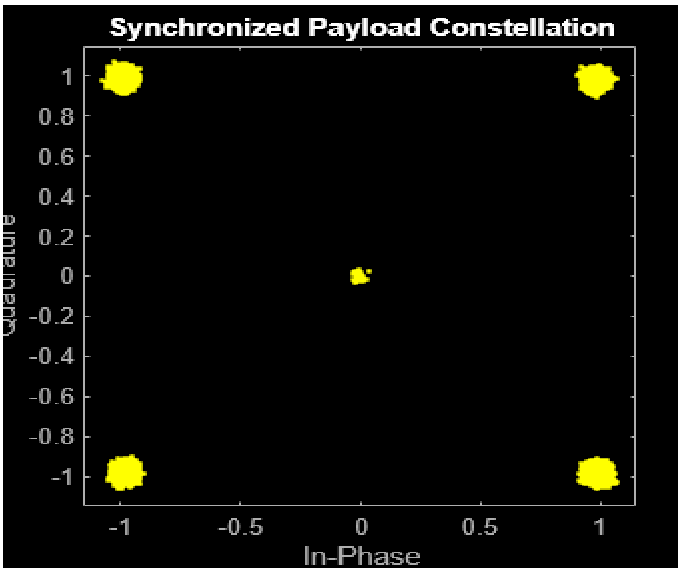
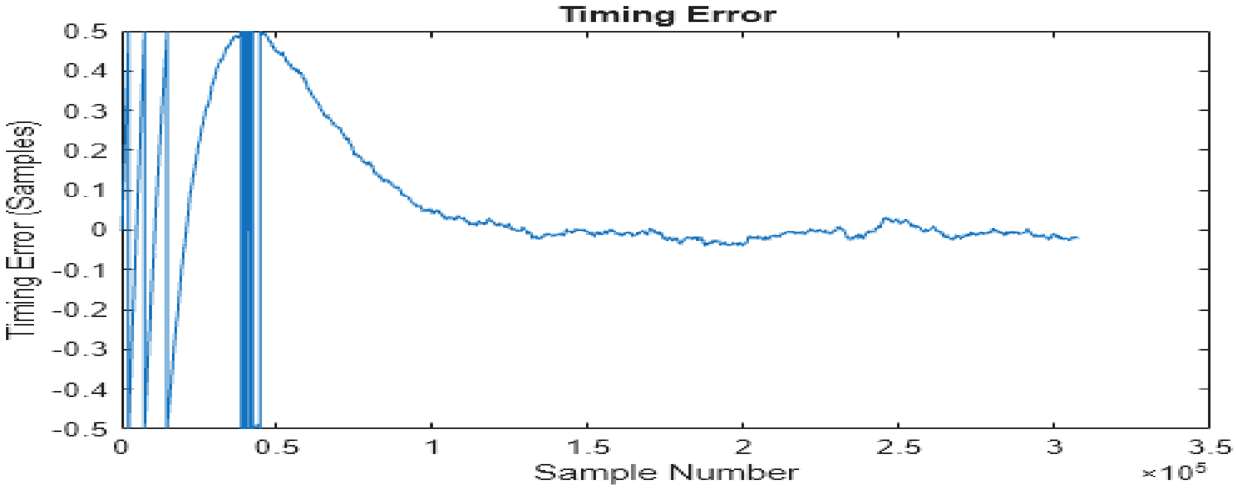


Plot the timing error. Differences in transmitter and receiver clocks show as a positive or negative sloping error over time.

```
figure;
plot((timingError-0.5)-sign(timingError-0.5)*0.5);
title('Timing Error');
xlabel('Sample Number');
ylabel('Timing Error (Samples)');
```



Demodulation and Phase Ambiguity Resolution

Use a `pskdemod` function to demodulate the OQPSK samples. OQPSK modulation and demodulation incur system delays that depend on the pulse shaping used. Account for the system delay when extracting the I and Q bitstreams for the demodulator.

During frequency compensation and timing recovery, the received constellation can rotate significantly while the algorithms lock on to the proper frequency and phase. Both algorithms are designed to rotate and stabilize the received constellation to match the reference QPSK constellation, but due to the symmetry of the QPSK constellation, the received signal can be out of phase by 90, 180, or -90 degrees. Furthermore, the signal timing might have drifted such that the I and Q streams are reversed due to the time offset of the I and Q streams. Test all 90 degree phase rotations and I-Q/Q-I combinations to determine the proper phase and timing by using the preamble as a reference source.

```
preambleDetector = comm.PreambleDetector(Threshold=1400,Detections='First');

% Form the chip sequence of the preamble for phase ambiguity resolution
preambleChips = reshape([DSSS.PRN_I(1:3200).'; DSSS.PRN_Q(1:3200).'],[],1);
```

```
% Rotate the constellation 90 degrees and search for preamble
for p=0:1
    for k=0:3
        % Demodulate and extract out the I & Q bit streams
        rxSig = pskdemod(syncedQPSK*exp(1i*k*pi/2),4,pi/4,"OutputType","bit");
        rx.Ci = [rxSig(1:2:numChips*2);0];
        rx.Cq = [rxSig(2:2:numChips*2);0];

        % Use the preamble detector to search for the preamble sequence
        preambleSymbol = preambleChips-0.5;
        preambleTest = reshape([rx.Ci(1+p:3300+p) rx.Cq(1:3300)]',[],1)-0.5;
        preambleDetector.Preamble = preambleSymbol;
        [pIdx,pMet] = preambleDetector(preambleTest);
        release(preambleDetector);

        if ~isempty(pIdx)
            fprintf('Preamble detected\n');
            break;
        end
    end
    if ~isempty(pIdx)
        break;
    end
end
```

```
Preamble detected
```

```
if isempty(pIdx)
    error('Preamble not decoded correctly');
end
% Compute the start indices for the I and Q streams
preambleTestStartIdx = pIdx - numPreambleChips + 1;
sIdxI = preambleTestStartIdx + 2*p;
sIdxQ = preambleTestStartIdx + 1;

% Extract preamble and payload
rx.Ci = rxSig(sIdxI:2:sIdxI+2*numChips-1);
rx.Cq = rxSig(sIdxQ:2:sIdxQ+2*numChips-1);
```

DSSS Despreading

Detect the binary sequence by correlating the PRN sequence with the received chips. Reshape each I and Q chip vector into an array with 256 rows and 150 columns, with each column corresponding to one bit with spreading factor 256. If the chips in a column correlate with the PRN sequence, it is decoded as a logical 0. For maximum likelihood (ML) decoding, a logical 0 is detected when more than half of the chips (more than 128 chips) are correlated with the PRN sequence. Otherwise, it is a logical 1. Interleave the despread I and Q streams into a single bit stream.

```
% Correlate the PRN sequence with the received chips
Ibdn = (xor(rx.Ci, DSSS.PRN_I));
Qbdn = (xor(rx.Cq, DSSS.PRN_Q));
Ichips = reshape(Ibdn,system.spreadingFactor,[]);
Qchips = reshape(Qbdn,system.spreadingFactor,[]);

% Perform ML decoding
threshold = system.spreadingFactor/2;
Ibits = sum(Ichips,1) > threshold;
Qbits = sum(Qchips,1) > threshold;

% Demultiplex the I and Q streams into a single bitstream
despreadMessage = reshape([Ibits;Qbits],[],1);
```

Error Detection and Correction

Perform Bose–Chaudhuri–Hocquenghem **(**BCH) decoding on the payload, detect the number of bit errors, and correct the bit errors. The (255,207) BCH code can correct up to six bit errors. If there are more than six bit errors detected, the codeword cannot be corrected.

```
hDecode = comm.BCHDecoder(255, 207, ...
    ['x48+x47+x46+x42+x41+x40+x39+x38+x37+x35+x33+x32+x31+x26+' ...
    'x24+x23+x22+x20+x19+x18+x17+x16+x13+x12+x11+x10+x7+x4+x2+x+1'], 202);
[rxPayload,errs] =  hDecode(despreadMessage(system.preambleLength+1:packetLength));

% The BCH decoder returns a negative number in errs if too many bit errors are detected
to be correctable
if errs >= 0
    fprintf('Error correction detected %d bit errors\n',errs);
else
    fprintf('Error correction detected too many bit errors to correct\n');
end
```

```
Error correction detected 0 bit errors
```

Summary

This example shows how to demodulate and decode OQPSK-modulated DSSS signal bursts that have been distorted with realistic channel and receiver impairments. After preamble detection, System objects establish rapid synchronization by estimating the frequency and phase offset and performing timing recovery before decoding the payload data. Finally, the example shows how to demodulate, despread, and correct errors in the payload data.

Helper Functions

This example uses these helper functions:

- `helperPolyphaseCorrelator.m` - Correlates a reference signal over different sample phases of a signal.
- `helperDSSSTransmitter.m` - Generate a DSSS baseband signal using OOQPSK modulation.

Local Functions

```
function checkSettings(settings)
    % Check that user settings are valid
    validateattributes(settings.oversampling, ...
        {'numeric'},{'>=',2,'scalar','integer'},mfilename,'oversampling factor');
    if settings.simTime < settings.burstDelay + 1.0
        error('Set simulation time to be more than the burst delay plus one second.');
    end
    validateattributes(settings.txPower, ...
        {'numeric'},{'<=',33,'scalar'},mfilename,'transmit power');
end

function out = saturate(in,absLimit)
    % Saturate a complex-valued signal
    out = in;
    satidx = abs(in) > absLimit; % identify indices of samples that exceed absLimit
    out(satidx) = absLimit * sign(in(satidx));
end
```

References

[1] COSPAS-SARSAT C/S T.018. "Specification for second-generation COSPAS-SARSAT 406-MHz distress beacons." *International Cospas-Sarsat Programme*

[2] International COSPAS-SARSAT Program

See Also

Functions

- `pskmod` | `pskdemod`

Objects

- `comm.PNSequence` | `comm.SymbolSynchronizer`