Smart Physics—Introduction to Data Acquisition and Processing

Bilal Aftab Usman and Muhammad Sabieh Anwar
Syed Babar Ali School of Science and Engineering, LUMS

Version 1; August 30, 2014

The Smart Physics lab has been designed to introduce the students to two new methods of data collection using commonly available consumer devices. We use a digital camera to record videos in slow motion which allows us to observe objects moving at high speed. We also show how a ubiquitous device like a smartphone can be used to collect data and perform experiments not only in the laboratory but also outside as well.

This primer is meant as a general introduction to the experiments in the Smart Physics Lab. The aim is to explain the operation and working principles of the two methods of data acquisition and processing the data to make assertions about physical principles.

1 High Speed Video Motion Analysis

High-Speed Video Analysis allows us to explore science that is difficult to observe at normal speeds even though video at normal speed is also a very useful medium to study physical phenomena. Video Analysis allows us to track the position of any object of interest with respect to time. We can process the data acquired to learn and analyse the underlying physics. For example, we qualitatively study the motion of the simple pendulum in usual experiments and using the data acquired, we try to ascertain the time period or the value of acceleration due to gravity. These results are very useful but follow a formal protocol. The aim of this primer is to highlight different facets of this formal procedure.

1.1 Image Processing Overview

Our aim is to identify a marker object which has a unique colour relative to surroundings. The steps involved in this “color discrimination” approach are described in the flowchart shown in figure (1).

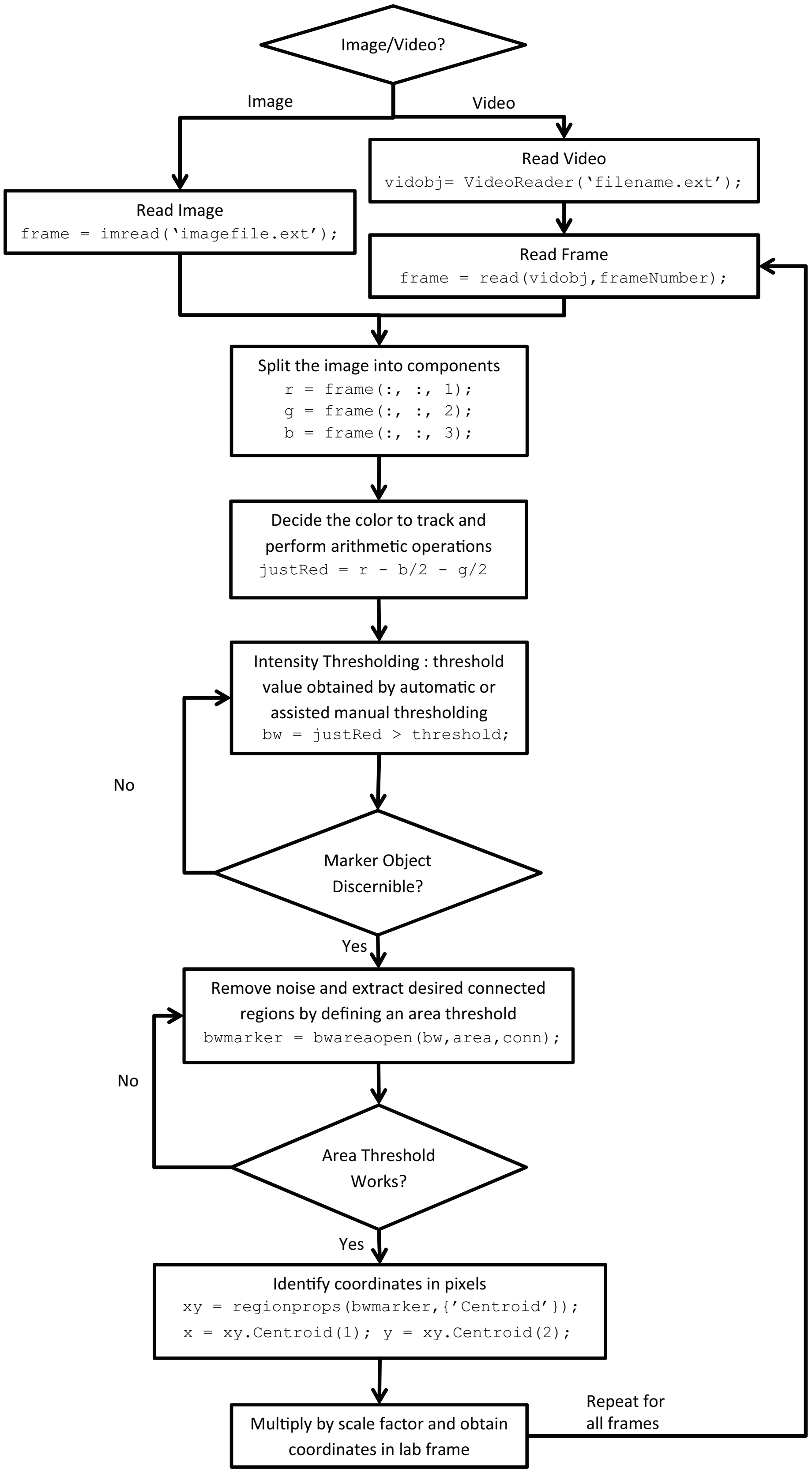

Image/Video?
Image
Video
Read Video
vidobj= VideoReader('filename.ext');
Read Image
frame = imread('imagefile.ext');
Read Frame
frame = read(vidobj,frameNumber);
Split the image into components
r = frame(:, :, 1);
g = frame(:, :, 2);
b = frame(:, :, 3);
Decide the color to track and perform arithmetic operations
justRed = r - b/2 - g/2
Intensity Thresholding : threshold value obtained by automatic or assisted manual thresholding
bw = justRed > threshold;
No
Marker Object Discernible?
Yes
Remove noise and extract desired connected regions by defining an area threshold
bwmarker = bwareaopen(bw,area,conn);
No
Area Threshold Works?
Yes
Identify coordinates in pixels
xy = regionprops(bwmarker,{'Centroid'});
x = xy.Centroid(1); y = xy.Centroid(2);
Multiply by scale factor and obtain coordinates in lab frame
Repeat for all frames

Figure 1: Image Processing Overview

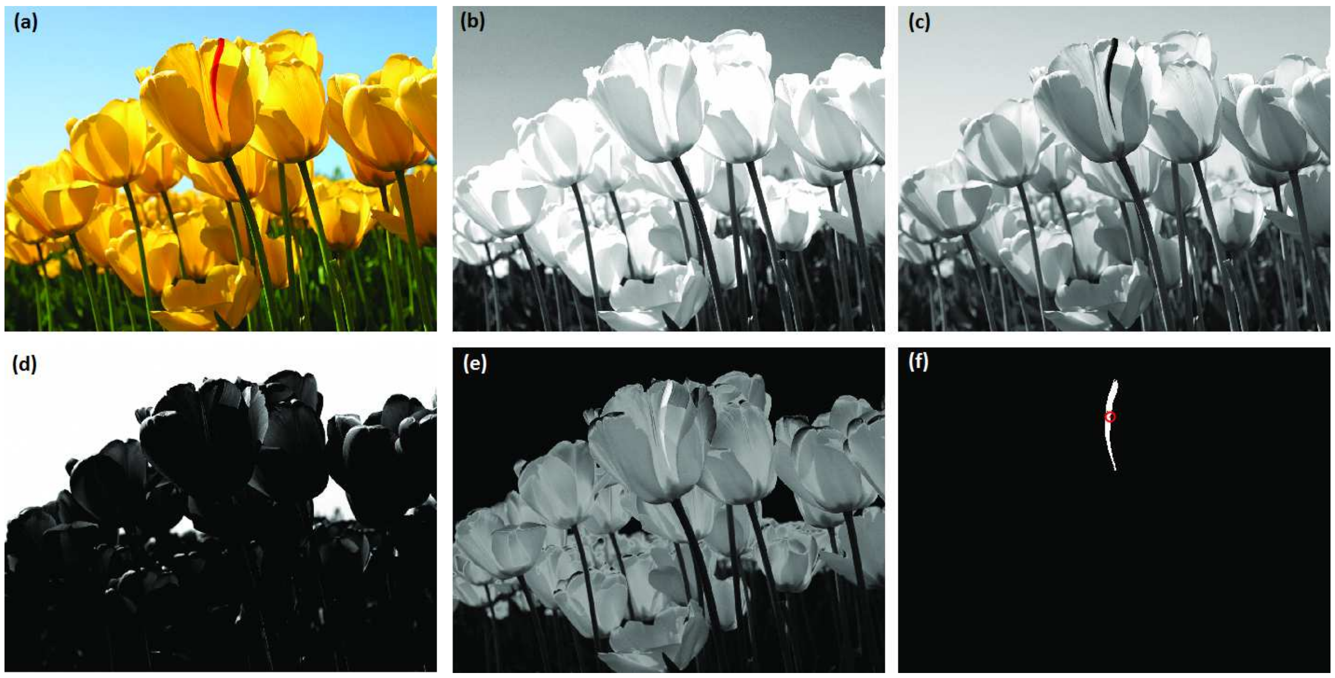


Figure 2: Digital Image Processing. (a) The original image and its (b) red, (c) green and (d) blue components; (e) intensity image after performing the arithmetic operation in equation (**??**) and, (f) the extracted image and centroid identified.

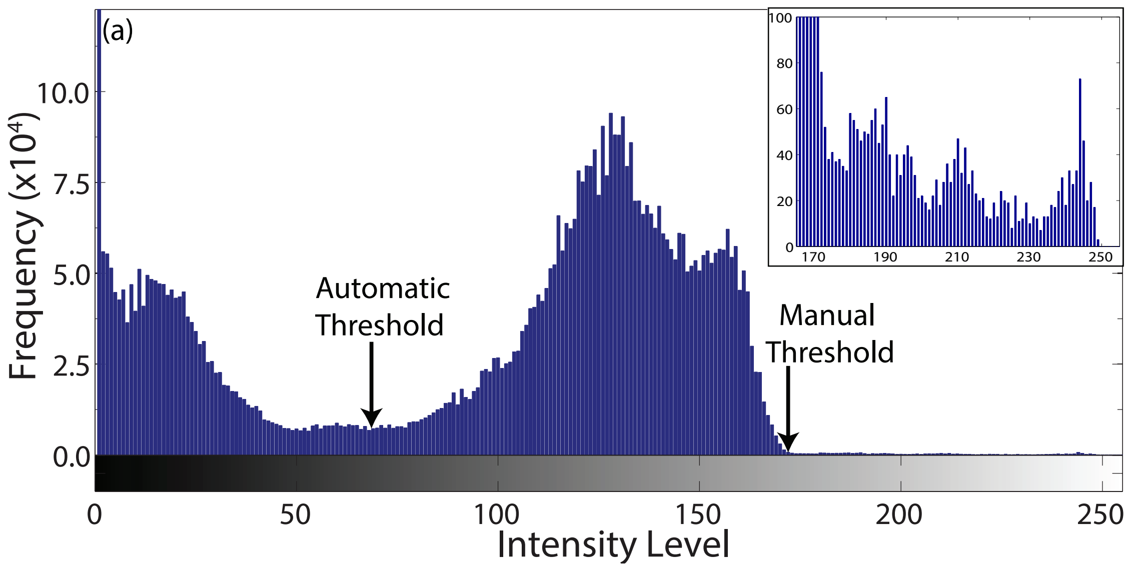
The first step is to read the image and convert it into a matrix. We use the Image Processing Toolbox of MATLAB which has a built-in function "imread" to read a single image file. If we're reading a video, the function "VideoReader" first reads a video file and stores all the frames in an object. Then we simply use the "read" function to extract the image from the video object. Each frame can then be read using the "read" function and processed separately. The image is stored as a $l \times m \times n$ uint8 matrix where l is the length of the frame in pixels, m is the height of the frame in pixels and n is the color component (red, green or blue). The uint8 data type is an unsigned 8-bit integer and means it can stores a value from 0 to 255 for each element. Using the uint8 data type gives us a total of about 16.7 million distinct colours which is more than sufficient for almost all applications in the physics laboratory.

The second step is to split the image into three color components. This can be done by reading the extracted frame's third argument which corresponds to the red (1), green (2) or blue (3) channels. Once the three channels are extracted and stored separately, we have separate intensity matrices for each channel. We can now perform arithmetic operations on these images. For example, adding a constant to any matrix brightens that particular color channel and multiplying by a constant increases the contrast.

Let's take the image in figure (2a) as an example. We would like to extract the red streak in the petal. We perform the following arithmetic operation on the image

$$justRed = r - \frac{g}{2} - \frac{b}{2}. \tag{1}$$

This operation, generally called the "elimination of whites," can be explained quite easily. The



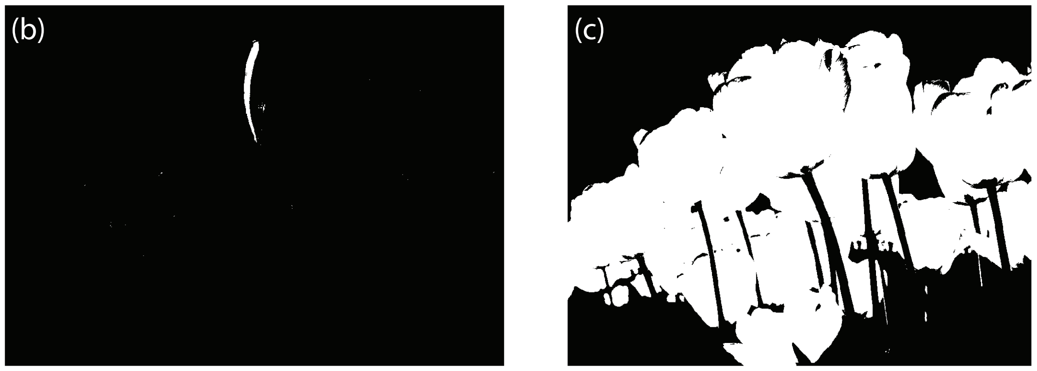


Figure 3: Thresholding (a) Histogram for justRed (the peak at IL=0 is at 200842); inset: histogram for intensity values above manual threshold IL=170, (b) manual thresholding at IL=170 and (b) automatic thresholding at calculated IL=69.

white color in an image is often problematic since it contains high intensities of all three of r, g and b. Thresholding on exclusively the r coordinate may therefore be insufficient in discriminating the pure reds against whites and shades thereof which are prevalent in ordinary laboratory environments. The regions where greens and blues are intermixed with the red therefore have smaller values of the '*justRed*" variable, leaving the pure reds at relatively higher intensities. Ultimately, by choosing a suitable threshold, it is able to easily select the pure red. Furthermore, this operation also minimizes the secondary colors that may comprise of a combination of either "red and blue" (yellow) or "red and green" (magenta). As a result, the secondary colors are converted to lower intensities of "*justRed*" because that is essentially what they are, part red only and cannot qualify as the object of interest. In short, this simple arithmetic operation eliminates all the secondary colors and whites and maps only purely red intensities in a grayscale image. Now, we need an appropriate intensity threshold to extract our region of interest. One way is to search for the threshold manually but this can become a brute force, hit-and-trial exercise albeit needs to be performed only once. For this reason, we start with an intelligent guess of the appropriate intensity threshold. We observe the histogram of the frame and pick the value where high frequency range ends as an intelligent starting guess. This "assisted manual thresholding" greatly reduces the time spent on this hit-and-trial exercise and is very effective since the histogram does not change with time for most experiments and the threshold needs to be selected just once.

Now, we have extracted a binary image in which we can clearly identify our region of interest. However, finding just the connected components is not enough. We need to find a single group of connected components which is our object of interest. There could be noise in the image which is still present in the form of small clusters of pixels not filtered earlier. We now need to define an area threshold where only connected components greater than a particular value of area (in pixels) are considered. We scan for connected components using the "*bwareaopen*" which returns a binary image showing exclusively our object of interest. or each pixel, there are 4 adjacent neighbors and 4 diagonal neighbours. If a pixel is exactly the same as any one of the 4 adjacent pixels, then it is said to be 4-connected. Similarly, if it is the same as 4 adjacent as well as 4 diagonal neighbours, then it is said to be 8-connected. It is also possible to scan for 16 or higher degree of connectedness but scanning for 8-connectedness is sufficient for all practical purposes. The choice of the area threshold is also straightforward. An arbitrarily large value of area, say 200 pixels is enough to eliminate all the smaller connected regions eliminating noise.

We now have a binary image showing exclusively our marker object. We now run the "*regionprops*" command on this binary image and calculate the centroid. The centroid is the mean position of the marker object coordinates. We now have the coordinates of our object of interest in pixels. The red streak has been identified very well as seen in figure (2f). This exercise is repeated for all frames of the video until the coordinates of the marker object have been identified in each frame.

The last step is to multiply the object by a coordinate transformation factor to convert the pixel coordinates to real world coordinates. The scale factor is computed by identifying two points of known distance in the initial frame of the video. After computing their distance in pixels, the known distance is divided by this result to obtain the scale factor.

Here is the snippet of the MATLAB code showing all the above operations for a single frame.

```
frame= read(vidobj,i);

r = frame(:, :, 1);
g = frame(:, :, 2);
b = frame(:, :, 3);

justRed = r − g/2 − b/2;
imhist(justRed); % histogram to select an appropriate intensity threshold
bw = justRed > 170; % intensity threshold set at 170 chosen by eyeballing the histogram
bwmarker = bwareaopen(bw, 200); % area threshold set at 200 pixels
xy  = regionprops(bwmarkergreen, {'centroid'});
x = xy.Centroid(1); % the x−coordinate of the tracked object
y = xy.Centroid(1); % the y−coordinate of the tracked object
```

1.2 Video Tracking - Some Helpful Considerations

It is very important to use a uniquely colored marker object. Using a sharply colored marker object allows the tracking script to clearly identify the marker among background and surroundings. The experimental area should also be brightly lit to get clear images.

It is also a good practice to place the camera about 5-6 feet away from the plane of motion to eliminate perspective distortion.

Another very important decision to make when considering high-speed video motion analysis is to consider which is more important; high frame rate or high resolution. For example, we use a Canon PowerShot SX280 HS camera in our lab. This camera provides us two very good quality high-speed video recording modes, 120 fps (frames per second) at a resolution of 640×480 pixels and 240 fps at a resolution of 320×240 pixels. Additionally, this camera also allows recording at 60 fps at a full HD resolution of 1920×1080 pixels. Using a higher frame rate for video recording means sacrificing the video resolution, so we should decide appropriately. A higher resolution minimizes the error in calculating the coordinates of the object being tracked.

Finally, it is advisable to use a large marker object. A marker object of a large area results in more pixels being identified as a single connected region. This in turn also minimizes the error in calculating the coordinates of the marker. With some careful consideration, this technique provides us an efficient and robust method of data collection. Feel free to float and work on your new ideas using the tools developed here.

2 Data Collection Using Smartphones

These days smartphones come packed with a variety of sensors like an accelerometer (force sensor), gyroscope (angular velocity sensor), touch screen, ambient light sensor, proximity sensor, magnetometer, barometer and even a heart rate monitor. We can use freely available applications in the marketplace to collect and record data being reported by these sensors. In this document, we tend to focus on the inertial motion sensors namely the linear motion sensor (accelerometer) and the rotational motion sensor (gyroscope).

The coordinate system of a smartphone is fixed and rotations about each axis are defined as shown in figure (4).

2.1 Accelerometer

An accelerometer is basically a linear motion sensor. They are sensitive to both linear acceleration and the local gravitational field. Accelerometers have now become indigenous in the automotive industry but have also found their applications in the consumer electronics. Almost all modern smartphones now ship with an accelerometer sensor. For example, the linear sensing provides the smartphone information about its motion and thus taps or shakes can be detected. Similarly,

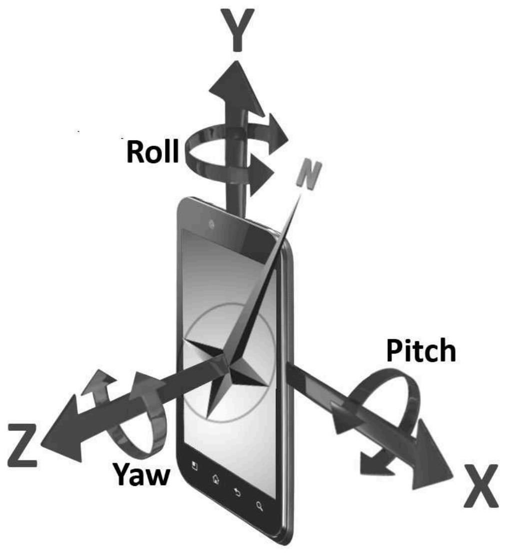


Figure 4: The fixed coordinate system for smartphones (reproduced with permission from *Yole Développement*)

orientation can be determined by the smartphone's sensitivity to the local gravitational field.

The basic principle of operation of an accelerometer is the second law of motion. An accelerometer comprises of a proof mass which is attached to nearly ideal springs and free to move about its mean position. When a force is applied, the accelerometer reports the force required to maintain the proof mass within the accelerometer casing. The accelerometer consists of a proof mass attached through (nearly) ideal springs to a substrate (base). The proof mass can only move up and down. The movable plates (proof mass) and fixed plates (case) construct the capacitors. Capacitive sensing is independent of the base material and relies on the variation of capacitance when the geometry of the capacitor is changing. Thus, a change in distance between the plates creates an electric signal which is detected and fed to a conditioning circuit and the output is then reported in the desired units. In a triaxial accelerometer, there are simply three one dimensional sensors oriented orthogonal to each other.

The data from the accelerometer is conventionally reported in units of g ($1g = 9.81$ m/s^2). We can simply multiply the reading by 9.81 to convert them to correct units of m/s^2. The accelerometer reports a value of $1g$ along the z-axis and 0 along the x and y axes when lying at rest face up on a flat table. This is the initial condition and calibration of any accelerometer in good working condition should give you these results. The gravity vector thus reported is used as a reference for all other linear motion sensing.

We use the Physics Toolbox Accelerometer app to acquire data reported by the accelerometer along the three axes. The recorded data is transferred to the computer via email or using a data cable. The values contain a component of acceleration due to gravity which needs to be eliminated prior to further processing to obtain the real acceleration.

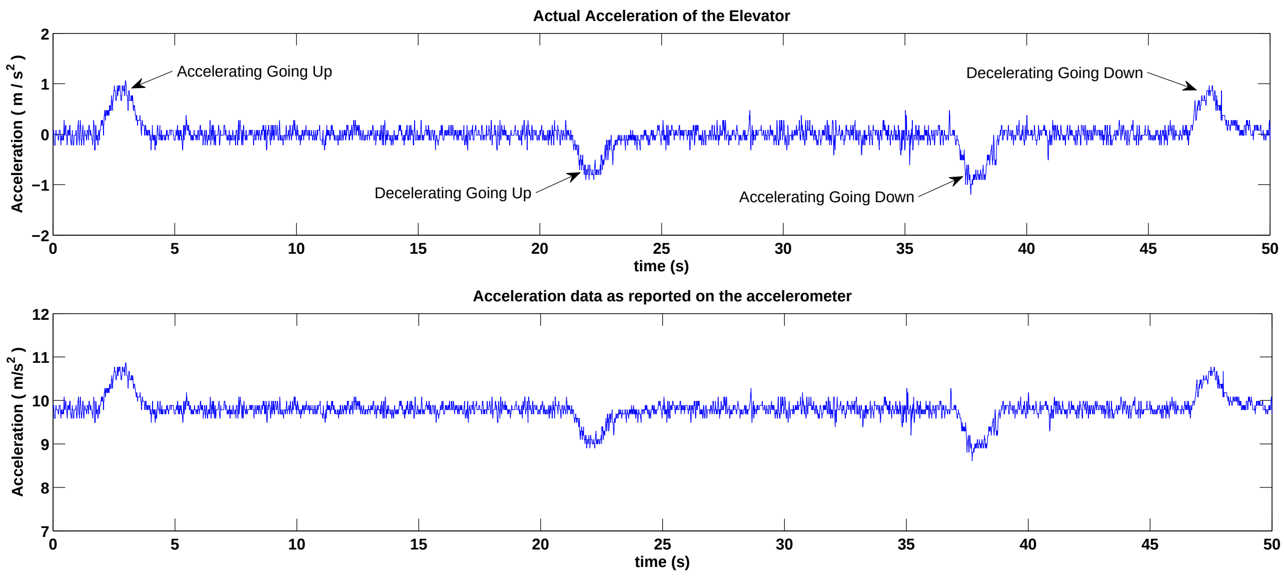


Figure 5: Accelerometer data for a moving elevator

A simple example to understand the readings on an accelerometer is to measure the accelerations of a moving elevator as shown in figure (5). The smartphone is placed on the floor of the car facing up so that it reports an upward acceleration of 9.81 m/s^2. Going up, the elevator starts from rest and accelerates upwards and stops accelerating when it reaches a threshold speed. Before it reaches the destination floor, it decelerates and then comes to a stop. When going down, the same is repeated again in the opposite direction. The rider feels pushed down when the elevator is accelerating upwards. This is of course due to the inertia of the rider (or smartphone in this case) but the effective acceleration reported on the accelerometer provides us a sense of the push or a pull that we feel. The accelerometer also reports the acceleration due to gravity and data must be adjusted for this offset before further analysis is undertaken. We can also observe that the acceleration is not constant throughout the accelerating region contrary to the common conception. We can also notice that the elevator only accelerates to a maximum of about 1 m/s^2. This can explain a safety precaution so as to prevent nausea and other medical conditions that may be induced by these changes in acceleration. This simple example provides a very crude introduction to accelerometer readings and how it can be used to demonstrate common physical phenomena.

A word of caution here. We need to be careful when trying to use the acceleration data in laboratory experiments. As we have already seen, the accelerometer only reports data along its own fixed coordinate system. During the experiments, this fixed coordinate system may be undergoing several transformations with respect to the lab frame. In fact, the equivalence principle renders it impossible to determine experimentally whether a system is subject to a gravitational field or a non-inertial field as a result of coordinate transformations.

2.2 Gyroscope

A gyroscope is a rotational motion sensing device. Gyrsoscopes measure rotation about a fixed axis and the smartphone's gyroscope comprises a small resonating mass which is shifted as the angular velocity changes. This movement is converted into very low current electrical signals which are amplified and read using a host microcontroller. Gyroscopes report data in units of rad/s. They are very sensitive to changes in orientation and even low-quality gyroscopes can sense changes us small as 0.01 rad/s.

However, we need to determine the axis about which we are measuring rotation. The smartphones gyroscope reports rotations about all of the three fixed coordinate axes. It is often useful to invoke the parallel axis theorem to make sense of the data being reported by the gyroscope sensor. For example, if we attach a smartphone to the periphery of a bicycle wheel and rotate, the rotations are about the axle which in turn defines the axis of rotation. However, using the parallel axis theorem, we can easily conclude that the wheel's rotation is the same as if the smartphone is rotating about the axis parallel to the axle (or the bicycle's axis of rotation). Analyzing the system and using such clever bits of knowledge (that one learns in physics) often makes life very easy for the experimental physicist.

3 Analyzing Data

3.1 Least-Squares Curve Fitting

Once we have acquired data, we need to be able to describe the physics behind it. To test the fidelity of data with respect to the physical principles, we fit our acquired data to a suitable physical model. For the purposes of this lab, we have prepared a generic function "lsqfun" based on MATLABs built-in function "lsqcurvefit". Thy syntax for using this function is very simple.

```
c = lsqfun(t,data,model,guess);
% enter mode = 1 for the damped simple harmonic oscillator model
% enter mode = 2 for the quadratic model for linear displacements
% enter a string enclosed in apostrophes for any other model e.g. 'c(1)*xdata+c(2)'
% guess is a starting guess for the parameters. This is a n x 1 matrix where n is
% the number of parameters to be determined. Enter any value if not sure
```

where t is the array of timestamps, *data* is the array of datapoints which is to be fit to the model function, *model* is the physical model to which you wish to fit the data, *guess* is the array of values which the solver can use as a starting guess and c stores the unknown parameters determined by the solver function. We use the following two model functions in most experiments,

1. Damped Simple Harmonic Oscillator Model: 5 unknown parameters.

$$f(t) = c_1(\sin(c_2 t + c_3) * \exp(c_4 t) + c_5). \tag{2}$$

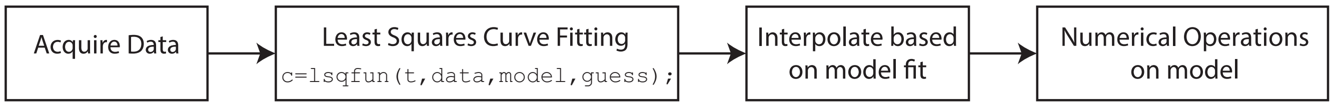

Acquire Data
Least Squares Curve Fitting
c=lsqfun(t,data,model,guess);
Interpolate based on model fit
Numerical Operations on model

Figure 6: Post-processing the acquired data

2. Quadratic Model for linear displacement: 3 unknown parameters.

$$f(t) = c_1 t^2 + c_2 t + c_3. \tag{3}$$

However, the MATLAB function can be modified to fit the data to any model function as long as data permits. You may give any values as a starting guess. However, an intelligent guess can only improve the performances. Therefore, the starting guess is recommended but not necessary.

3.2 Data Interpolation

Data Interpolation is also necessary for some post-processing on the acquired data. Numerical differentiation can be performed on the acquired data to obtain new physical quantities. However, it is possible that the sampling rate may not be high enough for post-processing. This is where data interpolation comes into play. Interpolation means creating new datapoints using the acquired data for further processing. The most common method is through curve fitting. The acquired data is fitted to a suitable physical model and a time array is generated as per the requirement, e.g., step size can be 10× smaller for 10× interpolation. The data points are then calculated by finding the value of the fitted function at each timestamp thus generated.

The following MATLAB code demonstrates how the data acquired through the accelerometer in a simple pendulum experiment is fit to the damped simple harmonic oscillator model and then interpolated.

```
t; % this variable has already been initialized
data; % this variable has also been initialized
c = lsqfun(t,data); % least-squares curve fitting
factor = 10; % for 10x interpolation
tInt = t(1):(t(2)-t(1))/factor:t(end); %interpolation on time array
dataInt = c(1)*cos(c(2)*tInt+c(3)).*exp(c(4)*tInt)+c(5); % data interpolation
% dataInt is the interpolated function from fitting
```

3.3 Numerical Differentiation

We can calculate new physical quantities by calculating derivatives (e.g. velocities from displacement data). Numerical Differentiation is possible to do directly on the acquired data. For this purpose, the generic function "deriv" has been developed and uses the four-point numerical differentiation method for accurate derivatives. Care must be taken that the data is sufficiently interpolated so that the timestep is of the order of at least 10^{-4} s. The syntax for this function is

```
[xd,yd] = deriv(xdata,ydata,order);
```

where *xdata* and *ydata* are input datasets, *order* is the order of differentiation (1 for the first derivative or 2 for the second derivative) and *xd* and *yd* are output datasets.

Using a four-point method means we lose a total of 3 points in the derivative which could be either the first three or the last three datapoints.

The four-point first derivative formula used in this function is

$$\frac{dy}{dx} = \frac{2y_{i+3} - 9y_{i+2} + 18y_{i+1} - 11y_i}{6h}. \tag{4}$$

whereas the four-point second derivative formula used in this function is

$$\frac{d^2y}{dx^2} = \frac{-y_{i+3} + 4y_{i+2} - 5y_{i+1} + 2y_i}{h^2}. \tag{5}$$

4 What are we measuring?

4.1 Uncertainty Analysis

4.1.1 Video Motion Analysis

We can only track and acquire position data along with the corresponding timestamps using video motion analysis. There are several sources of errors that must be accounted for when processing data.

The type A uncertainties present in the data acquired through video analysis are the measurement of the known distance in the frame. We make two measurements for the known distance, one using our measuring instrument like a ruler or vernier calliper and the other by marking two points on the screen to measure the distance in pixels. These two results then combine to form a scale factor. These uncertainties can be minimized by taking multiple readings and evaluating the standard error in the readings.

The type B uncertainties may arise due to least count of the measuring instrument. Also, when we mark the two points on the screen, we assume that each marking could be off by at least 1 pixel. If we wish to consider a 95% confidence interval, we need to consider two standard deviations and that gives an uncertainty of ± 2 pixels for each point that we mark. On top of this, we also need to realize that the same error is present across both coordinates, x and y. Therefore, the total uncertainty is a combination of all of these contributing factors.

4.1.2 Smartphone Sensors

The type A uncertainty in readings from both sensors can be minimized by taking multiple readings and evaluated using the statistical methods for treating data. For type B uncertainties, we need to check the datasheets and then make the decision.

Check the datasheet for the smartphone's accelerometer and gyroscope modules to get the accuracy of the device. For example, the smartphone used in these experiments shipped with a BMA050 3-axis Accelerometer module from Bosch with a maximum operating range of 32 ms^{-2} and an accuracy of 0.0039 ms^{-2}. We can use double this value with 95% confidence in our uncertainty calculations. Similarly, the gyroscope module is an MPU3050c gyroscope sensor from Invensensor with a max range of 34.91 rad s^{-1} and an accuracy of 0.01 rad s^{-1}. We can use double this value of accuracy as our type B uncertainty in the readings from gyroscope.

The two types of uncertainties must be combined to give the total uncertainty in the readings. The accelerometer is very sensitive to vibrations and usually the type A uncertainties dominate. On the other hand, the gyroscope is very sensitive to rotations and usually the type B uncertainty dominates.

4.2 Correlating Data

After we have acquired data through the two methods described above, we may want to check how well both of these methods correlate. The data from video analysis can be fitted to a suitable physical model, interpolated and then differentiated to obtain higher derivatives. For example, in simple pendulum experiment, the displacements can be thus differentiated to obtain velocity and acceleration. The acceleration thus computed can then be compared with the smartphone's accelerometer data.

Correlating data acquired from two entirely separate methods provide valuable insight into the integrity of data and these methods of data collection. A remarkable degree of agreement was found in most experiments and correlation of data will be left as an optional or compulsory exercise at the end of some experiments.

5 Further Reading and References

1. M. Monteiro, C. Cabezay, A.C. Marti, "*Acceleration Measurements Using Smartphone Sensors: Dealing with the Equivalence Principle*", arXiv:1406.3867*v*1 [physics.ed-ph] 15 Jun 2014.

2. A. McAndrew, "*Introduction to Digital Image Processing With MATALB*", (Course Technology - Thomson Learning, Inc., 2004).

3. M. Pedley, "*Tilt Sensing Using a Three-Axis Accelerometer*", Freescale Semiconductor Application Note AN3461, *Rev*.6, (March 2013).

4. J. Kuhn, P. Vogt, "*Applications and Examples of Experiments with Mobile Phones and Smartphones in Physics Lessons*", Frontiers in Sensors (FS) Volume 1 Issue 4, 67 – 73 (October 2013)

5. K. Morken, "*Numerical Algorithms and Digital Representation*" - Lecture Notes for course MATINF1100 Modelling and Computations, (University of Oslo, Ch. 11, 2010)