Trajectory optimization on a hybrid computer

1 Introduction

The idea of coupling an analog computer with a digital computer, thus creating a so-called *hybrid computer*, is by no means new, but nevertheless intriguing as it makes it possible to combine the best of both worlds of computing. The first hybrid computers were built in the mid-1950s to aid in aerospace research.[1]

The following application note describes the solution of a simple shell-trajectory problem with an Analog Paradigm prototype hybrid computer. This computer is based on the Analog Paradigm Model-1 in which the control unit (CU) has been replaced by a so-called *Hybrid Controller*, *HyCon* for short. This module plugs into the analog computer's backplane and connects to a digital computer by means of an USB-interface. It allows complete control of the analog computer, automatic readout of all computing elements, control of digital potentiometers etc. This module is supported by means of a Perl module, `HyCon.pm`, that offers an object oriented interface to the hybrid controller.

[1]Cf. [Ulmann, 2013][pp. 151 ff.]

2 The shell trajectory problem

The problem under consideration is extremely simple: The trajectory of an idealized shell experiencing no drag or other external influences is to be simulated in order to determine the initial velocity v_0 necessary to hit a target at a given position $(x_{\text{target}}, y_{\text{target}})$. The elevation of the cannon is fixed at some angle α. The cannon is mounted at an elevation of y_0 while the target is at sea level elevation i. e. $y_{\text{target}} = 0$.

The x- and y-components of the velocity of the shell are thus

$$\begin{aligned}\dot{x} &= v_0 \sin(\alpha) \text{ and}\\ \dot{y} &= \cos(\alpha) - gt\end{aligned}$$

with g and t denoting the gravitational acceleration and time. These two variables readily yield the x- and y-components of the shell's position by integration.

3 Analog computer program

The resulting computer setup is quite straight-forward and shown in figure 1.

Here *DPT0* denotes the digital potentiometer number 0 under control of the hybrid controller. All potentiometers are buffered so that they can be connected in series. The outputs of this circuit are as follows:

x and y: Position of the shell – these two outputs can be used to control a display set to XY-mode.

Δx: This is the x distance of the shell and its target.

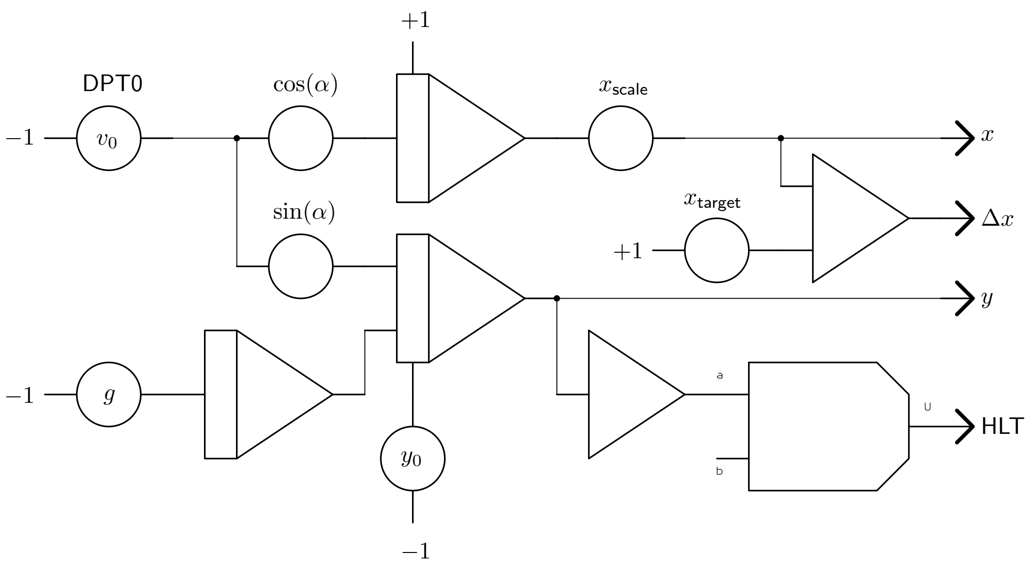

Figure 1: Setup of the analog computer for the basic trajectory problem

HLT: This logical output from a comparator is used to trigger the external halt input of the hybrid controller when the shell hits ground level. Therefore it is necessary that the height of the cannon satisfies $y_0 > 0$. Otherwise the comparator would trigger halt before the actual flight of the shell begins.

The patched analog computer (a prototype as can be easily seen) is shown in 2.

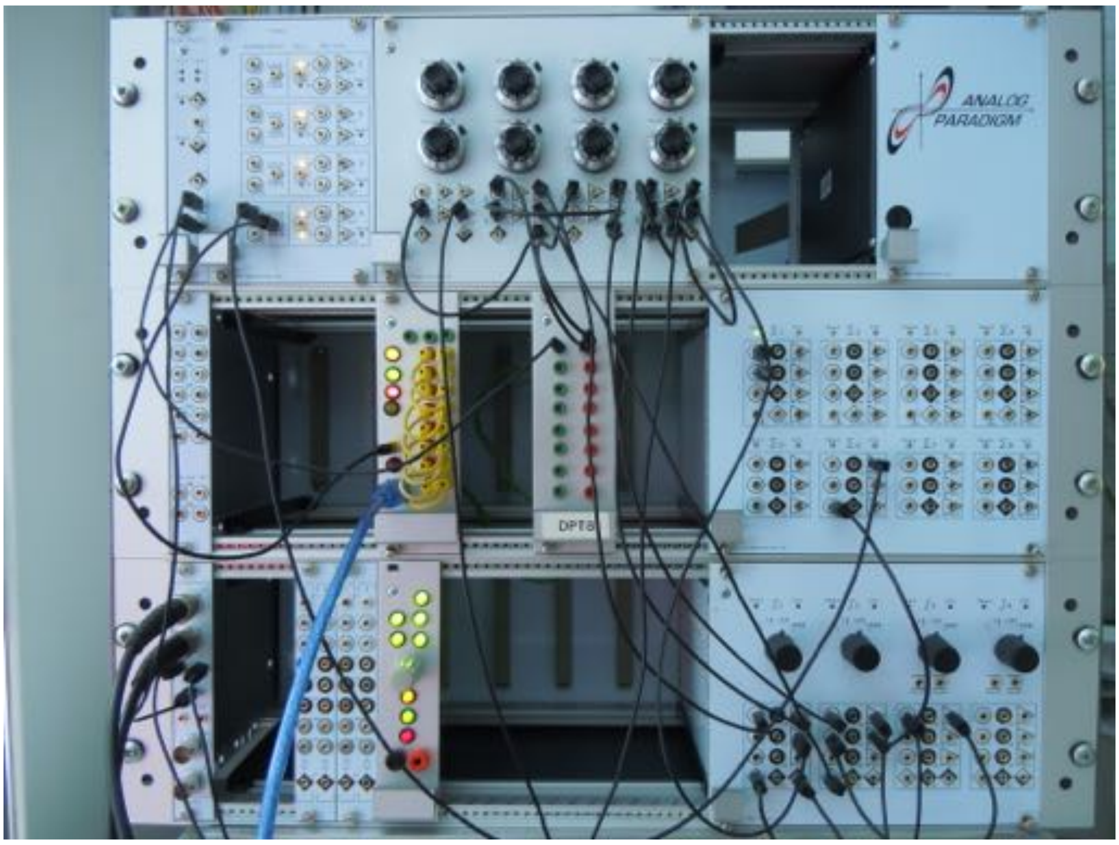

Figure 2: Setup of the analog computer program on a prototype hybrid computer

4 Digital computer program

The digital part of the hybrid computer is programmed in Perl using the `HyCon.pm` module developed by Analog Paradigm. This module requires a configuration YML-file as shown below in which the particular setup of the analog computer being used is specified. The sections shown are mandatory and specify the USB (serial) connection, the number of digital potentiometers and their resolution in bits, the calibration data required for the analog-digital-converter (factory supplied data), the various device types of the analog computing elements, and finally the names of computing elements which are to be readout by the hybrid controller.

trajectory.pl

```
serial:
    port: /dev/cu.usbmodem431
    bits: 8
    baud: 250000
    parity: none
    stopbits: 1
    poll_interval: 1000
    poll_attempts: 200
potentiometers:
    number: 8
    resolution: 10
calibration:
    analog_in:
        - 13202, 4271, 22133
        - 13208, 4278, 22137
        - 13204, 4271, 22137
    readout: 13195, 4271, 22117
types:
    0: PS
    1: SUM8
    2: INT4
    3: PT8
    4: CU
    5: MLT8
    6: MDS2
    7: CMP4
elements:
    SUM8-0: 0x0160
```

trajectory.pl

The overall control program is shown in the following listing. After using all of the required libraries, the terminal window is configured and cleared. Then a new `HyCon`-object is created based on the configuration data supplied in the corresponding YML-file. Next, the hybrid controller is reset and halt-on-overflow is disabled while the external halt input is enabled. This input is connected to the HLT-output of the comparator shown in figure 1. The initial-condition- and operation-times are set to 20 and 30 milliseconds. In addition to this the initial velocity v_0 is set to zero.

Then follows the central loop in which the analog computer performs a single simulation run over and over again until either the operation-time is exceeded or the external halt condition is met. After one of these simulation runs, the output of summer `SUM-0`, corresponding to Δx, is read out.

Based on this value, a new v_0 is computed by setting

$$v_0 := v_0 + \Delta x \cdot c$$

where c (`$conv` in the program source) controls the convergence speed of the optimization process. The current setting is clearly suboptimal but results in a nice output on an XY-display where the optimization process can be seen quite well. A more realistic setup would, of course, involve a different update scheme for v_0 as the linear approach shown above. A screen shot of a solution obtained by this setup is shown in figure 3.

trajectory.pl
```perl
use strict;
use warnings;

use lib '../..';                                        # HyCon lib path
use File::Basename;
use Term::ANSIScreen;
use HyCon;

$| = 1;                                                 # No stdout buff.
my $console = Term::ANSIScreen->new();                  # We need CLS
$console->Cls();

(my $config_filename = basename($0)) =~ s/\.pl$//;
my $ac = HyCon->new("$config_filename.yml");            # New HyCon-object

$ac->reset();
$ac->disable_ovl_halt();                                # No overflow halt
$ac->enable_ext_halt();                                 # but ext. halt

my ($ic_time, $op_time) = (20, 30);                     # Times in ms
$ac->set_ic_time($ic_time);
$ac->set_op_time($op_time);

my ($v_0, $conv) = (0, .1);                             # v_0, conv. speed
while (1) {
    my $halt = $ac->single_run_sync();                  # Single run
    my $delta = $ac->read_element('SUM8-0')->{value};   # Read distance

    my $increment = $conv * $delta;                     # Comp. increment
    printf("V0 = %+0.4f\tDelta = %+0.4f\tIncrement: %+0.4f\r",
        $v_0, $delta, $increment);
    $v_0 += $increment;                                 # v_0 for shell
    $ac->set_pt(0, $v_0);                               # Set DPT-0
}
```
trajectory.pl

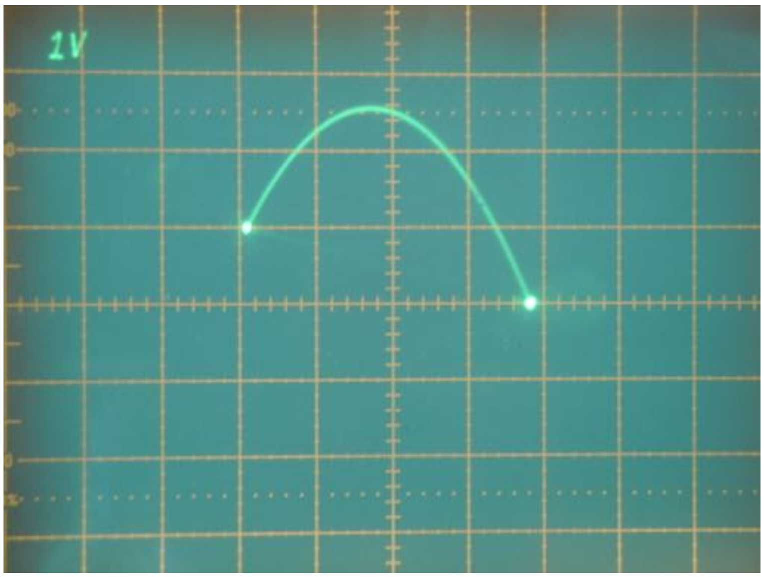

Figure 3: Typical solution of the simple shell-trajectory problem

References

[ULMANN, 2013] BERND ULMANN, *Analog Computing*, Oldenbourg Verlag, 2013