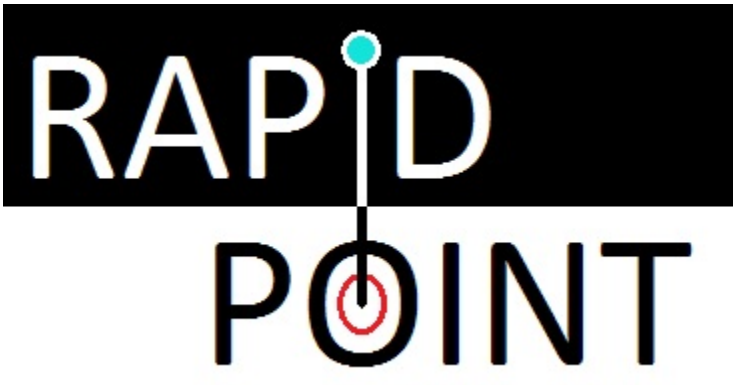
Design Packet

Kinematic Mapping Team

May 26, 2017



Contents

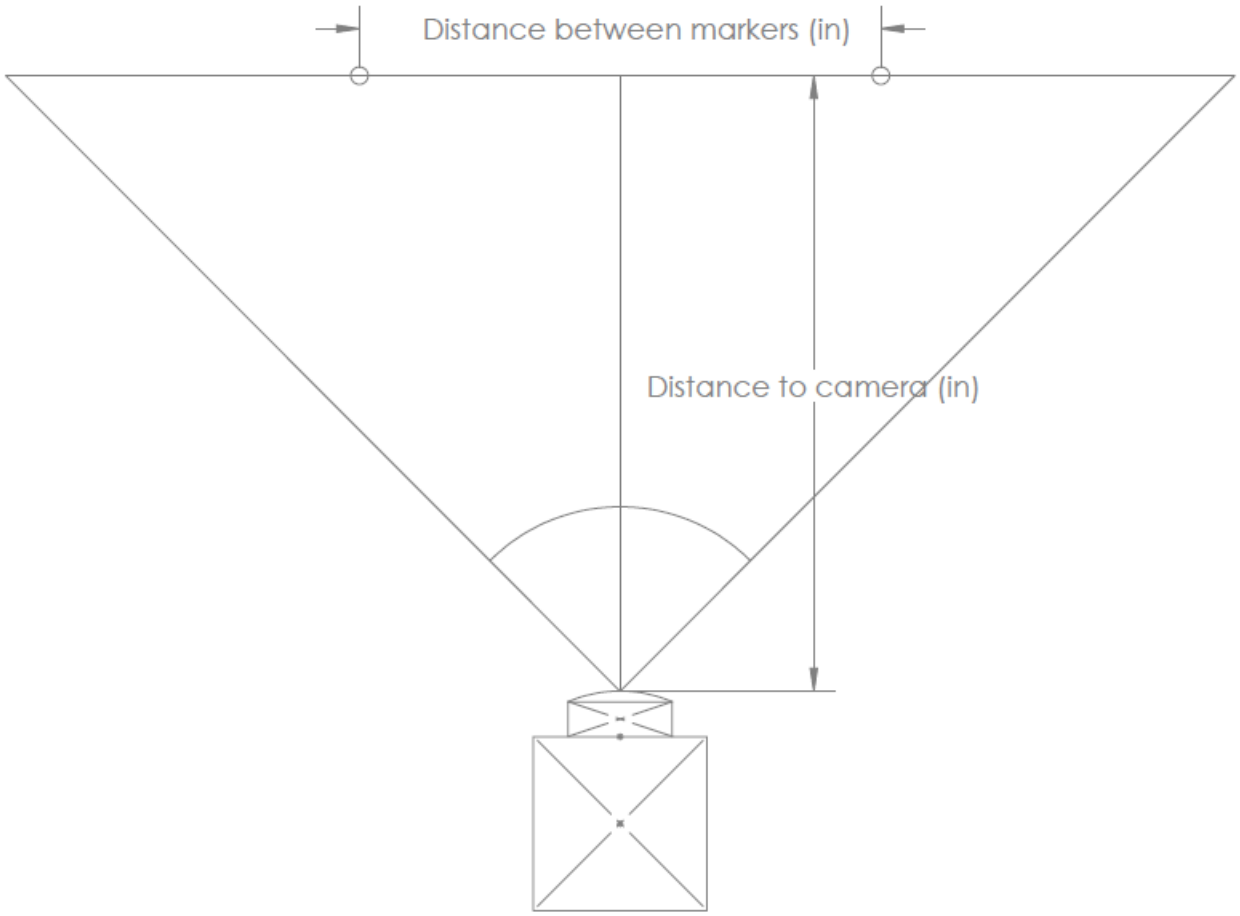
1 Product Description

The goal of this project is to design and construct a measurement system that will allow Orbital ATK to evaluate the performance of its deployable array systems by tracking their motion. Currently, this is done by visual observation only. We propose to automate this process using computer automated tracking of marked points on the array structure, using a video feed provided by a conventional visible light camera.

2 Specifications

- Positional accuracy good to ± 0.1in for arrays up to 30 ft in length.
- Adjustable to different array sizes, including 1m-10m.
- Ability to track up to 10 markers simultaneously.
- Ability to track up to 120 frames per second.

3 Assembly Drawing



4 Block Diagram

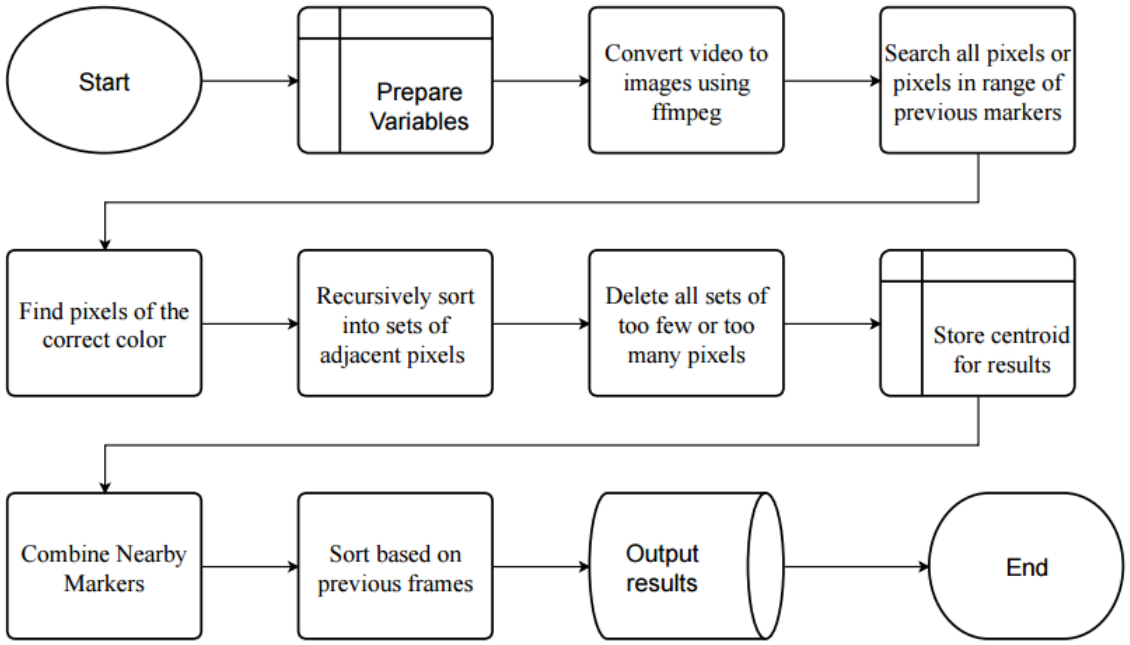
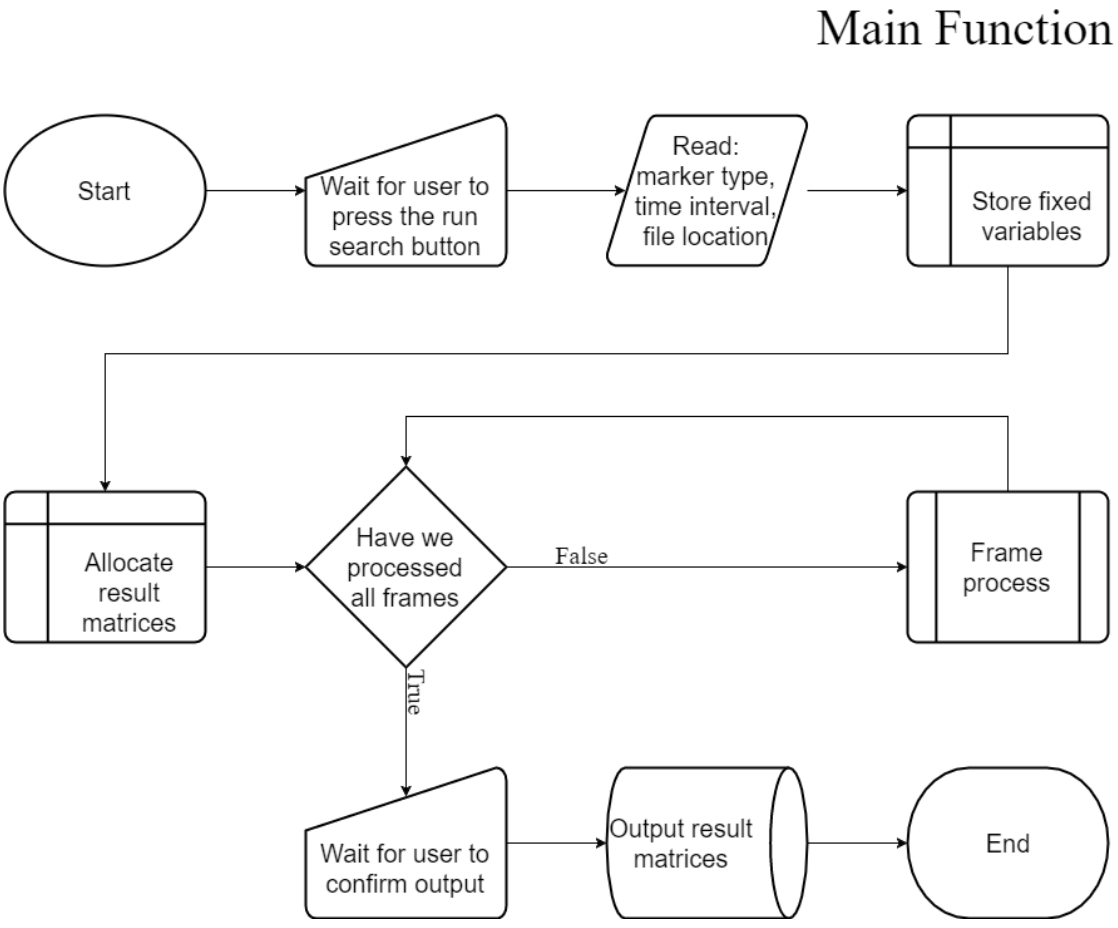
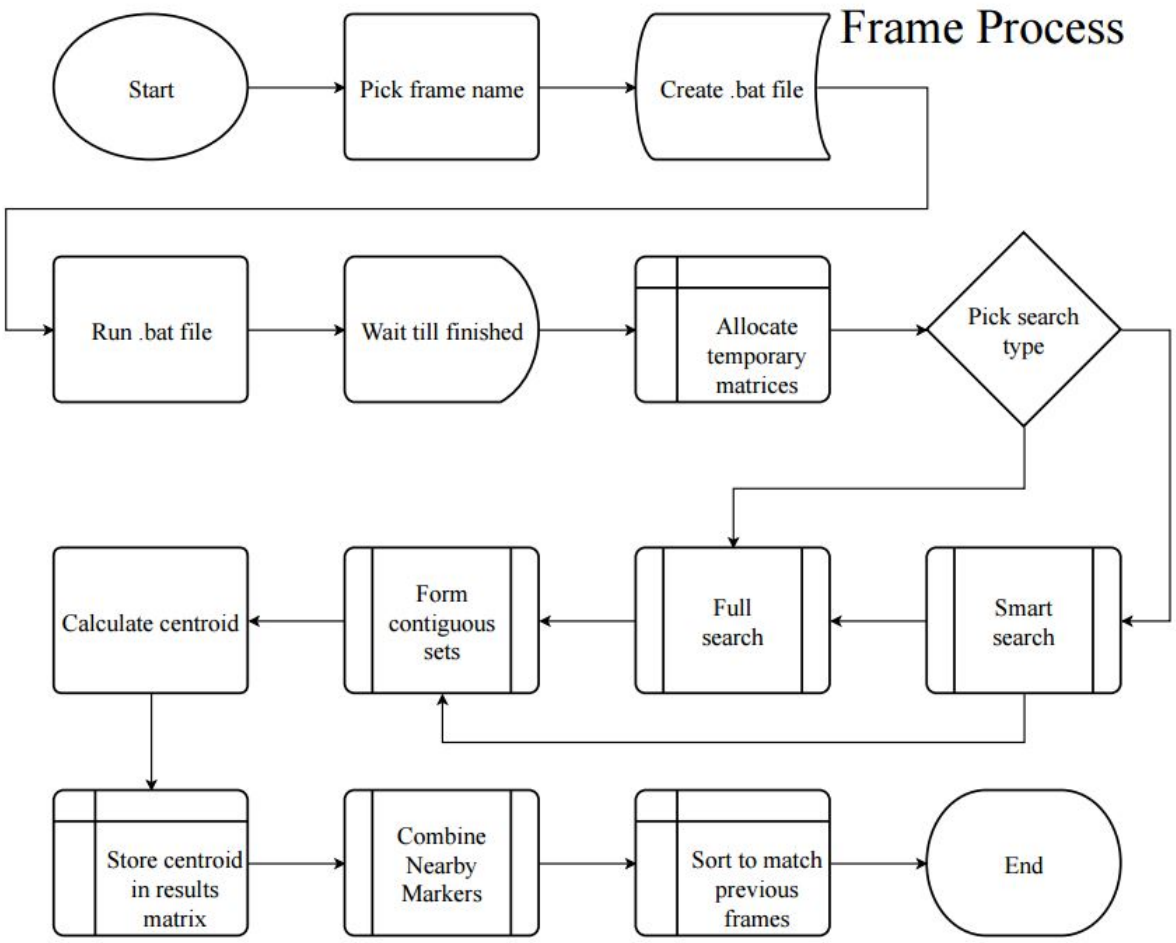
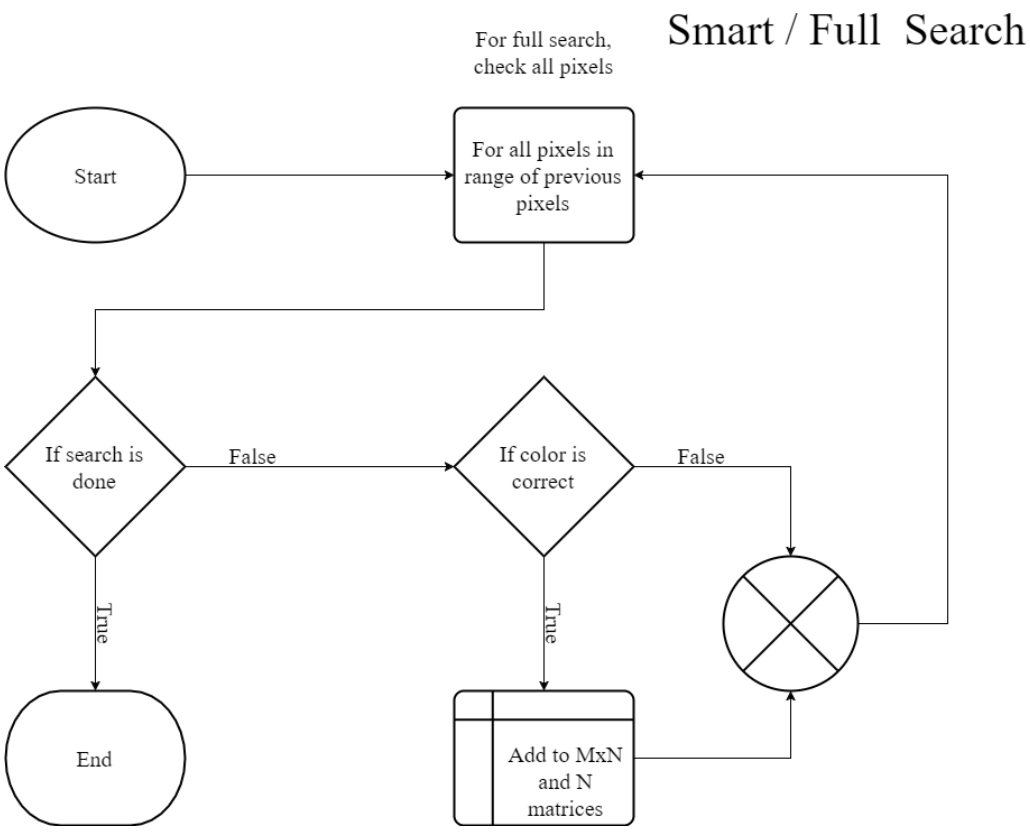
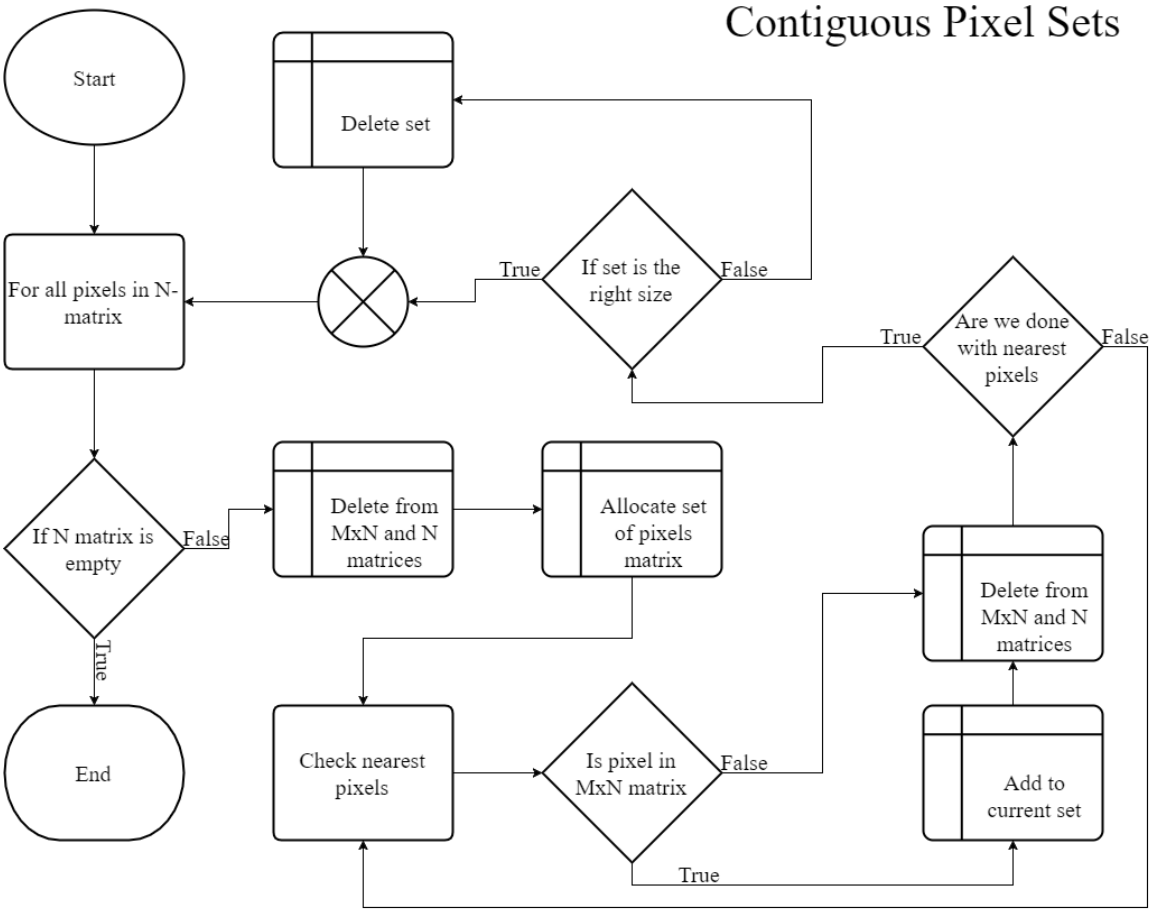


Figure 1: General Program Block Diagram




Frame Process
Start
Pick frame name
Create .bat file
Run .bat file
Wait till finished
Allocate temporary matrices
Pick search type
Calculate centroid
Form contiguous sets
Full search
Smart search
Store centroid in results matrix
Combine Nearby Markers
Sort to match previous frames
End


Smart / Full Search
For full search, check all pixels
Start
For all pixels in range of previous pixels
If search is done
False
If color is correct
False
True
True
End
Add to MxN and N matrices



4.1 Block Diagram Description

Our program processes the video produced by a camera to find and track specific marked points on the structure being filmed.

When the program starts, it waits for the user to confirm the data is entered via a button. Once the user presses the button, it reads the inputs and stores them as variables which direct how the search will be done. Next, it allocates space for a result matrix. It then checks every frame using a process which I will soon explain. Following this it waits for the user to confirm to output the resulting comma separated value file(.csv). Finally, it outputs the result matrix to a file, which is accessible to the user.

The first part of the frame processing step is to pick a frame name. It will then create a windows batch (.bat) file. This batch file uses the users settings, ie: start time, end time, and frame rate of the video, to instruct an external program, ffmpeg, to extract a specific frame from the video. Our program waits until this program outputs a bitmap image (.bmp) file. It then initializes the N matrix which will be a list of the markers with the correct color, and the MxN matrix, which will store whether the pixels have the correct color. There are two ways the program can scan the image, a full search, which searches every pixel in the frame, and a smart search, which starts searching in the vicinity of where the markers were found on the previous frame. The program must use the full search on the first frame, when no previous frames have been searched, but it also does the full search periodically to ensure that, if a marker is blocked from view and later reappears somewhere else, it will still be found. The program now groups the pixels together into contiguous sets. It calculates the centroid of these sets, and stores the centroid location into the result matrix.

For the full search, the program checks every pixel in the frame to see if their color parameters

match those the user has set for the markers that the user wishes to track. Each matching pixel is added to the MxN matrix and N matrix. For the smart search, the program checks every pixel in a certain square centered around the marker points from the previous frame.

To form the sets of contiguous marker pixels, the program looks through each pixel in the N matrix; it is then deleted from the MxN and N matrices. Following this, the pixel is added to a new pixel set. The program then begins checking the pixels nearest to this pixel, to see if they are in the MxN matrix. If they are, they are added to the set and deleted from the MxN matrix and, if relevant, N matrix. The program continues doing this until there are no more adjacent pixels to check. If the set is then too small or too large, the program deletes the set.

5 Marker

Red active markers are recommended to be with the RapidPoint program. Figure 2 shows the basic components of the provided red active markers.

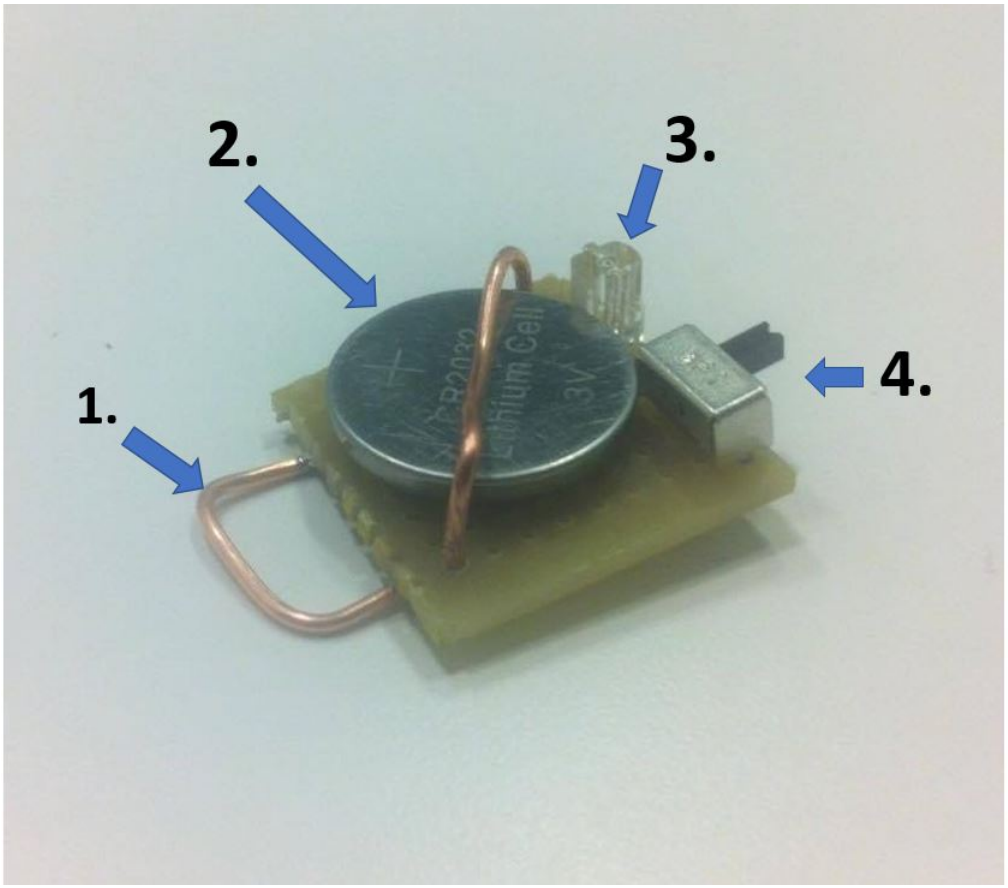


Figure 2: Red Active Marker

1. Attachment Ring
2. CR2032 Battery
3. On/Off Switch
4. Red LED

5.1 Circuit Diagram

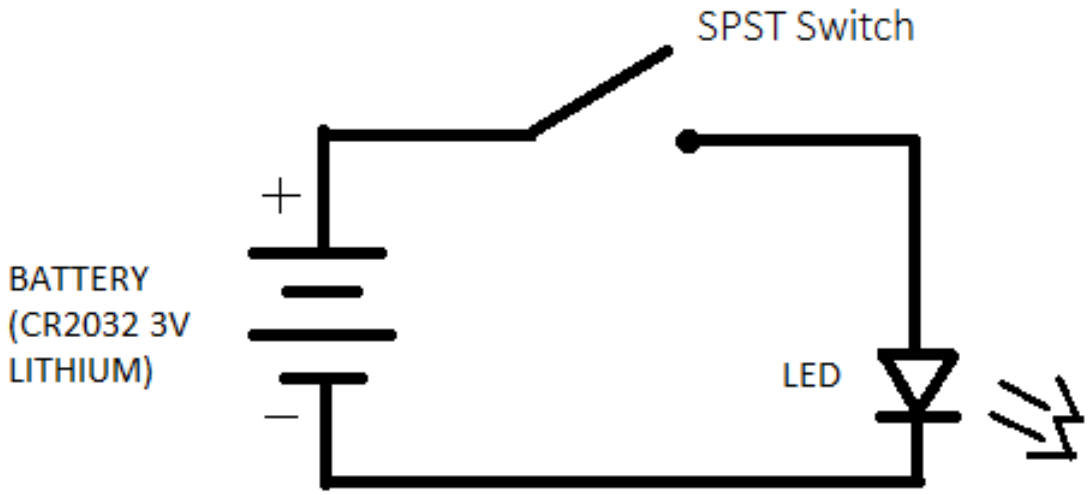


Figure 3: Active Marker Circuit

6 Bill of Materials

Part Number	Description	Unit Cost ($)	# of parts
1	Active Marker Circuitry	5	10
2	Passive Markers	0.05	10
3	Camera	250	1
4	Tripod	70	1
5	Calibration Frame	112	1

7 Budget

Item	QTY	$ / Unit	Total $
Active markers	10	5	50
Passive markers	10	0.05	0.5
Camera	1	250	250
Tripod	1	30	30
Calibration Frame	1	112	112
Total Budget			442.5

8 User Manual

8.1 Introduction

This project is designed to assist Orbital ATK in evaluating the performance of its deployable structures by tracking their motion. This is done by recording the motion of markers attached to the structures using a conventional visible light camera, and then using software to output the positions of the markers.

This software system isolates the potential markers using color, and refines that output with size elimination. This allows the program to be effective for a variety of sizes and geometries.

8.2 Quickstart

1. **Check camera** settings. (see section 3)
2. **Set up** the markers. (see section 4)
3. **Record** the experiment.
4. **Input calibration** settings. (see sections 5 and 6)
5. **Confirm output** with small time interval. (see sections 4 and 7)
6. **Run program** for the full experiment. (see sections 4 and 7)
7. **Analyze results** from the experiment. (see section 7)

8.3 Camera Settings

There are a few main things to consider when configuring a video camera to be used with this system.

- Frame Rate
- Shutter Speed
- Frame Resolution
- Interlacing
- Bit-Rate
- Focus
- White Balance
- Special Effects

The **Frame Rate** dictates the resolution (not the uncertainty) of the motion tracking across time. The higher the frame rate is, however, the higher the band-width requirement will be. While the program was designed to evaluate with 60 frames per second in mind, any lower frame rate will be processed faster while yielding a lower resolution data, and higher frame rates will be processed slower while yielding higher resolution data.

Figure 4: Image taken with different shutter speeds.

Figure 5: Different white balance setting on the Sony CX405.

The **Shutter Speed**, which is also called Exposure Time, dictates part of the uncertainty in motion tracking across time. A shorter exposure time helps deal with motion blur, which would otherwise effectively increases uncertainty and may eliminate data entirely. However, shorter shutter speeds may alter the colors detected and make tracking passive markers more difficult. This color alteration is most notably shown in Figure 4. A decent starting point for 60 frames per second would be a shutter speed of $\frac{1}{100}$ s, however shutter speeds as short as $\frac{1}{1000}$ s are very reasonable to work with.

Frame Resolution dictates the resolution and part of the uncertainty in motion tracking for all points in time, both statically and dynamically. Greater resolution will, however, take longer to process and take up more band-width.

Interlacing should always be turned off. This has a severe effect on tracking any object undergoing motion; instead a progressive scan video mode should be used. (i.e. 30p or 60p instead of 60i)

Bit-Rate or the quality of the video may affect the ease of tracking points, and because of this, it may have a slight effect on the uncertainty. Although this will increase the output file size, it is ideal to set this to maximum to ensure all of the data can be processed.

Focus or aperture should be set to obtain the best image possible where the marker is. Most importantly this should not be set to some automatic setting which might change during the test, as this could interfere with the data.

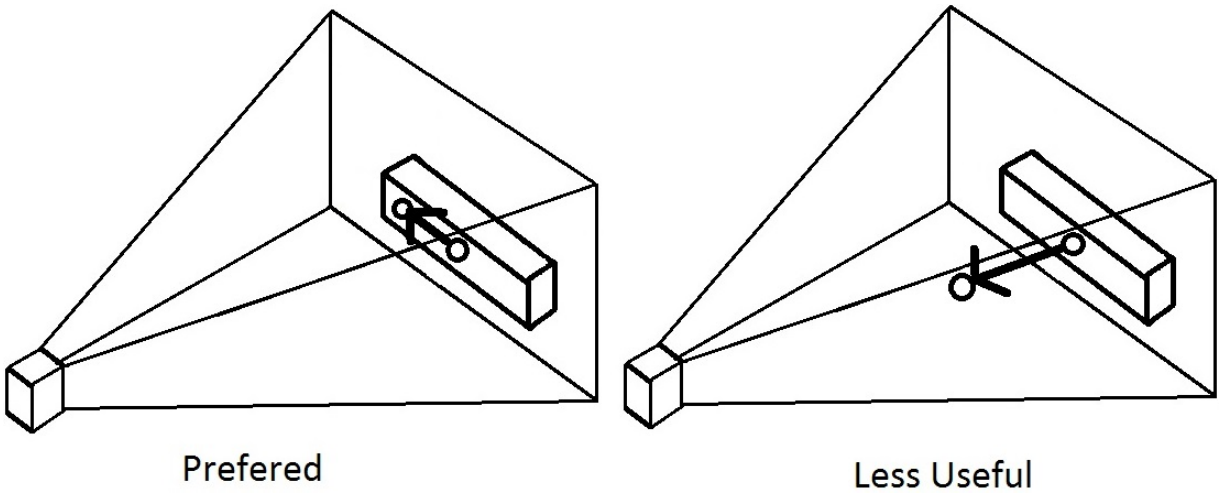


Figure 6: Single Camera Setup

White Balance preset should be set to "indoor", or similar, for maximum marker tracking capability. While outdoor should work, the color settings must be adapted, and isolating markers can be more difficult with the outdoor setting. These effects can be seen in Figure 5.

Any **Special Effects,** such as black light compensation, should be turned off or to default. These would harm the ability to track the markers, and could harm accuracy.

8.4 Marker Setup

The video camera should be placed at a distance that fully captures the desired motion of the marker throughout the experiment. **Record this distance** from the markers to the camera in inches.

Make sure the camera is orthogonal to the plane of motion of the measurement target if only using one camera. If possible, raise or lower the camera, as opposed to slanting it, as this will affect the accuracy of the results. A diagram of the assembly is shown in Figure 6.

You are now ready to record the experiment.

8.5 User Interface

Finding the Marker

Click on Setup.exe in the Capstone folder, and confirm any windows that pop up to install the program. Once installed, click on either Capstone.application or setup.exe in order to run the program and bring up the user interface (UI). The main elements of the program are shown in Figure 7.

1. **Version Number**
    - Check to make sure this is the latest version number.
    - Make sure to update this number manually for each update to the code.
2. **Data Folder location**
    - This should be the entire data folder address.
    - This includes a final "\" at the end, as shown in the image.

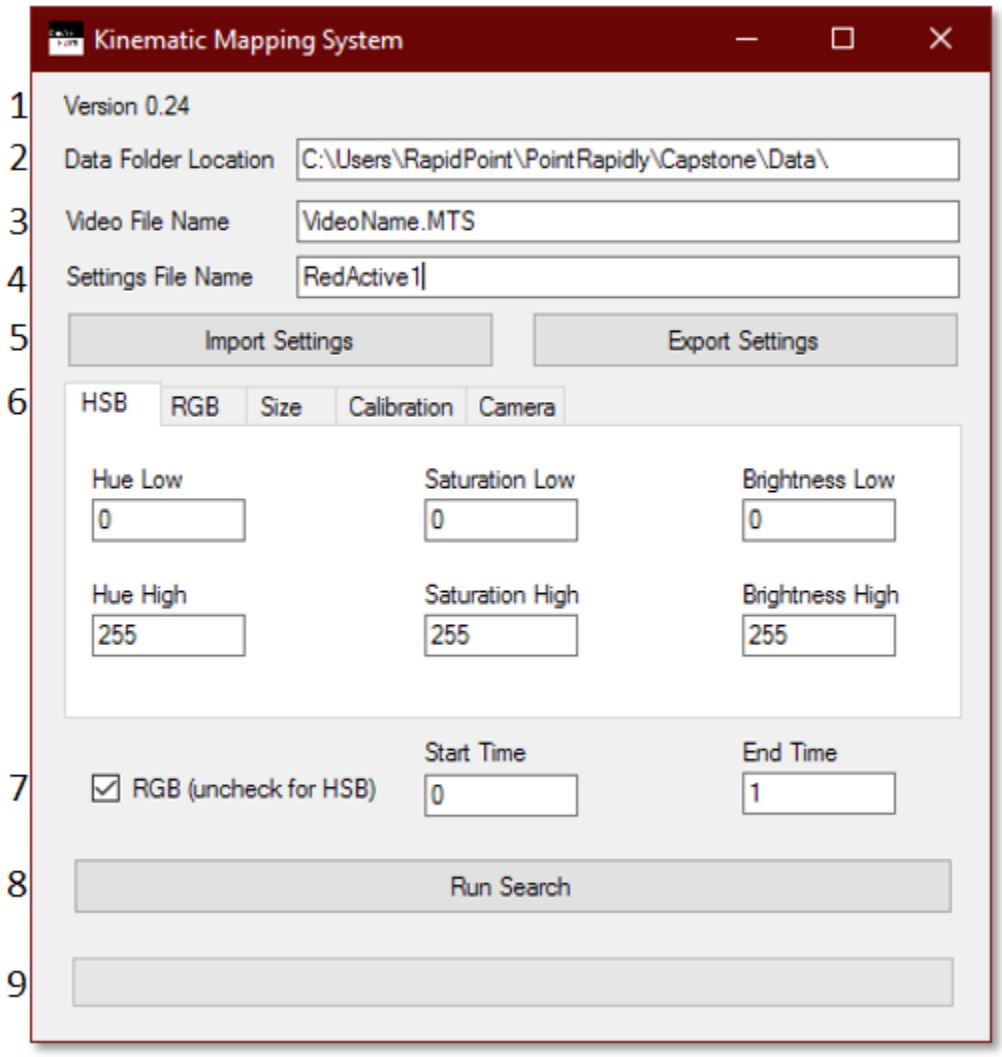


Figure 7: User Interface

3. **Video File Name**
   - Make sure the file is located within the Data folder.
   - Make sure to include the extension.
4. **Setting File Name**
   - If importing an old setting, use the appropriate settings file name.
   - If making a new setting, make sure your name is not an existing settings file name. Overwriting may not work as intended. If you need to overwrite, delete the old file first.
   - Make sure you do **not** include the extension in this name.
   - The settings file will be created with a .xml extension in the Data folder.
5. **Import Settings** and **Export Settings**
   - Import Settings will import the settings file specified above to the text-boxes, plus a .xml extension.

- Export Settings will export the settings from the text-boxes to the file specified above, plus a .xml extension.
- If no Settings File Name is specified, Import Settings will import the default values to the text-boxes.

6. Inputs Tab

- **HSB** tab

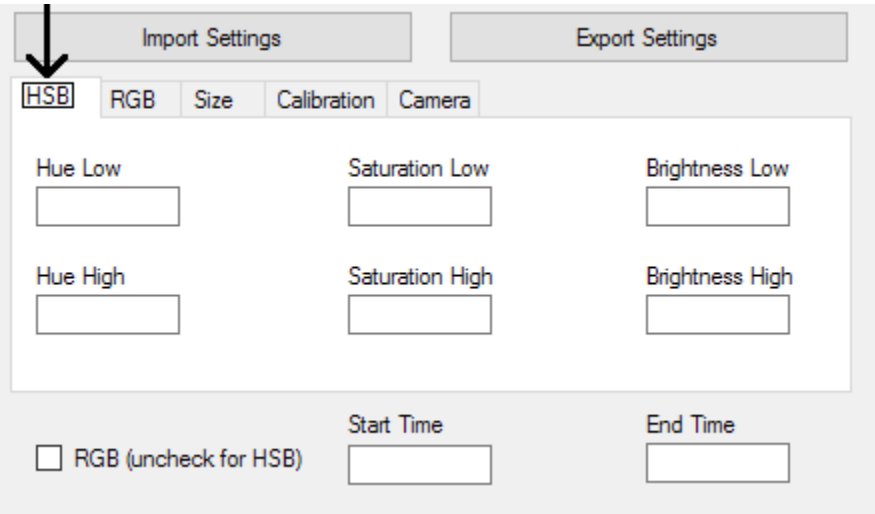


Figure 8: HSB tab

  - This tab is shown in Figure 8.
  - HSB stands for Hue-Saturation-Brightness, which is a color scheme for distinguishing the markers from the background.
  - These are integer values that range from 0 to 255. The bounds are inclusive. Thus, at a low and high of 250, the marker can have a value of 250, but no other value.
  - **Hue Low** is the minimum range of hue for a pixel to be a marker pixel.
  - **Saturation Low** is the minimum for the color range of saturation for a pixel to be a marker pixel.
  - **Brightness Low** is the minimum for the color range of brightness for a pixel to be a marker pixel.
  - **Hue High** is the maximum for the color range of hue for a pixel to be a marker pixel.
  - **Saturation High** is the maximum for the color range of saturation for a pixel to be a marker pixel.
  - **Brightness High** is the maximum for the color range of brightness for a pixel to be a marker pixel.
  - If any of these values are invalid, or left blank, they will be replaced by 0 (for the low values) or 255 (for high values).
  - Brightness, Value, or Lightness may be defined differently for different programs. Be cautious when using other programs to help calibrate this value.
- **RGB** Tab

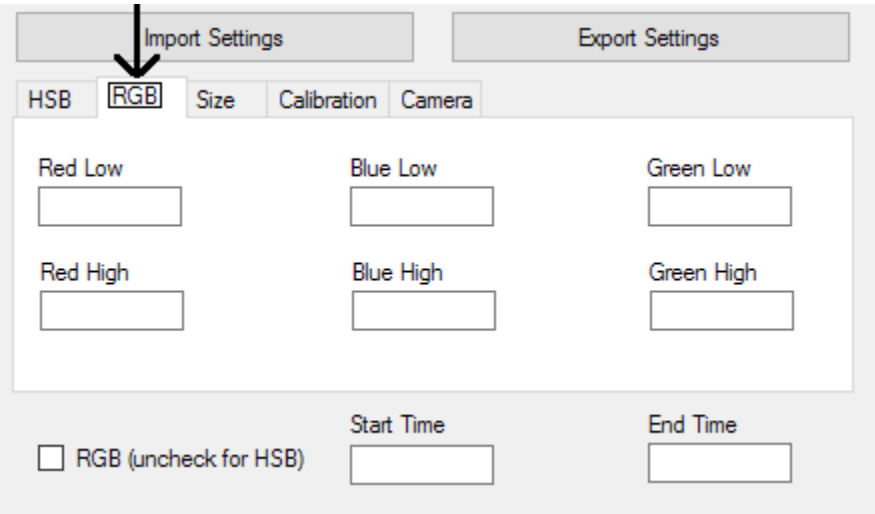


Figure 9: RGB tab

  - This tab is shown in Figure 9.
  - RGB stands for Red-Green-Blue, which is a color scheme for distinguishing the markers from the background.
  - These are integer values that range from 0 to 255. The bounds are inclusive. Thus, at a low and high of 250, the marker can have a value of 250, but no other value.
  - **Red Low** is the minimum range of red for a pixel to be a marker pixel.
  - **Green Low** is the minimum for the color range of green for a pixel to be a marker pixel.
  - **Blue Low** is the minimum for the color range of blue for a pixel to be a marker pixel.
  - **Red High** is the maximum for the color range of red for a pixel to be a marker pixel.
  - **Green High** is the maximum for the color range of green for a pixel to be a marker pixel.
  - **Blue High** is the maximum for the color range of blue for a pixel to be a marker pixel.
  - If any of these values are invalid, or left blank, they will be replaced by 0 (for the low values) or 255 (for high values).
- **Size** Tab
  - This tab is shown in Figure 10.
  - The **Min Size** is the smallest the marker could ever possibly get, in number of total continuous pixels. Defaults to 1.
  - The **Max Size** is the larger a marker could ever possibly get, in number of total continuous pixels. Defaults to 200. If this value is set too high, it could cause a crash.
  - The **Max Sets** is the maximum number of pixel sets the program expects to handle. Increasing this increases run time, and decreasing risks dropping markers or crashing.
  - This **Search Range** is the maximum distance the marker travels in between frames, in pixels. The program will check this distance of pixels in both the x and y axis, to form a square around it's previous point. For a slow moving marker of about 36 pixels, a good estimate might be 30. Larger values take longer to process and may pick up noise, and smaller values could lose the marker entirely.

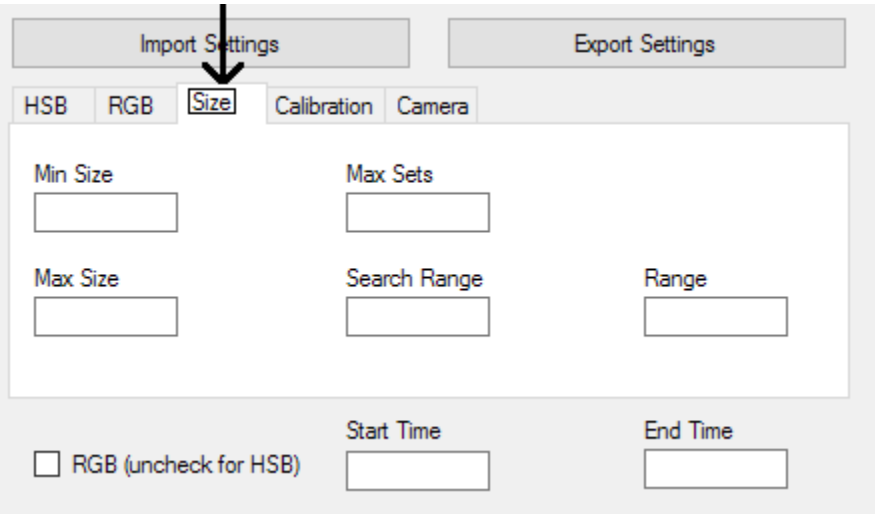


Figure 10: Size tab

- The **Range** is the distance, in inches, between the camera and the markers. Defaults to 10.

- **Calibration** Tab

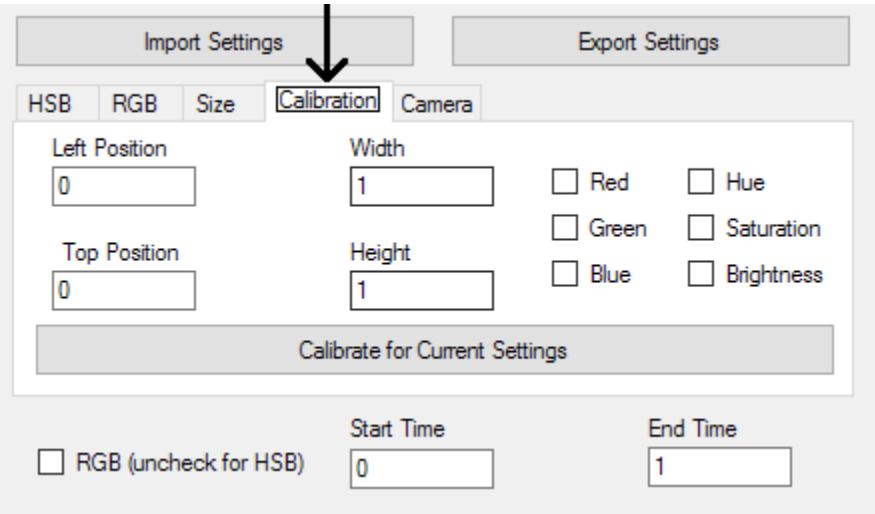


Figure 11: Calibration Tab

- This tab is shown in Figure 11.
- The **Left Position** is the leftmost edge of a bounding box surrounding the marker. Defaults to 0.
- The **Top Position** is the top edge of a bounding box surrounding the marker. Defaults to 0.
- The **Width** is the width of the bounding box surrounding the marker. Defaults to 1.
- The **Height** is the height of the bounding box surrounding the marker. Defaults to 1.

- The **Red** check-box determines whether to filter by red in the statistics (checked), or display the range of values in the statistics (unchecked). This also determines whether to output the csv file showing the red value of each pixel in the selected region.
- The **Green** check-box determines whether to filter by green in the statistics (checked), or display the range of values in the statistics (unchecked). This also determines whether to output the csv file showing the green value of each pixel in the selected region.
- The **Blue** check-box determines whether to filter by blue in the statistics (checked), or display the range of values in the statistics (unchecked). This also determines whether to output the csv file showing the blue value of each pixel in the selected region.
- The **Hue** check-box determines whether to filter by hue in the statistics (checked), or display the range of values in the statistics (unchecked). This also determines whether to output the csv file showing the hue value of each pixel in the selected region.
- The **Saturation** check-box determines whether to filter by Saturation in the statistics (checked), or display the range of values in the statistics (unchecked). This also determines whether to output the csv file showing the Saturation value of each pixel in the selected region.
- The **Brightness** check-box determines whether to filter by brightness in the statistics (checked), or display the range of values in the statistics (unchecked). This also determines whether to output the csv file showing the brightness value of each pixel in the selected region.
- The button starts the calibration process and outputs one to seven csv files, depending on how many of the check-boxes are checked (as explained above).

- **Calibration** Tab

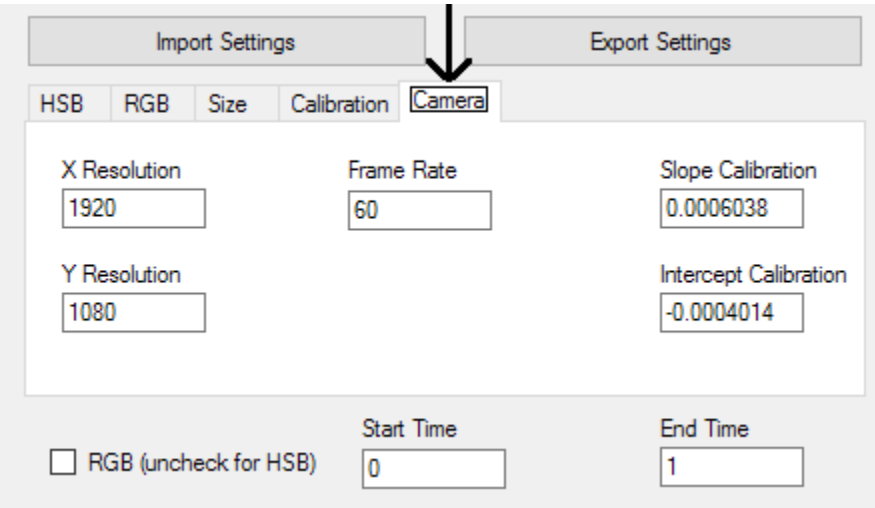


Figure 12: Camera Tab

- This tab is shown in Figure 12.
- The **X Resolution** is the x resolution of the camera (usually the larger of the two). This defaults to 1920. You can set this to be a smaller number, which will scale the image to the smaller size, which may speed up processing time and allow for less errors.

- The **Y resolution** is the y resolution of the camera (usually the smaller value). This defaults to 1080. You can set this to be a smaller number, which will scale the image to the smaller size, which may speed up processing time and allow for less errors.
- The **Frame Rate** is the frame-rate of the camera, or the amount of frames processed for each second of the video. Defaults to 60.
- The **Slope Calibration** is one of the two calibration terms used to calibrate the scaling for distance, specific to the camera and lens. Defaults to 0.0006038.
- The **Intercept Calibration** is one of the two calibration terms used to calibrate the scaling for distance, specific to the camera and lens. Defaults to -0.0004014.

7. Video Settings

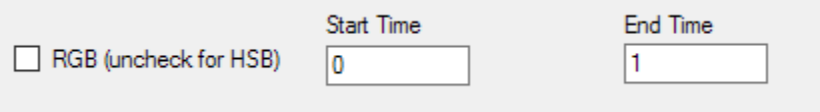


Figure 13: Video Settings

- Shown in Figure 13.
- The **RGB (uncheck for HSB)** checkbox tells the program to analyze the video using RGB if checked, HSB if unchecked. Defaults to HSB.
- The **Start Time** is the time to start processing, in seconds, relative to the video's start time. Defaults to 0.
- The **End Time** is the time to stop processing, in seconds, relative to the video's start time. Be sure to set the end time at least one second after the start time, to ensure the output corresponds with what you would expect. Defaults to 1.

8. **Run Search**

- Shown in Figure 14.
- This will start the program.
- The entire interface freezes when it starts. This is normal.
- Do not try to change the settings after the program has started running.

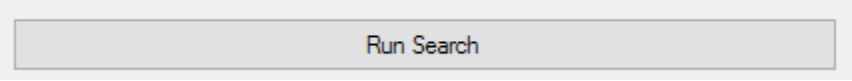


Figure 14: Run Search

9. **Loading Bar**

- Shown in Figure 15.
- This shows the progress of the program when running a search.
- This does not show progress for running the (brief) Calibration function.
- The loading bar will return to its default color upon successful completion.

Figure 15: Loading Bar

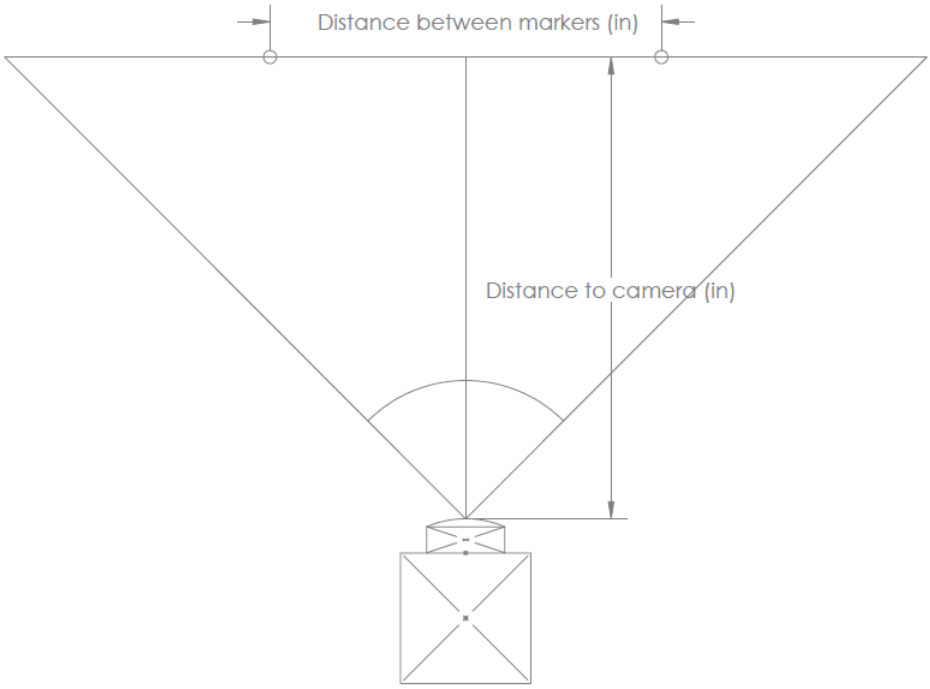


Figure 16: Top down view of system

8.6 Calibration

Calibration is a three step process. Step one is to find the calibration constants step two is to find where the marker is and about how big it is, step three is to find what color ranges work best to find the marker.

8.6.1 Distance Calibration

When a video camera records a test, it takes the image it sees and maps it over a pixel map. In order to translate the the pixel coordinate, the real life distance to a scaling factor must be used. The size of each pixel, in inches depends greatly on the distance of the test rig to the camera. Figure 16 provides a top down view of which measurements need to be taken.

To begin distance calibration, the following are required:

1. Tool for measuring distance
2. Calibration rig
3. Video Camera
4. Level work space to place camera and calibration rig
5. Computer program to view image pixel map (Gimp, Paint, etc.)

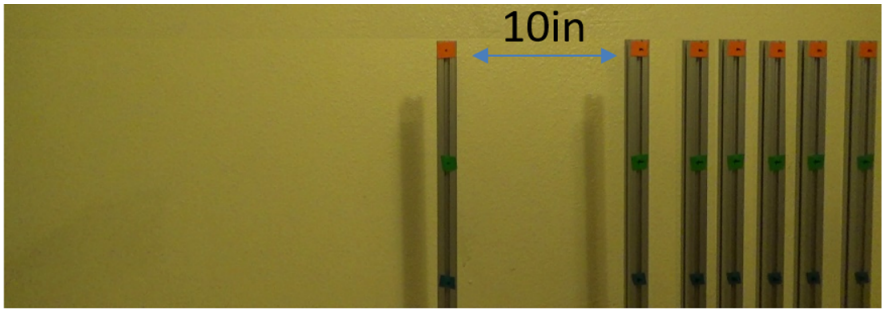


Figure 17: Top: Real distance.

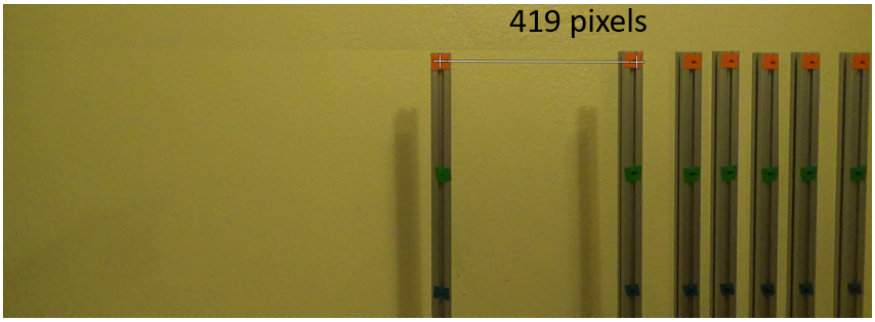


Figure 18: Bottom: measured distance.

Setting up Calibration rig

The calibration rig comprises of 4 major parts: One leg, one slider, and two arms. They are shown and labeled in Figure 17-18. Attach the arms and legs to the slider as shown. The arms and legs should be perpendicular to the slider bar. If they are not, loosen the screws that hold the arm or leg into place and realign them so that they are perpendicular.

Setting up the Video Camera

1. Secure the video camera onto a tripod.
2. Place video camera far enough from the calibration rig to fully capture it.
3. Use the Calibration rig leg to ensure you are perpendicular to the calibration rig.

Creating distance calibration coefficient

New cameras will require a new calibration scheme to give accurate results. To begin distance calibration, first set up the rig at a known distance. In Figure 17 the calibration rig is set so that one arm is in the center of the screen and the other is 10 inches away. After recording the real life distance between markers and the distance between the camera and the test subject, use the camera to take a photo of the of the test rig. Figure 16 provides an example for how to measure distances.

Step by Step Instructions:

1. Place one arm at the middle of the rig.
2. Place the other arm at one end of the rig.
3. Look through video camera view finder to ensure the marks on the arm rig is being fully captured.
4. Take a photo or short video of the calibration rig.
5. Measure and record the distance, in inches, between the markers on the calibration rig.
6. Without moving the camera or test rig, move the arm closer the edge of the rig towards the center arm by 2 inches.
7. Take another photo or short video of calibration rig.
8. Measure and record the distance, in inches, between the markers on the calibration rig.
9. Using any image or video editor, open up the images or videos obtained.
10. Measure the pixel distance between the marks in each axis independently.
11. Input the data in to excel as shown in Figure 19.

A1 | Stationary Test

	A	B	C	D	E	F	G	H
1	Stationary Test	Distance to camera	Distance between markers	Verticle pixel distance 6in	Verticle pixel distance 12in	Horizontal Pixel Distance Orange	Horizontal Pixel Distance Green	Horizontal Pixel Distance Blue
2								
3	46	33.75	21.35	245	496	935	924	919
4	47	33.75	19	247	500	819	814	808
5	48	33.75	17	247	504	736	730	724
6	49	33.75	15	248	506	639	639	633
7	50	33.75	13	249	507	561	555	554
8	51	33.75	10	250	508	429	417	414
9	Control		0	252	511			

Figure 19: Example of measurements taken for creating distance calibration coefficients.

12. Divide the measured distance (in) by the the pixel distance (pixels) the distance calibration for each independent axis.
13. The new calibration coefficient can be added to the code.

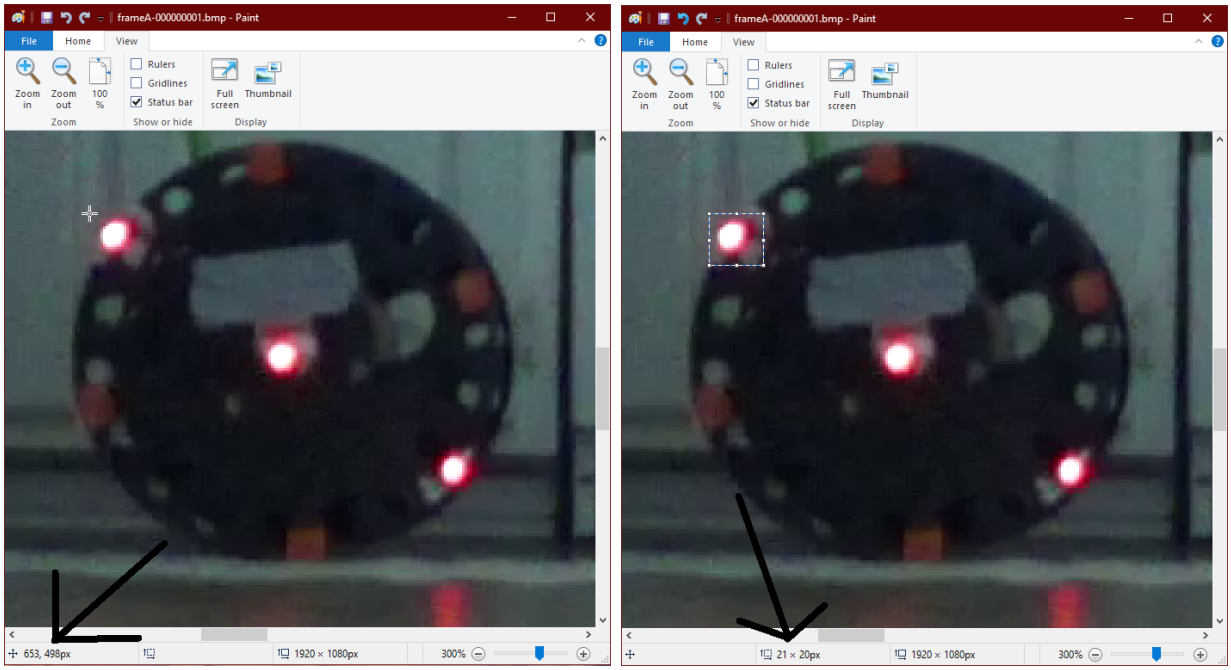


Figure 20: Finding Marker

8.6.2 Finding the Marker

This is mostly done out of the program, and may need to be repeated or refined. See section 5 for more details on each input box. See Figure 20 above, for help.

1. Open the Program.
2. Enter the Data Folder Location (ending in `\Data\`).
3. Enter the Video File Name (ending with an extension).
4. Enter the Start Time for your calibration. (ie: 0, if the marker is clearly visible at the beginning of the recording.)
5. Enter the End Time as 1+ Start Time.
6. Enter the frame-rate of your camera.
7. Enter the X and Y resolution for your camera or test.
8. Click Run Search.
9. Of the new images created (see Figure 22), find the first one (it may be A or B).
10. Open this image in paint or Gimp or any other program, preferably one that lists the pixel coordinates.
11. Locate the leftmost pixel for the marker, and note it's coordinates (number of pixels) under Left Position. The leftmost of the frame should be 0.

12. Locate the uppermost pixel for the maker, and note it's coordinates (number of pixels) under Top Position. The top of the frame should be 0.

13. Determine the width of the marker, from the left-most to the right-most (number of pixels), and note this under Width.

14. Determine the height of the marker, from the top to the bottom (number of pixels), and note this under Height.

8.6.3 Calibrating for Color

See Figure 21 for help viewing the calibration-red.csv, calibration-hue.csv, etc. files.

1. Mentally pick whether to use RGB or HSB. RGB is typically best for passive markers, like tape, and HSB is typically best for active markers like LED's.

2. Check all the boxes of the **opposite** choice. If you picked RGB, check the Hue, Saturation, and Brightness box. If you picked HSB, check the Red, Green, and Blue boxes.

3. Double-check that the text-boxes for the both the RGB and HSB tab are 0 (for the Low values) to 255 (for the High values).

4. Find the marker (see section 6.2).

5. Click the Calibrate for Current Settings button.

6. Use the Calibration files, red, green, blue, hue, saturation, and / or brightness to confirm you have selected the marker, and narrow your selection if it is too big.

   Each csv file shows the value of every pixel in the region. For example, in "calibration-blue.csv" it will show a grid of numbers from 0 to 255 which are the pixel's blue color values throughout the selected region. You should be able to discern a noticeable pattern in at least one of these files, where a collection of pixels have similar numbers that aren't present elsewhere in the image. Make sure you include the entire marker.

   The "calibration-statistics.csv" file shows, initially, the number of pixels of each color type (red, green, or blue) at every particular value both throughout the whole image, and throughout the selected region. In the example Figure 23, in B2, you can see how many pixels have green value of 0 in the entire image, and in C3 you can see how many pixels have green value 1 within the selected region.

7. Mentally pick one color type and a range to use as a filter.

   This should be a conservative range that is likely to include the entire marker for any random frame. For example, when tracking a red marker, the red value might always be 255, for the marker.

   Be sure you are not too restrictive, as you might lose the marker, or too loose, as you might include too much noise.

   As a general rule, try to avoid tracking purely white (red = 255, green = 255, blue = 255 pixels), as this is a common color, as opposed to slightly off-white (at least one color type (red, green, or blue) isn't 255, although the others may be 255).

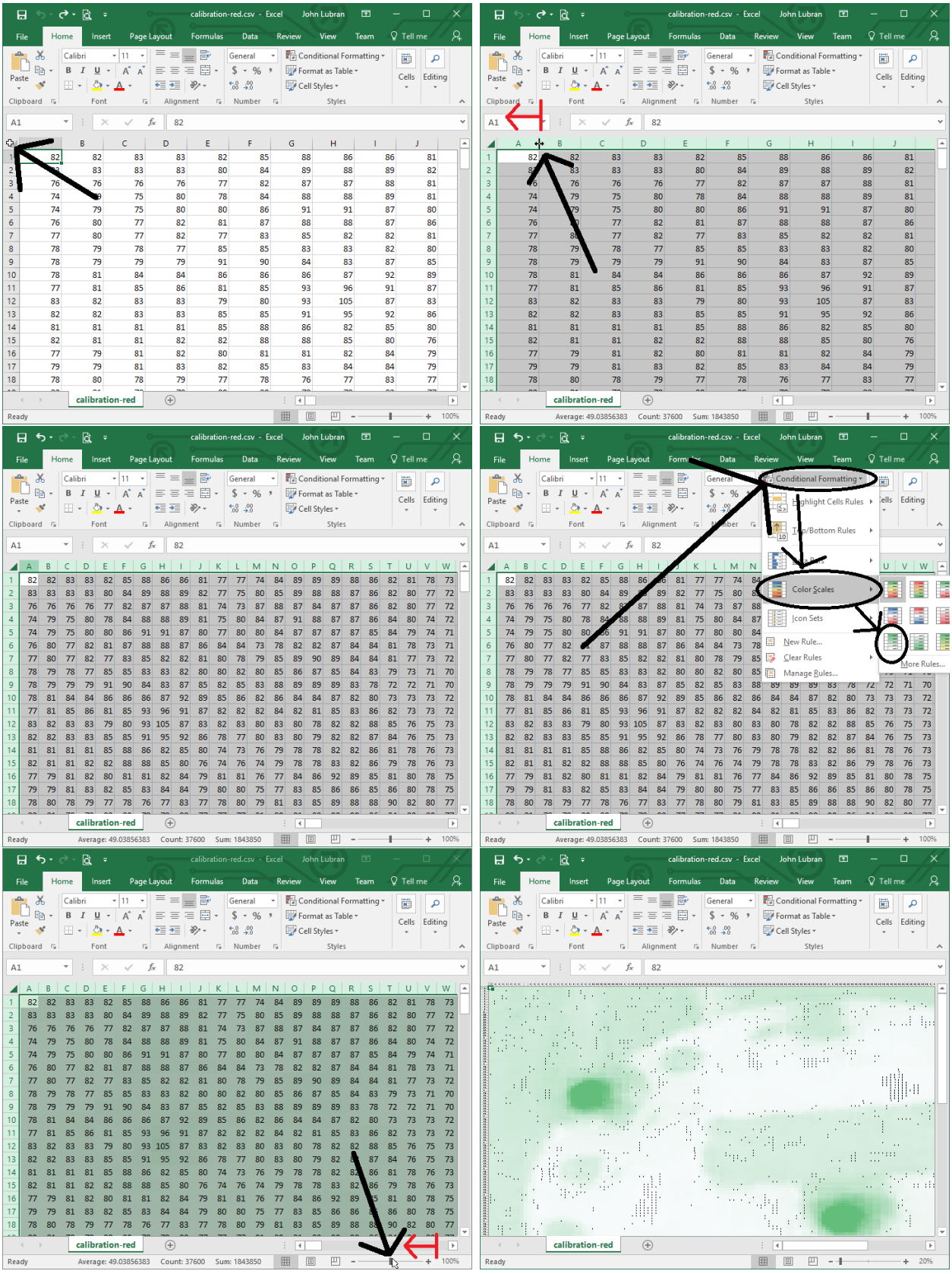


Figure 21: Viewing Calibration-color.csv

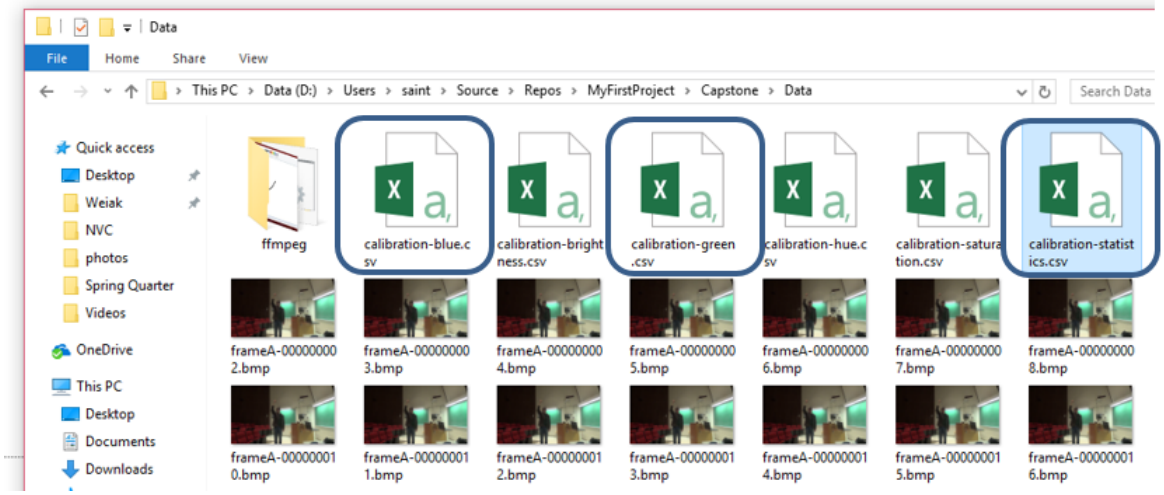


Figure 22: Color Calibration Files

	A	B	C	D	E	F	G
1	Value	Total Valid Green Points	Selected Valid Green Points	Total Valid Blue Points	Selected Valid Blue Points	Total Valid Points	Selected Valid Points
2	0	0	0	0	0	1782	35
3	1	0	0	0	0		
4	2	0	0	0	0		
5	3	0	0	0	0		
6	4	0	0	0	0		
7	5	0	0	0	0		
8	6	0	0	0	0		
9	7	0	0	0	0		
10	8	0	0	0	0		
11	9	0	0	0	0		
12	10	0	0	0	0		
13	11	0	0	0	0		
14	12	0	0	0	0		
249	247	0	0	3	0		
250	248	0	0	7	0		
251	249	0	0	1	0		
252	250	0	0	3	0		
253	251	0	0	4	0		
254	252	0	0	16	0		
255	253	0	0	10	0		
256	254	0	0	15	0		
257	255	1747	0	1684	0		

Figure 23: Calibration Statistics File

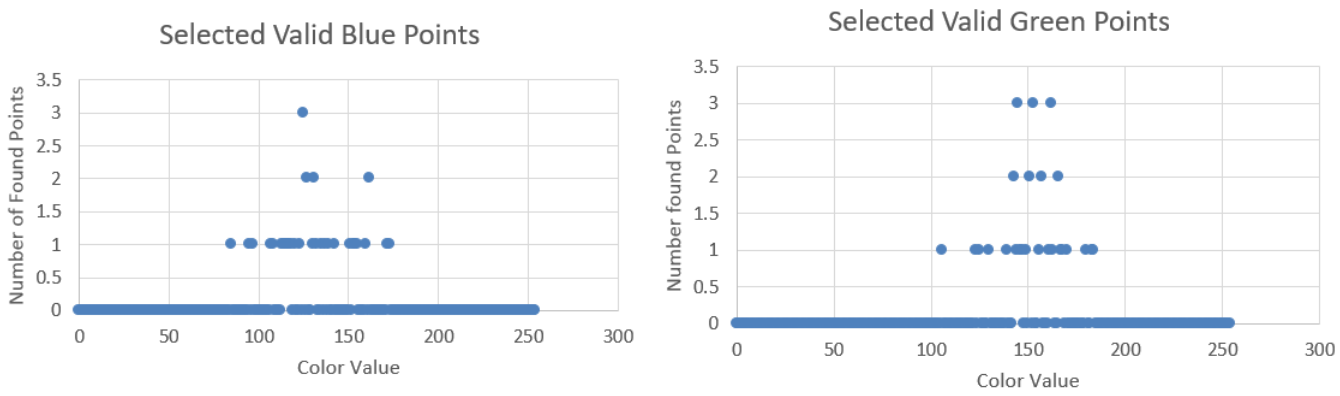


Figure 24: Valid Selected Points

	A	B	C
1	Value	Total Valid Points	Selected Valid Points
2	0	35	35

Figure 25: Ideal Calibration Stat

8. Enter the bounds appropriately in the HSB or RGB tabs, then select the check-box for that color in the Calibration tab.

   When you repeat, the "calibration-statistics.csv" will exclude all pixels that fail to pass the checked color's range restrictions. In the example, red was filtered at 255 only. Thus, B2 shows all pixels in the entire frame that have both red = 255 and green = 0, and C3 shows all pixels in the selected region that have both red = 255 and green = 1.

   When you repeat, the "calibration-statistics.csv" will update the total valid points to only include all points in the frame that match any checked color's ranges, as well as for the selected region.

   It may help to plot the selected values with the selected points of a particular color of that value in order to help narrow your fields. An example of this is shown in Figure 24.

9. Return to step 5.

10. Check that the number of valid points in the selected region is equal or nearly equal tot he number of valid points in the entire image (see Figure 25).

11. Test with other parts of time in the video or run the program to confirm you are not losing the marker at any frame. If you lose the marker on any frame, you probably have too stringent ranges and need to expand them.

12. Press the Export Settings button to export the settings.

8.7 Handling Errors

- If the program crashes, check the following:
  - The file location is correct, and includes a slash at the end, ie: `...\data\`

- The video file is correct, and includes the extension.
- Max Size is fairly small (under a few thousand).
- The Start time is either 0, or somewhere before the end of the video. Keep in mind the units are in seconds.

- Common Errors:

1. If the program finds too many points:
   - Check your RGB or HSB ranges, and see if you can refine them.
   - Check if your range includes Red = 0, Green = 0, and Blue = 0. Try to avoid this, if possible.
   - Check if your range includes Red = 255, Green = 255, and Blue = 255. Try to avoid this, if possible.
   - Check your Min Size and Max Size and see if you can narrow that range down.
   - Make sure the RGB (uncheck for HSB) is properly checked or unchecked.
2. If the program finds too few points:
   - Check your RGB or HSB ranges, and see if you can expand them.
   - Check your Min Size and Max Size and see if you can expand that range.
   - Make sure the RGB (uncheck for HSB) is properly checked or unchecked.

8.8 Post Processing

Now that you have your output.csv, we ready to begin analyzing the data.

8.8.1 Data Files

Figure 26 is a list of the data files generated by the program. These files can be found in the Data Folder Location.

Open on the output-#.csv in order to view the data. This data can be analyzed to give the position of the marker across time.

8.8.2 Example Output

The output file will be named in the format "Output-#.csv". For example, "Output-0.csv" or "Output-123.csv". The program always outputs to the lowest numbered output file not already occupied. Thus, if you delete or move some of the output files, the program will output using the gap names.

Here is an example of the output-#.csv file generated by the code,that was collected when analyzing a marker at the tip of a pendulum motion.

Columns explained:

- Time(s): The time in seconds in the video for the frame that was analyzed.
- x position(in): The distance of the found marker from the left edge of the screen.
- y position(in): The distance of the found marker from the bottom of the screen.
- Marker Size (pixels): The number of pixels that compose the marker.

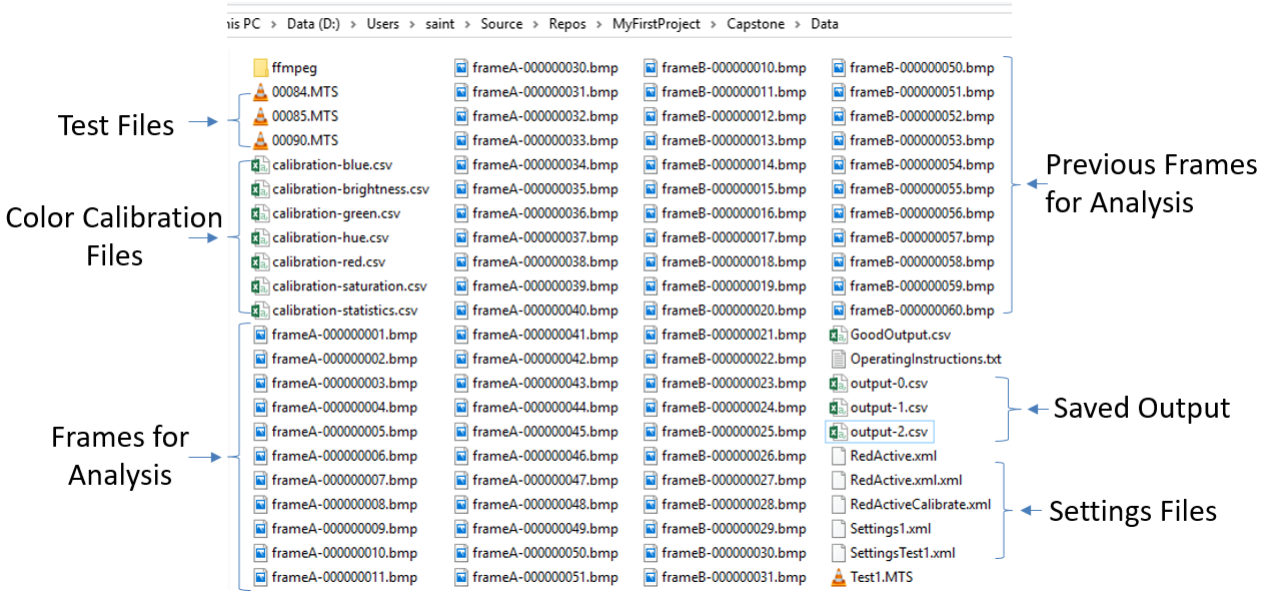


Figure 26: Example of Data Files

	A	B	C	D
1	time (s)	x position (in)	y position (in)	marker size (pixels)
2	0.016666667	48.808266	1.6101696	35
3	0.033333333	48.053499	1.6101696	32
4	0.05	47.2484142	1.6101696	36
5	0.066666667	46.4936472	1.6604874	24
6	0.083333333	45.6885624	1.7108052	31
7	0.1	44.9841132	1.761123	27
8	0.116666667	44.2293462	1.8114408	31
9	0.133333333	43.5752148	1.9120764	25
10	0.15	42.8707656	1.9623942	27
11	0.166666667	42.2166342	2.0630298	29
12	0.183333333	41.5625028	2.1636654	27
13	0.2	40.9586892	2.264301	28
14	0.216666667	40.3548756	2.4152544	27
15	0.233333333	39.8013798	2.51589	24
16	0.25	39.3485196	2.6165256	25
17	0.266666667	38.8453416	2.7171612	27
18	0.283333333	38.3421636	2.8681146	26
19	0.3	37.989939	2.9687502	25
20	0.316666667	37.5873966	3.0693858	24

Table 1: Example output-#.csv

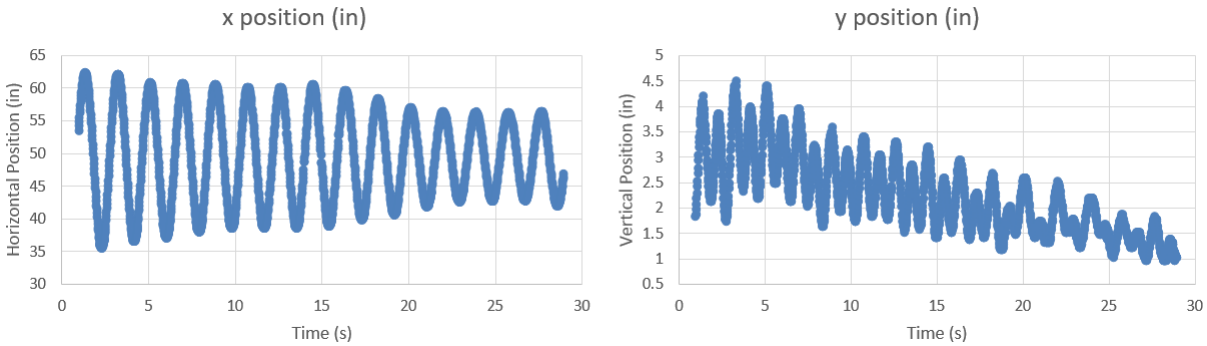


Figure 27: Example graph of X and Y motion over time.

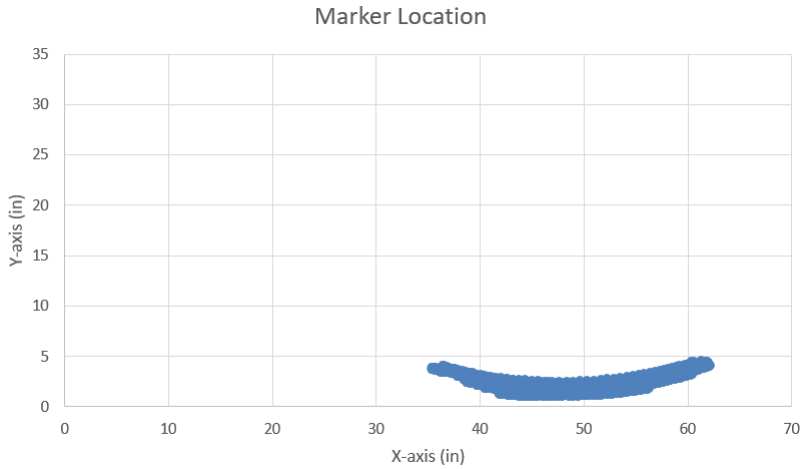


Figure 28: All recorded marker location of the course of the test.

These data tables can be plotted to analyze the motion of the marker over time.

The x-position and y-position columns can be graphed against each other to view the total motion of the marker over the period of the test.

8.9 Feature List

- Marker Tracking (Distinguish specific markers)
- Marker Calibration
- Focused Search
- Distance Calibration
- Color Calibration
- Active Maker Tracking
- Passive Marker tracking
- Loading Bar

8.10 Marker Configuration

8.10.1 Circuit Diagram

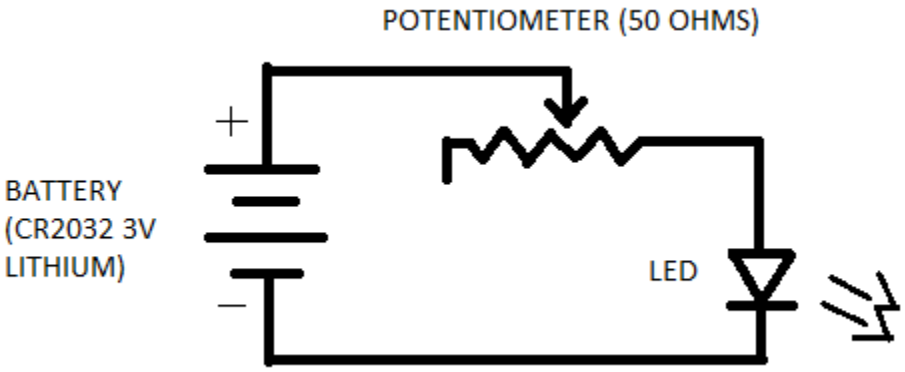


Figure 29: Active Marker Circuit

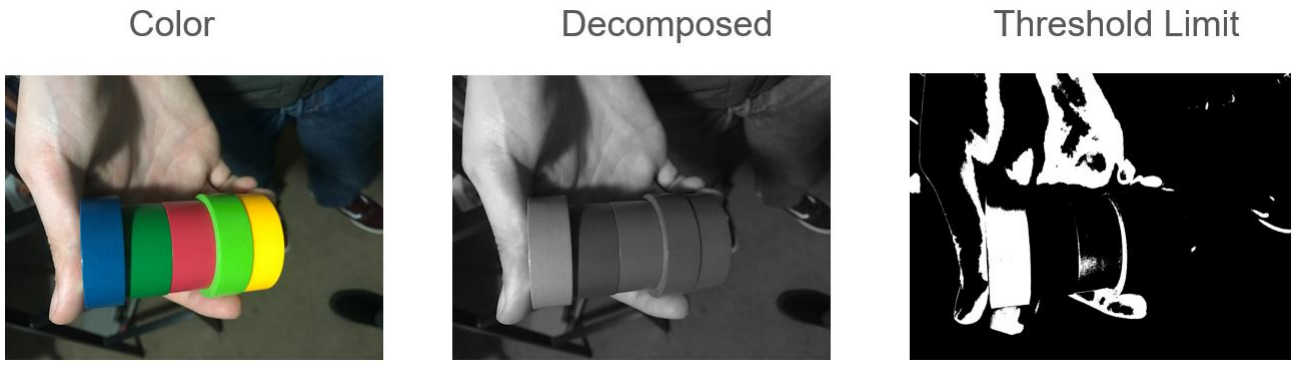
8.11 Bill of Materials

Part Number	Description	Unit Cost ($)	# of parts
1	Active Marker Circuitry	5	10
2	Passive Markers	0.05	10
3	Camera	250	1
4	Tripod	70	1
5	Calibration Frame	112	1

9 Spectral Analysis

9.1 General Trends from Initial Testing:

- Compared passive markers to active markers.
- Passive are much smaller and less intrusive.
- Fast moving objects require low exposure time to avoid motion blur.
- Very small objects are more difficult to track.
- Low exposure time degrades tracking effectiveness.
- Active markers are more noticeable than passive markers at low exposure time.



9.2 Analysis Criteria

% of Pixels filtered out by Threshold Criterion.

Marker	Exposure	Hue	Saturation	Red	Green	Blue
Passive Yellow	1/1.4	94.7	90.0	97.4	89.1	97.4
Passive Yellow	1/2	37.0	86.8	97.9	98.6	85.4
Active Red	1∞	0.00	55.9	100.0	99.2	37.9

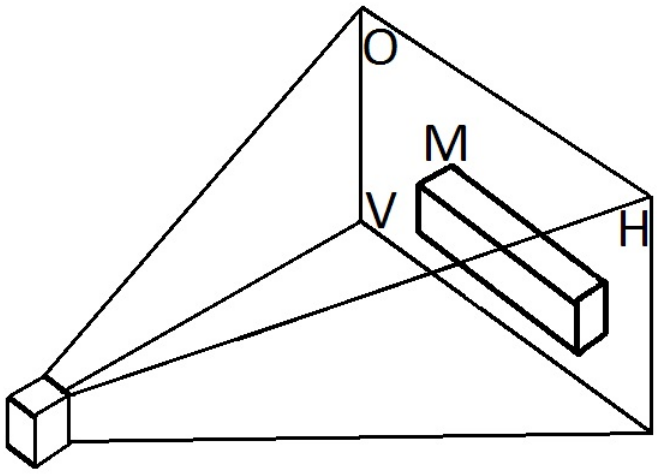
9.3 Conclusions from ATK Site Testing

- Interlaced video modes are problematic; progressive scan is preferred.
- Issue with directional output of LEDs; their visibility drops if they point away from the camera. Diffusers may be required.
- Active markers may oversaturate camera if they are too bright.

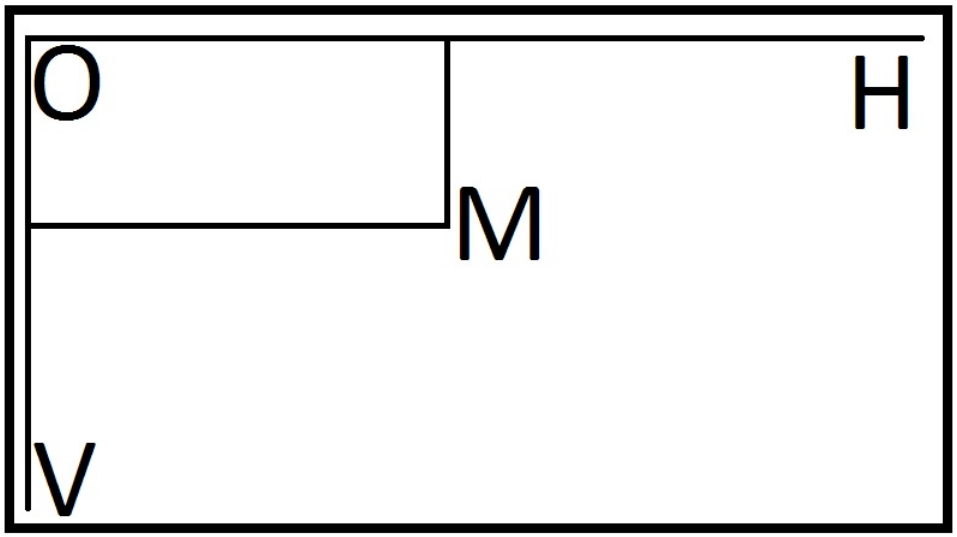
10 Theoretical Basis for Accuracy

10.1 Position

To find real($_r$) 3D coordinates, we first convert 2D virtual($_v$) coordinates to a 3D projection as follows:
Real Space



Virtual Space



$$\vec{MO}_r = \left|\frac{MO_{v,x}}{HO_v}\right| \cdot \vec{HO}_r + \left|\frac{MO_{v,y}}{VO_v}\right| \cdot \vec{VO}_r$$
$$\vec{MO}_r = \left|\frac{M_{v,x} - O_{v,x}}{H_v - O_v}\right| \cdot \vec{HO}_r + \left|\frac{M_{v,y} - O_{v,y}}{V_v - O_v}\right| \cdot \vec{VO}_r$$
$$(\delta MO_{r,x})^2 = \left(\delta M_{v,x}\frac{\partial \vec{MO}_r}{\partial M_{v,x}}\right)^2 + \left(\delta \vec{HO}_r\frac{\partial \vec{MO}_r}{\partial \vec{HO}_r}\right)^2 + \left(\delta \vec{H}_v\frac{\partial \vec{MO}_r}{\partial \vec{H}_v}\right)^2 + \left(\delta \vec{O}_v\frac{\partial \vec{MO}_r}{\partial \vec{O}_v}\right)^2$$
$$(\delta MO_{r,x})^2 = \left(\frac{\delta M_{v,x}\cdot HO_r}{HO_v}\right)^2 + \left(\frac{\delta HO_r \cdot MO_{v,x}}{HO_v}\right)^2 + \left(\frac{-\delta H_v \cdot MO_{v,x}\cdot HO_r}{(HO_v)^2}\right)^2 + \left(\frac{-\delta O_v(H_v - M_{v,x})\cdot HO_r}{HO_v^2}\right)^2$$
$$\delta M_{v,x} = \delta O_v = \delta H_v = \delta M_{v,y} = \delta V_v$$
$$\frac{MO_{v,x}}{HO_v} \equiv n \leq 1 \text{ and } \frac{MO_{v,y}}{VO_v} \equiv m \leq 1$$
$$(\delta MO_{r,x})^2 = 2(n^2 - n + 1)\left(\frac{\delta O_v \cdot HO_r}{HO_v}\right)^2 + (n\cdot \delta HO_r)^2$$
$$(\delta MO_{r,y})^2 = 2(m^2 - m + 1)\left(\frac{\delta O_v \cdot VO_r}{VO_v}\right)^2 + (m\cdot \delta VO_r)^2$$

We know that δO_v is at least 0.5, but this only depends on errors in pixel detection. (False readings, etc.) We know that the real values(OV_r and OH_r) have an uncertainty of about 0.004mm. At scales of 1-10 m, this term is insignificant. Thus, we get

$$\delta MO_{r,x} \leq \frac{\sqrt{2}}{2}\cdot\frac{HO_r}{HO_v} \text{ and } \delta MO_{r,y} \leq \frac{\sqrt{2}}{2}\cdot\frac{VO_r}{VO_v}$$

If we maximum use of a 1920 by 1080 pixel camera at either 1 meter, 5 meters, or 10 meters, then we get the following resolution(Smallest data value in output) and uncertainty (statistically likely deviance) values.

	1m	5m	10m
Resolution in X	|0.52mm|	|2.60mm|	|5.21mm|
Resolution in Y	|0.93mm|	|4.63mm|	|9.26mm|
Uncertainty in X	±0.37mm	±1.84mm	±3.68mm
Uncertainty in Y	±0.65mm	±3.27mm	±6.55mm

10.2 Velocity

Velocity must be calculated using some finite differencing method. This means that the uncertainty depending on the order of differencing used (how many points), your timescale, and your uncertainty in position data. For a two point central method, we get

$$\frac{\partial \vec{MO}(0)}{\partial t} = \frac{\vec{MO}(\Delta T) - \vec{MO}(-\Delta T)}{2\Delta T} - \frac{\vec{MO}'''(0)}{6}(\Delta T^2)$$
$$\frac{\partial \vec{MO}(0)}{\partial t} = \frac{-\vec{MO}(2\Delta T) + 8\vec{MO}(\Delta T) - 8\vec{MO}(-\Delta T) + \vec{MO}(-2\Delta T)}{12\Delta T} - \frac{\vec{MO}''''(0)}{30}(\Delta T^4)$$

The accuracy of finite differencing depends on the function we are evaluating itself. The first method is exact if jerk is zero. The second method is exact if the 2nd derivative of jerk is zero. For the purposes of uncertainty, we assume a sinusoidal position function.

$$\left(\delta\frac{\partial\vec{MO}(0)}{\partial t}\right)^2 = \left(\delta\vec{MO}(\Delta T)\frac{\partial\vec{MO}_r}{\partial\vec{MO}(\Delta T)}\right)^2 + \left(\delta\vec{MO}(-\Delta T)\frac{\partial\vec{MO}_r}{\partial\vec{MO}(-\Delta T)}\right)^2 + \left(\delta\Delta T\frac{\partial\vec{MO}_r}{\partial\Delta T}\right)^2 + \left(\frac{\Delta T^2}{6}\vec{MO}'''(0)\right)^2$$

$$\left(\delta\frac{\partial\vec{MO}(0)}{\partial t}\right)^2 = 2\left(\frac{\delta\vec{MO}}{2\Delta T}\right)^2 + \left(\delta\Delta T\frac{\vec{MO}(\Delta T)-\vec{MO}(-\Delta T)}{2\Delta T^2}\right)^2 + \left(\frac{\Delta T^2}{6}\vec{MO}'''(0)\right)^2$$

If we use the 4 point method, however, we get

$$\left(\delta\frac{\partial\vec{MO}(0)}{\partial t}\right)^2 = \left(\frac{65}{18}\right)\left(\frac{\delta\vec{MO}}{2\Delta T}\right)^2 + \left(\frac{\Delta T^4}{300}\vec{MO}''''(0)\right)^2 + \left(\frac{-\vec{MO}(2\Delta T)+8\vec{MO}(\Delta T)-8\vec{MO}(-\Delta T)+\vec{MO}(-2\Delta T)}{12}\right)^2\left(\frac{\delta\Delta T}{\Delta T^2}\right)^2$$

If the accuracy of the finite differencing method is, on average, equivalent(and based on it's derivation, it will be), then so long as $\frac{29}{72}\left(\delta\vec{MO}\right)^2 + \left(\frac{\Delta T^5}{30}\vec{MO}''''(0)\right)^2 < \left(\frac{\Delta T^3}{6}\vec{MO}'''(0)\right)^2$, the higher order method will be better.

Assuming we only use the two point method, we can calculate various uncertainties for various velocities and ranges. Here, I assume a uncertainty in frame rate of 1s/60.

Total time interval of 1 second ($\Delta T = 1s/3$), range = 1m.

	0.2m/s	1m/s	5m/s
V" = 1m/s^3	±21.1mm/s	±53.3mm/s	±251mm/s
V" = 5m/s^3	±93.1mm/s	±105mm/s	±267mm/s
V" = 10m/s^3	±185mm/s	±192mm/s	±311mm/s

Total time interval of 0.1 second ($\Delta T = 1s/30$), V" = 10m/s^3.

	1m	5m	10m
V = 0.1m/s	±52.5mm/s	±94.0mm/s	±167mm/s
V = 0.2m/s	±101mm/s	±128mm/s	±188mm/s
V = 0.5m/s	±251mm/s	±262mm/s	±296mm/s

From this information, we can see jerk affects the results more at higher intervals, which is great since at those intervals we can use higher order methods more readily.

We can also see that as the device speeds up, we will need to expand the time steps. Alternatively, we could use a higher frame rate, however, it's likely that most cameras will have a shorter window for actually capturing each frame than merely the refresh rate.

10.3 Acceleration

Acceleration is almost identical to velocity, except a different method is used for 2nd order differentiation.

$$\frac{\partial^2 \vec{MO}(0)}{\partial t^2} = \frac{\vec{MO}(\Delta T) - 2\vec{MO}(0) + \vec{MO}(-\Delta T)}{\Delta T^2} - \vec{MO}''''(0)\frac{\Delta T^2}{12}$$

$$\left(\delta \frac{\partial^2 \vec{MO}(0)}{\partial t^2}\right)^2 = (\vec{MO}''''(0)\frac{\Delta T^2}{12})^2 + 6\left(\frac{\delta \vec{MO}}{\Delta T^2}\right)^2 + \left(\frac{\vec{MO}(\Delta T) - 2\vec{MO}(0) + \vec{MO}(-\Delta T)}{\Delta T^2}\frac{2\delta \Delta T}{\Delta T}\right)^2$$

Assuming jerk is constant, we get a similar table to velocity
Total time interval of 1 second ($\Delta T = 1s/3$).

	1m	5m	10m
A = 1m/s^2	±0.1m/s^2	±0.13m/s^2	±0.19m/s^2
A = 5m/s^2	±0.5m/s^2	±0.51m/s^2	±0.53m/s^2

Of course, if we knew $\delta \Delta T$ more accurately, these would get a bit more accurate. This also happens when the acceleration is close to 0.

	1m	5m	10m
A = 0	±16.5mm/s^2	±82.7mm/s^2	±166mm/s^2

11 Pendulum Motion Experiment

11.1 Objective

The purpose of this experiment is to quantify the precision and accuracy of the optical kinematic tracking system across time. To this end, a pendulum is perturbed and its motion follows a circular path. The end is tracked to measure the position along that circle, and the radius can be derived at each point. These radii can be compared with other measures to determine the accuracy of the motion tracking system.

11.2 Background

The complete motion of a pendulum is difficult to accurately model, due to complications such as the frictional force at the hinge, the drag coefficient, etc. Instead, this experiment focuses on measuring position across time. The experiment utilizes the fact that a pendulum's length does not change throughout it's motion.

11.2.1 Pendulum Length Calculation

Given the position of two points in both real (P_{rba}) and virtual space (P_{vba}) and the position of the lowest point of the pendulum in both spaces(P_{rx0},P_{ry0},P_{vx0},P_{vy0}) and the length of the pendulum in real space(L), we can convert the raw data from our function, which is in terms of x(P_{vx}) and y(P_{vy}) virtual positions to real positions x(P_{rx}) and y(P_{ry}) with respect to the hinge.

$$P_{hrx} = P_{rx0} \tag{1}$$

$$P_{hry} = P_{ry0} - L \tag{2}$$

$$P_{rx} = P_{hrx} + (P_{vx} - P_{hvx}) \left(\frac{P_{rba}}{P_{vba}} \right) \tag{3}$$

$$P_{ry} = P_{hry} + (P_{vy} - P_{hvy}) \left(\frac{P_{rba}}{P_{vba}} \right) \tag{4}$$

We can then find the apparent length (L_a) of the pendulum from each point by calculating the distance from the pendulum as

$$L_a = \sqrt{P_{rx}^2 + P_{ry}^2} \tag{5}$$

11.3 Methods

A four foot steel rod was with a loop at the top was attached to a hook to be used as a hinge. This set-up was attached to the top of a door. Near the bottom of the rod, a measuring tape was affixed to the door to provide scaling. At the bottom of the rod, a battery, resistor, and the red marker LED were attached with masking tape.
About eight feet away from this, horizontally, an iPhone 7 was propped up on a futon. The futon lifts the iPhone up about a foot and the end of the pendulum hangs roughly 2 feet off the ground. The camera was positioned vertically and parallel with the door with respect to visual precision.
A pair of three 5000K color LED lights were directed at the pendulum with a plastic bag to distribute the light. The iPhone 7 was installed with an application which enabled shutter speed control, which was turned up for the minimum exposure time.

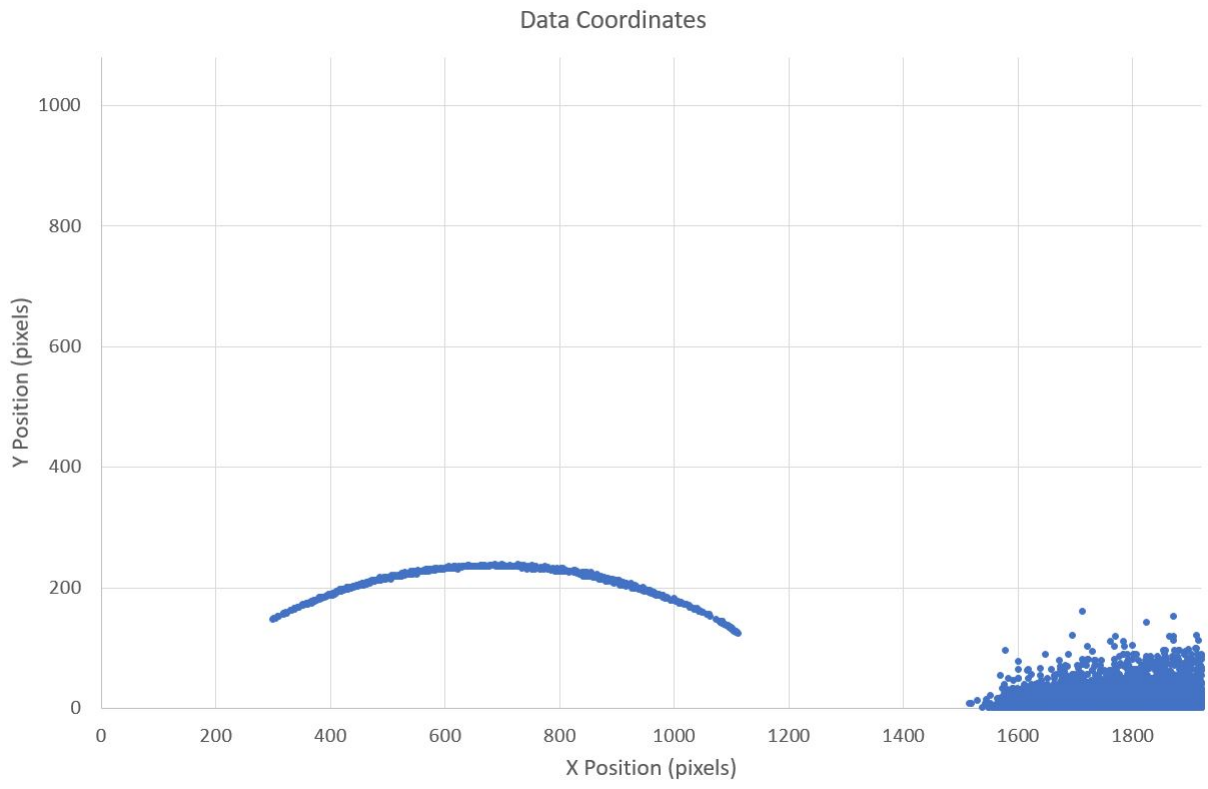
The pendulum was oscillated for roughly 90 seconds (although only the first 30 seconds are used). The plane of oscillation was as close to parallel to the door as possible.
The data recorded from the camera was then input into the program and some post-processing was done manually to convert the pixel coordinates into length position as explained in the background. The accuracy of the kinematic mapping system can be found to be at least as good as the experiment's uncertainty, if the predicted values are indeed within that uncertainty. The precision can be determined based off of the variance of the radius with respect to the average radius.

11.4 Data

11.4.1 Setup Geometry

Height of futon (ground base for camera): 14±1 inches
Height of camera from lens to bottom: 2.1±0.1 inches
Angle of camera out of plane (horizontally): 0± 10^o
Distance from camera to pendulum: 8±1 feet
Distance from ground to pendulum: 26±1 inches
Distance across reference points: 23.75±0.25 inches
Length of pendulum: 48±0.5 inches
Virtual position of lowest point of pendulum: (688,239)±3 in pixels
Virtual span across reference points: 428±3 pixels Note, uncertainty in virtual points is due to difficulty in discerning positions to the pixel. Low exposure time resulted in some ambiguity in interpreting pixels by eye for reference points.

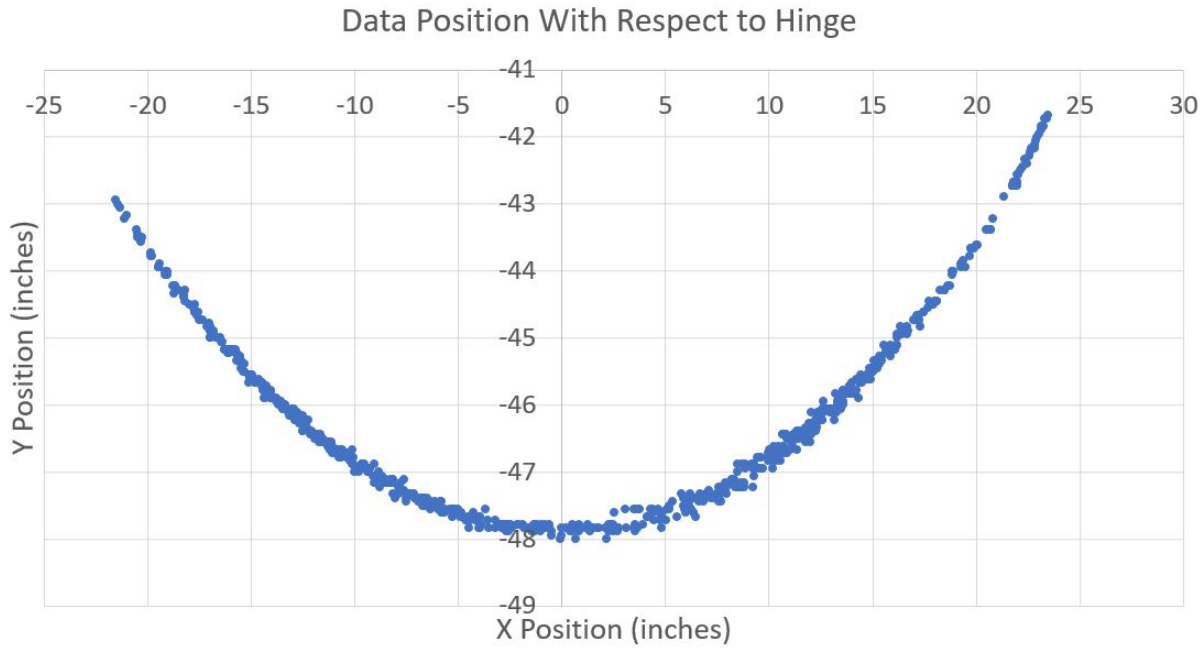
11.4.2 Raw output



Note the y-axis is flipped and there is some noise unrelated to the pendulum.

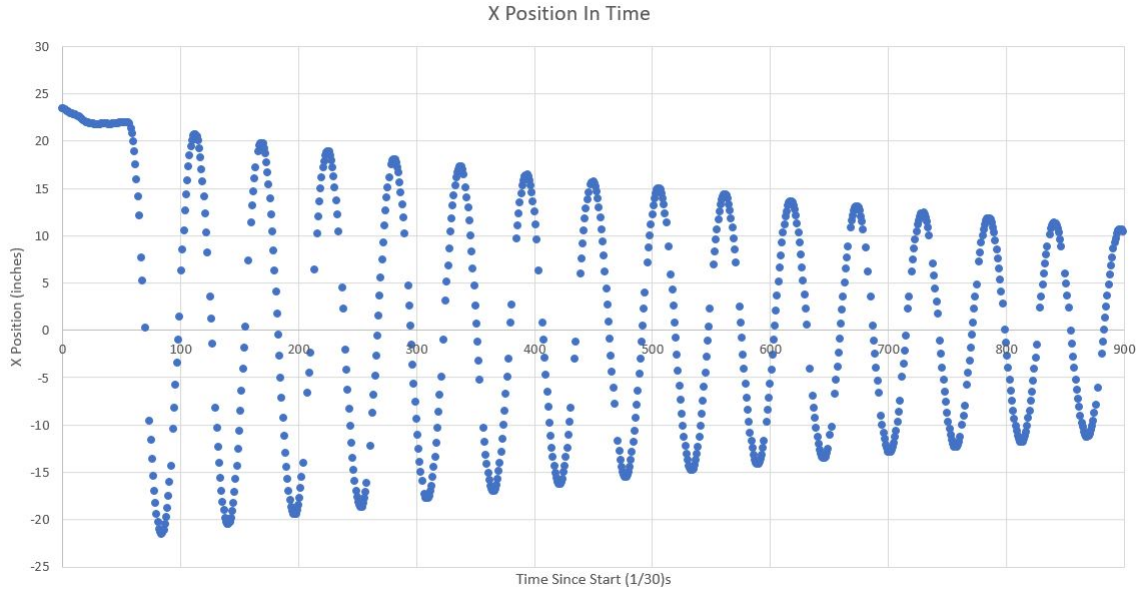
11.4.3 Refined output

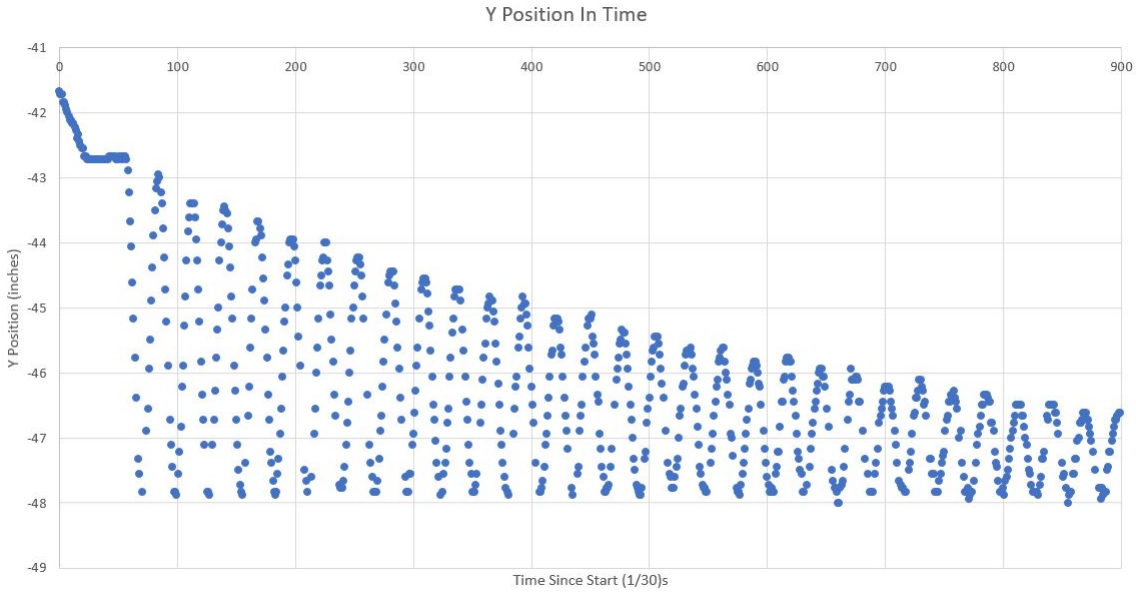
The noise was removed on the basis of it's x position (which was obviously a significant distance from all of the pendulum data). The data was then converted to real position with respect to the hinge to get the refined output.



11.4.4 Output in time

Although not needed to prove accuracy, the data in time may help demonstrate by example the validity of the argument in the extrapolation.





11.4.5 Uncertainty in Experiment

The generic equation for uncertainty is

$$\Delta f(x_1, x_2 ... x_n) = \sqrt{\Sigma_{i=1}^{n} \left(\frac{\partial f}{\partial x_i} \Delta x \right)^2} \tag{6}$$

We ultimately want to find the measured radius, so we get

$$L_a = \sqrt{(P_{vx} - P_{vx0})^2 \left(\frac{P_{rba}}{P_{vba}} \right)^2 + \left(P_{vy} - P_{vy0} + L \left(\frac{P_{vba}}{P_{rba}} \right) \right)^2 \left(\frac{P_{rba}}{P_{vba}} \right)^2} \tag{7}$$

If we assume the uncertainty in our points is zero, and we calculate an uncertainty that is greater than the uncertainty seen in our data, then we know the uncertainty in our experiment is an over-estimate and to whatever extent it is an over-estimate, the remainder in uncertainty is the uncertainty of our data. Therefore, we can upper-bound our data's accuracy as the total uncertainty from the experiment.

$$\Delta L_a = \sqrt{\left(\frac{\partial f}{\partial P_{vx0}} \Delta P_{vx0} \right)^2 + \left(\frac{\partial f}{\partial P_{vy0}} \Delta P_{vy0} \right)^2 + \left(\frac{\partial f}{\partial L} \Delta L \right)^2 + \left(\frac{\partial f}{\partial P_{rba}} \Delta P_{rba} \right)^2 + \left(\frac{\partial f}{\partial P_{vba}} \Delta P_{vba} \right)^2} \tag{8}$$

$$\frac{\partial f}{\partial P_{vx0}} = \frac{-(P_{vx} - P_{vx0})}{L_a} \left(\frac{P_{rba}}{P_{vba}} \right)^2 \approx 0.03 \tag{9}$$

$$\frac{\partial f}{\partial P_{vy0}} = \frac{-\left(P_{vy} - P_{vy0} + L \left(\frac{P_{vba}}{P_{rba}} \right) \right)}{L_a} \left(\frac{P_{rba}}{P_{vba}} \right)^2 \approx 0.04 \tag{10}$$

$$\frac{\partial f}{\partial L} = \frac{\left(P_{vy} - P_{vy0} + L \left(\frac{P_{vba}}{P_{rba}} \right) \right)}{L_a \left(\frac{P_{vba}}{P_{rba}} \right)} \approx 1 \tag{11}$$

$$\frac{\partial f}{\partial P_{rba}} = -L\left(\frac{P_{vba}}{P_{rba}^2}\right)\left|\frac{\partial f}{\partial P_{vy0}}\right| + \left(\frac{L_a}{P_{rba}}\right) \approx 0.3 \tag{12}$$

$$\frac{\partial f}{\partial P_{vba}} = L\left(\frac{1}{P_{rba}}\right)\left|\frac{\partial f}{\partial P_{vy0}}\right| + \left(\frac{-L_a}{P_{vba}}\right) \approx 0.03 \tag{13}$$

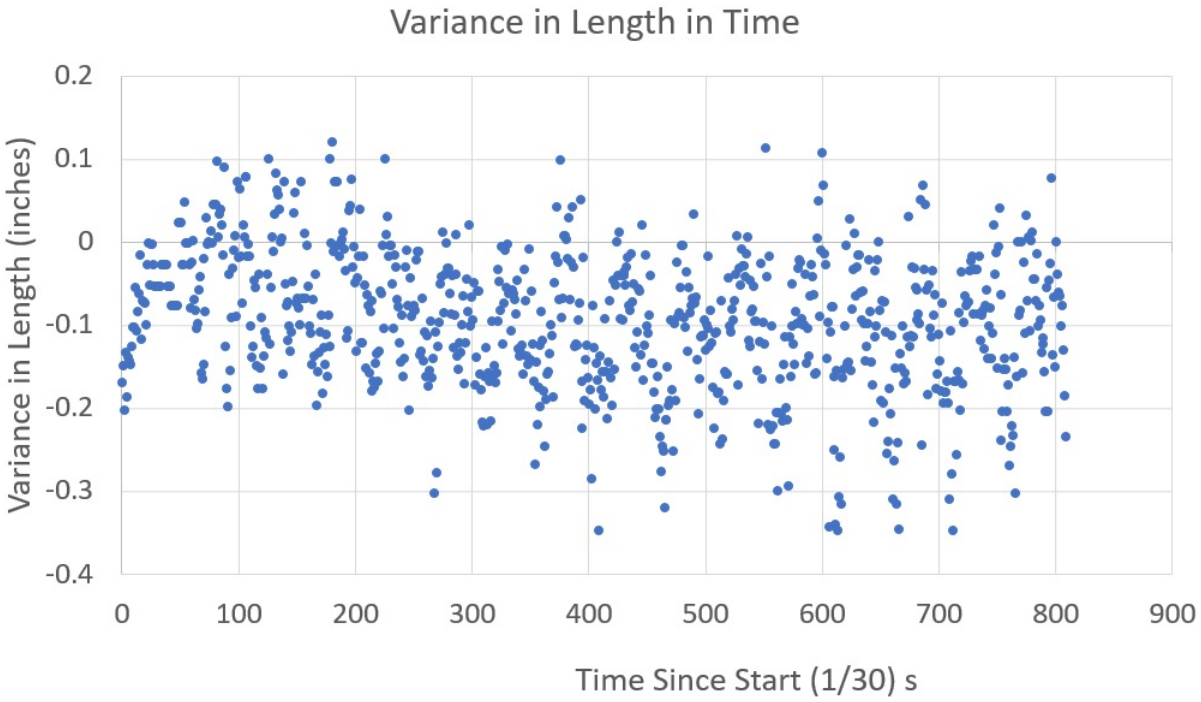
$$\Delta L_a \approx 0.535\text{inches} \tag{14}$$

Essentially, the half an inch uncertainty in the pendulum length dominates everything else.

11.5 Results

11.5.1 Accuracy

Accounting for the uncertainty in setup, roughly, we get an approximate uncertainty of about 0.544 inches. The data, meanwhile has a "standard deviation" of about 0.125 inches about the predicted value.



11.5.2 Precision

As can be seen from the former graph, the length does not have a true "standard deviation" about 48 inches. It varies closely around that, more tightly than the accuracy of the experiment would necessarily predict, but within that bound, it varies tightly about another point, which is the average length. The standard deviation is more appropriately defined about this average(which is about 47.906 inches), than some other value. This is our precision, which happens to be about 0.0823 inches.

11.6 Analysis

11.6.1 Uncertainty in Setup

One last point to touch on is the potential for error due to the camera angle.
If we take the screen resolution, multiply by our conversion factor to find it's real height we get about

59.93 inches. Since we know the pendulum is roughly 26 inches off the ground, and it appears at 239 pixels we can calculate the top of the screen to be 13.26 inches above the pendulum, therefore the bottom of the frame is 20.67 inches below the ground, and the midpoint of the screen is 9.3 inches above the ground. The camera is 16.1 inches from the ground. Thus, the screen drops 6.8 inches over 96 inches for an angle of about 4 degrees. This would introduce a scaling factor in the y direction of the cosine of that angle or about 0.25%.
That is, at 48 inches the scaling factor would modify the length by 0.1 inch, which is small compared to the rest of our uncertainty(notably our uncertainty in the length of the pendulum). If you play with these variables, it's easy to see how this small effect doesn't grow within our bounds of uncertainty, even though some of our uncertainty is considerably large.

11.7 Discussion

The easy answer is that our accuracy is within 0.544 inches and the precision is within 0.0823 inches, but what does this mean. Well, for one it means our experiment was a poor test of the limits of our system, because our experiment surpassed the limits of the test. By how much, we can't say for certain. However, I will make one argument to show that the precision is the more important number.

11.7.1 Limits of the Experiment

The biggest problem running the experiment was finding the marker against the background. Shutter speed plays into this heavily, but beyond that, color of the marker had a big effect on how easy it was to isolate the marker. Active markers were considerably easier to track, but passive markers were not impossible to track either. While all markers become somewhat harder to track at higher exposure times, passive markers become much more difficult than active markers do.
The other major consideration is the size of the marker. The larger the size of the marker, the easier it is to track, to a certain limit. Ideally the marker should occupy on the order of a few hundred pixels. Much less, as in this experiment, and you need low exposure times in order to eliminate motion blur. Much more, as in our first prototypes, and the program can't handle assembling the large markers.
I tracked the marker size, and noticed I was missing about 10% of the data points with a fairly lax size constraint. With better size constraints, noise becomes less of an issue, and the program can run faster. If the marker consistently registers every frame, even faster methods can be used to speed processing up. For reference, the marker size was limited to under 100 pixels in total, averaging 30.

11.8 Conclusion

Our system was as accurate as possibly determinable using this experiment, which is approximately 0.544 inches. It also had a precision of about 0.0823 inches which suggests further experimentation might be useful to confirm or deny whether or not this is truly accuracy. However, based on fundamental knowledge of how the system works, it is reasonable to predict that the precision is our true accuracy.

12 Usability Test

12.1 Experiment Objective

This project is designed to assist Orbital ATK in evaluating the performance of its deployable structures by tracking their motion. This is done by recording the motion of markers attached to the structures using a conventional visible light camera, and then using software to output the positions of the markers.

This test was conducted to evaluate the usability of the system, by allowing ATK technicians to perform a test with the assistance of the user manual and provide feedback.

12.2 Experiment Procedures

For this test, the following steps were performed in order so that the amount of time and concerns of the step could bee recorded.

Program Procedure:

1. Install markers
2. Recording test footage
3. Color Calibration for markers
4. Saving and loading configurations
5. Running program
6. Interpreting results

12.3 Evaluation Criteria

1. Can the ATK technicians successfully perform each procedure without assistance?
    - What sections need assistance?
    - How much assistance is needed?
2. How long does it take for the ATK technicians to successfully perform each step? a. Which steps take the longest?
    - Which steps take the longest?
    - Why did those steps take the longest?
3. Are any steps confusing or missing?
    - Which steps are confusing?
    - What steps need to be added?
4. The standard system usability test survey will be used to quantitatively determine the quality of the overall user experience.

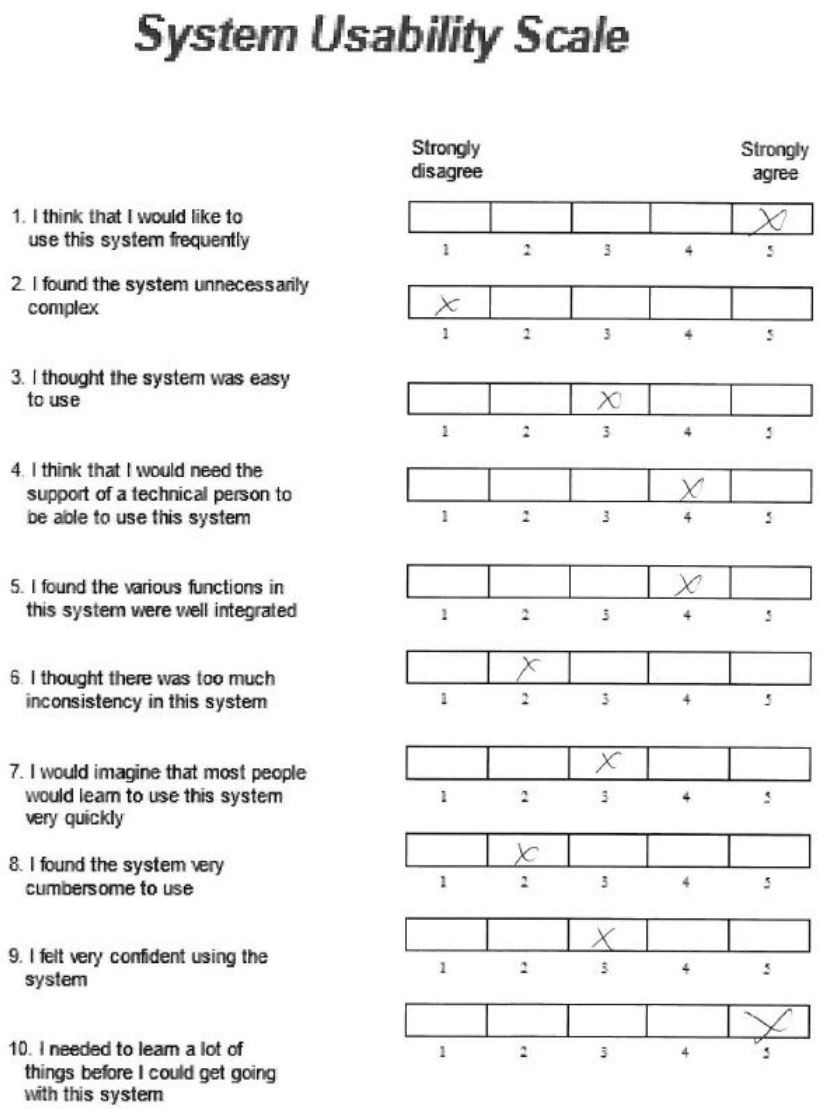
System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

12.4 Test Results

Time for each Step:

- Setting Up Camera and Markers - 32 mins
- Creating and inputting Calibration - 52 mins
- Confirm outpu - 8 mins
- Run Program - 12 mins



System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently					X
2. I found the system unnecessarily complex	X				
3. I thought the system was easy to use			X		
4. I think that I would need the support of a technical person to be able to use this system				X	
5. I found the various functions in this system were well integrated				X	
6. I thought there was too much inconsistency in this system		X			
7. I would imagine that most people would learn to use this system very quickly			X		
8. I found the system very cumbersome to use		X			
9. I felt very confident using the system			X		
10. I needed to learn a lot of things before I could get going with this system					X

12.5 Conclusions

ATK was very happy with the program; a new technician was able to setup the markers and camera, create input calibrations, and obtain an output. Several areas for improvement were identified, including:

- Layout of the User manual was adjusted to better reflect the workflow of the actual process. Steps need to appear in manual in the order as they should be performed and should be self-contained.
- Better clarification of how the marker should be attached to the test array and how the test footage should be recorded.
- Calibration routine took the longest and required the most assistance.
- Standard Usability Score was 67.5 which is comparable to the industry average of 68.

13 Multiple Marker Test

This test was to confirm the marker's ability to track at least ten points. Ten passive markers (orange tape) were attached to a pair of wheel and are shown in Figure 30.

Figure 30: Ten Marker Test

The wheels were rolled a short distance away from each-other, while recording their motion. The data was easily filtered of the minimal noise that was present, and then graphed. These are seen in Figure 31 and Figure 32.

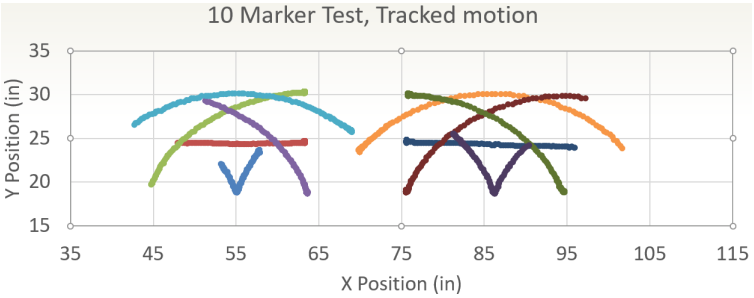


Figure 31: Ten Marker Test Graph Y vs X.

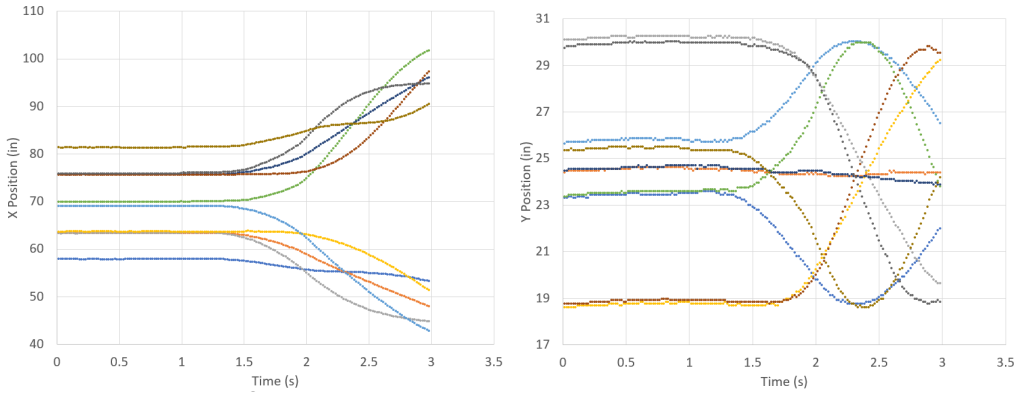


Figure 32: Ten Marker Test Graphs X vs time and Y vs time.

14 Documentation

14.1 Namespace Index

14.1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

14.2 Hierarchical Index

14.2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:
Form

14.3 Class Index

14.3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

14.4 Namespace Documentation

14.4.1 Capstone Namespace Reference

Classes

- class **Form1**
- class **Program**

14.5 Class Documentation

14.6 Capstone.Form1 Class Reference

Inherits Form.

Classes

- class **markerset**

  Stores multiple sets of data for markers.
- class **point**

  Stores data for marker points.

Public Member Functions

- **Form1** ()

  Builds GUI for program.
- bool **PixelCheck** (Color pixelColor)

  Evaluates whether a pixel is in the color range for a marker.
- bool **EvaluateSet** (int setNumber)

  Evaluates whether a set is the correct size.
- int **SetSize** (int setNumber)

  Evaluates the size of a set.
- int [] **Centroid** (int setNumber)

  Evaluates the centroid of a set.
- string **FindImageName** (int frameNumber, bool vidA)

  Finds the name of an image.
- string **FindCSVName** ()

  Finds the first nonexistent file for csv output.
- void **MatrixAllocation** ()

  Creates matrices used per video.
- void **MatrixCleaning** ()

  Cleans matrices for one frame step.
- void **SearchImage** (string fileLocation, int frameNumber, int secondNumber)

  Chooses and calls the full or fast search for an image.
- void **SetFormation** ()

  Forms sets of contiguous pixels.
- int **PointCount** ()

  Finds how many points are in pixelSets.
- void **CentroidOutput** (int timeValue)

  Stores results from a single frame for final output.
- void **CheckAt** (int x, int y, int fullSets)

  Checks if we should add the pixel and then the nearest pixels.
- void **CheckNear** (int x, int y, int fullSets)

  Checks if we should add the nearest pixels.
- void **AddPointToSearchMesh** (int x, int y)

  Adds point to nMatrix and mxnMatrix.

- void **MainFunction** ()

  Orchistrates the entire process.
- void **FrameStep** (int frameNumber, int secondNumber, bool vidA)

  Orchistrates the process for one frame.
- bool **SecondStep** (int timeNumber)

  Orchistrates the subprocess for one second.
- void **InputBMPFull** (string fileLocation)

  Scans the entire image for marker pixels.
- void **InputBMPFast** (string fileLocation, int frameNumber, int secondNumber)

  Scans the image based on previous marker points for marker pixels.
- void **TextBoxInputs** ()

  Inputs all the textbox inputs.
- void **TextBoxOutputs** ()

  Outputs all current variables to the textboxes.
- void **Outputcsv** ()

  Output the stored data to a csv file.
- void **BatFileGeneration** (int time, bool vidA)

  Writes a batch file to pull a set of frames from the video input.
- void **ImportSettings** (string fileName)

  Imports settings from the input file.
- void **ExportSettings** (string fileName)

  Exports settings from the input file.
- void **ManagePoints** (int frameNumber, int secondNumber)
- void **CombinePoints** (int frame)

  Combines points that are near each-other.
- void **SortPoints** (int frame)

  Sorts the points in outputData at a specified frame.
- bool **IfNearSet** (int frameA, int indexA, int frameB, int indexB)

  Determines whether two sets of points are near each other.
- void **MainCalibration** ()

  Helps calibration by showing RGB and HSB distributions.
- void **CalibrationAllocation** ()

  Allocates matrices for calibration.
- void **InputBMPCalibration** (string fileLocation)

  Records colors from an image into matrices for RGB and HSB.
- void **MxNCalibration** ()

  Filters the mxnMatrix of all pixels that do not match Lock criteria.
- void **OutputRegion** ()

  Outputs filtered color data for selected region to separate csv files.
- void **OutputStatistics** ()

  Outputs statistics about RGB and HSB composition to calbration-statistics.csv.

Public Attributes

- int **xResolution** = 1920

  Horizontal resolution of the video.
- int **yResolution** = 1080

  Vertical resolution of the video.
- int **frameRate** = 60

  Frame rate of the video.
- decimal **startTime** = 0

  Start time, from beginning of video, in seconds.
- decimal **endTime** = 1

  End time, from beginning of video, in seconds.
- int **upperSetCount** = 1000

  Maximum number of sets of pixels allowed per frame.
- int **lowerSetSize** = 1

  Minimum size for a set of pixels to be accepted.
- int **upperSetSize** = 200

  Maximum size for a set of pixels to be accepted.
- int **fastSearchRange** = 100

  Range from previous points a fast search will scan.
- **point** [] **nMatrix**

  List of pixels with the correct marker color; updates per frame.
- int **nMatrixSize** = 0

  Bookmark variable to store where we are in the nMatrix.
- bool [,] **mxnMatrix**

  Stores whether each pixel is being considered as a marker pixel.
- **markerset** [] **pixelSets**

  Stores data for sets of contiguous pixels, within a frame.
- **markerset** [] **outputData**

  Stores data for final output.
- string **folderPath** = ""

  Path of Data folder, used for all file navigation.
- string **videoFileName** = ""

  Name of video file being accessed.
- string **settingsFileName** = ""

  Name of settings file being accessed.
- bool **ifRGB** = false

  If RGB is being used to process colors, as opposed to HSB.
- decimal **redLow** = 0

  Minimum Red value, lowest is 0.

- decimal **redHigh** = 255

  Maximum Red value, highest is 255.
- decimal **greenLow** = 0

  Minimum Green value, lowest is 0.
- decimal **greenHigh** = 255

  Maximum Green value, highest is 255.
- decimal **blueLow** = 0

  Minimum Blue value, lowest is 0.
- decimal **blueHigh** = 255

  Maximum Blue value, highest is 255.
- decimal **hueLow** = 0

  Minimum Hue value, lowest is 0.
- decimal **hueHigh** = 255

  Maximum Hue value, highest is 255.
- decimal **saturationLow** = 0

  Minimum Saturation value, lowest is 0.
- decimal **saturationHigh** = 255

  Maximum Saturation value, highest is 255.
- decimal **brightnessLow** = 0

  Minimum Brightness value, lowest is 0.
- decimal **brightnessHigh** = 255

  Maximum Brightness value, highest is 255.
- int **leftBounding** = 0

  Left position of bounding box for calibration.
- int **topBounding** = 0

  Top position of bounding box for calibration.
- int **widthBounding** = 1

  Width of bounding box for calibration.
- int **heightBounding** = 1

  Height of bounding box for calibration.
- bool **rLock** = false

  Whether or not we are happy with the red calibration.
- bool **gLock** = false

  Whether or not we are happy with the green calibration.
- bool **bLock** = false

  Whether or not we are happy with the blue calibration.
- bool **hLock** = false

  Whether or not we are happy with the hue calibration.
- bool **sLock** = false

  Whether or not we are happy with the saturation calibration.

- bool **lLock** = false

  Whether or not we are happy with the brightness calibration.
- int [,] **redMatrix**

  Red Matrix for calibration.
- int [,] **greenMatrix**

  Green Matrix for calibration.
- int [,] **blueMatrix**

  Blue Matrix for calibration.
- int [,] **hueMatrix**

  Hue Matrix for calibration.
- int [,] **saturationMatrix**

  Saturation Matrix for calibration.
- int [,] **brightnessMatrix**

  Brightness Matrix for calibration.
- double **range** = 84

  Distance from camera to set-up; used for pixel-real space conversion.
- double **slopeCalibration** = 0.0006038

  Slope used for pixel-real space conversion.
- double **interceptCalibration** = -0.0004014

  Intercept used for pixel-real space conversion.
- bool **noPoints** = false

  If there are no points in the frame.

Protected Member Functions

- override void **Dispose** (bool disposing)

  Clean up any resources being used.

14.6.1 Member Function Documentation

AddPointToSearchMesh() `void Capstone.Form1.AddPointToSearchMesh (`
`int x,`
`int y ) [inline]`

Adds point to nMatrix and mxnMatrix.

If the point specified with the two integer inputs isn't in the mxnMatrix, which would imply we have already added it, add it to the mxnMatrix, and the nMatrix, and increment nMatrixSize.

Parameters

x	X position of the pixel being added.
y	Y position of the pixel being added.

BatFileGeneration() `void Capstone.Form1.BatFileGeneration (`
`int` *time*,
`bool` *vidA* `) [inline]`

Writes a batch file to pull a set of frames from the video input.

The first input (integer) sets the start time for pulling 1 second worth of frames from the video input. The second input (bool) sets whether to use naming scheme A (frameA-) for vidA = true, or B (frameB-) for vidA = false. Outputs a .bat file that will open ffmpeg, and call the appropriate command.

Parameters

time	Time value for current frame.
vidA	The bool for the type of frame naming scheme used.

CalibrationAllocation() `void Capstone.Form1.CalibrationAllocation ( ) [inline]`

Allocates matrices for calibration.

Creates mxnMatrix, redMatrix, greenMatrix, blueMatrix, hueMatrix, saturationMatrix, and brightness↩ Matrix.

Centroid() `int [] Capstone.Form1.Centroid (`
`int` *setNumber* `) [inline]`

Evaluates the centroid of a set.

Given an integer index, evaluates the mean x and y position of a set. Returns the resulting x and y position as int[2], or 0,0 if the set's size is 0.

Parameters

setNumber	The index of the set being evaluated.

CentroidOutput() `void Capstone.Form1.CentroidOutput (`
`int` *timeValue* `) [inline]`

Stores results from a single frame for final output.

Finds number of points in pixelSets using PointCount, allocates space in outputData for these, calculates the centroid for each set using Centroid, then stores the centroid and set size (calculated using SetSize) in outputData.

Parameters

timeValue	Index for the time value for the final outputs.

CheckAt() `void Capstone.Form1.CheckAt (`
`int x,`
`int y,`
`int fullSets ) [inline]`

Checks if we should add the pixel and then the nearest pixels.

If the pixel specified by the first and second integer inputs is in the mxnMatrix, checks the size of the set specified by the third integer input, then adds the pixels location to that position in that set. If the set is full this will overwrite the last point. Then calls CheckNear to check the nearest points.

Parameters

x	X position of the pixel being checked.
y	Y position of the pixel being checked.
fullSets	Index of the set being processed.

CheckNear() `void Capstone.Form1.CheckNear (`
`int x,`
`int y,`
`int fullSets ) [inline]`

Checks if we should add the nearest pixels.

Without checking out-of-bounds points, calls CheckAt for the pixels below, above, to the right, and to the left of the location specified with the first two integer inputs, and maintains the third integer input as the current set.

Parameters

x	X position of the pixel being checked.
y	Y position of the pixel being checked.
fullSets	Index of the set being processed.

CombinePoints() `void Capstone.Form1.CombinePoints (`
`int frame ) [inline]`

Combines points that are near each-other.

Looks at all pairs of points, and combines any that are near to each-other, setting the other's size to 0.

Parameters

frame	Frame being analyzed.

Dispose() `override void Capstone.Form1.Dispose (`

```
bool disposing ) [inline], [protected]
```

Clean up any resources being used.

Parameters

disposing	true if managed resources should be disposed; otherwise, false.

EvaluateSet() `bool Capstone.Form1.EvaluateSet (`

```
int setNumber ) [inline]
```

Evaluates whether a set is the correct size.

Given an integer index, tests whether a set's size is correct according to lowerSetSize and upper↩ SetSize (adjusting index to start at 1). Returns true if set is within the range, and false otherwise.

Parameters

setNumber	The index of the set being evaluated.

ExportSettings() `void Capstone.Form1.ExportSettings (`

```
string fileName ) [inline]
```

Exports settings from the input file.

Exports the variables at the file at the specified input string, using four intuitive categories: HSB includes variables: hueLow, hueHigh, saturationLow, saturationHigh, brightnessLow and brightness↩ High. RGB includes variables: redLow, redHigh, greenLow, greenHigh, blueLow and blueHigh. Size includes variables: lowerSetSize, upperSetSize, upperSetCount, and fastSearchRange. Time includes variables: startTime and endTime. Misc includes variables: ifRGB and range. Camera includes variables: xResolution, yResolution, frameRate, slopeCalibration, and interceptCalibration. The attribute name for each variable is identical to the variable name.

Parameters

fileName	The location of the settings file being exported.

FindCSVName() `string Capstone.Form1.FindCSVName ( ) [inline]`

Finds the first nonexistent file for csv output.

Searches 2,147,483,646 possible names for an output file in the scheme of "...\Data\output-123.↩ csv". If none exist, use "...\Data\TMarks.csv"

FindImageName() `string Capstone.Form1.FindImageName (`

```
int frameNumber,
bool vidA ) [inline]
```

Finds the name of an image.

The first input(integer) determines the format for the numbering such that it has three characters, ie "-123". Then determines the remaining format based on the second input(bool), ie ".../↩ Data/frameA-123.bmp". Warning. Highest possible frameRate is 10^9-1.

Parameters

frameNumber	The frame number of the imaged.
vidA	The bool for the type of frame naming scheme used.

FrameStep() `void Capstone.Form1.FrameStep (`
`int` *frameNumber,*
`int` *secondNumber,*
`bool` *vidA* `) [inline]`

Orchistrates the process for one frame.

Uses the first input as the frame number, the second input as the second number, and the third input as boolean for vidA (used in SearchImage). Finds the file address, then runs MatrixCleaning, SearchImage, SetFormation, and CentroidOutput, which scans each frame, sorts the pixels into sets, and stores those sets into the final output matrix.

Parameters

frameNumber	Frame number of the frame being used.
secondNumber	Second number of the frame being searched.
vidA	The bool for the type of frame naming scheme used.

IfNearSet() `bool Capstone.Form1.IfNearSet (`
`int` *frameA,*
`int` *indexA,*
`int` *frameB,*
`int` *indexB* `) [inline]`

Determines whether two sets of points are near each other.

First determines if either point is of zero size, and if so, returns false. Then calculates the distance between the two points. If it is less than the fastSearchRange, returns true, otherwise false.

Parameters

frameA	The frame for the first point
indexA	The index for the first point
frameB	The frame for the second point
indexB	The index for the second point

ImportSettings() `void Capstone.Form1.ImportSettings (`
`string` *fileName* `) [inline]`

Imports settings from the input file.

Reads the file at the specified input string, using four intuitive categories: HSB includes variables: hueLow, hueHigh, saturationLow, saturationHigh, brightnessLow and brightnessHigh. RGB includes variables: redLow, redHigh, greenLow, greenHigh, blueLow and blueHigh. Size includes variables: lowerSetSize, upperSetSize, upperSetCount, and fastSearchRange. Time includes variables: startTime and endTime. Camera includes variables: xResolution, yResolution, frameRate, slopeCalibration, and interceptCalibration. Misc includes variables: ifRGB and range. The attribute name for each variable is identical to the variable name.

Parameters

fileName	The location of the settings file being imported.

InputBMPCalibration() `void Capstone.Form1.InputBMPCalibration (`
`string` *fileLocation* `) [inline]`

Records colors from an image into matrices for RGB and HSB.

Stores the red, green, blue, hue, saturation, and brightness values of every pixel from an image into the respective redMatrix, greenMatrix, blueMatrix, hueMatrix, saturationMatrix, and brightnessMatrix. Auto-scales all values from 0 to 255 the appropriate internal bounds. Finally, sets mxnMatrix to true for all pixels.

Parameters

fileLocation	The location of the file being exported.

InputBMPFast() `void Capstone.Form1.InputBMPFast (`
`string` *fileLocation,*
`int` *frameNumber,*
`int` *secondNumber* `) [inline]`

Scans the image based on previous marker points for marker pixels.

Opens the image file using the first input (string) and forms a range to scan for each previous marker pixel. This is a square centered around the pixel with side-length equal to fastSearchRange + 1. This square is truncated if the range extends out of bounds, at the edge of the frame. The color for each pixel in each range is obtained and checked via PixelColor. If it is a marker color, adds to the appropriate matrices using AddPointToSearchMesh.

Parameters

fileLocation	File location being searched.
frameNumber	Frame number of the frame being used.
secondNumber	Second number of the frame being searched.

InputBMPFull() `void Capstone.Form1.InputBMPFull (`
`string` *`fileLocation`* `) [inline]`

Scans the entire image for marker pixels.

Opens the image file using the string input, and scans throughout the range specified by x↩ Resolution and yResolution. Gets the color of each pixel, and if it is the right color, via PixelColor, adds the pixel to the appropriate matrices using AddPointToSearchMesh.

Parameters

fileLocation	File location being searched.

MainCalibration() `void Capstone.Form1.MainCalibration ( ) [inline]`

Helps calibration by showing RGB and HSB distributions.

Calls TextBoxInputs, calls Input

MainFunction() `void Capstone.Form1.MainFunction ( ) [inline]`

Orchistrates the entire process.

Starts with progressBar1.Value at 0, then calls TextBoxInputs, then MatrixAllocation, then for every second runs SecondStep and for every frame within that, runs FrameStep, and evaluates the percentage of frames processed and sets the progressBar1.Value to this. Finally, runs Outputcsv and, if successful, sets progressBar1.Value to 0. If it catchs IOException, sets progressBar1.Value to 100, indicating the output was not successful.

ManagePoints() `void Capstone.Form1.ManagePoints (`
`int` *`frameNumber,`*
`int` *`secondNumber`* `) [inline]`

Runs CombinePoints and SortPoints.

Parameters

frameNumber	Frame number of the frame being used.
secondNumber	Second number of the frame being searched.

MatrixAllocation() `void Capstone.Form1.MatrixAllocation ( ) [inline]`

Creates matrices used per video.

Allocates space for pixelSets, mxnMatrix, nMatrix, and outputData matrices.

MatrixCleaning() `void Capstone.Form1.MatrixCleaning ( ) [inline]`

Cleans matrices for one frame step.

Sets zeros in used values of nMatrix, based on nMatrixSize, then resets nMatrixSize, then zeros all values in pixelSets.

MxNCalibration() `void Capstone.Form1.MxNCalibration ( ) [inline]`

Filters the mxnMatrix of all pixels that do not match Lock criteria.

For each rLock, gLock, bLock, hLock, sLock, lLock, if true, filters pixels according to the bounding critera: ie, for redMatrix lower than redLow OR higher than redHigh set the respective mxnMatrix to false. Uses redMatrix, greenMatrix, blueMatrix, hueMatrix, saturationMatrix, and brightnessMatrix for pixel values to be filtered, and redLow, redHigh, greenLow, greenHigh, blueLow, blueHigh, hue↩ Low, hueHigh, saturationLow, saturationHigh, brightnessLow, and brightnessHigh for filter settings.

Outputcsv() `void Capstone.Form1.Outputcsv ( ) [inline]`

Output the stored data to a csv file.

Finds a file name using findCSVName, outputs a header line, and one line per marker per frame including the time, calculated from the startTime, endTime, and frameRate, the x position, y position, slopeCalibration, interceptCalibration, and the size of the marker.

OutputRegion() `void Capstone.Form1.OutputRegion ( ) [inline]`

Outputs filtered color data for selected region to separate csv files.

Uses leftBounding, topBounding, widthBounding, and heightBounding bounds modified to prevent exceeding the borders of the image to output the RGB or HSB color data to csv files title "calibration-red.csv" or "calibration-blue.csv" etc.

OutputStatistics() `void Capstone.Form1.OutputStatistics ( ) [inline]`

Outputs statistics about RGB and HSB composition to calbration-statistics.csv.

Outputs columns that show the number of points in the total and selected region that have valid points of various values of a specific color type, excluding all colored types locked in using rLock, gLock, bLock, hLock, sLock, and lLock.

PixelCheck() `bool Capstone.Form1.PixelCheck (`
`Color` *`pixelColor`* `) [inline]`

Evaluates whether a pixel is in the color range for a marker.

Tests either RGB or HSB based on ifRGB. If ifRGB is true, tests whether the pixel is in range of the redLow, redHigh, greenLow, greenHigh, blueLow, and blueHigh values, accordingly. If ifRGB is false, tests whether the pixel is in range of the hueLow, hueHigh, saturationLow, saturationHigh, brightnessLow, and brightnessHigh values, accordingly. Returns true if pixel colors are in range, and false otherwise.

Parameters

pixelColor	The color input for the pixel being checked

PointCount() `int Capstone.Form1.PointCount ( ) [inline]`

Finds how many points are in pixelSets.

Looks for the first empty set in pixelSets and returns the index before that.

SearchImage() `void Capstone.Form1.SearchImage (`

```
string fileLocation,
int frameNumber,
int secondNumber ) [inline]
```

Chooses and calls the full or fast search for an image.

If frameNumber is a whole multiple of frameRate, use the full search, otherwise use the fast search.

Parameters

fileLocation	File location being searched.
frameNumber	Frame number of the frame being used.
secondNumber	Second number of the frame being searched.

SecondStep() `bool Capstone.Form1.SecondStep (`

```
int timeNumber ) [inline]
```

Orchistrates the subprocess for one second.

Uses the first input as the second number. If the second number is odd, uses the "A" naming scheme (vidA = true), otherwise uses the "B" naming scheme (vidA = false). Then runs BatFile↩ Generation. Next, tries to delete all image files to be used in current cycle, before running the bat file at "\Data\ffmpeg\Command.bat". If file in use, waits before trying again. This will pull all the frames within the second and save them according to the naming scheme from FindImageName. When it runs Command.bat, it does so without a window. Returns vidA.

Parameters

timeValue	Index for the time value for the final outputs.

SetFormation() `void Capstone.Form1.SetFormation ( ) [inline]`

Forms sets of contiguous pixels.

For each point in the nMatrix runs CheckAt, which combines all the nearby pixels into a set. Then, checks the size using EvaluateSet. If it's good, increments the set count. Otherwise, erases the invalid set. Skips points in nMatrix which aren't in mxnMatrix.

SetSize() `int Capstone.Form1.SetSize (`

```
int setNumber ) [inline]
```

Evaluates the size of a set.

Given an integer index, finds the last non-empty point of a pixelSet, and returns that index as the size. Otherwise, returns upperSetSize.

Parameters

setNumber	The index of the set being evaluated.

SortPoints() `void Capstone.Form1.SortPoints ( int frame ) [inline]`

Sorts the points in outputData at a specified frame.

Stores the data from outputData into temporary storage, checks every pair of points across this frame and the previous frame to see if they are close to each-other. If so, stores the point at that index. If no point is found, stores 0's in all values. Once all previous points have been checked, adds the remaining non-empty points to the back of the outputData for the new frame.

Parameters

frame	The frame for the set of points

TextBoxInputs() `void Capstone.Form1.TextBoxInputs ( ) [inline]`

Inputs all the textbox inputs.

Inputs the folderPath, videoFileName and settingsFileName directly from the appropriate textboxes, and tries to parse the huelow, hueHigh, brightnessLow, brightnessHigh, saturationLow, saturation↩ High, redLow, redHigh, blueLow, blueHigh, greenLow, greenHigh, startTime, endTime, frame↩ Rate, lowerSetSize, upperSetSize, upperSetCount, xResolution, yResolution, leftBounding, top↩ Bounding, widthBounding, heightBounding, rLock, gLock, bLock, hLock, sLock, lLock, ifRGB, slopeCalibration, and interceptCalibration variables.

TextBoxOutputs() `void Capstone.Form1.TextBoxOutputs ( ) [inline]`

Outputs all current variables to the textboxes.

Outputs folderPath, videoFileName, settingsFileName, huelow, hueHigh, brightnessLow, brightness↩ High, saturationLow, saturationHigh, redLow, redHigh, blueLow, blueHigh, greenLow, greenHigh, startTime, endTime, frameRate, lowerSetSize, upperSetSize, upperSetCount, xResolution, yResolution, fastSearchRange, ifRGB, leftBounding, topBounding, widthBounding, heightBounding, rLock, g↩ Lock, bLock, hLock, sLock, lLock, slopeCalibration, and interceptCalibration to the textboxes or check boxes as appropriate.

The documentation for this class was generated from the following files:

- C:/Users/JohnLubran/Source/Repos/MyFirstProject/Capstone/Capstone/Form1.cs
- C:/Users/JohnLubran/Source/Repos/MyFirstProject/Capstone/Capstone/Form1.Designer.cs

14.7 Capstone.Form1.markerset Class Reference

Stores multiple sets of data for markers.

Public Member Functions

- **markerset** (int size)

Properties

- **point** [ ] **marker** `[get, set]`

  Points included in this set.

14.7.1 Detailed Description

Stores multiple sets of data for markers.

Contains multiple points, which use the point class. The number of points included is specified with an integer when the constructor is called. This class is used to store marker point data during each frame process step and the marker locations stored for final output.

14.7.2 Constructor & Destructor Documentation

markerset() `Capstone.Form1.markerset.markerset ( int size ) [inline]`

Initializes the set of points.

Parameters

size	The number of points in this set.

The documentation for this class was generated from the following file:

- C:/Users/JohnLubran/Source/Repos/MyFirstProject/Capstone/Capstone/Form1.cs

14.8 Capstone.Form1.point Class Reference

Stores data for marker points.

Properties

- int **xValue** `[get, set]`

  X position for this point, or set of points.
- int **yValue** `[get, set]`

  Y position for this point, or set of points.
- int **size** `[get, set]`

  Number of points in this set, if this is a set.

14.8.1 Detailed Description

Stores data for marker points.

Contains three integers which represent the x and y position and the size of a market point. This class is used to store marker point data during each frame process step, the marker locations stored for final output.

The documentation for this class was generated from the following file:

- C:/Users/JohnLubran/Source/Repos/MyFirstProject/Capstone/Capstone/Form1.cs