Titan User's Guide

Revised: 3/25/04

Model-Based Embedded and Robotic Systems Group, MIT

Introduction

Titan is a model-based executive which provides fault management capability. It uses a plant model to monitor spacecraft health during execution. It is capable of identifying and diagnosing faults, and taking repair actions which allow the mission to proceed.

This document describes the purpose and usage of the following Titan executables:

- ModeEstimation_test.exe
- CompiledME_test.exe
- ModeReconfiguration_test.exe
- Titan_test.exe

The input to all of above listed executables, with the exception of CompiledME.exe, is an MOF file which contains the plant model. For instructions on how to generate an MOF file, refer to the "Compilation of MPL Models in Lisp" document. CompiledME.exe requires a *compiled* form of the MOF file which can be created using HaDMoRe.exe (use –help option for documentation).

Mode Estimation

The Mode Estimation (ME) component of Titan estimates the current system state (a state is an assignment to component modes) based on the plant model, observations and executed commands. If the observations disagree with the expected nominal behavior of the plant, the most likely fault modes will be estimated. It currently comes in two flavors: ModeEstimation_test.exe and CompiledME_test.exe, but both serve the same function.

ModeEstimation_test.exe

To run mode estimation tests on your plant model, go to a command prompt, switch to the directory which contains ModeEstimation_test.exe and the MOF file to be tested, and type

ModeEstimation_test ***modelName.mof***

An interactive menu will be displayed which lists the available options. You can type "**h**" at any time to display this menu. The following is a description of the available options.

a : show all variables

This will display all variables- state, observable, command and dependent variables. For example, one entry in the list may look like the following

[SWITCH.OUTPUT_PIN(2): {HIGH(0):0, LOW(1):0}]

This simply says that the "Switch" component has a variable "output_pin" with variable ID 2. The domain of this variable is HIGH (value ID = 0) or LOW (value ID = 1). The number following the colon indicated the –log of the probability for that value. Thus, 0 indicates a 100% probability.

v : show state variables
This will display only the state variables (modes of components). Note that a + sign in front of a domain entry indicates the value of the mode in the current state, a > sign indicates the current value, while **–** indicates a value which is unreachable based on the transition guards.

t : show Transitions
This will display all possible transitions in the plant model, with associated probabilities.

var val : give command/observation
This will set the current value of an observable or command variable to the given value (var = val), where var is the variable ID and val is the domain value ID. For the example given above, typing
2 1
will set the current value of the OUTPUT_PIN to LOW. If you type "a" now, you will see the > sign in front of the LOW value for the OUTPUT_PIN.

u var val : force current (t), not next (t+1), mode
This is to be used for setting state (mode) variables in the current state (not next state). For example, suppose you have the following mode variable entry:
[SWITCH.MODE(0): {BROKEN(0):1.1, ON(1):1.1, OFF(2):1.1}]
To set the current state to ON, type
u 0 1
If you type "a" now, you will see the + sign in front of the ON mode.

k : set number of estimate trajectories
This will set the number of most likely estimates to be generated. For example, to set the number of most likely estimates generated to three, type
k 3

e : estimate most likely next state
Typing "e" will generate the most likely k estimates for the next state, based on the current state and commands at time t and the observations at time t+1. Note that typing "e" will not set the current state to the best estimate for the next state. This is done by the command "p" described below.

p : progress current state (t) to best estimate for next state (t+1)
This will set the current state (indicated by the + sign) to be the best estimate (the most likely estimate generated by the command "e") for the next state.

Other options include "c" for displaying discrepancies between the next state and the model; "d" for switching debug mode on/off (debug mode displays additional information); "f" for toggling the full estimates/only changed modes; "r" for resetting current commands without progressing to the current estimated mode; “#” for echoing test commands.

q : Quits ModeEstimation_test.exe

CompiledME.exe

Compiled Mode Estimation operates in the same way as the previous version of ME from a user’s standpoint but uses a *compiled* model to more efficiently generate estimates; requiring less space and time. It requires two input files: the base model file (*modelName.bm*) and the compiled object file (*modelName.ccme*). The base model file specifies all the variables and their domains, and the compiled object file contains all the dissents and compiled transitions.

To run compiled ME tests interactively on your plant model, go to a command prompt, switch to the directory which contains CompiledME_test.exe and the input files to be used, and type
CompiledME_test ***modelName.bm modelName.ccme***

It is also possible to load an initialization script, a complete test script, and specify and output file. The initialization script and test script have the same command format as the interactive mode. Details on streaming options can be found by typing
CompiledME_test -*h*

Typing “**h [option]**” anytime will display an interactive menu which lists the available options described below.

var val… : give an observation/command/mode sequence
This will assert variable=value pairs for all types of variable types. The pairs can be specified by variable and value string names or UIDs. Note that if you use this to set mode variables, it will set the mode variable in the leading trajectory and remove all other trajectories. Specifying a set of trajectories can be done using the “**a**” command.

a [option] var val… : give an observation/command/mode/trajectory sequence
This command works the same way as the “**var val**” command above except it is more controlled in the sense that it checks what type of variables you are passing in. It also allows you to supply a set of trajectories. The options are

–s	: for state variables,
–u	: for command variables,
–o	: for observation variables, and
–b #	: for an entire trajectory.

To assert an entire trajectory, you must provide it’s probability as the # and a full assignment to mode variables. Additional trajectories can be added using the same command but must immediately follow the previous trajectory otherwise the trajectories will reset to a single trajectory.

e [option] : estimates the next state of the system
This command determines the leading estimates of the next k most likely trajectories. The options are –e to just generate the estimates without progressing and –p to estimate and progress the system into those estimates.

v [option] : displays variables and other assignments
This command is used to display most relevant information about the current state of the mode estimation engine. The options are

–a	: for all model variables,
–s	: for only state variables,
–u	: for only command variables,
–o	: for only observation variables,
–d	: for all dissents in the model,
–t	: for all compiled transitions in the model,
–e	: for all enabled dissents and compiled transitions,
–c	: for all conflicts, and
–b	: for the current trajectories at time t

The “+” symbol in front of a mode value means distinguishes that assignment as being reachable in the next time step and the “>” symbol distinguishes the current assignments to all variables.

Other commands include **r** which resets the commands and observations that have been asserted since the last state progression, **d #** which sets the debug level to #, and **#** which toggles the echo input (useful when using an input script).

Mode Reconfiguration

The Mode Reconfiguration (MR) component handles fault recovery and determines actions which reconfigure the spacecraft in order to achieve the mission goals.

ModeReconfiguration_test.exe

Modereconfigutaion_test.exe runs a reactive planner, which uses a plant model, an input state and a goal state, to determine actions which achieve the goal state.

To run mode reconfiguration tests on your plant model, go to a command prompt, switch to the directory which contains ModeReconfiguration_test.exe and the MOF file to be tested, and type
ModeReconfiguration_test ***modelName.mof***

Typing “h” anytime will display an interactive menu which lists the available options described below.

v : display variables
This will display all variables, identifying each variable as either state, observable, command (control) or dependent variable. Refer to the ModeEstimation_test.exe documentation above for a description of the display syntax for the variables.

c # # … : assert current state(s)
This is used to set the current state of the plant, which requires setting a current mode for all the state variables in the model (i.e. a mode for each of the components). For example,
c 0 2 4 1 6 0
makes the following assignments: (var ID 0 = val ID 2), (var ID 4 = val ID 1) and (var ID 6 = val ID 0)

g # # … : assert goal state(s)
This is used to specify the goal state.

n : generate next command
This generates the next command which will achieve the goal state.
If the next command is generated successfully, it will display:
Command: (1 var val)
where var is the command variable ID and val is the command ID
If it couldn't determine how to accomplish the goal, it will display:
Command: (2)
If it didn't need to generate any commands, it will display:
Command: (0)

Other options include "r" for displaying states reachable from the current state and "d" for toggling debug mode.

q : Quits ModeReconfiguration_test.exe

Titan

The Titan model-based executive uses both a plant model and a control program to autonomously track the state of the system and drive the plant towards a sequence of high level mission objectives. These objectives originate from the control program and are dispatched by the control sequencer in the form of state goals to the deductive controller. The deductive controller consists of ME and MR; using the plant model to generate estimates and determine sequences of commands needed to achieve the state goals.

Titan_test.exe

Titan_test.exe is a poor name for what is actually the deductive controller module of Titan. It runs both the Mode Estimation (ME) and Mode Reconfiguration (MR) components using the input plant model.

To run Titan tests on your plant model, go to a command prompt, switch to the directory which contains Titan_test.exe and the MOF file to be tested, and type
Titan_test ***modelName.mof***

Typing “h” anytime will display an interactive menu which lists the available options. You can get a description and synopsis for each of the available commands in the menu by typing “h <command>”. For example, to get information on the “c” command, type
h c

v : display variables
This will display all - state, observable, command and dependent - variables. Refer to the ModeEstimation_test.exe documentation above for a description of the display syntax for the variables. Note that probabilities are not displayed in Titan_test.

o : queues an observation to ME
This should be called initially to specify initial state. You can only observe state variables in the initial step. Otherwise, "o" can only be used with observable variables. For example, if we have an observable variable entry OUTPUT_PIN:
SWITCH.OUTPUT_PIN(2): {HIGH[0], LOW[1]}
Then you can queue a LOW observation of the OUTPUT_PIN as
o 2 1

e : queues find_mode_estimate to ME
This is typically used after specifying observations or executing commands, in order to estimate the current state of the system and propagate the system to the estimate. You may use the –f option with this command to estimate the full state:
e -f

g : queues a goal to Mode Reconfiguration

r : queues a reconfiguration request to MR
This will determine the next command which will accomplish a goal state, given a current state estimate. This is similar to the “n” command of ModeReconfiguration_test (see description above), and will output a command (1 var val) if it finds a command, (2) if it doesn’t determine how to accomplish a goal, and (0) if no command is required.

c : queues a command to ME
This can only be used with command variables. Once a command (for example (1 3 2)) is generated by MR, it can be queued to Mode Estimation by
c 3 2

q : Quits Titan_test.exe

Titan_test.exe Flow
Here's a description of the flow of a typical Titan_test.exe test run.

Initialize state using "o <var ID> <val ID>"
Propagate the initialization "e –f"

Then LOOP:

- Queue goals to MR using "g <var ID> <val ID>"
- Get a command from MR using "r"
    - This will print:
      Executing the following commands: (1 var val)
      If it prints (2), it couldn't figure out how to accomplish your goals, and if it prints (0), it didn't need to do anything.
- If you get a command (1 var val), execute the command using "c <var> <val>"
- You can optionally provide some observations now using "o <var ID> <val ID>"
    - The system assumes that anything it doesn't observe again has behaved nominally and become whatever it ought to be after the command execution. Therefore, if you don't observe anything, it will never assume something failed.
- You then finish it off by just running "e" to propagate to the new state estimate
    - You can optionally run "e –f" if you want to see the full state ("e" just prints things that changed)

REPEAT LOOP

You can type "v" anytime to print out a list of all the variables (state, observable, command and dependent)

You can type "h" or "h <command>" anytime to obtain help

You can quit anytime using "q"
