AE410 : NAVIGATION & GUIDANCE

ASSIGNMENT - 01

FUNDAMENTAL OF GPS | 1-ACQUISITION

Submitted by : Vishal
Roll no. : 200010085
Prog. : B.Tech (Aerospace)

Department of Aerospace Engineering
Indian Institute of Technology Bombay
Nov 06, 2023

1. Write a MATLAB/ Python/C/C++ program to implement serial search and parallel code phase search acquisition algorithm. Identify the satellites (PRN IDs), carrier frequency, and code phase using the acquisition algorithm in the data file provided.

Import the required library packages

```
import numpy as np
from tqdm import tqdm
```

```
def generate_PRN_code(prn):
    # Define the shift array for G2 code generation
     g2_shifts = [5, 6, 7, 8, 17, 18, 139, 140, 141, 251, 252, 254, 255, 256, 257, 258, 469,
470, 471, 472,
                   473, 474, 509, 512, 513, 514, 515, 516, 859, 860, 861, 862, 145, 175, 52,
21, 237, 235,
                 886, 657, 634, 762, 355, 1012, 176, 603, 130, 359, 595, 68, 386]

    # Determine the appropriate shift for the given PRN number for G2 code
    g2_shift = g2_shifts[prn - 1]

    # Generate G1 code sequence
    g1_code = np.ones(1023)
    g1_register = -np.ones(10)
    for i in range(1023):
        g1_code[i] = g1_register[9]
        feedback_bit = g1_register[2] * g1_register[9]
        g1_register[1:] = g1_register[:-1]
        g1_register[0] = feedback_bit

    # Generate G2 code sequence
    g2_code = np.ones(1023)
    g2_register = -np.ones(10)
    for i in range(1023):
        g2_code[i] = g2_register[9]
         feedback_bit = g2_register[1] * g2_register[2] * g2_register[5] * g2_register[7] *
g2_register[8] * g2_register[9]
        g2_register[1:] = g2_register[:-1]
        g2_register[0] = feedback_bit

    # Shift the G2 code sequence
    g2_code = np.roll(g2_code, g2_shift)

    # Form the single sample C/A code by multiplying G1 and G2
    ca_code = -(g1_code * g2_code)

    return ca_code
```

The code generates a function, generate_PRN_code(prn), designed for generating PRN codes utilized in GPS signal processing. It employs predetermined shift values, feedback shift registers (specifically G1 and G2), and element-wise operations to create CA code. This CA code enables GPS receivers to detect and synchronize with satellite signals.

```
# Define the path to the data file
data_path = 'C:/Users/visha/Documents/Jupyter notebook files/AE410 assignment/RTLSDR_Bands-L1.uint8'

# Load the data from the file
# The data is in 8-bit unsigned integer format
with open(data_path, 'rb') as file:
    raw_data = np.fromfile(file, dtype=np.uint8, count=4092)

# The I and Q components are interleaved in the file, so we need to separate them.
i_components = raw_data[0::2].astype(np.float64) - 128.0
q_components = raw_data[1::2].astype(np.float64) - 128.0

# Combine I and Q to form complex samples
iq_samples = i_components + 1j * q_components
```

This part of code loads raw GPS signal data from `RTLSDR_Bands-L1.uint8`. It then separates the In-phase (I) and Quadrature (Q) components, adjusting them to a suitable numerical range. These components are then combined to form complex samples, essential for further signal processing.

```
len(iq_samples)
```

```
def generate_ca_signal(ca_code, circular_shift, upsampling_factor, doppler_frequency):
    # Perform circular shift on the C/A code
    shifted_ca_code = np.roll(ca_code, circular_shift)

    # Upsample the code by repeating each element 'upsampling_factor' times
    upsampled_ca_code = np.repeat(shifted_ca_code, upsampling_factor)

    # Generate a time vector (assuming the code length is in milliseconds)
    time_vector = np.linspace(0, len(upsampled_ca_code) * 1e-3, len(upsampled_ca_code), endpoint=False)

    # Modulate the upsampled C/A code by a complex exponential to represent the Doppler shift
    doppler_shifted_ca = upsampled_ca_code * np.exp(1j * 2 * np.pi * doppler_frequency * time_vector)

    return doppler_shifted_ca
```

The function `generate_ca_signal(ca_code, circular_shift, upsampling_factor, doppler_frequency)` starts by applying a circular shift operation to the provided CA code, facilitating synchronization with incoming signals. Subsequently, the code undergoes upsampling, with each element being repeated a number of times determined by the `upsampling_factor`. A time vector is then generated, assuming the code length is in milliseconds, aiding in temporal representation. Finally, the upsampled CA code is modulated by a complex exponential, effectively simulating the Doppler effect introduced by relative motion between the GPS satellite and receiver.

```
import numpy as np
from scipy.signal import correlate
```

```
def     find_best_match(input_signal,     doppler_range,     num_prns,     sample_prn_code,
sampling_factor=2):
    best_match_info = {'doppler': 0, 'shift': 0, 'value': 0, 'prn': 0}
    max_correlation_value = 0

    for prn in range(1, num_prns + 1):
        prn_code = sample_prn_code(f'./CA_Codes/prn{prn}.txt')

        for shift in range(1023):  # 1023 shifts for PRN code
            for doppler in doppler_range:
                                doppler_shifted_ca_code  =  generate_ca_signal(prn_code,  shift,
sampling_factor, doppler)
                         correlation_result  =  correlate(input_signal,  doppler_shifted_ca_code,
mode='full')

                max_correlation = np.max(correlation_result)
                if max_correlation > max_correlation_value:
                    max_correlation_value = max_correlation
                     best_match_info.update({'doppler': doppler, 'shift': shift, 'prn': prn,
'value': max_correlation_value})

    return best_match_info

#  The  sample_prn_code  and  generate_ca_signal  functions  would  need  to  be  defined  in  Python
as well.
# For example:
def sample_prn_code(file_path):
    # Read the PRN code from a file and return it as a numpy array
    with open(file_path, 'r') as file:
        return np.array([int(line.strip()) for line in file])
```

The `find_best_match` function conducts an exhaustive search to identify the best match between the input signal and generated CA codes. The function iterates through different PRN codes, shifts, and Doppler frequencies to find the most correlated signal. The result is a dictionary containing information about the best match, including Doppler shift, shift value, PRN number, and correlation value.
The `sample_prn_code` function reads a PRN code from a file and returns it as a numpy array.

```
import numpy as np

def    find_best_matches_1(input_signal,    doppler_range,    num_prns,    sample_prn_code,
sampling_factor=2):
    top_five_matches = []

    for prn in tqdm(range(1, num_prns + 1), desc='Loop 1'):
          prn_code  =  sample_prn_code(C:/Users/visha/Documents/Jupyter  notebook  files/AE410
assignment/CA_Codes/prn{prn}.txt')

        for shift in tqdm(range(1023), desc='Loop 2'):  # Assuming 1023 shifts for PRN code
            for doppler in doppler_range:
                                doppler_shifted_ca_code  =  generate_ca_signal(prn_code,  shift,
sampling_factor, doppler)
```

```
                   correlation_result = np.correlate(input_signal, doppler_shifted_ca_code, mode='full')

                max_correlation = np.max(correlation_result)
                                 if  len(top_five_matches)  <  5  or  max_correlation  > top_five_matches[0]['value']:
                     match_info = {'doppler': doppler, 'shift': shift, 'prn': prn, 'value': max_correlation}
                    if len(top_five_matches) >= 5:
                        top_five_matches[0] = match_info  # Replace the smallest
                    else:
                        top_five_matches.append(match_info)  # Append to the list

                    # Sort the list based on the correlation value in descending order
                    top_five_matches.sort(key=lambda x: x['value'], reverse=True)

    return top_five_matches

# Make sure to define `generate_ca_signal` and `sample_prn_code` functions appropriately
```

The `find_best_matches_1` function conducts an extensive search to identify the top five matches between the input signal and generated CA codes. The function iterates through different PRN codes, shifts, and Doppler frequencies, evaluating correlation values. The result is a list containing information about the top five matches, including Doppler shift, shift value, PRN number, and correlation value.

```
print(find_best_matches_1(iq_samples,        list(range(-5000,5000,500)),32,sample_prn_code=sample_prn_code))
```

```
from numpy.fft import fft, ifft

def parallel_code_phase_search(input_data, doppler_range, doppler_step, sampling_rate):
    s = sampling_rate
    best_match_info = {'doppler': 0, 'shift': 0, 'prn': 0}
    max_correlation_value = 0
    t = np.arange(0, 1, 1 / (s * 1023)) * 1e-3
    doppler_freq_list = np.arange(*doppler_range, doppler_step)

    for prn in tqdm(range(1, 33)):
        ca_code = generate_PRN_code(prn)
        ca_sampled = np.repeat(ca_code, s)
        ca_fft_sampled = fft(ca_sampled)

        for dp_freq in doppler_freq_list:
            sampling_freq = 2.048e6
             doppler_shift = np.exp(-1j * 2 * np.pi * dp_freq * np.arange(len(input_data)) / sampling_freq)

            input_shifted_fft = fft(input_data) * doppler_shift
                   assert  len(input_shifted_fft)  ==  len(ca_fft_sampled),  "The  lengths  of Ca_shifted and input_data should be the same."
            result_fft = input_shifted_fft * np.conj(ca_fft_sampled)
```

```
            correlation = abs(ifft(result_fft))

            if np.max(correlation) > max_correlation_value:
                max_correlation_value = np.max(correlation)
                best_match_info['doppler'] = dp_freq
                best_match_info['shift'] = np.argmax(correlation)
                best_match_info['prn'] = prn
        print(best_match_info)
    return best_match_info
```

The `parallel_code_phase_search` function conducts an exhaustive search to identify the best match between the input signal and generated CA codes.

The function takes `input_data`, `doppler_range`, `doppler_step`, and `sampling_rate` as inputs. It initializes parameters and creates a time vector `t` based on the sampling rate.
The function iterates through PRN codes from 1 to 32. For each PRN code, it generates a CA code and prepares it for correlation. It then loops through a range of Doppler frequencies to account for relative motion effects.
Within the loops, the function performs Fourier Transforms, applies Doppler shifts, and conducts cross-correlation. The correlation results are analyzed to find the maximum correlation value and its corresponding parameters.
The function asserts that the lengths of the shifted input data and the CA code are the same. Finally, it updates the best match information based on correlation results.
The final output is a dictionary containing details of the best match, including Doppler frequency, shift, and PRN number.

```
parallel_code_phase_search(iq_samples, (-5000,5000), 500, 2)
```

It processes GPS signals, generating PRN codes and finding their best matches in received data for accurate navigation.

The jupyter notebook file can be found at the following link : https://github.com/Vish-2003/1-acquisition.git