Sonny Astani

Department of Civil and Environmental Engineering

The SynKpacK User Guide

Release 1.0 (beta)

Authors:
Preetham Aghalaya Manjunatha
Sami F. Masri

Updated On:
Saturday 11th January, 2014

Contents

Page

List of Tables v

List of Figures vii

1 Introduction 1
- 1.1 What is SynKpacK? . . . 1
- 1.2 Flow of the SynKpacK program . . . 1
- 1.3 What SynKpacK is not? . . . 3
- 1.4 Licensing . . . 3

2 Installation 4
- 2.1 Supported platforms . . . 4
- 2.2 External packages and toolboxes . . . 4
- 2.3 Installation instructions . . . 5

3 SynKpacK Input/Output Structures 6
- 3.1 Camera structure . . . 6
- 3.2 SensorData structure . . . 6
- 3.3 Signal processing structure . . . 7
- 3.4 Synchronization structure . . . 8
- 3.5 Post-processing structure . . . 8

4 SynKpacK Class Files 11
- 4.1 Camera constructor and methods . . . 11
- 4.2 SensorData constructor and methods . . . 11
- 4.3 Signal processing constructor and methods . . . 12
- 4.4 Synchronization constructor and methods . . . 12
- 4.5 Post-processing constructor and methods . . . 12

5 A Quick Start Example 14

Bibliography 17

List of Tables

Page

List of Figures

Page

Figure 1.1 A typical flowchart of the SynKpacK 2

Chapter 1

Introduction

1.1 What is SynKpacK?

SynKpacK is an object oriented data processing, calibration, synchronization, signal processing and post-processing toolkit written in Matlab® to access and synthesize the data collected by heterogeneous sensors. SynKpacK can synchronize the data obtained by any number of sensors, without loosing accuracy for single sensor calibration method. SynKpacK can also be used as a single handed signal processing toolkit for filtering, digital integration and other tasks including the default Matlab® functions. The prime strength of the SynKpack is to calibrate the color and depth sensor obtained against the conventional inertial sensors (such as linear variable differential transformer (LVDT) and accelerometers). Lastly, SynKpacK is highly capable of synchronizing the RGBD camera array data and concurrent visualization of the motion captured with aesthetic figures or plots. In the current version (release 1.0), SynKpacK can effectively handle Microsoft® Kinect based datasets, also provision is made to handle Asus and SoftKinetic model datasets.

To use SynKpacK effectively, one need to know at least a bit about objet oriented programming (OOP) language, signal processing and off-course intermediate skills in Matlab® (some of its relevant toolboxes that are required used in SynKpacK). For a good background in OOP, it is recommended to refer the best practice book like McLaughlin et al. [1]. As far as Matlab® object oriented programming references are concerned, author would recommend Matlab OOP basics and Matlab OOP complete manual which is freely available for download.

Most of the SynKpacK is implemented in Matlab®, although some of the OpenCV routines are complied from C/C++ to Matlab® by using Matlab® Mex files in order to succesfully read the binary video files of the Kinect cameras. OpenCV to Matlab® compilation are not the scope of this manual, reader can use the numerous references in OpenCV and Matlab Matlab-OpenCV communities. In particular Kinect Matlab by Dirk-Jan Kroon is highly recommended. Data and synchro specifications are constructed using common Matlab® operations and functions, and standard Matlab® code can be freely mixed with these specifications. This combination makes it simple to perform the calculations needed to decide upon the synchronization and data denoising problems, or to process the results obtained from their solution.

1.2 Flow of the SynKpacK program

Typically SynKpacK flow is similar to any data processing algorithm, refer to Figure 1.1. Firstly, it is all about reading and processing some of the memory underdogs like text files. So that the LED threshold and other basic input needed to proceed further are recorded. Secondly, accelerometer data will be synthesized in order to obtain the velocities and displacements of inertial sensors, off-course this involves internal functions that deals with filtering and other signal processing techniques. Thirdly, the trigger of LED is searched through first 20-60% of the RGBD camera data by utilizing the previously defined LED threshold. If the calibration mode is selected over the regular one, then relevant parameters such as corner detection and depth value averaging are automatically taken care by the software. Once all the pre-processing of the data are executed, then starts the main synchro loop. In synchro loop, camera frame numbers, its corresponding inertial sensors values, GPS data and other sensors data will be dragged back and forth in order to align with reference to the camera frames. Lastly, aligned data can be visualized by using series of plots and multi-tiled video to mimic

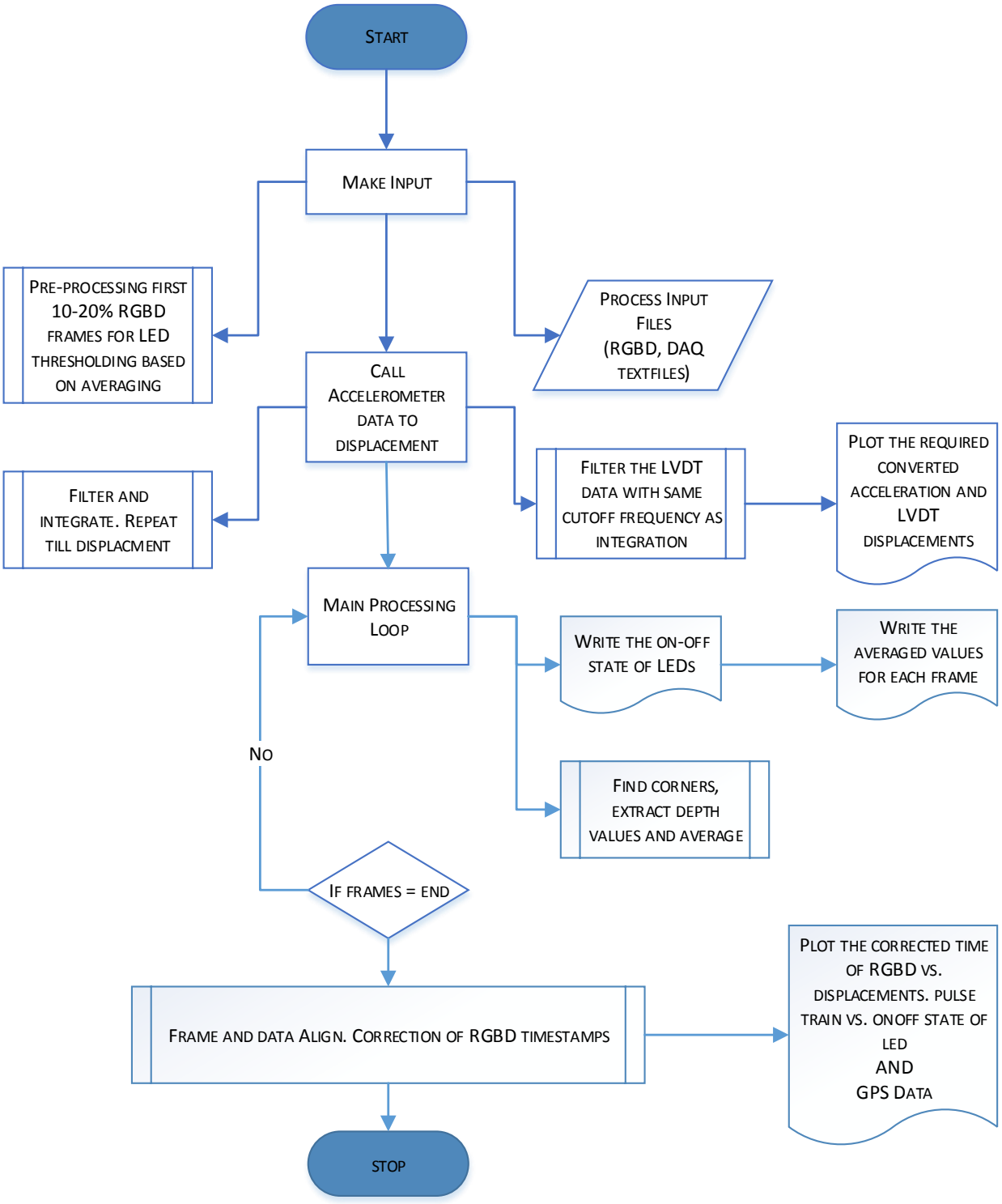

Figure 1.1: A typical flowchart of the SynKpacK

camera color space envelope. On a whole, few steps can fetch a synchronized results. It should be noted that the intricated details within each subroutine of the flowchart will be discussed in the following sections.

1.3 What SynKpacK is not?

SynKpacK is not a synchronization package based on color/depth spatial information. SynKpacK is not meant to be a tool for checking if the given problem dataset is synchronized. SynKpacK mainly assumes that the datasets are basically hardware level synchronized before hand, i.e. all the inertial, acoustic or other sensors are sampled at a specified rate using a analog to digital converter (example: National Instruments, NIDAQ 6008/6009) or data acquisition (DAQ) module. Further, in order to mark a superficial reference of start of the synchronization an LED is used to map the trigger point of DAQ module and the first frame which captured the LED blink event.

SynKpacK is not meant for very large problems, so if your problem is very large (for example, a large image processing or machine learning based synchronization), SynKpacK is unlikely to work faster. For such problems one shall need to directly call a image processing or other data processing high performance solver, or to develop your own methods, to get the efficiency you need.

SynKpacK will solve many medium and large scale problems, provided they have exploitable structure, and it is strongly recommended avoid for loops, which can be slow in Matlab®, and functions like log and exp that require successive approximation. For very large scale problems SynKpacK can play an important role in calibrating the synchronization procedures, however. Before starting to develop a specialized large-scale method, you can use SynKpacK to solve scaled-down or simplified versions of the problem, to rapidly experiment with exactly what problem one need (or want) to solve.

1.4 Licensing

Currently, SynKpacK is in primordial stage and with a considerable efforts this multi-device synchronization package will be made available with zero cost. In other words, the bulk of SynKpacK remains open source under a slightly modified version of the GPL Version 2 license in the future. Further, a small number of files that support the futuristic SynKpacK Professional functionality may remain closed source. In addition to this, all the source code of Matlab®, OpenCV and SynKpacK documentation files that are typesetted in LaTeX will be made available for public domain. The goal is to share and care!

Chapter 2

Installation

SynKpacK installation is in general a cakewalk. Hardly it takes half hour (presuming Matlab® and its toolboxes are already installed) to install the fully functional SynKpacK, a synchronization module. In order to install the SynKpacK module, it is assumed that end user is proficient in using Matlab®, Microsoft® Kinect drivers, OpenCV, and OpenNI open source module. It should be noted that exclusively OpenCV is not needed to run SynKpacK, but it is recommended to that any wrapper functions to Matlab® could be constructed with minimal effort in the later stages.

2.1 Supported platforms

SynKpacK is supported on 32-bit and 64-bit versions of Linux, Mac OSX, and Windows. For 32-bit platforms, MATLAB® version 7.5 (R2007b) or later is required; for 64-bit platforms, MATLAB® version 7.8 (R2009a) or later is required. There are some important platform-specific cautions, however:

- OpenNI 64-bit has to be installed if the OpenCV 64-bit is used
- Matlab® Mex for 64-bit shall be used if the OpenCV and OpenNI 64-bit are used
- As a word of caution, outdated versions of OpenNI may throw errors while execution
- Latest version of Java is recommended so the Matlab® compatibility is not jeopardized

As of version 1.0, support for versions 7.4 (R2007a) or older has been not confirmed. This version is particularly developed and tested in Matlab® R2013b (also in R2013a and R2012b), Windows 8.1, 64-bit operating system. Further, all other dependencies are of 64-bit version. So if one needs to use SynKpacK with the older versions of Matlab, end user is responsible for the incompatibility or missing functions in older versions. It is strongly encouraged to update your Matlab® installation to the latest version (atleast version 2011b) possible. Any bug report or voluntary bug fixes are highly appreciated. Please report any platform related bugs to aghalaya@usc.edu.

2.2 External packages and toolboxes

Some of the external software packages or toolboxes required to execute SynKpacK smoothly are provided in this section. Please note that these are minimal versions in which SynKpacK is developed and tested, this may not limit the use of older versions. Assuming the uncertainty that couple of functions responsible for SynKpacK execution is not provided in the older version is very thin. So here, author is presuming that SynKpacK might even work in older versions like Matlab® 6.5 too!

- Matlab® 7.14 (R2012b)
- Matlab® Mapping Toolbox, version 3.5
- Matlab® DSP System Toolbox, version 8.2
- Matlab® Image Processing Toolbox, version 8.0

- Matlab® Signal Processing Toolbox, version 6.17
- Matlab® Neural Network Toolbox, version 7.0.3
- Matlab® Statistics Toolbox, version 8.0
- OpenNI, Windows 64, version 1.5.2.23
- Kinect sensor driver, Windows 64, version 5.1.0.25
- OpenCV, version 2.4 +
- Kinect Matlab by Dirk-Jan Kroon

2.3 Installation instructions

First of to start with the installation procedure, download the OpenNI executables from the OpenNI stable releases and SensorKinect by PrimeSense. For Simplicity even one can use the executables given in utilities folder.

- Install OpenNI stable SDK, restart.
- Install SensorKinect and restart.
- Install Matlab® with all the above toolboxes, restart if required.
- Copy the SynKpacK_Full to any desired location. Example:
  Drive Letter:\users\Desktop\SynKpacK_Full
- Create a folder called *Test DataSets* with the same level of SynKpacK_Full folder i.e.
  Drive Letter:\users\Desktop\Test DataSets
- Create a folder inside *Test DataSets* that represents experiment set such as *Test-11-13-2013-1636-Hwy-110*, a rough convention is date, time in 24 hours and pavement reference. Inside the example folder *Test-11-13-2013-1636-Hwy-110*, all the data files corresponding to sensors are stored. Such as like Kinect's .ONI extension files, text files, GPS or other sensors output files.
- Lastly, OpenCV and other dependencies could be installed we required.

Chapter 3

SynKpacK Input/Output Structures

In SynKpacK structures are used as inputs for the various classes. Few obvious reasons are the simplicity and organization of all the independent variables corresponding to each class at single location. There are five classes for various tasks, each class has its own input, that are in the form of a structure. All the input structures with its default parameters are given below.

3.1 Camera structure

- camera_input_structure.nocams = n → Number of cameras (say n=4)
- camera_input_structure.newexp = 1 → Is it new experimental test dataset? [No-0 | Yes-1]
- camera_input_structure.LEDshift = 1 → Has LED shifted in Image [No-0 | Yes-1]. Asks to select LED centroid through mouse click. When RGB image appears (left side), press enter key. Next, the crosshair appears, use zoom in/out button to adjust the pixel of interest and left click to extract the centriod location
- camera_input_structure.ledwindow = 3 → LED window size to extract bounding box pixels [Eg. nxn, here n = 3]
- camera_input_structure.thresfrms = n → Thresholding frames needed, usually goes upto to n frames to find the LED threshold. n is 10-20% of the total frames captured by the camera array
- camera_input_structure.videostate = off → Turn video [on | off, a string] (Recommended: off)
- camera_input_structure.camTSname = cam_timestamp.txt → FileName of Kinect timestamps (a string)
- camera_input_structure.camUnixTSname = cam_unixtimestamps.txt → FileName of Kinect Unix timestamps (a string)
- camera_input_structure.LEDUnixTSname = LED_unixtimestamps.txt → FileName of LED blink Unix timestamps (a string)
- camera_input_structure.ONInames = [node_0.oni; node_1.oni; ...] → ONI FileNames Array (string)
- camera_input_structure.ONIname = node_3.oni → ONI FileName to process (a string)
- camera_input_structure.NIoutput = DAQ_Output.lvm → DAQ output FileName
- camera_input_structure.testfolder = Test-11-13-2013-1636-Hwy-110 → Test folder name, inside Test Datasets folder (a string)
- camera_input_structure.handles = [] → Option to pass Kinect handles [Extra]

3.2 SensorData structure

- SensorData_input_structure.TimearrIDX = 1 → Discrete time array of DAQ

- SensorData_input_structure.noacc = 6 → Number of accelerometers
- SensorData_input_structure.LVDTvoltConstant = 0 → LVDT constant for voltage to displacement conversion
- SensorData_input_structure.ACCsensitivity = [] → Accelerometer sensitivity initialize
- SensorData_input_structure.ACCbiasVolt = 2.5 → Accelerometer bias voltage
- SensorData_input_structure.ACCGravity = [] → Accelerometer gravity range
- SensorData_input_structure.accAxis = 1 → Number accelerometer axes
- SensorData_input_structure.AccIDX = 2:1:7 → Column index of channel data in LVM file
- SensorData_input_structure.LedIDX = 8 → Column index of LED channel
- SensorData_input_structure.AcousticIDX = [] → Column index of acoustic sensor channel
- SensorData_input_structure.LvdtIDX = [] → Column index of LVDT channel
- SensorData_input_structure.INPFileName = DAQ_Output.lvm → DAQ output FileName (a string)
- SensorData_input_structure.testfolder = Test-11-13-2013-1636-Hwy-110 → Test folder name, inside Test Datasets folder (a string)
- AccSensiArray = ones(1,sensorINPstruct.noacc) → Accelerometer Sensitivity array (refer accelerometer datasheet/manual)
- AccGravityArray = 9.81*ones(1,sensorINPstruct.noacc) → Accelerometer Gravity (Donot multiply by gravity range of accelerometer (i.e. 2 or 10g))

3.3 Signal processing structure

- Signal_Process_input_structure.FreqCutoff = 0.5 → Cutoff frequency [a | b] a = min and b = max
- Signal_Process_input_structure.FilterType = highpass → Filter type [highpass | lowpass | bandpass]
- Signal_Process_input_structure.FilterMethod = fft → Filter tool [fft | fir]
- Signal_Process_input_structure.Alpha = 0.0001 → FFT minimization constant (alpha)
- Signal_Process_input_structure.FIRorder = 900 → FIR filter polynomial order
- Signal_Process_input_structure.InputSignal = [] → Input signal
- Signal_Process_input_structure.time = [] → Time array
- Signal_Process_input_structure.INPFileName = DAQ_Output.lvm → DAQ output FileName (a string)
- Signal_Process_input_structure.testfolder = Test-11-13-2013-1636-Hwy-110 → Test folder name, inside Test Datasets folder (a string)
- Signal_Process_input_structure.SensorConst.LVDTVConstant = SensorData_input_structure.LVDTvoltConstant → LVDT voltage slope constant
- Signal_Process_input_structure.SensorConst.ACCsensitivity = SensorData_input_structure.ACCsensitivity → Accelerometer sensitivity initialize

- Signal_Process_input_structure.SensorConst.ACCbiasVolt = SensorData_input_structure.ACCbiasVolt → Accelerometer bias voltage
- Signal_Process_input_structure.SensorConst.ACCGravity = SensorData_input_structure.ACCGravity → Accelerometer gravity range

3.4 Synchronization structure

- Synchro_input_structure.LEDthreshld = [] → LED threshold initialise
- Synchro_input_structure.DVAarray = [] → Disp., vel., and acc. array
- Synchro_input_structure.ledwindow = camera_input_structure.ledwindow → LED window size to extract bounding box pixels [Eg. nxn, here n = 5]
- Synchro_input_structure.videostate = camera_input_structure.videostate → Turn video [on | off]
- Synchro_input_structure.ONInames = camera_input_structure.ONInames → ONI FileNames Array
- Synchro_input_structure.ONIname = camera_input_structure.ONIname → ONI FileName to process
- Synchro_input_structure.GPSfile = GPS_output.txt → FileName of GPS output
- Synchro_input_structure.testfolder = Test-11-13-2013-1636-Hwy-110 → Test folder name, inside Test Datasets folder (a string)
- Synchro_input_structure.handles = [] → Option to pass Kinect handles [Extra]
- Synchro_input_structure.alignerType = camfirst → Type of aligner [Camera first - camfirst | LED first - ledfirst] (a string)
- Synchro_input_structure.LEDcamdata = [] → Synthesized LED camera data, initialization to store camera outputs
- Synchro_input_structure.chessgrid_BBox = [225 235 400 360] → Chess grid bounding box, applicable when using the calibration mode
- Synchro_input_structure.NumofSensors = [6 1 1 1] → Number of sensors array [Acc. | LED | GPS | LVDT]
- Synchro_input_structure.sensordata = SensorData_input_structure → Passing a SensorData structure for further processing in Synchronization class

3.5 Post-processing structure

A glossary of distress plot classifier (refer: Post-process_input_structure.plotclassifier)

```
% CLASS UNIQUE ID (CUID)
% Positive integer values

% MARKERS (a string)
% '+' --> Plus sign
% 'o' --> Circle
% '*' --> Asterisk
```

```
% '.' --> Point
% 'x' --> Cross
% 'square' or 's' --> Square
% 'diamond' or 'd' --> Diamond
% '^' --> Upward-pointing triangle
% 'v' --> Downward-pointing triangle
% '>' --> Right-pointing triangle
% '<' --> Left-pointing triangle
% 'pentagram' or 'p'  --> Five-pointed star (pentagram)
% 'hexagram' or 'h''' --> Six-pointed star (hexagram)

% MARKER SIZE
% A positive integer 1-10

% COLOR FORMAT
% [R G B] array [max - 255 | min - 0]

% CLASS TITLE
% Include a class title (a string)

% CLASSIFIER PLOTTING TEMPLATE (as a cell array)
% [CUID | MARKER | MAKKERSIZE | COLOR FORMAT | CLASS TITLE]
```

Actual structure

- Post-process_input_structure.plotclassifier = {{1 '*' 10 [255 0 0] 'Large Distress'}; {2 'p' 10 [0 255 0] 'Medium Distress'}; {3 'h' 10 [0 0 255] 'Small Distress'}};
- Post-process_input_structure.linespecifiers = [0 0] → [Turn on/off speed markers | Turn on/off distress markers]
- Post-process_input_structure.testfolder = Test-11-13-2013-1636-Hwy-110 → Test folder name, inside Test Datasets folder (a string)
- Post-process_input_structure.GPSfile = GPS_output.txt → FileName of GPS output (a string)
- Post-process_input_structure.GPS_latnlog = [] → Initialize GPS latitude and logitude array
- Post-process_input_structure.classi_GPS_latnlog = [] → Initialize categorized GPS latitude and logitude array [Latitude Longitude CUID]
- Post-process_input_structure.x = [] → X array for log plot
- Post-process_input_structure.y = [] → Y array for log plot
- Post-process_input_structure.varargin = [] → Type of log plot [loglog-> log, log | logx -> log, linear | logy -> linear, log | linear -> linear, linear] (a string)
- Post-process_input_structure.GPS_latnlog = [] → GPS latitude and longitude
- Post-process_input_structure.GPS_vehiclespeed = [] → Vehicle speed matrix
- Post-process_input_structure.FFtsignal = [] → Signal to get FFT plot

- Post-process_input_structure.FFT_Fs = 0 → Sampling frequency to get FFT plot
- Post-process_input_structure.pathtype = continuous → How to join GPS latitude and longitude data [discrete | continuous] (a string)
- Post-process_input_structure.ExpName = Test-11-13-2013-1636-Hwy-110 → Test folder name, inside Test Datasets folder (a string)
- Post-process_input_structure.Ori_LEDstates = [] → Original LED state and its time
- Post-process_input_structure.SynKpacK_input = [] → Synchronized data as post-processor input
- Post-process_input_structure.User_Timerange = [275 285] → Time range in seconds [min max] (double or interger datatype)
- Post-process_input_structure.User_Acclerometers = [1 2 3 4 5 6] → Accelerometers ID/numbers for post-processing
- Post-process_input_structure.LEDcamdata = [] → LED camera input
- Post-process_input_structure.vid_delay = 0 → Video delay (seconds)
- Post-process_input_structure.videostate = on → Video state [on/off] (a string)
- Post-process_input_structure.ONIname = camera_input_structure.ONIname → ONI file name to process/display video

Chapter 4

SynKpacK Class Files

In the following sections all the classes, its constructor and methods are detailed. It is assumed that reader has some object oriented programming experience, if not author would strongly recommend some of the references like McLaughlin et al. [1], Matlab OOP basics and Matlab OOP complete manual.

4.1 Camera constructor and methods

Constructor and its object

inputObj = Class_CamData(camera_input_structure) → Outputs an object of the class

Class methods

- output = addtestdatapath (inputObj) → Add test folder path to Matlab® path
- kintextData = getKINdata (inputObj) → Get Kinect processed datasets
- kinhandlez = getKINhandles (inputObj) → Get Kinect handles
- refdata = refreshKINhandles (inputObj) → Kinect handles updater (refresher)
- oneLEDthreshold = getLEDthreshold_oneKinect (inputObj) → Get LED threshold (one camera only)
- skData = getSKdata (inputObj) → Get SoftKinetic processed datasets (currently inoperative)
- asusData = getASUSdata (inputObj) → Get Asus processed datasets (currently inoperative)

4.2 SensorData constructor and methods

Constructor and its object

inputObj = Class_SensorData(SensorData_input_structure) → Outputs an object of the class

Class methods

- output = addtestdatapath (inputObj) → Add test folder path to Matlab® path
- time = getTimeArray (inputObj) → Get the time vector
- Accmat = getAccArray (inputObj) → Get accelerometer data (in form of vector/matrix)
- LEDpulse = getLedArray (inputObj) → Get LED on/off data vector
- Acoumat = getAcousticArray (inputObj) → Get acoustic sensor data

- Lvdtmat = getLvdtArray (inputObj) → Get LVDT sensor data

4.3 Signal processing constructor and methods

Constructor and its object

inputObj = Class_DSPFilters(Signal_Process_input_structure) → Outputs an object of the class

Class methods

- output = addtestdatapath (inputObj) → Add test folder path to Matlab® path
- AccVelDisp = Acc2VDisp(inputObj) → Convert acceleration to velocity, later to displacement (digital integration)
- FilteredValues = DCSigFilter(inputObj) → Signal filter (either FFT or FIR filtered output)

4.4 Synchronization constructor and methods

Constructor and its object

inputObj = Class_Synchro(Synchro_input_structure) → Outputs an object of the class

Class methods

- output = addtestdatapath (inputObj) → Add test folder path to Matlab® path
- paveoutput = sync_Kinect_pavement_data (inputObj) → Synchronize the pavement dataset i.e. Kinect data, LED, GPS data and accelerometer data (currently to provision for acoustic sensors)
- caliboutput = sync_KinCamLedAccLVDT_data (inputObj) → Synchronize the Kinect, LED, accelerometer and LVDT data (calibration only)
- caliboutput = sync_KinCamLedAccLIDAR_data (inputObj) → Synchronize the Kinect, LED, accelerometer and LIDAR data (calibration only)

4.5 Post-processing constructor and methods

Constructor and its object

inputObj = Class_PPHandler(Post-process_input_structure) → Outputs an object of the class

Class methods

- output = addtestdatapath (inputObj) → Add test folder path to Matlab® path
- output = PlotRoadMap(inputObj) → Plotting the Google map through GPS data

- output = logPlots (inputObj) → Log-log plotting
- output = FFTPlota (inputObj) → Filtered signal plots
- output = FullSpan_PavementSynchroPlots (inputObj) → Full-span plots of GPS, accelerometer and LED data
- output = UserSpan_PavementSynchroPlots (inputObj) → User-defined time range plotting of GPS, accelerometer and LED data
- video_show = show_pavement_distress (inputObj) → Show captured video (currently applicable for one camera)

Chapter 5

A Quick Start Example

Step 1: Import the input structures:

```
[camera_input_structure, SensorData_input_structure,
Signal_Process_input_structure, Synchro_input_structure,
Post-process_input_structure, AccSensiArray,
AccGravityArray] = funct_SynKpacK_Struct;
```

Step 2: Call the camera data class instances and pre-requisites:

```
inpOBJ = Class_CamData (camera_input_structure);
(make input object)
inpOBJ.addtestdatapath; (add folder path)
output = inpOBJ.getKINdata; (get Kinect data output)
inpOBJ.getKINhandles; (get Kinect handles)
LEDthreshold = z_camtxtinpOBJ.getLEDthreshold_oneKinect;
(get LED threshold)
```

Step 3: Call the sensor data class instances and pre-requisites:

```
inpOBJ = Class_SensorData (SensorData_input_structure);
(make input object)
timevector = inpOBJ.getTimeArray; (get time vector)
accmatrix = inpOBJ.getAccArray;
(get accelerometer data in form of vector/matrix)
ledvector = inpOBJ.getLedArray; (get LED vector)
lvdtvector    = inpOBJ.getLvdtArray;
(get LVDT data vector)
```

Step 4: Call the signal processing (DSPFilters) class instances and pre-requisites:

```
for i = 1:number_accelerometers (Loop and
initialize for each Accelerometer)

    dspINPstruct.SensorConst.ACCsensitivity
      = AccSensiArray (i); (populate acc. sensitivity values)
    dspINPstruct.SensorConst.ACCGravity
      = AccGravityArray (i); (populate acc. due to
        gravity values)
    dspINPstruct.InputSignal
      = accmatrix (:,i); (populate acc. signal)
    dspINPstruct.time
      = time; (populate time vector)

% Call the constructor
    inpOBJ = Class_DSPFilters (Signal_Process_input_structure);
```

```
% Final Displacment, velocity and acceleration
  of all Accelerometers
    FinalDVA_Array(i)  = inpOBJ.Acc2VDisp;
end
```

Step 5: Call the synchronization class instances and pre-requisites:

```
% Initialize Synchro Inputs (populating variables)
Synchro_input_structure.LEDcamdata  = kinout; (camera output)
Synchro_input_structure.LEDthreshld = LEDthrs; (LED threshold)
Synchro_input_structure.DVAarray    = FinalDVA_Array;
(displacment, veleocity and acceleration array)
Synchro_input_structure.lvdtarray   = FiltLVDT.Signal;
(filtered signal)

% Synchro Instance and its Output
inpOBJ = Class_Synchro (Synchro_input_structure);

% Processed pavement output data
SynKpacKpaveoutput  = inpOBJ.sync_Kinect_pavement_data;
```

Step 6: Call the post-processing class instances and pre-requisites:

```
% Pavement input parameters (populating variables)
Post-process_input_structure.GPS_latnlog      =
SynKpacKpaveoutput.GPS_latnlog; (populate synchronized GPS
latitiude and logitude values)
Post-process_input_structure.GPS_vehiclespeed =
SynKpacKpaveoutput.GPS_vehiclespeed; (populate synchronized GPS
speed values)
Post-process_input_structure.FFtsignal        = []; (Input
signal to display FFT plot)
Post-process_input_structure.FFT_Fs           = 0; (Sampling
frequency of FFT plot)
Post-process_input_structure.Ori_LEDstates    =
[timevector ledvector]; (original DAQ sampled time
and LED vector for full-length plot)
Post-process_input_structure.SynKpacK_input   =
SynKpacKpaveoutput; (synchronized pavement output structure)

% Post-processing Instance and its Output
inpOBJ = Class_PPHandler (Post-process_input_structure);

% Show pavement related plots
roadmapOut = PlotRoadMap (inpOBJ); (Google roadmap with
distress markers and roadmapOut consists of track length)
FullSpan_PavementSynchroPlots (inpOBJ); (Show full-span plots)
UserSpan_PavementSynchroPlots (inpOBJ); (Show user-defined plots)
```



```
% Video input parameters (populating variables)
Post-process_input_structure.LEDcamdata = kinout; (pass
processed camera output)
Post-process_input_structure.vid_delay  = 0; (video
delay in seconds)
Post-process_input_structure.videostate = on; (turn
on/off video, a string)
Post-process_input_structure.ONIname    = node_1.oni (.oni file
to display motion pictures);

% Show pavement distress video
VIDOBJ = Class_PPHandler (PPINPstruct); (construct
show video object)
show_pavement_distress (VIDOBJ); (displaying the pavement video;
supporting one camera currently)
```

Bibliography

[1] McLaughlin, B. D., G. Pollice, and D. West (2006). *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D (Head First)*. O'Reilly Media, Inc.

