Communication Basics
Lap Report 4
PLL

Performed by:
Aidyn Ardabek
a.ardabek@jacobs-university.de

Instructor:
Mathias Bode
m.bode@jacobs-university.de

Date of lab: 21 January 2022

Pre-Lab Questions:

1. *What are the two functions of a PLL in a communications system?*

There are functions of a PLL in a communications system:

a. Carrier recovery – synchronizing the local and incoming signal's oscillator.
b. Symbol timing recovery – aligning the sample times at the matched filter output.

2. *What are the key components (operations) used to implement a PLL?*

First, local reference y(t) from the voltage-controlled oscillator (VCO) and input signal are multiplied, which results to a signal with two components: a low frequency component that depends on phase difference and a doubled frequency component. Then, a low pass filter removes the doubled frequency. VCO received the remaining component and produces another signal that can change y(t) accordingly to x(t). Finally, when phases reach proper alignment, VCO produces a signal that is close to 0, leaving only desired signal.

3. *When we write the second-order frequency response of the PLL, this expression relates the input and output* phase *of the references.*

4. *What does the corner frequency of the PLL loop filter control?*

The corner frequency controls how quickly the loop adjusts to phase changes. We want this frequency to be less than the frequency of the sinusoid or clock being tracked (<0.1).

5. *How should the loop filter corner frequency compare to the input reference frequency?*

The corner frequency has to be smaller than the input reference frequency.

6. *What does it mean for a PLL to track?*

Tracking for PLL means that input and output phases are linearly related by the transfer function (response of the loop filter).

7. *What does it mean for the PLL for loop to be overdamped or underdamped?*

Overdamped means that the system requires too much of time to adjust the phase changes. Underdamped means that the system responses too fast and tends to overshoot the target.

8. *Why is an accumulator useful for a PLL implementation?*

Accumulator is a single real number that helps us to keep track of the current phase, instead of hacing two variables for time and phase. Therefore, the accumulator tells us the current position in the sin() lookup table.

9. *Why do we need to sample sin() in our lookup table more finely than at the normal sample rate of the system?*

Usually, lookup table computes samples spaced by the sample period. However, for the PLL we need samples that are between sample periods. Also, the phase can change after some time to track the input.

10. *How do we keep our accumulator in the range [0, 1]?*

To keep the accumulator in the range [0, 1], we make the sinusoidal period 1.0. So, the accumulator needs to wrap when its value exceeds 1 and drops below 0.

11. What is the point of storing and restoring the state of the PLL for each block?

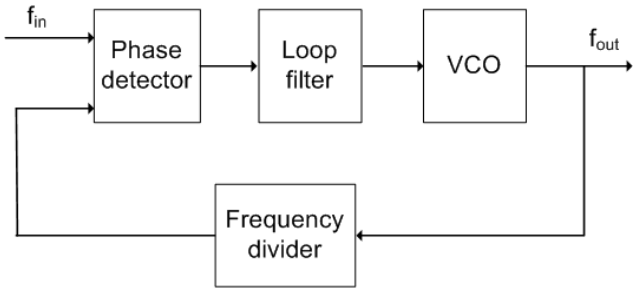
The previous samples are used by the PLL to keep track of the accumulator.

12. How do we handle signals with arbitrary amplitude?

In the PLL, every block of samples is scaled to have approximately unit amplitude. Since amplitude cannot change dramatically from one block of samples to another, amplitude of all samples will be around 1. The average amplitude of each block of samples is used to scale the next block of samples.

Lab Write Up:

- *A paper design of your PLL, showing the important parameters that are needed for implementation.*



- *A printout of your final MATLAB implementation of the PLL.*

```
function [s] = plliniti(f, D, k, w0,
                            T, table_size)

% Parameters
s.f = f;            % Nominal ref. frequency
s.D = D;            % Damping factor
s.k = k;            % Loop gain
s.w0 = w0;          % Loop corner frequency
s.T = T;            % Period

% Look up table
for i = 0:table_size-1
    s.cos_table(i+1) = cos(2*pi*i/table_size);
end

% Filter coeficients
s.tau1 = k/w0^2;
s.tau2 = 2*D/w0 - 1/k;
s.a1 = -(s.T - 2*s.tau1) / (s.T + 2*s.tau1);
s.b0 = (s.T + 2*s.tau2) / (s.T + 2*s.tau1);
s.b1 = (s.T - 2*s.tau2) / (s.T + 2*s.tau1);

% Create state variables
s.out_old = 0.0;
s.z_old = 0.0;
s.v_old = 0.0;
s.accum = 0.0;
s.amp_est = 1;
```

```
function [out, state_out] = pll(in, N,
                        state_in, table_size)

s = state_in;
out = zeros(size(in));
amp = 0;

for i = 1:N
    % Compute phase difference
    amp = amp + abs(in(i));
    z = in(i)/s.amp_est*s.out_old;
    v = s.a1*s.v_old + s.b0*z + s.b1*s.z_old;

    s.accum = s.accum + s.f - (s.k/(2*pi))*v;
    s.accum = s.accum - floor(s.accum);

    % Use cos_table to calculate output
    out(i) = s.cos_table(floor(table_size
                            *s.accum) + 1);

    % Update state variables
    s.out_old = out(i);
    s.z_old = z;
    s.v_old = v;
end

% Take the average of the amplitude
s.amp_est = amp/N/(2/pi);
state_out = s;
```

```
table_size = 1024;
[s] = plliniti(0.1, 1, 1, 2*pi/100,
                        1, table_size);

Ns = 100;        % Number of samples
Nb = 10;         % Number of blocks

load('ref_stepf');
in = reshape(ref_in, Ns, Nb);

out = zeros(Ns, Nb);
y_input = ref_in;

% Changing the amplitude of the input
for i = 1:1000
    y_input(i) = 2*y_input(i);
    in(i) = 2*in(i);
end

for n = 1:Nb
    [out(:, n), s] = pll(in(:, n),
                         Ns, s, table_size);
end

y_output = reshape(out, Ns*Nb, 1);

plot(1:length(y_input), y_input);
hold on;
plot(1:length(y_output), y_output);
ylim([-2, 2])
```

- A plot showing the simulated performance of the final PLL, demonstrating that the PLL can adapt to abrupt changes in frequency/phase.

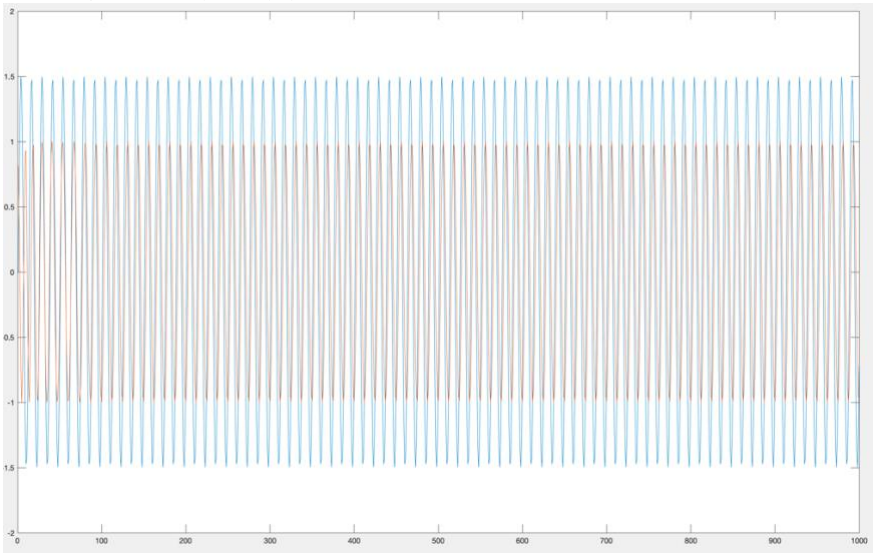


Figure 1: ref_800hz

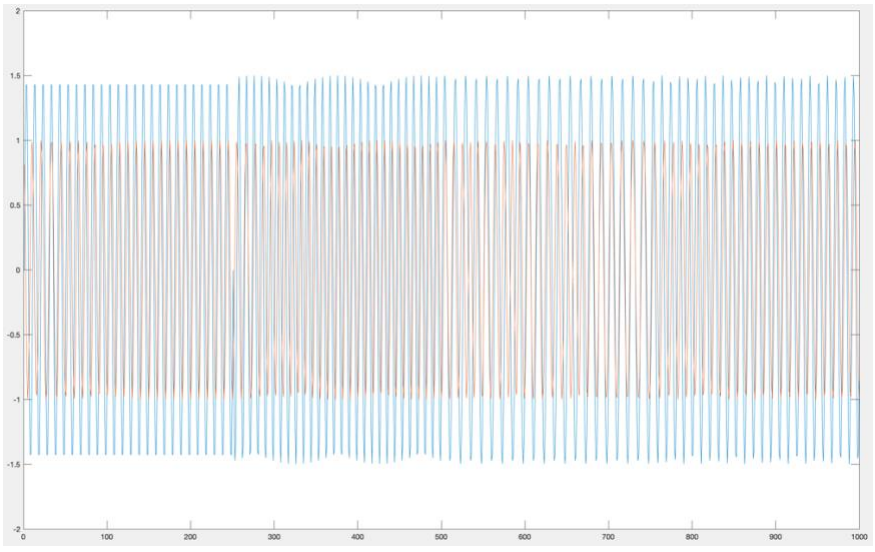


Figure 2: ref_stepf