ASEN 5090: Intro to GNSS
Assignment 10

Ian Thomas (107554041)

12/6/2021

Abstract

This study addresses the effectiveness of tracking on improving ranging accuracy and autocorrelation signal-to-noise ratio (SNR) on simulated signals when compared to coarse acquisition. We design a tracking loop that takes delays and dopplers from coarse acquisition and applies a delay locked loop (DLL) to the code delay and a phase locked loop (PLL) to handle carrier phase and frequency alignment. We find that a well designed tracking loop can improve pseudorange measurements to well within a chip and can improve the autocorrelation SNR for noisy but acquirable signals due to aligning code phase, carrier phase, and frequency errors arising from doppler and clock imperfections on real signals. This further solidifies the need for well tuned tracking algorithms which enable GNSS pseudorange measurements to be as precise as possible.

Contents

1 Objective

Obtaining accurate and precise pseudorange measurements is fundamental to obtaining usable results in GNSS applications, ranging from position solutions to GNSS reflectometry. Pseudorandom codes are modulated onto carrier signals transmitted from moving satellites, so a combination of code delays, carrier phase misalignments, and frequency errors due to doppler and clock errors accumulate in a search problem to find the right signal parameters so a pseudorange can be obtained. In coarse acquisition, replica signals are generated at code delays at the sampled frequency as well as different doppler bins as a function of integration time, but even still, unless the signal is sampled at an extremely high rate, the sample resolution is often not high enough to generate pseudoranges to within meter or 10-meter accuracy (for example, sampling at 5 MHz leads to a sample accuracy of 60 meters).

A tracking loop attempts to improve on this initial sample-resolution estimate of the code phase and integration time resolution estimate of frequency error, and introduces carrier phase as an additional signal parameter to correct for [1]. This improvement in estimation of the code phase and a good estimate of the carrier phase, is what enables GNSS systems to operate to meter if not better accuracy under the right conditions.

2 Approach

We first break up the data into blocks based on integer code period integration times. We used a closed loop approach to determine the code delay and carrier frequency via an early minus late discriminator. Since the theoretical autocorrelation peak around the true delay forms a triangle, a prompt correlator should result in a maximum value with early and late correlators returning similar values. If the early correlator is high and the late is low, we need to advance the replica and vice versa, and if the late correlator is high and the early is low, we need to delay the replica. The amount we advance or delay the replica is proportional to the difference between the early and late correlators, and that is how we get our code delay correction [1].

We can determine the carrier phase by taking the real and imaginary parts of the prompt correlator and using arctangent to determine a phase correction (if the carrier phase is correct, the real in-phase part of the signal will have near 100% of the signal power and the imaginary quadrature part will have near zero power). The amount we advance or delay our carrier phase is again proportional to the result we get from arctangent [1]. For this tracking loop, we accumulated the $\Delta\theta$ in a variable and performed integral control to bring the carrier phase error back to zero.

If we have a frequency error, we should see a constant phase difference between subsequent correlations. We can use this phase difference to design a discriminator, for which we can adjust frequency proportional to the value of the discriminator [2].

We can simulate signals with real code delays, carrier phases, and frequency offsets and compare the tracked signal parameters with their truth values.

3 Results

3.1 No Noise Environment

We first simulated results without noise for PRN 31, a simulated doppler of 434 Hz, and a simulated code phase of 434.2815 chips. The tracking loop blocks had integration times of 2 code periods, or 2 ms. The following shows the results of the acquisition and tracking process:

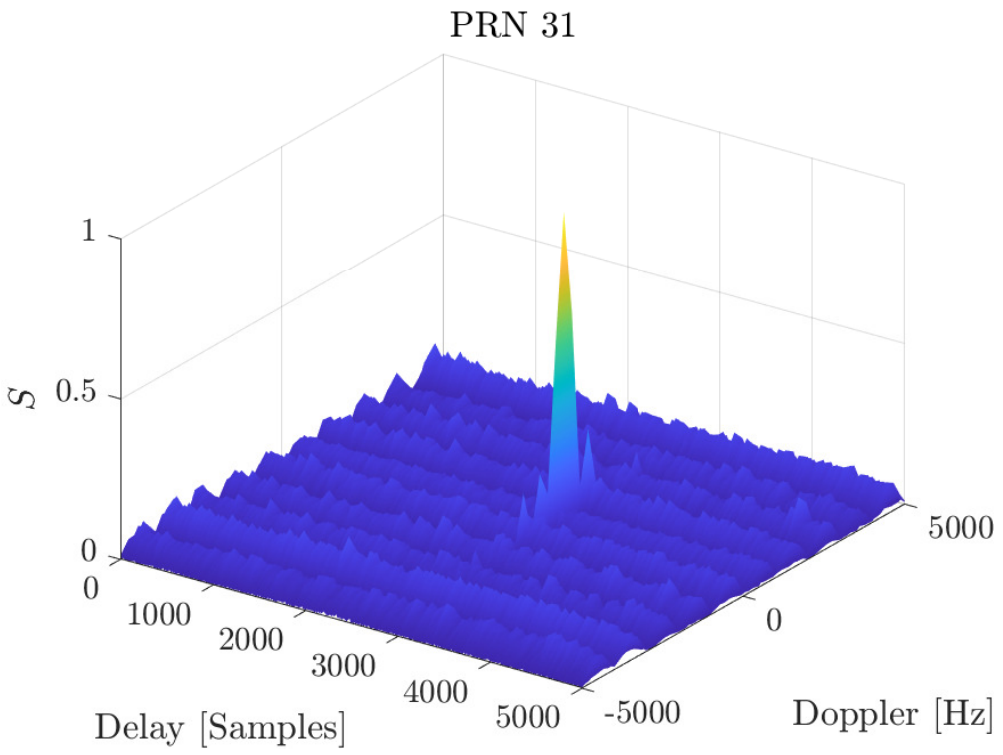
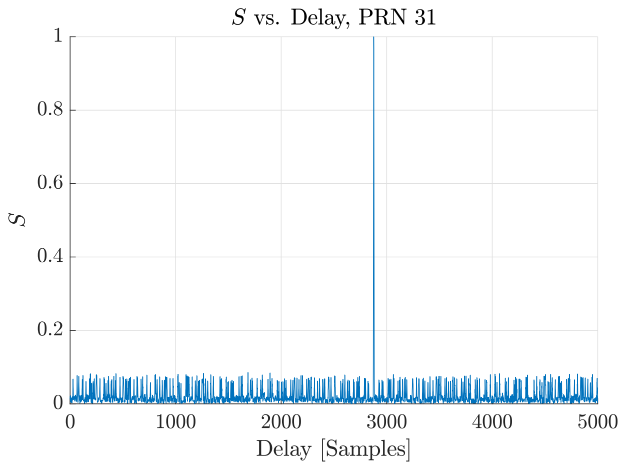
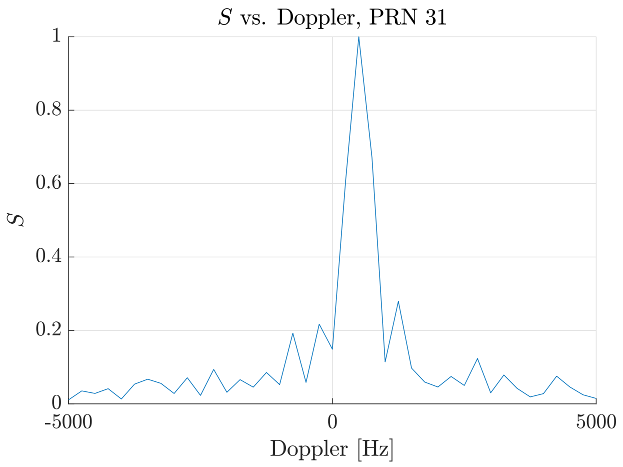


Figure 1: No Noise Acquisition of PRN31



(a) Delay across best doppler axis



(b) Doppler across best delay axis

Figure 2: Acquisition results for PRN31

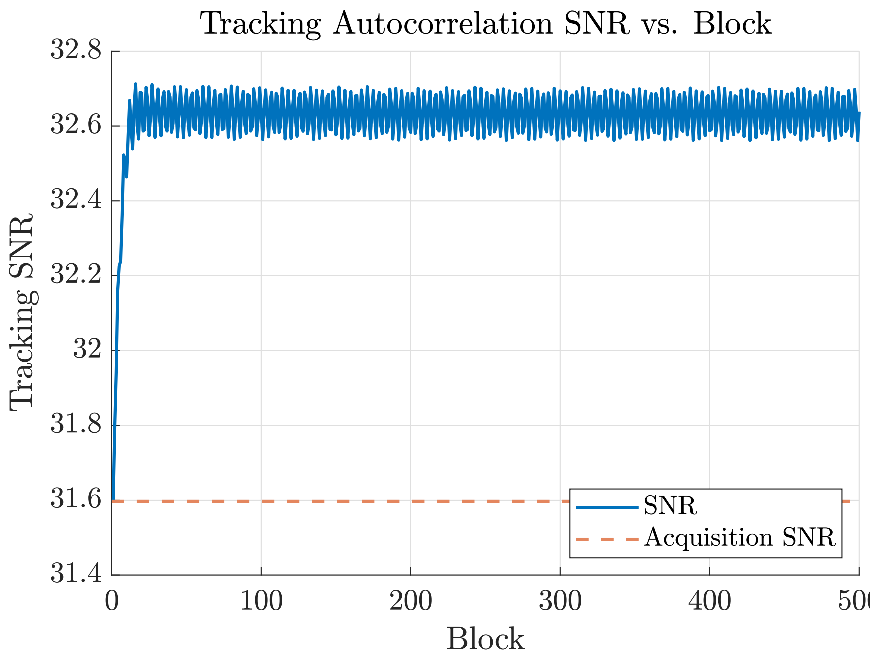
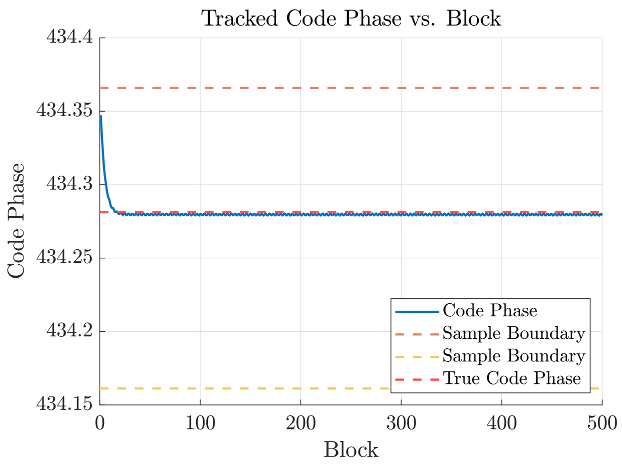
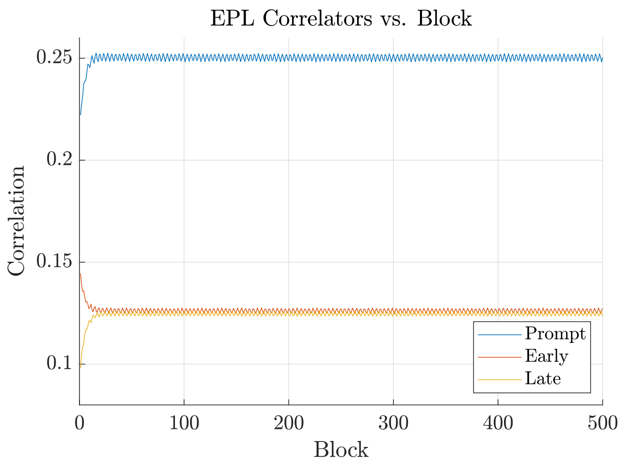


Figure 3: Tracking SNR

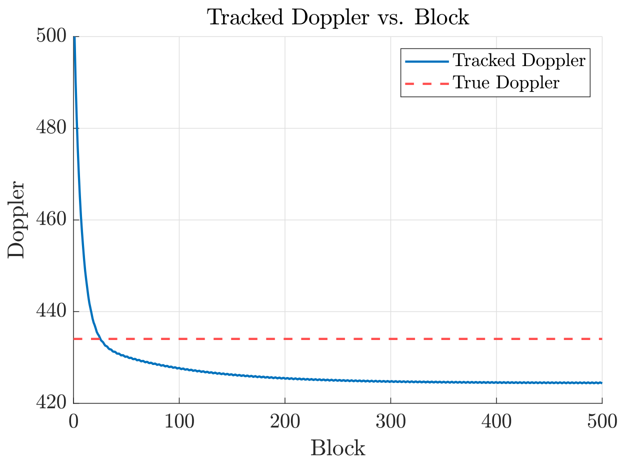


(a) Tracked Code Phase vs. Block

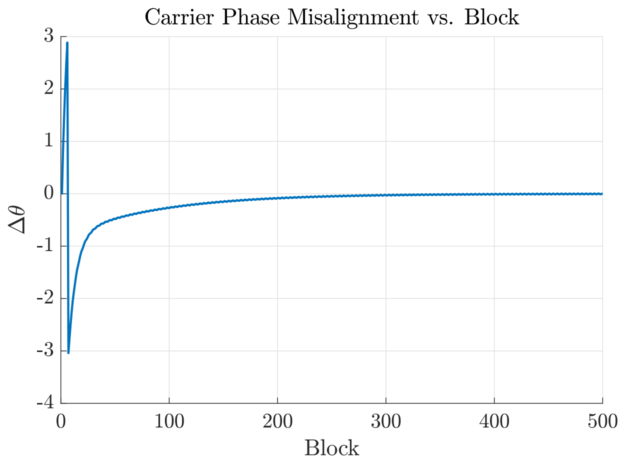


(b) EPL Correlations vs. Block

Figure 4: Tracked Code Phase and EPL Correlations



(a) Tracked Doppler vs. Block



(b) Carrier Phase vs. Block

Figure 5: Doppler and Carrier Phase

3.2 Noisy Signals

We then ran the simulation again with the same simulated doppler and code phase, but with a C/N_0 of 60 dB-Hz and an integration time of 1 ms:

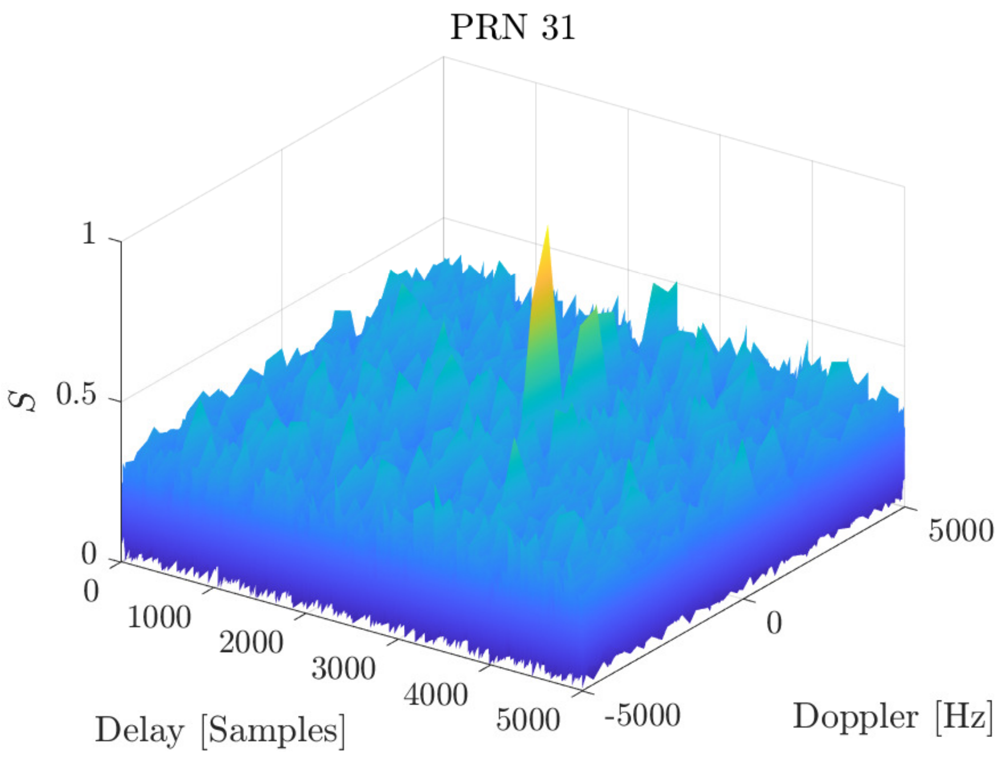
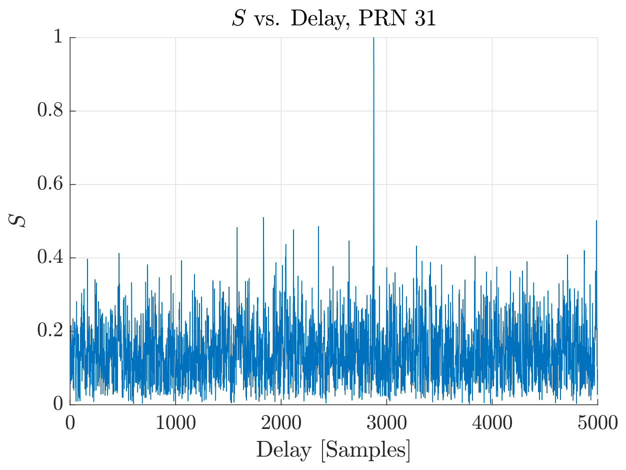
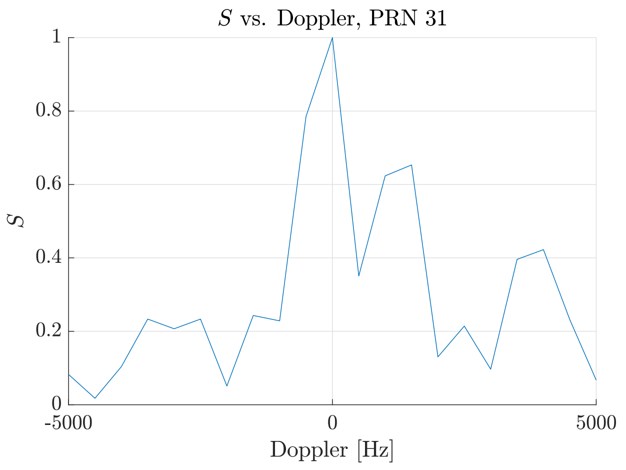


Figure 6: No Noise Acquisition of PRN31



(a) Delay across best doppler axis



(b) Doppler across best delay axis

Figure 7: Acquisition results for PRN31

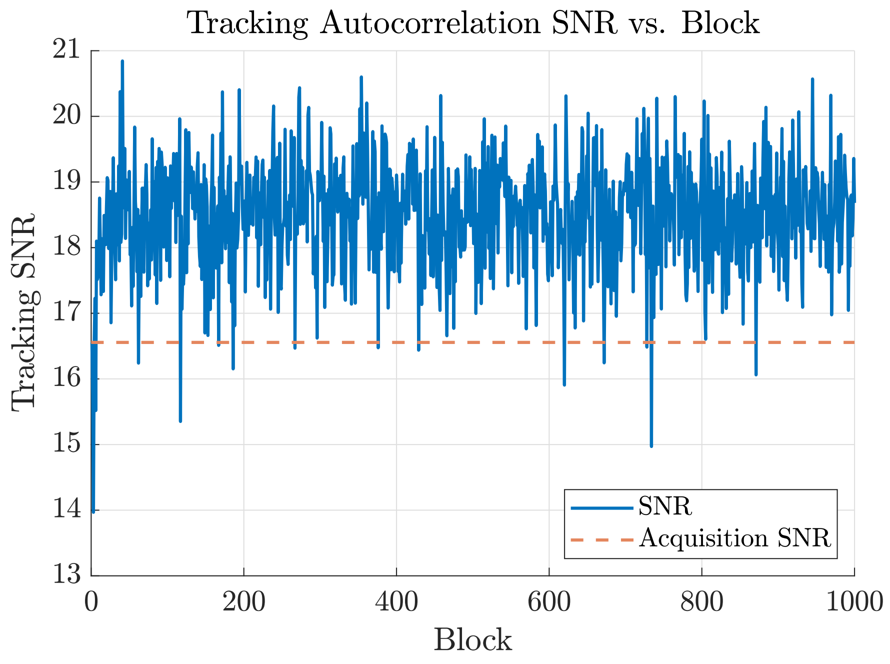
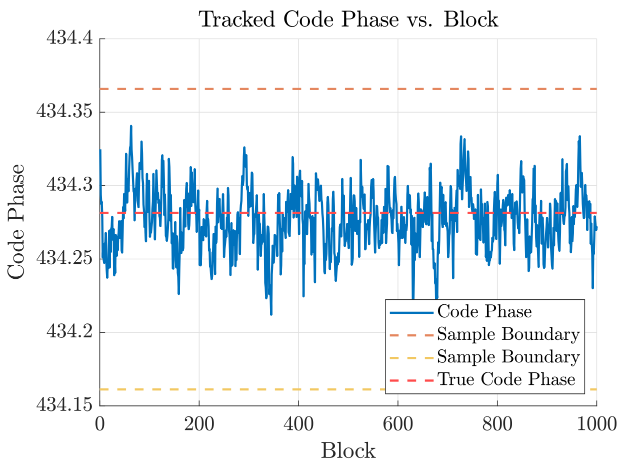
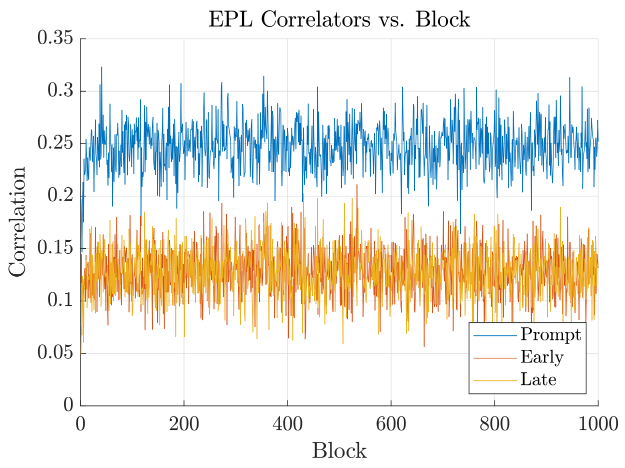


Figure 8: Tracking SNR

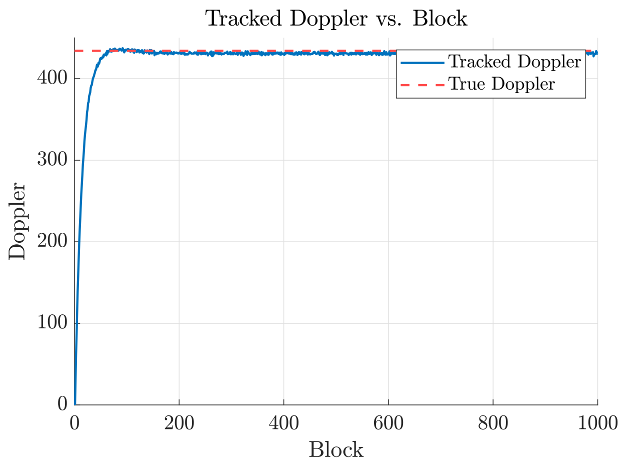


(a) Tracked Code Phase vs. Block

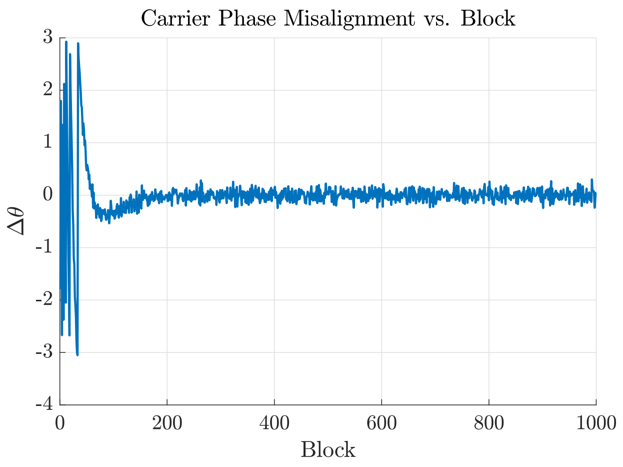


(b) EPL Correlations vs. Block

Figure 9: Tracked Code Phase and EPL Correlations



(a) Tracked Doppler vs. Block



(b) Carrier Phase vs. Block

Figure 10: Doppler and Carrier Phase

4 Discussion

We looked at the tracking loop under two circumstances: a no noise environment with an integration time of 2 ms, and a 60 dB-Hz noise environment with an integration time of 1 ms. In both cases, the tracking loop was able to find a more accurate code phase (figures 4a and 9a) and doppler (figures 5a and 10a) than just acquisition alone. This means, in both cases, the autocorrelation SNR from tracking should outperform the autocorrelation SNR that we see from acquisition, which we see in figures 3 and 8.

It is easier to see in figure 4b than in figure 9b, but the misaligned codes caused differences between the early and late correlators, so the tracking loop was able to align the code phase well. We also note on the code phase plot the two sample boundary lines, and we can see that the tracked code phases fall within sample boundaries and appear unbiased about the actual code phase.

The carrier phase plots in both figures 5b and 10b exhibit the integral control behavior we defined in the tracking loop, and we can see that they stay constant after a while.

The figure with the worst performance was the tracked doppler in figures 5 and 10, with the noisy environment actually outperforming the no-noise environment (but both still significantly outperformed acquisition). Even though the doppler is misaligned in figure 5, the carrier phase stayed constant in figure 5b, which doesn't make sense as a misaligned doppler should result in different carrier phase values for each block.

While this analysis demonstrated a lot of the usefulness of tracking algorithms for simulated signals, there are a number of environmental sources of error not considered for the simulation, such as multipath, atmospheric effects, ionospheric effects, relativistic effects, changing dynamics (we assumed a constant doppler), or other sources of interference. We also didn't consider the signals under a higher noise level, for which the tracking performance would significantly degrade, although we picked a noise level comparable to the noise environment seen in HW9 for real GPS data.

5 Conclusion

This example of a tracking loop was a first iteration of a simple tracking loop which applied integral control to the carrier phase, a DLL for the code phase, and an FLL to solve for frequency errors. While the tracking performance wasn't converging for one of the two doppler cases, the algorithm managed to estimate the signal parameters far better than acquisition alone, which affirms the need for tracking algorithms in GNSS applications.

This analysis isn't complete without results using real GNSS data, which models all environmental error sources that we didn't consider, and using those results to calculate actual pseudoranges and position solutions. Some further work is required to explore the discrepancy between the doppler not converging but the carrier phase alignment converging to zero.

References

[1] Axelrad, Penina. "ASEN 5090 L35 Tracking." University of Colorado, Boulder, 2021.

[2] "Frequency Lock Loop (FLL)." Navipedia, European Space Agency, https://gssc.esa.int/navipedia/index.php/Frequency_Lock_Loop_(FLL).

MATLAB$^{\circledR}$ Code

Main Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ian Thomas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
clear;
clc;

set(groot, 'defaulttextinterpreter', 'latex');
set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
set(groot, 'defaultLegendInterpreter', 'latex');

set(0, 'DefaultAxesLooseInset', [0,0,0,0])
set(0,'defaultAxesFontSize',14)

colors = get(gca, 'colororder');
close all;

addpath('../matlab');
addpath('data/');

rng(100);

%% Declaration of Parameters

Re = 6378.137e3; % semimajor axis of earth ellipsoid
f = 1/298.257223563; % flattening parameter of ellipsoid
e2 = 2*f - f^2; % eccentricity squared of ellipsoid
tol = 1e-10;
c = 299792458;

integration_periods = 1;

fs = 5e6;
ts = 1/fs;
fc = 1.023e6;
L1 = 1575.42e6;
nc = 1023;
T = nc/fc * integration_periods;
nt = T * fs;
nsp = nt / integration_periods;
```

```
t = (0:(nt-1))' / fs;

fIF = 1.25e6;
fdmin = -5e3;
fdmax = 5e3;
fdspacing = 1/(2*T);
fd = fdmin:fdspacing:fdmax;

phi_t = @(t, fIF, fd) 2*pi*(fIF+fd).*t;

%% Simulated Data

prn = 31;
code = gen_prn_SV(prn);

simulated_CN0 = 60; % dB-Hz
simulated_SNR = 10^(simulated_CN0/10) / fs;
simulated_AWGN_sigma = sqrt(1/simulated_SNR);

simulated_doppler = (rand() * 2 - 1) * fdmax;
% simulated_doppler = 250;
simulated_carrier_phase = pi/2 * rand();
% simulated_carrier_phase = 0;
simulated_code_phase = 1023 * rand();
% simulated_code_phase = 500.5;

n_simulated_periods = 1000;
simulated_t = (0:(n_simulated_periods*nsp - 1)) / fs;

Sc = resample_digital(simulated_t, code, fc , simulated_code_phase);
Scarr = cos(phi_t(simulated_t, fIF, simulated_doppler)+...
    simulated_carrier_phase);

S = Sc .* Scarr + simulated_AWGN_sigma*randn(size(Sc));

gpsdata = S';


%% Acquisition

first = gpsdata(1:nt);
second = gpsdata(nt+1:2*nt);
third = gpsdata(2*nt+1:3*nt);

prns = 31;
```

```
SNR_acq = zeros(size(prns));
delays = zeros(size(prns));
dopplers = zeros(size(prns));
delay_inds = zeros(size(prns));
doppler_inds = zeros(size(prns));
noise_floors = zeros(size(prns));

% run coarse acquisition on all the prns
for i = 1:length(prns)
    code = gen_prn_SV(prns(i)) * 2 - 1;
    S_replica = resample_digital(t, code, fc, 0)';
    [lags, corrs] = cxcorr(first,...
        S_replica.*exp(1i*phi_t(t, fIF, fd)), nt);
    lags = lags(1:(nt/integration_periods));
    corrs = corrs(1:(nt/integration_periods), :);
    abscorrs = abs(corrs) / nt;
    [maxcorrs, maxidx] = max(abscorrs(:));
    maxcol = floor(maxidx / (nt/integration_periods)) + 1;
    maxrow = mod(maxidx - 1, (nt/integration_periods)) + 1;
    delays(i) = lags(maxrow);
    dopplers(i) = fd(maxcol);
    delay_inds(i) = maxrow;
    doppler_inds(i) = maxcol;
    SNR_acq(i) = 20*log10(max(abscorrs(:)) / mean(abscorrs(:)));
    noise_floors(i) = mean(abscorrs(:));

    figure(3*i-2)
    hold on;
    grid on;
    xlabel('Delay [Samples]')
    ylabel('Doppler [Hz]')
    zlabel('$S$')
    title(sprintf("PRN %d", prns(i)))
    [X, Y] = meshgrid(lags, fd);
    surface(X, Y, abscorrs'/max(abscorrs(:)), 'edgecolor', 'none')
    shading interp;
    view(35, 35);
    saveas(gcf, sprintf("figures/prn%d_acq_i%d_CN0%d", prns(i),...
        integration_periods, simulated_CN0), 'epsc')

    figure(3*i-1)
    hold on;
    grid on;
    xlabel('Delay [Samples]')
    ylabel('$S$')
    title(sprintf('$S$ vs. Delay, PRN %d', prns(i)));
    plot(lags, abscorrs(:, maxcol) / maxcorrs);
    saveas(gcf, sprintf("figures/prn%d_del_i%d_CN0%d", prns(i),...
        integration_periods, simulated_CN0), 'epsc')

    figure(3*i)
    hold on;
    grid on;
    xlabel('Doppler [Hz]')
    ylabel('$S$')
    title(sprintf('$S$ vs. Doppler, PRN %d', prns(i)));
    plot(fd, abscorrs(maxrow, :) / maxcorrs);
    saveas(gcf, sprintf("figures/prn%d_dop_i%d_CN0%d", prns(i),...
        integration_periods, simulated_CN0), 'epsc')
end

% use an acquisition SNR of 15 dB as the cutoff
cutoff = SNR_acq >= 15;
SNR_acq = SNR_acq(cutoff);
prns = prns(cutoff);
delays = delays(cutoff);
dopplers = dopplers(cutoff);
delay_inds = delay_inds(cutoff);
doppler_inds = doppler_inds(cutoff);
noise_floors = noise_floors(cutoff)';


%% Tracking

ns = length(gpsdata);
nb = floor(ns / nt);
epl_delay = 0.5; % chip difference between early, prompt, and late correlators

B_DLL = 0.2; % tracking loop bandwidths
B_PLL = 0.01/(2*pi);
B_FLL = 12;

gpsdata = gpsdata(1:(nb*nt));
gpsblocks = reshape(gpsdata, [nt, nb]);

prn_code_phases = zeros(length(prns), nb);
prn_carrier_phases = zeros(length(prns), nb);
prn_doppler_freqs = zeros(length(prns), nb);
prn_ecorrs = zeros(length(prns), nb);
prn_pcorrs = zeros(length(prns), nb);
```

```
prn_lcorrs = zeros(length(prns), nb);
prn_delta_thetas = zeros(length(prns), nb);
prn_delta_f = zeros(length(prns), nb);

for i = 1:length(prns)
    prn = prns(i);
    code_phase = nc - delays(i) / nsp * nc;
    carrier_phase = 0;
    doppler_freq = dopplers(i);

    code_phases = zeros(size(1:nb));
    carrier_phases = zeros(size(1:nb));
    doppler_freqs = zeros(size(1:nb));
    ecorrs = zeros(size(1:nb));
    pcorrs = zeros(size(1:nb));
    lcorrs = zeros(size(1:nb));
    delta_thetas = zeros(size(1:nb));
    delta_f = zeros(size(1:nb));

    IP1 = 0;
    QP1 = 0;

    IC_delta_theta = 0;

    for j = 1:nb

        block = gpsblocks(:, j);
        block = block - mean(block);

        code = gen_prn_SV(prn);

        carrier_replica = exp(-1i*(phi_t(t, fIF, doppler_freq)...
            + carrier_phase));

        % early
        ecode = resample_digital(t, code, fc + (1 + doppler_freq/L1),...
            -epl_delay + code_phase) * 2 - 1;
        Se = ecode' .* carrier_replica;
        ecorr = sum(block .* conj(Se)) / nt;
        ecorrs(j) = ecorr;

        % prompt
        pcode = resample_digital(t, code, fc + (1 + doppler_freq/L1),...
            code_phase) * 2 - 1;
        Sp = pcode' .* carrier_replica;
        pcorr = sum(block .* conj(Sp)) / nt;
        pcorrs(j) = pcorr;

        % late
        lcode = resample_digital(t, code, fc + (1 + doppler_freq/L1),...
            epl_delay + code_phase) * 2 - 1;
        Sl = lcode' .* carrier_replica;
        lcorr = sum(block .* conj(Sl)) / nt;
        lcorrs(j) = lcorr;

        % DLL
        dll_discriminator = -epl_delay*(abs(ecorr) - abs(lcorr)) / ...
            (abs(ecorr) + abs(lcorr));
        code_phase_error = B_DLL * dll_discriminator;
        code_phase = code_phase + code_phase_error;

        % PLL
        pll_discriminator = atan2(imag(pcorr), real(pcorr));
        delta_thetas(j) = pll_discriminator;
        IC_delta_theta = IC_delta_theta + pll_discriminator;
        carrier_phase_error = -B_PLL * IC_delta_theta;
        carrier_phase = carrier_phase + carrier_phase_error;

        % FLL
        IP2 = real(pcorr);
        QP2 = imag(pcorr);

        fll_cross = IP1.*QP2 - IP2.*QP1;
        fll_dot = IP1.*IP2 + QP1.*QP2;
        fll_discriminator = mean(atan2(fll_cross, fll_dot));

        IP1 = IP2;
        QP1 = QP2;

        doppler_freq = doppler_freq - B_FLL * fll_discriminator;
        doppler_freqs(j) = doppler_freq;
        delta_f(j) = fll_discriminator;

        f_code_adj = fc * (1+ doppler_freq/L1);
        code_phase = code_phase + f_code_adj*T;
        code_phases(j) = mod(code_phase, nc);
        carrier_phase = carrier_phase + 2*pi*(fIF + doppler_freq)*T;
        carrier_phases(j) = mod(carrier_phase, 2*pi);
        carrier_phases(j) = carrier_phase;
    end
```

```
    prn_code_phases(i, :) = code_phases;
    prn_carrier_phases(i, :) = carrier_phases;
    prn_doppler_freqs(i, :) = doppler_freqs;
    prn_ecorrs(i, :) = ecorrs;
    prn_pcorrs(i, :) = pcorrs;
    prn_lcorrs(i, :) = lcorrs;
    prn_delta_thetas(i, :) = delta_thetas;
    prn_delta_f(i, :) = delta_f;

end


%% Compare new code phase correlations to old ones

figure(10)
hold on;
grid on;
xlabel('Block')
ylabel('Correlation')
title('EPL Correlators vs. Block')
plot(1:nb, abs(pcorrs));
plot(1:nb, abs(ecorrs));
plot(1:nb, abs(lcorrs));
legend('Prompt', 'Early', 'Late', 'location', 'se')
saveas(gcf, sprintf("figures/epl_i%d_CN0%d", integration_periods,...
    simulated_CN0), 'epsc')

figure(11)
hold on;
grid on;
xlabel('Block')
ylabel('$\Delta\theta$')
title('Carrier Phase Misalignment vs. Block')
plot(1:nb, prn_delta_thetas, 'LineWidth', 1.5)
saveas(gcf, sprintf("figures/dtheta_i%d_CN0%d", integration_periods,...
    simulated_CN0), 'epsc')

figure(12)
hold on;
grid on;
xlabel('Block')
ylabel('Tracking SNR')
title('Tracking Autocorrelation SNR vs. Block')
SNR_trk = 20*log10(abs(prn_pcorrs)./noise_floors)';
```

```
plot(1:nb, SNR_trk, 'LineWidth', 1.5)
yline(SNR_acq, '--', 'LineWidth', 1.5, 'Color', colors(2, :))
legend('SNR', 'Acquisition SNR', 'location', 'se')
saveas(gcf, sprintf("figures/SNR_i%d_CN0%d", integration_periods,...
    simulated_CN0), 'epsc')

figure(14)
hold on;
grid on;
xlabel('Block')
ylabel('Doppler')
title('Tracked Doppler vs. Block')
plot(1:nb, prn_doppler_freqs, 'LineWidth', 1.5)
yline(simulated_doppler, '--r', 'LineWidth', 1.5)
legend('Tracked Doppler', 'True Doppler')
saveas(gcf, sprintf("figures/doppler_i%d_CN0%d", integration_periods,...
    simulated_CN0), 'epsc')


figure(15)
hold on;
grid on;
xlabel('Block')
ylabel('Code Phase')
title('Tracked Code Phase vs. Block')
plot(1:nb, prn_code_phases, 'LineWidth', 1.5)
yline(nc - delays(i) / nsp * nc, '--', 'LineWidth', 1.5,...
    'Color', colors(2, :))
yline(nc - (delays(i)+1) / nsp * nc, '--', 'LineWidth', 1.5,...
    'Color', colors(3, :))
yline(simulated_code_phase, '--r', 'LineWidth', 1.5)
legend('Code Phase', 'Sample Boundary', 'Sample Boundary',...
    'True Code Phase', 'location', 'se')
saveas(gcf, sprintf("figures/code_phase_i%d_CN0%d",...
    integration_periods, simulated_CN0), 'epsc')
```

Resample Digital

```
function S = resample_digital(t, X, fx, phi)
    nc = length(X);
    delay = phi / nc;
    T = nc / fx;
    t = t + T*(delay+1);
    S = X(mod(floor(t*fx), nc) + 1);
end
```

Circular Cross Correlation

```
function [lags, c] = cxcorr(a, b, n)
%% Circularly correlate a and b across first dimension
    c = ifft(fft(a, n, 1).*conj(fft(b, n, 1)), n, 1);
    lags = 0:n-1;
end
```

Generate PRN for SV

```
function [c, cG1, cG2] = gen_prn_SV(sv)
%% Function gps_CA
%
% Purpose:
% To generate the course-acquisition code for a GPS signal using an LFSR
%
% Inputs:
% sv - integer from 1 - 32
%
% Output:
% c - the C/A code for the sv
%
% Author: Ian Thomas
% Last Modified: 9/4/2021

    svs = [
      2 6
      3 7
      4 8
      5 9
      1 9
      2 10
      1 8
      2 9
      3 10
      2 3
      3 4
      5 6
      6 7
      7 8
      8 9
      9 10
      1 4
      2 5
      3 6
      4 7
      5 8
      6 9
      1 3
      4 6
      5 7
      6 8
      7 9
      8 10
      1 6
      2 7
      3 8
      4 9];

    n = 10;
    phase_selector = svs(sv, :);
    taps1 = [3 10];
    taps2 = [2 3 6 8 9 10];
    G1 = cast(1023, 'uint16');
    G2 = cast(1023, 'uint16');
    c = zeros(1, 2^n-1);
    cG1 = zeros(1, 2^n-1);
    cG2 = zeros(1, 2^n-1);
    for i = 1:1023
        o1 = bitget(G1, n);
        cG1(i) = o1;
        o2 = bitget(sum(bitget(G2, phase_selector)), 1);
        cG2(i) = o2;
        c(i) = xor(o1, o2);
        i1 = bitget(sum(bitget(G1, taps1)), 1);
        i2 = bitget(sum(bitget(G2, taps2)), 1);
        G1 = bitset(bitshift(G1, 1), 1, i1);
        G2 = bitset(bitshift(G2, 1), 1, i2);
    end
    c = logical(c(1:i));
    cG1 = logical(cG1(1:i));
    cG2 = logical(cG2(1:i));
end
```