PHYSTRACK

Experimenter's Reference Manual

Authors:

Muhammad Umar Hasan

Muhammad Sabieh Anwar

Version: 2017-1

Date: April 13th 2017

PhysLab

LUMS

Contents

1. Introduction

PhysTrack is a Matlab based video tracking solution for analyzing kinematics of moving bodies. It is developed by the team of Dr. Sabieh Anwar in PhysLab, LUMS. Since Matlab is a very popular analysis tool in physics laboratories around the globe, we have tried to combine the robustness of Matlab computation power with such a friendly user interface which can be found in commercial video tracking software.

This document serves as a reference guide for the experimenter as well as the teachers who seek to create experiments using PhysTrack. The source code can be downloaded from GitHub repository located at https://goo.gl/DFD6Oa. PhysLab's home page at http://PhysLab.org serves as the primary source of information about video tracking and this library.

1.1. Features

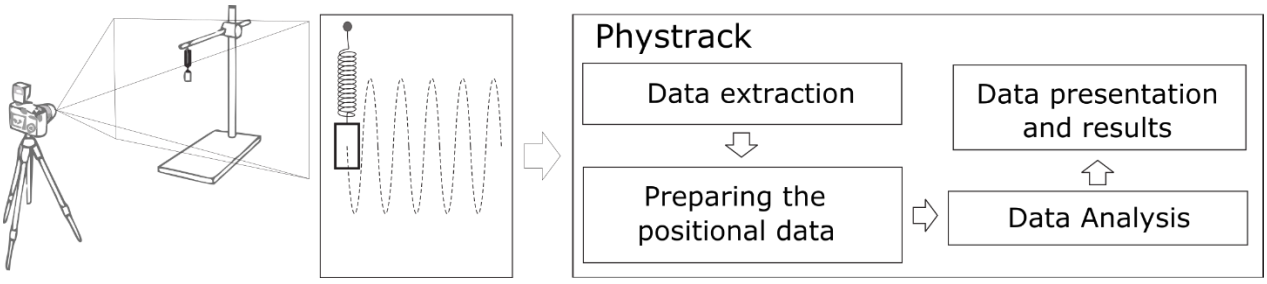
- Along with useful tools for post processing of the data, PhysTrack provides two kinds of object trackers, both of which can automatically track the position and orientation of user selectable objects throughout the video. When considered together, these trackers can work in a diverse range of scenarios. For example, they can simultaneously track multiple objects while ignoring undesired objects, work with non-uniform backgrounds and compensate for jittery camera movements.
- The main aim was to minimize the programming skills to enable the experimenter focus on the physics rather than the program itself. The experimenter can, of course, add new routines to the library if desired. Matlab allows the user to analyze the acquired data using the existing tools like curve fitting, Fourier transformation and plotting. Furthermore, there is no restriction for using specific hardware for video capturing nor the need for fixed sampling rates.
- As far as the kind and morphology of objects that can be tracked are concerned, there are generally two kinds of situations. There are the rigid bodies which can be physically marked with single or multiple track markers and secondly, there are bodies which cannot be marked physically. The latter objects can be considered as particles. Typical examples of the first kind are rotating and translating discs[1], spring pendulum systems[2], bodies colliding on a plane and projectiles[3], while the second kind includes microspheres exhibiting Brownian motion[4], liquid droplets falling down a stream and the movement of a fruitfly[5].

1.2. Getting Started

You are only a couple of steps away from using PhysTrack for your own experiments.

- Download and extract the latest PhysTrack source from https://goo.gl/SuFtuH. It also contains some additional experiment scripts which serve as an example for creating your own experiments.
- Also, choose and download a sample video from the https://goo.gl/MYznt5.
- Open Matlab and change the current directory to the downloaded package. You should see the "+PhysTrack" and "GUIs" directory along with some "analyze motion" scripts in the current address window.
- Depending upon the type of experiment required, select any script from the window and run it. (Or you can type in the name of that experiment in the command window and hit *"Enter"*.
- The scripts are designed to be user interactive. They communicate with the user through GUI's and message boxes and guide through the whole process. The script will also ask to load the experiment video you downloaded in the previous steps.
- Once done, peruse the sample scripts, which are well commented, and try to understand how you can modify the codes for your own experiment. This wiki contains help on all the functions used in the sample scripts.
- Capture your own experiment's video and start making robust video tracking experiment scripts.

1.3. Performing a Physics Experiment with PhysTrack



Performing a classical mechanics experiments using video tracking and performing advance analysis is very simple.

- We capture video of the moving object using a digital camera,

- use one of the automated trackers of PhysTrack to track the objects and generate position and orientation data,
- to investigate the motion, use the in-built Matlab tools or those included in PhysTrack like numerical differentiation, curve fitting, object stitching and coordinate system transformation and
- present the results using Matlab plots and video plots included in PhysTrack.

1.4. Usage Scenarios

Although, PhysTrack can be used to collect positional data of many moving objects and perform analysis on it, its productivity can be fully harnessed with experiment templates. i.e. students don't have to write the whole Matlab code every time they perform an experiment, rather, a sequence of operations is defined and a Matlab script is generated. This script calls all the necessary functions and sub routines to setup an appropriate reference system, unit conversion and analysis environment.

Although, creating a new template may require some expertise with both PhysTrack and Matlab, but we provide many useful scripts on the GitHub repository which can be adapted to create new ones. Links to these scripts can be found in Appendix [7]. All of t

1.5. Requirements

- In addition to having primary knowledge of kinematics, we assume that the user is accustomed with the basics of Matlab as well.
- To capture videos a good slow motion camera is required. In PhysLab, we usually use a Canon PowerShot SX280HS mounted on a tripod stand which works very well with most of the mechanics experiments. This camera can capture video at as high a frame rate as 240fps with a frame size of 320x240 pixels.
- For investigating microscopic motion, a video microscope is also required. In PhysLab, we usually use a Motic BA210 Trinocular for investigating the Brownian motion of micro particles.
- Usually, to move the objects in required fashion, an apparatus is also recommended.
- A computer with RAM >=3GB and an installation of Matlab 2006 (or above) with Image Acquisition Toolbox and Computer Vision Toolbox is also required.

2. Organization of the Codes

PhysTrack consists of two directories.

- +PhysTrack (https://goo.gl/e987ke)
- GUIs (https://goo.gl/guA5kJ)

To use the codes, one needs to browse in the root of PhysTrack library. All the functions and scripts situated in +PhysTrack package can be accessed using a "PhysTrack." prefix. "PhysTrack." followed by the TAB key in the command window will popup a suggestions list containing all the functions and scripts contained within the PhysTrack Package.

Normally, one doesn't need to add GUIs directory to current path because the wrapper functions for all the GUI's are included in +PhysTrack package and they automatically add the GUIs directory to Matlab path when necessary.

Most of the codes have in-line comments and help which can be viewed in Matlab but some important functions are explained in the coming chapters.

Some codes may require a detailed explanation of the underling technique. The reader is suggested to read the PhysLab's primer on video tracking, "Smart Physics with Video Tracking" which can be found on https://goo.gl/jOCgWk.

3. The PhysTrack package

To access the functions inside +PhysTrack directory, one doesn't need to include the folder in current path of Matlab. The "+" sign in the start indicates that PhysTrack is a package which means that all of it's contents can be accessed using a "PhysTrack." prefix. For example, function "VideoReader2" returns a VideoReader2 struct. To access it, following code can be used:

```
vro = PhysTrack.VideoReader2;
```

Some important functions in this directory, their working and the usual usage scenarios are explained in the following sections.

The PhysTrack package includes all of the user level functions to be used for processing videos, extracting positional data and performing analysis.

All the functions are further divided into four major groups:

3.1. Preparing the video for analysis

Before a video can be used with the object trackers, it must be processed first to help the automated algorithms work more efficiently. Some of the following scripts are mandatory and others are available for additional processing.

3.1.1. AdjustBrightnessContrast(image, brightnessCorrectionFactor, contrastCorrectionFactor)

AdjustBrightnessContrast Adjusts brightness and contrast of an image. Correction factors' domain is [-1,+1]. This function can also be used with SetPreProcessingFunction to hook with the read2function. This way, just after applying the cropping and trimming, read2 will run the specified function on the final frame. The processed frame will be used in further analysis.

- AdjustBrightnessContrast(I, 0, 0), AdjustBrightnessContrast(I) returns the original image
- AdjustBrightnessContrast(I, 1, 1) returns an image with maximum contrast and brightness
- AdjustBrightnessContrast(I, -1, -1) returns an image with minimum contrast and brightness

3.1.2. ConvertVideoToBinary (videoReader2Object)

This function is a helper for using the binary video conversion GUI. To convert an existing RGB video reader object to binary use this function to call the convertToBin GUI. The GUI provides necessary tools to specify different thresholds at different indices and color of background of the frame

3.1.3. CorrectHSV (image, hue, saturation, vibrance)

Similar to AdjustBrightnessContrast, this function Applies correction factors on the Hue, Saturation and Vibrance of the given image I. Domain for each factor is [-1,+1].

3.1.4. GetObjects(videoReader2Object)

Gets objects in the first frame of the video reader object. It will show a GUI tool to manually define objects. The tool can also detect object on the basis of binary threshold and movement detection.

3.1.5. read2(vro, frameNumber, indIsAbsolute, forceRGB)

This function has many working modes and depending upon the kind and types of arguments, can give different result. Basically, it is used to read an image frame from a PHYSTRACK.VIDEOREADER2 object. It will use the trimming, cropping, rescaling and binary conversion information embedded in the video reader object and return a final frame after all operations. If a pre-processing function is assigned, it will then apply that function on the final frame and return the image.

```
I = PhysTrack.read2(vro, 1);
%       returns the first frame of the video reader after
```

```
% performing all operations. It will return a binary image if
% the video reader contains binary conversion data.

I = PhysTrack.read2(vro, 1, 1);
% returns the first image of the video sequence
% regardless of the trimming information.

I = PhysTrack.read2(vro, 1, false, true);
% returns the first frame of the trimmed video in RGB, regardless
% of the binary conversion data which may be embedded in the video
% reader object.
```

3.1.6. SetPreProcessingFunction

This function assigns another function which will be used to pre-process each video frame before further processing by the library. The function must take NxM wide uint8 RGB array and return back an image of the same dimensions and bit depth.

3.1.7. TrimVideo

Interacts with the user using a GUI tool to obtain the trimming information for a given videoreader2 object. The usage of the GUI is extremely simple and self explanatory.

3.1.8. VideoDecoder

This function simply converts a compressed video to raw AVI. When using a compressed video, the whole process is slowed down because of the way video reading function works. To fetch any frame N in a video sequence, the video reader function has to convert all the preceding frames first because in a compressed video, no frame is independent of its previous frames. This is resolved by converting the whole video into raw AVI which stores a bitmap image for every frame, independent of the history. This could have been done on the RAM but for bigger video, the memory requirements are so enormous that it's better to wait for the video conversion than to fill the whole RAM space with giant Bitmap data arrays.

3.1.9. VideoReader2

VideoReader2 gets a new video reader 2 object. In contrast to a usual video reader object from the image acquisition toolbox, the video reader2 object embeds the cropping, trimming, and pre-magnification and binary conversion information in a single object. This is necessary to reduce the repetitive process to be done to fetch every frame from the video file.

There are separate GUIs available for each step of conversion, but PhysTrack.VideoReader2 presents all these tools in sequence to allow the user to go through all the steps interactively. Finally, it returns a video reader 2 object.

See Appendix [1] for more information.

3.2. Extracting the Positional Data

Once the video is ready for processing, and the objects for tracking are selected, one of the automated trackers are used to track the objects through the rest of the video.

3.2.1. Kanade Lucas Tomasi Tracker (KLT)

PhysTrack includes a KLT based tracker which uses the KLT algorithm from computer vision toolbox of Matlab to automatically track the desired objects through the video. The tracker identifies some distinct trackable points, called "features", in the small region of interests of the objects and tries to track them in the coming frames. Corners and tiny objects are the ideal features to be tracked. A track point may also lose track somewhere between the processing and the user must manually remove those points using the KLT track point filtration tool. The tracker then averages the position of these trackers in each individual frame to obtain the trajectory of the selected object in all of the video.

Following code can be used to use a KLT tracker.

```
% select a video file, trim and crop it as required.
vro = PhysTrack.VideoReader2;

% graphically select all the objects to be tracked
obs = PhysTrack.GetObjects(vro);

trajectories = PhysTrack.KLT(vro, obs);
% the algorithm first tracks and continuously shows the progress of
% the process.
% Once done, it presents a GUI to filter out the wrong track
% points.
% Eventually, it returns the trajectory in form of a struct
% containing sub-structs named tpX,
% namely, trackPointX for each object identified in the second
% step.
```

3.2.2. Binary Object Tracker (BOT)

The other tracker works on a background difference technique. It's called the Binary Object Tracker (BOT). The algorithm takes the first frame of the binary image and identifies objects by detecting connected regions. Each connected region is an object. Then it matches the location and size of these objects with the object collection given for tracking. By doing this, it actually tries to map each desired object with a detected object approximately sharing its location and size. It then updates the new location of these objects and repeats the process through the rest of the video.

```
vro = PhysTrack.VideoReader2;
% select a video file, trim and crop it as required.
% Also, convert the video to binary using the GUI presented in the
% process.
% Use the binary object detector to detect the objects
obs = PhysTrack.GetObjects(vro);
trajectories = PhysTrack.BOT(vro, obs);
% the algorithm tracks the objects throughout the video, also
% showing which objects lose the track.
% Eventually, it returns the trajectories in form of a struct
% containing sub-structs named tpX,
% namely, trackPointX for each object identified in the second
% step.
```

3.2.3. Real World Coordinate System

Both of the trackers return the trajectories in Image's native coordinate system. It is obvious that we need to convert this data to some real world coordinate system with appropriate distance units. For this purpose, PhysTrack includes a Coordinate System Tool.

We can either define a stationary coordinate system which remains at a fixed place of the image throughout the video, or we could stitch a coordinate system to one or two trajectories. When stitching to one point, a coordinate system will move throughout the video with that tracked point without changing the orientation. When stitching to two points, the system will also rotate with the two points in addition to translation.

Read more about coordinate systems in Appendix [6].

3.3. Analyzing the Positional Data

Once the trajectories are extracted, it's now time for analyzing the positional data. PhysTrack provides several tools to perform some important operations on the data including transformation, numerical differentiation, curve fitting, stitching and video plotting.

3.3.1. TransformCart2Cart (trajectories, coordinateSystem) and InverseTransformCart2Cart (trajectories, coordinateSystem)

One of the very useful functions in PhysTrack is TransformCart2Cart. Once we have a coordinate system, we can transform our calculated trajectories to any coordinate system we define using the coordinate system tool. This function can work well with all three kinds of coordinate systems used in PhysTrack.

One interesting application of using a floating coordinate system is that we don't need complex transformation functions to view the trajectory of one moving object with reference to another moving object. We can simply stitch a coordinate system with the moving object and the coordinate system starts to move with the object. Transforming the trajectory from image's coordinate system to the floating coordinate system would give us out desired trajectory.

The inverse transformation is not a lot different. Once a trajectory is transformed in a specific coordinate system, inverse transforming it with respect to the original coordinate system will return the original trajectory from which that transformation took place.

```
% select a video file, trim and crop it as required.
vro = PhysTrack.VideoReader2;
% graphically select all the objects to be tracked
obs = PhysTrack.GetObjects(vro);
% track the objects
% Once done, it use the GUI to filter out the wrong track points.
trajectories = PhysTrack.KLT(vro, obs);
% define a coordinate system
[rwRCS, ppm] = PhysTrack.DrawCoordinateSystem(vro);
% transform into the coordinate system.
transformedTrajectories = PhysTrack.TransformCart2Cart(trajectories
, rwRCS);
```

3.3.2. GenerateTimeStamps(vro)

The trajectories of the objects are actually arrays of positional data. This data isn't very useful without knowing the time stamps at which that position is observed in the video. This function uses the frame rate information of the video to generate time stamps for the tracked points.

```
% vro = PhysTrack.VideoReader;
% define a video reader 2 object first
t = PhysTrack.GenerateTimeStamps(vro);
```

3.3.3. StitchObjectToPoints(PointOrPointsToStitch, Trajectory1, Trajetory2)

Object Stitching is an extremely useful technique primarily inspired from Adobe After Effects. The idea is to find out the trajectory of an object/point on the object, on which we cannot attach a physical track marker for tracking. So instead, we track the trajectories of two other points and affix the required point with the computed trajectories.

To stitch a graphically acquired point with an object with two track marker, we can use the following code:

```
% suppose we have untransformed trajectories of two points on the
% same rigid body as "traj.tp1" and "traj.tp2", and the video
% reader 2 object as vro.
% We can acquire a point from user using ginput().
% display a new figure
figure;
% fetch the first frame of the video
imshow(PhysTrack.read2(vro, 1));
% acquire the point
P1 = ginput(1);
```

```
% stitch this point to the trajectory
trajofP1 = PhysTrack.StitchObjectToPoints(P1, traj);
```

3.3.4. GetAngDispFrom2DtrackPoints(traj1, traj2)

When we have the trajectories of two points on a rigid body, we can compute its angular displacement without knowing the center. Angular displacement will actually be the angle of the vctor formed by the two points whose trajectory is known. You can read more on this technique in chapter 2.4 section 1 in “Smart Physics with Video Tracking” on https://goo.gl/jOCgWk.

```
% suppose we have untransformed trajectories of two points on the
% same rigid body as "traj.tp1" and "traj.tp2"
dAng = PhysTrack.GetAngDispFrom2DtrackPoints(traj.tp1, traj.tp2);
% actually, while working with struct, we also can simplify it even
% more, the following statement will give the same result as the
% previous one
% dAng = PhysTrack.GetAngDispFrom2DtrackPoints(traj);
```

3.3.5. deriv(xdata, ydata, order)

Numerical differentiation is a very important part of the analysis. For example, we can differentiate displacement computed from the trajectories of the objects with time to obtain velocity and acceleration.

```
% suppose we already have a trajectory named "traj", and time
% stamps named "time"
horizontalVelocity = deriv(time, traj.x, 1);
verticalAcceleration = deriv(time, traj.y, 2);
```

3.3.6. lsqCFit(xdata, ydata, dependent, model, independent, startingValues)

This function uses the Matlab's in-built curve fitting tool "cftool" to obtain a least squares fit of the xdata and the ydata on a mathematical model. Suppose we need to fit the displacement obtained from a trajectory on Newton's second equation of motion, then we could use the following code

```
% suppose we have the displacement as "disp", time stamps as
% "time", and we need to fit this data in the model
% S = So + Vi * t + 0.5 * g * t ^ 2, (So, Vi, and g are unknown).

dispFit = PhysTrack.lsqCFit(time, disp, 'S', 'So + Vi * t + 0.5 * g
* t ^ 2', 't');
% use the fit result to find displacement at some interval 0.452)
dispAtMyTime = dispFit(0.452);
```

3.3.7. VidPlot(vro, xdata, ydata)

This tiny function can create a video plot of the given data. It previews the x and the y data on the actual frame using a continuously changing color. This way, one can visualize how the trajectory was constructed, which, definitely, is useful in understanding the physics behind the motion.

```
% suppose we have the trajectory of a track marker as "traj", and
```

```
% the video reader as "vro"
% Create the video plot
VidPlot(vro, traj.x, traj.y);
% the function automatically allows the user to specify a file for
% saving the video into.
```

3.4. Auxiliary Functions

PhysTrack contains numerous functions and scripts which may not be of interest in video tracking process. Still, there are a couple of functions which can be sued to minimize amount of work and effort required for the whole process.

3.4.1. cascade

This function arranges all the open figures in a uniform manner, making it very easy to view all the data at once.

3.4.2. StructToArr(struct), ArrToStr(array)

The trajectories returned by the automatic trackers are in form of structs. Most of the functions included in PhysTrack can actually work very well with these structs but still, some users like to have their data in arrays. These two simple functions can easily perform this conversion between structs to arrays.

3.4.3. StructOp(struct, operand, operator)

Matlab doesn't allow to operate a scalar quantity directly with a struct. However, PhysTrack contains a workaround to operate structs with simple mathematical operators like "*", "/", "+", "-", ">" etc.

```
twoTimesTheTrajectory = StructOp(trajectory, 2, '*');
ValueIsGreaterThan4 = StructOp(trajectory, 4, '>');
```

3.4.4. GetColor(RequiredColor, GetName, GetInverseColor)

In plotting, we often need a color, the RGB values of which are unknown. For example, we need to display a line in Rosy Brown, we can use the following code to get the RGB values of Rosy Brown

```
rgb = PhysTrack.GetColor('Rosybrown');
ColorName = PhysTrack.GetColor([176, 23, 31], true);
% this will return 'Indian Red'
```

4. GUIs and their associated scripts

To make the user experience more interactive, PhysTrack contains a lot of GUI's (Graphical User Interfaces). Unlike some dedicated GUI creation platforms like Microsoft .Net and Java, Matlab GUI's can be a little challenging when it comes to keeping a modular software design. To hide the complexities of coding from a physics experimenter, the PhysTrack package doesn't contain any GUI. Instead, there are functions included in the main package which create and remove all required global workspace variables before and after calling the main GUI functions. Following are some important GUIs and their usage with and without the helper function is explained.

4.1. Video Trimming and Cropping Tool

This tool can let the user graphically determine a starting and ending frame and a region of interest.

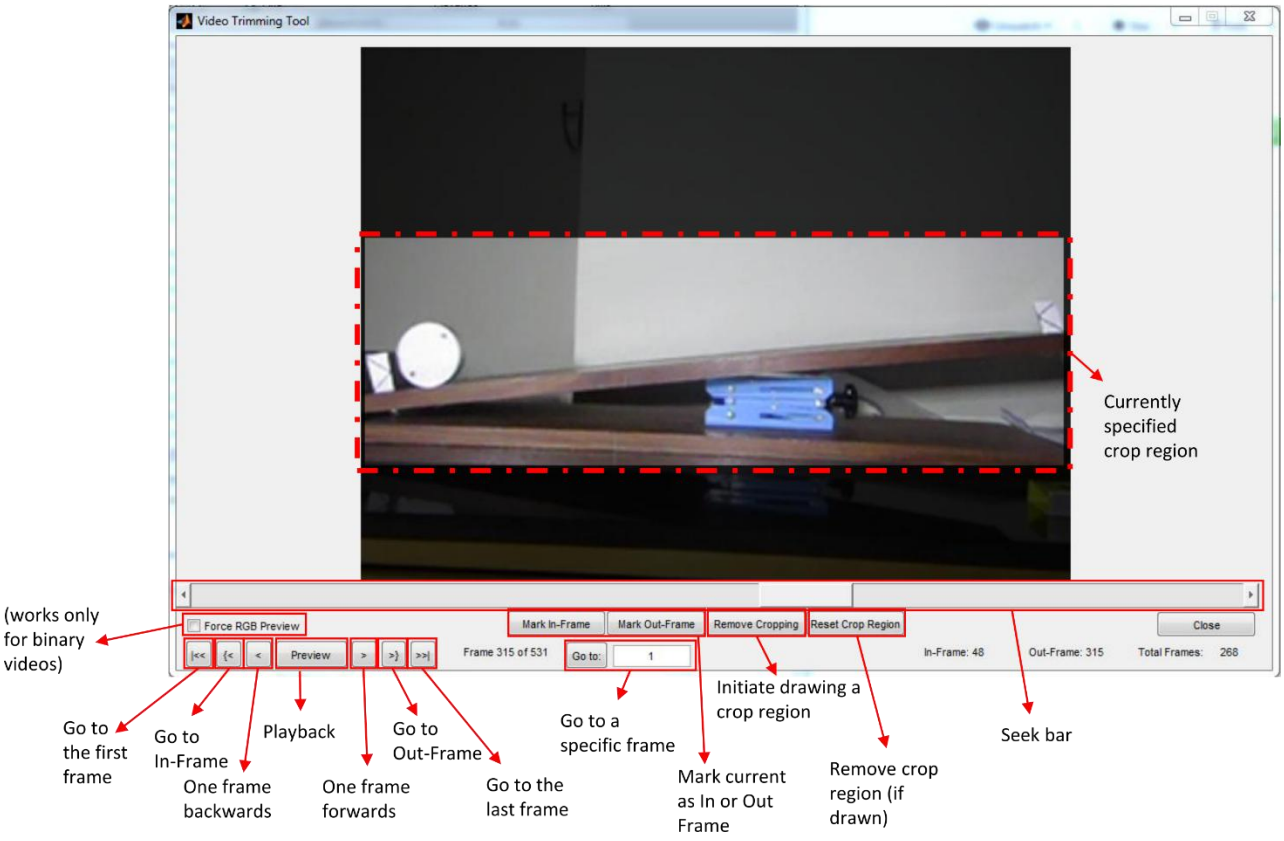
This GUI assumes to see a specific variable in the global workspace, namely vtt_vr2o_00. For a more useful usage, the GUI shouldn't be called directly, instead, TrimVideo should be used.

```
% vro = PhysTrack.VideoReader; % define a video reader 2 object
% first
vroTrimmed = PhysTrack.TrimVideo;
```

The GUI contains a lot of useful tools which can be used in different scenarios. The seek bar is used to quickly walk through the whole video. The control buttons are used to precisely and carefully move to specific frames to mark the in/out frames.

Kindly note that when working with compressed videos, the video trimming tool may take a very long time to scroll through different parts of the video. This problem and a possible resolve is suggested in the VideoDecoder section.

Video Trimming and Cropping Tool



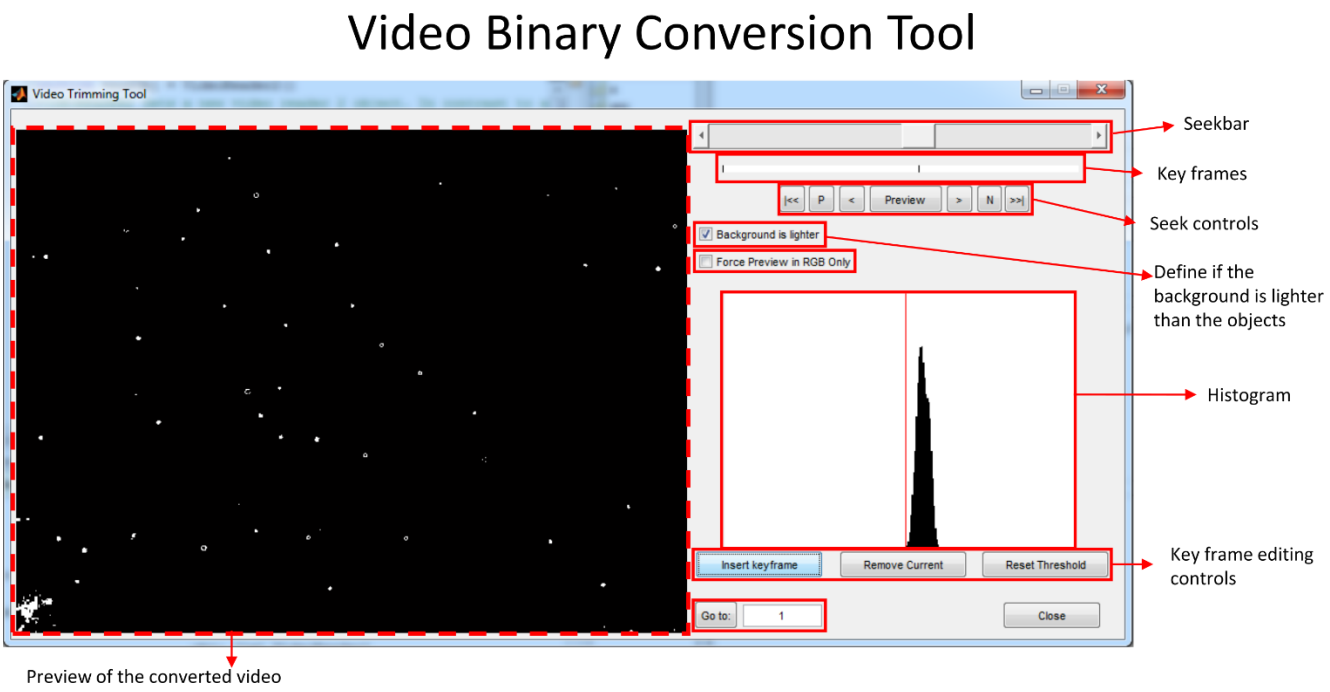
4.2. Video Binary Conversion Tool

A binary image is an image in which the pixels are either true or false, meaning, either they are white or black. An RGB can be converted to binary using many methods one of them being grayscale thersholding. What this technique does is that it converts the image to grayscale first, and then operate each pixel with a binary "Greater than" operator. The value used with this operator is called the binary threshold. As a result, the algorithm traverses through all the pixels, comparing their grayscale level to the threshold value. If the threshold is smaller or equal, the pixel is converted to black or to white if it is greater. This can also be done with inverted color output.

Now, for video tracking, we usually need to convert the RGB frames to binary in such a way that the desired moving objects are distinguished clearly from the background in the final image. i.e. if the grayscale level of object clearly remains different than the background's extreme levels, after thresholding, both parts will be converted to opposite colors, making the objects detectable by the

machine. This way, the algorithm can track the position of the object just by estimating its new position in coming frames.

For each image, defining an accurate threshold can be a challenging job without a proper tool. One very useful tool to determine threshold is the image histogram. A histogram is an intensity-frequency plot of the image. It contains intensity levels from 0 to maximum possible (255 in an uint8 image) on the horizontal axis and the number of pixels sharing that level from 0 to a scaled maximum on the vertical axis. I the following screen shot, tiny dark particles can be seen floating on a uniform background. A sharp peek can be seen in the histogram indicating that most of the pixels of the image share a similar grayscale level which is the background color. The peek for the particles is so small that it cannot be seen in comparison to the background's peek. But, it is very obvious to mark a threshold just before (or after, in some cases) the peek.



Now, for a video, sometimes, a single threshold isn't suitable for the whole video. So, we need to specify a trend in the change of this threshold. The video binary conversion tool provides the facility of insert "Key-Frames" on any frame of the video. When the user inserts a key frame at a specific index, the GUI associates the currently previewing threshold with that index. The user can insert as many key frames as required. If no, or a single key frame is inserted, the same threshold is used for the whole video. When more than 1 thresholds are specified, PhysTrack will create a polynomial function with the maximum possible degree and fit the indices-threshold data to that function, thus interpolating a value for any frame that comes in between these frames.

To handle these key frames, the GUI gives some useful controls. Seek bar can be used to seek different frames of the video. The inserted key frames are previewed beneath the seek bar in a horizontal bar as seen in the above screenshot. Seek controls contain different buttons to scroll through specific frames of the video.

- |<<, >>| Start/end of video
- P, N Previous/Next Key Frame
- <, > Previous/Next Frame
- Preview, play through all frames

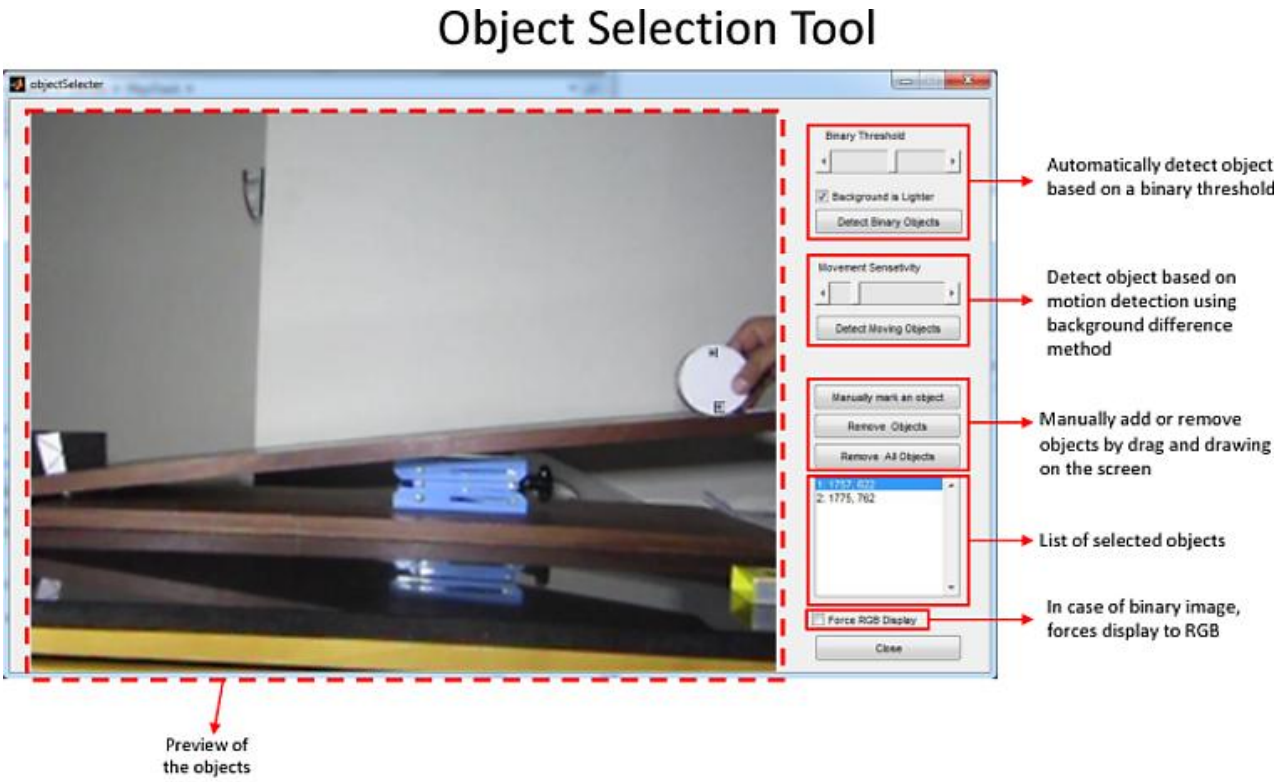
Pressing the "Insert keyframe" button inserts the currently set threshold level to the current frame of the video. An intensity-frequency plot of all the image. The vertical red line is the threshold. Similarly, "remove current" will remove any keyframe present on the current index. "Reset Threshold lets the user click on the histogram to visually set a threshold level.

Following Code should be used to convert a video into binary:

```
% vro = PhysTrack.VideoReader; % create a video reader object if
% absent
% vroBinary = PhysTrack.ConvertVideoToBinary(vro);
```

4.3. Object Selection Tool

Both of the automated object trackers, the "BOT" and the "KLT", require a collection of objects in the very first frame of the video which they track through the reset of the video. To obtain the objects graphically, the object selection GUI is great tool.



Using the controls shown in the above screenshot, the user can manually identify the objects by drawing bounding boxes around them, remove them and modify them. It is also possible to try to automatically detect objects based on a binary threshold or movement detection. Both of the later method are not absolute and may not work in all the scenarios. Mostly, it is is desirable to manually identify the objects.

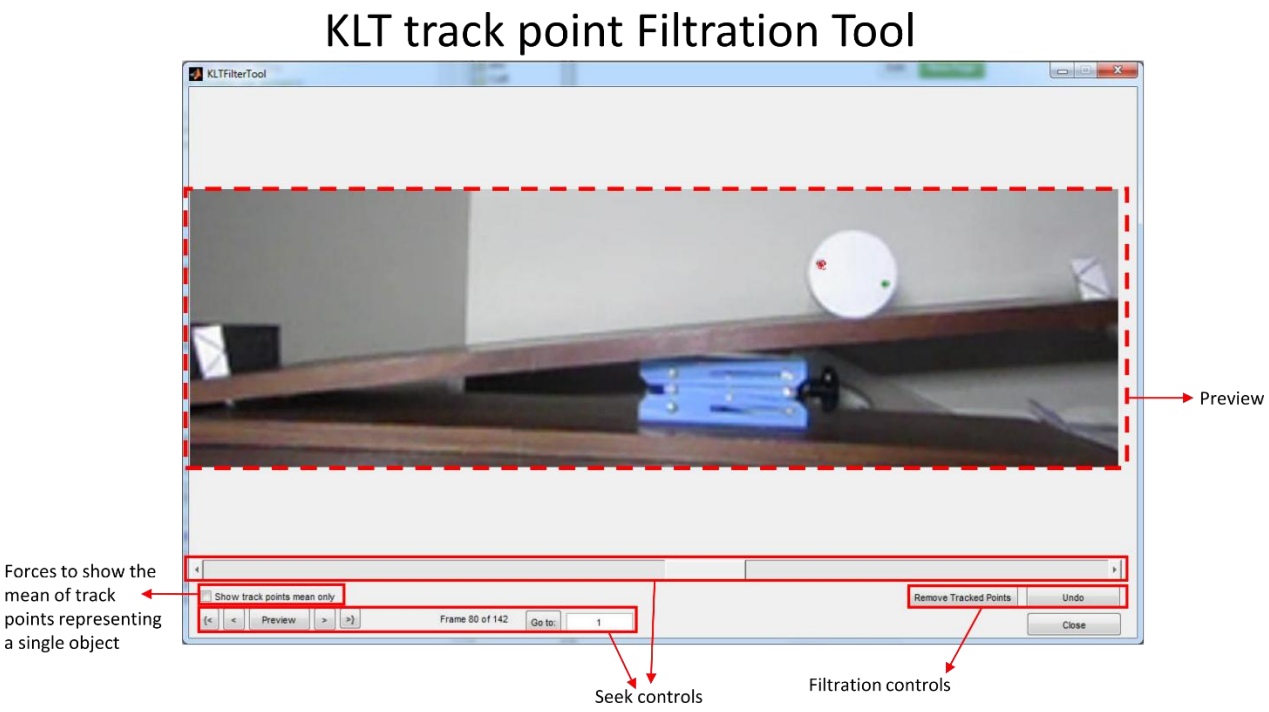
Following routine can be used to get a collection of objects in an RGB video

```
% Crop and trim the video to desired paramters.
vro = PhysTrack.VideoReaderObject;
obs = PhysTrack.GetObjects(vro);
% and manually draw tight bounding boxes around the object
% pressing the "manually add objects" button each time
```

Following routine can be used to get a collection of objects in a binary video

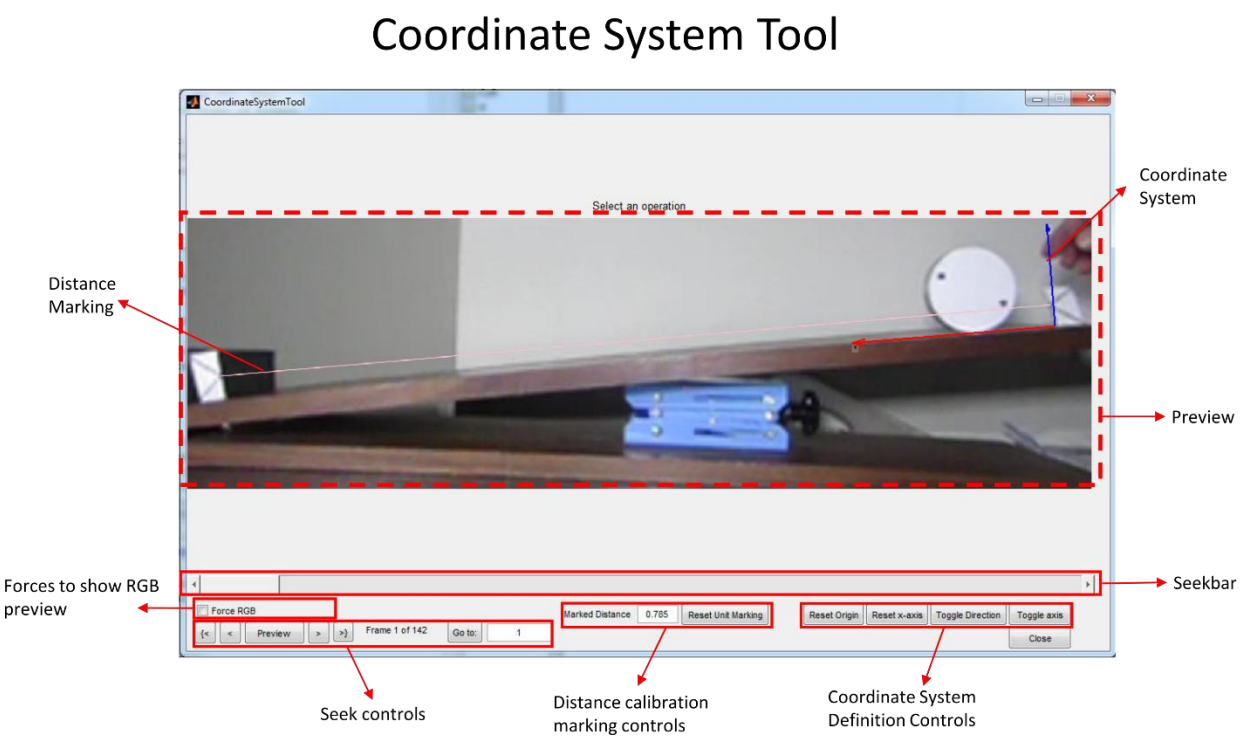
```
% Crop, trim threshold the video to desired paramters.
vro = PhysTrack.VideoReaderObject;
obs = PhysTrack.GetObjects(vro);
% Press the "detect binary objects" button to automatically detect
% all the objects based on the threshold of the first frame of the
% binary video.
```

4.4. KLT Track Point Filtration Tool



This tool is presented by PhysTrack.KLT which assumes that the user will manually identify wrong or detracked track points in the trajectory of the objects. User can seek through the whole video and see which track point lost the way. Drawing a rectangle around the points using "Remove Tracked Points" button will remove all of them. This process can be undone by as many steps backwards as necessary.

4.5. Coordinate System Tool



This tool can be used to create a coordinate system. The user can define a distance calibration constant. As shown in above screenshot, using the coordinate system, controls, user can place the coordinate system at a desired place and orientation. Each operation is performed after pressing the respective button on the screen. Following Codes can be used to define different kinds of coordinate systems

```
% create a video reader object if not already present
% vro = PhysTrack.VideoReader2;
[rwRCS1, ppm] = PhysTrack.DrawCoordinateSystem(vro);
```

```
% to obtain a floating coordinate system, we first need to track
% objects. then we will stitch our system to those objects.
% graphically select up to two objects to be tracked
objs = PhysTrack.GetObjects(vro);
trajectories = PhysTrack.KLT(vro, objs);
rwRCS2 = PhysTrack.DrawFloatingCoordinateSystemNonRotating(vro,
trajectories.tp1, rwRCS);
rwRCS3 = PhysTrack.DrawFloatingCoordinateSystem(vro,
trajectories.tp1, trajectories.tp2, rwRCS);
```

5. References

1 J. Poonyawatpornkul and P. Wattanakasiwich, *"High-speed video analysis of a rolling disc in three dimensions"*, Eur. J. Phys. 36 065027 (2015).

2 J. Poonyawatpornkul and P. Wattanakasiwich, *"High-speed video analysis of damped harmonic motion"*, Eur. J. Phys. 48 6 (2013).

3 Loo Kang Wee, Charles Chew, Giam Hwee Goh, Samuel Tan and Tat Leong Lee, *"Using Tracker as a pedagogical tool for understanding projectile motion"* Eur. J. Phys. 47 448 (2012).

4 Paul Nakroshis, Matthew Amoroso, Jason Legere and Christian Smith, *"Measuring Boltzmann's constant using video microscopy of* Brownian motion", Am. J. Phys, 71, 568 (2003).

5 *"Tracking kinematics of a fruitfly using video analysis"*, (http://goo.gl/ljypdC)

6. Appendix

[1] VideoReader2

For reading video files, Matlab provides two main video reading systems. One is the *"VideoReader"*, included in *Image Acquisition Toolbox* and the other one is *"VideoFileReader"* system, included in the *Computer Vision Toolbox*. The former is used with function *"read(VideoReaderObject, FrameIndex)"* and the later is a system, thus is used with *"step"* function.

Both of these methods are not easy to be used in video tracking. In video tracking, we usually need to trim the ends of a video file and crop a specific region of interest out of each frame so that we only process the data which is significant. Additionally, we also sometimes need to resize the whole frame to a suitable resolution for best results. These three simple steps introduce a lot of complexity in handling the video file because to for reading every frame, we would need to keep account of "in-frame index", "out-frame index", "crop rectangle" and "resizing factor". To encapsulate this process, we have created a *"VideoReader2"* object which combines a simple VideoReader with cropping, trimming and resizing information. With this object, we give a "read2" function for reading the video.

VideoReader2 object can be created using the following code:

```
vro = VideoReader2;
% to modify an existing video reader,
% vro = VideoReader2(vro)
```

[2] Trimming

In video tracking, we often use a high speed camera for capturing video. High speed implies that only a couple of seconds in real life will create hundreds of frames in the video file. So, just before and after the actual object motion, it becomes inevitable for the expreimenter to avoid the extra frames. We don't want to process those frames because it would impart a lot of load on the processor and more importantly, because in most cases, the data we need to analyze won't be useful if those extra frames are also included. So, in the VideoReader2, there are 3 variables called ifi, ofi and TotalNumberOfFrames. ifi is the In-Frame, at which the video actually should have started, Out-Frame is the last desirable frame and TotalNumberofFrames just gives the number of frames included in the trimmed section. So now after trimming, ifi will serve as the first frame of the whole video.

[3] Cropping

In video processing, to avoid processing extra pixels of a video frame, we often desire to define a region of interest. To incorporate this cropping, there is a CropRect variable in the video reader 2 which contains the cropping rectangle's coordinates and dimensions. So, now, the (1,1) pixel of a video frame is actually the top left pixel of the crop rectangle instead.

[4] Pre-magnification

Matlab imresize function not just increases the number of pixels in an image, it also interpolates to create new information. This process actually enhances the image quality artificially. To maintain a balance between quality and the memory requirements to handle big images, there is this variable PreMag, in video reader 2 objects which magnify the finally cropped frames.

[5] BinrayThreshold

To convert a grayscale to binary logical image, the practice is to define a gray level above or below which pixels are converted to white or black. Usually, the same threshold is used for the whole video but in PhysTrack, we can assign different thresholds for different frames of the video. In this way, the system interpolates in between these KeyFrames to obtain another threshold. This is useful for processing those videos where lighting conditions don't remain the same throughout the video.

Normally, this variable is empty but if a Nx2 array is defined, PhysTrack will consider (:,1) entries to be the frame numbers and (:, 2) there respective threshold. For interpolation, PhysTrack fits the points on polynomial functions of maximum possible degrees and fits the threshold into it.

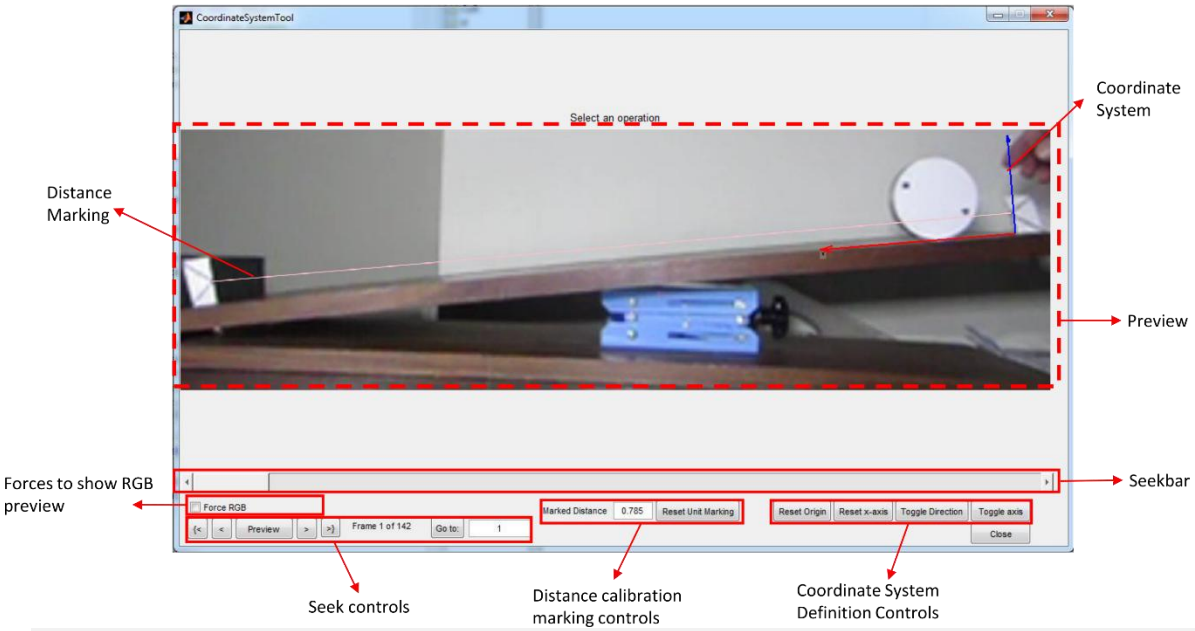
[6] Cooedinate System

This tool can be used to create a coordinate system. The user can define a distance calibration constant. As shown in above screenshot, using the coordinate system, controls, user can place the coordinate system at a desired place and orientation. Each operation is performed after pressing the respective button on the screen. Following Codes can be used to define different kinds of coordinate systems

```
% create a video reader object if not already present
% vro = PhysTrack.VideoReader2;
[rwRCS1, ppm] = PhysTrack.DrawCoordinateSystem(vro);
```

```
% to obtain a floating coordinate system, we first need to track
% objects. then we will stitch our system to those objects.
```

Coordinate System Tool




```
% graphically select up to two objects to be tracked
objs = PhysTrack.GetObjects(vro);
trajectories = PhysTrack.KLT(vro, objs);
rwRCS2 = PhysTrack.DrawFloatingCoordinateSystemNonRotating(vro,
trajectories.tp1, rwRCS);
rwRCS3 = PhysTrack.DrawFloatingCoordinateSystem(vro,
trajectories.tp1, trajectories.tp2, rwRCS);
```

[7] Experimental Scripts

Experiment title	Student manual and resources	Sample analysis codes	Sample videos
Spring pendulum	https://goo.gl/Mk08IH	https://goo.gl/RQ9vxr	https://goo.gl/vqegU0
2d collisions	https://goo.gl/oHyE1F	https://goo.gl/fZV9Dr	https://goo.gl/WABNhi
Projectile motion	https://goo.gl/X5LmXA	https://goo.gl/SMtL5L	https://goo.gl/Lq9bqc
Sliding friction	https://goo.gl/4SYX7X	https://goo.gl/v9E91R	https://goo.gl/IHNM7L
Rotation on a fixed pivot	https://goo.gl/w4aZYy	https://goo.gl/v36veO	https://goo.gl/Z4Or3N
Brownian motion	https://goo.gl/VmTHWF	https://goo.gl/XabN5L	https://goo.gl/pBGOwy
Rotational friction	--	https://goo.gl/0ouAMv	https://goo.gl/xRFWDI