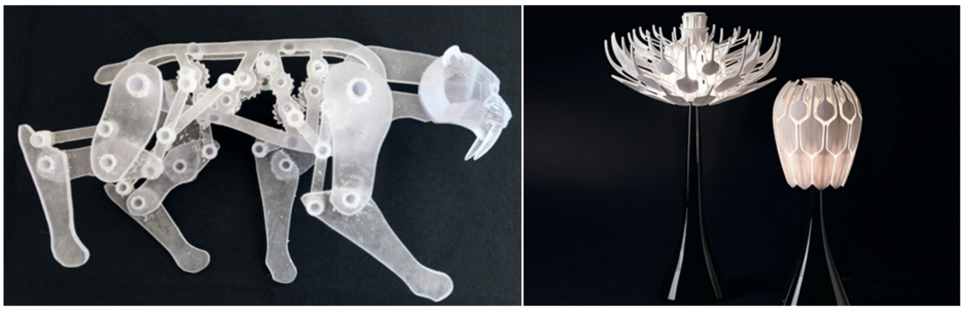
Thesis Title



Jonas Stolle

Bachelor Thesis
May 2021

Supervisors:
Moritz Geilinger
Prof. Dr. Stelian Coros

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

This thesis addresses the development of a novel sample thesis. We analyze the requirements of a general template, as it can be used with the LaTeX text processing system. (And so on…) The abstract should not exceed half a page in size!

Contents

List of Figures

List of Tables

1 Introduction

also state here that we are dealing with mechanical systems in particular

2

Related Work

Sample references are [?] and [?].

2.1. Robustness Overview

Robustness is a desirable property of robotic systems as it allows for errors and uncertainties without system failure. yet a universal definition or approach in quantifying it is nowhere to be found. To acquire an overview of the topic, a couple of definitions are presented in the following.

2.1.1. The Robustness Principle

The Transmission Control Protocol from 1981 [?], which contains guidelines for host-to-host communication via networks, states: “Be conservative in what you do, be liberal in what you accept from others”. The idea being that sent data out should be as concise as possible, while on the receiving side deviations from the standard should be accounted for.Translated to our problem, a robot behaves robustly when given concise commands it successfully executes them even in the presence of unexpected disturbances. A margin of error should always be accounted for to bridge the gap between theory and practice.

2.1.2. Robustness against Failure

N. Hazon et al. [?] have improved upon multi-agent algorithms for coverage to account for failure of agents. Here robustness was quantified by proving that the presented algorithms would always succeed, given that at least one agent is still functioning. On a similar note, M. Hofbaur et al. [?] have worked on making mobile robots robust against failure, however here the robustness was defined with respect to unexpected internal changes, for example the failure of an actuator. The issue with these failures is that the robot is usually not aware of the fault and continues to try to use the broken component, resulting in the failure of the entire robot. By actively monitoring the system and changing the internal model in, the robot is able to overcome its impairment in many cases. In both cases, robustness is merely a binary measure and its value can be quantified quite decisively. In contrast, the problem at hand seeks a measure that can be evaluated continuously, especially as the robustness is to be compared between designs and eventually maximized.

2.1.3. Robust Optimization

Robust optimization is an extension to ordinary optimization problems, in which uncertainty sets are introduced as additional parameters, as described and summarized by [?]. The argument goes that if a solution is feasible not for only one parameter but covers a set of uncertainties, it will in turn be generally robust against them. While optimally the solution would be robust against any arbitrary uncertainty, covering a larger amount of cases usually also results in the deterioration of the solution for any particular case. The optimization algorithm is applied to find a balance within this trade-off. The way the uncertainties are defined and considered results in different "strengths" of robustness, however there exists no method for measuring the resulting robustness of the solution, meaning that it cannot be assessed independently of the optimization problem. This makes it impossible to compare different systems as optimizing over the same set of uncertainties in different systems in no way guarantees that their behaviour in any specific case will be comparable.

2.1.4. Residual Force Polytopes

Residual force polytopes have been used by [?] for trajectory optimization on the ANYmal quadruped platform. The approach here is to represent all largest possible outside forces that the system can compensate as a polytope, i.e. a high (usually 3) dimensional volume. Here the vector from the actuator to any point on the surface represents the critical force in that direction. The largest sphere that can be placed within a polytope represents the worst case scenario for that actuator. The radius of this sphere can be used as a concrete measure for robustness which encompasses all possible disturbances. Given a rough trajectory, Ferrolho et al. [?] were able to perform an optimization with this measure, maximizing the sum of all of the inscribed spheres for a maximally robust position at every point along the trajectory. On a first look this seems promising, however there might be a multitude of physical constellations where the end effector resides at the same position, as shown in Figure 2.1. Each constellation has a different polytope and therefore a different robustness measure, a problem which becomes worse for systems with high degrees of freedom. This dependence on the specific state renders this measure unfeasible for our problem.

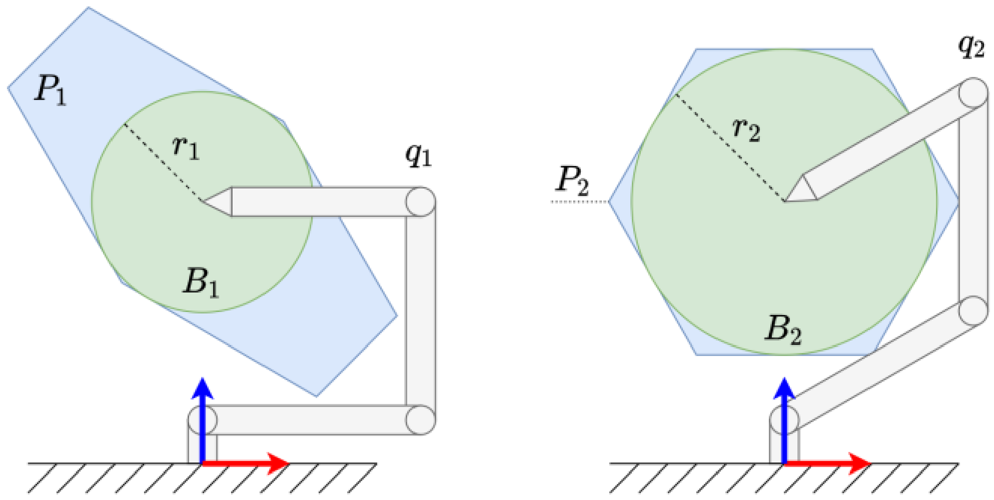


Figure 2.1.: *Comparison of two residual force polytopes with identical endeffector position. The blue area represents the maximal forces the system can compensate with the green circle being the worst case scenario. Note that the shapes change with respect to the configuration of the arm.[**?**]*

2.2. Global Robustness in Nonlinear Dynamic Systems

It should now become clear that the way robustness is defined and quantified strongly depends on the problem it is applied to. The hurdle in finding a measure for the particular problem at hand is that neither the underlying system nor the applied disturbances are predefined as to preserve generality. To find a suitable solution, it is a good idea to formalize the question to a degree. The mathematical representation of mechanical systems is usually given in the form of differential equations $\dot{\mathbf{x}} = F(\mathbf{x})$, describing the relationship between the states of a system and their change with respect to time. Nonlinearities caused by compliance as well as impacts typically occurring in legged locomotion exclude methods applicable to linear systems of equations. What remains are nonlinear systems of differential equations which are to be analyzed and ultimately quantified with respect to robustness.

2.2.1. Phase Space Representation of Dynamic Systems

A popular method of visualizing dynamic systems are phase space diagrams, which encompass all possible states of a system. For mechanical systems these states are usually the generalized coordinates and their temporal derivatives, where the dimension of the space scales with the degrees of freedom. Solving the differential equations repeatedly, taking various states as the initial conditions, reveals the global behaviour in the form of trajectories within the phase space. An two dimensional example of this can be seen in Figure 2.2.

While the trajectories go on indefinitely, most quickly go to infinity or end up in a cyclic pattern. The latter are called attractors, which are of large interest in the discussion of nonlinear dynamic systems. They represent a set of states which the trajectories will not leave once they encounter them. This might represent the repeating motion of a robotic foot at the end of a leg. An important insight here is that around any attractor there exists a set of states who's trajectories will always converge towards the attractor. This set is called the Basin of Attraction (BoA), however it should be noted that the terms “Domain of Attraction”, “Region of Attraction” as well as “Basin Attraction” are used synonymously in literature.

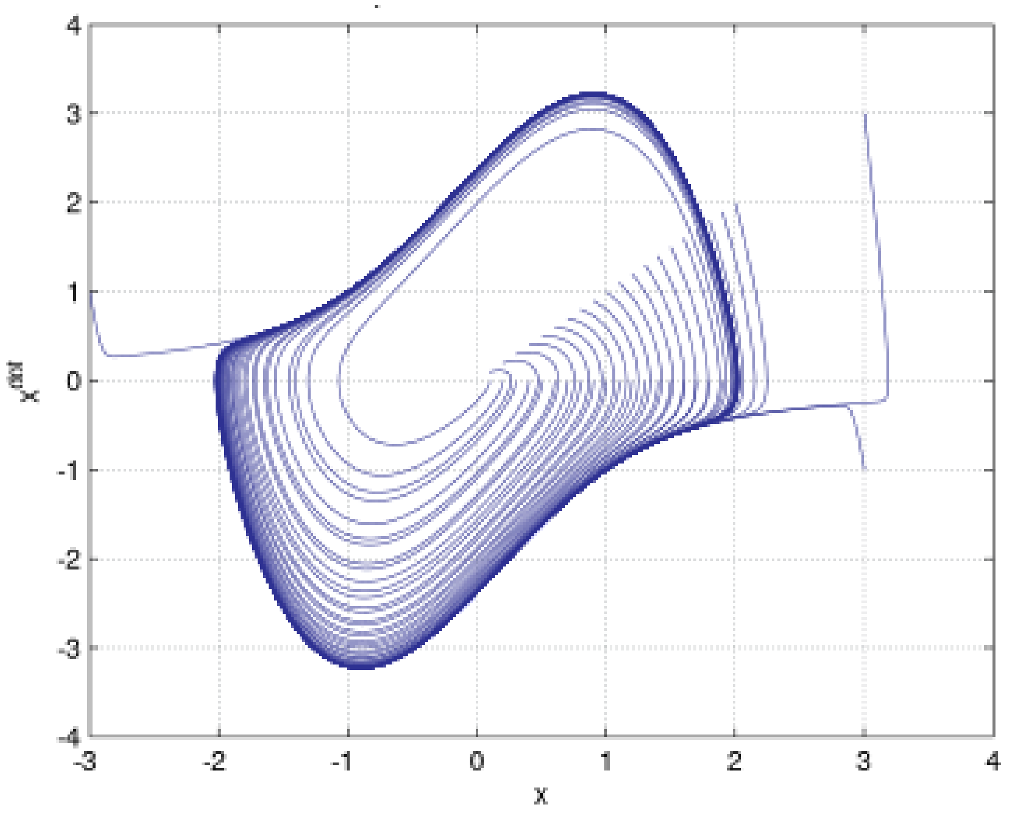


Figure 2.2.: *Phase Space Diagram of a Van der Pol Oscillator. The attractor is indicated by the thick line and the trajectories by the thin lines, all converging towards the attractor. [**?**]*

2.2.2. Robustness Measures via the Basin of Attraction

Given that all trajectories starting in the BoA will also always stay within it, a robustness formulation can be derived by taking the effect of disturbances into account. Disturbances generally result in a state changes, which in the context of the phase space corresponds to a jump onto a different trajectory. As long as the new state lies within the BoA, the trajectory will return back to the stable solution and the robot will recover. However if the disturbance is too large, the system will land on a diverging trajectory outside the BoA, quickly leading to failure of the system. To increase robustness, we want to decrease the likelihood of a disturbance moving the system state outside of the basin. Having no control over the types of disturbances that our robot leg might encounter, the geometry of the basin itself needs to be changed. A larger basin will make it more likely that the new trajectory will stay within it, given a random disturbance. The geometry of the BoA is therefore a representation of the system's robustness. To obtain a quantified measure of the robustness, one simple approach is to take the high dimensional volume of the basin. Rummel et al. [?] have applied this robustness formulation to identify stable walking gaits in a two degree of freedom spring-mass walker model. In this low dimensional case the volume translates to the area of the BoA, which was always bounded for the system analyzed. Of course these basins may take many complicated forms, including fractals and regions that stretch to infinity. Sprott et al. [?] have come up with a method to sort the BoA into classes, listed in Table 2.1 and shown in part in Figure 2.3. They achieved this by finding a power law describing the probability of a point laying within the basin, depending on it's distance from the attractor.

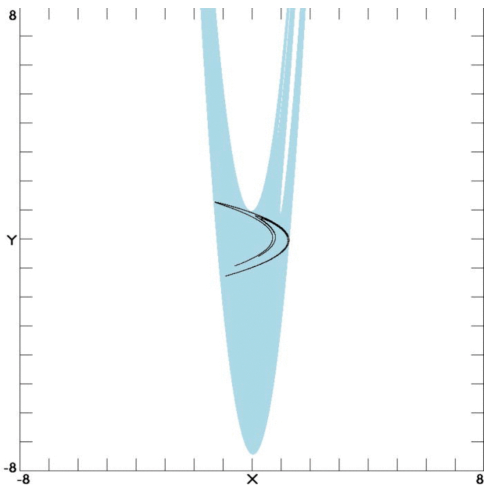
Table 2.1.: *Classes of Basins. [?]*

1	globally attracting basins
2	attracting a fixed fraction of phase space
3	basins extending to ∞ in some directions
4	bounded basins

Knowing that BoA are not uniform in most cases, the direction of a disturbance within the phase space matters. If the basin border is particularly close to the attractor at any point, the system will be less robust to disturbances in that direction. Comparable to the methodology in Section 2.1.4 Horibe et al. [?] have considering the worst case scenario by finding the shortest distance from the attractor to the boundary of the BoA. This minimal radius of the BoA is a more conservative measure of robustness. It also allows to disregard complicated parts of the basin that extend to infinity or have strong fractal properties. In [?] this robustness measure was also determined, however under the name "Integrity Measure". Coinciding with the goals of this project, [?] have used this robustness formulation to successfully optimize the physical design of inverted multi link pendulums with respect to control effort.

2.2.3. Numerical Methods for Basin of Attraction Analysis

In order to quantify the robustness of a system using the BoA, the shape of the basin needs to be explored. Solving the systems of differential equations analytically is often not feasible, requiring numerical methods to be applied. The general procedure is first the sampling of a finite set of initial conditions from the continuous phase space and then integration of the differential equations. One popular method, applied by Brzeski et al. [?] to quantify the responses of multistable systems, uses Monte Carlo methods for the sampling in which the initial conditions are chosen randomly from a given probability distribution. Finding the corresponding trajectories and determining whether they converge to the attractor gives a picture of the basin

[Class 3]


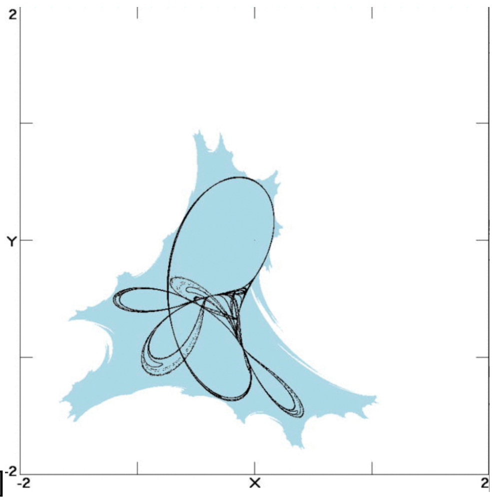
[Class 4]


Figure 2.3.: *Examples of basin classes. The black lines represent the attractors and the blue areas the basins of attraction. Note that in (a) parts of the basin extend to infinity while (b) is bounded. [**?**]*

geometry. [?] explores potential issues with this method and finds that chaotic systems can lead to large cumulative errors in the integration of the trajectories. Furthermore the fraction of space that the basins take up tends to become smaller in higher dimensions, leading to more samples having to be evaluated and in turn increasing the computational cost. An alternative approach are the "Cell Mapping Methods", first summarized in [?] and recently updated in [?] as a comprehensive book. For cell mapping methods, the entire phase space is discretized along each dimension, resulting in a grid of possibly high dimensional, equally sized cells. The center point of each cell is chosen as the initial condition with which the differential equations are solved. The resulting trajectory is computed for a short time period to determine the mapping to a following cell. Repeating this procedure for all cells reveals the global behavior of the system dynamics. The benefit here is that integration errors are only local and stay confined to the basin boundaries. In addition, as the integration of the differential equations is bound to short intervals, each evaluation becomes computationally cheaper.

Figure 2.4.: *One dimensional SCM.[?]*

Figure 2.5.: *One dimensional SCM. The rightmost cell goes to infinity, which is mapped to the 0 position, the so called sink-cell. With the remaining cells an attractor can be seen at x = 0.5. [?]*

This deterministic mapping between cells represents the Simple Cell Mapping (SCM). An issue with it is that it does not consider the infinite amount of initial conditions covered by each cell, which could all potentially result in vastly different trajectories. This discrepancy is overcome with the General Cell Mapping (GCM), a stochastic approach where from each starting cell the probability of landing in any other cell is found. This problem translates to a Markov chain representation, which has been well researched and for which efficient algorithms exist. The GCM is also better at dealing with fractal basin boundaries and can handle chaotic systems. As seen [?], there are many extensions to, as well as hybrids of the SCM and GCM methods, yielding better results. But of course cell mapping methods do have their limitations. As always, the curse of dimensionality persists and adding more dimensions exponentially increases the number of cells that have to be evaluated. For the overarching project leg designs with many degrees of freedom need to be analyzed, resulting in high dimensional phase spaces and increasing the computational cost. To overcome this hurdle, there have been efforts in parallelization of the computation by [?],[?] and [?], allowing for the exploration of higher dimensional systems, with promising results (see Figure 2.6). Given the results from any of the Cell Mapping Measures, either the volume or the minimal radius of a BoA can then be found to act as the concrete measure of robustness.

Looking again at [?], the robustness was determined without a global exploration, but by strategically sampling initial conditions and using the bisection algorithm to approximate the minimal radius, as seen in Figure 2.7. For this method the trajectories had to again be evaluated until their convergence could be determined. Still, with the strong reduction of the total number of evaluations to be performed, the computational cost of quantifying the robustness was greatly decreased.

2.3. Discussion

The previous sections presented robustness definitions in different fields as well as concrete measures applicable to nonlinear systems. Cell Mapping Methods and random sampling in combination with trajectory evaluation were discussed and their shortcomings as well as their benefits were pointed out. While Cell Mapping Methods have low evaluation costs for each of their cells, the total computational cost rises exponentially when applied to higher dimensions. At the same time they are easily parallelizable, leaving a possibility of reducing the total computational time. The random sampling methods face similar issues in high dimensions with shrinking basin ratios and large errors in the trajectory exploration caused by chaos. Given the geometry of the BoA, the volume of the basin and its minimal radius were presented as specific measures for robustness. The later introduced bisection method presented an improvement on the random sampling approach by determining only the minimal radius of the BoA. An important assumption for this method is the knowledge of the location of the BoA, as the algorithm depends on knowing the distance from the center of attraction. The bisection method seems to be the most promising, especially as it has been applied to a problem very similar to the goal of the project. This however also assumes that the input data is a set of differential equations. Given experimental data, it might make sense to incorporate them into the Cell Mapping Methods, possibly mitigating the need for computing the trajectories. In any case, these are all very powerful tools, which can obtain excellent results, given a certain amount of computational resources.

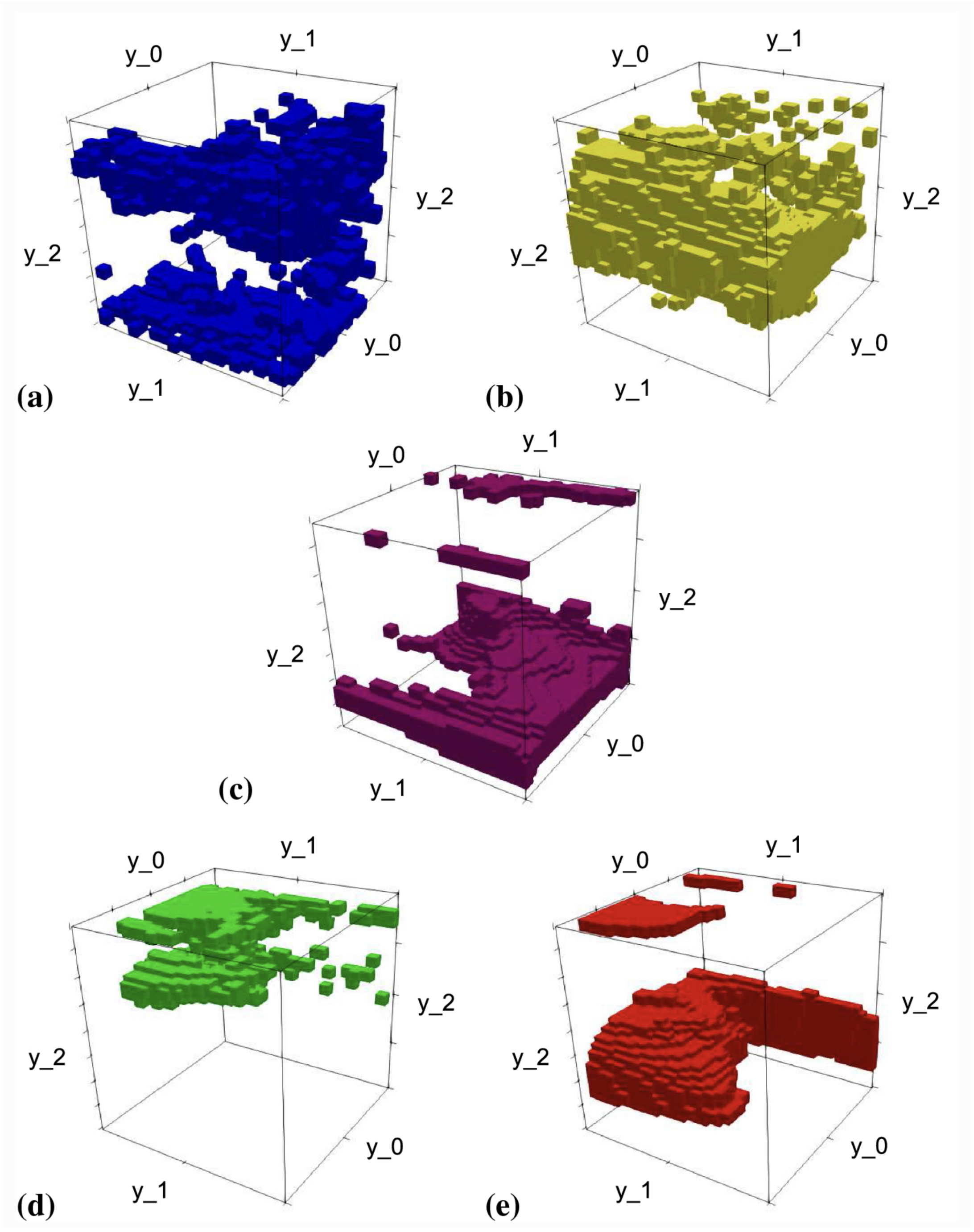


Figure 2.6.: *3D sections of multiple six dimensional Basins of Attraction. [?]*

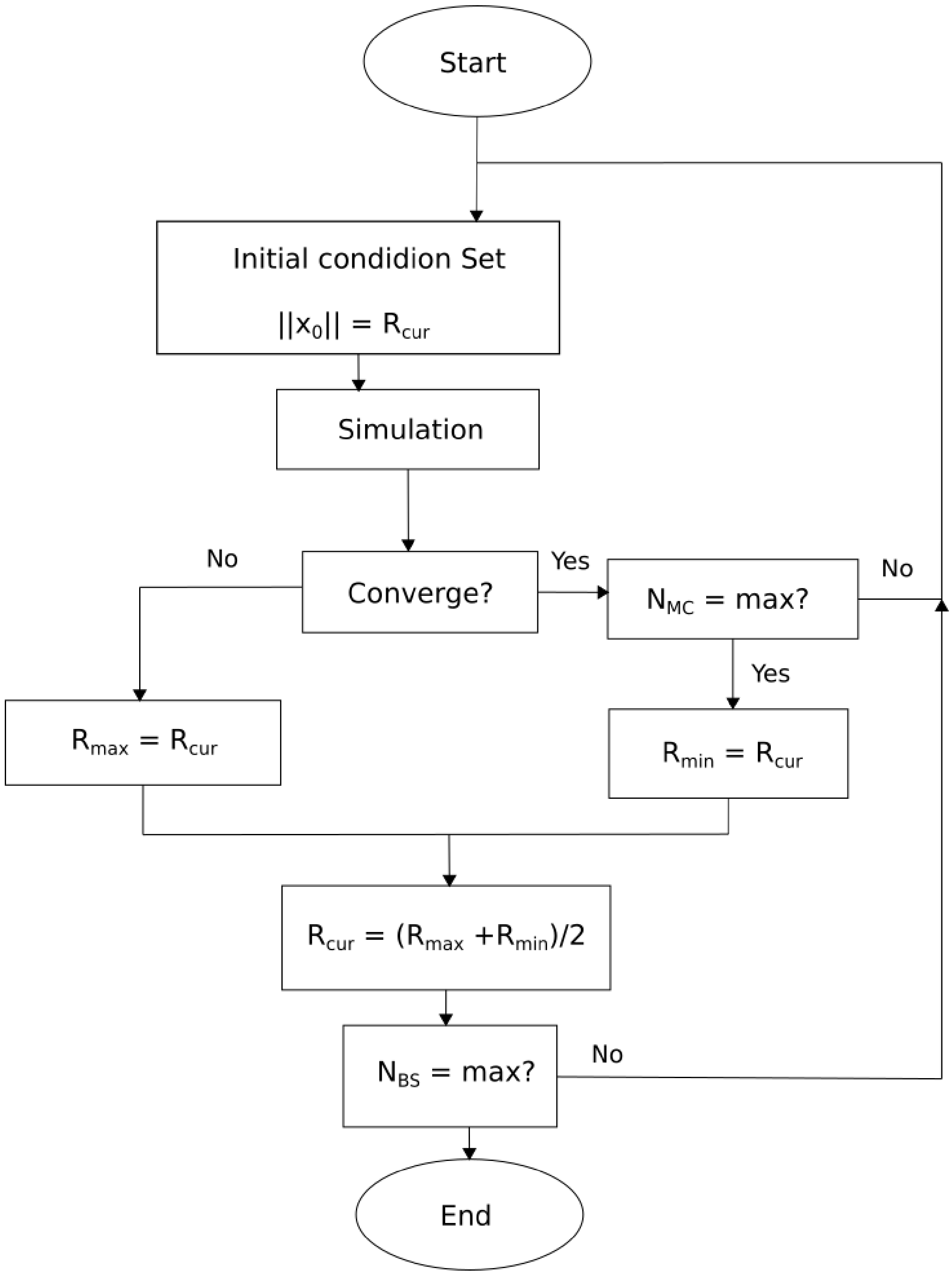


Figure 2.7.: *Bisection Algorithm. Using two initial guesses N_{MC} samples at the current radius are evaluated for convergence. If all samples converge the minimal radius is updated, otherwise the maximal radius. This is repeated for N_{BS} iterations. [**?**]*

3

Your Central Work

3.1. Fundamentals and Problem Formulation

In this sectionlays out the relvant physical definitions from which the robustness measure is derived and which will be referred to in the later code implementation. all of which are important for understanding and some of which are directly used for the implementation.

Examples will be given n terms of laikag quadruped robot as most of the testing of the framework was done with that.

3.1.1. Dynamical Systems

Dynamical systems are distinguished by an evolution of their state $\mathbf{x}(t)$ through time. This evolution can be fully described by a set of ordinary differential equations of the form $\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), t)$, where $\mathbf{F}$ is some nonlinear function. This simplifies to $\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t))$ under the assumption that the dynamical system is autonomous, i.e. is not explicitly dependent on time. When solving for the explicit solution $\mathbf{x}(t)$, an initial condition $\mathbf{x_0} = \mathbf{x}(t_0)$ is required, which is a system state at an initial time. For simplicity and without loss of generality for autonomous systems, we set $t_0 = 0$. With this the Initial Value Problem (IVP) can be formulated:

$$find\ \mathbf{x}(t) \tag{3.1}$$

$$s.t.\ \dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t)) \tag{3.2}$$

$$and\ \mathbf{x}(0) = \mathbf{x}_0 \tag{3.3}$$

We denote the solution to the IVP as $\mathbf{x}(t, \mathbf{x}_0)$. It represents trajectory of the system state through time given an initial condition. The space in which this trajectory lies is spanned by all possible states x and termed the $state\ space$. Note that any future state of the trajectoy $\mathrm{x}(\tau, \mathrm{x}_0)$ at time τ can be taken as an initial condition of the IVP itself. It turns out that the new solution coincides with the initial one, i.e. $\mathrm{x}(t, \mathrm{x}_0) = \mathrm{x}(t, \mathrm{x}(\tau, \mathrm{x}_0))$, which illustrates that any state $\mathrm{x}(t)$ of a trajectory $\mathrm{x}(t, \mathrm{x}_0)$ is suffitient to represent the trajectory as a whole.

Mechanical systems (on which we will focus on form here on out) tend to be described in terms of so called generalized coordinates:

$$\mathbf{q}(t) = \begin{pmatrix} q_1(t) & q_2(t) & \dots & q_n(t) \end{pmatrix}^{\mathsf{T}}. \tag{3.4}$$

They are the minimal set of coordinates needed to fully describe the position and orientation of all of the systems elements. Their dimension n cooincides with the number of degrees of freedom of the system. The corresponding differential equations are of second order, depending on $\ddot{\mathrm{q}}(t)$ in addition to $\dot{\mathrm{q}}(t)$ and $\mathbf{q}(t)$. Simply put, this is due to their derivation by Newton's second method, where forces acting on the system are related to the second time derivative via $F = ma$. Through an order reduction (Appendix) these differential equations can be cast into the reviously mentioned general form 3.1, where

$$\mathrm{x}(t) = \begin{pmatrix} \mathbf{q}(t) & \dot{\mathbf{q}}(t) \end{pmatrix}^{\mathsf{T}}.$$

This implies that in order to solve the IVP, initial conditions $\mathbf{q}_0$ and $\dot{\mathbf{q}}_0$ are required. We can also state that for a system with n degrees of freedom, $\mathrm{x}(t) \in \mathbb{R}^{2n}$. The particular state space spanned by generalized coordinates and velocities is termed the $phase\ space$. Within it, any states are single points and state trajectories are smooth curves. The phase space is used for generalizable qualitative analysis of the behaviour of nonlinear dynamical systems and plays a pivotal role in the fromulation of the robustness measure. It shall be stressed at this point, that any instantaneous configuration of the isolated system one can think of is a point in the phase space and it is impossible for the system state to leave or exist outside of it without fundamentally changing the system.

When discussing high level concepts we will refer to the system state $\mathrm{x}(t)$ for simplicity, while in the code implementation the generalized coordinates $\mathbf{q}(t)$ and velocities $\dot{\mathbf{q}}(t)$ will be more relevant. Keep in mind that both are equivalent.

3.1.2. Attractors and Convergence

In nonlinear dynamical systems, there may exist sets of states in the phase space which show an attracting behaviour. By "attracting" we mean that once a trajectory reaches an element of such a set, all of its future states will also be part of that set. Define an attractor as a set of states:

$$\mathbf{A} \subset phase\ space, \tag{3.5}$$
$$s.t.\ if\ \mathrm{x}(t_0) = \mathrm{x}_0\ \in \mathbf{A}, \tag{3.6}$$
$$\mathrm{x}(t, \mathrm{x}_0) \in \mathbf{A} \quad \forall\, t > t_0. \tag{3.7}$$

Attractors can be divided into two fundamental variants. If the attracting set consists of only one state, it is called a fixed point. Fixed points are associated with the condition

$$\dot{\mathrm{x}}(t) = \mathbf{0}\ \ \forall\, t \in \mathbb{R}, \tag{3.8}$$

meaning that the state of the fixed point is unchanging and the related trajectory is reduced to a point in the phase space. Whether a state $\mathbf{x}(t)$ is a fixed point can be easily determined by checking $\mathbf{F}(\mathbf{x}(t)) = \mathbf{0}$. A classical example of a fixed point is the stable bottom position of a pendulum, where if given zero initial velocity, it won't leave the stable position. This is not the case for any other position as gravity will act on the mass (except for the inverted position, however this is practically not realizable). In the general case with A containing of more than one state, (reference above eq) is not true. Rather the trajectory is moving through the sets of A, visiting every element at some point in time and returning to it at future times in a periodic fashion. These types of attractors are called limit cycles. Finding them is generally a hard problem, but for simpler cases one can check:

$$If\ for\, \mathbf{x}(t, \mathbf{x}_0),\ t > 0 \quad \exists \quad \mathbf{x}(\tau) = \mathbf{x}(\tau + h),\ \tau > 0,\ h > 0, \tag{3.9}$$
$$\{\ \mathbf{x}(t) \mid t \in [\tau, \tau + h)\ \},\ is\ a\ \textbf{limit cycle}. \tag{3.10}$$

An example for this case are the compliant linkages described in (ref strandbeest compliant version), where the end effector follows a cyclic trajectory, i.e. set of states if undisturbed. In the code implementation section, we provide methods for dealing with both types of attractors and the problem of applying the continuous analysis in a discrete setting.

For any initial condition not part of the attractor, the related trajectory may land and stay on the attractor after some time $t > t_0$. We define this occurance:

$$Given\ an\ attractor\ \mathbf{A},\ for\ any\, \mathbf{x}_0 \notin \mathbf{A} \tag{3.11}$$
$$if\ \underset{t\to\infty}{lim}\, \mathbf{x}(t, \mathbf{x}_0) \in \mathbf{A}\ \Rightarrow\ \mathbf{x}(t, \mathbf{x}_0)\ \textbf{converges}\ to\ \mathbf{A} \tag{3.12}$$

Should the definition above not hold, we denote the trajectoy as diverging (as in cell mapping methods).

Note that there may exist any number of attractors in the phase space of a system (reference paper with multitude of attractors). Convergence is always defined with respect to a particular attractor $\mathbf{A}$, which needs to be specified. Therefore if the trajectory converges to a different attractor, it is still defined as diverging from the attractor of interest.

The set of all states for which the related trajectories converge to the attractor of interest is called the $Basin\ of\ Attraction$ (BoA) and is defined as:

$$\{\ \mathbf{x}_0 \in \mathbb{R}^{2n} \mid \underset{t\to\infty}{lim}\, \mathbf{x}(t, \mathbf{x}_0) \in \mathbf{A}\ \} \tag{3.13}$$

where $\mathbf{A}$ is an attractor as defined in 3.8.

From here on we will represent the general attractor of interest with $\mathbf{A}$. Additionally, we define disturbances as any event that acts upon the system, inducing a state change. This new state implies a new trajectory as a solution of the IVP, meaning the future behaviour of the system may differ vastly from the undisturbed case.

3.1.3. Robustness Measure

The robustness measure proposed in REF and implemented in the following is derived using the concepts of nonlinear dynamics outlined in the previous section. In the actual implementation and testing compromises have to be made (rephrase) in order to improve feasibility, which is why we propose a slight reframing of some definitions to accomodate for those variations.

First we classify trajectories in successful recovery from disturbances or failure to do so. Given a dynamical system, assume that it shows some stable and desirable behaviour in its undisturbed state. Examples of this might be holding a specific pose or walking with a periodic gait. We can interpret these types of behaviour as attractors in the phase space of the system. Applying disturbances will disturbe the state trajectory in ways that are hard to predict as they don't follow the IVP. Once the distubances stop however, the disturbed state of the system at that point in time can be used to solve the IVP and an new traectory can be computed. may be move the system state outside of the attracting set.

Following the new trajectory, we can detemine whether the system will converge back to the attractor. In such a case successful recovery of the system from the disturbance is achieved; one might say the system is robust to that disturbance.

(add hand drawn representations of the effect of disturbances in the phase space)

Note (add into previous paragraph, that disturbances can do "anything" to the trajectory, i.e. mustn't be continuous, etc. But once the disturbance stops, we can take the state that the system is in at that time to compute the future dynamics, given that no more disturbances are applied. Else we repeat until no further disturbances.)

Note that a this holds for multiple consecutive or distributed disturbances as well, as only eventual convergence of the trajectory is relevant. Here the IVP has to be solved repeatedly after every new disturbance.

(Analyzing all the states in the phase space corresponds to analyzing the effects of any type of disturbance, for times after the end of the disturbance) Because of the direct relation between disturbances and state changes, one may choose to only analyze the convergence behaviour of states in phase space. This is how REF formulated their robustness measure. The size of the set of states in the phase space which converge back to the attractor is a measure of robustness as it is a representation of the amount of disturbances the system can recover from. Notice that this is precisely the basin of attraction defined in REF. As outlined in REF, finding the size of the BOA is nontrivial, for which REF introduced the conervative measure of the minimal Radius as the shortest distance from the origin to the boundary of the BOA. (quickly describe the minimal Radius here, also do a mathematical formulation).

While this approach has a strong foundation in nonlinear dynamics theory, it quickly becomes infeasible for more complex applications. In the paper the most complex system had six DOFs and the computational time was already high (numbers??). The system on which most of the test were executed is a simple model of a quadruped which aready has 78 degrees of freedom implying a 156 dimensional phase space. If one for example wanted to discretize the space with just 3 nodes along each dimension, $2.697x10^74$ initial conditions would need to be evaluated. We clearly need to restrict ourselves to a subset of the phase space. One approach would be choosing only small number of dimensions of the phase space, however then the robustness measure is valid only for a part of the system, making it applications less useful, as one could only look at the robustness of a single leg for example. (too long)

(The thing is, we don't actually want to discretize the phase space. That's why we use the minimal Radius in the first place. Rather we need to achieve a certain sample density on the surface of the hypersphere defined by the minimal radius in order for the minRad algorithm to work properly. As the surface of a n-dim hypersphere actually becomes quite small for very high dof, this should actually not be an issue... What is an issue, however is that the scaling of all of the different coorinates q_i matters. Findin a scaling such that relevant disturbances have more of an effect on the robustness measure seems quite tough. It would be better to restrict the choice of disturbances/I.C. derived from disturbances. One argument from a practical perspective is that one needs to verify the results from the minRad algorithm somehow (we did that! overlay!) this becomes tough for any D with $dim(\mathsf{D}) > 2$ already, as visualization becomes difficult, but then we can point to the issue of subspaces of the phase space only having limited meaning, i.e. robustness of only a small part of the system

A different way of choosing a subset of disturbances is needed.

Here we remind ourselves that the disturbed states in the phase space are results of underlying disturbances. Instead of sampling initial conditions, we propose sampling and applying disturbances to the model in simulation. We define a d-dimensional disturbance space D from which we will sample disturbances and in which we will measure robustness. The idea is to apply the same concept of a minimal Radius in this space to approximate the size of the set of disturbances the system can recover from in order to measure robustness. Note that recovery of the system under a disturbance will still be evaluated by checking convergence of the state to the attractor in the phase space.

(mathematical formulation of robustness measure)

"find the minimal distance to boundary of converging set, where converging set is all elements of DS, for which trajectory will go to attractor in limit within the phase space"

The goal of finding a robustness measure can be though of as $RM : \mathbb{R}^p \rightarrow \mathbb{R}$, i.e. a mapping from the p-dimensional parameter space to a scalar value.

3.1.4. Parameter space and Disturbance space

In order to apply the concept of the minimal radius as a measure of robustness in the disturbance space D, we must impose some conditions on it. In the phase space, the origin is generally chosen such that the attractor of interest lies on or in close viscinity to it. This means sampled points $s_\mathsf{D} \in \mathsf{D}$ with a small norm represent no or very small disturbances, guaranteeing recovery if s_D is chosen small enough. The minimal Radius approach builds on the fact that the origin lies within the converging set and the it's boundary is reached at some point when moving farther away. (said two sentences before) For this reason me must ensure that in D the origin also represents the state being on the attractor, implying no disturbance. Each element of the disturbance space is just a d dimensional vector of coordinates representing some disturbance. To impose the above condition we must ensure that the 0 vector in D represents no disturbance. Taking oscillations as an example we might want to represent them in a 2 dimensional DS with amplitude and frequency as the coordinates. An oscillation with either 0 amplitude or 0 frequency implies no oscillation at all, making this choice valid. An invalid example of a coordinate is the direction of a force applied. If the angle of attack is 0, the disturbance itself is clearly non zero, breaking our condition. Here we define the $set\ of\ recoveries$ D to be quantified by the minimal radius as the equivalent to the BOA $\mathit{phase\ space}$.

(actually formulate it mathematically), aka "disturbed trajectory" converges to $\mathbf{A}$.

For any set of valid coordinates, their scaling wrt each other turns out to stronly affect the resulting robustness measure. Just choosing different units for any coordinate will stretch D along the corresponding dimension, changing the shape of the converging set and in turn changing the minimal Radius to its boundary. This issue is alluded to in the paper REF by allowing the minimal radius to trace an elliptical shape, which corresponds to rescaling one dimension. There seems to be no comprehensive solution to this issue, which is why we suggest finding a well posed D by trial and error and not changing it as long as possible. Conversely, this problem may also be leveraged for controlling later optimization of robustness. If the boundary of the converging set is a given distance away along a dimensions, shrinking that dimension will move that boundary closer to the origin, making it more likely that the minimal Radius lies in that direction. With this the scaling of the coordinates could be seen as a weighting of how important robustness against that part of the disturbance is. (illustration)

For the aforementioned optimization, we define the parameter space P to perform the optimization in. Each element of P is a vector of parameters describing some parts of the underlying system. A different vector of parameters will fundamentally change the system and for each element of P, robustness can be evaluated. Eventually we want to find the vector of parameters for which robustness is maximized. We can formulate the optimization problem:

Note that the choice of P is fundamental for the results.

3.2. Code Implementation

This section details the implementations of the robustness measure and particular challenges that were encountered and overcome. (rephrase)

3.2.1. Framework Overview

differential equations are implicitly represented as simulation objects in dde. Their parameters can be set. The state x as well. Use the solver to compute the trajectory under a disturbance for a given amount of timesteps. check if trajectory converges or diverges. Do this repeatedly with initial conditions sample talk about how knowing the exact shape of the basin of attraction is not necessary for optimization of the robustness, with this and because of the additional work needed to implement the cell mapping algorithms, the method with full trajectories was chosen. For this we need a solver. boom. Great segue!

A framework was built up to take in all necesseaty information, run minrad with multithreading, .. (this is kinda the flowchart I wanted to put at the beginning of the code implementation section)

... it was built up such that everything was implemented and only (insert two relevant) functions needed to be specified for the particular application, in addition to specifying all the relevant variables (ds, ps, boundaries, tolerances, etc.). Maybe state this more as: It is advised to build a framework that can handle general cases of sampling, convergenc evaluaion and where only (insert relevant blocks) need to be specified.

Written in c++

We choose the method from REF over cell mapping because of it's simplicity in implementation with the given solver. I.e. cell mapping also needs the solver, but in addition also a lot of other structure around it.

specify laikago as an example. Or rather briefly describe what it is and use it to illustrate issue that might arise,

here we want a nice block diagram.

Also here we want

platform (laptop) stats

3.2.2. Solver

One of the hardes part of the process is actually finding the state trajectories of the system. Finding an explicit analytical solution to the IVP is possible for simple cases, but very hard if not impossible for more complex systems. Numerical solvers represent a general approach to approximate the trajectories. For this the differential equations of the IVP are integrated over small time steps Δt to approximate future states. Iterating this process gives a discrete approximation of the continuous state trajectory. Note that we represent general discrete points in time with t_n. The state trajectory is therefore just a set of states $\mathbf{x}(t_n)$ at time points $[t_0, t_1, \ldots, t_n]$ with $t_{i+1} - t_i = \Delta t \quad \forall\ i \in [0, n-1]$. A smaller time step will result in a more accurate approximation, however it will take more computational effort to progress through the same amount of time. It is advised to find a time step that is a sufficient compromise between accuracy and cumputational time and keeping it fixed from there on. Here $\Delta t = 0.01$ was found to be appropriate. Still numerical errors and will always be present, fundamentally limiting the precision that can be achieved.

For this project CRL's Differentiable Dynamics Engine (DDE) was provided, doing most of the heavy lifting. It simulates mechanical systems by considering multibody dynamics and contact forces. The latter are modeled via spring dampener elements, the dynamics of which again depend on the time step so it is imperative to keep it constant between different test. Here, it was f..? DDE represents the system states by generalized coorinates and provides the generalized accelerations $\ddot{\mathbf{q}}(t_n)$ in addition to $\mathbf{q}(t_n)$ and $\dot{\mathbf{q}}(t_n)$ for ever iteration of the trajectory. One particularity to keep in mind is the choice y is the vertical axis, while x and z lie in the horizontal plane.

(anything else important to note?)

3.2.3. Detecting Attractors

In order to evaluate the convergence of a trajectory resulting from a disturbance, first the attracting set of states must be found. For this we follow the nonlinear dynamics definitions for a general approach.

When setting up a simulation, especially when compliance is present, the undisturbed initial state resulting from the construction is rarely part of an attractor. To remedy this simulations of the undisturbed system were run to let the state trajectory settle to a valid attractor. As in (REF seciton), we distinguish between fixed points and limit cycles.

In the case of fixed points, it suffices to check every state along the computed trajectory for the fixed point condition (REF section for fixed points). Whith DDE checking this is trivial as the generalized accelerations $\ddot{\mathrm{q}}(t_n)$ are provided. The first state at time $t_n = t_{fp}$ for which $\dot{\mathrm{q}}(t_{fp}) = \mathbf{0}$ and $\ddot{\mathrm{q}}(t_{fp}) = \mathbf{0}$ hold, can be saved as an attractor in the form of $\mathbf{x} = \begin{pmatrix}\mathbf{q}(t_{pf}) & \mathbf{0}\end{pmatrix}^{\mathsf{T}}$, for disturbed trajectories to be compared to. If $\ddot{\mathrm{q}}(t_n)$ is not eailsy obtainable, one may check a number of states $\mathbf{x}(t_n),\ t_n > t_{pf}$ for $\dot{\mathrm{q}}(t_n) = \mathbf{0}$. This does not guarantee that state $\mathbf{x}(t_{pf})$ is a fixed point, but it makes it more probable the more additional states are checked.

(Insert example graph of a 1d mass spring system over time.)

Finding limit cycles is a bit more challenging as they may show chaotic behaviour and because of the discretization of time, aliasing effects could arise. For practicality we make the following assumptions. First we choose the forcing period T of the system to be an integer multiple of the time step $T = m \cdot \Delta t, m \in \mathbb{N}$ to mitigate aliasing. In addiontion we assume the period of the limit cycle to coincide with the period of the forcing period that causes the periodic behaviour in the first place. So if we have a controller tasked to move a leg in a periodic fashion, we assume the actual end effector trajectory to show that same period, with out phase shift. Note that this is not true generally and needs to be verified. In this scenario we can apply the conditions for a limit cycle in continuous time to the discrete case directly. If a state $\mathbf{x}(t_n) = \mathbf{x}(t_n + T)$ is found, the set of states $\{\ \mathbf{x}(t_n) \mid t_n \in [t_i, t_{i+1}, \dots, t_i + T]\ \}$ can be saved as the attractor. Denote $t_{lc,0}$ as the time where the trajectory first reaches the limit cycle. To rule ot numerical errors one may want to verify the attractor by choosing one period of states after the initial occurance of the attractor and compare each and every state $\mathbf{x}(t_{lc,0} + i\Delta t) = \mathbf{x}(t_{lc,0} + i\Delta t + T)\ \forall\, i \in \{1, 2, \dots, m-1\}$.

(insert example graph for forced 1d pendulum)

Another approach for detecting limit cycles tested is recasting the problem such that it is equivalent finding a fixed point. For systems with a dominant oscillation of a particular coordinate, this can be done using Poincare Sections. The ides here is to reduce the trajectory by only picking out states at which the oscillating coordinate is at a specific value and it's derivative has the same sign. This essentially removes the oscillations and the fixed point condition can be applied to the reduced trajectory. This approach is well-founded in the theory of dynamical systems, however it is infeasible once there exist oscillations along multiple coordinates. In addition because of the reduction, much longer trajectories are needed, ultimately increasing computational effort, which is why this approach is not recommended.

(maybe some graphic about poincare section attempts)

Note that when comparing two states $\mathbf{x}$ and $\mathbf{y}$ numerically, checking for equality can give misleading results because of rounding errors. It is rather suggested to check $|| \mathbf{x} - \mathbf{y} || < \epsilon$, for some small positive ϵ and some vector norm $|| \cdot ||$. For the norm, the root mean square error

$$|| \mathbf{x} - \mathbf{y} ||_{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \tag{3.14}$$

was used. This approach is reflected in Figures (the two above, that have yet to be created) by error margins.

It is also important to inspect the resulting attractor. As generally there exist multiple attractors it has to be verified that the found attractor does represent the correct undisturbed behaviour. In initial testing a square cloth model (see pic.) made up of mass spring elements was made dynamic by applying oscillations at the top two corners. When computing the limit cycle for this system, the detected attrator seemed to be shifted from its expected location (oscillations about the "hanging down" position). Visual inspection using a graphical interface for dde showed, that for some initial condition, the entire cloth stabilized in the inverted position, which was reflected in the detected attractor. This effect is known from inverted pendulums, where applying oscillations stabilizes the inverted position (REF). This exemplifies the need for verification.

(add figs of cloth in hanging and inverted position)

For certain systems and application, fully determining all coordinates of the attractor might turn out to be difficult or simply excessively precise. These cases are further detaile in the following section. (don't like this last part)

3.2.4. Evaluating Convergence

Given an attractor, evaluation of convergence simply follows definition 3.12. Clearly we cannot be letting time go to infinity, however by the definition 3.7, we can deduce that if a state $\mathbf{x}(t_{conv})$ lies on the attractor, all future states $\mathbf{x}(t_n),\ t_n > t_{conv}$ will as well, fullfilling the definition of convergence. Evaluating convergence therefore simplifies to finding a state on the trajectory that coincides with an element of the attractor at any point in time. If undisturbed after t_{conv}, the state should stay on the attractor for all future times. This holds for all types of attractors. Similar to the attractor detection, additional states $\mathbf{x}(t_n),\ t_n > t_{conv}$ should be verified to also be on the attractor to make the results more robust to numerical errors. If future disturbances will be applied or disturbances are continuous, future states need to be verified until either the disturbances or the simulation itself stops, i.e. t_{max} is reached.

Until now, the approaches were hevaily based on the nonlinear considerations outlined in sections (REF). Ultimately, how one determines wheather the system converges to the attractor under disturbances may be implemented in any fashion that works. One may come up with many case specific simplifications. One applied to testing of the quadruped is oulined belof.

(this following part should probably be cut) Also from here on convergence and divergence will be dropped in favor general of recovery or failure of the system under a disturbance, as we will stray further from the nonlinear dynamical rigor with the following simplifications.

In systems with high degrees of freedom, defining precise attractors may be too restrictive. In testing, when applying disturbances to the standing quadruped, "convergence to the attractor" really just means the quadruped not tipping over. For this there doesn't just exist one single valid state as any translation along the horizontal plane or rotations about the vertical axis leave the robot still standing upright. In these cases it is simpler to actively check wheather the system diverges, i.e. tips over, as it already starts on the attractor and we intuitively know that it won't stand up by itself and can therefore not return after leaving it. In this specific case this also meant that once "tipping over" was detected, the individual simulation was stopped. Divergence was detected by verifying that the hight of the core of the robot was suffitiently high above the ground and both pitch and roll didn't deviate much from the original position. We do still compute an attractor, however only choose the three relevant coordinates of the core for future states to be compared to. (maybe put this part to the results?)

Also that with high DOF systems, as there are many different coordionates, small errors might add up in the norm and convergence following (REF) is never detected.

especially in multibody dynamics it might suffice to only track the generalized coordinates of one body, the core for example. This needs to be decided on a case by case basis. Relate this to laikago, as we don't care about the particulat leg pose, as long as the robot does not tip over.

The nonlinear dynamics approach with evaluating convergence of the system state to the attractor will always work in theory (enough computational power and precision), but ultimately, if there exists a computationally simpler way to decide whether the system recovers from disturbances, it should be chosen over the rigorous approach.

3.2.5. minRad algorithm

With the tools outlined above we can evaluate the response of a system to any individual disturbance. In order to finally quantify robustness, we need to measure the minimal radius of the set of recovering responses as defined in section (REF). We approximate this value following (REF to appropriate figure), applying the bisection algorithm to an initial upper guess $R_{curr} = R_{max,init}$ and iteratively updating R_{curr} until the desired accuracy is reached. For any iteration, n_s number of disturbances are randomly sampled on the surface of the d dimensional hypersphere of radius R_{curr} in D. This achieved by enforcing $\| \ s_{\mathsf{D}} \ \| = R_{curr}$ for all samples at that iteration. For any R_{curr}, if it lies below the true minimal radius, all disturbances s_D sampled at R_{curr} and applied to the underlying system will result in recovery. Conversely, a single failure of recovery indicates that the true minimal radius must be smaller than R_{curr}. The bisection algorithm requires an inital lower bound as well, but as the minimal radius R_m must by construction always be positive, 0 is a generally valid choice here. As the algorithm is bisecting the domain of possible values at every iteration, we can compute the resolution h of the resulting robustness measure after n_i iterations as: $h = R_{max,init} \cdot \frac{1}{2}^{n_i}$. The choice of n_i depends on the overarching application of the robustnes measure, i.e. how much precision is needed. The other parameter to be considered is the number of samples n_s at every iteration. The sampling is stochastic in nature, so no determenistic results can be guaranteed. A n_s that is too low might miss some disturbances that result in failure of recovery of the system, leading the algorithm down a wrong path. High n_s on the other hand lead to an unnecessary increase in computational effort. n_s should be chosen in a way that with a set number of iterations, the resulting robustness measures are consistent when computed repeatedly. Monotonic evolution of R_{curr} over all iterations indicates issues. If R_{curr} monotonically decreasing, it implies that it is always larger than the true minimal radius. Here either $R_{max,init}$ needs to be decreased or the number of iterations increased. On the other hand, monotonical increase implies $R_{max,init}$ was chosen too small.

As the scaling of the dimensions in D w.r.t. each other has a significant impact on the resulting robustness measure (as oulined in Sec. ..), the algorithm was implemented for a general case. Given a d dimensional D and lower and upper bounds $b_{low,i}, b_{upp,i} \in \mathbb{R},\ i \in \{1, 2, \dots, d\ \}$ for every dimension, the domain restricted to $[0, 1]$ for the minRad algorithm, and scaled up to $[b_{low,i}, b_{upp,i}]$ when applying the disturbances for the system. $\mathsf{R}_{\mathsf{max,init}}$ was always set to 1, restricting $\mathsf{R}_{\mathsf{curr}}$ to $[0, 1]$ and giving different robustness measures some degree of comparability. Changing to a different D then only requires definining it's dimensionality and corresponding boundaries.

(verifying minRad results by discretizing disturbance space and brute force exploring the space by evaluating every single node.)

(the graphic for the bisection algorithm would probably already be in previous work. I feel like it would be better located here?, also I guess we don't use the same one as we are applying it to D)

visualizations Plot variance as a function of nsamples for some example.

3.2.6. Optimization

At this point it is possible to change the system parameters in order to increase its robustness measure. Without applying any further algorithms, one may discretize P in every dimension and compute the robustness for every possible combination of parameter values. The optimal parameters are the ones with the maximal robustness within the resolution of the chosen discretization step size h. This approach is computationally expensive and becomes infeasible for high (p) dimensional P as the computational effort is $\mathcal{O}(\frac{1}{h^d})$ with smaller h yielding higher precision.

A second approach is utilizing optimizers, which can drastically reduce the number of robustness computations needed to get (close) to an optimum. As derivatives of the robustness measure are not available, the choice fell on evolutionary algorithms. In particular, the Covariance Matrix Adaptation Evolution Strategy (short CMA-ES) was used with algorithm parameters being kept as suggested by the documentation. Combining both approaches enables comparison and verification of either one.

3.2.7. Boundaries of D and P

The boundaries of D are relevant for the minRad algorithm during sampling of the disturbances. Because of their fundamental effect on the robustness measure, they have to be chosen thoughtfully. In the tests carried out D was restricted to 2 dimensions for simplicity and ease of visualization, making finding appropriate boundaries by trial and error feasible. This would be significantly more challenging for D with of high dimensions. The approach was to find a balanced set of boundaries where at the minimal radius all elements $s_{\mathsf{D},i} = 0.5$, implying $R_{minRad} = \left\|s_{\mathsf{D}}\right\| = \sqrt{\sum_{i=1}^{d} s_{\mathsf{D},i}^2} = \frac{\sqrt{d}}{2}$ for some set of system parameters. These boundaries were then used when computing robustness measures for different sets of parameters. Here it shall be noted that some elements of disturbance may be negative. In order to keep the framework simple, these cases were handled by randomly negating those values and scaling them appropriately.

The boundaries of P dictate the possible range of parameters and are particularly relevant for the optimizer. Basic functionality must be given within the boundaries as detecting the attractor is based on that assumption. Mostly they dictated by the underlying system in form of physical constraints, however there may exist parameters that are valid in this sense but lead to the system not expressing the desired behaviour in the first place. (2=>1 sentences)

3.2.8. Multithreading

The computation of the simulations takes up a bulk of the computational time. As for every minRad iteration a number trajectories need to be found, this process benefits from parallelization. A simple multithreading pipeline was implement, computing n_{th} simulations on n_{th} threads in parallel. For this to be effitient, it is necessary for the functions to give feedback to the pipeline on whether the simulations are done. Completed simulations are restarted with different disturbance samples until either all n_s samples are computed or a failure of recovery is detected. In this case all threads are stopped and the minRad algorithm proceeds to the next iteration. With this the computational time can approximatiely be reduced by a factor of n_{th}, assuming there exist n_{th} cores on the platform.

3.2.9. Application to specific systems

First iterations of the implementation were naively done in a "top down" fashion that was specific to the particular system. Later this was changed such that the relevant (and changing) functions were isolated and could quickly be redefined for new applications.

The first of these is the function applying the parameters to the system. This is generally as simple as changing some variables but fully depends on and needs to be customized for the specific system at hand.

The second is arguably the more essential one. This is the function called by the multithreading pipeline. It needs to run the simulation, apply disturbances, whether continuously or only initially, and evaluate the convergence, returning the boolean result. Concrete examples are listed in the TESTs section.

analysis of high dof motion tricky (bad 3d image), rather plot every coordinate over time (image).

3.3. Tests

Platform (describe laptop specs)

Computations were performed on a ...

Initially, simple systems were analyzed to gain familiarity with dde and it's implementation. Visualizing 3d time dependent trajectories all in one plot becomes very convoluted quickly as can be seen in Fig .. . This is especially true when working with multibody systems. Attempts were made to apply principle component analysis to reduce the dimensionality, however this only yielded usable data when working with multibody systems where all bodies moved in a coordinated fashion. In these cases picking out and tracking individual bodies yielded

Another issue is that for effitient computation, yada yada let's see how much we want to talk about PCA anyways.

It is much more practical to plot the evolution of individual coordinates over time, as trends are much more evident and

"While visually appealing, these graphs were less useful for actual inspection of the trajectories."

Initial familiarization with data. plot coordinates over time individually and maybe more importantly, in 2d plots. Compare to convoluted 3d attempts. maybe failed approaches with pca. better to just track less coordinates

Results of detecting attractors poincare example

Results of detecting convergence

Introduction to laikago (probably should do that earlier?) high dof rather long computational time when solving trajectories (compared to simpler systems, duh) how it's implemented (no walking yet) laikago, that it doesn't do anything but try to keep its limbs in the predefined state.

optimization of robustness examples

Laikago Droptest for this and the high precision example, do the buildup. DS, minRad, PS, CMAES for this and the next example. Laikago Swingtest note that here we kind of broke the mathematical framework as technically the underlying diff eq were changed. But we just ignored this, NO effects of this need to be investigated further.

Maybe we could do one high precision, high resolution DS swingtest to see if we can spot resonance

This is kind of a twist on the concept as we are starting at a fixed point and continually applying disturbances to see for what disturbances the trajectory conveges, which in this case is expressed by the system staying at the initial state. (now is explain in earlier sections already.)

4 Conclusion and Outlook

physical explanation why there can't be that much optimization for swing and droptest

Implication that the system must be dynamical in nature (i.e. it must evolve). So a rudimentary control strategy must be implemented or outside forced must be applied. Don't quite know where to put this.

further explore effects of combining disturbences of differnent sorts and the effects of the choice of bounds.

applying to systems of high dof with full phase space. Is it even possible? How much can the code be optimized?

apply to physical dimensions of systems (intuitively more drastic effects)

using the phase space as the disturbance space but heavily restricting it (actuator limits, improbable constellations etc)

Optimizing with limit cycles as attractors.

solver (or rather finding the trajectories in general) will always be the largest bottleneck and finding methods to reduce number of trajectories to be evaluated or the computational time per trajectory would be very benefitial

Seems unlikely that this will ever be plug and play. Lots of tuning, lots of fiddling around. But there could definitely be applications, especially in novel systems where rigorous analysis is lacking.

High complexity is still an issue.

A

Information For The Few (Appendix)

dot = f(q,qdot) qddot = g(q,qdot)

order reduction:

x = [q,qdot] => xdot = [qdot, qddot] = F(x) = [f(x), g(x)]

q,qdot,qddot

A.1. Foo Bar Baz

A.2. Barontes

A.3. A Long Table with Booktabs

Table A.1.: *A sample list of words.*

ID	Word	Word Length	WD	ETL	PTL	WDplus
1	Eis	3	4	0.42	1.83	0.19
2	Mai	3	5	0.49	1.92	0.19
3	Art	3	5	0.27	1.67	0.14
4	Uhr	3	5	0.57	1.87	0.36

continued on next page

Table A.1.: *(Continued)*

ID	Word	Word Length	WD	ETL	PTL	WDplus
5	Rat	3	5	0.36	1.71	0.14
6	weit	4	6	0.21	1.65	0.25
7	eins	4	6	0.38	1.79	0.26
8	Wort	4	6	0.30	1.62	0.20
9	Wolf	4	6	0.18	1.54	0.19
10	Wald	4	6	0.31	1.63	0.19
11	Amt	3	6	0.30	1.67	0.14
12	Wahl	4	7	0.36	1.77	0.42
13	Volk	4	7	0.45	1.81	0.20
14	Ziel	4	7	0.48	1.78	0.42
15	vier	4	7	0.38	1.81	0.42
16	Kreis	5	7	0.26	1.62	0.33
17	Preis	5	7	0.28	1.51	0.33
18	Re-de	4	7	0.22	1.56	0.33
19	Saal	4	7	0.75	2.10	0.43
20	voll	4	7	0.48	1.82	0.24
21	weiss	5	7	0.21	1.59	0.36
22	-ger	5	7	1.16	2.69	0.59
23	bald	4	7	0.18	1.56	0.19
24	hier	4	7	0.40	1.70	0.43
25	neun	4	7	0.17	1.52	0.26
26	sehr	4	7	0.36	1.85	0.43
27	Jahr	4	7	0.50	1.82	0.43
28	Gold	4	7	0.04	1.35	0.20
29	Ter	5	8	0.15	1.39	0.59
30	Tei-le	5	8	0.30	1.71	0.46
31	Na-tur	5	8	0.18	1.59	0.41
32	Feu-er	5	8	0.30	1.71	0.45
33	Rol-le	5	8	0.15	1.46	0.45
34	Rock	4	8	0.29	1.68	0.25
35	Spass	5	8	0.28	1.64	0.32
36	Gte	5	8	0.49	1.75	0.66
37	En-de	4	8	0.36	1.72	0.33
38	Kunst	5	8	0.26	1.59	0.35
39	Li-nie	5	8	0.45	1.88	0.63
40	Bme	5	8	0.48	1.92	0.45
41	Bh-ne	5	9	0.94	2.48	0.62
42	Bahn	4	9	0.21	1.62	0.42
43	Br-ger	6	9	0.38	1.70	0.65
44	Druck	5	9	0.60	2.03	0.31
45	zehn	4	9	0.41	1.84	0.42

continued on next page

Table A.1.: *(Continued)*

ID	Word	Word Length	WD	ETL	PTL	WDplus
46	Va-ter	5	9	0.36	1.78	0.40
47	Angst	5	9	0.29	1.56	0.35
48	lei-der	6	9	0.13	1.47	0.52
49	hfig	6	9	0.82	2.31	0.52
50	le-ben	5	9	0.38	1.85	0.40
51	aus-ser	6	9	1.20	2.26	0.57
52	be-vor	5	9	1.28	2.75	0.39
53	Kai-ser	6	9	0.92	2.37	0.53
54	Markt	5	9	0.23	1.58	0.28
55	Os-ten	5	9	0.21	1.54	0.48
56	Krieg	5	9	0.33	1.67	0.50
57	Mann	4	9	0.31	1.47	0.25
58	Hal-le	5	9	0.24	1.65	0.45
59	heu-te	5	9	0.44	1.87	0.46
60	in-nen	5	10	0.36	1.80	0.45
61	Na-men	5	10	0.28	1.72	0.41
62	jetzt	5	10	0.70	2.07	0.32
63	kei-ner	6	10	0.28	1.62	0.53
64	Schu-le	6	10	1.02	2.12	0.48
65	Ar-beit	6	10	0.34	1.70	0.52
66	An-teil	6	10	0.27	1.63	0.53
67	di-rekt	6	10	0.67	2.04	0.47
68	vor-her	6	10	0.78	2.25	0.47
69	wol-len	6	10	0.44	1.85	0.51
70	Kampf	5	10	0.70	1.96	0.27
71	dern	6	10	1.18	2.62	0.65
72	lau-fen	6	10	0.21	1.64	0.52
73	Eu-ro-pa	6	10	0.23	1.53	0.66
74	statt	5	10	1.61	2.86	0.39
75	Wes-ten	6	10	0.29	1.60	0.54