The Worst Toolbox Documentation

Stephanie Tsuei

Version 0.1a

1 Introduction

This toolbox implements the algorithm described in:

> J. E. Tierno, R. M. Murray, J. C. Doyle, and I. M. Gregory, "Numerically Efficient Robustness Analysis of Trajectory Tracking for Nonlinear Systems," *J. Guid. Control. Dyn.*, vol. 20, no. 4, pp. 640-647, Jul. 1997.

which uses theory presented in:

> A. Bryson and Y. Ho, *Applied optimal control: optimization, estimation, and control*, vol. 59, no. 8. 1975.

1.1 The Robust Trajectory Tracking Problem

Consider the dynamical system pictured in Figure 1.1 with dynamics

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{U}(t), \mathbf{u}(t), \mathbf{v}(t), \boldsymbol{\delta})$$

and outputs

$$\begin{aligned}\mathbf{Y}(t) &= g(\mathbf{x}(t), \mathbf{U}(t), \mathbf{u}(t), \mathbf{v}(t), \boldsymbol{\delta}) \\ \mathbf{z}(t) &= h(\mathbf{x}(t), \mathbf{U}(t), \mathbf{u}(t), \mathbf{v}(t), \boldsymbol{\delta})\end{aligned}$$

where $\mathbf{x}(t)$ is the state, $\mathbf{U}(t)$ is a nominal input signal, $\mathbf{u}(t)$ is a disturbance signal, δ is a vector of uncertain parameters, Δ is an uncertain block of unit norm, and $\mathbf{v}(t)$ and $\mathbf{z}(t)$ are signals representing any possible unmodeled dynamics. Therefore, the nominal trajectory, is given by

$$\mathbf{Y}_n(t) = g(\mathbf{x}(t), \mathbf{U}(t), 0, 0, \boldsymbol{\delta}_n)$$

where $\boldsymbol{\delta}_n$ are the nominal values of the uncertain parameters and the quantity $\mathbf{y}(t) = \mathbf{Y}(t) - \mathbf{Y}_n(t)$ is the error signal.

We make the following assumptions:

- $\|\mathbf{v}(t)\| = \|\mathbf{z}(t)\|$
- $\mathbf{v}(t)$ and $\mathbf{z}(t)$ can be written in block form $\mathbf{v}(t) = \begin{bmatrix}\mathbf{v}_1(t) & \mathbf{v}_2(t) & \dots & \mathbf{v}_p(t)\end{bmatrix}^T$ and $\mathbf{z}(t) = \begin{bmatrix}\mathbf{z}_1(t) & \mathbf{z}_2(t) & \dots & \mathbf{z}_p(t)\end{bmatrix}^T$. The total number of blocks in $\mathbf{v}(t)$ and $\mathbf{z}(t)$ must be the same, but $\mathbf{v}_i(t)$ and $\mathbf{z}_i(t)$ can be of different dimensions.
- $\|\mathbf{u}(t)\|$ is a known constant scalar M if there is only one disturbance input. If there are multiple disturbance inputs $\mathbf{u}_1(t)$, $\mathbf{u}_2(t)$, then each disturbance input i has a known norm M_i.

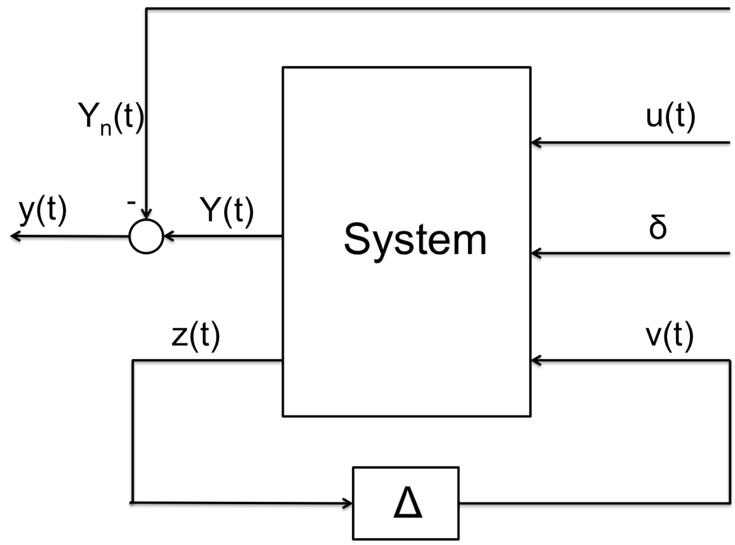


Figure 1: A block diagram dipicting the robust trajectory generation problem.

- $\underline{\boldsymbol{\delta}} \leq \boldsymbol{\delta}_n \leq \bar{\boldsymbol{\delta}}$ (elementwise) where $\underline{\boldsymbol{\delta}}$ and $\bar{\boldsymbol{\delta}}$ are known constants.
- We are only interested in the finite time interval (t_i, t_f) and for a vector-valued time signal $\mathbf{w}(t)$,

$$\|\mathbf{w}(t)\| = \left(\frac{1}{t_f - t_i}\int_{t_i}^{t_f} \mathbf{w}^T(t)\mathbf{w}(t)dt\right)^{\frac{1}{2}}$$

Given the information above, this toolbox computes the functions $\mathbf{u}(t)$, $\mathbf{v}(t)$ and the value of $\boldsymbol{\delta}(t)$ that *maximizes* the quantity below:

$$J = \|\mathbf{y}(t)\| = \left(\frac{1}{t_f - t_i}\int_{t_i}^{t_f} \mathbf{y}(t)^T\mathbf{y}(t)dt\right)^{\frac{1}{2}}$$

1.2 Notes on the Algorithm

1. This algorithm is a power algorithm that is not a contraction. Therefore, it is not guaranteed to converge, but does in practice.
2. The algorithm searches for local extrema (local minima *and* local maxima). Therefore the value of J that is computed is a lower bound on the actual value of J.
3. Initial values for the worst case disturbance are randomly generated and the algorithm is sensitive to the initial value of the disturbance. Therefore, it is helpful to run the algorithm several times and plot a histogram of the values that it computes, as seen in the examples.
4. Let $\mathbf{y}_1(t) = \begin{bmatrix} y_{1,1}(t) & y_{1,2}(t) & \dots & y_{1,m}(t)\end{bmatrix}$ be the error output signal from the previous iteration and $\mathbf{y}_2(t) = \begin{bmatrix} y_{2,1}(t) & y_{2,2}(t) & \dots & y_{m,2}(t)\end{bmatrix}$ be the error output signal from the current iteration. The algorithm declares that it has converged when

$$\max_{i=1,2,\ldots,m} \frac{\sum_{j=1}^{T}(y_{1,i}(j) - y_{2,i}(j))^2}{\sum_{j=1}^{T} y_{1,i}(j)^2} < e$$

where e is a small positive number and T is the number of timesteps.

1.3 System Requirements

Requirements:

- Matlab R2012b or later
- Simulink

The toolbox may work on some earlier versions of Matlab and Simulink, but was developed on Matlab R2012b.

1.4 Install Instructions

To install the Worst Toolbox, place the folder `worst/` anywhere on your local machine and add it (but not its subdirectories) to the Matlab path. This is equivalent to the running the command:

```
addpath('/path/to/robust/worst')
```

2 Using Worst

2.1 Simulink Model Configuration

The Worst toolbox makes some assumptions about the configuration of Simulink models:

Order of Inputs

The program assumes that models have up to four input signals and two output signals. Each of the signals may have arbitrary dimension to accomodate multiple inputs, disturbances, and feedback signals. The order assumed is:

1. Nominal inputs, $\mathbf{U}(t)$
2. Disturbance inputs, $\mathbf{u}(t)$
3. Unmodeled feedback inputs, $\mathbf{v}(t)$
4. Uncertain Parameters, $\boldsymbol{\delta}$

Order of Outputs

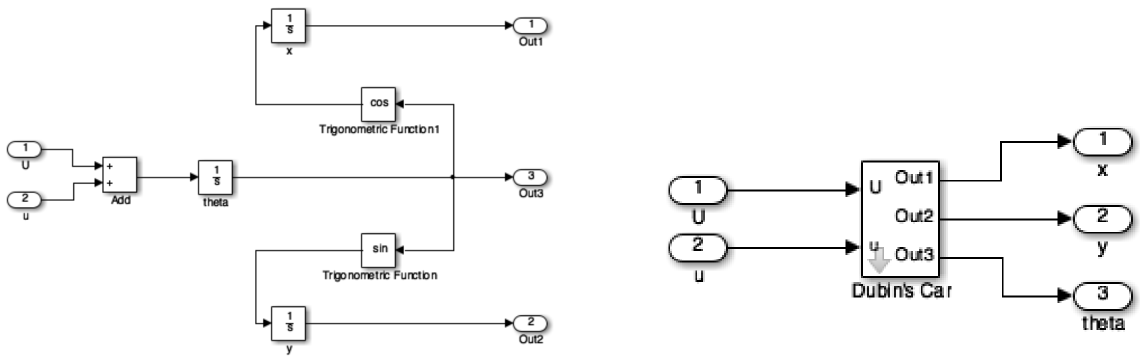
Models are allowed to have an arbitrary number of outputs. If the Simulink model has a total of r outputs, and the first $m \leq r$ outputs are outputs included in computing the performance measure J, then the remaining $r - m$ outputs are unmodeled feedback outputs; the coordinates of z.

2.2 Syntax of `worst`

The syntax for `worst` is:

```
worst(system_name, output_dimension, 'Parameter1', value1, 'Parameter2', value2, ...)
```

`system_name` is the name of the Simulink model (make sure the model is in the MATLAB path!) and `output_dimension` is the dimension of the system's output. The parameters are optional values, described in the table below:



(a) Original Simulink Diagram

(b) Diagram with S-Function

Figure 2: A model of Dubin's Car compiled into an S-function.

Name	Default Value	Description
`ti`	0	Simulation start time in seconds
`tf`	10	Simulation end time in seconds
`params`	`[]`	A p by 3 matrix where p is the number of uncertain scalar parameters. Each row has the format: `[lower_bound, nominal_value, upper_bound]`
`disturbance_specs`	`[]`	A d by 2 matrix where d is the number of disturbances. Each row has the format: `[disturbance_dimension, disturbance_norm]`
`unmodeled_io`	`[]`	A b by 2 matrix where b is the number of unmodeled input-output pairs. The left column has all the dimensions of the unmodeled inputs $\mathbf{v}$ and the right column has all the corresponding dimensions of the unmodeled outputs $\mathbf{z}$.
`max_iter`	40	The maximum number of times to iterate the algorithm when computing a single value of J.
`error_tol`	.01	The value e in section 1.2
`num_iter`	10	The number of different values of J to compute.
`nominal_time`	`[]`	The time axis, a column vector for the nominal input, which is described below.
`nominal_input`	`[]`	A T by q matrix describing the nominal input signal, where T is the length of `nominal_time` and q is the dimension of the nominal input. If a value for this parameter is specified, then so must a value for the nominal time axis.

Optional inputs whose default value is `[]` do not need to be specified and are not necessary for the program to run. If there are no uncertain parameters in the system, do not specify any uncertain parameters.

2.3 Speeding up the Program

The program repeatedly calls `linmod` in order to build and simulate an adjoint system. This greatly slows down the program because Simulink will recompile the model with every call. To hasten computation, compile your Simulink model into an S-function before running the program, as shown in Figure 2.3. The more blocks in the original Simulink model, the greater the improvement in speed.

3 Examples

A few examples are below. Since compiled S-functions are not compatible across all systems, the Simulink models of the examples are not compiled.

3.1 A Very Simple Linear System

3.2 A Less Simple Linear System

3.3 Dubin's Car