell(0,0) = 1$ and $\xi(0,0) \in \mathcal{E}_{k(0,0)}$ is complete and remains inside $\mathcal{X}_1 \cup \mathcal{X}_2$ for all $t$ and $i$ in its domain. If $k(0,0) \in \mathbb{K}^\infty$, trajectories are confined to $\mathcal{X}_1$, as was shown in the Explanation of item 2 of the proof. If $k(0,0) \in \mathbb{K}^T$, then a trajectory of $\xi$ either indefinitely flows inside a funnel and then jumps into entrances of a funnel with a finite time domain when it arrives at $O_k$, in which case $(x,\ell)$ trajectories are confined to $\mathcal{X}_2$, or, for some value of $(t,i), (t',i') \in \text{dom}((x,\ell))$, $x(t',i')$ is such that for the first time $\xi(t',i') \in \mathcal{E}_{k(t',i')}$ and $k(t',i') \in \mathbb{K}^\infty$. In the latter case, $(x,\ell) \in \mathcal{X}_2$ for all $t + i < t' + i'$ and $(x,\ell) \in \mathcal{X}_1$ for all $t + i \geq t' + i'$.

Define $\mathcal{X} := \mathcal{X}_1 \cup \mathcal{X}_2$. We have shown that $\mathcal{X}$ is the infinite-horizon reachable set of all trajectories $(x,\ell)$ starting with $\xi \in \mathcal{E}_k$ and $\ell = 1$ and that it is compact.

**Proof of Lemma 4**

Lemma 4 follows immediately from Lemmas 2 and 3.

**Proof of Lemma 5**

Lemma 5 follows immediately from Lemmas 1 and 4.

**Proof of Lemma 6**

Since we have shown that all existing solutions starting from $\mathbb{R}^{n+1} \times \mathbb{K} \times \{0,1\}$ are complete, pre-asymptotic stability (as defined in [21]) translates to simple stability as in non-hybrid systems. The rest of the statements of Lemma 6 are self-explanatory.

### 2.7.3 Reduced Set $\mathcal{X}$

In theorem 3 we showed that under some assumptions, solutions of the system from any compact set $S$ of initial conditions will eventually reach a compact set $\mathcal{X}$ and remain there forever. Before further specifying, $\mathcal{X}$ was defined as a union of the infinite-time reachability sets of all trajectories $(\xi, k, \ell)$ starting with $\xi \in \mathcal{E}_k$ and $\ell = 1$, for any $k \in \mathcal{K}$. 


2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

We showed that $\mathcal{X}$ contains a UGAS set to which all trajectories converge. The smaller the set $\mathcal{X}$ is, the tighter is the bound on the UGAS set. However, it is intuitive to see that for some systems of funnels, it can be inferred that $\mathcal{X}$ can be smaller than its most general definition, if we take into account the graph structure with respect to which the system of funnels is defined.

For instance, consider a system of $K < \infty$ funnels such that the set of edges is $\Upsilon = \{(1,2),(2,3),\ldots,(k,k+1),\ldots,(K-1,K)\}$. Note that this set of edges implies that $\text{dom}(F_k) = [0,\infty)$. One can easily see that under the assumptions of Theorem 3 all non-trivial solutions $(x,\ell)$ starting in $\mathbb{R}^{n+3}$ eventually arrive into $F_K \times \{K\} \times \{1\}$ in finite time and remain there forever. The limiting behaviour of all such solutions is therefore characterized by the dynamics $\dot{z} = f(z,u_K(z))$ and $\omega$-limit set will never have ‘tails’ outside $F_K \times \{K\} \times \{1\}$. Recall that $\mathcal{R}_k$ is the infinite-horizon reachable set of the dynamic system

\[
\begin{cases}
\dot{\tau} &= 1 & \text{if } \text{dom}(F_k) = [0,T_k] \\
&= -\tau & \text{if } \text{dom}(F_k) = [0,\infty) \\
\dot{z} &= f(z,u_k(z)) \\
\dot{k} &= 0 \\
\dot{\ell} &= 0
\end{cases}
\] (2.25)

initialized from $(\tau_0,z_0,k_0,\ell_0) \in \mathcal{E}_k \times \{k\} \times \{1\}$ and confined to $F_k \times \{k\} \times \{1\}$.

Hence, in this case $\mathcal{X}$ can be reduced to only $\mathcal{R}_K$

We now generalize this intuition to a general system of funnels based on a set of edges with respect to which it is defined. Consider the system of funnels $\Sigma$ for which $\mathcal{K}$ is the set of funnel indices and $\Upsilon$ is its set of edges. Define $\mathcal{G}(\mathcal{K},\Upsilon)$ to be a directed graph of which the elements of $\mathcal{K}$ are the vertices and the elements of $\Upsilon$ are the edges.

We proceed by defining sinks and cycles for a graph $\mathcal{G}(V,E)$ ($V$ is its set of vertices and $E$ is its set of directed edges). Let a path on the graph $\mathcal{G}$, denoted $P_\mathcal{G}$, be defined as a sequence (that is, an ordered set) of vertices from $V$ and distinct (i.e., non-repeating) edges from $E$ such that each edge $(v,w) \in E$ is located between the the vertices $v \in V$ and $w \in V$. A closed path is a path for which the first and the last vertices are identi-

\[10\] A trivial solution can occur, for instance, when $k \notin \mathcal{K}$ or when $\ell \notin \{0,1\}$.
2.7. THE LIMITING BEHAVIOUR OF SOLUTIONS TO THE HYBRID SYSTEM

cal. A path \( P \) is said to connect vertex \( v \) to vertex \( w \), if \( v \) and \( w \) are the first and last elements of \( P \), respectively. A Graph \( G_P(\mathcal{V}, \mathcal{E}_P) \subseteq G(\mathcal{V}, \mathcal{E}) \) can be constructed from elements of a path \( P \) by regrouping all vertices in \( P \) into \( \mathcal{V} \) and regrouping all edges in \( P \) into \( \mathcal{E} \). A cycle \( G_C(\mathcal{V}_C, \mathcal{E}_C) \) is defined as the graph constructed from the closed path \( C \). A sink is defined as a vertex \( v \in \mathcal{V} \) such that the set of edges \( \mathcal{E} \) does not contain any outgoing edges from \( v \), that is, there is no \( w \in \mathcal{V} \) such that \((v, w) \in \mathcal{E}\).

Define \( G^c(\mathcal{K}^c, \mathcal{Y}^c) \subseteq G(\mathcal{K}, \mathcal{Y}) \) to be the union of all cycles in \( G \). Also, define \( \mathcal{K}^\infty \) as the union of all sinks in \( \mathcal{K} \) (note that all sinks are indices of funnels with an infinite-time-domain, due to the definition of a system-of-funnels). Define a union of all vertices from cycles and all sinks in \( G \) as \( \bar{\mathcal{K}} := \mathcal{K}^c \cup \mathcal{K}^\infty \). Clearly, \( \bar{\mathcal{K}} \subseteq \mathcal{K} \). Let \( \hat{\mathcal{K}} \) be the set of all vertices from \( \bar{\mathcal{K}} \) which are elements of any path connecting any two vertices from \( \bar{\mathcal{K}} \), that is:

\[
\hat{\mathcal{K}} := \{k \in \mathcal{K} \mid \exists v, w \in \bar{\mathcal{K}}, \exists P_G, s.t. \ {v, k, w} \subset P_G \} \quad (2.26)
\]

Finally, we conclude that \( \hat{\mathcal{K}} \) is the (potentially reduced) set of funnel indices which is essential to define \( \mathcal{X} \) in a general case when \( G(\mathcal{K}, \mathcal{Y}) \) is known. Now, the definition of \( \mathcal{X} \) can be replaced from \( \bigcup_{k \in \mathcal{K}} \mathcal{R}_k \) to \( \bigcup_{k \in \hat{\mathcal{K}}} \mathcal{R}_k \), where \( \mathcal{R}_k \) is defined as in eq. (2.25).

The reason that we need to include vertices in paths connecting any vertices from \( \mathcal{K}^c \) to \( \mathcal{K}^\infty \) can be illustrated by the following example (see Fig. 2.1):

\[ \text{Figure 2.1: Example of a graph } G \text{ of a system of funnels.} \]

Assume that a graph \( G(\mathcal{K}, \mathcal{Y}) \) of some system of three funnels is given by \( \mathcal{K} = \{1, 2, 3\} \), \( \mathcal{Y} = \{(1, 1), (1, 2), (2, 3)\} \). It can be inferred that \( 3 \in \mathcal{K}^\infty \). Since a solution of the system can be initialized in funnel 1 and cycle through it arbitrarily many times before passing
through funnel 2 for a finite amount of time and then arriving and staying in funnel 3, the $\omega$-limit-set of this solution will be contained in $\bigcup_{k \in \{1,2,3\}} \mathcal{R}_k$, by the definition of an $\omega$-limit set.

2.8 Other Approaches as Special Cases of System-of-Funnels

2.8.1 Corollary 1 - The Limiting Case of ‘Throw-and-Catch’

In this section, it will be demonstrated that the work in [72] can be closely interpreted in the context of a system of funnels. [72] presents a method for global and robust stabilization of an equilibrium set of a non-linear system, called ‘throw-and-catch’.

The ‘throw-and-catch’ method defines a directed tree structure of all of the equilibrium sets of the system which are relevant for establishing the desired control. The root of the tree is the equilibrium state which is desired to be globally stabilized and it will be termed here the ultimate equilibrium set and denoted $\mathcal{A}$. The tree structure has $J$ paths and each path $j \in \{1, ..., J\}$ has $I_j \geq 2$ nodes on it denoted $\mathcal{A}_{1,j}, ..., \mathcal{A}_{i,j}, ..., \mathcal{A}_{I_j,j}$. Each node corresponds to an equilibrium set of the system. For each path $j$, the node $\mathcal{A}_{I_j,j}$ is the root node and refers to the ultimate equilibrium set $\mathcal{A}$ (that is to be stabilized by the approach).

The method defines three types of controllers: catching, throwing and bootstrap. Catching control is defined for all equilibrium sets $\mathcal{A}_{i,j}$ for which $i > 1$ and is designed to asymptotically attract states of the system in some neighbourhood of $\mathcal{A}_{i,j}$ towards it. Throwing control is designed for all equilibrium sets except for the ultimate one ($\mathcal{A}$) and is such that it ‘throws’ states in some neighbourhood of $\mathcal{A}_{i,j}$ ($i < I_j$) to some neighbourhood of $\mathcal{A}_{i+1,j}$. The bootstrap control is responsible for steering states of the system towards a locality of any of the equilibrium sets from any initial condition that is not in the locality of one of these equilibrium sets. Assume that the bootstrap control has brought the state of the system to some specified locality of $\mathcal{A}_{i,j}$ or that the system was initialized there. Consequently, the other types of controllers (catching and throwing) can be used in alternation with increasing values of $i$ along the same path $j$ until $i = I_j$ and then only catching control is applied until the ultimate equilibrium set is attained.
This will be explained more precisely in the following text in the context of funnels, but first we illustrate the method on a simple example that is taken from [72].

Consider a double pendulum mechanism (with generalised coordinates $\phi_1$ and $\phi_2$) as shown in Fig. 2.2 with only the elbow joint actuated. This mechanism can have 4 equilibrium states with zero actuation. The various (colour-coded) equilibria configurations together with the tree structure chosen is shown in Fig. 2.3. Note that the root of the tree (coloured blue) refers to the equilibrium state in which both links are aligned vertically upwards. All of the four equilibrium configurations are drawn in Fig. 2.4 on the $\phi_1 - \phi_2$ plane.

Figure 2.4 demonstrates qualitatively the behaviour of a trajectory in the $\phi_1 - \phi_2$ plane: The trajectory starts outside any of the relevant localities of equilibria and hence the bootstrap control is operating first (black trajectory). After arrival of the trajectory into some neighbourhood of the green equilibrium state from where a throw mode can ensue, the control is switched to throw mode (pink trajectory). When the trajectory arrives at some locality of the yellow equilibrium where a catch can start, the controller switches to catch mode and the trajectory is steered towards the yellow equilibrium (teal trajectory). This process continues according to the tree structure in Fig. 2.3 until the last catch to the blue (root) equilibrium is performed. A more precise description of the switching sets will be done later in this section.

We start by introducing some terminology and assumptions made in [72].

Consider a non-linear control system of the form:

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

(2.27)

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is continuous. Let $\mathcal{A} \subset \mathbb{R}^n$ be compact.

The following is assumed:
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

Assumptions

1. For any \( j \in \{1, ..., J \} \) and any \( i \in \{1, ..., I_j \} \), \( A_{i,j} \) is disjoint from the other equilibrium sets, and when \( i = I_j \), \( A_{i,j} = A \).

2. For all \( j \in \{1, ..., J \} \) all \( i \in \{2, ..., I_j \} \), there exists a state-feedback law \( \kappa_{i,j} : \mathbb{R}^n \rightarrow \mathbb{R}^m \) such that \( A_{i,j} \) is asymptotically stable for \( \dot{z} = f(z, \kappa_{i,j}(z)) \) with a basin of attraction \( B_{A_{i,j}} \subset \mathbb{R}^n \).

3. For every \( j \in \{1, ..., J \} \) and \( i \in \{1, ..., I_j - 1 \} \), there exists a piecewise continuous function \( \alpha_{(i,j)\rightarrow(i+1,j)} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^m \) which steers trajectories of \( \dot{z} = f(z, \alpha_{(i,j)\rightarrow(i+1,j)}(t)) \)
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

with initial conditions in the closed set $S_{i,j}$ to an open set $E_{i,j}$ within $\tau_{i,j}^f \in \mathbb{R}_{\geq 0}$ time units, where $S_{i,j} \subset \mathbb{R}^n$ contains an open neighbourhood of $A_{i,j}$, $E_{i,j}$ contains an open neighbourhood of $A_{i+1,j}$ and is such that an open $\delta_{i,j}^c$-neighbourhood ($\delta_{i,j}^c > 0$) of itself is contained in $B_{A_{i+1,j}}$.

4. There exists a continuous state-feedback law $\kappa_0 : \mathbb{R}^n \to \mathbb{R}^m$, such that for each solution $z$ to $\dot{z} = f(z, \kappa_0(z))$, there exists a finite $T > 0$ such that $z(T) \in \left( \bigcup_{j \in \{1,\ldots,J\}} \bigcup_{i \in \{i_{j-1},\ldots,i_j\}} E_{i,j} + \frac{\delta_{i,j}^c}{2} B \right) \cup \left( \bigcup_{j \in \{1,\ldots,J\}} \bigcup_{i \in \{1,\ldots,i_{j-1}\}} S_{i,j} \right).$ \(^{11}\)

Auxiliary Definitions

For every $j \in \{1,\ldots,J\}$ we define the following auxiliary sets and set-valued mappings:

- For each $i \in \{1,\ldots,i_j\}$, let $\hat{S}_{i,j}^t$ be a closed set which satisfies:

\[ \hat{S}_{i,j}^t + \frac{\delta_{i,j}^c}{2} B \subset S_{i,j}. \]

\(^{11}\delta B\) denotes the open ball of radius $\delta > 0$ centred at the origin in $\mathbb{R}^n$. 

---

\[ \phi_2 \]

\[ \phi_1 \]

Figure 2.4: A qualitative illustration of the behaviour of the system under the application of the hybrid controller
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

for some $\delta^t_{i,j} > 0$

- For each $i \in \{2, ..., I_j - 1\}$, let $\hat{S}^c_{i,j}$ be an open set which satisfies:
  
  $$\hat{S}^c_{i,j} + \delta^t_{i,j}B \subset S_{i,j}.$$ 

- For each $i \in \{1, ..., I_j - 1\}$, let $\hat{E}^c_{i,j}$ and $\hat{E}^t_{i,j}$ be defined as:
  
  $$\hat{E}^c_{i,j} := \text{cl} \left( E_{i,j} + \delta^c_{i,j}B \right)$$

  and

  $$\hat{E}^t_{i,j} := \text{cl} \left( E_{i,j} + \frac{\delta^c_{i,j}}{2}B \right).$$

- For each $i \in \{1, ..., I_j - 2\}$, let $\bar{\tau}^c_{i,j}$ be the maximal time $t$ for which the following property holds:
  A solution $z$ of $\dot{z} = f(z, \kappa_{i+1,j}(z))$ starting with $z(0) \in \hat{E}^c_{i,j}$ is such that $z(t)$ reaches $\hat{S}^c_{i+1,j}$ for the first time. Define $\bar{\tau}^c_{i,j} : \mathbb{R}^n \Rightarrow \mathbb{R}_{\geq 0}$ to be a set-valued mapping which allocates to each $z_0 \in S_{i,j}$ the set of all times $t$ in the interval $[0, \bar{\tau}^c_{i,j}]$ for which $z(t) \in E_{i,j}$ according to the dynamics $\dot{z} = f(z, \alpha_{(i,j)\to(i+1,j)}(t))$ with $z(0) = z_0$.

[Throwing Funnel]

For every $j \in \{1, ..., J\}$ and $i \in \{1, ..., I_j - 1\}$, define a throwing funnel from node $(i, j)$ in the tree, $F^t_{i,j}$, in the following manner:

- The domain of the funnel $\text{dom}(F^t_{i,j})$ is $[0, \bar{\tau}^t_{i,j}]$.

- The funnel mapping is $F^t_{i,j}(\tau) := \bigcup_{z_0 \in S_{i,j}} \{ z(\tau) \mid \dot{\tau} = 1, \dot{z} = f(z, \alpha_{(i,j)\to(i+1,j)}(\tau)), \tau(0) = \tau_0, z(0) = z_0 \}$. In other words, the funnel is composed of the union of all trajectories starting in $S_{i,j}$ and ending in $E_{i,j}$ (according to assumption 3) after flowing for $\tau^t_{i,j}$ time units.

- The funnel is the set $F^t_{i,j} = \{ (\tau, z) \mid \tau \in \text{dom}(F^t_{i,j}), z \in F^t_{i,j}(\tau) \}.$
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

- The entrance of the funnel is defined as the set \( \mathcal{E}_{i,j}^t := \hat{S}_{i,j}^t \times \{0\} \)
- The control policy associated with the funnel is \( u_{i,j}^t(\tau, z) = \alpha_{(i,j)\rightarrow(i+1,j)}(\tau) \).
- The outlet of the funnel is the set \( \mathcal{O}_{i,j}^t := \bigcup_{z_0 \in \hat{S}_{i,j}^t} \{(\tau, z) \mid \dot{\tau} = 1, \dot{z} = f(z, u_{i,j}^t), \tau(0) = 0, z(0) = z_0, \tau \in \tau_{i,j}^t(z_0)\} \).

Catching Funnel

For every \( j \in \{1,\ldots,J\} \) and \( i \in \{1,\ldots,I_j-1\} \), define a catching funnel to node \((i, j)\) in the tree, \( \mathcal{F}_{i,j}^c \), in the following manner:

- The domain of the funnel \( \text{dom}(\mathcal{F}_{i,j}^c) \) is \([0, \bar{\tau}_{i,j}^c]\) when \( i \in \{1,\ldots,I_j-1\} \) and \([0, \infty)\) when \( i = I_j-1 \).
- The funnel mapping is \( \mathcal{F}_{i,j}^c(\tau) := \bigcup_{z_0 \in \hat{E}_{i,j}^c} \{z(\tau) \mid \dot{\tau} = 1, \dot{z} = f(z, \kappa_{i+1,j}(z)), \tau(0) = \tau_0, z(0) = z_0\} \). In other words, the funnel is composed of the union of all trajectories starting in \( \hat{E}_{i,j}^c \) and flowing according to the stabilizing control \( \kappa_{i+1,j}(z) \) for either \( \tau_{i,j}^c \) time units (thus necessarily arriving at \( \hat{S}_{i+1,j}^c \)), if \( i < I_j-1 \), or otherwise, indefinitely.
- The funnel is the set \( \mathcal{F}_{i,j}^c = \{(\tau, z) \mid \tau \in \text{dom}(\mathcal{F}_{i,j}^c), z \in \mathcal{F}_{i,j}^c(\tau)\} \).
- The entrance of the funnel is defined as the set \( \mathcal{E}_{i,j}^c := \hat{E}_{i,j}^c \times \{0\} \).
- The control policy associated with the funnel is \( u_{i,j}^c(\tau, z) = \kappa_{i+1,j}(z) \).
- If \( i = I_j-1 \) the outlet of the funnel, \( \mathcal{O}_{i,j}^c \), is an empty set. If \( i < I_j-1 \), then the outlet is the set \( \mathcal{O}_{i,j}^c := \bigcup_{z_0 \in \hat{E}_{i,j}^c} \{(\tau, z) \mid \dot{\tau} = 1, \dot{z} = f(z, u_{i,j}^c), \tau(0) = 0, z(0) = z_0, \tau \in \tau_{i,j}^c(z_0)\} \).

Bootstrap Control

The bootstrap control \( u_0 \) is a state-feedback control policy defined such that for any initial state \( z \) of \( \dot{z} = f(z, u_0(z)) \), there exists \( T \in \mathbb{R}_{\geq 0} \) such that \( (z(T), 0) \in \bigcup_{j \in \{1,\ldots,J\}, i \in \{1,\ldots,I_j-1\}} (\mathcal{E}_{i,j}^l \cup \mathcal{E}_{i,j}^c) \).
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

Set of Edges

Associate the indices $k_{t,j}$ and $k_{c,j}$ with the funnels $F_{t,j}$ and $F_{c,j}$, respectively, for each $j \in \{1, ..., J\}$ and $i \in \{1, ..., I_j - 1\}$. We define the set of edges $\Upsilon$ (used to define the system of funnels) as follows:

$$
\Upsilon : = \{(k_{t,j}, k_{c,j}) \mid j \in \{1, ..., J\}, \ i \in \{1, ..., I_j - 1\}\} \cup \{(k_{c,j}, k_{t,i+1,j}) \mid j \in \{1, ..., J\}, \ i \in \{1, ..., I_j - 2\}\}
$$

(2.28)

Set of Funnels

For simplicity of later presentation, we introduce an order to the set of funnel indices so that each index is an integer which uniquely defines a node in the tree (note that several nodes may relate to the same equilibrium state):

$$
\begin{align*}
    k_{1,1}^t &= 1 \\
    k_{1,1}^c &= 2 \\
    k_{2,1}^t &= 3 \\
    &\vdots \\
    k_{I_1-1,1}^c &= 2(I_1 - 1) \\
    k_{1,2}^t &= 2(I_1 - 1) + 1 \\
    k_{1,2}^c &= 2(I_1 - 1) + 2 \\
    &\vdots \\
    k_{I_2-1,2}^c &= 2(I_1 - 1) + 2(I_2 - 1) \\
    &\vdots \\
    k_{I_{j-1},j}^c &= 2\sum_{j=1}^{J} (I_j - 1)
\end{align*}
$$

This results in a total of $2\sum_{j=1}^{J} (I_j - 1)$ funnels in the system.

Define the set of funnel indices as accordingly as $\mathbb{K} := \{1, 2, ..., 2\sum_{j=1}^{J} (I_j - 1)\}$
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

Hybrid State

We return now to our hybrid state defined as \((\tau, z, k, \ell)\) which allows us to define the hybrid controller for the system and the behaviour of the system with it.

Hybrid Controller

The general hybrid control scheme in (2.23) can be rewritten more explicitly for this particular case in the following manner:

The flow dynamics are:

- **throw-and-catch “in-funnel” control:**
  - When \(\ell = 1\) or \((\tau, z) \in \text{cl}(\mathcal{F}_k \setminus \mathcal{O}_k)\):
    \[
    \begin{align*}
    \dot{\tau} & = 1, \quad (2.29) \\
    \dot{z} & = f(z, u_k(\tau, z)), \quad (2.30) \\
    \dot{k} & = 0, \quad (2.31) \\
    \dot{\ell} & = 0. \quad (2.32)
    \end{align*}
    \]

- **Bootstrap control:**
  - When \(\ell = 0\) or \((\tau, z) \in \text{cl}(\mathbb{R}^{n+1} \setminus \mathcal{F}_k)\):
    \[
    \begin{align*}
    \dot{\tau} & = 0, \quad (2.33) \\
    \dot{z} & = f(z, u_0(z)), \quad (2.34) \\
    \dot{k} & = 0, \quad (2.35) \\
    \dot{\ell} & = 0. \quad (2.36)
    \end{align*}
    \]

The jump dynamics are:

- **Jumps between funnels:**
  - When \(\ell = 1\) and \((\tau, z) \in \mathcal{O}_k\):
    \[
    \begin{align*}
    \tau^+ & = 0, \quad (2.37) \\
    z^+ & = z, \quad (2.38) \\
    k^+ & = k + 1, \quad (2.39) \\
    \ell^+ & = 1. \quad (2.40)
    \end{align*}
    \]
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

- Jumps from “in-funnel” to bootstrap:
  When \( \ell = 1 \) or \((\tau, z) \in \text{cl}(\mathbb{R}^{n+1} \setminus \mathcal{F}_k)\):
  \[
  \tau^+ = \tau, \quad z^+ = z, \quad k^+ = k, \quad \ell^+ = 0. \tag{2.41, 2.42, 2.43, 2.44}
  \]

- Jumps from bootstrap to “in-funnel”:
  When \( \ell = 0 \) or \((0, z) \in \mathcal{E}_{k'} \) for some \( k' \in \mathbb{K} \):
  \[
  \tau^+ = 0, \quad z^+ = z, \quad k^+ = k', \quad \ell^+ = 1. \tag{2.45, 2.46, 2.47, 2.48}
  \]

The behaviour of the controller is summarised below:

- If the initial hybrid state of the system is such that throw-and-catch “in-funnel” control is not possible, then the bootstrap controller brings \( z \) to be such that \((0, z) \in \mathcal{E}_{k'} \) for some \( k' \in \mathbb{K} \) in finite time \( T \) and within up to two jumps.

- If the initial state (or the state after the bootstrap control has been turned off, or after a jump ‘from catch to throw’ has occurred) is such that \( \ell = 1 \) and \((\tau, z) \in \mathcal{E}_{i,j} \) (that is, \( k = k_{i,j}^1 \)), then trajectories of \((\tau, z)\) will flow inside \( \mathcal{F}_{i,j}^i \) according to the throwing control \( \alpha_{(i,j) \to (i+1,j)}(\tau) \), until they reach the outlet \( \mathcal{O}_{i,j}^i \). When \((\tau, z) \in \mathcal{O}_{i,j}^i \) we have that \( z \in \mathcal{E}_{i,j} \). Since \( \mathcal{E}_{i,j} \subset \mathcal{E}_{i,j}^c \), after the jump ‘from throw to catch’, we have that \((\tau^+, z^+) \in \mathcal{E}_{i,j}^c \). Therefore, the consequent behaviour will be a flow of \((\tau, z)\) in the catching funnel \( \mathcal{F}_{i,j}^c \), which asymptotically drives \( z \) towards \( \mathcal{A}_{i+1,j} \).

- If the initial state (or the state after the bootstrap control has been turned off, or after a jump ‘from throw to catch’ has occurred) is such that \( \ell = 1 \) and \((\tau, z) \in \mathcal{E}_{i,j}^c \) (that is, \( k = k_{i,j}^c \)), then, if \( i = I_j - 1 \), trajectories will flow indefinitely inside the catching funnel, and otherwise, trajectories of \((\tau, z)\) will flow inside \( \mathcal{F}_{i,j}^c \) according to the throwing control \( \kappa_{i,j}(z) \), until they reach the outlet \( \mathcal{O}_{i,j}^c \). When \((\tau, z) \in \mathcal{O}_{i,j}^c \)
we have that \( z \in \hat{S}_{c}^{i,j} \). In the latter case, since \( \hat{S}_{c}^{i,j} \subset \hat{S}_{t}^{i,j} \), after the ensuing jump ‘from catch to throw’, we have that \( (\tau_{+}, z_{+}) \in \mathcal{E}_{i+1,j}^{c} \) and the consequent behaviour will be a flow in the throw funnel \( F_{i+1,j}^{t} \) which will ‘throw’ \( z \) towards \( A_{i+2} \).

2.8.2 Corollary 2 - The Limiting Case of LQR-Trees

The LQR-Trees approach suggested in [88] can be used to steer trajectories of a non-linear system towards an equilibrium state from a large region in the system’s state space. This is done by first computing an inner-approximation of the region of attraction about the equilibrium state obtained by applying a time-invariant LQR control. This region (denoted \( B_{i} \)) will be addressed as an infinite-time-domain funnel. Then, using RRT algorithm (see Chapters 1 and 3 for more details) to randomly grow the tree and a particular trajectory generator, constructing a tree of trajectories as depicted in Fig. 2.6. Each trajectory connects to either another trajectory or to the equilibrium state (denoted \( z_{e} \) in Fig. 2.6). For each trajectory \( z_{k}(t) (k > 1) \), a time-varying LQR control \( u_{k}(t) \) is used to stabilize it locally and a time-varying region denoted \( B_{k} \) that satisfies a finite-time invariance property is computed about it. These regions represent finite-time-domain funnels much like in our framework. The funnels are connected in such a form that ensures that trajectories arriving to the outlet of a funnel are inside a new funnel, which is referred here as a parent funnel. In this manner, trajectories are steered to the region of attraction about the equilibrium state \( z_{e} \) and then asymptotically converge to it.

Note that the main specific characteristics of the work in [88] are:

- The use of a tree structure for the funnels (whereas a system-of-funnels can in general be defined with respect to a general graph);
- The exact connection of nominal trajectories to each other (this is not necessary when using our framework);
- The funnels are constructed by calculating regions of finite-time-invariance using SOS programs (whereas the system of funnels framework allows a different funnel construction).
- The choice to use LQR control for local stabilization (while the general system-of-funnels framework does not specify a particular control law);
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

- The LQR-Trees method requires to cover some predefined bounded region of the state-space with the projections of the funnels onto the state-space (however, at times, using a bootstrap control may alleviate the necessity to compute many funnels in order to achieve a desired behaviour).

In this section, we present the LQR-Trees approach as a special case of our system-of-funnels framework, with small modifications to the original approach aiming to achieve robustness. In other words, we show how LQR-Trees approach fits into our framework by defining a system of funnels which complies with its properties.

In order to construct a tree of funnels as in [88], consider a tree graph \( T(\mathbb{K}, \mathbb{Y}) \), where elements in the set of vertices \( \mathbb{K} \) are funnel indices in the set of funnels \( \Sigma \) and elements in the set of edges \( \mathbb{Y} \) define all of the funnels connections in \( \Sigma \). An LQR-Tree is constructed in the following manner:

For the dynamic control system \( \dot{z} = f(z, u) \) (where \( z \in \mathbb{R}^n \) and \( u \in \mathbb{R}^m \)), \( z_e \) is an equilibrium state maintained by the constant input \( u_e \), that is, \( f(z_e, u_e) = 0 \).

Let \( \tilde{z} := z - z_e, \tilde{u} := u - u_e \) and define a region about the equilibrium \( z_e \), \( B_1(r) \) as:

\[
B_1(r) := \{ z \mid 0 \leq V_1(\tilde{z}) \leq r \},
\]

where \( r \) is a constant and \( V_1 \) is a Lyapunov function defined in [88] according to the optimal time-invariant LQR cost to go \( J = \int_0^\infty (\tilde{z}^T Q \tilde{z} + \tilde{u}^T R \tilde{u}) \) of the state control stabilizing \( z_e \):

\[
V_1(\tilde{z}) := \tilde{z}^T S_1 \tilde{z},
\]

where \( S_1 \) is the solution to the algebraic Riccati equation \( 0 = Q - S_1 B R^{-1} B^T S_1 + S A + A^T S_1 \), where \( A := \frac{\partial f}{\partial x} \big|_{z_e, u_e} \) and \( B := \frac{\partial f}{\partial u} \big|_{z_e, u_e} \). A time-invariant LQR control which asymptotically and optimally stabilizes \( z_e \) is: \( u_1 := u_e - R^{-1} B^T S_1 \).

\( z_0^k : [0, T_k] \to \mathbb{R}^n \) is a (nominal) trajectory of finite time duration which is obtained from some initial condition \( z_0^k(0) \) using the application of the (nominal) input \( u_0^k : [0, T_k] \to \mathbb{R}^m \) to the system. Let \( \tilde{z}(t) := z - z_0^k(t) \), \( \tilde{u}(t) := u - u_0^k(t) \) and define a time-varying

\[ \text{See eq. (9) in [88].} \]
region about the trajectory $z^0_k(\cdot), B_k(\tau, r(\cdot))$ as:\textsuperscript{13}

$$
B_k(\tau, r(\cdot)) := \{(\tau, z) \mid 0 \leq V_k(\tau, \tilde{z}(\tau)) \leq r(\tau)\},
$$

(2.50)

where $r(\cdot)$ is a scalar function defined on $[0, T_k]$ and $V_k$ is a Lyapunov function defined in [88] according to the optimal cost to go of the time-varying LQR cost to go ($J = \tilde{z}^T Q_f \tilde{z} + \int_t^{T_k} (\tilde{z}^T Q \tilde{z} + \tilde{u}^T R \tilde{u})$) of the time-varying LQR control stabilizing $z^0_k(\cdot)$:

$$
V_k(t, \tilde{z}) := \tilde{z}^T S_k(t) \tilde{z},
$$

where $S_k$ is the solution to the differential Riccati equation

$$
-\dot{S}_k = Q - S_k(t) B(t) R^{-1} B^T(t) S_k(t) + S_k(t) A(t) + A(t)^T S_k(t)
$$

with the final condition $S_k(T_k) = Q_f$, where $A(t) := \frac{\partial f}{\partial z} |_{z^0_k(t), u^0_k(t)}$ and $B := \frac{\partial f}{\partial u} |_{z^0_k(t), u^0_k(t)}$. A time-invariant LQR control which asymptotically and optimally stabilizes $z^0_k(\cdot)$ is:

$$
u_k(t) := u^0_k - R^{-1} B(t)^T S_k(t).
$$

$\mathcal{F}_1$ is the root of the tree, where $1 \in \mathbb{K}$. For every other funnel we define a unique parent funnel and connection depth using the single-valued mapping $P : \mathbb{K} \setminus \{1\} \to \mathbb{K} \times \mathbb{R}_{\geq 0}$, namely, a funnel $\mathcal{F}_p \in \Sigma$ is termed the parent of a funnel $\mathcal{F}_k \in \Sigma$ and $\tau \in \text{dom}(\mathcal{F}_p)$ is its connection depth, if $(p, \tau) = P(k)$.

The root funnel and its entrance are defined as:

$$
\mathcal{F}_1 := \{(\tau, z) \mid \tau \in [0, \infty), \ z \in B_1(\rho_{\mathcal{F}_1})\},
$$

(2.51)

$$
\mathcal{E}_1 := \{(\tau, z) \mid \tau = 0, \ z \in B_1(\rho_{\mathcal{E}_1})\},
$$

(2.52)

where $\rho_{\mathcal{F}_1}$ is obtained from a SOS program as in equation (19) of [88], and $\rho_{\mathcal{E}_1}$ can be chosen according to design considerations in the set $(0, \rho_{\mathcal{F}_1})$.

The funnels with $k \neq 1$, their entrances and outlets are defined as:

\textsuperscript{13}See eq. (28) in [88].
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

\[ F_k := \{ (\tau, z) \mid \tau \in [0, T_k], \ z \in B_k(\tau, \rho_{F_k}(\tau)) \}, \]  
\[ E_k := \{ (\tau, z) \mid \tau \in [0, T_k], \ z \in B_k(\tau, \rho_{E_k}(\tau)) \}, \]  
\[ O := \{ (\tau, z) \in F_k \mid \tau = T_k \}, \]  

where \( \rho_{F_k}(\cdot) \) is function defined on \([0, T_k]\) and obtained from a SOS program as in equation (42) of [88], and \( \rho_{E_k}(\cdot) \) is also a function defined on \([0, T_k]\) and can be chosen according to design considerations such that \( 0 < \rho_{E_k}(\tau) < \rho_{F_k}(\tau) \) for each \( \tau \in [0, T_k] \).

For each \( k \in \mathbb{K}\setminus\{1\} \), there exists a unique \( p \in \mathbb{K} \) such that \( (k, p) \in \Upsilon \) (unique, in the sense that there is no \( m \neq p \) for which \( (k, m) \in \Upsilon \)) and there exists a unique \( \tau' \in \text{dom}(F_p) \) such that \( (p, \tau') = P(k) \) and \( (\tau, z) \in O_k \) implies that \( (\tau', z) \in E_p \). Note that for each \( k \in \mathbb{K}\setminus\{1\} \), the values of \( p \) and \( \tau' \) are dictated by the trajectory generator used in [88] which connects the trajectories in the RRT fashion.

For a bounded region \( D \) in the state space of the dynamic system, the LQR-tree algorithm progresses until all of \( D \) is covered with covered with funnels in the sense that for each \( z \in D \), there exist \( \tau \) and \( k \in \mathbb{K} \) such that \( (\tau, z) \in F_k \). However, since here we construct the funnel with entrances (which makes the connections more restrictive), we require that the \( D \) is covered with entrances, that is for each \( z \in D \), there exist \( \tau \) and \( k \in \mathbb{K} \) such that \( (\tau, z) \in E_k \). In this manner, we alleviate the need to have a particular bootstrap controller, since the bootstrap controller assumption is automatically satisfied for any \( z \in D \). As a result, by setting \( u_0(z) \) to be identically zero, we obtain that all exits out of funnels due to disturbances or noise result in entering into entrances within two jumps and no time lapse.

To sum up, we demonstrated how the LQR-trees approach can be viewed from the system-of-funnels perspective, by adjusting the funnels to have entrances and connecting them outlet-to-entrance, instead of outlet-to-funnel as in [88]. The covering of the state space area \( D \) in entrances (instead of funnels) results in no need in a bootstrap control other than any arbitrary bounded expression, such as zero. The graph which describes connections of funnels is of a tree type, and hence there are no loops in it. This immediately indicates (due to Theorem 3 and the discussion of the reduced set \( X \)) that all of the trajectories will converge to the infinite-time reachability set of from the
2.8. OTHER APPROACHES AS SPECIAL CASES OF SYSTEM-OF-FUNNELS

entrance to the root funnel. Due to the stabilizing dynamics occurring by using $u_1(z)$ in the root funnel (that is, stabilization of $z_e$ is guaranteed for all initial conditions in $\mathcal{F}_1$) we conclude that all trajectories of $z$ initialized in $\mathcal{D}$ arrive asymptotically to $z_e$. 
Figure 2.5: The different sets defined.
\[ \mathcal{T}(K, \Upsilon) \text{ with:} \]

- \( K = \{1, 2, 3, 4, 5, 6, 7\} \)
- \( \Upsilon = \{(2, 1), (3, 2), (5, 1), (4, 5), (6, 5), (7, 6)\} \)

---

Figure 2.6: LQR-Tree \( \mathcal{T}(K, \Upsilon) \) construction illustrated example
This chapter intends to explain how funnels which fit our framework can be constructed. We first present six types of funnels, preserving the flexibility of choice of control for each, so that the specifics can be decided upon by the designer. We then focus on the two most common and basic types of funnels and demonstrate how they can be numerically computed using LQR control and SOS programming.

3.1 Funnel Types

In this section it is shown, on a conceptual level, how several types of funnels can be constructed. Each type of funnel can be used for a different purpose depending on the overall control objective that has to be achieved.

For each of these funnels, we demonstrate the mathematical expression of the sets $F$ and $E$, which define the funnel and its entrance, respectively, and where relevant, we discuss the outlet $O$. Each of these funnel types can be connected to by other funnels of which $O$ is non-empty. Therefore, these funnel types can be viewed as building-blocks with which systems of funnels, as discussed in Chapter 2, can be constructed and possess various properties described in theorems therein.

We start by describing the simplest funnel created about a stabilizable equilibrium state. For simplicity of presentation, this funnel is time invariant. Next, we show the construction of a time-varying funnel about a trajectory. These two funnels are of the same nature as the funnels used in the development of the LQR-Trees and similar approaches for estimating regions of attraction [88, 69, 63, 41, 52, 53, 62, 42, 54, 43, 18].
This is not surprising since the LQR-Trees approach can be interpreted as a spacial case of the system-of-funnels framework (as shown in Section 2.8.2). However, funnels in our framework are not limited to these two types. A third example is a, so-called, bi-directional funnel, which is constructed about two trajectories, to ensure that the undesired gap between these trajectories can be dealt with using feedback control. After designing such bi-directional funnel (using two properly connected sub-funnels), it can be viewed as a stand-alone funnel type, and can be connected to other funnels in the standard manner. The fourth funnel type is also a cascaded one, with the additional requirement that the last sub-funnel connects to the first, thus creating a closed set in which trajectories can exist. This is useful if we want to synthesize an attractive limit cycle in state space using feedback control. The fifth type of funnel, in contrast to the fourth, is constructed about an existing periodic orbit, without the use of sub-funnels. This funnel type resembles the work in [48, 46], which calculate a region of attraction of a stable periodic orbit. The last type of funnel shown in this chapter is a one constructed about an invariant set with some region of attraction. A similar type of funnel is used in the throw-and-catch approach for global stabilization of equilibrium sets (which are not necessarily isolated states) [72, 83], which hints again that this approach fits into the system-of-funnels framework as well, as shown in Section 2.8.1.

Apart from the six types described above, more funnel types can be defined as well, however, we limit our discussion to only these types, which are sufficient for many control applications. The next six subsections will describe each of these fundamental types one-by-one. More detailed explanations about the numerical construction of a funnel about an equilibrium and a funnel about a finite-length trajectory is presented in the next section.

### 3.1.1 A Funnel About an Equilibrium State

In this section we describe a simple funnel defined about a stabilisable equilibrium state. Consider a time-varying control system of the form:

\[
\dot{z} = f(z, u),
\]

where \( z \in \mathbb{R}^n \) and \( u \in \mathbb{R}^m \).

Assume the system is in equilibrium when \( z = z_e \) and \( u = u_e \). Define the following
3.1. FUNNEL TYPES

change of coordinates:

\[
\begin{align*}
\tilde{z} & := z - z_e \\
\tilde{u} & := u - u_e
\end{align*}
\] (3.2)

with which we obtain the error system

\[
\dot{\tilde{z}} = f(\tilde{z} + z_e, \tilde{u} + u_e) := g(\tilde{z}, \tilde{u}).
\] (3.4)

There exist many local control designs for \( \tilde{u} \) which can achieve asymptotic stabilization of \( z_e \) for the system (3.1). For instance, time-invariant LQR control can be used for this sake by linearising \( g(\tilde{z}, \tilde{u}) \) about \((\tilde{z} = 0, \tilde{u} = 0)\) and finding the optimal \( \tilde{u} \) for which some quadratic (in \( \tilde{z} \) and \( \tilde{u} \)) cost function is minimised. For the chosen local stabilizing control \( \tilde{u} \), let \( V(\tilde{z}) \) be a positive definite Lyapunov function of the system (3.4), that is, assume that the time-derivative

\[
\frac{\partial V(\tilde{z})}{\partial \tilde{z}} g(\tilde{z}, \tilde{u})
\] (3.5)

is negative definite in some sub-level-set \( \rho > 0 \) of the origin, denoted \( \tilde{E} := \{ \tilde{z} \mid 0 \leq V(\tilde{z}) \leq \rho \} \). This ensures that any state \( \tilde{z} \in \tilde{E} \) is guaranteed to asymptotically arrive at \( \tilde{z} = 0 \). This motivates us to construct a funnel \( \tilde{F} := \text{gph}(\tilde{F}(\tau)) \) and its entrance \( \tilde{E} := \text{gph}(\tilde{E}(\tau)) \), where \( \tilde{F} \) is defined over the infinite time domain \( \text{dom}(\tilde{F}) = [0, \infty) \) and \( \tilde{E} \) is defined over some finite time domain of the form \( \text{dom}(\tilde{E}) = [0, T_{\tilde{E}}] \), where \( T_{\tilde{E}} \in \mathbb{R}_{\geq 0} \), with the property that \( \tilde{E}(\tau) \subseteq \tilde{E} \) for all \( \tau \in \text{dom}(\tilde{E}) \). For simplicity, \( \tilde{E}(\tau) \) can be chosen as the constant set \( \tilde{E} \). Clearly, \( \tilde{E} \) is within the region of attraction to the equilibrium \((\tilde{z} = 0, \tilde{u} = 0)\), denoted as \( \tilde{R}_e \).

Since, in fact, no state \( \tilde{z} \in \tilde{E} \) can exit \( \tilde{E} \) in any future time, and because, by the definition of a funnel, we must have \( \tilde{E}(\tau) \subset \text{int}(\tilde{F}(\tau)) \) for all \( \tau \in \text{dom}(\tilde{E}) \), we can somewhat arbitrarily define \( \tilde{F}(\tau) \) to be a constant set \( \tilde{F} \) that satisfies \( \tilde{E} \subset \text{int}(\tilde{F}) \). Since states \( \tilde{z} \) which start in \( \tilde{E} \), will never reach states outside \( \tilde{E} \), and therefore also \( \tilde{R}_e \), it is acceptable that \( \tilde{F} \) contains states outside \( \tilde{R}_e \) as they will not be reached within the use of our framework.

We can define the equivalent funnel, entrance and region of attraction in the original
system of coordinates as follows:

\[ F := \text{gph}(F(\tau)), \quad (3.6) \]

where \( F(\tau) \) is defined as the constant set \( F := \{ z | \tilde{z} \in \tilde{F} \} \) and \( \text{dom}(F) = \text{dom}(\tilde{F}) \).

Similarly,

\[ E := \text{gph}(E(\tau)), \quad (3.7) \]

where \( E(\tau) \) is defined as the constant set \( E := \{ z | \tilde{z} \in \tilde{E} \} \) and \( \text{dom}(E) = \text{dom}(\tilde{E}) \).

Finally, let \( R_e \) be defined as \( \{ z | \tilde{z} \in \tilde{R_e} \} \).

Figure 3.1 depicts the sets defined above for a two-dimensional system. On top, \( \tilde{F} \) and \( \tilde{E} \) are shown in the \( \tilde{x}_1 - \tilde{x}_2 \) state-space augmented by time \( \tau \), as well as a projection on the \( \tilde{x}_1 - \tilde{x}_2 \) plane, where also \( \tilde{R}_e \) is depicted. In the bottom, the non-tilde-equivalents are drawn on the original set of coordinates. Notice that \( F(\tau) \) and \( E(\tau) \) are constant, hence they are shown as protruded tubes defined for \( \tau \in [0, \infty) \) and \( \tau \in [0, T_E] \), respectively. Note that it is sometimes useful to set \( T_E \) to zero. Finally, Note that \( E \subseteq R_e \) must hold but \( F \subseteq R_e \) is not required.

3.1.2 A Funnel About a Single Trajectory

Consider again the non-linear time-invariant system (3.1). Denote by \( I := [0, T] \) a finite time interval, where \( T \in \mathbb{R} \), and assume we have access to some open-loop control policy defined over \( I \), \( \pi_I(\tau) \). We denote by \( \varphi(\tau, z_0, \pi_I) \) a nominal trajectory which is obtained by applying \( \pi_I(\tau) \) to (3.1) starting from the initial state \( z_0 \). Note that for all \( \tau \in I \) we have:

\[ \dot{\varphi}(\tau, z_0, \pi_I) = f(\varphi(\tau, z_0, \pi_I), \pi_I(\tau)). \quad (3.8) \]

In this section we show how a funnel can be constructed about the nominal trajectory \( \varphi(\tau, z_0, \pi_I) \). The entrance to this funnel will represent a finite-time invariant set, that is, if a state of the system starts within the \( \tau = 0 \) section of the entrance, it is guaranteed to remain inside it for a finite amount of time, which will be chosen as \( T \), the duration of the trajectory. The following method is directly adapted from the LQR-Trees approach [88], however, differences exist, for instance, here we define the notion of an entrance, which is separate from the notion of a funnel. By detaching the definition of
3.1. FUNNEL TYPES

Shifted system of coordinates:

Original system of coordinates:

Figure 3.1: A time-invariant funnel: (On the right:) time-invariant funnels $\tilde{F}$ (top) and $F$ (bottom) shown either in cyan or transparent and defined for an infinite time domain $[0, \infty)$, and their time-invariant entrance $\tilde{E}$ (top) and $E$ (bottom) shown in magenta and defined for some finite time-domain. The funnels are positioned about an equilibrium state $z_e$ (shown in red) for a two-dimensional system. (On the left:) a projection on the $\tilde{z}_1 - \tilde{z}_2$ plane (top) where $\tilde{F}$, $\tilde{E}$ and $\tilde{R}_e$ are defined, and a projection on the $z_1 - z_2$ plane (top) where $F$, $E$ and $R_e$ are defined.

an entrance from the definition of a funnel, similar finite-time funnel structures about trajectories can exist with different construction strategies. Therefore, our framework is more general in this sense.

The funnel which is obtained in this section, though can be viewed as a stand-alone building block, will be used as a basis for constructing other funnel types in the next two sections.
3.1. FUNNEL TYPES

In order to obtain a funnel about $\varphi$, we first need to define a local stabilizing closed-loop control and then seek a region of finite-time invariance about it, which will essentially form the funnel.

We start by defining the following shift of coordinates (valid for each $\tau \in I$):

\[
\tilde{z} := x(\tau) - \varphi(\tau, z_0, \pi_I), \quad (3.9)
\]
\[
\tilde{u} := u(\tau) - \pi_I(\tau). \quad (3.10)
\]

Now, let us focus on the error system:

\[
\dot{\tilde{z}} = f(\tilde{z} + \varphi, u_{CL}^I) - \dot{\varphi}, \quad (3.11)
\]

where $u_{CL}^I := u_{CL}^I(\tau, \tilde{z}, \pi_I)$ is a local stabilizing closed-loop control that asymptotically drives the system (3.11) towards the equilibrium $(\tilde{z} = 0, \tilde{u} = 0)$ (that is, asymptotically drives $z(\tau)$ towards $\varphi(\tau, z_0, \pi_I)$ while the control $u_{CL}^I$ asymptotically approaches $\pi_I$).

After finding such stabilizing control $u_{CL}^I$ (for instance, using time-varying LQR by linearizing (3.11) about the origin) and substituting it into (3.11), we obtain some (known) time-varying system,

\[
\dot{\tilde{z}} = g(\tau, \tilde{z}), \quad (3.12)
\]

for which we construct a continuous positive definite Lyapunov candidate function of the form:

\[
V(\tau, \tilde{z}). \quad (3.13)
\]

Note that the time-derivative of $V(\tau, \tilde{z})$ is:

\[
\frac{\partial V(\tau, \tilde{z})}{\partial \tau} + \frac{\partial V(\tau, \tilde{z})}{\partial \tilde{z}} g(\tau, \tilde{z}). \quad (3.14)
\]

We denote by $\alpha(\tau)$ and $\rho(\tau)$ functions which define level sets of $V(\tau, \tilde{z})$ for each $\tau \in I$.

Define the set-valued mapping

\[
F(\tau) := \{ z | 0 \leq V(\tau, \tilde{z}) \leq \alpha(\tau) \} \quad (3.15)
\]
over the domain \( \text{dom}(F) := I \). Let \( F \) be the graph of \( F \), that is,
\[
F = \text{gph}(F) := \{ (\tau, z) | \tau \in I, z \in F(\tau) \}.
\]  
(3.16)

Similarly, we define the set-valued mapping
\[
E(\tau) := \{ z | 0 \leq V(\tau, \tilde{z}) \leq \rho(\tau) \}
\]  
(3.17)

over the same domain \( \text{dom}(E) := I \). Let \( E \) be the graph of \( E \), that is,
\[
E = \text{gph}(E) := \{ (\tau, z) | \tau \in I, z \in E(\tau) \}.
\]  
(3.18)

We say that \( F \) is a funnel about \( \varphi \) and \( E \) is its entrance, if \( \alpha(\tau) \) and \( \rho(\tau) \) satisfy the following properties:

- The right derivative of \( \rho(\tau) \), \( \frac{d}{d\tau} + \rho(\tau) \), is well defined and satisfies:
\[
V(\tau, \tilde{z}) = \rho(\tau) \quad \Rightarrow \quad \frac{\partial V(\tau, \tilde{z})}{\partial \tau} + \frac{\partial V(\tau, \tilde{z})}{\partial \tilde{z}} g(t, \tilde{z}) \leq \frac{d}{d\tau} + \rho(\tau)
\]  
(3.19)

- For each \( \tau \in I \),
\[
\rho(\tau) < \alpha(\tau),
\]  
(3.20)

that is, \( E(\tau) \) is inside the interior of \( F(\tau) \).

This construction ensures that if \((\tau_0, z_0) \in E\), then for each \( \tau \in I \setminus [0, \tau_0) \), we have that \( z(\tau) \in E(\tau) \subset \text{int}(F(\tau)) \). Therefore, ideally, we have that \( \varepsilon(\tau) := \alpha(\tau) - \rho(\tau) \ll \rho(\tau) \).

Note that the outlet of \( F \) is defined as:
\[
\mathcal{O} := (T, F(T)).
\]  
(3.21)

In the special case that \( \varphi(T, z_0, \pi_I) \) is an equilibrium state, or in a vicinity of an equilibrium state, \( z_e \), with a region of attraction \( \mathcal{R}_e \), it is desired to ensure that \( \mathcal{O} \subset \{ T \} \times \mathcal{R}_e \), which sets an additional constraint for designing \( \alpha(\tau) \):
\[
\{ z | 0 \leq V(T, \tilde{z}) \leq \alpha(T) \} \subseteq \mathcal{R}_e.
\]  
(3.22)
3.1. FUNNEL TYPES

Figure 3.2 depicts a funnel according to the description above.

![Funnel Diagram](image)

Figure 3.2: A funnel \( \mathcal{F} \) (cyan or transparent) defined about a nominal trajectory, \( \varphi \): The shadowed magenta area (with its blue or transparent extrusion) shows the entrance \( \mathcal{E} \) of the funnel and the green area represents the outlet, \( \mathcal{O} \). Note that for all \( \tau \in I \), \( \rho(\tau) < \alpha(\tau) \).

3.1.3 Two Connected Funnels

In many applications it is desired to steer the state of a non-linear system from some given initial state to another given, possibly distant, target state. This task can be very hard for several classes of systems, such as under-actuated systems (which are not feedback linearisable), hybrid systems, etc..

There are various available trajectory generators which can be used to attempt to connect the start and target states. We focus our attention on the bi-directional trajectory generators which simultaneously attempt to grow ‘forward’ trajectories from the start state and ‘backward’ trajectories from the target state with the aim of connecting the other edges.
3.1. FUNNEL TYPES

This task usually results in two trajectories which are not entirely connected, but have some gap between the finite state of the ‘forward’ trajectory and the initial state of the ‘backward’ trajectory. Nevertheless, this gap can be dealt with using feedback control which attempts to asymptotically stabilize the combined (non-continuous) trajectory. Indeed, the target state will not be exactly reached, however, we can ensure being in some desired neighbourhood of it, if some conditions hold.

For this sake, we propose a ‘bi-directional’ funnel type, which can be built to ensure the robustness of the feedback control used, that is, to ensure that nearby states of the ‘forward’ trajectory end up being close to the ‘backward’ trajectory, and eventually at the designated neighbourhood of the target state.

As its name suggests, the bi-directional funnel is composed of two sub-funnels, one designed about each of the two trajectories returned by the bi-directional trajectory generator used. In order to ensure robust control, we demand that the outlet of the forward sub-funnel ‘enters’ to the entrance of the backward sub-funnel. After these two sub-funnels are designed, the combination can be viewed as a single building-block in our system-of-funnels framework, since bi-directional funnels can be connected to and connected by other funnels in the normal manner.

It should be mentioned that such bi-directional funnels have not appeared in the literature yet, however, in [88], a similar idea is suggested as a possible extension of the LQR-trees approach.

The development below relies on the previous section, since each sub-funnel is constructed with large similarity to the funnel type which is designed about a single trajectory. The main difference is that here we define the condition that has to be satisfied in order for the two funnels to be properly connected. Usually, this means that the sub-funnels are not designed simultaneously (in contrast to the two given trajectories).

Given two states, \( z_{\text{start}} \) and \( z_{\text{target}} \), we use a bi-directional trajectory generator (such as the bi-directional RRT algorithm) to obtain two open loop policies:

- \( \pi_{\text{start}}(\tau) \) which is defined over the finite time interval \( I_{\text{start}} := [0, T_{\text{start}}] \) (where \( T_{\text{start}} \in \mathbb{R} \)), and
3.1. FUNNEL TYPES

- \( \pi_{\text{target}}(\tau) \) which is defined over the finite time interval \( I_{\text{target}} := [0, T_{\text{target}}] \) (where \( T_{\text{target}} \in \mathbb{R} \)).

When \( \pi_{\text{start}}(\tau) \) is applied on the system (3.1) starting from the start state, \( z_{\text{start}} \), we obtain a trajectory which we denote as:

\[
\varphi_{\text{start}}(\tau, z_{\text{start}}, \pi_{\text{start}}).
\]  \hspace{1cm} (3.23)

Note that \( \varphi_{\text{start}} \) satisfies:

\[
\frac{d}{d\tau}\varphi_{\text{start}} = f(\varphi_{\text{start}}, \pi_{\text{start}}),
\]  \hspace{1cm} (3.24)

\[
\varphi_{\text{start}}(0, z_{\text{start}}, \pi_{\text{start}}) = z_{\text{start}}.
\]  \hspace{1cm} (3.25)

Let \( z_{\text{start}}^{\text{END}} \) be the final state of \( \varphi_{\text{start}} \), that is:

\[
z_{\text{start}}^{\text{END}} := \varphi_{\text{start}}(T_{\text{start}}, z_{\text{start}}, \pi_{\text{start}}).
\]  \hspace{1cm} (3.26)

Similarly, let \( z_{\text{target}}^{\text{BEGIN}} \) be the initial condition of the trajectory that arrives at the target state after applying \( \pi_{\text{target}}(\tau) \) during \( I_{\text{target}} \), denoted as:

\[
\varphi_{\text{target}}(\tau, z_{\text{target}}^{\text{BEGIN}}, \pi_{\text{target}}).
\]  \hspace{1cm} (3.27)

Note that \( \varphi_{\text{target}} \) satisfies:

\[
\frac{d}{d\tau}\varphi_{\text{target}} = f(\varphi_{\text{target}}, \pi_{\text{target}}),
\]  \hspace{1cm} (3.28)

\[
\varphi_{\text{target}}(0, z_{\text{target}}^{\text{BEGIN}}, \pi_{\text{target}}) = z_{\text{target}}^{\text{BEGIN}},
\]  \hspace{1cm} (3.29)

\[
\varphi_{\text{target}}(T_{\text{target}}, z_{\text{target}}^{\text{BEGIN}}, \pi_{\text{target}}) = z_{\text{target}}.
\]  \hspace{1cm} (3.30)

Normally, even though the bi-directional trajectory generator attempts to find such trajectories for which \( z_{\text{start}}^{\text{END}} \) and \( z_{\text{target}}^{\text{BEGIN}} \) are as close as possible, there would be a gap between them, especially when the dynamics (3.1) are of a non-holonomic system. This is because it is not always possible to find a control input to navigate the state of the non-holonomic system to any arbitrary desired new state. Hence, without loss of generality, we assume that \( z_{\text{start}}^{\text{END}} \neq z_{\text{target}}^{\text{BEGIN}} \), since otherwise only one funnel is necessary since
we get the case of a funnel about a single (continuous) trajectory.

Denote by $F_{\text{start}}$ and $F_{\text{target}}$ the sub-funnels defined about $\varphi_{\text{start}}$ and $\varphi_{\text{target}}$, respectively, by $E_{\text{start}}$ and $E_{\text{target}}$ their entrances, and by $O_{\text{start}}$ and $O_{\text{target}}$, their outlets.

Let $F_{\text{start}}(\tau)$ be such that $\text{gph}(F_{\text{start}}) = F_{\text{start}}$ and $\text{dom}(F_{\text{start}}) = I_{\text{start}}$. Let $F_{\text{target}}(\tau)$, $E_{\text{start}}(\tau)$, $E_{\text{target}}(\tau)$ and their domains be defined in the same manner.

In order for our method to work, it is necessary that:

$$z_{\text{END}}^{\text{start}} \in \text{int}(E_{\text{target}}(0)). \quad (3.31)$$

Note that this is a special case of the more general condition $x_{\text{END}}^{\text{start}} \in \text{int}(E_{\text{target}}(\tau))$ for some $\tau \in I_{\text{target}}$, however, we utilise the condition (3.31) without loss of generality, since if the general condition is satisfied for some non-zero $\tau^*$, then the ‘backward’ trajectory can be truncated in the time-interval $[0, \tau^*)$.

If the condition (3.31) is not satisfied for the largest set $E_{\text{target}}(0)$ that we can find, then the trajectory generator must be used for a longer time to ensure that this condition is satisfied.

Therefore, $F_{\text{target}}$ and $E_{\text{target}}$ are designed first until the condition (3.31) is verified. As a second stage, $F_{\text{start}}$ and $E_{\text{start}}$ are designed. The process is briefly described below. More details about the numerics will be given in the next chapter.

Initially, the sub-funnel $F_{\text{target}}$ is constructed about $\varphi_{\text{target}}$ such that its outlet, $O_{\text{target}}$ is inside some desired region, $R_{\text{target}}$, about the target state $z_{\text{target}}$. For instance, if $z_{\text{target}}$ is a stabilizable equilibrium state with some local closed-loop control, then $R_{\text{target}}$ can represent a conservative estimate of its region of attraction, thus ensuring that every state $z(\tau)$ for which $(\tau, z(\tau)) \in E_{\text{target}}$ is guaranteed to satisfy $(T_{\text{target}}, z(T_{\text{target}})) \in R_{\text{target}}$. If $z_{\text{target}}$ is not an equilibrium, then other considerations can be made while choosing an appropriate region $R_{\text{target}}$.

After such sub-funnel $F_{\text{target}}$ and its entrance, $E_{\text{target}}$, are designed, we can construct the sub-funnel $F_{\text{start}}$ and its entrance, $E_{\text{start}}$, about $\varphi_{\text{start}}$ with the following condition for a
proper connection:

\[ O_{\text{start}} \subseteq \{ T_{\text{start}} \} \times E_{\text{target}}(0). \] (3.32)

See Fig. 3.3 for an illustration of the two connected sub-funnels which comprise the bi-directional funnel.

Figure 3.3: A bi-directional funnel composed from two sub-funnels about disconnected trajectories: The forward sub-funnel, \( F_{\text{start}} \) and its entrance \( E_{\text{start}} \), are shown on the left. The backward sub-funnel, \( F_{\text{target}} \) and its entrance \( E_{\text{target}} \), are shown on the right. It can be seen that the outlet of the forward funnel, \( O_{\text{start}} \), is contained in the set \( \{ T_{\text{start}} \} \times E_{\text{target}}(0) \), which means that the two sub-funnels are properly connected.

### 3.1.4 A Periodic Funnel

Sometimes it is desired to design a limit cycle trajectory in the state space of the system (3.1). For most physical systems, designing a single continuous trajectory can be a very challenging task and therefore it makes sense to break it down to the designs of several
3.1. FUNNEL TYPES

shorter trajectories which form a kind of ‘broken’ periodic orbit, that is, they comprise a piecewise continuous limit cycle. An example of a similar idea of artificially creating a periodical funnel using sub-funnels about nominal trajectories that form a periodic orbit can be found in [22]. In this work, a method to conservatively estimate non-local regions of stability of a periodic orbit of a Piecewise Linear System (or a nonlinear system which can be approximated by one) is described. The stability guarantees are formed using appropriate Linear Matrix Inequalities.

In this section, we design for each trajectory a sub-funnel (for instance, using the bi-directional funnel type described in the previous section), and then ensure that all of the sub-funnels are connected ‘head-to-tail’ to form our new periodic funnel type. Doing that requires ensuring that trajectories of the system that start in any of the entrances to the sub-funnels will remain close enough, and even converge to, some periodic orbit inside the overall periodic funnel.

The motivation for designing such cycles comes frequently from various motion planning tasks. Take for example the problem of designing a walking, climbing or running gait for a bipedal under-actuated robot. This can be done by first selecting several desired states to be on the planned limit-cycle and then trying to connect them with either bi-directional funnels or funnels designed about a single trajectory, etc..

Each such low-level funnel is then used a sub-funnel for the more elaborate periodic funnel. Figure 3.4 shows an example of a periodic funnel which is composed of other sub-funnels. The sub-funnels can be in general of different types (as long as they have a non-empty outlet), therefore, for simplicity, the discontinuities are not shown in the figure.

Note that each periodic funnel is then treated as a single unit. Other funnels can be connected to it (by treating the union of the entrances of the sub-funnels as the entrance to the periodic funnel). This funnel can be connected to other funnels (by selecting any of the outlets of the sub-funnels to serve as the outlet of the periodic funnel). In this manner, including this type of periodic funnels in a system of funnels means that we can, for instance, switch between one type of gait to another, by representing each gait by its suitable periodic funnel and interconnecting these two periodic funnels, either directly or using some bridging funnels. In a gait example, impact between the foot and
the ground is often a feature in the resulting limit cycle describing the gait. In this case, the impact map can be integrated into the design of the periodic funnel by means of considering the states of the funnel that intersect with the impact surface and verifying that their impact mapping is inside the entrance of the periodic funnel.

In order to define a periodic funnel, we use a similar process to the one used to design a bi-directional funnel, but with the additional constraint that the outlet of the last sub-funnel must be inside the entrance of the first sub-funnel.

Assume that we have \( N \) local trajectories (normally, with gaps between them) which form a discontinuous periodic orbit when concatenated:

\[
\varphi_1(\tau, z_1, \pi_{I_1}), \varphi_2(\tau, z_2, \pi_{I_2}), ..., \varphi_N(\tau, z_N, \pi_{I_N})
\]  \hspace{1cm} (3.33)

where for each \( i \in \{1, ..., N\} \), \( z_i \) is the initial state of the trajectory \( \varphi_i \), which is obtained by applying the (open-loop) control low \( \pi_{I_i} \) for the duration of the finite time interval \( I_i := [0, T_i] \), where \( T_i \in \mathbb{R}_{\geq 0} \).
3.1. FUNNEL TYPES

We wish to design a sub-funnel for each of these \( N \) trajectories. Define by \( F_i := \text{gph}(\mathcal{F}_i(\tau)) \), \( E_i := \text{gph}(\mathcal{E}_i(\tau)) \) and \( O_i \), the funnel, entrance and outlet related to the trajectory \( \varphi_i \).

For a valid periodic trajectory, we require the following:

- For each \( i \in \{1, ..., N - 1\} \), we have that \( O_i \in \{T_i\} \times E_{i+1}(0) \), and
- \( O_N \in \{T_N\} \times E_1(0) \).

3.1.5 A Funnel About a Limit Cycle

In this section we describe a type of funnel which is similar to that of the funnel about a single trajectory, with the main difference being that this funnel is periodic. Starting from the original non-linear control system (3.1), we assume the existence of a period \( T_p \), an initial state \( z_0 \) and a periodic control input \( \pi_{T_p}(\tau) \), where \( \tau \in [0, \infty) \) with period \( T_p \) (that is, \( \pi_{T_p}(\tau) = \pi_{T_p}(\tau + T_p) \) for all \( \tau \geq 0 \)) for which the system’s behaviour is periodic, that is the trajectory \( \varphi_{T_p}(\tau, z_0, \pi_{T_p}) \) obtained is a periodic orbit with a period \( T_p \):

\[
\begin{align*}
\varphi_{T_p}(0, z_0, \pi_{T_p}) &= z_0 \quad (3.34) \\
\dot{\varphi}_{T_p} &= f(\varphi_{T_p}, \pi_{T_p}) \quad (3.35) \\
\varphi_{T_p}(\tau, z_0, \pi_{T_p}) &= \varphi_{T_p}(\tau + T_p, z_0, \pi_{T_p}) \quad (3.36)
\end{align*}
\]

Replicating the same procedure as for the case of a funnel about a single trajectory, we define an error system which we try to regulate to zero using feedback control such as time-varying LQR.

First we define a shift of coordinates:

\[
\begin{align*}
\tilde{z}(\tau) &:= z(\tau) - \varphi_{T_p}(\tau, z_0, \pi_{T_p}) \quad (3.37) \\
\tilde{u}(\tau) &:= u(\tau) - \pi_{T_p}(\tau) \quad (3.38)
\end{align*}
\]
3.1. FUNNEL TYPES

Then we obtain the error system

\[ \dot{\tilde{z}} = f(\tilde{z} + \varphi_{T_p}, u_{T_p}^{CL}) - \dot{\varphi}_{T_p}, \quad (3.40) \]

where \( u_{T_p}^{CL} := u_{T_p}^{CL}(\tau, \tilde{z}, \pi) \) is a local periodic stabilizing closed-loop control (with period \( T_p \)) that asymptotically drives the system (3.40) towards the equilibrium \((\tilde{z} = 0, \tilde{u} = 0)\) (that is, asymptotically drives \( z(\tau) \) towards \( \varphi_{T_p}(\tau, z_0, \pi_{T_p}) \) while the control \( u_{T_p}^{CL} \) asymptotically approaches \( \pi_{T_p} \)).

After finding such stabilizing control \( u_{T_p}^{CL} \) and substituting it into (3.40), we obtain some (known) time-varying system,

\[ \dot{\tilde{z}} = g_{T_p}(\tau, \tilde{z}). \quad (3.41) \]

Note that \( g_{T_p} \) is also periodic with period \( T_p \), and that in the special case when \( u_{T_p}^{CL} \) depends only on \( \tilde{z} \), \( g_{T_p} \) is time-invariant (this is possible when the open loop input \( \pi_{T_p} \) is a constant).

We construct a continuous positive definite and periodic Lyapunov candidate function of the form:

\[ V_{T_p}(\tau, \tilde{z}). \quad (3.42) \]

Note that the time-derivative of \( V_{T_p}(\tau, \tilde{z}) \) is:

\[ \frac{\partial V_{T_p}(\tau, \tilde{z})}{\partial \tau} + \frac{\partial V_{T_p}(\tau, \tilde{z})}{\partial \tilde{z}} g_{T_p}(\tau, \tilde{z}). \quad (3.43) \]

We denote by \( \alpha(\tau) \) and \( \rho_{T_p}(\tau) \) functions which define level the sets of \( V_{T_p}(\tau, \tilde{z}) \) for each \( \tau \geq 0 \).

Like in the case of a funnel about a single trajectory, we define the set-valued mappings

\[ F(\tau) := \{ z | 0 \leq V_{T_p}(\tau, \tilde{z}) \leq \alpha(\tau) \}, \quad (3.44) \]

which is defined over the infinite domain \( \text{dom}(F) := [0, \infty) \), and

\[ E(\tau) := \{ z | 0 \leq V_{T_p}(\tau, \tilde{z}) \leq \rho_{T_p}(\tau) \}, \quad (3.45) \]

which is defined over the finite-time domain \( \text{dom}(E) := [0, T_p] \).
Let $\mathcal{F}$ and $\mathcal{E}$ be the graphs of $\mathbf{F}$ and $\mathbf{E}$, respectively, that is,

$$
\mathcal{F} = \text{gph}(\mathbf{F}) \quad (3.46)
$$

$$
\mathcal{E} = \text{gph}(\mathbf{E}) \quad (3.47)
$$

We say that $\mathcal{F}$ is a funnel about $\varphi$ and $\mathcal{E}$ is its entrance, if $\alpha(\tau)$ and $\rho_{T_p}(\tau)$ satisfy the following conditions:

- The right derivative of $\rho_{T_p}(\tau)$, $\frac{d}{d\tau} \rho_{T_p}(\tau)$, is well defined and satisfies:
  $$
  V_{T_p}(\tau, \tilde{z}) = \rho_{T_p}(\tau) \implies \frac{\partial V_{T_p}(\tau, \tilde{z})}{\partial \tau} + \frac{\partial V_{T_p}(\tau, \tilde{z})}{\partial \tilde{z}} g_{T_p}(t, \tilde{z}) \leq \frac{d}{d\tau} \rho_{T_p}(\tau) \quad (3.48)
  $$

- $\rho_{T_p}(\tau)$ is periodic with a period $T_p$:
  $$
  \rho_{T_p}(\tau) = \rho_{T_p}(\tau + T_p). \quad (3.49)
  $$

- For each $\tau \in \mathbb{R}_{\geq 0}$,
  $$
  \rho_{T_p}(\tau) < \alpha(\tau), \quad (3.50)
  $$

and therefore, $\mathcal{E}(\tau) \subset \mathcal{F}(\tau)$.

This construction ensures that:

1. For each $\tau \in [0, T_p]$, $\mathcal{E}(\tau)$ is inside the region of attraction of the periodic orbit, which we denote as $\mathcal{R}_{\phi_{T_p}}$.

2. If $(\tau_0, z_0) \in \mathcal{E}$, then for each $\tau \in [\tau_0, \infty)$, we have that $z(\tau) \in \text{int}(\mathbf{F}(\tau))$. Therefore, ideally, we have that $\varepsilon(\tau) := \alpha(\tau) - \rho_{T_p}(\tau) \ll \rho_{T_p}(\tau)$.

Note that, by definition, $\mathcal{F}$ has an empty outlet.

Figure 3.5 depicts the projection on a two-dimensional state-space of an infinite-time-domain funnel, $\mathcal{F}$, about a limit cycle $\phi_{T_p}$ (a periodic orbit with period $T_p$) and its finite-time-domain entrance, $\mathcal{E}$, according to the description above. The region of attraction of the limit cycle, $\mathcal{R}_{\phi_{T_p}}$, is also shown and it can be seen that $\mathcal{E}$ is inside $[0, T_p] \times \mathcal{R}_{\phi_{T_p}}$. 

---

132
3.1. FUNNEL TYPES

The reader is referred to [46] for an alternative computational approach of estimating regions of attraction about periodic orbits.

3.1.6 A Funnel About an invariant Set

With similarity to the funnel defined about an equilibrium state, we can also define a funnel about an equilibrium set which can be stabilized. In fact, we can generalize this idea and define a funnel about a positively invariant set which does not have to contain any equilibria states in it.

Assume that the system (3.1) is controlled by a time-invariant control input $u_{\Omega}(z)$ for which there exists a non-empty compact and uniformly finite-time-attractive set $\Omega$ which is positively invariant. Note that the development below is done assuming a time-invariant resulting closed-loop system, however, a time-varying case can also be studied in a similar manner to the developments in the previous four sections.

A positively invariant set $\Omega$ is a set for which no trajectory can exit if started in it. In
3.1. FUNNEL TYPES

other words, for any \( z_0 \in \Omega \), the trajectory which is obtained by applying \( u = u_\Omega(z) \) in (3.1) which is initialized in \( z_0 \), denoted \( \varphi(\tau, z_0, u_\Omega) \), remains inside \( \Omega \) for all \( \tau \geq 0 \).

If \( \Omega \) is finite-time-attractive, then there exists a region of attraction \( R_\Omega \) such that if a trajectory starts inside it, it is guaranteed to enter the positively invariant set \( \Omega \) in finite time, that is:

\[
z_0 \in R_\Omega \implies \exists T \in \mathbb{R} \text{ such that } \forall \tau \geq T, \varphi(\tau, z_0, u_\Omega) \in \Omega\tag{3.51}
\]

We proceed by defining a funnel about the invariant set described above with the aim that each trajectory that enters the entrance of the funnel will arrive in the invariant set and then remain forever inside it.

First, we search for a continuous function \( V(z) \) with the following properties:

- \( V(z) \leq 0 \) on the closed set \( \Omega \).
- \( V(z) > 0 \) on the open set \( \mathbb{R}^n \setminus \Omega \).
- \( V(z) \) is radially unbounded.

We then search for the maximal value of the level set \( \rho > 0 \) for which the following holds:

\[
z \in \{ z | 0 \leq V(z) \leq \rho \} \setminus \Omega \implies \dot{V}(z) = \frac{\partial V(z)}{\partial z} f(z, u_\Omega(z)) < 0. \tag{3.52}
\]

We then define the entrance to the funnel to be the constant sub-level-set \( \rho \) of \( V(z) \), defined over a finite-time-domain:

\[
E(\tau) := E := \{ z | 0 \leq V(z) \leq \rho \} \quad \text{and} \quad \mathcal{E} := [0, T_\mathcal{E}] \times E, \tag{3.53}
\]

where \( T_\mathcal{E} \in \mathbb{R}_{\geq 0} \).

Finally, the infinite-time-domain time-invariant funnel \( F(\tau) := F \) can be constructed as some \( \mathbb{B}_\varepsilon \) environment of \( E \), where \( \mathbb{B}_\varepsilon \) is an \( n \)-dimensional ball of small radius \( \varepsilon \):
3.1. FUNNEL TYPES

\[
F(\tau) := F := \bigcup_{z \in E, z \epsilon \in B_\epsilon} z + z_\epsilon
\]
\[
\mathcal{F} := [0, \infty) \times F.
\]

Note that it is not necessary that \( F \subseteq \mathcal{R}_\Omega \). An example of a funnel about an invariant set is shown in Fig. 3.6. It can be seen that a trajectory starting inside \( E \) is guaranteed to enter \( \Omega \) in finite time, and then remain there forever.

Figure 3.6: An infinite-time-domain funnel \( \mathcal{F} \) about an invariant set and its finite-time-domain entrance \( \mathcal{E} \) shown as projections on a two-dimensional state-space. The broken line represents the region of finite-time attraction about the invariant set, \( \mathcal{R}_\Omega \). A typical trajectory which starts inside the entrance set and is pulled by the controller into the invariant set is also shown.

Similar types of funnels (albeit without the finite-time attractivity requirement) are defined in the catch-and-throw approach for the special case for which the attractive invariant set is a stabilizable equilibrium set. This further shows that the system-of-funnels framework can be used in a wide range of applications and it proposes a unified approach for various non-trivial control tasks, among which are global stabilization and path planning for various classes of systems.

A computational framework which allows the numerical computation of such funnels (without having to compute or estimate any reachability sets) is described in [64, 65]. In their work, the authors present a safety verification framework which relies on computing
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

barrier certificates (using SOS programming). A barrier certificate is Lyapunov-like function of which value along trajectories of the system decreases and, in addition, it is strictly positive in the forbidden regions of the state-space and non-positive in a specific set of initial conditions of interest. The barrier itself is the zero level-set of this function. The computation of such functions is enabled via implementing suitable SOS programs. Using this technique, some regions of the state-space could be proved positively-invariant when the forbidden region is defined outside them.

3.2 Numerical Construction of Funnels about Equilibria and Trajectories

In this section we demonstrate how funnels of the first two types (presented in the previous section) can be computed using LQR control and SOS Programming. First we formulate a local stabilising controller for an equilibrium state and a local tracking controller for a finite-length trajectory. Then we present a numerical method of calculating the funnel region. The material in this section is taken from [88] and is brought for the sake of completeness of the thesis, however, it is not our contribution.

3.2.1 Stabilizing an Equilibrium State using Time-Invariant LQR

Consider a smooth non-linear system:

\[ \dot{z} = f(z, u). \]  (3.57)

where \( z \in \mathbb{R}^n \) is the state of the system and \( u \in \mathbb{R}^m \) is the control input. Assume the system (3.57) has an equilibrium state \( z_e \) which is maintained by the input \( u_e \), that is, \( f(z_e, u_e) = 0 \). To locally stabilize the state of the system (3.57) about \( z_e \), we design a time-invariant LQR control (TI-LQR). Later (in Section 3.2.4) we give a conservative estimate of the resulting region of attraction (RoA) to allow us define when is is safe to switch to the this TI-LQR control.
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

We start by using the coordinate transformation:

\[
\begin{align*}
\tilde{z} &= z - z_e, \quad (3.58) \\
\tilde{u} &= u - u_e, \quad (3.59)
\end{align*}
\]

which effectively shifts the equilibrium state to the origin.

We now linearise the system in the neighbourhood of the equilibrium state \((z_e, u_e)\):

\[
\dot{\tilde{z}} \approx A\tilde{z}(t) + B\tilde{u}(t), \quad (3.60)
\]

where

\[
A = \left. \frac{\partial f}{\partial z} \right|_{z=z_e, u=u_e}, \quad B = \left. \frac{\partial f}{\partial u} \right|_{z=z_e, u=u_e}. \quad (3.61)
\]

Assume that the linearised system is controllable. Define the quadratic regulator cost-to-go function as:

\[
J(\tilde{z}') = \int_0^\infty \left[ \tilde{z}^T(t)Q\tilde{z}(t) + \tilde{u}^T(t)R\tilde{u}(t) \right] dt, \quad (3.62)
\]

where

\[
Q = Q^T \geq 0, \quad R = R^T > 0, \quad \tilde{z}(0) = \tilde{z}'.
\]

The optimal cost-to-go function for the linearized system is

\[
J^*(\tilde{z}) = \tilde{z}^T S\tilde{z}, \quad (3.63)
\]

where \(S\) is the positive-definite solution to the infinity Ricatti equation:

\[
0 = Q - SBR^{-1}B^TS + SA + A^TS. \quad (3.64)
\]

The optimal feedback policy for the linear system is then given by:

\[
\tilde{u^*} = -R^{-1}B^TS\tilde{z}. \quad (3.65)
\]

3.2.2 Stabilizing a Trajectory using Time-Varying LQR

As seen in Section 3.1.2, a funnel can be defined around a nominal trajectory. In Chapter 4 we will show how an appropriate input signal \(u_0(t)\) can be found to design a nominal
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

trajectory $z_0(t)$.

Given a nominal trajectory $z_0(t)$ and the matching input signal $u_0(t)$, both defined over a finite time interval, $t \in [t_0, t_f]$, we stabilize the trajectory using a time-varying LQR controller (TV-LQR). By linearising the system around the trajectory, we obtain:

$$\dot{\tilde{z}} \approx A\tilde{z}(t) + B\tilde{u}(t),$$

(3.66)

where

$$\tilde{z} = z - z_0(t),$$

(3.67)

$$\tilde{u} = u - u_0(t).$$

(3.68)

Define a quadratic regulator cost function with a terminal term:

$$J(\tilde{z}', t') = \tilde{z}^T(t_f)Q_f\tilde{z}(t_f) + \int_{t'}^{t_f} [\tilde{z}^T(t)Q\tilde{z}(t) + \tilde{u}^T(t)R\tilde{u}(t)] dt,$$

(3.69)

where

$$Q_f = Q_f^T \geq 0, Q = Q^T \geq 0, R = R^T > 0, \quad \tilde{z}(t') = \tilde{z}' .$$

Since the linearised system is time-varying, the cost-to-go is time-varying.

The optimal cost-to-go, $J^*(\tilde{z})$, takes the form:

$$J^*(\tilde{z}) = \tilde{z}^T S(t) \tilde{z} ,$$

(3.70)

where $S(t) = S^T(t) > 0$ is the solution to the time-varying Ricatti equation:

$$-\dot{S} = Q - SBR^{-1}B^T S + SA + A^T S, S(t_f) = Q_f.$$

(3.71)

The optimal feedback policy for the time-varying linear system is given by:

$$\tilde{u}^*(t) := -R^{-1}B^T(t)S(t)\tilde{z}(t) =: -K(t)\tilde{z}(t).$$

(3.72)
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

3.2.3 Sum-of-Squares (SOS) Programming - Background

In order to obtain an estimate of a region of attraction to the equilibrium state (using the TI-LQR control) or a finite-time invariance region about the trajectory which is tracked using the (TI-LQR control) we can use Lyapunov local analysis. Useful Lyapunov functions are obtained using the quadratic positive definite optimal cost-to-go functions. These functions can prove global stability for the linearised system, however, for the non-linear model, we can find a local region where the time-derivative of the positive definite Lyapunov function satisfies some conditions required for the relevant stability property. In order to verify these conditions for a given Lyapunov function, one can use a SOS program to ensure non-negativity of certain polynomial systems. This process will be explained in this and the following subsections. We will use the resulting regions of attraction to construct funnels later on.

Consider the polynomial function:

\[ f(z) = az^4 + bz^3 + cz^2 + dz + e, \quad z \in \mathbb{R}. \]

In order to check if \( f(z) \) is non-negative for every \( z \), it is sufficient to check whether there exists a positive semi-definite matrix:

\[ M = \begin{bmatrix}
\alpha_{11} & \alpha_{12} & \alpha_{13} \\
\alpha_{21} & \alpha_{22} & \alpha_{22} \\
\alpha_{31} & \alpha_{32} & \alpha_{33}
\end{bmatrix}, \]

satisfying:

\[
\begin{bmatrix}
1 \\
z \\
z^2
\end{bmatrix}^T
\begin{bmatrix}
\alpha_{11} & \alpha_{12} & \alpha_{13} \\
\alpha_{21} & \alpha_{22} & \alpha_{22} \\
\alpha_{31} & \alpha_{32} & \alpha_{33}
\end{bmatrix}
\begin{bmatrix}
1 \\
z \\
z^2
\end{bmatrix} = az^4 + bz^3 + cz^2 + dz + e.
\]

The SOS program can be used to guarantee that a given Lyapunov function \( V(z) \) is positive-definite and its derivative \( \frac{\partial V}{\partial z} f(z) \) is negative-definite (or possesses similar properties), which will be used to ensure invariance of certain sets. A useful MATLAB toolbox named SOSTOOLS [66] is used later to solve polynomial inequality problems.
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

numerically.

We first show how to calculate a region of attraction about an equilibrium state, and then a region of finite-time invariance about a nominal finite-length continuous trajectory.

Recall from Sections 3.1.1 and 3.1.2 that the mapping \( \tilde{F}(\tau) \) can be defined somewhat arbitrarily as long as for each \( \tau \) in the domain of the funnel, we have that \( \tilde{E}(\tau) \subset \text{int}(\tilde{F}(\tau)) \). Therefore, we can compute a certain region of attraction and relate it to the entrance of a funnel, and then define the funnel itself such that the condition \( \tilde{E}(\tau) \subset \text{int}(\tilde{F}(\tau)) \) holds. For instance, if the region of attraction is a sub-level-set \( \rho \) of the Lyapunov function, then this region can be used to define the entrance to the funnel and the funnel itself can be defined using the sub-level-set of \( \rho + \varepsilon \), where \( \varepsilon \) is some positive constant.

Note that in this manner, we can guarantee the separateness of the funnel and its entrance for each value of \( \tau \). Hence, we can ensure robustness, unlike the method in [88]. On the other hand, the computation methods for the regions of attraction comply well with our definitions of an entrance due to the guarantees of invariance they provide. That is, using the techniques in [88], we can find a region \( \mathcal{E} \) which satisfies the definition of an entrance in our framework.

3.2.4 A Funnel about an Equilibrium using SOS Programming

In order to estimate the region of attraction (which will be interpreted here as the entrance to the funnel defined in Section 3.1.1), we need to find a local Lyapunov function \( V(z) \) for the non-linear system. We wish to ensure that trajectories starting form the region of attraction, do not exit it any future time. Hence, we define the region of attraction as such a sub-level set:

\[
B_\varepsilon(\rho) = \{ z | 0 \leq V(z) \leq \rho \}. \tag{3.73}
\]

It is required that in the set \( B_\varepsilon(\rho) \), \( V(z) \) is positive definite and \( \dot{V}(z) \) is negative definite. It is natural to choose cost-to-go function as a local Lyapunov function for the non-linear system, since it is a valid a Lyapunov function for linearised system.
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

Therefore,

\[ V(\tilde{z}) = J^*(\tilde{z}). \]

Now, in order to find the largest region \( B_e(\rho) \), we need to search for the maximum \( \rho \) which satisfies:

\[
\begin{align*}
J^*(\tilde{z}) & \leq 0 \quad \forall \tilde{z} \in B_e(\rho) \setminus \{0\}, \\
J^*(0) & = 0.
\end{align*}
\]  

(3.74) \hspace{1cm} (3.75)

where,

\[ \dot{J}(\tilde{z}) = 2\tilde{z}^T S f(z_e + \tilde{z}, u_e - K \tilde{z}). \]

(3.76)

Then we take three steps to make the negativity constraint (3.74) solvable via an SOS program. First, we transform the negativity constraint (3.74) into a non-positivity constraint using a small positive \( \varepsilon \):

\[ \dot{J}^*(\tilde{z}) \leq -\varepsilon ||\tilde{z}||_2^2 \quad \tilde{z} \in B_e(\rho), \]

(3.77)

Then, we replace the constrained condition (3.77) with a non-constrained condition using a Lagrange multiplier which is a non-negative polynomial variable \( h(z) \):

\[ \dot{J}^*(\tilde{z}) + h(\tilde{z})(\rho - J^*(\tilde{z})) \leq -\varepsilon ||\tilde{z}||_2^2. \]

(3.78)

If for some \( \rho_0 \) we can find \( h(z) \geq 0 \) such that equation (3.78) is satisfied, we obtain that \( B_e(\rho_0) \) is an inner estimation of the region of attraction about \( z_e \).

However, the SOS program still cannot handle such an inequality because in some cases the function \( f \) is not polynomial. Then, as a last step, we use a truncated Taylor expansion to obtain a polynomial approximation of \( f \).

The closed-loop dynamics are:

\[ f^{(d)} = f(z, u_e - K(z - z_e)). \]

(3.79)

The truncated Taylor expansion of \( f^{(d)} \) about \( z_e \) to order \( N_f > 1 \) is denoted \( \hat{f}^{(d)}(\tilde{z}) \)
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

(recall that \( f^{(cl)}(z_e) = 0 \)).

For example, when \( N_f = 2 \), the k-th component of the closed-loop dynamics is approximated by:

\[
\hat{f}^{(cl)}(\tilde{z}) = \sum_i \tilde{z}_i \left[ \frac{\partial f_k^{(cl)}(z)}{\partial z_i} \right]_{z=z_e} + \frac{1}{2} \sum_{i,j} \tilde{z}_i \tilde{z}_j \left[ \frac{\partial^2 f_k^{(cl)}(z)}{\partial z_i \partial z_j} \right]_{z=z_e}
\]  

(3.80)

When using the truncated Taylor expansion to approximate the derivative of the Lyapunov function, we denote the approximation as:

\[
\hat{\dot{J}}(\tilde{z}) = 2 \tilde{z}^T S \hat{f}^{(cl)}(\tilde{z}).
\]  

(3.81)

Now, by fixing the value of \( \rho \) we can attempt to solve the SOS program. If a solution is found (i.e., a SOS function \( h(z) \) is found via the convex search) for some value of \( \rho \), it means that \( B_e(\rho) \) is definitely inside the true region of attraction. By attempting to solve the SOS program iteratively using initially small values of \( \rho \) and increasing them gradually until no positive \( h(z) \) can be found, we can find a larger estimation. This is formalized as:

maximize \( \rho \)  
subject to \( \hat{\dot{J}}^*(\tilde{z}) + h(\tilde{z})(\rho - \hat{\dot{J}}^*(\tilde{z})) \leq -\varepsilon ||\tilde{z}||_2^2 \)

\( \rho \geq 0 \)
\( h(\tilde{z}) \geq 0 \)

(3.82)

3.2.5 A Funnel about a Trajectory using SOS Programming

In this section we look for a time-varying region which defines a finite-time invariance set about a nominal trajectory defined over the finite time-interval \([0, T]\). The method is taken directly from [88]. The time-varying region can be defined as:

\[
B(\rho(\cdot), t) = \{ z | 0 \leq V(z, t) \leq \rho(t) \}.
\]  

(3.83)
3.2. NUMERICAL CONSTRUCTION OF FUNNELS ABOUT EQUILIBRIA AND TRAJECTORIES

where $V(z, t)$ is a positive-definite, radially unbounded function as defined in 3.2.2.

We are interested to design $B(\rho(\cdot), t)$ such that trajectories starting inside it will necessarily arrive at some goal region of the form:

$$B_T = \{ z \mid 0 \leq V(z, T) \leq \rho_T \}. \quad (3.84)$$

That is, we want to guarantee:

$$z(t) \in B(\rho(\cdot), t) \Rightarrow z(T) \in B(\rho(\cdot), T) \quad (3.85)$$

This is useful since in this manner we can ensure that the outlet of some funnel is in the entrance of the funnel to which it connects. To achieve this, we can build similar regions $B$, the one which is computed here using a suitable $\rho(t)$ will define the funnel boundary, whereas the entrance boundary will be defined using $\rho(t) - \varepsilon$, where $\varepsilon$ is a positive number (it can also vary with time).

In order to satisfy the condition (3.85) we look for a time-varying level-set function $\rho(t)$ starting from time $T$. To ensure (3.85), we choose the function $\rho(t) : [0, T] \mapsto \mathbb{R}_{\geq 0}$ with the following conservative properties:

- $\rho(t_f) \leq \rho_f$.
- $\rho(t)$ is discontinuous at finitely many points, $\tau_1, \cdots, \tau_M$, and:

$$\lim_{t \to \tau_m^+} \rho(t) \leq \lim_{t \to \tau_m^-} \rho(t). \quad (3.86)$$

- The right derivative, $\frac{d}{dt_+} \rho(t)$, exists and:

$$V(z, t) = \rho(t) \Rightarrow \dot{V}(z, t) \leq \frac{d}{dt_+} \rho(t) \quad (3.87)$$

This ensures that the value $V(z, t)$ decreases faster along trajectories than its level-set $\rho(t)$, which means that trajectories do not exit the set $B$ once they are inside it. Note that $V(z_0(t), t) = 0$, that is the Lyapunov function is zero on the nominal trajectory.

Again, we choose here to use $V(z, t) = J^*(z, t)$, which is positive definite as the LQR
derivation ensures $S(t)$ is uniformly positive definite. In particular, this gives us:

$$B_f = \{z|0 \leq \hat{z}^TQ_f\hat{z} \leq \rho_f\}. \quad (3.88)$$

Now we have:

$$\dot{J}^*(\hat{z},t) = \hat{z}^T S(t)\hat{z} + 2\hat{z}^T S(t)[f(z_0(t) + \hat{z}, u_0(t) - K(t)\hat{z}) - \dot{z}_0(t)]. \quad (3.89)$$

Even if $f$ is polynomial in $z$ and $u$ and the nominal input signal $u_0(t)$ was polynomial, we cannot ensure that $z_0(t)$, $S(t)$ and $K(t)$ are polynomials. Therefore, we use piecewise truncated Taylor expansions of some order $N_t$ (often $N_t = 3$) to approximate all the non-polynomial values between knot points chosen as the time-steps of the numerical integration, denoted $t_0, t_1, \cdots, t_N$, with $t_N = t_T$, e.g:

$$\forall \in [t_k, t_{k+1}], \quad S_{ij}(t) \approx \sum_{m=0}^{N_t} \alpha_{ijm}(t - t_k)^m = \hat{S}_{ij}(t) \quad (3.90)$$

$$\dot{J}^*(\hat{z},t) = \sum_i \sum_j \hat{z}_i \hat{z}_j \hat{S}_{ij}(t) \quad (3.91)$$

If $f$ is non-polynomial, then $\dot{J}^*$ can be approximated by first computing Taylor expansions in $\hat{z}$ to arrive at polynomials at each knot point, $t_k$, and then interpolate these coefficients of the resulting polynomials as piecewise polynomials of time.

We search for a piecewise-linear function $\rho(t)$ of the form:

$$\rho(t) = \left\{ \begin{array}{ll} \rho_k(t), & \forall t \in [t_k, t_{k+1}) \\ \rho_f, & t_k = t_f \end{array} \right. \quad (3.92)$$

where

$$\rho_k(t) = \beta_{1k}t + \beta_{0k}. \quad (3.93)$$

Higher order functions $\rho$ are harder to compute numerically using SOS programs.

We construct $\rho(t)$ backwards in time for each segment separately, starting with $k = N - 1$. For a given $\rho_k$, it is easy to ensure that $\rho(t_{k+1}) = \beta_{1k}t_{k+1} + \beta_{0k} \leq \rho(t_{k+1})$. We
3.3. CONCLUSION OF CHAPTER 3

now have to ensure that:

\[ \hat{J}^*(\tilde{z},t) = \rho_k(t) \Rightarrow \hat{J}^* \leq \dot{\rho}_k(t) = \beta_{1k} \quad \forall t \in [t_k, t_{k+1}). \] (3.94)

This can be tested by the SOS feasibility program:

\[
\begin{align*}
& \text{find} \quad h_1(\tilde{z},t), \ h_2(\tilde{z},t), \ h_3(\tilde{z},t) \\
& \text{subject to} \quad \hat{J}^*(\tilde{z},t) - \dot{\rho}_k(t) + h_1(\tilde{z},t)(\rho_k - \hat{J}^*(\tilde{z},t)) \\
& \quad + h_2(\tilde{z},t)(t - t_k)h_3(\tilde{z},t)(t_{k+1} - t) \leq -\varepsilon ||\tilde{z}||^2_2 \\
& \quad h_2(\tilde{z},t) \geq 0 \\
& \quad h_3(\tilde{z},t) \geq 0
\end{align*}
\] (3.95)

where the three Lagrange multipliers, \( h_1, h_2 \) and \( h_3 \) are chosen to be polynomials of a sufficient order to account for high order terms in \( \hat{J}^*(\tilde{z},t) \).

Finally, we construct the following optimization problem to allow us to enlarge the region \( B \) as much as we can:

\[
\begin{align*}
& \text{maximize} \quad \rho_k(t_k) = \beta_{1k}t_k + \beta_{0k}, \\
& \text{subject to} \quad \rho_k(t_{k+1}) \leq \rho(t_{k+1}) \\
& \quad \text{SOS Program(3.95)}
\end{align*}
\] (3.96)

3.3 Conclusion of chapter 3

In this chapter we have described six different types of funnels which can be used as building blocks in defining a system-of-funnels based on the control objectives. Some of the funnels have finite-time-domain while others have infinite-time-domains. A typical system of funnels will normally contain funnels of one or more types as described in this chapter, however, other funnel types are also possible to define. We also explained in detail how funnels about equilibrium states and finite-length trajectories can be computed in practice using SOS programs, as was suggested in [88].
In the previous chapter we showed that some types of funnels can be constructed about nominal trajectories. Therefore, as a first stage we to constructing some useful funnels, we present in this chapter how such nominal trajectories can be computed using Bi-directional RRT. In particular, we show several modifications to the nominal RRT algorithm which are designed (based on heuristics) to improve the efficiency of the algorithm for motion planning problems involving non-holonomic constraints and under-actuation. The structure of this chapter is as follows: Section 4.1 presents some relevant background on the modifications of the RRT algorithm on which we rely as a basis for our modifications. Sections 4.2 and 4.3 present the first two modifications. Their effectiveness is discussed in Sections 4.4 and 4.5. Section 4.6 describes the modifications we implemented for the Bi-Directional RRT. Section 4.7 concludes the chapter.

4.1 The RRT Algorithm and its Existing Modifications

The method of RRT aims to grow a search tree in state space by iteratively expanding one of the nodes in the tree towards a randomly sampled state. In order to bias the search towards a target state, the target is occasionally chosen as the sample instead of a random state.

The expansion procedure involves two main stages: (1) the distance between all nodes in the tree and the sample is calculated (or estimated) to find the nearest node, which is then chosen to be expanded (2) an appropriate command torque is then chosen to be applied for a short time duration, either from a discrete set of possible torques or
by analytically calculating the most appropriate torque. A forward integration of the equations of motion is then performed and the resulting final state is added as a node in the tree if the path leading to it does not violate any constraints.

The Reachability Guided RRT (RG-RRT) method [75], builds on the RRT algorithm: it expands the tree towards random samples, but it also rejects some of the samples using a simple but very useful rejection criterion. Every node in the tree holds information of the next set of states towards which it can be expanded in one step, namely, the reachable states. The union of all these states is called “the reachable set”. The rejection criterion for a given sample is if the sample is closer to the tree itself (i.e. any of the nodes comprising it) rather than to the reachable set. This encourages the tree to reach out more rapidly and avoid forming a dense tangle. The logic behind this criterion is that when a node is close to a sample, this does not necessarily imply that expanding the node would grow the tree towards the sample when kinodynamic constraint is present, such as underactuation, which is of particular interest in this thesis.

Conventionally, the Euclidean distance function is used to define how close two states are in most cases found in literature. This is sometimes disadvantageous, as the Euclidean distance between two states is not necessary a good indication for how easy it is to reach one state from the other in an under-actuated system [8]. Clearly, the chance that a certain node of the search tree will develop towards a sample is not determined solely by the Euclidean distance between the node and the sample, but it is also affected by the directionality of the tree at the neighborhood of the node. This observation is illustrated in Fig. 4.1, where two nodes are shown in the neighborhood of a target state. The closest to the target belongs to node B (in terms of the Euclidean distance $d_1$), but the reachable states of node A has more chance of reaching the target state with less computational effort. We would prefer to have more nodes like A in the vicinity of the target state. For this purpose, we introduce a “directionality” term $d_2$ into the distance function in order to encourage the selection of nodes like A instead of nodes like B.

The method proposed in this thesis is based on a geometrical analysis of the states as opposed to other variations to the distance function suggested in the literature, which, in the absence of kinematic constraints, rely mostly on performing an analytical estimation of the optimal ‘cost-to-go’ function [17, 88]. We believe that our approach is both easier to implement (when no optimality criteria are specified) and more time-effective.
4.2 DIRECTED DISTANCE FUNCTION

Figure 4.1: Two nodes, A and B, in a tree and their reachable states. It is clear in this example that it is easier to develop node A to reach the target than node B. The measure of Euclidean distance, however, does not indicate the level of difficulty to reach the target from each node.

in calculating the distance between two states.

4.2 Directed Distance Function

Suppose that a node N in the tree has \( n > 1 \) reachable states and S is a sample state. In order for our algorithm to prefer node A over B in the example illustrated at Fig. 4.1, the standard Euclidean distance function was replaced with the following modified distance function:

\[
D(N, S) = d_1 + d_2
\]

where: \( d_1 = \sqrt{\left\langle N\hat{S}, S\hat{S} \right\rangle} \) is the standard Euclidean distance between the N and S states; and \( d_2 \), which is the directionality term, is defined as the scaled angular shift from N to S:

\[
d_2 = \lambda \cdot \left\{ \begin{array}{ll}
\frac{L^2_{\text{min}}}{W} \min_k [1 - \cos(\theta_k)] & d_1 < \gamma L_{\text{max}} \\
\frac{2L^2_{\text{max}}}{W} & \text{otherwise}
\end{array} \right.
\]

\( \lambda \) is a scaling factor to adjust the magnitude of \( d_2 \) according to the relative difficulty of reaching the target from each node.
where $\gamma$ and $\lambda$ are scalar design parameters; $L_{\text{max}}$ is the distance between $N$ and its furthest reachable state; $L_{\text{mean}}$ is the mean distance between $N$ and all of its reachable states; $\theta_k$ is the angle between the two vectors rooted at $N$ and pointing towards $S$ and the $k^{\text{th}}$ reachable state of $N$ ($1 \leq k \leq n$), $R_k$, respectively:

$$\cos(\theta_k) = \frac{\langle \overrightarrow{NS}, \overrightarrow{NR_k} \rangle}{\sqrt{\langle \overrightarrow{NS}, \overrightarrow{NS} \rangle \langle \overrightarrow{NR_k}, \overrightarrow{NR_k} \rangle}} \tag{4.3}$$

and $W$ is the “opening” length of the node defined as: $W = \sqrt{\langle \overrightarrow{R_1R_n}, \overrightarrow{R_1R_n} \rangle}$.

A disadvantage of this distance function is that it takes substantially more computation time than the simple Euclidean distance function. Note, however, that $d_2$ is replaced by a conservative upper bound when the sample is relatively far from the node and hence the dynamics in the vicinity of $N$ and its reachable states is irrelevant. Note also that $W$, $L_{\text{max}}$, and $L_{\text{mean}}$ are only calculated once when the node is created. In addition, by comparing the coverage area of non-targeted trees over grid-discretised state-space [17], it was observed that using the modified distance function $d_1 + d_2$ makes the tree cover fewer cells in the grid than using the Euclidean distance function $d_1$. Hence, it is more beneficial to use the modified distance function only when the chosen sample, $S$, happens to be the target state. In this manner, computation time is significantly decreased with the tree being well dispersed in state-space, while around the vicinity of the target, it obtains good node density directed towards the target.

4.3 Nested Search

A successful search is generally defined as a one which returns a node within distance $\varepsilon$ to the target, where $\varepsilon$ is the tolerance allowed. If $\varepsilon$ is very small, then a tree with long trajectories between its nodes and their parents is not well suited for this task. On the other hand, very fine trees fail to expand quickly enough to reach various states in the state-space. In order to overcome this dilemma of choosing a suitable time interval between nodes, $\Delta t$, and a maximal number of iterations before stopping the algorithm, a nested search was proposed in order to gradually converge to the target without wasting too much time on very far branches of the tree. The additional motivation stems from
the exponential-time nature of tree-searches: it is much less time-consuming to have \( c \) searches of \( I \) iterations each, than to have a single search with \( cI \) iterations.

The idea behind a nested search is that after a substantial number of nodes reached the neighborhood of the target, the search area (i.e. the portion of the state-space inside which random samples are generated) can be shrunk and only the closest set of nodes was considered for the subsequent iterations. Moreover, \( \Delta t \), and possibly the torque resolution between sibling nodes, \( \Delta u \), may be reduced in magnitude as well.

The main drawback in performing a nested search is the loss of completeness guarantee that holds for the non-nested RRT searches because it might occur that all the nodes in the set used to seed the next stage of the search will never lead to the target. However, this can be resolved by going back to the previous stage and expanding the tree more before reducing the search area. Later on, we show that this undesired scenario is less probable when using the modified distance function presented in Eq. 4.1.

The proposed nested-search algorithm is summarised in Algorithm 1 below. The idea is to define a series of \( P \)-dimensional spherical volumes in the state-space, \( B_i \), of radius \( r_i \) centered about the target, where we define

\[
    r_i = \sqrt{\langle \text{StartTarget}, \text{StartTarget} \rangle} / 2^i, \quad (i \geq 0)
\]  

(4.4)

where \( P \) is the number of states in the system.

The stack named “Tree1” holds all the nodes created in the whole search and the information about their parents for later retrieval of the path. The RG-RRT search [75] (line 6, see Algorithm 4.3.1) is performed at each stage on a temporary stack called “Tree2”. At the end of each refinement stage \( s \) (1 \( \leq s \leq \text{NumOfStages} \)), we insert all the nodes which are within \( B_s \) (i.e., with distance \( d_1 \leq r_s \) from the target) in the stack named “Tree3” (lines 11-13), which is then used to initialize Tree2 in stage \( s + 1 \) (line 14). At the end of each stage of the nested search, the search area, the integration interval and the torque resolution are decreased (provided Tree3 \( \neq \emptyset \)) in order to make the next stage search more focused around the target (Line 16).
Algorithm 4.3.1: Directionality-Guided-Nested-RG-RRT(Start, Target)

1 \{SearchArea; Δt; Δu\} = InitializeSearchParameters() ;
2 \{Tree1; Tree2; Tree3\} = \{Start; Start; Null\} ;
3 InitDist = EuclideanDistance(Start, Target);
4 for \( s = 1 \) to NumOfStages do
5     for \( i = 1 \) to NumOfIterationsPerStage do
6         \{Node\*, k\*\} ← ChooseNodeForExpansionRG-RRT(Tree2) ;
7         \( N \leftarrow \) CreateNewNode(State ← \( R_{K_{Node\*}} \), Parent ← Node\*);
8         \{\( R_{1}^{N} \); ... ; \( R_{n}^{N} \}\} ← CreatePrimaryReachableStates(N);
9         \{\( E_{1}^{N} \); ... ; \( E_{nn}^{N} \}\} ← CreateSecondaryReachableStates(\{\( R_{1}^{N} \); ... ; \( R_{n}^{N} \}\});
10        \{Tree1 ; Tree2\} ← \{Tree1 \cup N; Tree2 \cup N\};
11        if (∥\( N - Target \)∥ < \( \varepsilon \)) then
12            GoTo Line 19 ; % solution found
13            foreach \( N_{j} \in Tree1 \) do
14                if (∥\( N_{j} - Target \)∥ < \( \frac{InitDist}{2} \)) then
15                    Tree3 ← Tree3 \cup N_{j}
16            Tree2 ← Tree3;
17            Tree3 ← Null;
18        \{SearchArea; Δt; Δu\} = ResizeSearchParameters(\{SearchArea; Δt; Δu; Tree1\});
19 return PrintPathFromStartToBest(Tree);
4.4. SIMULATION SET-UP

Line 6 of the algorithm makes use of RG-RRT algorithm to generate an admissible sample and the distance function discussed previously to choose the most appropriate node for development at each iteration. In line 7, a database for the new node $N$ is created and it includes the state of the node, the time it occurs, the torque used to produce it from its parent and a pointer to the parent node. Similarly, $n$ primary reachable states associated with $N$, $R_1^N$, ..., $R_n^N$, are created (line 8). In order to use the RG-RRT method using (4.1) as the distance function, each primary reachable state $R_k^N$ has to have its $n$ secondary reachable states: $E_{k1}^N$, ..., $E_{kn}^N$. All the $n^2$ secondary reachable states associated with node $N$ are created in line 9.

4.4 Simulation Set-up

The algorithm was applied on two systems $^1$:

A **torque limited simple pendulum** with a point mass of $m_1 = 1[Kg]$ at a distance $L_1 = 1[m]$ from the hinge. The task was to move the pendulum state $(\alpha, \dot{\alpha})$ from horizontal rest state $(\frac{\pi}{2}, 0)$ to a chosen target state $(-1, 3)$. The magnitude of the torque was upper bounded at 10[Nm].

An **under-actuated double pendulum** with no actuation at the base hinge, with a point mass $m_1$ at a distance $L_1$ from the base hinge and a point mass $m_2 = 2[Kg]$ at a distance $L_2 = 2[m]$ from the actuated joint. The task was to move the double pendulum to state $(\alpha_1, \dot{\alpha}_1, \alpha_2, \dot{\alpha}_2)$ from horizontal rest state $(0, 0, 0, 0)$ to the vertical downwards rest target state $(-\frac{\pi}{2}, 0, 0, 0)$. No torque upper bound was set for the actuated joint.

For both systems, each node had $n = 3$ primary reachable states (which themselves had $n = 3$ secondary reachable states each) that were produced by three different torques (for the simple pendulum, these were $U = \{+10, 0, -10\} [Nm]$, and for the double pendulum these were $u_{\text{parent}} + U$) applied for the interval $\Delta t$. Initially, $\Delta t$ was set to $0.1[sec]$ and it was halved at every following stage. The torque resolution was not changed across the different stages. The search interval was halved at every stage in each dimension, decreasing the search area by a factor of 4 for the simple pendulum and 16 for the double pendulum.

$^1$A separate reference frame was used for each system.
4.5 Results and Discussion

4.5.1 Torque Limited Simple Pendulum

The resulting distribution of nodes for three consecutive stages is presented in Table 4.1. It is shown that the density of the nodes near the target is greater when using the modified distance function, (i.e. $d_1 + d_2$), and hence the odds of converging to the target with high precision are greater. This trend was observed for different levels of nested search (shown in Table 4.1 for $s = 1, 2, 3$). Note that distance to the target reduces exponentially with $i$ (see Eq. 4.4).

The resulting tree is shown in Fig. 4.2. The overall motion trajectories from the start to the target states are drawn in solid line in Eq. 4.2 (a) and (b) for the two cost functions being compared. Closer views around the target states of the tree are given in Fig. 4.2 (c) and (d), and Fig. 4.3(a)-(d). The effect of the nested search can be seen in Fig. 4.2 (c) and (d) where node density is higher in the later stages of the search, marked by rectangular boxes. It can also be seen that the distribution of nodes is more concentrated around the target state for the modified distance function (Fig. 4.2(c) and Fig. 4.3(a)) than that for the conventional Euclidean distance function (Fig. 4.2(d) and Fig. 4.3(b)). A closer look at the immediate vicinity of the target (Fig. 4.3 (c) and (d)) reveals not only the higher number of nodes near the target state as the result of implementing the proposed modified distance function but also a more accurate arrival at the target point.
4.5.2 Underactuated Double Pendulum

The resulting nodes distribution for the double pendulum system is shown in Table 4.2 across 4 stages of the nested search. Note that at every stage, Tree1 for both cost functions being compared have the same number of nodes. Results similar to that of the simple pendulum were confirmed in this under-actuated scenario, where the density of the nodes is higher near the target for the proposed modified distance function. Furthermore, it is expected that directionality plays an even more important role in under-actuated cases, due to the limitation in the possible directions of actuation (hence direction of local tree growth).

Due to the higher dimension of the system, it is not possible to present the plots of the tree clearly. It can be noted from Table 4.2 that the number of nodes at every stage of nested search is of order(s) of magnitude higher for the proposed modified distance function than that for the conventional Euclidean distance function. Note also that for $s = 3$ for the double pendulum, Tree3 is an empty set for the Euclidean distance function, meaning that no further refinement stage could be attempted without further developing the previous stages, while the tree from the modified distance function is still capable of further refinement.

4.6 The Bi-Directional RRT Algorithm and its Modifications

A Bi-Directional RG-RRT algorithm is a modification of the original one directional algorithm. Instead of growing a single tree towards a target state, two trees are grown simultaneously in the state space with the goal of making the two trees connect. Having two trees instead on one results in more nodes overall, and there exists a risk of the bi-directional algorithm taking more time than the single-directional RRT as a result. However, the bi-directional RRT was empirically shown to improve the efficiency of finding a solution compared with the single-directional RRT algorithm.

In our application the roots of the trees are the start state and the target state respectively which are given for a particular motion planning task. The tree that grows from
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

Table 4.1: Algorithm comparison for the torque-limited pendulum.

<table>
<thead>
<tr>
<th>stage</th>
<th>$s = 1$ ($\Delta t = 0.1$)</th>
<th>$s = 2$ ($\Delta t = 0.05$)</th>
<th>$s = 3$ ($\Delta t = 0.025$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_1$</td>
<td>$d_1 + d_2$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>$i &gt; 0$</td>
<td>910</td>
<td>4,429</td>
<td>4,264</td>
</tr>
<tr>
<td>$i &gt; 1$</td>
<td>198</td>
<td>3,983</td>
<td>1,029</td>
</tr>
<tr>
<td>$i &gt; 2$</td>
<td>49</td>
<td>615</td>
<td>232</td>
</tr>
<tr>
<td>$i &gt; 3$</td>
<td>11</td>
<td>142</td>
<td>58</td>
</tr>
<tr>
<td>$i &gt; 4$</td>
<td>5</td>
<td>46</td>
<td>17</td>
</tr>
<tr>
<td>$i &gt; 5$</td>
<td>1</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>$i &gt; 6$</td>
<td>—</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$i &gt; 7$</td>
<td>—</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>$i &gt; 8$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>$i &gt; 9$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>$i &gt; 10$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

| best Euclidean distance [m] | 0.0416 | 0.0127 | 0.0309 | 0.0127 | 0.0126 | 0.0030 |
| elapsed time [sec] | 50 | 45 | 156 | 169 | 414 | 450 |

Table 4.2: Algorithm comparison for the Acrobot.

<table>
<thead>
<tr>
<th>stage</th>
<th>$s = 1$ ($\Delta t = 0.1$)</th>
<th>$s = 2$ ($\Delta t = 0.05$)</th>
<th>$s = 3$ ($\Delta t = 0.025$)</th>
<th>$s = 4$ ($\Delta t = 0.0125$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_1$</td>
<td>$d_1 + d_2$</td>
<td>$d_1$</td>
<td>$d_1 + d_2$</td>
</tr>
<tr>
<td>$i &gt; 0$</td>
<td>21</td>
<td>6,665</td>
<td>311</td>
<td>32,077</td>
</tr>
<tr>
<td>$i &gt; 1$</td>
<td>—</td>
<td>115</td>
<td>26</td>
<td>2,144</td>
</tr>
<tr>
<td>$i &gt; 2$</td>
<td>—</td>
<td>4</td>
<td>1,482</td>
<td>11</td>
</tr>
<tr>
<td>$i &gt; 3$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>35</td>
</tr>
<tr>
<td>$i &gt; 4$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

| best Euclidean distance [m] | 0.4325 | 0.1538 | 0.2311 | 0.0883 | 0.1571 | 0.0879 | 0.1571 | 0.0879 |
| elapsed time [sec] | 7,987 | 9,055 | 14,011 | 17,560 | 20,645 | 25,368 | 27,674 | 38,829 |
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

Figure 4.2: A comparison between the resulting trees for the torque-limited simple pendulum with NumOfStages = 3. Plots (a), and its magnified view (c), refer to the modified distance function Eq. 4.1, whereas plots (b), and its magnified view (d), refer to the conventional Euclidean distance function. The rectangles represent the three nested search areas (SA) used for the three stages and the circles represent the different $B_i$. The start states and target states (which is $2\pi$-periodic) are marked with bold square and circles, respectively. The best path found is shown in solid line.

the initial state is grown using the dynamics described by $\dot{x} = f(x, u)$ and is called the forward tree. The tree that grows from the target state is grown using the backward in time dynamics described by $\dot{x} = -f(x, u)$ (since the system is time invariant) and is called the backward tree.

In this work we implement an approach to Bi-Directional RRT which is different to
Figure 4.3: The magnified views of Fig. 4.2 in the vicinity of the target. The accurate distribution of nodes in the various $B_i$ for the two cost functions at each stage of the search is presented in Table 4.1.

We define 2 new terms, *mate*, and *best couple*. The *mate* of a node is a node from the reciprocal tree that is nearest to it. The *best couple* stores the two nodes in the two trees which have the shortest distance and the distance between them in a given iteration.
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

The structure of a node is:

\{node\_state, torque\_from\_parent, time, parent, mate, direction, IsSampleTarget\}.

The structure of a couple is:

\{n1\_state, n2\_state, distance\}.

The main algorithm appears in Algorithm 4.6.1. Initially the program parameters are set, the trees are created, the start and target nodes are created and added to the relevant trees (see Line 1 in Algorithm 4.6.1). Algorithm 4.6.2 describes the procedure of initialization.

Lines 2-12 in Algorithm 4.6.1 iterate the different stages in which the algorithm is run in the same fashion as in Algorithm 4.3.1 such that at the end of each stage, the PrepareForNewStage() function is called and the parameters are adjusted for a more refined later stage. The function PrepareForNewStage() is described in Algorithm 4.6.6. At each stage a predefined number of iterations is used to develop both trees simultaneously. That is the forward tree is expanded and then the backward tree is expanded. This comprises 1 iteration (lines 3-11). The variable dir in line 4 refers to the current tree direction. When dir = 0, the forward tree is expanded, and when dir = 1, the backward tree is expanded.

Line 5 in Algorithm 4.6.1 generates a node for expansion in a similar manner as in line 6 of Algorithm 4.3.1. The function ChooseNodeForExpansionRG – RRT() is explained in more detail in Algorithm 4.6.3. In line 6, a new node is generated using the CreateNewNode() function which is described in Algorithm 4.6.4. If the new node satisfies the constraints of the problem, it is then added to the appropriate trees (lines 7-8). Creating a new node may require an update to the best couple. This is done in line 9 and described in Algorithm 4.6.5.

Lines 10-11 of Algorithm 4.6.1 provide early termination of the algorithm in the case that the best\_couple distance less than some predefined threshold \(\varepsilon\).
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

Algorithm 4.6.1: Directionality-Guided-Nested-Bi-RG-RRT

1 AlgorithmInitialisationAndSettings();
2 for $s = 1$ to NumberOfStages do
3     for $i = 1$ to NumberOfIterations do
4         for $d = 0$ to 1 do
5             \{ $N_{near}$, $R_{best}$, $IsSampleTarget$ \} ←—
6             ChooseNodeForExpansionRG-RRT($Tree2[0]$, $Tree2[1]$, $dir$, ...);
7             $N$ ←—
8             CreateNewNode($N_{near}$, $R_{best}$, $Tree2[1-dir]$, $dir$, $IsSampleTarget$) ;
9             if $N \neq NULL$ then
10                \{ $Tree1[dir]$ ; $Tree2[dir]$ \} ←— \{ $Tree1[dir]$ $\cup$ $N$; $Tree2[dir] \cup N$ \} ;
11                UpdateBestCouple($N$) ;
12                if $||best\_couple.dist\_to\_mate|| < \epsilon$ then
13                    Break. Solution Found.
14             end if
15         end if
16     end for
17 end for
18 prepareForNewStage($AllTrees$, boundaries, $\Delta t$, $\Delta u$);
19 return PrintPaths($Tree[0]$, $Tree[1]$)

In the initialization (Algorithm 4.6.2) some data of the system is defined: the system dynamics that describe the equations of motion, the possible torques for expansion of reachable states (the number of different torques and the range), the initial integration time to produce reachable states from a given node, the initial search boundaries which define a box in the state space which will be centred about different nodes of the tree within which random samples are generated (note that the torque values, the integration time and the search boundaries are nominal, and are prone to changes during the running of the algorithm), the number of stages which determines how many refinements are made, and the number of iterations for development of trees at each stage. Task related data is also defined: Initial and target states which are the root of the forward and backward trees respectively as well as the initial and target torques.

After these definitions (Line 1 in Algorithm 4.6.2) 3 trees are defined for each root node in lines 2-3. That is $Tree1[0]$, $Tree2[0]$, $Tree3[0]$ from the start state and $Tree1[1]$, $Tree2[1]$, $Tree3[1]$ from the target state. The functionality of these trees are identi-
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

cal to the functionality of the Trees in Algorithm 4.3.1: Tree1[0] and Tree1[1] store the whole trees developed so far from the start state and the target state respectively. Tree2[0] and Tree2[1] are temporary subsets of Tree1[0] and Tree1[1] respectively which store the nodes which are considered for development in the current stage. Tree3[0] and Tree3[1] are occupied at the end of every stage with the nodes (also called leaves) that will be used for development in the next stage. Initially all the trees are empty.

The variable NumOfLeaves is an ordered pair which stores the number of leaves (i.e. size of Tree3 from the previous stage) for each direction (line 4). In line 5 the start node, Nstart, is defined as a root node without a parent and without a mate. In line 6 the target node, Nstart, is defined without a parent and the already created Nstart as its mate. In line 7 Nstart can now be assigned as the mate of Nstart. Then in lines 8-9 Nstart is used to populate Tree1[0], Tree2[0] (the forward tree) and Ntarget is used to populate trees Tree1[1] and Tree2[1] (the backward tree). The Euclidean distance between the start and target nodes, init_dist, is defined in line 10 and is used later in order to decide which nodes will be considered as leaves in each iteration. The initial best_couple is defined in line 11 using the start and target nodes and their distance. This variable will also be updated as the trees are developed.

**Algorithm 4.6.2: AlgorithmInitialisationAndSettings()**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>input:</strong> System Dynamics, Set of torques, Initial Δt, Initial Search Boundaries, NumberOfStages, NumberOfIterations, Initial and Target States, Initial and Target Torques ;</td>
</tr>
<tr>
<td>2</td>
<td><code>{Tree1[0], Tree2[0], Tree3[0]} ← {}</code> ;</td>
</tr>
<tr>
<td>3</td>
<td><code>{Tree1[1], Tree2[1], Tree3[1]} ← {}</code> ;</td>
</tr>
<tr>
<td>4</td>
<td>NumOfLeaves ← {1, 1} ;</td>
</tr>
<tr>
<td>5</td>
<td>Nstart ← {xstart, ustart, 0, Null, Null, 0, 0} ;</td>
</tr>
<tr>
<td>6</td>
<td>Ntarget ← {xtarget, utarget, 0, Null, Nstart, 0, 0} ;</td>
</tr>
<tr>
<td>7</td>
<td>Nstart.mate ← Ntarget ;</td>
</tr>
<tr>
<td>8</td>
<td>Tree1[0].push_back(Nstart), Tree2[0].push_back(Nstart) ;</td>
</tr>
<tr>
<td>9</td>
<td>Tree1[1].push_back(Ntarget), Tree2[1].push_back(Ntarget) ;</td>
</tr>
<tr>
<td>10</td>
<td>init_dist = EuclideanDistance(Nstart, Ntarget) ;</td>
</tr>
<tr>
<td>11</td>
<td>best_couple ← {Nstart, Ntarget, init_dist} ;</td>
</tr>
</tbody>
</table>

When a tree of direction dir is expanded one of its nodes is chosen for expansion using
the function ChooseNodeForExpansionRG – RRT() (Algorithm 4.6.3). This algorithm receives Tree2[0], Tree2[1] and the direction dir of the tree to be expanded as well as the current boundaries and the current best couple. The RG-RRT idea is implemented in order to reject samples that are close to nodes of the tree than to reachable states of the tree. The variable IsTreeClosest, which is initially set to TRUE in line 2 of Algorithm 4.6.3 is a flag which indicates if the sample is closer to the nodes of the tree. Lines 3-5 in Algorithm 4.6.3 generate new samples about random nodes in Tree2[dir − 1] until a sample for which IsTreeClosest = FALSE is found. At the first stage a random node is generated from Tree2[dir − 1] and this can be either a node from the best couple (in which case IsSampleTarget = TRUE since we attempt to bias the search in a manner that will create a connection between the couple that is most likely to result in a connection at this stage) or another node from the tree in (in which case IsSampleTarget = FALSE). If the chosen node is from the best couple then its state is used as the random state, $x_{rand}$ otherwise a random state is chosen from a box in state space defined about the node using the current boundaries.

Once the state is chosen the NearestNeighbourRG – RRT() function is used to find which node in Tree2[dir] is closest to the sample $x_{rand}$. The nearest node is $N_{near}$ and its reachable state which is nearest the sample is $R_{best}$. If the distance between $R_{best}$ and the sample is smaller than the distance from $N_{near}$ to the sample IsTreeClosest = FALSE and the node and its best reachable state are returned in line 6. If the sample is from the best couple then looking for the nearest neighbour uses the modified distance metric in Eq. (4.1) otherwise the Euclidean distance is used.

<table>
<thead>
<tr>
<th>Algorithm 4.6.3: ChooseNodeForExpansionRG-RRT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> input: Tree2[0], Tree2[1], dir, boundaries, best couple ;</td>
</tr>
<tr>
<td><strong>2</strong> IsTreeClosest = TRUE;</td>
</tr>
<tr>
<td><strong>3</strong> while IsTreeClosest do</td>
</tr>
<tr>
<td><strong>4</strong> { $x_{rand}$, IsSampleTarget } ← RandomState(best couple, Tree2[dir − 1], boundaries) ;</td>
</tr>
<tr>
<td><strong>5</strong> { $N_{near}$, IsTreeClosest, $R_{best}$ } ← NearestNeighbourRGRRT($x_{rand}$, Tree2[dir], IsSampleTarget) ;</td>
</tr>
<tr>
<td><strong>6</strong> return $N_{near}$, $R_{best}$, IsSampleTarget</td>
</tr>
</tbody>
</table>

After a node has been chosen for expansion, the algorithm attempts to create a new
node, \( N_{\text{near}} \) (see Algorithm 4.6.4). The \texttt{CreateNewNode()} function receives the node \( N_{\text{near}} \), the reachable state \( R_{\text{best}} \) which is considered to become a new node in the tree, the current direction \( \text{dir} \) and the flag \textit{IsSampleTarget}. In line 2 the reachable state \( R_{\text{best}} \) is invalidated for the node \( N_{\text{near}} \) which means \( R_{\text{best}} \) will never again be considered as a reachable state of \( N_{\text{near}} \). In line 3 the algorithm checks whether or not \( R_{\text{best}} \) (or the path from \( N_{\text{near}} \) to \( R_{\text{best}} \)) complies with the constraints of the problem. If it does then it is created as a new node (line 4) otherwise it is assigned an empty pointer (line 21). Upon creation of the new node (line 4) the structure of the node is used: the state of the node is \( N_{\text{near}}.R_{\text{best}}.\text{state} \), the torque is \( R_{\text{best}}.\text{torque} \) which defines the torque signal which was used to generate \( R_{\text{best}} \) from its parent, the time in the tree is \( N_{\text{near}}.\text{reachable\_time} \) which is an accumulated time required to implement the trajectory that connects the root of the tree to \( R_{\text{best}} \), \( N_{\text{near}} \) is the parent node, and \( N_{\text{near}}.\text{mate} \) is set as our current guess for the best mate which is later possibly updated, the direction \( \text{dir} \) defines the dynamics with which the reachable states of the new node are calculated, the \textit{IsSampleTarget} flag defines which method of calculation will be used for computing the reachable states.

If \textit{IsSampleTarget} = FALSE then the current values of \( \Delta t \) and the set of torques \( \Delta u \) are used to define the reachable states. However, if \textit{IsSampleTarget} = TRUE which means it is more important to hit the target, alternative values are potentially used for computation of the reachable states. In that case, initially the reachable states are calculated as usual and then are potentially corrected using the following approach. Instead of the current integration time \( \Delta t \) we use

\[
\Delta t' = \alpha \Delta t, \\
\Delta u' = \alpha \Delta u,
\]

where \( \alpha \) is a scaling factor used to compute a more refined reachable state and is calculated according to

\[
\alpha = \begin{cases} 
1.0 & \beta > 1.2, \\
0.8\beta + 0.04 & 0.2 < \beta < 1.2, \\
0.24 & \beta < 0.25,
\end{cases}
\]

where \( \beta \) is a decision variable based on the ratio between the distance from the initially attempted reachable state to the target and the distance from the new node to the
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

target. In addition upon creation of $N_{\text{new}}$ some parameters are calculated such as the
distance to the current mate.

After the node is created it may be the case that some nodes in the opposite tree will
have to be assigned a new mate and the mate of the newly created node may have to
be updated. In order to avoid an overly exhaustive computation of all the distances
between the new node and all the nodes of the other tree we introduce the following
strategy for updating mates. Define the following distances (see Table 4.3):

<table>
<thead>
<tr>
<th>Distance</th>
<th>From -- to --</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>$N_{\text{near}}$ to $N_{\text{new}}$</td>
<td>Upperbounded by $L_{\text{max}}$ which is known</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$N_{\text{near}}$ to $N_{\text{near.mate}}$</td>
<td>Accessible</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$N_{\text{new}}$ to $N_{\text{new.mate}}$</td>
<td>Accessible (updated as the algorithm progresses)</td>
</tr>
<tr>
<td>$d_4$</td>
<td>$N_{\text{new}}$ to some node $W$ from the other tree</td>
<td>Not Accessible</td>
</tr>
<tr>
<td>$d_5$</td>
<td>$W$ to $W.mate$</td>
<td>Accessible</td>
</tr>
<tr>
<td>$d_6$</td>
<td>$N_{\text{near}}$ to $W$</td>
<td>Not Accessible</td>
</tr>
</tbody>
</table>

Table 4.3: Distances

In lines 5-7 $d_2$, $d_3$ and $L_{\text{max}}$ are retrieved (see Table 4.3). In line 8 the variable
$\text{best_dist_to_mate}$ is initialized to the distance to the mate between $N_{\text{new}}$ and its mate.
In lines 9-19 the algorithm passes through all of the nodes in $\text{Tree2}[1 - \text{dir}]$ and for
each node $W$ in $\text{Tree2}[1 - \text{dir}]$ we retrieve $d_5$ (line 10) and if necessary we update the
mate of $N_{\text{new}}$ (lines 11-15) and the mate of $W$ (lines 16-19). We now explain the logic of
determining if the distance between the new node $N_{\text{new}}$ and $W$, $d_4$, has to be calculated.

Note that the mate of $W$ has to be changed if

$$d_5 > d_4$$

In addition the lowest possible value of $d_4$, $d^*_4$ is

$$d^*_4 = d_6 - d_1.$$
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

Since \( d_6 > d_2 \) (by definition of a mate as the nearest node of the opposite tree) and \( d_1 < L_{\text{max}} \) we get that

\[
d_4 \geq d_4^* \geq d_6 - d_1 \geq d_2 - L_{\text{max}}
\]

This means that if

\[
d_5 \leq d_2 - L_{\text{max}}
\]

then surely

\[
d_5 \leq d_4
\]

and there is no need to calculate \( d_4 \) for the node \( W \) which may save computation time (line 15). If however

\[
d_5 \geq d_2 - L_{\text{max}}
\]

we calculate \( d_4 \) (line 16) and compare to \( d_5 \). If \( d_5 > d_4 \) we need to set the mate of \( W \) to be \( N_{\text{new}} \) (lines 17-18).

In order for \( W \) to be the best mate of \( N_{\text{new}} \) we need

\[
d_3 > d_4
\]

We show that if

\[
d_5 \geq L_{\text{max}} + d_3
\]

then it is certain that \( d_4 \geq d_3 \) and there is no need to change \( N_{\text{new}}.\text{mate} \) to be \( W \). If

\[
d_5 \geq L_{\text{max}} + d_3
\]

then

\[
d_5 \geq d_1 + d_3
\]

and therefore (since \( d_6 \geq d_5 \)):

\[
d_6 \geq d_1 + d_3
\]

Looking at the triangle from the 3 nodes \( N_{\text{new}}, N_{\text{near}} \) and \( W \) we get:

\[
d_1 + d_4 \geq d_6 \geq d_1 + d_3
\]

and therefore

\[
d_4 \geq d_3.
\]
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

If \( d_5 < L_{\text{max}} + d_3 \) (line 10) then we do need to calculate \( d_4 \) (line 11) and if it is smaller than the current best mate we make \( N_{\text{new}.\text{mate}} = W \) and update the value of \( d_3 \) (lines 12-14). Note that it the current form \( d_4 \) may get calculated twice. This is undesirable so we can place a check to see if \( d_4 \) has already been calculated before doing it again.

**Algorithm 4.6.4: CreateNewNode**

```plaintext
input: \( N_{\text{near}}, R_{\text{best}}, Tree2[1 - \text{dir}], \text{IsSampleTarget} \);  
\( N_{\text{near}.R_{\text{best}}}.isValid \leftarrow \text{FALSE} \);  
if CheckKinematicConstraints \( (N_{\text{near}.R_{\text{best}}}.state) = \text{TRUE} \) then  
    \( N_{\text{new}} \leftarrow \{N_{\text{near}.R_{\text{best}}.state, R_{\text{best}}.torque, N_{\text{near}}.\text{reachable_time}, N_{\text{near}}, N_{\text{near}.\text{mate}}, \text{dir, IsSampleTarget}\} \);  
    \( d_2 = N_{\text{near}}.\text{dist_to_mate} \);  
    \( d_3 = N_{\text{new}}.\text{dist_to_mate} \);  
    \( L_{\text{max}} = N_{\text{near}}.L_{\text{max}} \);  
    foreach \( W \) in Tree2[1 - \text{dir}] do  
        \( d_5 = W.\text{dist_to_mate} \);  
        if \( d_5 < L_{\text{max}} + d_3 \) then  
            \( d_4 = \text{EuclideanDistance}(N_{\text{new}.state, W.state}) \);  
            if \( d_4 < \text{best.dist_to_mate} \) then  
                \( N_{\text{new}.\text{mate}} \leftarrow W \);  
                \( d_3 = d_4 \);  
        else  
            if \( d_5 > d_2 - L_{\text{max}} \) then  
                \( d_4 = \text{EuclideanDistance}(N_{\text{new}.state, W.state}) \);  
                if \( d_4 < d_5 \) then  
                    \( W.\text{mate} \leftarrow N_{\text{new}} \);  
        else  
            \( N_{\text{new}} = \text{NULL} \);  
    end  
end  
return \( N_{\text{new}} \)
```

After the new node, \( N_{\text{new}} \) is created, it is added into \( Tree1[\text{dir}] \) and \( Tree2[\text{dir}] \) (recall line 8 in Algorithm 4.6.1. It may be necessary to update the best couple. This is done using the \( \text{UpdateBestCouple()} \) function (Algorithm 4.6.5) which gets \( N_{\text{new}} \) and the current \textit{best.couple} as input data. It then checks whether the distance from the new node to its best mate is smaller than the distance between the two nodes which form the current \textit{best.couple} and if yes, updates the couple by replacing the old \textit{best.couple} by \( N_{\text{new}} \) and its mate \( N_{\text{new}.\text{mate}} \) as well as the distance.
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

**Algorithm 4.6.5: UpdateBestCouple**

1. `input: N_{new}, best_{couple}`
2. `if N_{new}.dist_to_mate < best_{couple}.dist_to_mate then`
3.  `if dir = 0 then`
4.     `best_{couple} ← {N_{new}, N_{new}.mate, N_{new}.dist_to_mate}`
5. `return best_{couple}`

The function `PrepareForNewStage()` is called in order to change the storage of the trees as well as resize the nominal values of $\Delta t, \Delta u$ and the search boundaries prior to initiating the next refinement stage of the algorithm (see Algorithm 4.6.6). Lines 3-8 of the algorithm change the nodes in the trees for each of the two directions: first all the leaves (that is the nodes for which the distance to their mate is smaller than $\frac{\text{init\_dist}}{2^s}$; see Equation 4.4) are entered to $Tree3$ (lines 4-5) and the number of leaves is updated (line 6). $Tree2$ is replaced with $Tree3$ so that in the next stage only nodes which are in the current leaves will be used for development and $Tree3$ is emptied (lines 7-8). In line 9 the `boundaries`, $\Delta t$, $\Delta u$ are modified in order to refine the search. In our implementation we simply reduced the `boundaries` and $\Delta t$ by factor of 2 each stage and $\Delta u$ was left unchanged.

**Algorithm 4.6.6: PrepareForNewStage**

1. `input: Tree1[0], Tree2[0], Tree3[0], Tree1[1], Tree2[1], Tree3[1], boundaries, $\Delta t, \Delta u$`
2. `for dir = 0 to 1 do`
3.     `foreach N_j in Tree1[dir] do`
4.         `if ||N_j.dist_to_mate|| < $\frac{\text{init\_dist}}{2^s}$ then`
5.             `Tree3[dir].push_back(N_j);`
7.     `Tree2[dir] ← Tree3[dir];`
8.     `Tree3[dir] ← NULL;`
9.     `{boundaries; $\Delta t; \Delta u}` = ResizeSearchParameters({`boundaries; $\Delta t; \Delta u`});
4.6. THE BI-DIRECTIONAL RRT ALGORITHM AND ITS MODIFICATIONS

Figure 4.4: Illustration of how mates are updated when a new node is created.
4.7 Conclusion

This chapter presented a computationally efficient method to estimate the reachability of states by biasing the choice of nodes towards a target state utilising the knowledge of the system dynamics. Additionally, a nested search technique was proposed in order to focus the search more rapidly around the target. It was demonstrated that the modified distance function which comprises both a Euclidean distance term and a directionality term was more successful in directing the search towards the target. By comparing two trees, one grown with the Euclidean distance function and the other with the modified one, it was shown that there were more nodes in the vicinity of the target for the latter case. Moreover, the directionality term, in contrast to the Euclidean term, helps develop the tree nodes in accordance to the knowledge of the system dynamics, which is highly advantageous in the case of under-actuated systems. In addition, we presented in detail our implementation of a bi-directional RRT algorithm, which includes modifications to the current bi-directional implementations found in the literature. A detailed investigation of the effectiveness of the suggested bi-directional RRT algorithm compared to the nominal bi-directional one is left for future work.
This chapter provides the illustrative examples of the construction of the systems of funnels investigated in this thesis. Example 1 demonstrates the concept using an infinite time domain funnel in a system of two funnels. Example 2 illustrates the construction of a system of four funnels, connected head to tail to form a periodic motion. Example 3 provides a more realistic example of a dynamic system (mechanism) of a simple pendulum and demonstrates how the desired behaviour is achieved using our framework.

5.1 Example 1: Two Funnels Example

This example demonstrates a system of two funnels and bootstrap control. The dynamical system is:

\[ \dot{z} = u \]  

(5.1)

where \( z \in \mathbb{R} \) and \( u \in \mathbb{R} \).

The system of funnels is composed of two funnels: \( \mathcal{F}_1 \) with a final time domain and \( \mathcal{F}_2 \) with an infinite time domain. The funnels and their entrances, together with their respective domains, are defined below along with the respective controllers:

\[ \mathbf{F}_1(\tau) := \{ z \mid |z| \leq 4 - \frac{1}{3} \tau \} \quad \text{,} \quad \mathcal{F}_1 := \text{gph}(\mathbf{F}_1) \]  

(5.2)

\[ \mathbf{E}_1(\tau) := \{ z \mid |z| \leq 3 - \frac{1}{3} \tau \} \quad \text{,} \quad \mathcal{E}_1 := \text{gph}(\mathbf{E}_1) \]  

(5.3)
5.1. EXAMPLE 1: TWO FUNNELS EXAMPLE

\[
\text{dom}(F_1) := [0, 6] \quad (5.4)
\]

\[
\text{dom}(E_1) := [0, 6] \quad (5.5)
\]

\[
u_1(\tau, z) := -z + \frac{5}{2} \sin(\tau) \quad (5.6)
\]

\[
F_2(\tau) := \{ z \mid |z| \leq 3 \} \quad , \quad F_2 := \text{gph}(F_2) \quad (5.7)
\]

\[
E_2(\tau) := \{ z \mid |z| \leq 2 \} \quad , \quad E_2 := \text{gph}(E_2) \quad (5.8)
\]

\[
\text{dom}(F_2) := [0, \infty) \quad (5.9)
\]

\[
\text{dom}(E_2) := [0, 3] \quad (5.10)
\]

\[
u_2(\tau, z) := \frac{2}{5} \sin(\tau) \quad (5.11)
\]

The set of edges (showing funnel connections) is:

\[
E := \{(1, 1), (1, 2)\} \quad (5.12)
\]

The bootstrap controller is defined as:

\[
u_0 = -z \quad (5.13)
\]

Notice that \( O_1 := \{ (\tau, z) \mid \tau = 6, |z| < 2 \} \) is such that there exist \( r_1 \) and \( r_2 \) such that \((r_k, z) \in E_k\) for \( k \in \{1, 2\} \). Therefore, when a trajectory of \( x \) hits \( O_1 \), it can jump either to \( F_1 \) or \( F_2 \). This is implemented as a random choice with an even distribution. Also, whenever there is a range of available values for \( \tau \), one value is randomly chosen with an even distribution between the allowed values.

The trajectories of the system plotted for several different initial conditions. Each simulation was limited to up to 30 seconds of flow and up to 10 jumps.
5.2. EXAMPLE 2: FOUR FUNNELS EXAMPLE

The following examples show four cases of initial conditions leading to different scenarios:

1. The initial conditions are such that $\ell_0 = 1$ (i.e., bootstrap control is off), $k_0 = 1$ (i.e., referring to $F_1$), $\tau_0 = 0$ and $z_0 = -3$.

This example, depicted in Fig. 5.1a, shows normal operation of the dynamical system inside the system of funnels. The trajectory of $\xi$ starts inside $E_1$ and remains within $F_1$ until it hits $O_1$, then jumps to $E_2$, and remains forever inside $F_2$.

2. The initial conditions are such that $\ell_0 = 1$ (i.e., bootstrap control is off), $k_0 = 2$ (i.e., referring to $F_2$), $\tau_0 = 1.5$ and $z_0 = 5$.

In this example, depicted in Fig. 5.1b, the initial conditions are such that $\xi \notin F_2$. Therefore, at $t = 0$ there occurs a jump such that $\ell$ is set to 0 after the jump, that is, the bootstrap control is turned on. The bootstrap control operates until $G_0(\xi) \neq \emptyset$, which happens after $t = 0.5108$ seconds, when $z = 3$. At that instant, another jump occurs, when now $\tau$ is set to 0 (this is the only possible value for this value of $z$), $k$ is set to 1, and $\ell$ is set to 1. After that, $\xi$ flows in $F_1$ until hitting $O_1$. This time, the jump is into $E_1$ again. The next time that $\xi$ hits $O_1$, the jump is into $E_2$ as in the first example.

3. Fig. 5.2a shows an example where the initial condition is inside a funnel but outside its entrance, such that it escapes the funnel before reaching the outlet when $\xi = (3.3 + 1.91, -2.263)$. As a result, the bootstrap control is initiated but then instantly turned off again as $z = -2.263$ is such that $G_0$ is non-empty. The initial conditions are $\tau_0 = 3.3$, $z_0 = -2.8$, $k_0 = 1$ and $\ell_0 = 1$. The rest of the motion is similar to the previous case.

4. This case, shown in Fig. 5.2b is similar to the previous one, except for the initial setting of $k_0$ to be 2.

5.2 Example 2: Four Funnels Example

This example demonstrates a system of four funnels connected “head to tail” (in order to form a periodic motion) and bootstrap control.
The dynamical system is:
\[ \dot{z} = Az + u, \]
where \( z \in \mathbb{R}^2 \), \( u \in \mathbb{R}^2 \) and
\[ A = \begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix}. \]
Note that \( A \) is Hurwitz with eigenvalues \( \lambda_{1,2} = -1 \pm j \).
The system of funnels is composed of four similar funnels, all with a finite time domain.
We have \( \Sigma = \{ \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4 \} \) and the connections are described by the set of edges:
\( E = \{(1, 2), (2, 3), (3, 4), (4, 1)\} \).
5.2. EXAMPLE 2: FOUR FUNNELS EXAMPLE

The control inputs for the four funnels are: \( u_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad u_2 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \quad u_3 = \begin{pmatrix} -2 \\ 0 \end{pmatrix}, \) 
\[ u_4 = \begin{pmatrix} 0 \\ -2 \end{pmatrix}. \]

Applying each of these control inputs to the linear system (5.14) asymptotically stabilizes the states \( \bar{z}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \bar{z}_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad \bar{z}_3 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad \bar{z}_4 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \) respectively.

We describe below the geometry of funnel \( \mathcal{F}_1 \) and its entrance \( \mathcal{E}_1 \). The other funnels are constructed in a similar manner.

Normally, funnel \( \mathcal{F}_1 \) brings states towards its equilibrium state \( \bar{z}_1 \) from states arriving from the outlet of funnel \( \mathcal{F}_4 \).
5.2. EXAMPLE 2: FOUR FUNNELS EXAMPLE

We define the entrance $E_1$ as a disk of radius 0.25 about $\bar{z}_4$:

$$E_1 := \{ (\tau, z) \mid \tau = 0, \ |z - \bar{z}_4| \leq 0.25 \}. \quad (5.15)$$

In order to define the funnel, we first define a set of initial conditions as a disk of radius 0.5 about $\bar{z}_4$:

$$S_1 := \{ z \mid |z - \bar{z}_4| \leq 0.5 \} \quad (5.16)$$

Let the domain of the funnel be:

$$\text{dom}(F_1) := [0, 4] \quad (5.17)$$

Let $\phi_1(\tau, z_0)$ be a solution of (5.14) with $u = u_1$ starting from the initial condition $z_0$. The mapping $F_1$ is defined as:

$$F_1(\tau) := \{ z \mid z = \phi_1(\tau, z_0), \ z_0 \in S_1 \}, \ \mathcal{F}_1 := \text{gph}(F_1) \quad (5.18)$$

The boundaries of the four funnels and the boundaries of their entrances are depicted below in several different views (see Fig. 5.3-5.4): In blue - $\mathcal{F}_1$, in magenta - $\mathcal{F}_2$, in cyan - $\mathcal{F}_3$, in green - $\mathcal{F}_2$. 


5.2. EXAMPLE 2: FOUR FUNNELS EXAMPLE

Figure 5.3: Top View
5.2. EXAMPLE 2: FOUR FUNNELS EXAMPLE

The bootstrap control $u_0$ is arbitrarily chosen to be any of the control policies $u_k$, $k \in K := \{1, 2, 3, 4\}$. The choice is made as a random sample from an even distribution.

Since the system is linear and we consider no disturbances, starting within the funnel cannot make the trajectory escape it and it is clear from the figures that the each outlet is contained in the subsequent entrance (after setting $\tau$ to zero).

We simulate the solution of the hybrid system for 12 consecutive jumps and 13 flows. A top view (Fig. 5.5a) and a 3D view (Fig. 5.5b) of the resulting hybrid trajectory are plotted in the figures below. The colors represent the funnel in which the trajectory is in as before, and a black color is for the trajectory obtained with the bootstrap controller.
5.3. **Example 3: Wing Swing Motion - Task Description**

We would like to demonstrate some basic concepts of our framework on a toy-problem which aims to achieve a continuous wing-swing-like motion for a simple pendulum. The task is to induce a repetitive back-and-forth motion from a low angle of $\theta_L := 30^\circ$ to a high angle of $\theta_H := 150^\circ$ from any initial condition in the state space of the simple pendulum.

For simplicity, we want to use a bootstrap control of no actuation at all: $u_0 = 0$. Therefore, we consider the top and bottom rest configurations, corresponding to $\theta_\pi := 180^\circ$ and $\theta_0 := 0^\circ$. Fig. 5.6 shows the four configurations of interest. In order to create the repetitive swinging motion, we will choose to plan motion from $\theta_L$ with zero velocity to $\theta_H$ with zero velocity and vice versa. When the bootstrap control will be applied, the states with $\theta_0$ or $\theta_\pi$ angle and zero velocity can occur, and hence, we also choose to
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

\[ \theta_H := 150^\circ \]
\[ \theta_\pi := 180^\circ \]
\[ \theta_L := 30^\circ \]
\[ \theta_0 := 0^\circ \]

Figure 5.6: Four configurations for the design of the system of funnels.

plan motion from these states to the state of \( \theta_L \) angle and zero velocity. Clearly, other choices of motion planning could be done to achieve qualitatively the same steady-state behaviour.

5.3.1 Simple Pendulum Model

We start by introducing the dynamic system of the simple pendulum. We choose the same model as described in [88]. The equation of motion of the simple pendulum (with zero position in the downwards vertical configuration and frictional losses) is given by:

The dynamics of a

\[ I\ddot{\theta} + b\dot{\theta} + mgl\sin(\theta) = \tau, \tag{5.19} \]

where \( b \) is the friction term, \( m \) is the pendulum mass, \( l \) is the pendulum length, \( g \) is gravity constant and \( I \) is the moment of inertia about the joint. The values of the parameters used are summarised in Table 5.1.

The state of the system is defined as \( z := \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \in \mathbb{R}^2 \), and the input is \( u := \tau \in \mathbb{R} \).

Note that the both of the angles \( \theta_L \) and \( \theta_H \) can maintain their values in equilibrium when the input torque is \( 0.5mgl \).
5.3. Example 3: Wing Swing Motion - Task Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>1.0</td>
<td>Kg</td>
</tr>
<tr>
<td>l</td>
<td>0.5</td>
<td>m</td>
</tr>
<tr>
<td>b</td>
<td>0.1</td>
<td>Kg·m²/s</td>
</tr>
<tr>
<td>l</td>
<td>ml²</td>
<td>Kg·m²</td>
</tr>
<tr>
<td>g</td>
<td>9.8</td>
<td>m/s²</td>
</tr>
</tbody>
</table>

Table 5.1: Model Parameters

5.3.2 Obtaining Nominal Trajectories

We would like to design funnels about four nominal final-length trajectories:

1. A trajectory from $z_{\text{start}} = (\theta_0, 0)$ to $z_{\text{target}} = (\theta_L, 0)$, denoted $\varphi_1$
2. A trajectory from $z_{\text{start}} = (\theta_L, 0)$ to $z_{\text{target}} = (\theta_H, 0)$, denoted $\varphi_2$
3. A trajectory from $z_{\text{start}} = (\theta_H, 0)$ to $z_{\text{target}} = (\theta_L, 0)$, denoted $\varphi_3$
4. A trajectory from $z_{\text{start}} = (\theta_\pi, 0)$ to $z_{\text{target}} = (\theta_L, 0)$, denoted $\varphi_4$

Each of these nominal trajectories was obtained using a bi-directional RRT algorithm as described in Section 4.6. We specify below the parameters used in the running of the program and then show the obtained trajectories.

The initial (relative) boundaries are set to be: $\text{boundaries} = \begin{bmatrix} z_{\text{min}} \\ z_{\text{max}} \end{bmatrix} = \begin{bmatrix} -3\pi/2 \\ -12.0 \\ \pi \\ 12.0 \end{bmatrix}$

The torque distribution is an evenly spaced set over the interval $[u_{\text{centre}} - \Delta u, u_{\text{centre}} + \Delta u]$ with a predefined number of torques, defined by the variable $\text{NumPossibleTorques}$. The centre torque $u_{\text{centre}}$ is set as the torque of the parent node (or set in advance for the root nodes).

For example: in the case of $\text{NumPossibleTorques} = 5$ we get

$$u = \begin{bmatrix} u_{\text{centre}} - \Delta u \\ u_{\text{centre}} - 0.5\Delta u \\ u_{\text{centre}} \\ u_{\text{centre}} + 0.5\Delta u \\ u_{\text{centre}} + \Delta u \end{bmatrix}$$

Alternatively, we can also set $u_{\text{centre}} = 0$ if we want to limit the torque to $\pm \Delta u$.  

---

1 Alternatively, we can also set $u_{\text{centre}} = 0$ if we want to limit the torque to $\pm \Delta u$.  

179
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

where $u_{centre} = N_{parent.torque}$ when the node is not a root.

At the end of each stage $s$ of the refinement procedure, $\Delta t$, $z_{min}$ and $z_{max}$ (the boundaries) are refined by a certain divisor $\text{RefinementDivisor}$ such that

$$\Delta t(s + 1) = \frac{\Delta t(s)}{\text{RefinementDivisor}}$$
$$z_{min}(s + 1) = \frac{z_{max}(s)}{\text{RefinementDivisor}}$$
$$z_{max}(s + 1) = \frac{z_{max}(s)}{\text{RefinementDivisor}}$$

The value of $\Delta u$ remains unaltered. In this example, it was found that a single stage only was sufficient to obtain good enough trajectories. However, we mention the above detail as it was used in more complicated examples that are excluded from this thesis.

Some parameters which were used for obtaining all of the four trajectories are specified in Table 5.3:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Stages</td>
<td>1</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>2000</td>
</tr>
<tr>
<td>Number of Possible Torques</td>
<td>5</td>
</tr>
<tr>
<td>$\Delta u_{init}$</td>
<td>1.0</td>
</tr>
<tr>
<td>$\Delta t_{init}$</td>
<td>0.01</td>
</tr>
<tr>
<td>Refinement Divisor</td>
<td>2.0</td>
</tr>
<tr>
<td>Probability of Sampling Best Couple</td>
<td>%10</td>
</tr>
<tr>
<td>Centre Torque</td>
<td>$N_{parent.torque}$</td>
</tr>
</tbody>
</table>

Table 5.2: Settings for the Simple Pendulum

The results of the four ran searches are summarized in Table 5.4.

Figure 5.7 shows the four resulting bi-directional trajectories with respect to time. Figure 5.8 shows the same trajectories in state space. Note that the forward and backward trajectories are disconnected, but the gap is too small to show in the figure. Figure 5.9 shows the torques (input signals) used to generate the four different bi-directional trajectories. Figure 5.10a shows the final bi-directional RRT tree that was developed to find the trajectory $\varphi_1$, and Fig. 5.10b shows the magnification of the tree around the
5.3. **EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION**

### Table 5.3: Settings for four trajectories

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\varphi_1$</th>
<th>$\varphi_2$</th>
<th>$\varphi_3$</th>
<th>$\varphi_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_{\text{start}}$</td>
<td>(0°, 0)</td>
<td>(30°, 0)</td>
<td>(150°, 0)</td>
<td>(180°, 0)</td>
</tr>
<tr>
<td>$z_{\text{target}}$</td>
<td>(30°, 0)</td>
<td>(150°, 0)</td>
<td>(30°, 0)</td>
<td>(30°, 0)</td>
</tr>
<tr>
<td>$u_{\text{start}}$</td>
<td>0</td>
<td>0.5mgl</td>
<td>0.5mgl</td>
<td>0</td>
</tr>
<tr>
<td>$u_{\text{target}}$</td>
<td>0.5mgl</td>
<td>0.5mgl</td>
<td>0.5mgl</td>
<td>0.5mgl</td>
</tr>
</tbody>
</table>

Table 5.4: Results from the Simple Pendulum Pathfinder

resulting nominal trajectory $\varphi_1$. The other three search trees look similar and are not shown here.

### 5.3.3 Hybrid System Design

In this problem, we wish to construct four funnels, each about a bi-directional trajectory found in the previous section. The “tasks” of the funnels are:

<table>
<thead>
<tr>
<th>Funnel</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F}_1$</td>
<td>bring states from the vicinity of $(\theta_0, 0)$ to the vicinity of $(\theta_L, 0)$</td>
</tr>
<tr>
<td>$\mathcal{F}_2$</td>
<td>bring states from the vicinity of $(\theta_L, 0)$ to the vicinity of $(\theta_H, 0)$</td>
</tr>
<tr>
<td>$\mathcal{F}_3$</td>
<td>bring states from the vicinity of $(\theta_H, 0)$ to the vicinity of $(\theta_L, 0)$</td>
</tr>
<tr>
<td>$\mathcal{F}_4$</td>
<td>bring states from the vicinity of $(\theta_\pi, 0)$ to the vicinity of $(\theta_L, 0)$</td>
</tr>
</tbody>
</table>

The desired transitions between the funnels can be described by the set of edges:

$$\Upsilon := \{(1,2), (2,3), (3,2), (4,2)\}$$

These tasks can be achieved using the directed graph shown in Fig. 5.11 with respect to which the system-of-funnels will be constructed. One can notice that there are no
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

![Nominal Bi-Directional Trajectories](image)

Figure 5.7: All four trajectories vs. time

sinks in this graph and there is only one cycle, comprised of funnels 2 and 3 and the edges connecting them. This shows that when designing an appropriate system of funnels according to the definitions of Chapter 2, trajectories of the hybrid control system starting from any initial condition will eventually cycle through these two funnels.

In this example, for the sake of simple programming, we chose to define the domain of the entrances to all of the funnels as \{0\}. Therefore, $E(0)$ is the only non-empty set of the mapping $E$ and it is in the interior of the set $F(0)$. Our system of four funnels has the set of funnel indices $K := \{1, 2, 3, 4\}$. Note that, as a result of this choice, the jump map which assigns new values to $\tau$, $z$ and $k$ after $(\tau, z)$ has reached the outlet $O_k$ is given by:

$$G_k(\tau, z) := \text{cl}(\{(0, z, j) \mid (k, j) \in \mathcal{Y}\})$$  \hspace{1cm} (5.21)
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

Since funnels 2 and 3 are interconnected, and have to satisfy the co-dependent conditions:

- The outlet of funnel 3 is in the entrance of funnel 2, and
- The outlet of funnel 2 is in the entrance of funnel 3,

we begin by designing these two funnels. Afterwards, we design funnels 1 and 4, which is a simpler procedure.

5.3.3.1 Funnels 2 and 3

Since the end states of each nominal trajectory are equilibria, we can make an initial choice of the outlets of the funnels 2 and 3 ($O_2$ and $O_3$, respectively) by obtaining estimations of the regions of attraction about the equilibria $z_{e,L} := (30^\circ, 0)$ and $z_{e,H} := (150^\circ, 0)$.
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

(both maintained by the nominal torque $u_{e,L} := 0.5mgI$) when using a local time-invariant LQR control that stabilises each of these equilibria (see Section 3.2.1 for details about the time-invariant LQR stabilisation). The region of attraction can be estimated by using a SOS program as explained in Section 3.2.4. Then, we can compute a bi-directional funnel backwards in time as was presented in Section 3.1.3 such that each of its components (i.e., the sub-funnels) are computed using the SOS program presented in Section 3.2.5. Note that due to the choice of funnels’ computation technique, the input $u_k$ associated with the funnel $\mathcal{F}_k$, $k \in \mathbb{K}$, is given by the time-varying LQR control designed to allow tracking the nominal trajectory about which the funnel is constructed. Also note that when using this control law when the state of the system is inside the entrance of the computed funnel (via the SOS program) guarantees that trajectories do not reach the boundary the funnel.

Figure 5.9: Torques used to generate the bi-directional trajectories.
Figure 5.10: Bi-directional tree for finding the trajectory $\varphi_1$ (a) and the magnified view of the trajectory

The flow map of the state variables $\tau$, $z$ and $k$, associated with the ‘in-funnels’ flow
Figure 5.11: A graph of 4 funnels for wing swing motion of a pendulum.

dynamics is given by:

\[ f_0 := (1, u_k, 0) \]  

(5.22)

where \( u_k \) is the time-varying LQR (TV_LQR) control associated with the tracking of \( \varphi_k, k \in \mathcal{K} \).

This initial design does not guarantee that funnels 2 and 3 are properly connected according to the co-dependent conditions stated above. To see this, we show in Fig. 5.12a to Fig. 5.13b such an initial attempt to compute the funnels \( \mathcal{F}_2 \) and \( \mathcal{F}_3 \) about the nominal trajectories \( \varphi_2 \) and \( \varphi_3 \). Figures 5.12a and 5.12b show funnels 2 and 3, respectively, separately on the state-space of the system (i.e., the boundaries of the sets \( \mathcal{F}(\tau) \) for a finite amount of \( \tau \) values), where Fig. 5.13a shows both of these funnels on the same plot. The nominal trajectories \( \varphi_2 \) and \( \varphi_3 \) are also drawn. Figure 5.13b shows only the boundaries of \( \mathcal{F}(0) \) and \( \mathcal{F}(T) \) of the two funnels, from which it can be clearly seen that the desired co-dependent conditions of proper interconnections are not satisfied.

In order to solve this problem, we extend the durations of the nominal trajectories at their target equilibrium state such that effectively a local stabilization about each of the equilibria is performed for a finite time duration. For instance, the new nominal trajectory \( \varphi_2 \) will be a concatenation of the old trajectory \( \varphi_2 \) and an additional constant trajectory with the angle value of \( \theta_H \) and zero velocity for some finite amount of time, to allow trajectories to get closer to the equilibrium \((\theta_H, 0)\), thus shrink the outlet \( \mathcal{O}_2 \)
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

Figure 5.12: Initial attempt to construct Funnel 2 and Funnel 3

without affecting its entrance. We aim to extend the trajectories just enough to have a sufficient margin between the entrances and the outlets of the incoming funnels (in more practical applications, the size of the margin should be chosen by taking into account the expected disturbances and noise in the system).

With the extended trajectories, the resulting funnels have a smaller outlet as predicted and as can be seen from Figures 5.14a to 5.15. Figures 5.14a and 5.14b show the new funnels 2 and 3, respectively, and Fig. 5.15 shows the boundaries of $F(0)$ and $F(T)$ of
5.3. **EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION**

Figure 5.13: (a) Initial attempts of Funnels 2 and 3 plotted together (b) Funnels 2 and 3 - only start and end sections

the two funnels, as well as the boundaries of the designed entrances (drawn in broken lines), from which it can be clearly seen that the desired co-dependent conditions of proper interconnections are now satisfied.

5.3.3.2 **Funnels 1 and 4**

In order to design funnels 1 and 4, we notice that both of them have to connect into funnel 2. Therefore, we chose for funnels 1 and 4 an outlet set which is in the interior of
the entrance to funnel 2, and then construct a bi-directional funnel backwards in time as usual. Figures 5.16a and 5.16b show the resulting funnels 1 and 4, respectively. Figure 5.17 shows the boundaries of $F(0)$ and $F(T)$ for all the four funnels and demonstrates that all of the conditions that have to be satisfied for $\Sigma = \{ F_1, F_2, F_3, F_4 \}$ to be a system-of-funnels are indeed satisfied.
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

For simplicity, as was mentioned earlier, we choose to use the trivial bootstrap control law $u_0 = 0$. When using $u_0$ as the bootstrap control for this system, we must remember that states can be brought either to the downright equilibrium $(0^\circ, 0)$ due to the friction in the system from any initial state other than $(180^\circ, 0)$, or, can theoretically remain in the upright equilibrium ($(180^\circ, 0)$). Therefore, we must account for both cases when constructing the system of funnels and the set $K'$. In order to simplify coding, we choose to allow a switch from bootstrap control to ‘in-funnel’ control only when the state of the system $(Z)$ reaches the sets $E_1(0)$ or $E_4(0)$, in which case a jump of $k$ will be to 1 or 4, respectively. We are not interested in switching from bootstrap control to the control associated with either funnels 2 or 3 directly. Therefore, we have that $K' := \{1, 4\}$.

The jump map from bootstrap control to ‘in-funnel’ control in our case is therefore given

Figure 5.15: Final construction of Funnels 2 and 3 - only start and end sections (and entrances)
Figure 5.16: Final construction of Funnel 1 (a) and Funnel 4 (b).

as

\[
\mathbf{G}_0(\tau, z) := \text{cl}(\{(0, z, j) \mid (0, \zeta) \in \mathcal{E}_j, \ j \in \mathcal{K}'\})
\]

This is a bit simplified formula than the original precise definition, however it captures the desired behaviour of the jump. Note that it is already assumed that the domain of the entrance is \(\{0\}\), and hence \(\tau\) is always re-set to zero when jumping into a funnel.
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

Figure 5.17: Final construction of all four Funnels - only start and end sections (and entrances)

The flow map associated with the continuous bootstrap dynamics is therefore simply given by:

\[ f_0 := (1, u_0, 0), \quad (5.24) \]

where \( u_0 = 0 \).

5.3.3.4 The Hybrid Controller

Now that the system of funnels is properly designed and the bootstrap control assumption is also satisfied, we re-write the hybrid controller equations for the specific system of funnels of this problem. Note that these definitions exactly match the original ones.

The flow set associated with bootstrap control is:

\[ C_0 := \text{cl}(\{(\tau, z, k) \mid G_0(\tau, z) = \emptyset\}) \times \{0\} \quad (5.25) \]

The flow set associated with ‘in-funnel’ control is:

\[ C_1 := \{(\tau, z, k) \mid (\tau, z) \in \mathcal{F}_k, \ k \in \mathbb{K}\} \times \{1\} \quad (5.26) \]
5.3. EXAMPLE 3: WING SWING MOTION - TASK DESCRIPTION

The jump set associated with jumps from bootstrap control to ‘in-funnel’ control is:

\[ D_0 := \{(\tau, z, k) \mid G_0(\tau, z) \neq \emptyset\} \times \{0\} \]  \hspace{1cm} (5.27)

The jump set associated with jumps to bootstrap control is:

\[ D_1 := \text{cl}(\{(\tau, z, k) \mid (\tau, z) \notin F_k, k \in \mathbb{K}\}) \times \{1\} \]  \hspace{1cm} (5.28)

The jump set associated with jumps between the different funnels is:

\[ D_2 := \{(\tau, z, k) \mid (\tau, z) \in O_k, k \in \mathbb{K}\} \times \{1\} \]  \hspace{1cm} (5.29)

The overall hybrid controller is:

\[
\mathcal{H}_{cl} : \begin{cases} 
\text{Flow dynamics:} & \begin{pmatrix} \dot{x} \\ \ell \end{pmatrix} = \begin{pmatrix} f_{k\ell}(\xi) \\ 0 \end{pmatrix}, \quad \begin{pmatrix} x \\ \ell \end{pmatrix} \in C_0 \cup C_1 \\
\text{Jump into bootstrap mode:} & \begin{pmatrix} x^+ \\ \ell^+ \end{pmatrix} = \begin{pmatrix} x \\ 0 \end{pmatrix}, \quad \begin{pmatrix} x \\ \ell \end{pmatrix} \in D_1 \\
\text{Jump into an entrance:} & \begin{pmatrix} x^+ \\ \ell^+ \end{pmatrix} = \begin{pmatrix} G_{k\ell}(\xi) \\ 1 \end{pmatrix}, \quad \begin{pmatrix} x \\ \ell \end{pmatrix} \in D_0 \cup D_2
\end{cases}
\]  \hspace{1cm} (5.30)

Note that in this construction of the system of funnels and the hybrid controller, the solutions to the system are unique.

5.3.4 Simulated Motion

We show results of three simulations of the motion obtained with our hybrid controller. Figures 5.18, 5.19 and 5.20 show the trajectories of the closed-loop system when started from the initial conditions \((\tau_0, z_0, k_0, \ell_0) = (0, 30^\circ, 0, 2, 1), (\tau_0, z_0, k_0, \ell_0) = (0, 180^\circ, 0, 4, 0)\) and \((\tau_0, z_0, k_0, \ell_0) = (0, -90^\circ, 5, 1, 1)\), respectively, in the state space of the system. The entrance to funnel 1 is also shown in Fig. 5.20.
Figures 5.21 and 5.22, show the behaviour of $\theta$ and $\dot{\theta}$, respectively, when initialised with $(\tau_0, z_0, k_0, \ell_0) = (0, -90^\circ, 5, 1, 1)$. The short “dwelling” of the trajectories near the equilibria states $(30^\circ, 0)$ and $(180^\circ, 0)$ can be seen in these two figures.

Note that even when the system is initialised exactly at an equilibrium, the nominal trajectories are not tracked with zero error, since all of the nominal trajectories have discontinuities. From Fig. 5.18 we can see that the desired wing-swing behaviour is achieved as expected.

When the system is initialised at $(\tau_0, z_0, k_0, \ell_0) = (0, 180^\circ, 0, 4, 0)$, since initially $\ell = 0$ but the mapping $G_0$ is not empty, an immediate jump to funnel 4 is possible. As a result, the trajectory flows inside funnel 4 (close to the nominal trajectory $\phi_4$) for the time duration $T_4$ and then jumps to funnel 2, and so on. When the system is initialised at $(\tau_0, z_0, k_0, \ell_0) = (0, -90^\circ, 5, 1, 1)$, the hybrid controller detects that $(\tau, z) \notin \mathcal{F}_k$ and that $G_0(\tau, z)$ is empty. As a result a flow with bootstrap control occurs, until the value of $z$ is inside the set $E_1(0)$, when the bootstrap control is switched off and the trajectory flows in funnel 1 for $T_1$ time, then cycles through funnels 2 and 3 as desired.

![Resulting Trajectory Starting from (30, 0)](image_url)

Figure 5.18: A trajectory from the initial condition $(30^\circ, 0)$ - state-space plot
5.4. Conclusions

In this chapter, three examples of systems of funnels are presented. The examples illustrate the computational implementation of all the important features of the systems of funnel framework, including the funnel construction, bootstrap controller, outlets of funnels that are contained in the entrance(s) of the next. The specific constructions created the desired behaviour. More complex systems and behaviours can be designed and accommodated using our framework, such as the construction of limit cycles for underactuated gait mechanisms and the switching between different gaits in a bipedal mechanisms.

Figure 5.19: A trajectory from the initial condition (180°, 0) - state-space plot
5.4. CONCLUSIONS

Figure 5.20: A trajectory from the initial condition (−90°, 5) - state-space plot

Figure 5.21: A trajectory from the initial condition (−90°, 5) - θ vs. time
Figure 5.22: A trajectory from the initial condition $(-90^\circ, 5) - \dot{\theta}$ vs. time
In this work, we presented approaches for designing and globally stabilizing behaviours of non-linear systems. This was done in a decomposed manner by presenting a technique to generate nominal motion plans and a general hybrid control framework that can be used to ensure global stabilisation of the desired motion. The contributions of the two ideas are summarised below, followed by suggestions for future work to improve the presented techniques.

6.1 Summary of Contributions

6.1.1 System-of-Funnels Framework for Global Stabilisation of Nominal Motions

In this thesis, we presented the system-of-funnels framework for achieving various motions in non-linear systems in a robust manner. The framework combines ideas from hybrid systems theory, which allows guarantees of robustness, a thorough stability analysis and a prediction of the limiting behaviour for the solutions to the hybrid system, with ideas of computable regions of finite-time-invariance which can be seen as funnels in the state-time domain.

The framework contains a generalization of the definition of a funnel, including additional definitions of an entrance and an outlet. Two types of funnels are considered: funnels having finite- and infinite-time domains. A funnel is then treated as a general object which can be connected with other funnels. We presented ideas for 6 classes of funnels which can be used in common applications and the mathematical specification
6.1. SUMMARY OF CONTRIBUTIONS

which defines them. We defined the conditions for a set of funnels to be considered a system of funnels via proper interconnections between the funnels, which are defined with respect to a set of edges, or, equivalently, a directed graph.

We presented a hybrid control system which can switch between local controllers related to the different funnels in the system while ensuring that when the system is initialized inside any of the entrances, its solutions remain inside the total system of funnels. In addition, we incorporated the idea of a bootstrap control into the hybrid controller and showed that using it (that is, if the bootstrap assumption is valid), the system can be initialized anywhere in the state space and be steered into some entrance within a finite amount of time and up to two jumps, after which the trajectories are guaranteed to remain within the system-of-funnels.

Moreover, we analysed the limiting behaviour of the solutions of the hybrid control system under some assumptions on the system-of-funnels. For a system of funnels which complies with these assumptions, there exists a UGAS \( \omega \)-limit set which is contained in a compact set, denoted \( \mathcal{X} \). We also showed that \( \mathcal{X} \) can be defined as the union of the reachability sets from entrances of a subset of funnels. The funnels in this subset can be found from the graph with respect to which the system of funnels is defined. Essentially, this graph can be used to both design the system of funnels and analyse its behaviour.

The system-of-funnels framework is very general and can be used in various applications. Normally, in specific implementations, additional assumptions are made to define the specific system of funnels and the graph with respect to which it is defined. These assumptions can lead to stronger stability results for the system, which go hand in hand with the behaviour which is desired to be achieved.

Finally, we showed that two important existing frameworks - Throw-and-Catch and LQR-Trees - are in fact special cases of the general framework presented in this thesis.

6.1.2 A Bi-Directional RRT Algorithm for Constructing Nominal Trajectories

In order to demonstrate how the framework can be applied in practice, we chose to use it for three illustrative examples, one of which involves motion planning for mechanical
systems. For this sake, we first presented in detail a bi-directional RRT algorithm which was designed to enhance the chances of finding a nominal trajectory. This algorithm addresses specifically problems containing under-actuation, for which it is normally hard to plan motion, using some heuristics which proved useful in our experience. For the nominal bi-directional trajectories, we computed bi-directional funnels which are properly connected. For some of the the example we also use a simple bootstrap control. We then show that our system controls trajectories from various initial conditions according to its design.

The main modifications to the nominal bi-directional RG-RRT algorithm are: modifying the distance metric to have a term related to the local directionality of the vector field of the dynamic system; introducing a new approach for growing the bi-directional tree by properly and efficiently bookkeeping the pair of nearest nodes from the opposite trees to attempt a quick connection between the trees; presenting the nested search idea to allow a refined reaching-the-goal mechanism on the expense of losing probabilistic completeness. All of these modifications are empirically shown to assist the efficiency of the search.

6.2 Suggested Future Work

6.2.1 Hybrid Frameworks for Global Stabilisation

Even though our framework includes the work of Throw-and-Catch as a special case and hence in this case the robustness to disturbances and noise of some measure is guaranteed, we did not present a systematic proof for guaranteeing robustness to disturbances, uncertainties and noise in the general case. It is very intuitive to say that such guarantees should exist and hence, as a future work, we hope to establish general robustness results.

Furthermore, we wish to use the system-of-funnels framework in a wider range of applications, and hopefully, not limited to simulations, where we can demonstrate various desired system behaviours which comply with the intended system-of-funnels design.

Some generalizations can be readily introduced to our framework. We list here three:

- For the sake of stating stability results, we used $\tau = -1$ and we also assumed
6.2. SUGGESTED FUTURE WORK

constant mappings for $F(\tau)$ and $E(\tau)$ for infinite-time domain funnels. However, these assumptions are too restrictive and can be relaxed in various manners.

- We assumed that the overall dynamic system is continuous, and that jumps are due to our hybrid controller only, however, it is possible to generalize our framework to dynamics systems which are naturally hybrid, such as bipedal walking models.

- The assumption of an existing bootstrap control might be realistic many times, however, for cases where it is impossible to use a single bootstrap controller which satisfies the properties assumed in our framework, it is sometimes possible to have more than one controllers that together achieve the desired behaviour. Therefore, the bootstrap control assumption can be made more general as well.

6.2.2 Nominal Trajectory Planning using Randomised-Sampling Algorithms

We have shown how the Euclidean nearest distance metric could be augmented with a directionality term, based on angles between certain vectors. There are more possible ways to extend this idea. For instance, it is possible to calculate the angular distance between certain planes and vectors which can shed more light on the local directionality of the dynamics for state spaces of order more than two. On the other hand, these computations are more computation-time consuming. We suggest examining more elaborated extensions to the distance metric for non-holonomic and under-actuated systems in order to make better decisions on which distance metric should be used.

Additional immediate extension to our work is to construct funnels about the best bi-directional path during the ongoing search and update these funnels whenever the best pair of nodes is updated. In this manner, we can have a good indication of when we can stop the search, by checking whether or not the outlet of the forward funnel is in the entrance of the backward funnel during the running of the algorithm.

This idea can be taken further when parallel computing is employed. Consider, for example, the task of transitioning from one periodic orbit to another. One can use the bi-directional search in parallel from many initial states on the initial periodic orbit and from many final states on the final periodic orbit. The search can then be terminated when the outlet of any of the forward funnels is in the union of any of the entrances
to backward funnels, provided that forward and backward funnels are computed not only for the best pair of nodes but for the whole bi-directional tree. Note that parallel computing can also be employed to find several trajectories at the same time. However, we did not extend our algorithm to be compatible with parallel computing.
Bibliography


[34] Steven M LaValle and James J Kuffner Jr. “Rapidly-exploring random trees: Progress and prospects”. In: (2000).


Author/s: 
SHVARTSMAN, RINA

Title: 
System of funnels framework for robust non-linear control

Date: 
2015

Persistent Link: 
http://hdl.handle.net/11343/56396

File Description: 
System of Funnels Framework for Robust Non-linear Control