Implementing an example satellite

Controlling a Keithley 2410

H. Wennlöf

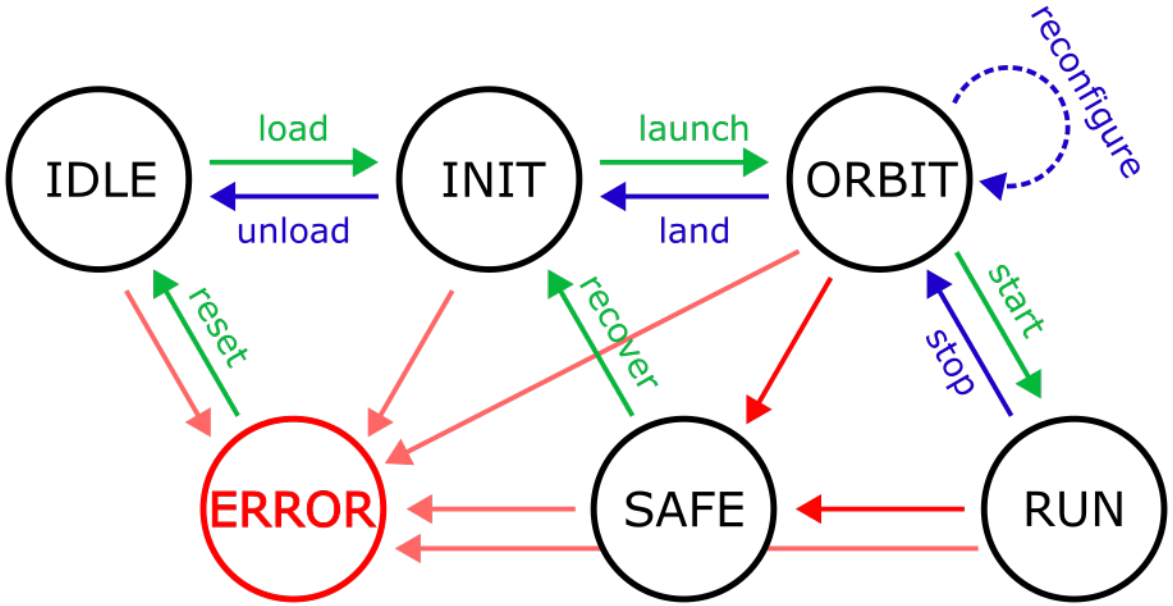
27/11 -23

Introduction

- I had old Python code around to control a Keithley 2400 series using a serial RS232 interface and SCPI/SCPI-like commands
- Prime candidate for something we'll want to use Constellation for in the end
  - E.g. ramping up and down voltages while logging currents
- I made a prototype satellite for this purpose
  - Should work for any 2400 series
  - Made to source voltage, but with small tweaks can be made to source current instead

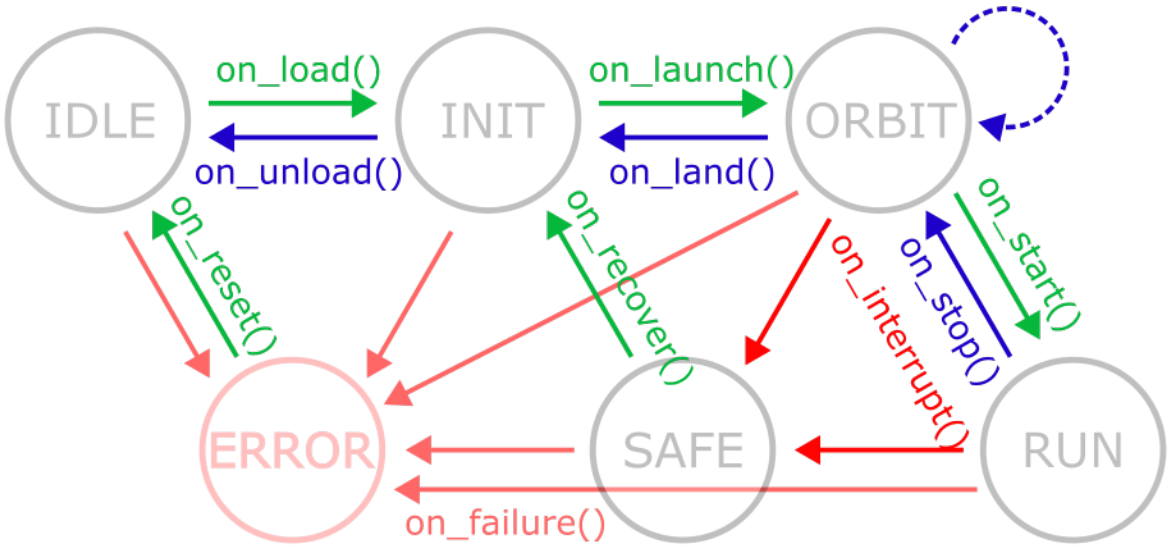
What needs to be implemented?

- Reminder: current state machine
- Need to implement the transitions to create a satellite
- Workflow:
  - Derive from base class
  - Override user transition functions



What needs to be implemented?

- Transitions have corresponding functions
- These and the do_run() can be overridden to create a Satellite derived from the base class
- Note: “reconfigure” would be relevant here (e.g. changing voltage without first going back to the safe level), but we don’t have a consensus on how to do that yet



Implementation

- Using **pyserial** for the communication with the Keithley
- Using YAML (and **PyYAML**) for writing and reading a configuration file
  - But this is of course easy to change, used this mainly bcause it was used in the old code
- The goal:
  - Ramping voltages up and down
  - Publishing current values while the device output is active
- Not all transitions have to be implemented for this satellite, but I did so anyway for demonstrational purposes, just publishing log messages
- Prototype available on the Gitlab

```
class KeithleyControl(Satellite):

    """Constellation Satellite to control a Keithley2410."""

    def __init__(self, cmd_port, hb_port, log_port,
config_file):

        # Initialise
```

Implementation

```
class KeithleyControl(Satellite):
    """Constellation Satellite to control a Keithley2410."""
    def __init__(self, cmd_port, hb_port, log_port, config_file):
    def on_load(self):
        # Create the device, which loads the config
    def on_unload(self):
        # Unload config, reset everything to default
    def on_launch(self):
        # Ramp to safe voltage before anything else
        # ramp to V_Set
        # Start publishing the current
    def on_land(self):
        # Stop current-publishing thread
        # ramp to safe
```

```
    def on_start(self):
        # Do nothing special, just keep on logging
    def on_stop(self):
        # Keep logging
    def do_run(self):
        # Doing nothing special here
    def on_failure(self):
        # Stop current-publishing thread
        # Ramp down
    def on_interrupt(self):
        # Stop current-publishing thread
        # Ramp down
    def on_recover(self):
        # Do nothing here
    def on_reset(self):
        # Remove device
```

Implementation

- Rather simple satellite;
  - Creating device and loading config into it in **on_load()**
  - Ramping voltage and starting logging thread (i.e. publishing “CURRENT” stats) in **on_launch()**
  - Ramping down and stopping logging thread in **on_land()**, **on_failure()**, **on_interrrupt()**
  - Disconnecting and removing device in **on_unload()** and **on_reset()**
- Seems to work nicely with logging and controller!
  - We’ve played with both receiving logs and stats, and controlling the Keithley over the network

Backup slides