
VideoF2B

Alberto Solera

Apr 27, 2024

TABLE OF CONTENTS

| | | |
|-----------|--|-----------|
| 1 | Installing VideoF2B | 3 |
| 2 | The User Interface | 5 |
| 3 | Ways to use VideoF2B | 7 |
| 3.1 | Basic | 7 |
| 3.2 | Augmented Reality | 8 |
| 4 | Field setup | 9 |
| 5 | Placing the Camera | 11 |
| 5.1 | General Procedure | 11 |
| 5.2 | Determining Camera Distance | 12 |
| 6 | Producing Basic videos | 15 |
| 7 | Producing Augmented-Reality Videos | 19 |
| 8 | Camera Calibration | 21 |
| 8.1 | Obtain the Calibration Pattern | 21 |
| 8.2 | Record the Video | 21 |
| 8.3 | Process the Video | 23 |
| 9 | Loading a Flight | 27 |
| 10 | Locating a Flight | 29 |
| 11 | The World Coordinate System | 33 |
| 12 | User Controls | 35 |
| 12.1 | Pausing/Resuming Processing | 36 |
| 12.2 | Clearing the Trace | 36 |
| 12.3 | Manipulating the Flight Hemisphere | 36 |
| 12.4 | Displaying Nominal Figures | 37 |
| 12.5 | Displaying Start/End Points | 37 |
| 12.6 | Displaying Diagnostic Points | 38 |
| 13 | Cheatsheet of User Controls | 41 |
| 14 | Getting help | 43 |
| 15 | FAQ | 45 |

| | |
|---|-----------|
| 16 Glossary | 47 |
| 17 videof2b | 49 |
| 17.1 videof2b package | 49 |
| 18 Guide for Editors of This Guide | 81 |
| Python Module Index | 83 |
| Index | 85 |

Hello everyone!

VideoF2B is a **desktop application** that draws figures in videos of **Control Line F2B** stunt flights.

To get started, navigate the **table of contents** or follow one of these links:

If you find any errors, omissions, opportunities for improvement, etc., let us know via [email](#) or [GitHub](#)!

INSTALLING VIDEOF2B

VideoF2B is a single-file executable. Installation is simple: just download the appropriate file for your operating system.

Choose instructions below based on your operating system:

1. Download the latest `VideoF2B.exe` from [here](#).
 2. Move the file to any folder of your choice.
 3. Run the application by double-clicking it.
 4. Enjoy!
-
1. Download the latest `videof2b` binary from [here](#).
 2. Move the file to any directory of your choice.
 3. Run the application.
 4. Enjoy!

Having trouble with installation? See [Getting help](#) or [FAQ](#) for guidance.

THE USER INTERFACE

VideoF2B processes one video at a time in the main window.

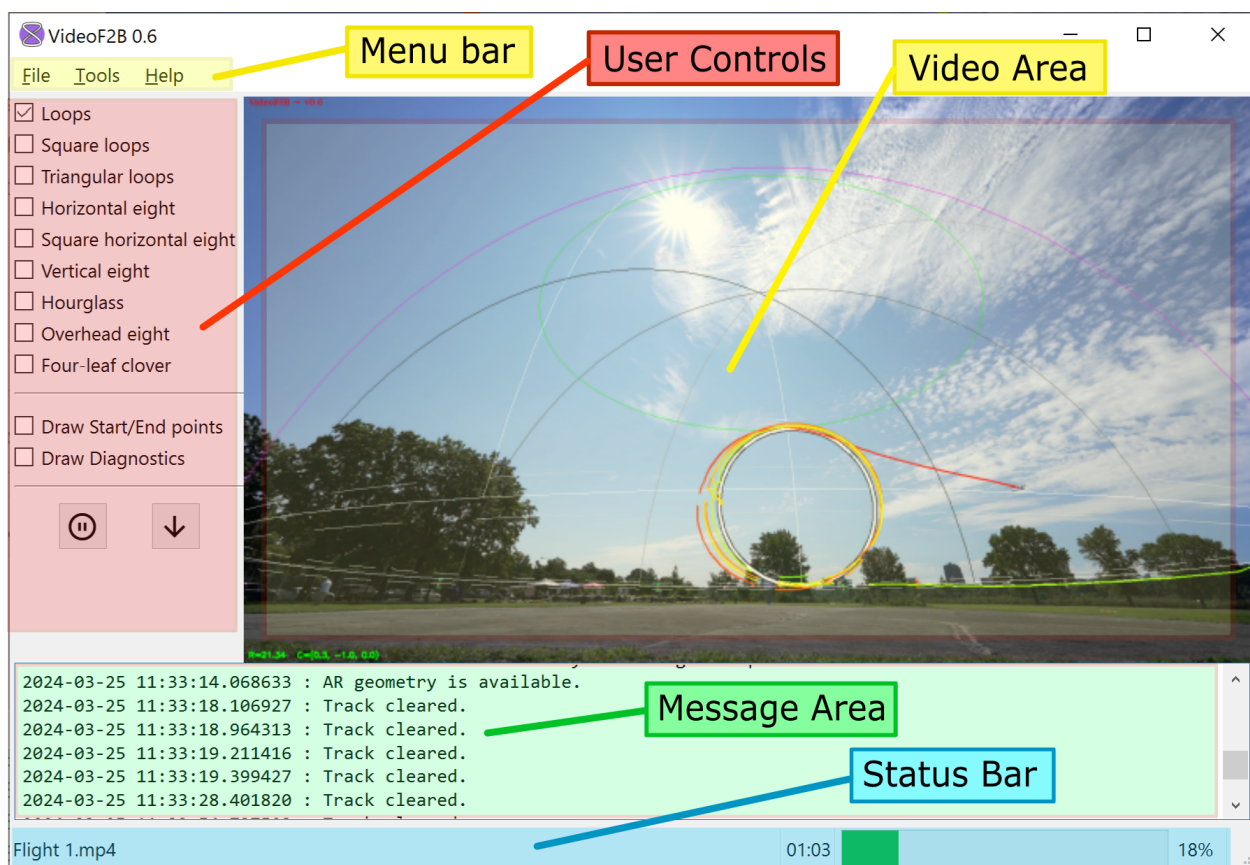


Fig. 2.1: The main VideoF2B window

The general areas of the main application window are as follows:

- The **menu bar** at the top provides access to general operations in VideoF2B. Use the *File* menu to load videos for processing. The *Tools* menu provides useful tools and calculators.
- **User controls** are enabled when processing *Augmented-Reality* videos. This area is disabled when processing *Basic* videos.
- The **video area** is the largest portion of the main window. It displays the video that is being processed.

- Messages to the user are displayed in the **message area** below the video and user controls. Every message is time-stamped with the local date and time of the user's computer.
- The **status bar** along the bottom displays the name of the loaded video file, occasional instructions, the elapsed time in the video, and a progress bar.

WAYS TO USE VIDEOF2B

The most prominent feature of VideoF2B is the drawing, or **tracing**, of a path behind a Control Line aircraft in video. These traces help us to visualize the figures that a Stunt pilot performs during a flight.

There are two general ways you can use VideoF2B to produce videos: *Basic*, and *Augmented Reality*.

3.1 Basic

This is the simplest use of VideoF2B. The result is a video where the path of the aircraft is traced with a colored line. No additional geometry is drawn.

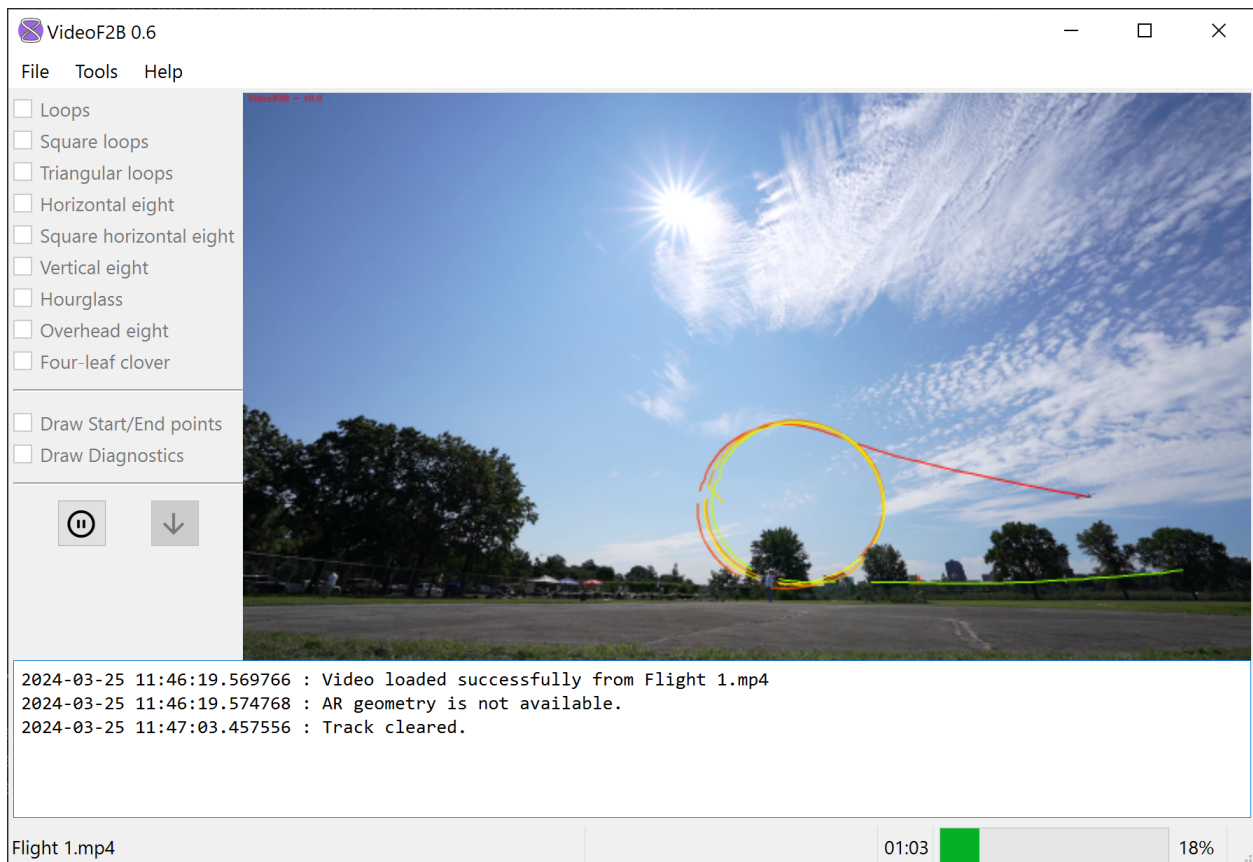


Fig. 3.1: Example of Basic video production in progress.

To learn how to produce Basic video, see *Producing Basic videos*.

3.2 Augmented Reality

In this mode, VideoF2B draws the traced path as well as reference geometry that includes a wireframe of the *flight hemisphere* and all F2B Stunt *figures* of the correct shape and size per the current FAI rules.

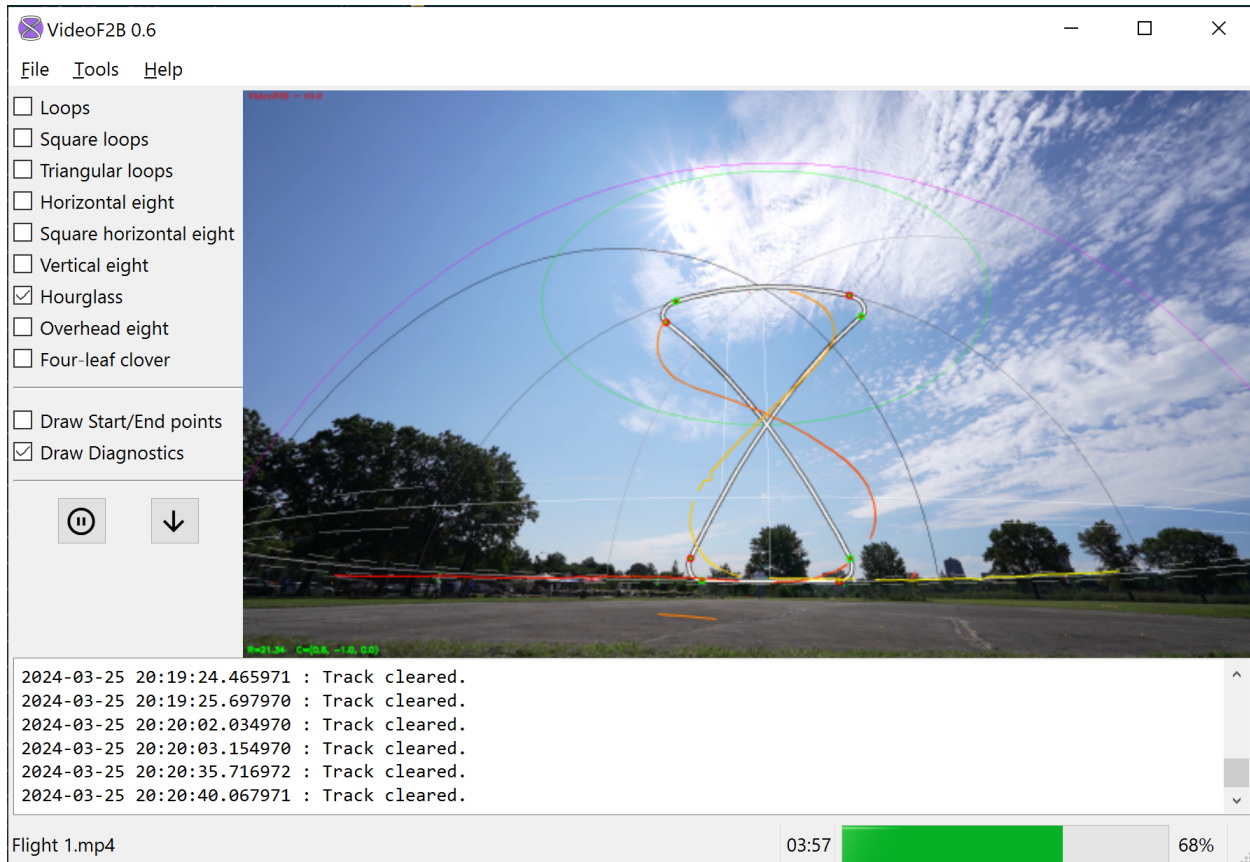


Fig. 3.2: Example of AR video production in progress.

To learn how to produce Augmented Reality video, see *Producing Augmented-Reality videos*.

FIELD SETUP

Before you go to the field, please read *Placing the Camera* to learn how to place the camera correctly in the field for best results.

Important: DO NOT record the videos handheld!

ALWAYS mount the camera to a sturdy tripod or similar.

DO NOT move or adjust the camera setup while recording a flight.

To produce *Basic videos* of flights, you only need a suitable camera and a tripod.

TODO: photo of an example setup for basic video.

To produce *AR videos*, more effort may be required. If your flying site already has FAI F2B markers installed around the *flight circle*, then the surveying work is already done. Just measure the distance from circle center to the markers. The elevation of the markers above the circle center should be 1.5 meters in that case.

TODO: photo of an example setup for AR video.

If your field does not have F2B markers, you can install them yourself with some specialized equipment. You will need at least a self-leveling rotary laser system and a laser distance measuring tool.

TODO: perhaps a separate chapter on how to DIY field markers, the recommended layout tools and technique, etc.???

The recommended dimensions and placement of markers are described in **Annex 4F, Appendix II** of the *FAI Sporting Code*.

PLACING THE CAMERA

Camera placement is important for capturing quality video of a Stunt flight. While it is acceptable to place the camera farther than recommended and capturing the entire *flight hemisphere*, it is generally preferable to capture as much of the core of *maneuver* space as possible by placing the camera closer. This approach sacrifices the outer edges of the *base*; but takeoff, level/inverted flight, and landing maneuvers are not easy to evaluate in video anyway. The other maneuvers should be the primary focus of video recording.

5.1 General Procedure

1. Select a location outside the *flight circle* upwind of the expected maneuvers. This is typically where the contest judges stand. The correct distance of the camera from the center of the circle depends on the focal length of your optical system. Details are discussed *below*.
2. Deploy your tripod at the selected location. Weigh it down if possible so that it remains stable.
3. Mount the camera on the tripod. The result video must be in landscape orientation. If using a mobile device, this means that you must orient the device horizontally.
4. Adjust the tripod so that the camera height is approximately between 1.0—1.5 m (about 3—5 ft).
5. Turn on the camera and make sure it is in video mode. In photo mode the aspect ratio of the image frame will likely be different from that of the video, resulting in incorrect alignment.
6. Point the camera approximately at the center of the circle.
7. Tilt the camera upward so that there is a visible margin between the pilot's feet at the center of the circle and the bottom edge of the frame. During this adjustment make sure that the camera is *level*. Most modern cameras have a built-in leveling guide—take advantage of it.
8. Pan the camera so that the frame's vertical centerline aligns with the center of the flight circle.

When the above steps are followed, you will find that the top of the flight hemisphere is near the top of the frame in your AR videos. You will also generally find that the center of the frame points somewhat above the 45° elevation at the far side of the hemisphere. This is usually the desired outcome.

5.2 Determining Camera Distance

When you don't know the focal length of your camera system, the camera distance must be determined by trial and error in the field. However, if you know your system's focal length, we recommend that you use this [Field of View calculator](#) to determine your camera system's **angle of view**. Look for the value labeled "Height" in degrees, in the section "Angle of View":

| Field of View Calculator | | Field Dimension | Angle of View |
|---|---------------------------------------|--|--|
| Focal Length | <input type="text" value="14"/> mm | Width Dimension | Width Degrees |
| Field Distance | <input type="text" value="28"/> units | Height Dimension | Height Degrees |
| Any units for Distance, feet or meters. Dimension results are the same units . Ft' In" | | Diagonal Dimension | Diagonal Degrees |
| Show field size at 2nd distance (like at the background?) | | Show 2nd distance at <input type="text" value="30"/> units | |
| | | 2nd W×H is 77.1 × 43.4 units | |
| Option 1: Sensor 36 × 20.25 mm, Diagonal 41.305 mm Aspect Ratio 16:9 (1.7778:1, Crop Factor 1.048x (native 1x) Magnification at 28 feet: 0.00164 (1:610). At 28 meters: 0.0005 (1:2000) The 14 mm lens view is 35 mm film Equivalent focal length 14.67 mm. | | | <div style="border: 1px solid orange; border-radius: 10px; padding: 5px; display: inline-block;">ReCompute</div> |
| <input checked="" type="radio"/> | Native Sensor Size, Width | <input type="text" value="36"/> mm | Option 1 Aspect ratio uses |
| <input type="radio"/> | Native Sensor Size, Height | <input type="text" value="24"/> mm | 16:9 crop in this sensor ▾ |

Fig. 5.1: FOV calculator (via scantips.com).

If documentation for your lens is available, verify that your result is reasonably close to the manufacturer's listed specifications.

VideoF2B includes a calculator for estimating the camera distance from circle center that will provide the best video coverage. To use it, choose *Tools* → *Place camera..* in the main menu:

Hint: Hover the mouse cursor over the values in the tables for detailed explanations of each value.

Enter the input values to the best of your knowledge:

- The flight radius R is the distance from the pilot's chest to the centerline of the aircraft.
- The camera height C is relative to the flight base. For example, if the camera is 1 m above the pilot's feet, then C = -0.5.
- Ground level G is also relative to the flight base. Under F2B rules, this value in meters is -1.50 and there should be no reason to adjust it.

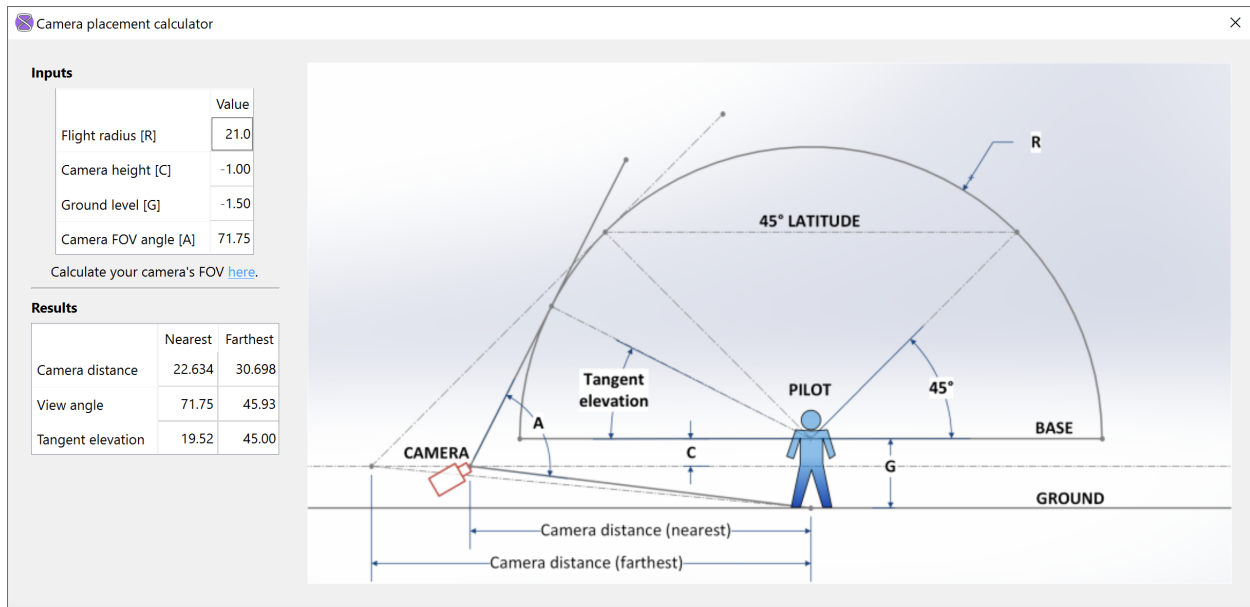


Fig. 5.2: Camera placement calculator in VideoF2B.

- Camera FOV angle A is the maximum vertical **angle of view** of your camera system as determined above.

As you adjust each input value, the values in the **Results** table will update accordingly. The values of interest are in the row labeled **Camera distance**. These numbers represent the range of recommended distances for the camera. Place the camera within this range for best results.

Danger: Please be aware that the outboard wing of the aircraft extends outside the flight hemisphere, and the pilot never stays exactly in the center of the circle during a flight. Do not place the camera too close to the flight radius even when the calculated “nearest” distance value is very close to R!

Hint: You may use any suitable distance units for values of R, C, and G, just stay consistent. The default values are in meters. All angular values are always in degrees.

Important: For safety reasons, **the calculator does not allow the camera inside the flight hemisphere**. That is, the calculated “nearest” value of “camera distance” should **never** be less than the flight radius R. If you encounter a calculation where this is not true, please submit a bug report with your input values.

With the above precautions in mind, you are ready to produce *Basic* or *Augmented-Reality* videos.

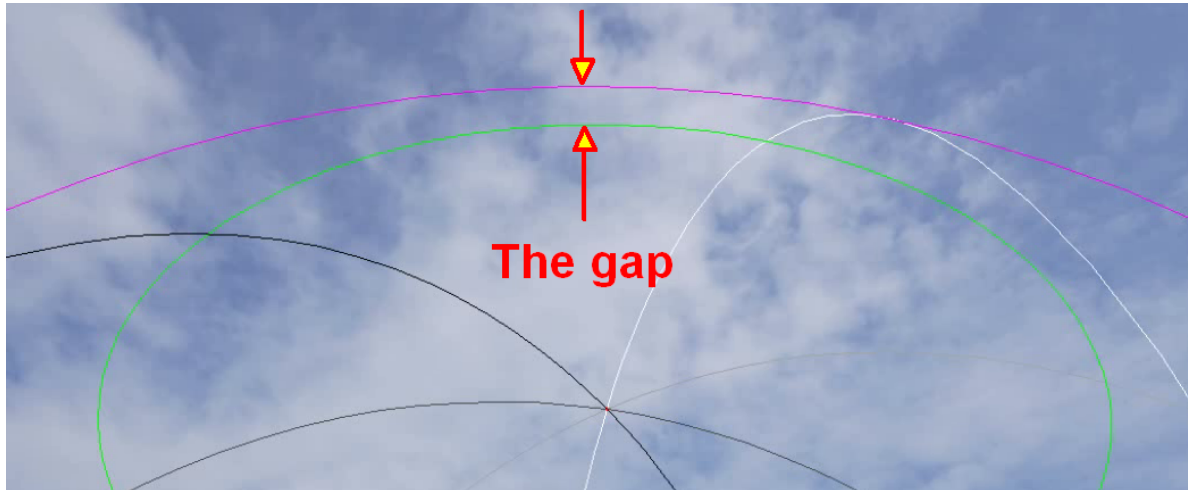
For the technically inclined...

There are two criteria for camera placement.

The first may be obvious—the center of the flight circle must be visible in the FOV so that users may select it during AR processing. This is shown in the calculator diagram by extending the bottom of the FOV angle A to the point on the ground at the pilot’s feet.

The second criterion may not be immediately obvious. It is based on two facts:

1. The “camera cone” formed by the camera’s angle of view separates the AR hemisphere into two parts: the “near” and the “far” volume. Image space is represented by integers, resulting in a “dead zone” between the two volumes where the aircraft’s location cannot be determined. Whenever the aircraft passes through this zone, the motion trace generated by VideoF2B “jumps” across the boundary without any information between the two points. Note that this information is irrelevant during AR processing, but it is vitally important during 3D tracking.
2. The **Overhead Eight** maneuver is critically close to the “dead zone”. To minimize the chances of the aircraft passing across this boundary during the overhead eight, the calculator ensures that the point labeled as “Tangent elevation” on the diagram is never above the 45° elevation of the flight hemisphere. This criterion enforces a visible gap in video between the circle of 45° elevation (drawn in bright green) and the visible edge of the flight hemisphere (drawn in magenta):

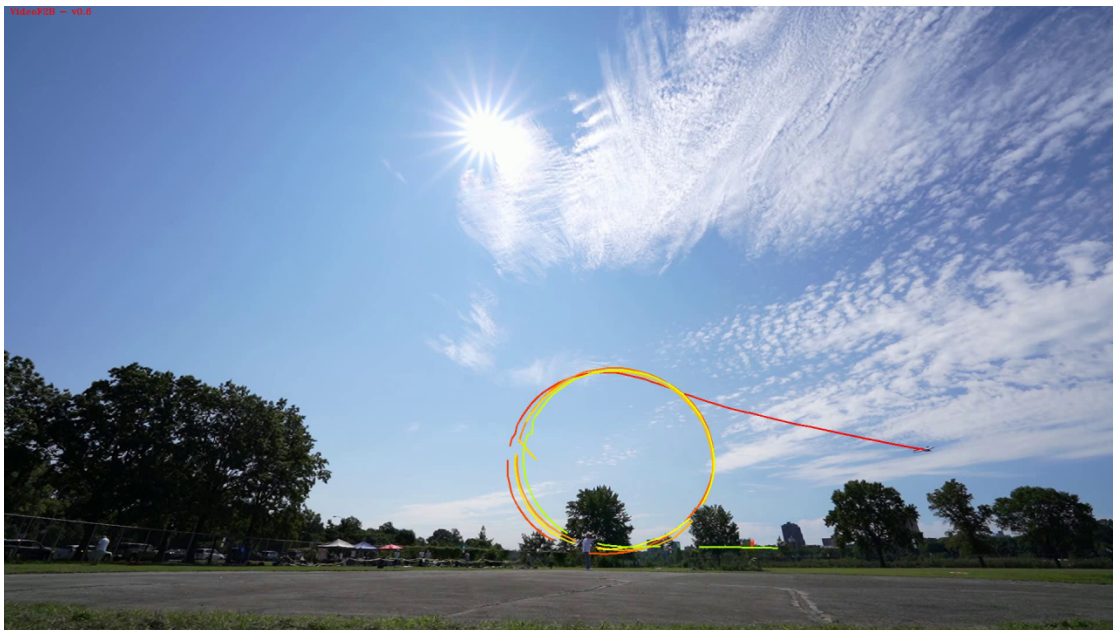


TODO: an example AR sphere due to a badly placed camera (too far from circle) that results in loss of “gap”.

PRODUCING BASIC VIDEOS

A Basic video contains a colored **trace** of the path of the aircraft. No additional geometry is drawn.

Here is an example:



To produce a Basic video, follow these steps:

1. Record a Stunt flight using a video camera. For guidelines on how to position the camera in the field, see [Placing the Camera](#). Save the video file to your computer.
2. Start the VideoF2B application. The main window looks like this when the application starts:
3. Choose *File* → *Load* in the main menu.
4. Click the *Browse for file* button of the *Video source* box:
5. Choose the desired video on your computer and click the *Open* button.
6. Click the *Load* button or just press the **Enter** key. The video will begin processing in the main window.
7. The trace behind the aircraft grows up to 15 seconds long. During processing, you can clear the trace at any time by pressing the **Space** bar.

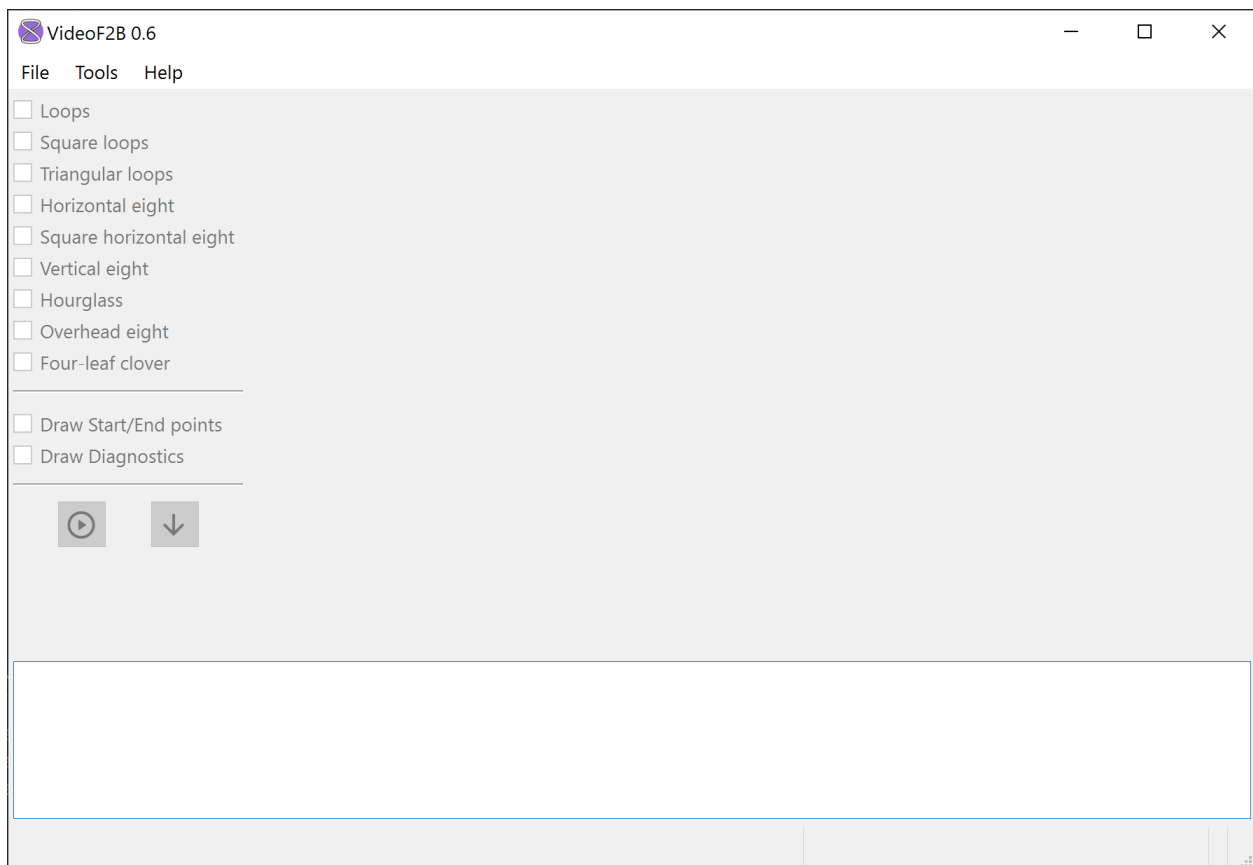


Fig. 6.1: Main window of VideoF2B.

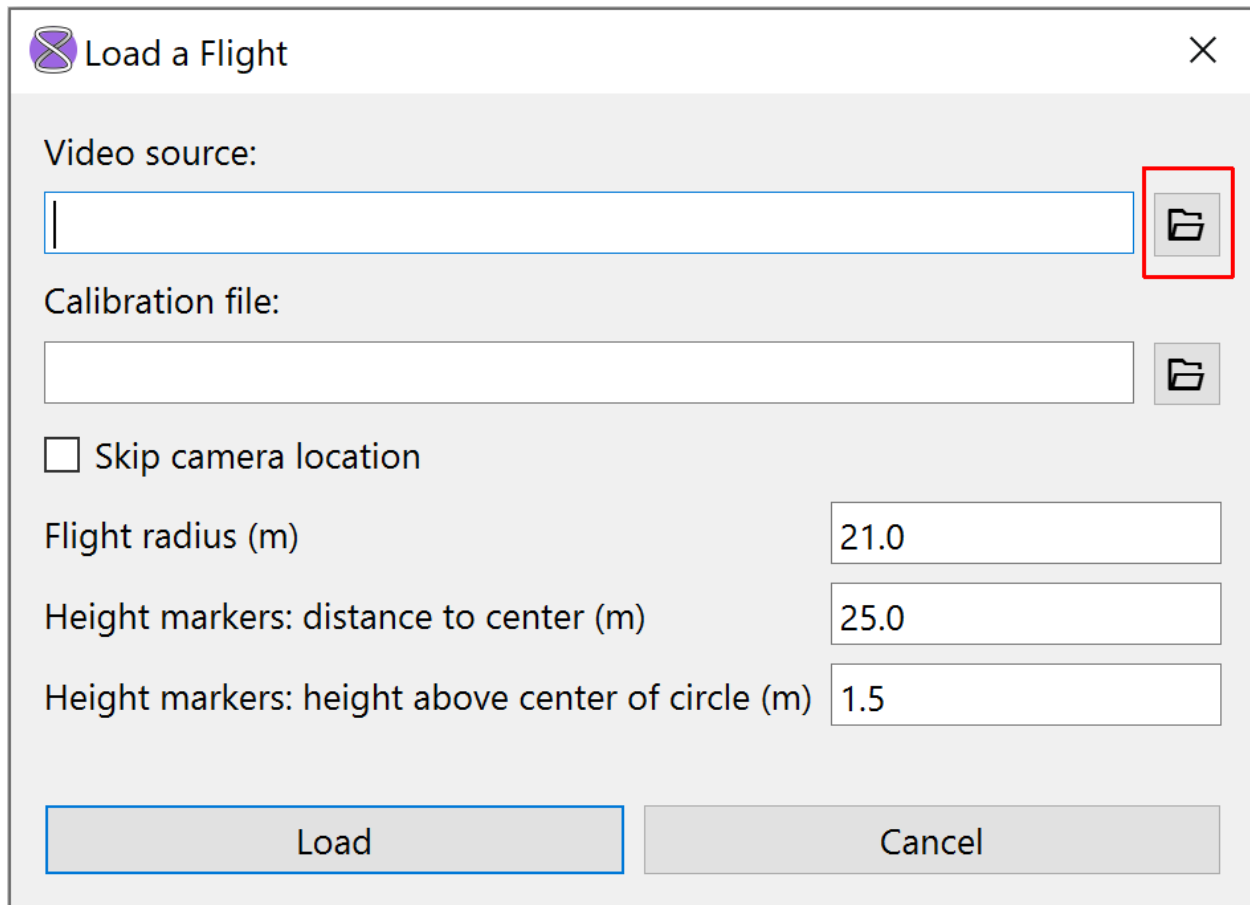


Fig. 6.2: The "Load a Flight" dialog window. Just choose your video file from here.

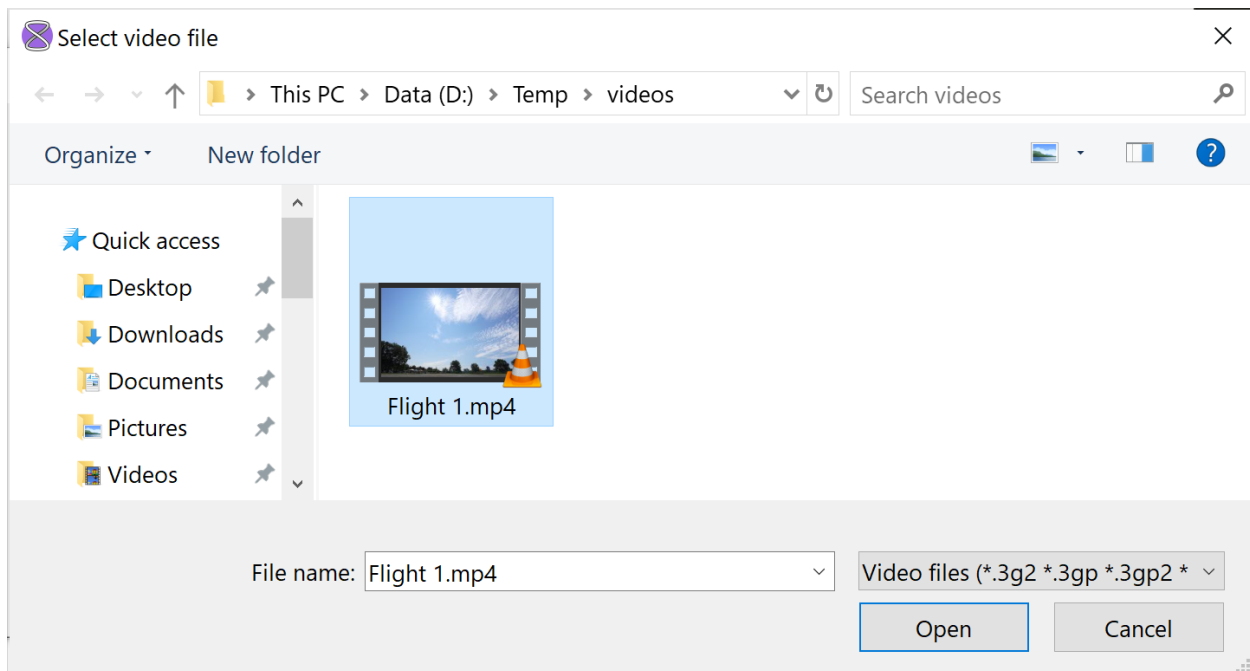


Fig. 6.3: File browsing dialog. This may look different on your computer.

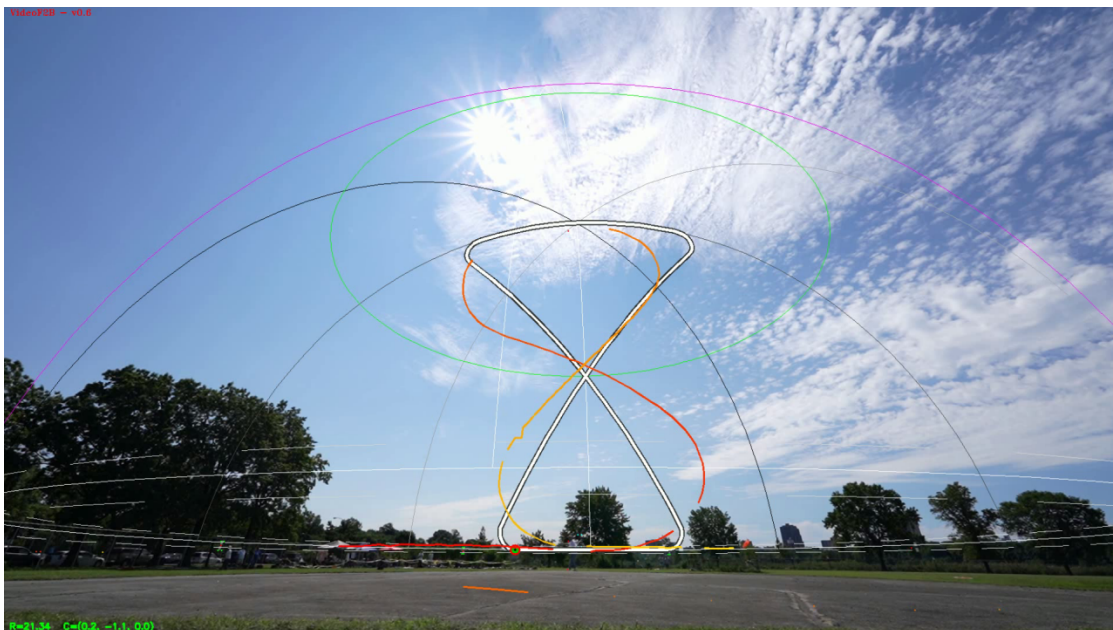
8. If you wish to stop processing the video for any reason before VideoF2B finishes tracing it, press the Esc key on the keyboard. This will stop the tracing, and the result will be a partially processed video.
9. When finished, you will find the traced video file in the same location as the original video. The traced video will have the same name as the original, but with a `_out` suffix. For example, if your original video is named `Flight 1.mp4`, the traced video will be named `Flight 1_out.mp4`.

PRODUCING AUGMENTED-REALITY VIDEOS

An *Augmented-Reality* (AR) video contains overlays of various reference graphics on top of the original video footage. In addition to the motion trace behind the model aircraft, the AR graphics may include the following:

- A wireframe representation of the *flight hemisphere*, which includes:
 - the *base*;
 - tolerance *horizontal*s 0.30 m above and 0.30 m below the base;
 - the 45° horizontal;
 - *vertical*s at every 1/8 lap, from base to *top of circle*; and
 - the visible edge of the hemisphere.
- Marker points.
- *Nominal* figure representations according to current rules.
- Start and end points of maneuvers.
- Diagnostic points in figures.

Here is an example:



To produce an AR video, follow these steps:

1. *Calibrate your camera.*
2. To enable AR graphics, see *Field setup*.
3. Record a Stunt flight using a video camera. For guidelines on how to position the camera in the field, see *Placing the Camera*. Save the video file to your computer.
4. *Load the flight* into VideoF2B.
5. *Locate the flight* in VideoF2B.
6. Process the flight video. See *User Controls* for guidance on manipulation of AR graphics.
7. When finished, you will find the processed AR video file in the same location as the original video. The AR video will have the same name as the original, but with a `_out` suffix. For example, if your original video is named `Flight 1.mp4`, the traced video will be named `Flight 1_out.mp4`.

CAMERA CALIBRATION

Before you can produce *Augmented-Reality* videos, you must calibrate your camera system. Camera calibration accomplishes two things in one step. First, it calculates distortion parameters of the camera's optical system. This allows the processor in VideoF2B to "undistort" every video frame so that straight lines in the real world remain straight in video. Undistorted frames are essential to many image processing tasks. Second, it establishes a relationship between the size of objects in video versus the size of the same objects in the real world. This is important for drawing Augmented-Reality geometry of the correct size and shape in the video.

Calibration involves the recording of a special video and consists of three easy steps. To begin, start VideoF2B and choose *Tools* → *Calibrate camera..* in the main menu. You will see the following window:

8.1 Obtain the Calibration Pattern

You have two choices for the calibration pattern: **display** it on screen or **print** it to paper. The recommended method is to print. However, if you do not have access to a printer, displaying it on screen is also acceptable.

Important: To **print** the pattern you will need a PDF reader application, such as Adobe Acrobat, Foxit, or similar.

If you decide to **print** the pattern, make sure to **mount it flat** to a suitable piece of cardboard or poster board for easy handling while maintaining accuracy.

Note: The absolute size of the pattern is not important. Whether you display it or print it, do not worry about its true size. It is only important that the **entire pattern is flat and visible**.

8.2 Record the Video

Record the video using your camera system. The video should be fairly short; about 30-50 seconds is enough. The pattern should be visible in its entirety throughout the video. Move and tilt the camera so that you record as many perspectives of the pattern as possible. To see an example video, **click the thumbnail under Step 2** in the calibration window. An alternative method is to mount the camera on a tripod, then move and tilt the printed pattern in front of the camera.

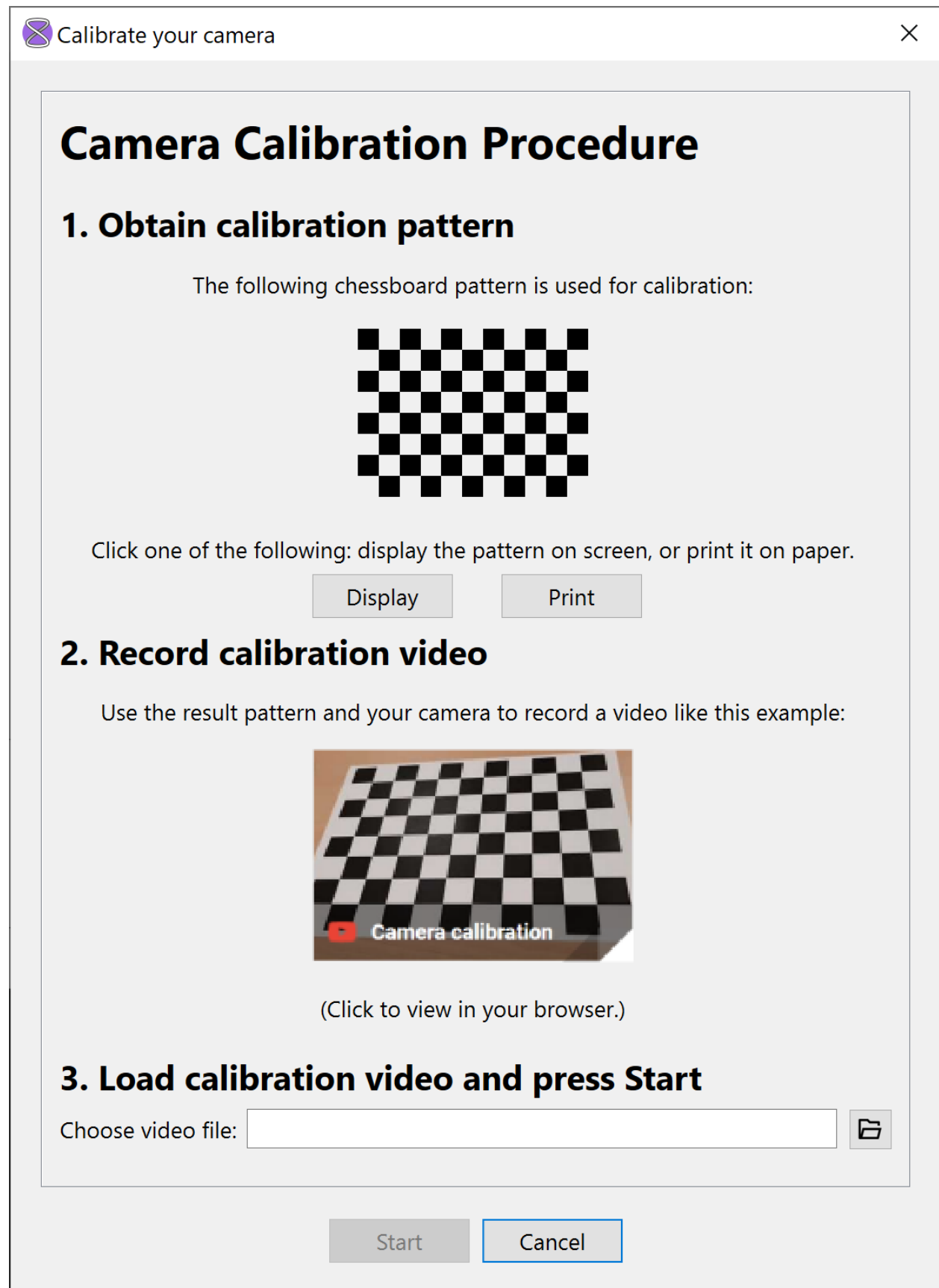


Fig. 8.1: Camera calibration dialog.

Attention: Configure your camera with the **same** video settings that you will use in the field to record the flights. This means that your choice of **lens**, its **focal length**, and **video resolution** all **must be the same** during calibration and during field recordings. If the focal length is adjustable (also known as a “zoom lens”), then you must make sure to set the focal length to the same value during field recordings as you did during calibration. When using the camera of a mobile device, always orient the device in **landscape** mode (horizontally) and make sure you always choose the same **zoom factor** and **video resolution** as you did during calibration. If you neglect to follow this rule, you will get unexpected results in your *Augmented-Reality* videos. This rule does not apply to the frame rate of the video.

If you chose the **Display** option for the pattern in **Step 1**, press the Esc key to return to the calibration window after recording the video.

8.3 Process the Video

Transfer the video file to your computer. Under **Step 3**, browse to the file. Finally, press the *Start* button at the bottom of the window. VideoF2B will begin processing the calibration video in the main window:

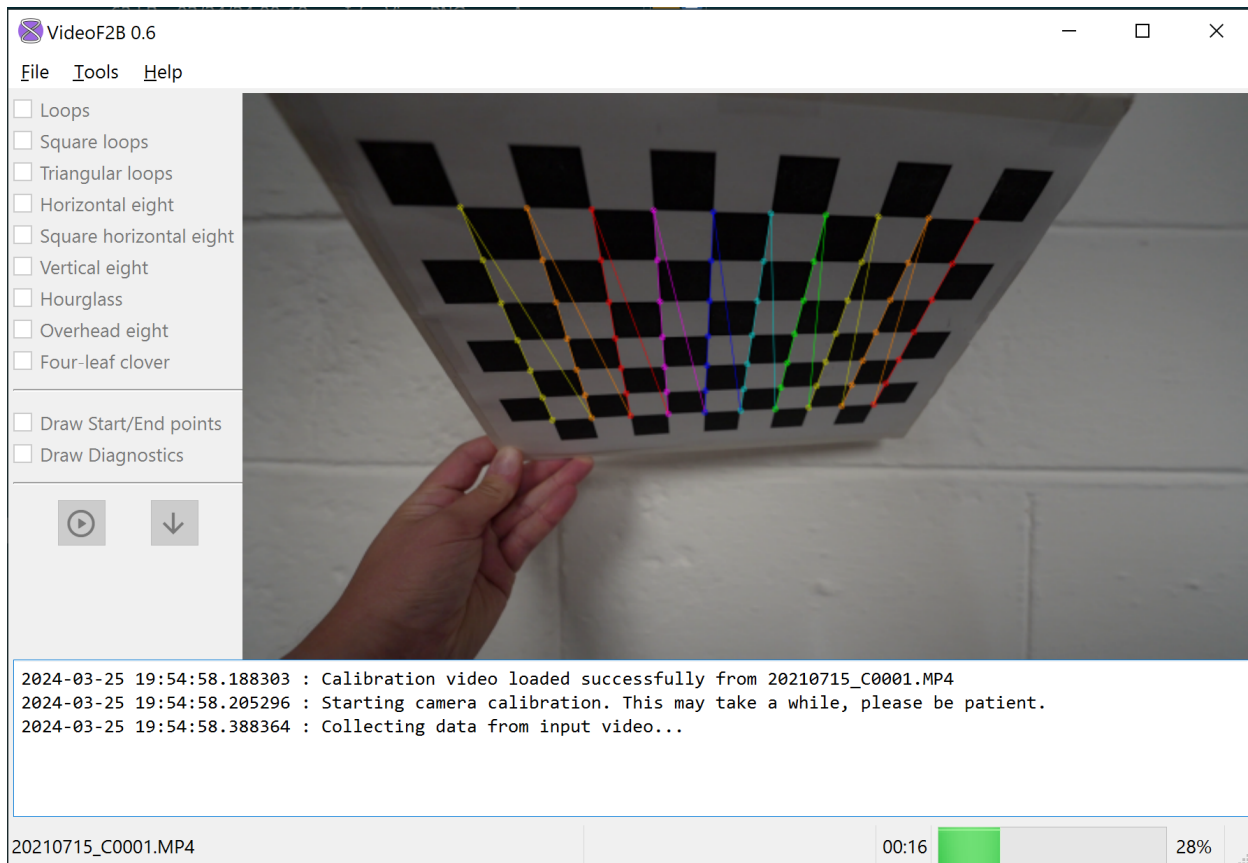


Fig. 8.2: Main window of VideoF2B showing camera calibration in progress.

As stated in the message window, the calibration process takes a while. The video playback will appear in slow motion, and it will seem to “skip” and “freeze” at times, but do not fret – all is well. The calibration process is computationally intensive. If you do want to stop the calibration at any time for any reason, just press the Esc key. Otherwise, grab a

cup of coffee, relax, and wait patiently until the progress bar reaches 100%. When finished, the video will disappear from the main window, and you will see some information about the results in the message window:

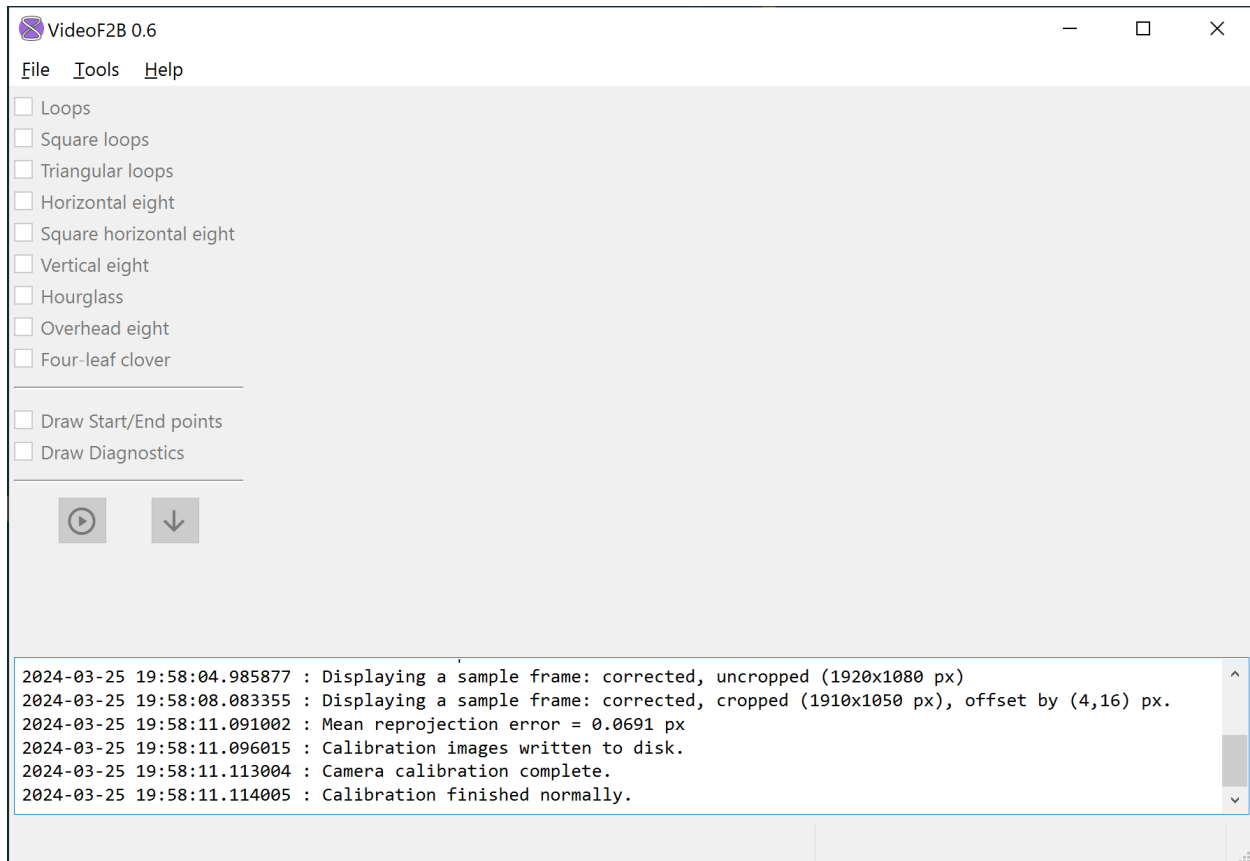


Fig. 8.3: Main window at end of camera calibration. Take note of the messages in the message window.

If the calibration fails, most likely your video is too short and/or it does not show the complete pattern from a sufficient number of points of view. In that case, record another video while paying attention to those details.

If the calibration succeeds, VideoF2B will create a file named `CamCalibration.npz` and two image files in the same folder as the calibration video. The `CamCalibration.npz` file is the calibration file for your camera system. **Do not lose it.** You will need it for producing every Augmented-Reality video of the flights you will record with your camera. You may also share it with others who have the same camera system as you.

For the technically inclined...

The two image files show a sample frame from the calibration video. The image `calibresult_nocrop.png` is a full-size frame that is “undistorted”, i.e., straight lines of the pattern should appear straight in the image. To achieve this, the calibration process transforms the original frame in such a way that empty pixels appear around the edges of the undistorted image, giving the edges a “pincushion” look:



Fig. 8.4: Uncropped calibrated frame.

The strength of the pincushion effect depends mostly on the distortion inherent to the lens, and on the focal length. Wide-angle action cameras typically show a stronger effect than longer lenses.

The other image file is `calibresult.png`. It is the same image as the “no-crop” image above, with one important difference. It is cropped to the **maximum usable area** so that the empty pixels are no longer visible:



Fig. 8.5: Cropped calibrated frame.

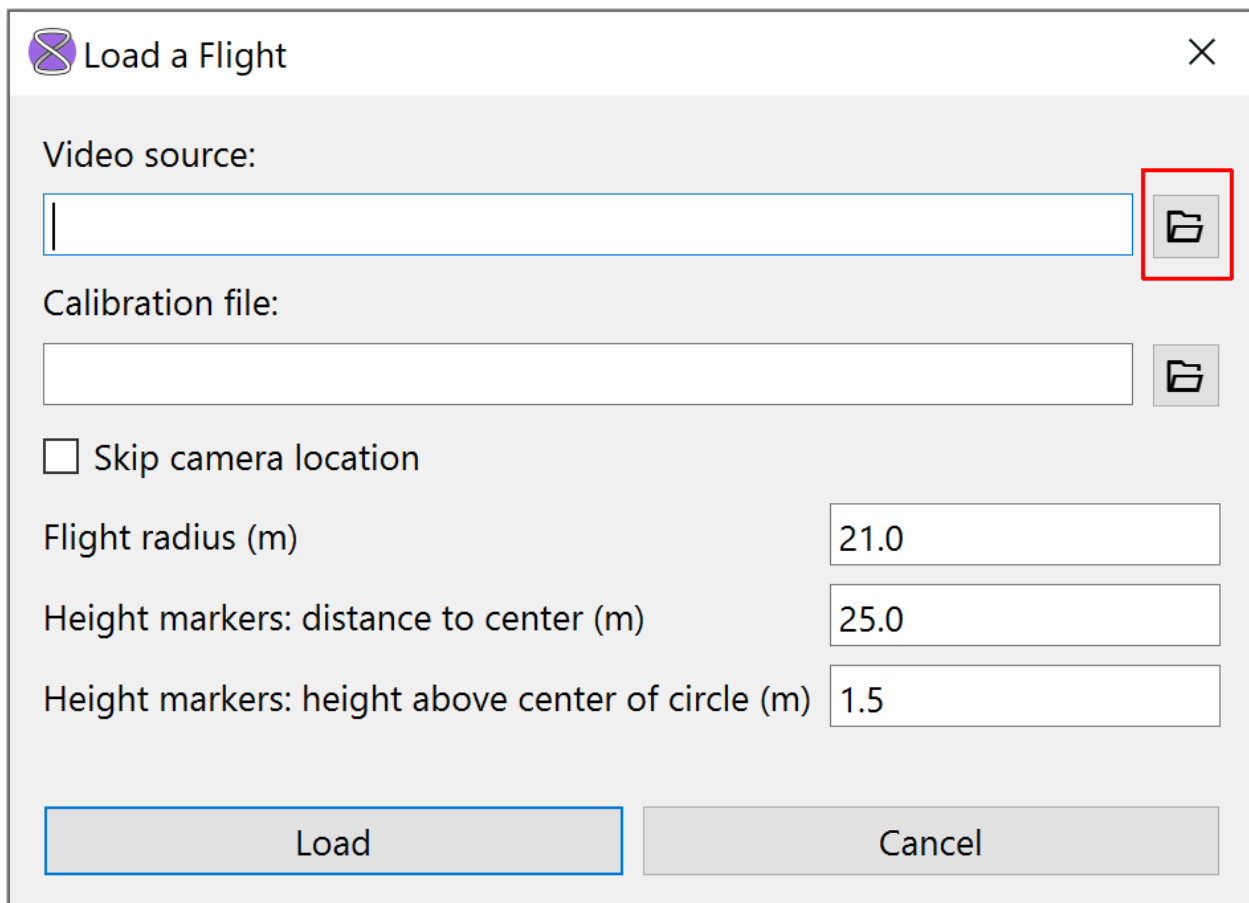
Note that this always results in a smaller image than the full-size video frame that you see in the camera. In the above examples, the “no-crop” image size is the original Full HD, or 1920x1080 pixels. The cropped image size is 1910x1050 pixels. So a total of 10 pixels were lost from the sides, and a total of 30 pixels from the top and bottom of the original frame. It is important to keep this in mind when placing the camera in the field. Give yourself some room, especially at the bottom of the frame, to account for the lost pixels. VideoF2B will “upscale” calibrated video to the size of the original input video whenever possible, but some pixels around the border of the original video will be lost due to calibration.

Congratulations, you are ready to record Control Line Stunt videos! The next step is *field setup*.


LOADING A FLIGHT


To load a flight into VideoF2B for AR processing, follow these steps:

1. Choose *File* → *Load* in the main menu.
2. Click the *Browse for file* button of the *Video source* box:



Load a Flight

Video source: 

Calibration file: 

☐ Skip camera location

Flight radius (m)

Height markers: distance to center (m)

Height markers: height above center of circle (m)

Fig. 9.1: The “Load a Flight” dialog window. Specify all flight parameters here.

3. Choose the desired video on your computer and click the *Open* button.

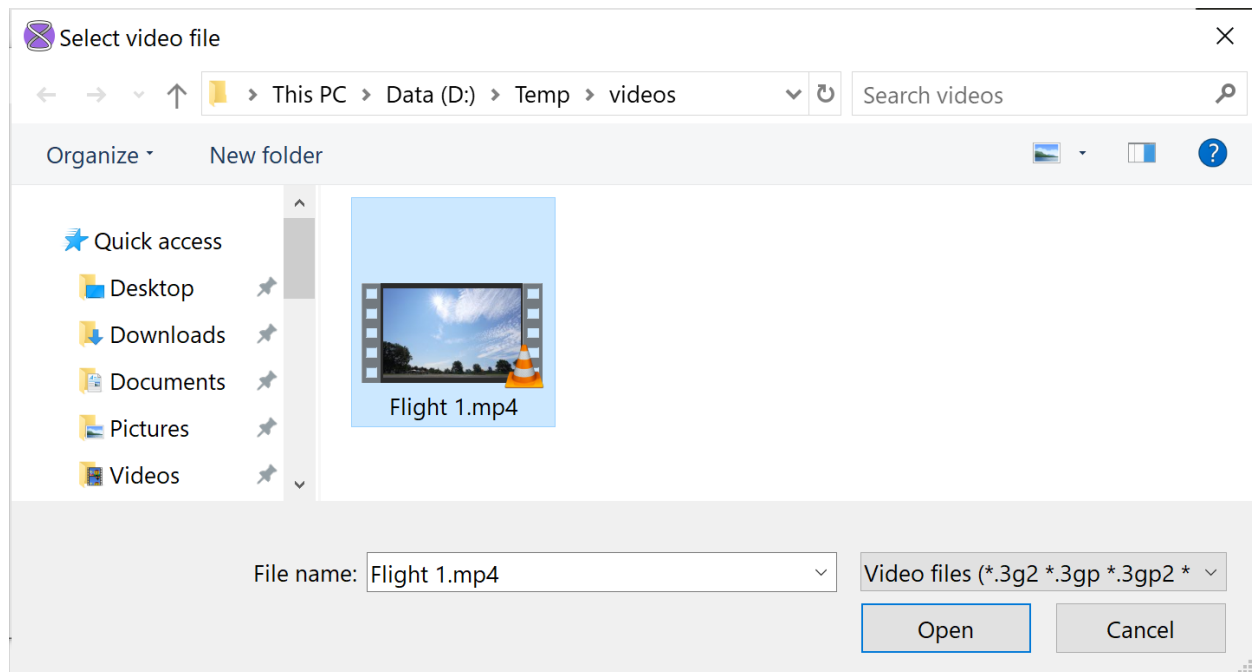


Fig. 9.2: File browsing dialog. This may look different on your computer.

4. Click the *Browse for file* button of the *Calibration file* box.
5. Choose the calibration file that you created during *camera calibration*.

Important: Be sure to choose the calibration file that corresponds to the camera and lens that were used to record the video you selected above.

Note: If F2B markers are not available, but you still want to create video that corrects for camera distortion (using your camera's calibration file), turn on the option *Skip camera location*. Note that in this case, entry of AR-related parameters is disabled and Augmented-Reality graphics *will not be drawn*.

6. Enter the following AR-related parameters:

Flight radius (m)

The *flight radius* of the recorded flight, in meters.

Height markers: distance to center (m)

The horizontal distance from the center of the flight circle to the F2B markers, in meters.

Height markers: height above center of circle (m)

The elevation of F2B markers above the pilot's feet at the center of the flight circle, in meters.

Important: Please use meters for the above three parameters.

7. Click the *Load* button or just press the **Enter** key.

LOCATING A FLIGHT

After you load a flight for AR processing, the AR graphics can only be drawn in video after the flight has been located. This is done via the **locating procedure** described below. This procedure establishes a relationship between the real 3D world and the 2D video that was used to record it. The procedure consists of picking objects in video that are positioned at known locations in the real world.

To locate a flight, follow these steps after loading it:

1. The video window will display the first frame of your video so that you can select F2B markers. This procedure locates the camera in video relative to the flight circle so that AR geometry can be displayed.



Fig. 10.1: First step of camera locating: begin selecting markers.

Follow the prompts in the middle of the status bar to select markers. Be as accurate as possible when selecting each marker.

To **select** a marker, point the mouse cursor to it and click the **left** mouse button.

To **unselect** the last selected marker, click the **right** mouse button anywhere in the video window.

You will be prompted to select the following four items:

Circle center

Select a point **on the ground in the center of the circle**. If you know that the pilot is standing exactly in the center at the start of the video, select a point at his or her feet. If the pilot is not standing in the center of the pilot circle at the start of the video, select a point on the ground where you estimate the center of the pilot circle to be. This can be done by reviewing the video separately in a video player. Fast-forwarding the video to a time when the pilot is in the middle of a maneuver is the recommended method of estimating the location of the circle center.

Front marker

Select the center of a marker on the far side of the flight circle that is nearest to the middle of the video frame. It does not matter which marker you choose to be the front, as long as markers adjacent to it are visible in the video frame.

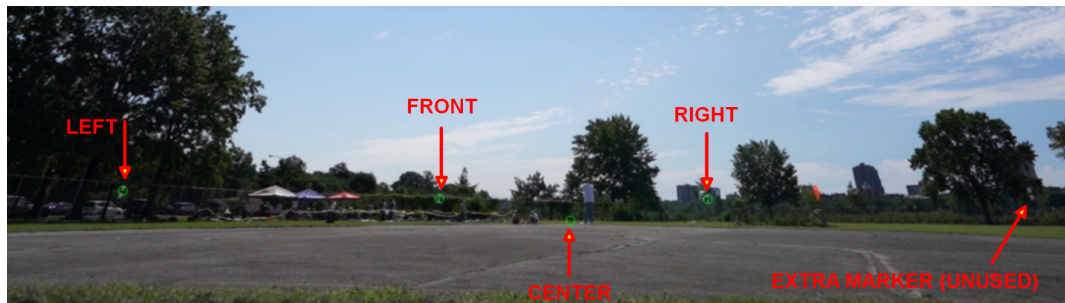
Left marker

Select the center of the nearest marker to the **left** of the front marker on the far side of the flight circle, i.e., the next marker in the counterclockwise direction.

Right marker

Select the center of the nearest marker to the **right** of the front marker on the far side of the flight circle, i.e., the next marker in the clockwise direction.

When you select a marker, VideoF2B draws a small green circle around the selected point. Here is an example of all four markers after selection:



2. When you select the final marker, you will see this prompt:

If you made incorrect selections, click *No*. The current marker selections will be cleared, and you will have a chance to select all of them again.

If you are satisfied with your selections, click *Yes*. Processing will begin.

See [User Controls](#) to learn how to control AR geometry during the processing. You may want to review information about [The World Coordinate System](#) first.



Fig. 10.2: Camera locating in progress. Center, front, and left markers have been selected in this example.

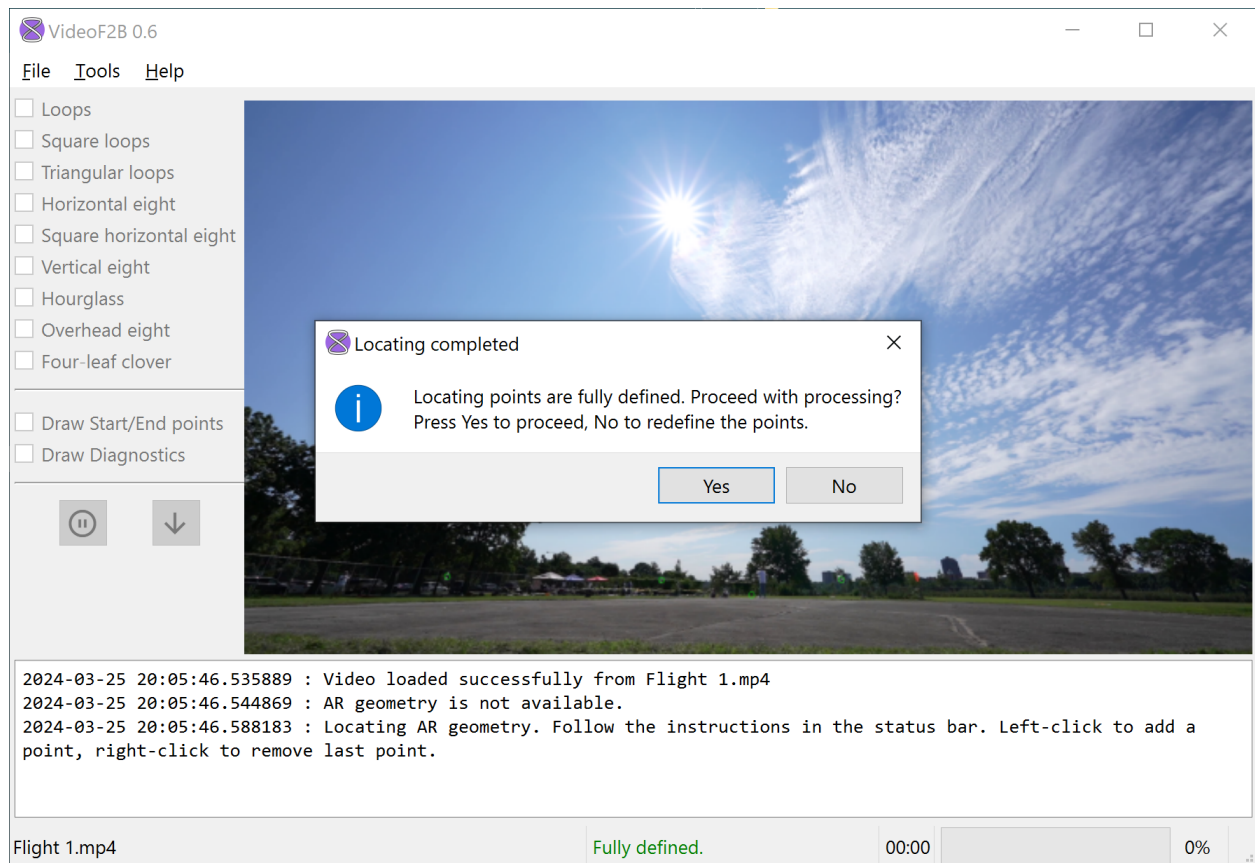


Fig. 10.3: Confirmation prompt at end of camera locating procedure.

THE WORLD COORDINATE SYSTEM

The *flight locating* procedure establishes a World Coordinate System (WCS) based on the selected markers. The WCS is a *right-handed Cartesian* coordinate system. VideoF2B uses the WCS to draw all AR graphics correctly in video. The position and orientation of the WCS is as follows:

- Origin is at the center of the *base*. Thus, its elevation is 1.5 m above the pilot circle.
- Positive Y-axis passes through the **front marker**.
- Positive Z-axis points vertically upward, and passes through *top of circle*.
- Positive X-axis is perpendicular to both Y and Z axes, and generally points to the right in video.

TODO: replace the photo below with a field photo that shows:

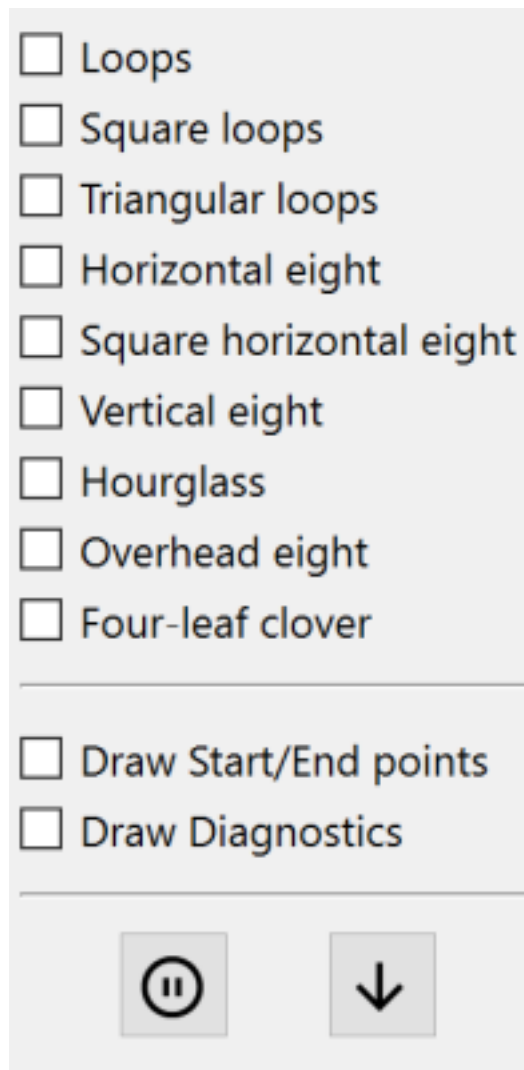
- all four markers and the camera on a tripod outside the circle.
- SVG graphics that depict the WCS.
- AR graphics that depict the flight hemisphere.



USER CONTROLS

This chapter applies to the user controls that are available during *Augmented-Reality (AR) processing*.

The following user controls on the left side of the main window are enabled during AR processing:



- ☐ Loops
- ☐ Square loops
- ☐ Triangular loops
- ☐ Horizontal eight
- ☐ Square horizontal eight
- ☐ Vertical eight
- ☐ Hourglass
- ☐ Overhead eight
- ☐ Four-leaf clover

- ☐ Draw Start/End points
- ☐ Draw Diagnostics





Fig. 12.1: User controls available during AR processing.

VideoF2B reads frames from a given video in strict sequence from beginning to end. The user interface is designed for an efficient workflow via the keyboard alone, while some of those controls are also available in the main window, as seen above.

12.1 Pausing/Resuming Processing

There is no fast-forward or rewind functionality. However, you can pause the processing at any time by pressing P on the keyboard or clicking the button. While processing is paused, you can perform various manipulations of AR geometry, taking as much time as you need. When ready to continue, press P again or click the button.

12.2 Clearing the Trace

To clear the trace behind the model aircraft, press Space. This is useful for presenting clear traces of maneuvers — clear the trace shortly before an upcoming maneuver to present the traced maneuver clearly.

12.3 Manipulating the Flight Hemisphere

To account for the pilot's movement in the pilot circle during a flight, use the WASD keys to move the flight hemisphere in the world XY plane. The keys operate as follows:

W moves the hemisphere in +Y direction (**forward**, away from the camera).

S moves the hemisphere in -Y direction (**backward**, toward the camera).

A moves the hemisphere in -X direction (to the **left**).

D moves the hemisphere in +X direction (to the **right**).

Every stroke of the above keys moves the hemisphere in the commanded direction by **0.1 m**.

Pressing X resets the hemisphere's center to the origin of the World Coordinate System.

The flight radius (R) is always displayed in the bottom left corner of AR videos. Additionally, when the AR hemisphere's center (C) is not at the origin, its XYZ offset will be displayed next to the flight radius:

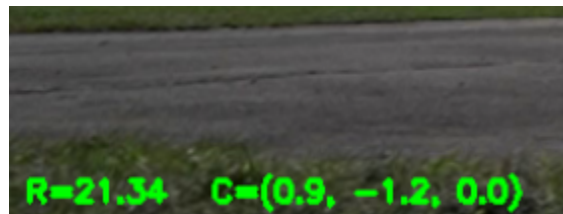


Fig. 12.2: Sphere information in AR video. All dimensions are in meters.

Tip: *Compensating for the pilot's off-center displacement*

Position of the pilot along the X-axis is easy to match accurately. Position along the Y-axis is more difficult to estimate because depth is difficult to gauge in video. Take advantage of the **Reverse Wingover** maneuver to assess the pilot's initial position. You will be able to adjust the hemisphere's position so that the aircraft's centerline crosses the visible edge of the sphere, while keeping the hemisphere's center on the pilot. As the flight proceeds, use your best judgment.

Other maneuvers whose approaches cross the visible edge of the hemisphere above the base (entry and exit of **Outside Square Loops** and entry of **Overhead Eight**) also help to correct for the pilot's position along the Y-axis throughout the flight.

To match the *nominal* figure to the maneuver flown by the pilot, use the **arrow keys** to rotate the hemisphere. The keys operate as follows:

Left Arrow rotates the AR hemisphere **counterclockwise** on its vertical axis (i.e., the nominal figure moves to the **left** as seen by the pilot).

Right Arrow rotates the AR hemisphere **clockwise** on its vertical axis (i.e., the nominal figure moves to the **right** as seen by the pilot).

Every stroke of these arrow keys rotates the hemisphere in the commanded direction by **0.5°**.

12.4 Displaying Nominal Figures

To toggle the display of any nominal figure, click its corresponding checkbox in the user controls. You can also use the **Down Arrow** key or the **button** to advance to the next figure in the Stunt Pattern sequence. If no figures are selected in the controls, the advancing function will select loops. If one figure is selected, the advance function will unselect the current figure and select the next figure in the sequence. If the current figure is the four-leaf clover, the figure selection will remain and the advancing function will not have any effect. If more than one figure is selected, the advancing function will likewise have no effect.

Note: Any combination of nominal figures can be displayed, even if only for training and/or demonstration purposes.

12.5 Displaying Start/End Points

Every maneuver has a start and an end point for judging purposes, as defined by the FAI F2B Rules. To toggle the display of start and end points on the displayed nominal figure(s), click the **Draw Start/End points** checkbox at any time during AR processing:



Fig. 12.3: This controls the display of start/end points in displayed figure(s).

The start point is displayed in green



, and the end point is displayed in red



12.6 Displaying Diagnostic Points

VideoF2B can optionally display diagnostic points. These are just visual aids for presentation. They are defined as endpoints of the arcs that make up a figure. In simple loops, they're at the bottom of the loop. In more complex figures, diagnostic points help to visualize where the connections between the “straight” segments and the corners or loops of the figure are located.

To toggle the display of diagnostic points on the displayed nominal figure(s), click the **Draw Diagnostics** checkbox at any time during AR processing:

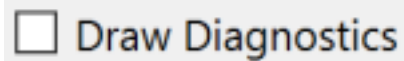


Fig. 12.4: This controls the display of diagnostic points in displayed figure(s).

Diagnostic points are displayed in alternating green and red colors per figure. For example, this is the square horizontal eight with diagnostics displayed:

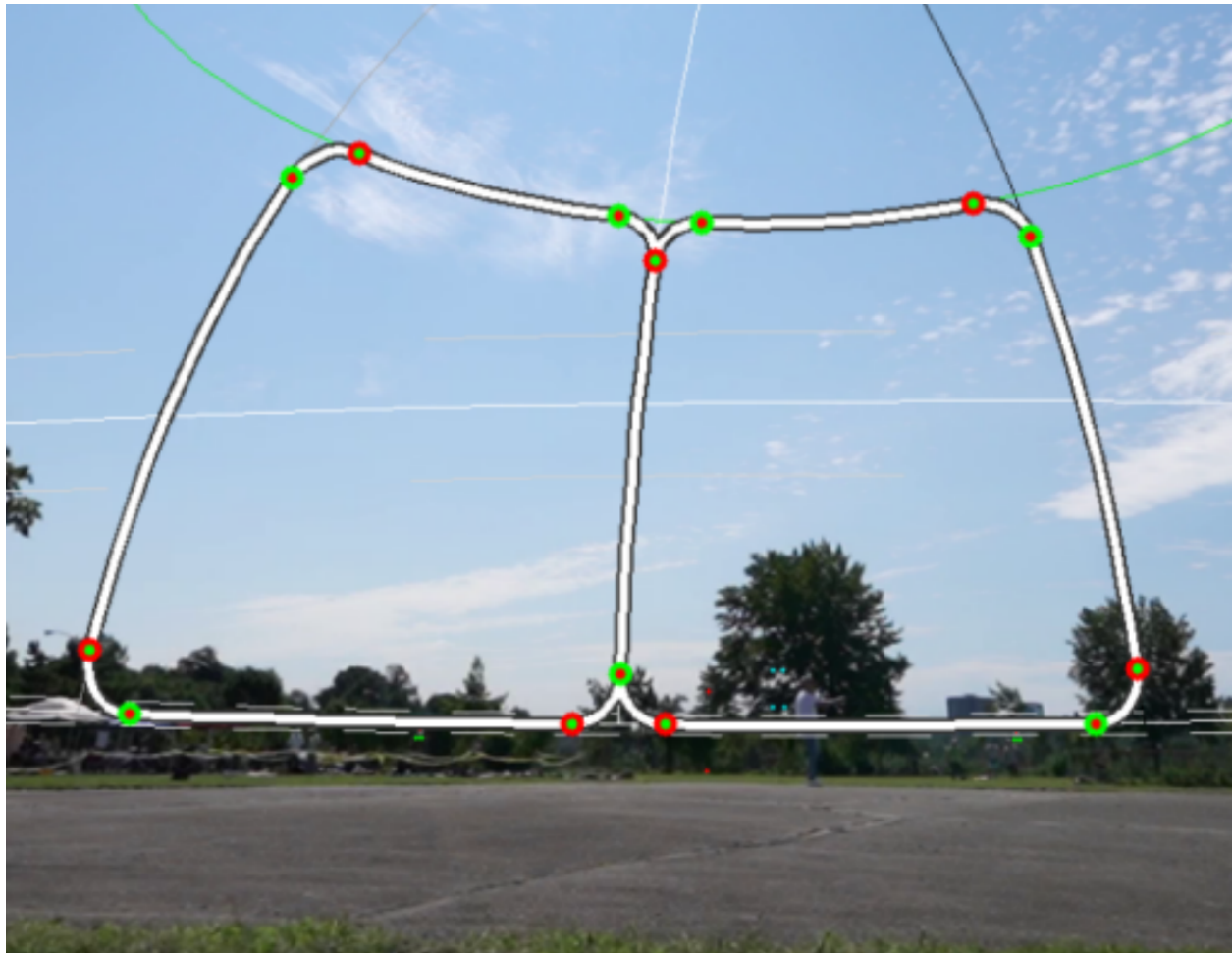


Fig. 12.5: Example: figure with diagnostics enabled.

The start/end points are drawn on top of the diagnostic points in case both options are enabled for display.

CHEATSHEET OF USER CONTROLS

| Key(s) | Operation |
|-------------|---|
| Ctrl + O | Open (load) a video for processing |
| Ctrl + Q | Quit the application |
| Esc | Stop processing a video or a calibration |
| P | Pause/Resume video processing |
| Space | Clear current trace |
| Down Arrow | Next figure |
| Left Arrow | Rotate AR sphere counterclockwise (stunt figure rotates left wrt pilot) |
| Right Arrow | Rotate AR sphere clockwise (stunt figure rotates right wrt pilot) |
| W | Move AR sphere forward (away from camera) |
| A | Move AR sphere left |
| S | Move AR sphere backward (toward camera) |
| D | Move AR sphere right |
| X | Reset AR sphere center to origin |
| C | Relocate Camera |

GETTING HELP

When you encounter issues while using VideoF2B, you are encouraged to take advantage of the following help channels:

- [GitHub](#): if you have an account, submit issues there directly.
- [Email](#) the developers.
- Post a topic on the [StuntHangar](#) forum in the **Open Forum** section.

For a list of commonly encountered issues and questions, see the [FAQ page](#).

FAQ

Q: Can I record handheld videos?

A: Absolutely not! Mount your recording device to a sturdy tripod or similar. Do not move it or adjust it while recording a flight. See *Field setup* for details.

Q: Why do I need markers?

A: F2B markers provide a base reference in the field for the pilot and for the judges. For VideoF2B, they also relate the real world to camera images, so that augmented reality geometry can be drawn accurately in video. Without the markers, augmented-reality geometry is not possible.

Q: My flying site does not have F2B markers, and installing them is not practical. Is there an alternative method for creating AR videos that does not require markers?

A: This capability is a research & development project that is currently in progress.

Q: Can I move the camera during a flight once recording has started?

A: No. Doing so will require you to select the markers in video in the new camera location.

Q: Does the trace drawn in video follow the CG of the model aircraft?

A: Not necessarily, but it tends to be fairly close to it. The motion detector follows the *centroid* (geometric center) of the silhouette of the largest moving object in video.

Q: Why are background objects tracked instead of the model aircraft?

A: VideoF2B currently does not distinguish between objects and just follows the largest moving object. If possible, avoid having moving objects as background (e.g. a road).

Q: How accurate are the Augmented Reality graphics?

A: Field tests have proven that the augmented-reality geometry drawn in video is accurate within *10 cm* throughout the entire flight envelope.

Q: Can this system be used for computerized scoring of Stunt flights?

A: The concept of tracking the flights in three dimensions using recorded video for the purpose of automated scoring is definitely under consideration. Some maneuvers are problematic to track accurately (takeoff, level/inverted flight, landing), but the majority of the flight maneuvers are potential candidates.

GLOSSARY

Augmented Reality

AR

An enhanced version of reality created by the use of technology to overlay digital information on an image of something being viewed through a device. See [Augmented Reality](#).

Base

The equator of the *flight hemisphere*. This lies at a height of 1.5 m above the center of the *flight circle*.

Field of view

FOV

The area visible through the lens of an optical instrument.

Figure

A shape, which makes up a separately recognizable complete part of a whole *maneuver*. For example, the first loop of the three consecutive inside loops maneuver is referred to as a figure; but the first loop which makes the first half of the first complete figure eight in the two consecutive overhead eight maneuver is not referred to as a figure.

Flight circle

A horizontal circle whose radius is equal to the *flight radius*.

Flight hemisphere

A half-globe shape whose base is level above the ground.

Flight radius

The distance from the pilot's chest to the centerline of the model aircraft.

Great circle

The circular intersection of a *sphere* and a plane passing through the sphere's center point. See [Great circle](#).

Horizontal

Flight along or parallel to the *base*.

Inverted

Condition when the model aircraft is flying in an attitude which is the reverse of *upright* flight (colloquially, the model aircraft is "flying on its back", is "flying upside-down", or is "flying inverted").

Lateral reference

An imaginary line drawn at right angles (90 degrees) to the *horizontal* and is used as a reference line when flying and scoring the size, positioning, symmetry and the superimposition of various *figures* and *maneuvers*.

Level

At right angles to the direction aligned with the direction of the force of gravity, as materialized with a plumb line.

Manoeuvre**Maneuver**

The full total of *figures* and *segments* necessary to complete the maneuver marked under a separate numbered heading with bold type. For example, the take-off maneuver, the three consecutive inside loops maneuver, and the single four-leaf clover maneuver, are all referred to as a single whole maneuver throughout F2B Rules.

Momentary**Momentarily**

Used throughout F2B Rules in their original dictionary definition sense (that is: something that lasts only for a very brief period of time). So, for example, the very short period during which the model aircraft is required to be in a vertically-banked “knife-edge” attitude above the competitor’s head during the two consecutive overhead eights maneuver is described in F2B Rules as “momentarily”.

Nominal

A dimension, shape, or size that represents the true profile of a *figure* or *maneuver*, and is used as the template against which actual shapes and sizes of figures or maneuvers are compared.

Parallel

An imaginary line on the surface of the *flight hemisphere* equidistant to the equator of the flight hemisphere and marking the latitude.

Segment

A specifically defined part of a *figure* (or of a whole *maneuver*) in which certain particular points are detailed. For example, the first loop which makes the first half of the first complete figure eight in the two consecutive overhead eight maneuver is referred to as a segment.

Small circle

The circular intersection of a *sphere* and a plane that does not pass through the sphere’s center point. See *Small circle*.

Sphere

A three-dimensional surface, all points of which are equidistant from a fixed point.

Straight line

A *great circle* path or part thereof.

Top of circle

A point at the top of the *flight hemisphere*, vertically above the center of the hemisphere.

Upright

Condition when the model aircraft is flying in its “normal” upright attitude (that is: with its landing gear nearest to the ground).

Vertical

Flight at right angles to the *base*, along an imaginary circle on the surface of the *flight hemisphere* marking the longitude.

Wingover path

The *vertical* climbing and diving flight path defined as a *segment* of the single reverse wingover *maneuver*.

17.1 videof2b package

17.1.1 Subpackages

`videof2b.core` package

Subpackages

`videof2b.core.common` package

Submodules

`videof2b.core.common.path` module

Path-related utilities.

`videof2b.core.common.path.files_to_paths(file_names)`

Convert a list of file names to a list of file Path objects.

Parameters

file_names (*list[str]*) – The list of file names to convert.

Returns

The list converted to file paths

Return type

`list[Path]`

`videof2b.core.common.path.path_to_str(path=None)`

Convert a Path object or NoneType to a string equivalent.

Parameters

path (*Path | None*) – The value to convert to a string

Returns

An empty string if *path* is None, else a string representation of the *path*

Return type

`str`

`videof2b.core.common.path.replace_params(args, kwargs, params)`

Transform the specified args or kwargs

Parameters

- **args** (*tuple*) – Positional arguments
- **kwargs** (*dict*) – Keyword arguments
- **params** – A tuple of tuples with the position and the keyword to replace

Returns

The modified positional and keyword arguments

Return type

tuple[tuple, dict]

Usage: Given a method with the following signature, assume we want to apply the *str* function to *arg2*: *def method(arg1=None, arg2=None, arg3=None)*

Since *arg2* can be specified positionally as the second argument (1 with a zero index) or as a keyword, we would call this function as follows: `replace_params(args, kwargs, ((1, 'arg2', str),))`

`videof2b.core.common.path.str_to_path(string)`

Convert a str object to a Path or NoneType.

This is especially useful because constructing a Path object with an empty string causes the Path object to point to the current working directory, which is not desirable.

Parameters

string (*str*) – The string to convert

Returns

None if *string* is empty, or a Path object representation of *string*

Return type

Path | None

videof2b.core.common.settings module

Module for handling persistent settings.

`class videof2b.core.common.settings.Settings(*args, **kwargs)`

Bases: QSettings

Simple wrapper around QSettings. Contains core definitions of all known keys and their default values. Does not contain a strategy for versioning of settings. Handles lookup of default values in the most basic manner.

`staticMetaObject = PySide6.QtCore.QMetaObject("Settings" inherits "QSettings":)`

`value(key)`

Return the value for the given key. The key must exist in *Settings.__defaults__*. If value not found, return the value from *Settings.__defaults__*

videof2b.core.common.singleton module

Contains the Singleton class definition.

class videof2b.core.common.singleton.Singleton

Bases: type

Implementation of a Singleton metaclass.

videof2b.core.common.store module

This module contains objects that enable persistence of custom data.

class videof2b.core.common.store.Store(*args, **kwargs)

Bases: object

An object store. This is a singleton that provides access to object references that are shared within a process.

add(key, item)

Add an item to the store.

classmethod create()

The constructor for the Store.

get(key)

Get the specified object from the store.

remove(key)

Remove an item from the store.

class videof2b.core.common.store.StoreProperties

Bases: object

Adds shared components to classes for use at run time.

property application

Dynamically-added application object.

property settings

Dynamically-added settings object.

Module contents

Common definitions and constants for VideoF2B.

class videof2b.core.common.FigureTypes(value)

Bases: Enum

All F2B figures in sequence.

FOUR_LEAF_CLOVER = 14

HORIZONTAL_EIGHTS = 9

HORIZONTAL_SQUARE_EIGHTS = 10

HOURLASS = 12

```
INSIDE_LOOPS = 3
INSIDE_SQUARE_LOOPS = 6
INSIDE_TRIANGULAR_LOOPS = 8
INVERTED_FLIGHT = 4
LANDING = 15
OUTSIDE_LOOPS = 5
OUTSIDE_SQUARE_LOOPS = 7
OVERHEAD_EIGHTS = 13
REVERSE_WINGOVER = 2
TAKEOFF = 1
VERTICAL_EIGHTS = 11
```

```
class videof2b.core.common.SphereManipulations(value)
```

Bases: Enum

Possible manipulations of the AR sphere during processing.

```
MOVE_EAST = 4
MOVE_NORTH = 5
MOVE_SOUTH = 6
MOVE_WEST = 3
RESET_CENTER = 0
ROTATE_CCW = 1
ROTATE_CW = 2
```

```
videof2b.core.common.get_app_metadata()
```

Get basic app information. Returns (name, version) as a tuple of strings.

Return type

Tuple

```
videof2b.core.common.get_bundle_dir()
```

Return the path of the bundle directory. When frozen as a one-file app, this is the `_MEI###` dir in temp. When frozen as a one-dir app, this is that dir. When running as a script, this is the project's root dir.

```
videof2b.core.common.get_frozen_path(path_when_frozen, path_when_non_frozen)
```

Return one of the given paths based on status of `sys.frozen`.

```
videof2b.core.common.get_lib_versions()
```

Get User-friendly names and versions of libraries that we care about for bug reports. This is just a sub-list of `install_requires` items in our `setup.cfg`.

`videof2b.core.common.is_linux()`

Returns True if running on a Linux OS.

Return type

bool

Returns

True if running on a Linux OS.

`videof2b.core.common.is_win()`

Returns True if running on a Windows OS.

Return type

bool

Returns

True if running on a Windows OS.

`videof2b.core.common.launch_document(path)`

Open the specified document using the default application.

Return type

None

Submodules

`videof2b.core.calibration module`

Module for calibrating cameras.

class `videof2b.core.calibration.CalibratorReturnCodes(value)`

Bases: `IntEnum`

Definition of the return codes from CameraCalibrator's processing loop.

EXCEPTION_OCCURRED = -2

INSUFFICIENT_VALID_FRAMES = 3

NORMAL = 0

NO_VALID_FRAMES = 2

UNDEFINED = -1

USER_CANCELED = 1

class `videof2b.core.calibration.CameraCalibrator(path, is_fisheye=False)`

Bases: `QObject`

Calibrates a camera.

error_occurred

finished

new_frame_available

progress_updated

run()

Calibrate the camera using the chessboard pattern.

```
staticMetaObject = PySide6.QtCore.QMetaObject("CameraCalibrator" inherits "QObject":
Methods: #5 type=Signal, signature=new_frame_available(QImage), parameters=QImage
#6 type=Signal, signature=progress_updated(PyObject), parameters=PyObject #7
type=Signal, signature=finished(int), parameters=int #8 type=Signal,
signature=error_occurred(QString,QString), parameters=QString, QString )
```

stop()

Cancel the calibration procedure.

videof2b.core.camera module

This module contains representations of cameras.

```
class videof2b.core.camera.CalCamera(frame_size, flight)
```

Bases: QObject

Represents a real-world camera whose intrinsic and extrinsic optical properties are known.

load_calibration(*path*)

Load a camera calibration from the specified path.

locate(*flight*)

Locate a given Flight instance using this camera.

Return type

bool

```
staticMetaObject = PySide6.QtCore.QMetaObject("CalCamera" inherits "QObject": )
```

undistort(*img*)

Undistort a given image according to the camera's calibration.

videof2b.core.camera_director module

Calculator for placing a camera in the field.

```
class videof2b.core.camera_director.CamDirector
```

Bases: object

The core calculator of cam placement geometry.

property A: float

Maximum vertical viewing angle of the camera, in degrees.

property C: float

Signed camera height relative to flight equator. Up is positive.

```
DEFAULT_A = 71.75
```

```
DEFAULT_C = -1.0
```

```
DEFAULT_G = -1.5
```

DEFAULT_R = 21.0

property G: float

Signed ground level relative to flight equator. Up is positive.

property R: float

Sphere radius. Must be positive and nonzero.

property cam_distance_limits: Iterable

The limits (min, max) of camera distance from flight center.

property cam_tangent_elev_limits: Iterable

The limits (min, max) of the elevation of the tangent point from camera to sphere at the respective *cam_distance_limits*. The point's elevation is measured from flight center in degrees.

property cam_view_limits: Iterable

The limits (min, max) of camera vertical viewing angle, in degrees, at the respective *cam_distance_limits*.

solve()

For the current state, solve for the following:

d_min and *d_max* such that $\alpha \leq \max_alpha$

Subject to the constraint:

- maximum vertical view angle must include the 45-degree latitude and the center field marker on the ground.

videof2b.core.detection module

This module performs motion detection in video.

class videof2b.core.detection.Detector(maxlen, scale)

Bases: object

The primary motion detector in VideoF2B.

clear()

Clear the currently detected track.

process(img)

Detect a moving object in a given image frame.

videof2b.core.drawing module

Module for drawing flight track and Augmented Reality in video.

class videof2b.core.drawing.Colors

Bases: object

Shortcuts for OpenCV-compatible colors.

BLACK = (0, 0, 0)

BLUE = (255, 0, 0)

CYAN = (255, 255, 0)

DarkGreen = (0, 204, 0)

GRAY20 = (50, 50, 50)

GREEN = (0, 255, 0)

MAGENTA = (255, 0, 255)

RED = (0, 0, 255)

WHITE = (255, 255, 255)

YELLOW = (0, 255, 255)

class videof2b.core.drawing.DashedPolyline(*obj_pts*, ***kwargs*)

Bases: *Plot*

Defines a polyline that is drawn dashed.

draw(*img*, *key*, ***kwargs*)

Draw this polyline using its attributes.

class videof2b.core.drawing.Drawing(*detector*, ***kwargs*)

Bases: object

Container that performs all the drawing of AR sphere, track, figures, etc., in any given image frame.

DEFAULT_N = 100

draw(*img*)

Draw all relevant geometry in the given image frame.

property draw_diags

Controls the drawing of diagnostics.

property draw_endpts

Controls the drawing of maneuver endpoints.

locate(*cam*, *flight=None*, ***kwargs*)

Locate a new Flight or relocate an existing one using the given camera.

Return type

None

move_center_x(*delta*)

Move sphere center by *delta* along world X direction, in meters.

move_center_y(*delta*)

Move sphere center by *delta* along world Y direction, in meters.

reset_center()

Reset sphere center to default.

set_azimuth(*azimuth*)

Set the azimuth of the AR sphere, in degrees.

class videof2b.core.drawing.DummyScene

Bases: object

Placeholder object for an empty scene.

draw(*args, **kwargs)

No-op method.

class videof2b.core.drawing.**EdgePolyline**(*R*, *cam_pos*, **kwargs)

Bases: [Polyline](#)

Defines a special polyline that represents the visible edge of the sphere. This polyline is aware of the camera's position.

class videof2b.core.drawing.**ManeuverPoint**(*point*, *size*, *color*)

Bases: object

Represents either endpoint of a maneuver (start or end).

class videof2b.core.drawing.**Plot**(*obj_pts*, **kwargs)

Bases: object

Base class for plotting primitives. * Call *draw()* to draw the Plot instance in your image. kwargs:

size: the line thickness or point radius.

color: the color of the primitives in this Plot.

is_fixed: bool indicating whether this Plot is fixed in object space or not.

If True, world transforms do not affect the object coordinates. If False (default), then world transforms will rotate, scale, and translate the object coordinates according to the rules in the *_calculate()* method.

draw(*key*, **kwargs)

Call this method from derived classes before drawing the image points.

class videof2b.core.drawing.**Polyline**(*obj_pts*, **kwargs)

Bases: [Plot](#)

Defines a polyline.

draw(*img*, *key*, **kwargs)

Draw this polyline using its attributes.

class videof2b.core.drawing.**Scatter**(*obj_pts*, **kwargs)

Bases: [Plot](#)

Defines a collection of scattered points.

draw(*img*, *key*, **kwargs)

Draw scatter points as solid-filled circles using their attributes.

class videof2b.core.drawing.**Scene**(*items)

Bases: object

A scene consists of a collection of Plot-like objects.

add(*item*)

Add an item to this scene.

add_diag(*item*)

Add a diagnostic item to this scene.

add_endpt(*item*)

Add an endpoint item to this scene.

property diags_on

Boolean flag that controls drawing of diagnostics.

draw(*img, key, **kwargs*)

Draw this scene in the given image.

property endpts_on

Boolean flag that controls drawing of maneuver endpoints.

videof2b.core.figure_tracker module

F2B Figure tracker.

```
class videof2b.core.figure_tracker.FigureTracker(callback=<function FigureTracker.<lambda>>, **kwargs)
```

Bases: object

Container that tracks F2B figures. May be used for fitting the actual flight path to the nominal figure to determine a score.

All FAI figures, per “F2, Annex 4J – F2B Manoeuvre Diagrams” for reference.

Not all may be easy to track:

- 4.J.1 Take-off (Rule 4.2.15.3)
- 4.J.2 Reverse wingover (Rule 4.2.15.4)
- 4.J.3 Three consecutive inside loops (Rule 4.2.15.5)
- 4.J.4 Two consecutive laps of inverted level flight (Rule 4.2.15.6)
- 4.J.5 Three consecutive outside loops (Rule 4.2.15.7)
- 4.J.6 Two consecutive inside square loops (Rule 4.2.15.8)
- 4.J.7 Two consecutive outside square loops (Rule 4.2.15.9)
- 4.J.8 Two consecutive inside triangular loops (Rule 4.2.15.10)
- 4.J.9 Two consecutive horizontal eight (Rule 4.2.15.11)
- 4.J.10 Two consecutive horizontal square eight (Rule 4.2.15.12)
- 4.J.11 Two consecutive vertical eight (Rule 4.2.15.13)
- 4.J.12 Hourglass (Rule 4.2.15.14)
- 4.J.13 Two consecutive overhead eight (Rule 4.2.15.15)
- 4.J.14 Four-leaf clover manoeuvre (Rule 4.2.15.16)
- 4.J.15 Landing manoeuvre (Rule 4.2.15.17)

```
FIGURE_MAP = {0: FigureTypes.INSIDE_LOOPS}
```

add_actual_point(*idx, point*)

Add a measured (actual) point to the currently tracked figure at a given index. If no figure is currently being tracked, this call has no effect.

export(*path*)

Export all tracked figures as numpy arrays to the specified file. Arrays are labeled “fig0”, “fig1”, etc.

finish_all()

Clean-up method. Finish current figure, if any figure is in progress.

finish_figure()

Finish `trfig_type_constructor`he currently tracked figure.

start_figure()

Start tracking a new figure.

exception `videof2b.core.figure_tracker.UserError`

Bases: `Exception`

Class for exception that occur during Figure tracking due to user errors.

videof2b.core.figures module

Geometric definitions for F2B figures.

class `videof2b.core.figures.Figure`(*R=None, actuals=None, **kwargs*)

Bases: `object`

Base class for all F2B figures.

static create(*which_figure, R=None, actuals=None, **kwargs*)

Factory method for creating a given Figure on a sphere of radius R with specified actual path points.

fit()

Perform best fit of actual points against the nominals of the figure.

get_nom_point(*a, b, c, *t*)

Returns the nominal point at a given $0.0 < t < 1.0$ using the figure's parameters.

u

Suggested count of nominal points for a reasonably smooth-looking figure. Override in subclasses as needed. This is primarily used for drawing.

class `videof2b.core.figures.FigureDiagnostics`(*enabled=False*)

Bases: `object`

Contains the diagnostic settings in a figure.

class `videof2b.core.figures.InsideLoops`(*R=None, actuals=None, **kwargs*)

Bases: `Figure`

Represents three consecutive inside loops per F2B Rule 4.2.15.5 and Diagram 4.J.3 in the Annex. `kwargs`:

Parameters

enable_diags – enables diagnostic output and plotting of various behind-the-scenes stuff.

fit()

Fit actual flight path to this Figure's nominal path.

`videof2b.core.figures.find_min_gss`(*f, a, b, eps=0.0001*)

Find Minimum by Golden Section Search Method Returns the value of x that minimizes the function f(x) on interval [a, b]

`videof2b.core.figures.test()`

Test cases.

videof2b.core.flight module

Defines a recorded flight.

```
class videof2b.core.flight.Flight(vid_path, is_live=False, cal_path=None, **kwargs)
    Bases: QObject
    Contains information about a flight to be processed.
    add_locator_point(point)
        Add a potential locator point.
        Return type
        None
    clear_locator_points()
        Clear all locator points.
    locator_points_changed
    locator_points_defined
    obj_point_names = ('circle center', 'front marker', 'left marker', 'right marker')
    on_locator_points_changed()
        Signals that locator points changed, and the new instruction message.
    pop_locator_point()
        Remove the last added locator point, if it exists.
        Return type
        None
    restart()
        Restart this flight's video stream.
    staticMetaObject = PySide6.QtCore.QMetaObject("Flight" inherits "QObject":  Methods:
    #5 type=Signal, signature=locator_points_changed(PyObject,QString),
    parameters=PyObject, QString #6 type=Signal, signature=locator_points_defined() )
```

videof2b.core.geometry module

General geometry related to F2B figures.

```
exception videof2b.core.geometry.ArgumentError
    Bases: Exception
    Thrown when the provided combination of arguments is invalid.
class videof2b.core.geometry.Fillet(R, r, psi, is_degrees=False)
    Bases: object
    Represents a spherical fillet.
    On a sphere of radius R, a fillet is defined as an arc of a small circle of radius r between two great arcs of the
    sphere with angle  $\psi$  ( $\psi$ ) between the arcs such that the small circle is tangent to both arcs. The constructor of
    this class tries to calculate the parameters of the fillet via the calculate\(\) method.
```

Parameters

- **R** (float) – radius of the sphere.
- **r** (float) – radius of the fillet.
- **psi** (float) – angle between two great arcs that define the fillet.
- **is_degrees** (Optional[bool], default: False) – If True, psi is given in degrees, otherwise it is given in radians.

calculate()

Calculate the fillet as follows:

Given two intersecting planes with angle ψ between them, a cone of slant height R and base radius r whose apex rests on the intersection line of the planes will rest tangent to both planes when the cone's axis makes an angle θ with the intersection line of the planes in the bisecting plane.

If we set up a coordinate system on the cone's base such that: :rtype: None

- the origin is at the cone's apex;
- the $-z$ axis is along the cone's axis toward the cone's base; and
- the $+x$ axis is toward the intersection line,

then the coordinates of the points of tangency between the cone and the planes are $(x_p, y_p, -d)$ and $(x_p, -y_p, -d)$.

Let β be the central angle of the arc along the cone's base that joins the two tangency points on the side of the intersection (the shorter of the two possible arcs).

Perform the following:

- Ensure that tangency is possible. This requires that

$$2\alpha \leq \psi \leq \pi$$

where $\alpha = \arcsin\left(\frac{r}{R}\right)$ is the half-angle of the cone's apex.

If this condition fails, the instance attribute `is_valid` is set to False and this method returns early.

- Find angle θ (Gorjanc solution). Store it in the instance attribute `theta`.
- Find x_p , y_p , and d . Store them in instance attributes `x_p`, `y_p`, and `d`, respectively.
- Find angle β . Store it in the instance attribute `beta`.
- Set `is_valid` to True and return.

See also:

method `get_fillet_theta()`.

static get_fillet_theta(R, r, psi)

Calculate the angle θ between cone axis and intersection line of the planes that are tangent to a [Fillet](#).

Implements the [Gorjanc](#) solution. Values of x_p and y_p follow from this as well.

Parameters

- **R** (float) – radius of the sphere on which the fillet is defined.
- **r** (float) – radius of the fillet, which is also equal to the base radius of the fillet's cone.
- **psi** (float) – angle between the planes that are tangent to the fillet's cone.

Return type
float

Note: This method is provided as a static method for optimizer use in addition to use by *Fillet* here.

See also:

The *calculate()* method.

`videof2b.core.geometry.angle(a, b)`

Calculate the angle between two vectors in radians.

Parameters

- **a** (Union[_SupportsArray[dtype], _NestedSequence[_SupportsArray[dtype]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – any vector in 2D or 3D.
- **b** (Union[_SupportsArray[dtype], _NestedSequence[_SupportsArray[dtype]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – any vector in 2D or 3D.

Return type
float

Returns

the angle between a and b in radians.

`videof2b.core.geometry.calc_equilateral_sigma(height=0.7853981633974483)`

Solve for the side of an equilateral spherical triangle whose height is given.

This is the companion to the *get_equilateral_height()* function.

Parameters

height (Optional[float], default: 0.7853981633974483) – height of the equilateral spherical triangle, in radians. Defaults to $\frac{\pi}{4}$.

Return type
float

Returns

side angle σ , in radians.

`videof2b.core.geometry.calc_tri_loop_params(R, r, target_elev=0.7853981633974483)`

Calculate the basic parameters of a triangular loop figure on a sphere.

Given an equilateral spherical triangle on the surface of a sphere of radius R such that the top of a corner turn of radius r is located at target_elev on the sphere, calculate:

- The central angle σ of the side of the triangle,
- The angle ϕ between adjacent sides of the triangle.

Parameters

- **R** (float) – radius of the sphere.
- **r** (float) – radius of the loop's corner turns.
- **target_elev** (Optional[float], default: 0.7853981633974483) – elevation of the highest point in the top turn. Defaults to $\frac{\pi}{4}$.

Return type

Tuple[float]

Returns

sigma, phi

All angles are in radians.

`videof2b.core.geometry.cartesian_to_spherical(p)`

Convert a point from Cartesian coordinates to elevation-based spherical coordinates.

Parameters

p (Union[_SupportsArray[dtype], _NestedSequence[_SupportsArray[dtype]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – an array or sequence representing a point (x, y, z) in Cartesian space.

Return type

ndarray

Returns

an array containing (r, theta, phi) where

- **r** = radius,
- **theta** = elevation angle θ ,
- **phi** = azimuth angle ϕ .

Seealso

the inverse conversion function is `spherical_to_cartesian()`.

All angles are in radians.

`videof2b.core.geometry.get_arc(r, alpha, rho=100)`

Create an array of 3D points that represent a circular arc.

Return 3D points for an arc of radius **r** and included angle **alpha** with point density **rho**, where **rho** is the number of points per 2π . Arc center is (0, 0, 0). The arc lies in the *xy* plane. The arc starts at zero angle, i.e., at (r, 0, 0), and proceeds counterclockwise until it ends at **alpha**. Angle measurements are in radians. The endpoint is always included.

Parameters

- **r** (float) – radius of the arc.
- **alpha** (float) – included angle of the arc in radians.
- **rho** (Optional[int], default: 100) – angular density of generated points. Defaults to 100.

Return type

ndarray

Returns

(N, 3) array of points, where $N \geq 3$ and is proportional to **alpha** and **rho**.

Warning: The meaning of the **rho** parameter may change in the future from angular density to circumferential (linear) density to provide more consistent point spacing on arcs of different radii in the same scene.

`videof2b.core.geometry.get_cone_alpha(R, r)`

Calculates the half-aperture of a cone with slant height R and base radius r .

Parameters

- **R** (float) – slant height of cone.
- **r** (float) – base radius of cone.

Return type

float

Returns

angle α in radians.

`videof2b.core.geometry.get_cone_d(R, r)`

Perpendicular height of a cone with slant height R and base radius r .

Parameters

- **R** (float) – slant height of cone.
- **r** (float) – base radius of cone.

Raises

`ValueError` if $R < r$

Return type

float

Returns

height d of the cone.

`videof2b.core.geometry.get_cone_delta(alpha, theta=None, beta=None)`

Calculate the properties of a cone rotated from the flight base to a certain elevation.

Consider a base cone whose axis lies in the xy plane, and whose ruled surface contains the y axis. Rotate this cone around the x axis by an angle β such that the elevation of the cone's axis is at angle θ . This result is important because it preserves the cone's tangency point with the yz plane after the rotation. In the cone's base plane, a line segment from the cone axis to this point of tangency lies in the xy plane when the cone is unrotated (the "base" cone). After rotation of the cone by β , this same line segment is no longer parallel to the xy plane. Effectively, it has been rotated around the cone's axis by an angle that we hereby call δ . The goal is to calculate δ and one of the other angles such that the caller has all three angles δ , θ , and β at its disposal.

Parameters

- **alpha** (float) – the cone's half-aperture, in radians. This is required.
- **theta** (Optional[float], default: `None`) – the elevation of the cone's axis, in radians.
- **beta** (Optional[float], default: `None`) – the rotation of the cone around the x axis from the base, in radians.

Raises

`ArgumentError` – when `theta` and `beta` arguments are supplied inconsistently, i.e., when both are given or neither is given.

Return type

`Tuple[float]`

Returns

angle δ and one of the missing angles θ or β . Two mutually exclusive cases are possible:

- **theta is known (as in top corners of square loops)**
→ return (delta, beta)
- **beta is known (as in clover loops).**
→ return (delta, theta)

`videof2b.core.geometry.get_equilateral_height(sigma)`

Calculate the height of an equilateral spherical triangle as a function of its side angle σ . Takes advantage of the [cosine rule](#) in spherical trigonometry.

This is the companion to the `calc_equilateral_sigma()` function.

Parameters

sigma (float) – the side angle σ of the spherical triangle.

Return type

float

Returns

the height of the equilateral spherical triangle in radians.

`videof2b.core.geometry.get_equilateral_phi(sigma)`

Calculate the angle between sides of an equilateral spherical triangle as a function of side angle σ . See the derivation [here](#).

Parameters

sigma (float) – the side angle σ of the spherical triangle.

Return type

float

Returns

the angle ϕ between the sides of the equilateral spherical triangle.

`videof2b.core.geometry.spherical_to_cartesian(p)`

Convert a point from elevation-based spherical coordinates to Cartesian coordinates.

Parameters

p (Union[_SupportsArray[dtype], _NestedSequence[_SupportsArray[dtype]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – an array or sequence like (r, theta, phi) where

- r = radius,
- theta = elevation angle θ ,
- phi = azimuth angle ϕ .

Return type

ndarray

Returns

an array containing (x, y, z).

Seealso

the inverse conversion function is `cartesian_to_spherical()`.

All angles are in radians.

videof2b.core.imaging module

Imaging functions.

`videof2b.core.imaging.cv_img_to_qimg(cv_img)`

Convert a cv2 image to a QImage for display in QPixmap objects.

Return type

QImage

videof2b.core.processor module

The main flight processor in VideoF2B.

class `videof2b.core.processor.ProcessorReturnCodes(value)`

Bases: `IntEnum`

Definition of the return codes from VideoProcessor's processing loop.

EXCEPTION_OCCURRED = -2

NORMAL = 0

POSE_ESTIMATION_FAILED = 2

TOO_MANY_EMPTY_FRAMES = 3

UNDEFINED = -1

USER_CANCELED = 1

class `videof2b.core.processor.ProcessorSettings`

Bases: `object`

Stores persistable user settings.

im_width = 960

live_videos = `PosixPath('../VideoF2B_videos')`

max_track_time = 15

perform_3d_tracking = `False`

sphere_rot_delta = 0.5

sphere_xy_delta = 0.1

class `videof2b.core.processor.VideoProcessor`

Bases: `QObject`, [*StoreProperties*](#)

Main video processor. Handles processing of a video input from start to finish.

add_locator_point(*point*)

Add a potential locator point.

ar_geometry_available

clear_track()

Clear the aircraft's existing flight track.

error_occurred**finished****load_flight(*flight*)**

Load a Flight and prepare for processing.

Parameters

flight (*Flight*) – a properly populated Flight instance.

Return type

None

locating_started**locator_points_changed****locator_points_defined****manipulate_sphere(*command*)**

Manipulate the AR sphere via the specified command.

Return type

None

mark_figure(*is_start*)

Mark the start/end of a tracked figure.

Parameters

is_start (bool) – True to mark the start of the figure, False to mark the end.

Return type

None

new_frame_available**on_locator_points_changed(*points, msg*)**

Handles changes in locator points during the camera locating procedure.

on_locator_points_defined()

Locator points are completely defined. Let the world know.

pause_resume()

Pause/Resume processing at the current frame. Allows the following functionality with immediate feedback while paused: * To quit processing. * To clear the track. * To manipulate sphere rotation and movement.

paused**pop_locator_point(_)**

Remove the last locator point, if one exists.

progress_updated**relocate()**

Relocate the flight.

run()

Run the processor.

```
staticMetaObject = PySide6.QtCore.QMetaObject("VideoProcessor" inherits "QObject":
Methods: #5 type=Signal, signature=locating_started() #6 type=Signal,
signature=locator_points_changed(PyObject,QString), parameters=PyObject, QString #7
type=Signal, signature=locator_points_defined() #8 type=Signal,
signature=ar_geometry_available(bool), parameters=bool #9 type=Signal,
signature=new_frame_available(QImage), parameters=QImage #10 type=Signal,
signature=progress_updated(PyObject), parameters=PyObject #11 type=Signal,
signature=finished(int), parameters=int #12 type=Signal, signature=track_cleared()
#13 type=Signal, signature=paused(bool), parameters=bool #14 type=Signal,
signature=error_occurred(QString,QString), parameters=QString, QString )
```

stop()

Respond to a “nice” request to stop our processing loop.

stop_locating()

Cancel the flight locating procedure.

track_cleared**update_figure_diags(val)**

Update figure diags state in the drawing.

Return type

None

update_figure_state(figure_type, val)

Update figure state in the drawing.

Return type

None

update_maneuver_endpts(val)

Update maneuver endpoints in the drawing.

Return type

None

videof2b.core.projection module

Project image points onto world sphere.

```
videof2b.core.projection.project_image_point_to_sphere(frame, cam, radius, center, img_point,
                                                         data_writer)
```

Project image point to world sphere given a calibrated camera and frame.

Parameters

- **frame** – the image frame of interest.
- **cam** – instance of CalCamera.
- **radius** – radius of sphere.
- **center** – XYZ location of sphere center wrt world pose estimation.
- **img_point** – the image point that we wish to project onto the sphere.

- **data_writer** – a handle to a file-like object.

`videof2b.core.projection.project_sphere_points_to_image(cam, world_pts, frame=None)`

Project the given sphere points to image space.

Module contents

Core modules of VideoF2B application.

videof2b.ui package

Submodules

videof2b.ui.about_window module

The dialog window for the “about” information.

class `videof2b.ui.about_window.AboutDialog(parent)`

Bases: `QDialog`

About window.

setup_ui()

Designs the UI.

staticMetaObject = `PySide6.QtCore.QMetaObject("AboutDialog" inherits "QDialog":)`

videof2b.ui.camera_cal_dialog module

The dialog window for camera calibration.

class `videof2b.ui.camera_cal_dialog.CameraCalibrationDialog(parent)`

Bases: `QDialog`, [*StoreProperties*](#)

Camera calibration UI.

on_doc_open()

Open the calibration pattern PDF file.

on_fisheye_changed(state)

Update fisheye flag when checkbox state changes.

on_image_display()

Show a lightweight window in full screen with the calibration pattern. Esc key closes this window by default.

on_path_changed(new_path)

Update UI when the calibration path changes.

setup_ui()

Designs the UI.

staticMetaObject = `PySide6.QtCore.QMetaObject("CameraCalibrationDialog" inherits "QDialog":)`

videof2b.ui.camera_director_dialog module

Interactive tool for placing a camera in the field.

```
class videof2b.ui.camera_director_dialog.CamDirectorDialog(parent)
```

Bases: QDialog

Interactive camera placement aid.

```
on_new_solution(_)
```

Update results view when a new solution is available.

```
setup_model()
```

Create the models.

```
setup_ui()
```

Create the UI.

```
staticMetaObject = PySide6.QtCore.QMetaObject("CamDirectorDialog" inherits  
"QDialog": )
```

```
class videof2b.ui.camera_director_dialog.CamDirectorInputsModel(biz_obj, parent=None)
```

Bases: QAbstractListModel

Model that represents the cam director's inputs.

```
data(index, role=Ellipsis)
```

Return type

Any

```
flags(index)
```

Return type

~qtpy.QtCore.

```
headerData(section, orientation, role=Ellipsis)
```

Return type

Any

```
rowCount(parent=Ellipsis)
```

Return type

int

```
setData(index, value, role=Ellipsis)
```

Return type

bool

```
staticMetaObject = PySide6.QtCore.QMetaObject("CamDirectorInputsModel" inherits  
"QAbstractListModel": )
```

```
class videof2b.ui.camera_director_dialog.CamDirectorResultsModel(biz_obj, parent=None)
```

Bases: QAbstractTableModel

Model that represents the cam director's results.

columnCount(*parent=Ellipsis*)

Return type
int

data(*index, role=Ellipsis*)

Return type
Any

flags(*index*)

Return type
~qtpy.QtCore.

headerData(*section, orientation, role=Ellipsis*)

Return type
Any

rowCount(*parent=Ellipsis*)

Return type
int

staticMetaObject = PySide6.QtCore.QMetaObject("CamDirectorResultsModel" inherits "QAbstractTableModel":)

videof2b.ui.exception_dialog module

The exception dialog form.

class videof2b.ui.exception_dialog.**ExceptionDialog**(*exc_msg*)

Bases: QDialog, [StoreProperties](#)

User-friendly exception dialog.

exec()

Show the dialog.

on_attach_file_button_clicked()

Attach files to the bug report e-mail.

on_description_updated()

Update the minimum number of characters needed in the description.

on_save_report_button_clicked()

Save exception log and system information to a file.

setup_ui()

Set up the UI.

staticMetaObject = PySide6.QtCore.QMetaObject("ExceptionDialog" inherits "QDialog":)

videof2b.ui.icons module

Provides icons for the VideoF2B application.

class videof2b.ui.icons.**MyIcons**(*args, **kwargs)

Bases: object

Provide application-wide icons.

videof2b.ui.load_flight_dialog module

The dialog that loads the input video.

class videof2b.ui.load_flight_dialog.**LoadFlightDialog**(parent)

Bases: QDialog, [StoreProperties](#)

The dialog window that collects user inputs for a Flight instance.

accept()

Create a Flight instance if all inputs are valid. Store the instance as our *flight* attribute.

Return type

None

on_skip_locate_changed()

Enable/disable the measurement widgets when the “skip locate” checkbox changes.

setup_ui()

Lay out the UI elements

staticMetaObject = PySide6.QtCore.QMetaObject("LoadFlightDialog" inherits "QDialog":
)

videof2b.ui.main_window module

The main GUI window of VideoF2B application.

class videof2b.ui.main_window.**MainWindow**

Bases: QMainWindow, [UIMainWindow](#), [StoreProperties](#)

The main UI window of the VideoF2B application.

clear_track

closeEvent(event)

Overridden to handle the closing of the main window in a safe manner. Handles all exit/close/quit requests here, ensuring all threads are stopped before we close.

figure_diags_changed

figure_mark

figure_state_changed

load_settings()

Load the settings of MainWindow.

locating_completed

maneuver_endpts_changed

manipulate_sphere

on_ar_geometry_available(*is_available*)

Update UI controls based on availability of AR geometry.

on_cal_finished(*retcode*)

Handle return codes and any possible exceptions that are reported by CameraCalibrator when its processing loop finishes. Update the UI as appropriate.

on_cal_thread_finished()

Handle cleanup when the calibrator thread finishes.

on_calibrate_cam()

Calibrate a camera via a specified video file.

on_chk_diag_changed()

Tell the processor to toggle the drawn state of figure diagnostics.

on_chk_endpts_changed()

Tell the processor to toggle the drawn state of maneuver start/end points.

on_chk_figure_changed()

Tell the processor to toggle the drawn state of a figure.

on_clear_track()

Clear the aircraft's existing flight track.

on_figure_end()

Mark the end of a figure in 3D.

on_figure_start()

Mark the start of a figure in 3D.

on_help_about()

Display About window.

on_load_flight()

Loads a flight via LoadFlightDialog and starts processing it.

on_loc_pts_changed(*points, msg*)

Echoes changes in the VideoProcessor's locator points and updates the instruction message.

on_loc_pts_defined()

Present the user with a confirm/redefine choice via messagebox.

on_locating_started()

Prepare UI for the camera locating procedure.

on_move_east()

Move AR sphere East.

on_move_north()

Move AR sphere North.

on_move_reset()

Reset AR sphere's center to world origin.

on_move_south()

Move AR sphere South.

on_move_west()

Move AR sphere West.

on_next_figure()

Advance the current figure checkbox to next figure if appropriate.

Behavior: If multiple figures are checked, do nothing. If the last figure is checked, do nothing. If no figures are checked, check the first one. In all other cases, uncheck the current figure and check the next one.

on_pause_resume()

Pause/resume processing at the current frame.

on_paused_resumed(*is_paused*)

Slot that responds to VideoProcessor's signal.

on_place_cam()

Open the camera placement dialog.

on_proc_finished(*retcode*)

Handle return codes and any possible exceptions that are reported by VideoProcessor when its processing loop finishes. Also update the UI as appropriate.

on_proc_starting()

Handle required preparations when the processing thread starts.

on_proc_thread_finished()

Handle cleanup when the processing thread finishes.

on_progress_updated(*data*)

Display video processing progress.

on_relocate_cam()

Relocate the camera.

on_restart_flight()

Reload the current flight and restart it.

on_rotate_ccw()

Rotate AR sphere CCW.

on_rotate_cw()

Rotate AR sphere CW.

on_stop_proc()

Request to stop the video processor.

on_track_cleared()

Slot that responds to the processor's signal.

pause_resume

relocate_cam

save_settings()

Save the settings of MainWindow.

start_cal_thread()

Starts the calibrator on a worker thread.

start_proc_thread()

Starts the video processor on a worker thread.

```
staticMetaObject = PySide6.QtCore.QMetaObject("MainWindow" inherits "QMainWindow":
Methods: #40 type=Signal, signature=stop_processor() #41 type=Signal,
signature=locating_completed() #42 type=Signal, signature=clear_track() #43
type=Signal, signature=figure_state_changed(PyObject,bool), parameters=PyObject,
bool #44 type=Signal, signature=figure_diags_changed(bool), parameters=bool #45
type=Signal, signature=maneuver_endpts_changed(bool), parameters=bool #46
type=Signal, signature=relocate_cam() #47 type=Signal,
signature=manipulate_sphere(PyObject), parameters=PyObject #48 type=Signal,
signature=figure_mark(bool), parameters=bool #49 type=Signal,
signature=pause_resume() )
```

stop_processor

class videof2b.ui.main_window.UIMainWindow

Bases: object

Define the UI layout.

setup_ui(*main_window*)

Create the UI here.

videof2b.ui.style module

Define UI styles.

videof2b.ui.video_window module

The video window of VideoF2B application.

class videof2b.ui.video_window.VideoWindow(*parent*, ***kwargs*)

Bases: QLabel

The window that displays video frames during processing.

clear()

Overridden method to clear our custom pixmap.

Return type

None

property is_mouse_enabled

Indicates whether the video window reacts to mouse events.

mousePressEvent(*event*)

Overridden event so that we react to mouse clicks as needed.

Return type

None

point_added**point_removed****resizeEvent**(*event*)

Overridden event so that our window resizes with its parent while maintaining the loaded image's original aspect ratio.

Return type

None

staticMetaObject = PySide6.QtCore.QMetaObject("VideoWindow" inherits "QLabel":
Methods: #47 type=Signal, signature=point_added(PyObject), parameters=PyObject #48
type=Signal, signature=point_removed(PyObject), parameters=PyObject)

update_frame(*frame*)

Set a new video frame in the window.

Return type

None

videof2b.ui.widgets module

This module contains various custom widgets for the UI.

class videof2b.ui.widgets.**FileDialog**

Bases: QFileDialog

A wrapped QFileDialog compatible with Path objects.

classmethod **getExistingDirectory**(*args, **kwargs)

Thin wrapper of *getExistingDirectory* compatible with Path objects as args.

Return type

pathlib.Path

classmethod **getOpenFileName**(*args, **kwargs)

Thin wrapper of *getOpenFileName* compatible with Path objects as args.

Return type

tuple[pathlib.Path, str]

classmethod **getOpenFileNames**(*args, **kwargs)

Thin wrapper of *getOpenFileNames* compatible with Path objects as args.

Return type

tuple[list[pathlib.Path], str]

classmethod **getSaveFileName**(*args, **kwargs)

Thin wrapper of *getSaveFileName* compatible with Path objects as args.

Return type

tuple[pathlib.Path | None, str]

```
staticMetaObject = PySide6.QtCore.QMetaObject("FileDialog" inherits "QFileDialog":
)
```

```
class videof2b.ui.widgets.PathEdit(parent=None, path_type=PathEditType.FILES, caption=None,
                                   initial_path=None)
```

Bases: QWidget

Custom QWidget subclass for selecting a file or directory.

on_browse_button_clicked()

Shows the QFileDialog when the browse button is clicked. Emits *path_changed* if appropriate.

Return type

None

on_line_edit_editing_finished()

Updates *path* and emits *path_changed* when the line edit has finished being edited.

Return type

None

on_new_path(path)

If the given path is different from current *path*, updates *path* and emits the *path_changed* Signal.

Parameters

path (*Path*) – The new path

Return type

None

property path

Returns the selected path.

Returns

The selected path

Return type

Path

path_changed

property path_type

Returns the path type. Path type specifies selecting a file or directory.

Returns

The type selected

Return type

PathType

```
staticMetaObject = PySide6.QtCore.QMetaObject("PathEdit" inherits "QWidget":
```

```
Methods: #34 type=Signal, signature=path_changed(PyObject), parameters=PyObject )
```

update_button_tool_tips()

Updates the button tooltips during init and when *path_type* changes.

Return type

None

```
class videof2b.ui.widgets.PathEditType(value)
    Bases: Enum
    Specifies the type of browser in a PathEdit.
    DIRECTORIES = 2
    FILES = 1

class videof2b.ui.widgets.QHLine
    Bases: QFrame
    A horizontal line widget.
    staticMetaObject = PySide6.QtCore.QMetaObject("QHLine" inherits "QFrame": )

class videof2b.ui.widgets.QVLine
    Bases: QFrame
    A vertical line widget.
    staticMetaObject = PySide6.QtCore.QMetaObject("QVLine" inherits "QFrame": )
```

Module contents

The ui module provides the core user interface for VideoF2B

17.1.2 Submodules

17.1.3 videof2b.app module

Main VideoF2B application.

```
class videof2b.app.VideoF2B
    Bases: QObject
    The main application runner for VideoF2B.
    hook_exception(exc_type, value, traceback)
        Add an exception hook so that any uncaught exceptions are displayed in this window rather than someplace
        where users cannot see it and cannot report when we encounter these problems.

        Parameters
        • exc_type – The class of exception.
        • value – The actual exception object.
        • traceback – A traceback object with the details of where the exception occurred.

    run(app)
        The main method. Makes the necessary preparations, then runs the given app.

    static set_busy_cursor()
        Sets the Busy Cursor for the Application.

    static set_normal_cursor()
        Sets the Normal Cursor for the Application.
```

```
staticMetaObject = PySide6.QtCore.QMetaObject("VideoF2B" inherits "QObject": )
```

`videof2b.app.start()`
Programmatic entry point of VideoF2B.

17.1.4 Module contents

The *videof2b* package contains all VideoF2B functionality.

GUIDE FOR EDITORS OF THIS GUIDE

This documentation uses **reStructured Text** (reST) as its markup language. The structure and syntax borrow shamelessly from the excellent [Inkscape Beginners' Guide](#). Please refer to the [Sample Chapter](#) for an outline of the syntax and most of the elements used in this guide.

Note: Only some of the custom styling used in the Inkscape guide applies to this guide. When in doubt, refer to files in the `static` folder of this documentation for implementation details. Edit them as necessary to suit our styling needs.

PYTHON MODULE INDEX

V

- [videof2b](#), 79
- [videof2b.app](#), 78
- [videof2b.core](#), 69
 - [videof2b.core.calibration](#), 53
 - [videof2b.core.camera](#), 54
 - [videof2b.core.camera_director](#), 54
 - [videof2b.core.common](#), 51
 - [videof2b.core.common.path](#), 49
 - [videof2b.core.common.settings](#), 50
 - [videof2b.core.common.singleton](#), 51
 - [videof2b.core.common.store](#), 51
 - [videof2b.core.detection](#), 55
 - [videof2b.core.drawing](#), 55
 - [videof2b.core.figure_tracker](#), 58
 - [videof2b.core.figures](#), 59
 - [videof2b.core.flight](#), 60
 - [videof2b.core.geometry](#), 60
 - [videof2b.core.imaging](#), 66
 - [videof2b.core.processor](#), 66
 - [videof2b.core.projection](#), 68
- [videof2b.ui](#), 78
 - [videof2b.ui.about_window](#), 69
 - [videof2b.ui.camera_cal_dialog](#), 69
 - [videof2b.ui.camera_director_dialog](#), 70
 - [videof2b.ui.exception_dialog](#), 71
 - [videof2b.ui.icons](#), 72
 - [videof2b.ui.load_flight_dialog](#), 72
 - [videof2b.ui.main_window](#), 72
 - [videof2b.ui.style](#), 75
 - [videof2b.ui.video_window](#), 75
 - [videof2b.ui.widgets](#), 76

A

A (videof2b.core.camera_director.CamDirector property), 54
 AboutDialog (class in videof2b.ui.about_window), 69
 accept() (videof2b.ui.load_flight_dialog.LoadFlightDialog method), 72
 add() (videof2b.core.common.store.Store method), 51
 add() (videof2b.core.drawing.Scene method), 57
 add_actual_point() (videof2b.core.figure_tracker.FigureTracker method), 58
 add_diag() (videof2b.core.drawing.Scene method), 57
 add_endpt() (videof2b.core.drawing.Scene method), 57
 add_locator_point() (videof2b.core.flight.Flight method), 60
 add_locator_point() (videof2b.core.processor.VideoProcessor method), 66
 angle() (in module videof2b.core.geometry), 62
 application (videof2b.core.common.store.StoreProperties property), 51
 AR, 47
 ar_geometry_available (videof2b.core.processor.VideoProcessor attribute), 66
 ArgumentError, 60
 Augmented Reality, 47

B

Base, 47
 BLACK (videof2b.core.drawing.Colors attribute), 55
 BLUE (videof2b.core.drawing.Colors attribute), 55

C

C (videof2b.core.camera_director.CamDirector property), 54
 calc_equilateral_sigma() (in module videof2b.core.geometry), 62
 calc_tri_loop_params() (in module videof2b.core.geometry), 62
 CalCamera (class in videof2b.core.camera), 54
 calculate() (videof2b.core.geometry.Fillet method), 61

CalibratorReturnCodes (class in videof2b.core.calibration), 53
 cam_distance_limits (videof2b.core.camera_director.CamDirector property), 55
 cam_tangent_elev_limits (videof2b.core.camera_director.CamDirector property), 55
 cam_view_limits (videof2b.core.camera_director.CamDirector property), 55
 CamDirector (class in videof2b.core.camera_director), 54
 CamDirectorDialog (class in videof2b.ui.camera_director_dialog), 70
 CamDirectorInputsModel (class in videof2b.ui.camera_director_dialog), 70
 CamDirectorResultsModel (class in videof2b.ui.camera_director_dialog), 70
 CameraCalibrationDialog (class in videof2b.ui.camera_cal_dialog), 69
 CameraCalibrator (class in videof2b.core.calibration), 53
 cartesian_to_spherical() (in module videof2b.core.geometry), 63
 clear() (videof2b.core.detection.Detector method), 55
 clear() (videof2b.ui.video_window.VideoWindow method), 75
 clear_locator_points() (videof2b.core.flight.Flight method), 60
 clear_track (videof2b.ui.main_window.MainWindow attribute), 72
 clear_track() (videof2b.core.processor.VideoProcessor method), 66
 closeEvent() (videof2b.ui.main_window.MainWindow method), 72
 Colors (class in videof2b.core.drawing), 55
 columnCount() (videof2b.ui.camera_director_dialog.CamDirectorResultsModel method), 70
 create() (videof2b.core.common.store.Store class method), 51
 create() (videof2b.core.figures.Figure static method), 59

`cv_img_to_qimg()` (in module `videof2b.core.imaging`), 66

CYAN (`videof2b.core.drawing.Colors` attribute), 55

D

DarkGreen (`videof2b.core.drawing.Colors` attribute), 55

DashedPolyline (class in `videof2b.core.drawing`), 56

`data()` (`videof2b.ui.camera_director_dialog.CamDirectorInputsModel` method), 70

`data()` (`videof2b.ui.camera_director_dialog.CamDirectorResultsModel` method), 71

DEFAULT_A (`videof2b.core.camera_director.CamDirector` attribute), 54

DEFAULT_C (`videof2b.core.camera_director.CamDirector` attribute), 54

DEFAULT_G (`videof2b.core.camera_director.CamDirector` attribute), 54

DEFAULT_N (`videof2b.core.drawing.Drawing` attribute), 56

DEFAULT_R (`videof2b.core.camera_director.CamDirector` attribute), 54

Detector (class in `videof2b.core.detection`), 55

`diags_on` (`videof2b.core.drawing.Scene` property), 57

DIRECTORIES (`videof2b.ui.widgets.PathEditType` attribute), 78

`draw()` (`videof2b.core.drawing.DashedPolyline` method), 56

`draw()` (`videof2b.core.drawing.Drawing` method), 56

`draw()` (`videof2b.core.drawing.DummyScene` method), 56

`draw()` (`videof2b.core.drawing.Plot` method), 57

`draw()` (`videof2b.core.drawing.Polyline` method), 57

`draw()` (`videof2b.core.drawing.Scatter` method), 57

`draw()` (`videof2b.core.drawing.Scene` method), 58

`draw_diags` (`videof2b.core.drawing.Drawing` property), 56

`draw_endpts` (`videof2b.core.drawing.Drawing` property), 56

Drawing (class in `videof2b.core.drawing`), 56

DummyScene (class in `videof2b.core.drawing`), 56

E

EdgePolyline (class in `videof2b.core.drawing`), 57

`endpts_on` (`videof2b.core.drawing.Scene` property), 58

`error_occurred` (`videof2b.core.calibration.CameraCalibrator` attribute), 53

`error_occurred` (`videof2b.core.processor.VideoProcessor` attribute), 67

EXCEPTION_OCCURRED (`videof2b.core.calibration.CalibratorReturnCodes` attribute), 53

EXCEPTION_OCCURRED (`videof2b.core.processor.ProcessorReturnCodes` attribute), 66

ExceptionDialog (class in `G` (`videof2b.core.camera_director.CamDirector` property), 71

`exec()` (`videof2b.ui.exception_dialog.ExceptionDialog` method), 71

`export()` (`videof2b.core.figure_tracker.FigureTracker` method), 58

F

Field of view, 47

Figure, 47

Figure (class in `videof2b.core.figures`), 59

`FigureModel.diags_changed`

(`videof2b.ui.main_window.MainWindow` attribute), 72

FIGURE_MAP (`videof2b.core.figure_tracker.FigureTracker` attribute), 58

`figure_mark` (`videof2b.ui.main_window.MainWindow` attribute), 72

`figure_state_changed` (`videof2b.ui.main_window.MainWindow` attribute), 72

FigureDiagnostics (class in `videof2b.core.figures`), 59

FigureTracker (class in `videof2b.core.figure_tracker`), 58

FigureTypes (class in `videof2b.core.common`), 51

FileDialog (class in `videof2b.ui.widgets`), 76

FILES (`videof2b.ui.widgets.PathEditType` attribute), 78

`files_to_paths()` (in module `videof2b.core.common.path`), 49

Fillet (class in `videof2b.core.geometry`), 60

`find_min_gss()` (in module `videof2b.core.figures`), 59

`finish_all()` (`videof2b.core.figure_tracker.FigureTracker` method), 58

`finish_figure()` (`videof2b.core.figure_tracker.FigureTracker` method), 59

`finished` (`videof2b.core.calibration.CameraCalibrator` attribute), 53

`finished` (`videof2b.core.processor.VideoProcessor` attribute), 67

`fit()` (`videof2b.core.figures.Figure` method), 59

`fit()` (`videof2b.core.figures.InsideLoops` method), 59

`flags()` (`videof2b.ui.camera_director_dialog.CamDirectorInputsModel` method), 70

`flags()` (`videof2b.ui.camera_director_dialog.CamDirectorResultsModel` method), 71

Flight (class in `videof2b.core.flight`), 60

Flight circle, 47

Flight hemisphere, 47

Flight radius, 47

FOUR_LEAF_CLOVER (`videof2b.core.common.FigureTypes` attribute), 51

FOV, 47

G

G (class in `G` (`videof2b.core.camera_director.CamDirector` property), 55

get() (*videof2b.core.common.store.Store* method), 51
 get_app_metadata() (in module *videof2b.core.common*), 52
 get_arc() (in module *videof2b.core.geometry*), 63
 get_bundle_dir() (in module *videof2b.core.common*), 52
 get_cone_alpha() (in module *videof2b.core.geometry*), 63
 get_cone_d() (in module *videof2b.core.geometry*), 64
 get_cone_delta() (in module *videof2b.core.geometry*), 64
 get_equilateral_height() (in module *videof2b.core.geometry*), 65
 get_equilateral_phi() (in module *videof2b.core.geometry*), 65
 get_fillet_theta() (*videof2b.core.geometry.Fillet* static method), 61
 get_frozen_path() (in module *videof2b.core.common*), 52
 get_lib_versions() (in module *videof2b.core.common*), 52
 get_nom_point() (*videof2b.core.figures.Figure* method), 59
 getExistingDirectory() (*videof2b.ui.widgets.FileDialog* class method), 76
 getOpenFileName() (*videof2b.ui.widgets.FileDialog* class method), 76
 getOpenFileNames() (*videof2b.ui.widgets.FileDialog* class method), 76
 getSaveFileName() (*videof2b.ui.widgets.FileDialog* class method), 76
 GRAY20 (*videof2b.core.drawing.Colors* attribute), 56
 Great circle, 47
 GREEN (*videof2b.core.drawing.Colors* attribute), 56
H
 headerData() (*videof2b.ui.camera_director_dialog.CamDirectorInputModel* method), 70
 headerData() (*videof2b.ui.camera_director_dialog.CamDirectorResultsModel* method), 71
 hook_exception() (*videof2b.app.VideoF2B* method), 78
 Horizontal, 47
 HORIZONTAL_EIGHTS (*videof2b.core.common.FigureTypes* attribute), 51
 HORIZONTAL_SQUARE_EIGHTS (*videof2b.core.common.FigureTypes* attribute), 51
 HOURGLASS (*videof2b.core.common.FigureTypes* attribute), 51
I
 im_width (*videof2b.core.processor.ProcessorSettings* attribute), 66
 INSIDE_LOOPS (*videof2b.core.common.FigureTypes* attribute), 51
 INSIDE_SQUARE_LOOPS (*videof2b.core.common.FigureTypes* attribute), 52
 INSIDE_TRIANGULAR_LOOPS (*videof2b.core.common.FigureTypes* attribute), 52
 InsideLoops (class in *videof2b.core.figures*), 59
 INSUFFICIENT_VALID_FRAMES (*videof2b.core.calibration.CalibratorReturnCodes* attribute), 53
 Inverted, 47
 INVERTED_FLIGHT (*videof2b.core.common.FigureTypes* attribute), 52
 is_linux() (in module *videof2b.core.common*), 52
 is_mouse_enabled (*videof2b.ui.video_window.VideoWindow* property), 75
 is_win() (in module *videof2b.core.common*), 53
L
 LANDING (*videof2b.core.common.FigureTypes* attribute), 52
 Lateral reference, 47
 launch_document() (in module *videof2b.core.common*), 53
 Level, 47
 live_videos (*videof2b.core.processor.ProcessorSettings* attribute), 66
 load_calibration() (*videof2b.core.camera.CalCamera* method), 54
 load_flight() (*videof2b.core.processor.VideoProcessor* method), 67
 load_settings() (*videof2b.ui.main_window.MainWindow* method), 72
 LoadFlightDialog (class in *videof2b.ui.load_flight_dialog*), 72
 locate() (*videof2b.core.camera.CalCamera* method), 54
 locate() (*videof2b.core.drawing.Drawing* method), 56
 locating_completed (*videof2b.ui.main_window.MainWindow* attribute), 72
 locating_started (*videof2b.core.processor.VideoProcessor* attribute), 67
 locator_points_changed (*videof2b.core.flight.Flight* attribute), 60
 locator_points_changed (*videof2b.core.processor.VideoProcessor* attribute), 67
 locator_points_defined (*videof2b.core.flight.Flight* attribute), 60
 locator_points_defined (*videof2b.core.processor.VideoProcessor* attribute), 67

attribute), 67

M

MAGENTA (*videof2b.core.drawing.Colors attribute*), 56

MainWindow (*class in videof2b.ui.main_window*), 72

Maneuver, 48

maneuver_endpts_changed
(*videof2b.ui.main_window.MainWindow attribute*), 73

ManeuverPoint (*class in videof2b.core.drawing*), 57

manipulate_sphere (*videof2b.ui.main_window.MainWindow attribute*), 73

manipulate_sphere()
(*videof2b.core.processor.VideoProcessor method*), 67

Manoeuvre, 48

mark_figure() (*videof2b.core.processor.VideoProcessor method*), 67

max_track_time (*videof2b.core.processor.ProcessorSettings attribute*), 66

module

- videof2b, 79
- videof2b.app, 78
- videof2b.core, 69
- videof2b.core.calibration, 53
- videof2b.core.camera, 54
- videof2b.core.camera_director, 54
- videof2b.core.common, 51
- videof2b.core.common.path, 49
- videof2b.core.common.settings, 50
- videof2b.core.common.singleton, 51
- videof2b.core.common.store, 51
- videof2b.core.detection, 55
- videof2b.core.drawing, 55
- videof2b.core.figure_tracker, 58
- videof2b.core.figures, 59
- videof2b.core.flight, 60
- videof2b.core.geometry, 60
- videof2b.core.imaging, 66
- videof2b.core.processor, 66
- videof2b.core.projection, 68
- videof2b.ui, 78
- videof2b.ui.about_window, 69
- videof2b.ui.camera_cal_dialog, 69
- videof2b.ui.camera_director_dialog, 70
- videof2b.ui.exception_dialog, 71
- videof2b.ui.icons, 72
- videof2b.ui.load_flight_dialog, 72
- videof2b.ui.main_window, 72
- videof2b.ui.style, 75
- videof2b.ui.video_window, 75
- videof2b.ui.widgets, 76

Momentarily, 48

Momentary, 48

mousePressEvent() (*videof2b.ui.video_window.VideoWindow method*), 75

move_center_x() (*videof2b.core.drawing.Drawing method*), 56

move_center_y() (*videof2b.core.drawing.Drawing method*), 56

MOVE_EAST (*videof2b.core.common.SphereManipulations attribute*), 52

MOVE_NORTH (*videof2b.core.common.SphereManipulations attribute*), 52

MOVE_SOUTH (*videof2b.core.common.SphereManipulations attribute*), 52

MOVE_WEST (*videof2b.core.common.SphereManipulations attribute*), 52

MyIcons (*class in videof2b.ui.icons*), 72

N

new_frame_available
(*videof2b.core.calibration.CameraCalibrator attribute*), 53

new_frame_available
(*videof2b.core.processor.VideoProcessor attribute*), 67

NO_VALID_FRAMES (*videof2b.core.calibration.CalibratorReturnCodes attribute*), 53

Nominal, 48

NORMAL (*videof2b.core.calibration.CalibratorReturnCodes attribute*), 53

NORMAL (*videof2b.core.processor.ProcessorReturnCodes attribute*), 66

O

obj_point_names (*videof2b.core.flight.Flight attribute*), 60

on_ar_geometry_available()
(*videof2b.ui.main_window.MainWindow method*), 73

on_attach_file_button_clicked()
(*videof2b.ui.exception_dialog.ExceptionDialog method*), 71

on_browse_button_clicked()
(*videof2b.ui.widgets.PathEdit method*), 77

on_cal_finished() (*videof2b.ui.main_window.MainWindow method*), 73

on_cal_thread_finished()
(*videof2b.ui.main_window.MainWindow method*), 73

on_calibrate_cam() (*videof2b.ui.main_window.MainWindow method*), 73

on_chk_diag_changed()
(*videof2b.ui.main_window.MainWindow method*), 73

on_chk_endpts_changed()
(*videof2b.ui.main_window.MainWindow method*), 73

`method`), 73
`on_chk_figure_changed()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_clear_track()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_description_updated()`
(`videof2b.ui.exception_dialog.ExceptionDialog`
`method`), 71
`on_doc_open()` (`videof2b.ui.camera_cal_dialog.CameraCalibrationDialog`
`method`), 69
`on_figure_end()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_figure_start()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_fisheye_changed()`
(`videof2b.ui.camera_cal_dialog.CameraCalibrationDialog`
`method`), 69
`on_help_about()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_image_display()` (`videof2b.ui.camera_cal_dialog.CameraCalibrationDialog`
`method`), 69
`on_line_edit_editing_finished()`
(`videof2b.ui.widgets.PathEdit` `method`), 77
`on_load_flight()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_loc_pts_changed()`
(`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_loc_pts_defined()`
(`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_locating_started()`
(`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_locator_points_changed()`
(`videof2b.core.flight.Flight` `method`), 60
`on_locator_points_changed()`
(`videof2b.core.processor.VideoProcessor`
`method`), 67
`on_locator_points_defined()`
(`videof2b.core.processor.VideoProcessor`
`method`), 67
`on_move_east()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_move_north()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_move_reset()` (`videof2b.ui.main_window.MainWindow`
`method`), 73
`on_move_south()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_move_west()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_new_path()` (`videof2b.ui.widgets.PathEdit` `method`), 77
`on_new_solution()` (`videof2b.ui.camera_director_dialog.CamDirectorDialog`
`method`), 70
`on_next_figure()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_path_changed()` (`videof2b.ui.camera_cal_dialog.CameraCalibrationDialog`
`method`), 69
`on_pause_resume()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_pause_dialog_resumed()`
(`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_place_cam()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_proc_finished()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_proc_starting()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_proc_thread_finished()`
(`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_progress_updated()`
(`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_relocate_cam()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_restart_flight()`
(`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_rotate_ccw()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_rotate_cw()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_save_report_button_clicked()`
(`videof2b.ui.exception_dialog.ExceptionDialog`
`method`), 71
`on_skip_locate_changed()`
(`videof2b.ui.load_flight_dialog.LoadFlightDialog`
`method`), 72
`on_stop_proc()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
`on_track_cleared()` (`videof2b.ui.main_window.MainWindow`
`method`), 74
OUTSIDE_LOOPS (`videof2b.core.common.FigureTypes` at-
tribute), 52
OUTSIDE_SQUARE_LOOPS
(`videof2b.core.common.FigureTypes` attribute),
52
OVERHEAD_EIGHTS (`videof2b.core.common.FigureTypes`
attribute), 52

P

Parallel, 48
path (`videof2b.ui.widgets.PathEdit` property), 77

- path_changed (videof2b.ui.widgets.PathEdit attribute), 77
 path_to_str() (in module videof2b.core.common.path), 49
 path_type (videof2b.ui.widgets.PathEdit property), 77
 PathEdit (class in videof2b.ui.widgets), 77
 PathEditType (class in videof2b.ui.widgets), 77
 pause_resume (videof2b.ui.main_window.MainWindow attribute), 74
 pause_resume() (videof2b.core.processor.VideoProcessor method), 67
 paused (videof2b.core.processor.VideoProcessor attribute), 67
 perform_3d_tracking (videof2b.core.processor.ProcessorSettings attribute), 66
 Plot (class in videof2b.core.drawing), 57
 point_added (videof2b.ui.video_window.VideoWindow attribute), 76
 point_removed (videof2b.ui.video_window.VideoWindow attribute), 76
 Polyline (class in videof2b.core.drawing), 57
 pop_locator_point() (videof2b.core.flight.Flight method), 60
 pop_locator_point() (videof2b.core.processor.VideoProcessor method), 67
 POSE_ESTIMATION_FAILED (videof2b.core.processor.ProcessorReturnCodes attribute), 66
 process() (videof2b.core.detection.Detector method), 55
 ProcessorReturnCodes (class in videof2b.core.processor), 66
 ProcessorSettings (class in videof2b.core.processor), 66
 progress_updated (videof2b.core.calibration.CameraCalibrator attribute), 53
 progress_updated (videof2b.core.processor.VideoProcessor attribute), 67
 project_image_point_to_sphere() (in module videof2b.core.projection), 68
 project_sphere_points_to_image() (in module videof2b.core.projection), 69
- ## Q
- QHLine (class in videof2b.ui.widgets), 78
 QVLine (class in videof2b.ui.widgets), 78
- ## R
- R (videof2b.core.camera_director.CamDirector property), 55
 RED (videof2b.core.drawing.Colors attribute), 56
 relocate() (videof2b.core.processor.VideoProcessor method), 67
 relocate_cam (videof2b.ui.main_window.MainWindow attribute), 74
 remove() (videof2b.core.common.store.Store method), 51
 replace_params() (in module videof2b.core.common.path), 49
 RESET_CENTER (videof2b.core.common.SphereManipulations attribute), 52
 reset_center() (videof2b.core.drawing.Drawing method), 56
 resizeEvent() (videof2b.ui.video_window.VideoWindow method), 76
 restart() (videof2b.core.flight.Flight method), 60
 REVERSE_WINGOVER (videof2b.core.common.FigureTypes attribute), 52
 ROTATE_CCW (videof2b.core.common.SphereManipulations attribute), 52
 ROTATE_CW (videof2b.core.common.SphereManipulations attribute), 52
 rowCount() (videof2b.ui.camera_director_dialog.CamDirectorInputsModel method), 70
 rowCount() (videof2b.ui.camera_director_dialog.CamDirectorResultsModel method), 71
 run() (videof2b.app.VideoF2B method), 78
 run() (videof2b.core.calibration.CameraCalibrator method), 53
 run() (videof2b.core.processor.VideoProcessor method), 67
- ## S
- save_settings() (videof2b.ui.main_window.MainWindow method), 75
 Scatter (class in videof2b.core.drawing), 57
 Scene (class in videof2b.core.drawing), 57
 Segment (class in videof2b.core.drawing), 48
 set_azimuth() (videof2b.core.drawing.Drawing method), 56
 set_busy_cursor() (videof2b.app.VideoF2B static method), 78
 set_normal_cursor() (videof2b.app.VideoF2B static method), 78
 setData() (videof2b.ui.camera_director_dialog.CamDirectorInputsModel method), 70
 Settings (class in videof2b.core.common.settings), 50
 settings (videof2b.core.common.store.StoreProperties property), 51
 setup_model() (videof2b.ui.camera_director_dialog.CamDirectorDialog method), 70
 setup_ui() (videof2b.ui.about_window.AboutDialog method), 69
 setup_ui() (videof2b.ui.camera_cal_dialog.CameraCalibrationDialog method), 69

setup_ui() (videof2b.ui.camera_director_dialog.CamDirectorDialog method), 70
 setup_ui() (videof2b.ui.exception_dialog.ExceptionDialog method), 71
 setup_ui() (videof2b.ui.load_flight_dialog.LoadFlightDialog method), 72
 setup_ui() (videof2b.ui.main_window.UIMainWindow method), 75
 Singleton (class in videof2b.core.common.singleton), 51
 Small circle, 48
 solve() (videof2b.core.camera_director.CamDirector method), 55
 Sphere, 48
 sphere_rot_delta (videof2b.core.processor.ProcessorSettings attribute), 66
 sphere_xy_delta (videof2b.core.processor.ProcessorSettings attribute), 66
 SphereManipulations (class in videof2b.core.common), 52
 spherical_to_cartesian() (in module videof2b.core.geometry), 65
 start() (in module videof2b.app), 79
 start_cal_thread() (videof2b.ui.main_window.MainWindow method), 75
 start_figure() (videof2b.core.figure_tracker.FigureTracker method), 59
 start_proc_thread() (videof2b.ui.main_window.MainWindow method), 75
 staticMetaObject (videof2b.app.VideoF2B attribute), 78
 staticMetaObject (videof2b.core.calibration.CameraCalibrator attribute), 54
 staticMetaObject (videof2b.core.camera.CalCamera attribute), 54
 staticMetaObject (videof2b.core.common.settings.Settings attribute), 50
 staticMetaObject (videof2b.core.flight.Flight attribute), 60
 staticMetaObject (videof2b.core.processor.VideoProcessor attribute), 68
 staticMetaObject (videof2b.ui.about_window.AboutDialog attribute), 69
 staticMetaObject (videof2b.ui.camera_cal_dialog.CameraCalibrationDialog attribute), 69
 staticMetaObject (videof2b.ui.camera_director_dialog.CamDirectorDialog attribute), 70
 staticMetaObject (videof2b.ui.camera_director_dialog.CamDirectorInputsModel attribute), 70
 staticMetaObject (videof2b.ui.camera_director_dialog.CamDirectorResultsModel attribute), 71
 staticMetaObject (videof2b.ui.exception_dialog.ExceptionDialog attribute), 71
 staticMetaObject (videof2b.ui.load_flight_dialog.LoadFlightDialog attribute), 72
 staticMetaObject (videof2b.ui.main_window.MainWindow attribute), 75
 staticMetaObject (videof2b.ui.video_window.VideoWindow attribute), 76
 staticMetaObject (videof2b.ui.widgets.FileDialog attribute), 76
 staticMetaObject (videof2b.ui.widgets.PathEdit attribute), 77
 staticMetaObject (videof2b.ui.widgets.QHLine attribute), 78
 staticMetaObject (videof2b.ui.widgets.QVLine attribute), 78
 stop() (videof2b.core.calibration.CameraCalibrator method), 54
 stop() (videof2b.core.processor.VideoProcessor method), 68
 stop_locating() (videof2b.core.processor.VideoProcessor method), 68
 stop_processor (videof2b.ui.main_window.MainWindow attribute), 75
 Store (class in videof2b.core.common.store), 51
 StoreProperties (class in videof2b.core.common.store), 51
 str_to_path() (in module videof2b.core.common.path), 50
 Straight line, 48
T
 TAKEOFF (videof2b.core.common.FigureTypes attribute), 52
 test() (in module videof2b.core.figures), 59
 TOO_MANY_EMPTY_FRAMES (videof2b.core.processor.ProcessorReturnCodes attribute), 66
 Top of circle, 48
 track_cleared (videof2b.core.processor.VideoProcessor attribute), 68
U
 u (videof2b.core.figures.Figure attribute), 59
 UIMainWindow (class in videof2b.ui.main_window), 75
 UNDEFINED (videof2b.core.calibration.CalibratorReturnCodes attribute), 53
 UNDEFINED (videof2b.core.processor.ProcessorReturnCodes attribute), 66
 undistort() (videof2b.core.camera.CalCamera method), 54
 update_button_tool_tips() (videof2b.ui.widgets.PathEdit method), 77
 update_figure_diags() (videof2b.core.processor.VideoProcessor method), 68

`update_figure_state()`
(*videof2b.core.processor.VideoProcessor*
method), 68

`update_frame()` (*videof2b.ui.video_window.VideoWindow*
method), 76

`update_maneuver_endpts()`
(*videof2b.core.processor.VideoProcessor*
method), 68

`Upright`, 48

`USER_CANCELED` (*videof2b.core.calibration.CalibratorReturnCode* attribute), 53

`USER_CANCELED` (*videof2b.core.processor.ProcessorReturnCode* attribute), 66

`UserError`, 59

V

`value()` (*videof2b.core.common.settings.Settings*
method), 50

`Vertical`, 48

`VERTICAL_EIGHTS` (*videof2b.core.common.FigureTypes*
attribute), 52

`videof2b`
module, 79

`VideoF2B` (class in *videof2b.app*), 78

`videof2b.app`
module, 78

`videof2b.core`
module, 69

`videof2b.core.calibration`
module, 53

`videof2b.core.camera`
module, 54

`videof2b.core.camera_director`
module, 54

`videof2b.core.common`
module, 51

`videof2b.core.common.path`
module, 49

`videof2b.core.common.settings`
module, 50

`videof2b.core.common.singleton`
module, 51

`videof2b.core.common.store`
module, 51

`videof2b.core.detection`
module, 55

`videof2b.core.drawing`
module, 55

`videof2b.core.figure_tracker`
module, 58

`videof2b.core.figures`
module, 59

`videof2b.core.flight`
module, 60

`videof2b.core.geometry`
module, 60

`videof2b.core.imaging`
module, 66

`videof2b.core.processor`
module, 66

`videof2b.core.projection`
module, 68

`videof2b.ui`
module, 78

`videof2b.ui.about_window`
module, 69

`videof2b.ui.camera_cal_dialog`
module, 69

`videof2b.ui.camera_director_dialog`
module, 70

`videof2b.ui.exception_dialog`
module, 71

`videof2b.ui.icons`
module, 72

`videof2b.ui.load_flight_dialog`
module, 72

`videof2b.ui.main_window`
module, 72

`videof2b.ui.style`
module, 75

`videof2b.ui.video_window`
module, 75

`videof2b.ui.widgets`
module, 76

`VideoProcessor` (class in *videof2b.core.processor*), 66

`VideoWindow` (class in *videof2b.ui.video_window*), 75

W

`WHITE` (*videof2b.core.drawing.Colors* attribute), 56

`Wingover` path, 48

Y

`YELLOW` (*videof2b.core.drawing.Colors* attribute), 56