EAN-Startup Guide Tracker Integration

2022-06-12

Exports: Export Summary Sheet

EULA: End User License Agreement

Web: sightlineapplications.com

Sales: sales@sightlineapplications.com

Support: support@sightlineapplications.com

Phone: +1 (541) 716-5137

CAUTION: Alerts to a potential hazard that may result in personal injury, or an unsafe practice that causes damage to the equipment if not avoided.

IMPORTANT: Identifies crucial information that is important to setup and configuration procedures.

Used to emphasize points or reminds the user of something. Supplementary information that aids in the use or understanding of the equipment or subject that is not critical to system use.

1 Overview

This document helps familiarize customers with SightLine protocols through an example project using SightLine software, hardware, and a camera interface to create a control loop system that moves a camera so that the tracked object remains in the center of the scene.

The Pelco D PTZ camera interface used in this example is not a requirement since the principles are applicable regardless of actual motor control protocol.

1.1 Recommended Knowledge Base

The document requires the following knowledge base before starting:

- C/C++ knowledge
- Experience with video cameras, including setting up and working with protocols.
- Knowledge of logging into remote servers and running executables
- Setting up embedded hardware in a benchtop environment.
- Using Panel Plus for configuring SightLine hardware
- SightLine protocol (the IDD)
- Sending and receiving UDP packets or serial communication

1.2 Additional Support Documentation

Reference links to Sightline documentation used in this guide are provided in the relevant sections.

Additional Engineering Application Notes (EANs) can be found on the Documentation page of the SightLine Applications website.

The Panel Plus User Guide provides a complete overview of settings and dialog windows located in the Help menu of the Panel Plus application.

The SightLine protocol IDD (Interface Design Document) describes the native communications protocol used by the SightLine Applications product line. The IDD is also available as a PDF download on the Documentation page under Software Support Documentation.

1.3 SightLine Software Requirements

IMPORTANT: This project requires the 3.5.0 beta ARM processor example code. VideoTrack version 3.3.x and higher is required for this application.

IMPORTANT: The Panel Plus software version should match the firmware version running on the board. Firmware and Panel Plus software versions are available on the Software Download page.

1.4 Third Party Software

Code editor such as Visual Studio.

A file transfer program such as WinSCP or similar.

Tera Term (or PuTTY): SightLine recommends Tera Term for troubleshooting, debugging, and issuing commands on SightLine hardware.

1.5 Application Bit Requirements

The functions described in this EAN require Application Bits (app bits) purchased from SightLine. App bits are enabled with a license file provided by SightLine at initial unit purchase or during a license upgrade process. License files use a hardware ID that is applicable to a specific hardware serial number. For questions and upgrade support contact Sales.

Table 1: Application Bits Requirement Table

Tracking Features	Initial Software Release	Required Application Bit(s) v7 License
Vehicle	2.22.xx	Full Tracking 0x0000 0010
Stationary		
Scene		
No Registration		
Static		
Acquisition Assist		
Intelligent Assist		
Drone	2.24.xx	
Person	3.01.xx	
High Bit Depth/Temp Telemetry	3.00.xx	High Bit Depth/Temp Telemetry 0x0002 0000

2 Hardware Connections

The bench setup for this example uses the 4000-OEM and requires a PC to run the Panel Plus application. Although the focus of this guide is on the 4000-OEM, the Arm Example Code also includes a workspace for use on the 3000-OEM. The same principles apply to the 3000-OME as for the 4000-OEM. See the ARM Application Development EAN for details on how to install ARM example code onto the 3000-OEM.

For additional options, Eval Kits, and interface boards, contact Sales. To review all the interface board options, see the 4000-OEM Accessories page on the SightLine Applications website.

ⓘ **IMPORTANT**: To prevent damage to the hardware boards, use a conductive wrist strap attached to a good earth ground. Before picking up an ESD sensitive electronic component, discharge built-up static by touching a grounded bare metal surface or approved anti-static mat.

Interface and adapter boards:

- SLA-3000-AB: Provides camera interface to the Pelco D camera.

Cable connections:

- SLA-CAB-SMA2BNC: Connects to J3 (SMA jack) on the 3000-AB and to the Pelco D camera.
- SLA-CAB-0403: Connects to J4 on the 4000-OEM. Provides an RJ45 Ethernet connection.
- SLA-CAB-ETH0: Connects to the Ethernet port and to the network switch or host PC.
- SLA-CAB-0804: Connects to J25 on the 4000-OEM. SLA-CAB-0304.
- SLA-CAB-0304: Connects to the RS232 to 3.3V TTL serial converter and SLA-CAB-0804.
- SLA-CAB-1504 / SLA-PWR-B12V-36W (110-250VAC input / 12VDC output): Connects to J50 on the 4000-OEM.

Additional hardware:

- 3.3V TTL to RS-232 serial converter
- RS232 to RS422/485 serial port converter
- Null modem adapter or DB-9 gender changer.
- Pelco D compliant PTZ camera (PN: BG-BPTZ-3XU by BZB Gear)

Power and network connectivity LEDs:

A green light (D1) on the 4000-OEM indicates that all boards are powered on. An amber light (D5) verifies network connection.

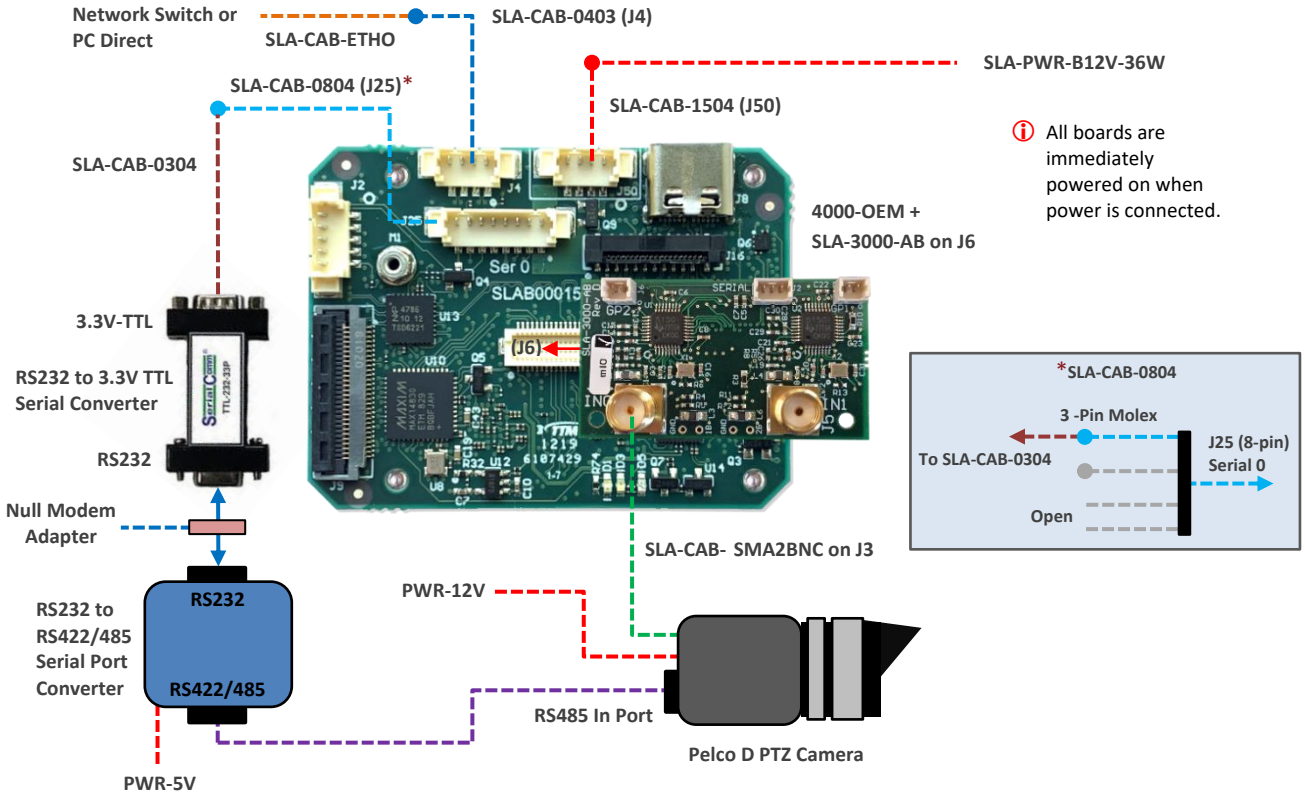

Figure 1: Hardware Bench Setup - 4000-OEM / SLA-3000-AB

3 Integration Design and Project Objectives

The following example implements an ARM application running on the 4000-OEM using the SightLine C++ API. The purpose of this application is to create a Proportional Integral Derivative Controller (PID) that moves a camera so that the tracked object remains in the center of the scene.

This is done by obtaining telemetry information from VideoTrack and then feeding the location information of the tracked object into the PID controller. The application then sends a pan and tilt command to the camera, closing the control loop.

Panel Plus runs on the host PC and communicates to the 4000-OEM through either a direct Ethernet connection, or by connecting to the local network.

Panel Plus is used to start a track in this example. Users can also develop their own UI with Panel Minus available as a download on the SightLine Command / Control webpage.

The OEM is also connected to the local network through a network switch. The analog adapter board is connected to the 4000-OEM. The analog cable connects the camera to the adapter board and transmits the video stream.

A serial connection is used to send commands to the camera. Since the serial output of the OEM board is in a 3.3V TTL format, adapters are needed to connect the camera serial port to the OEM.

Figure 3 shows how the software distributes the flow of data to the physical components.

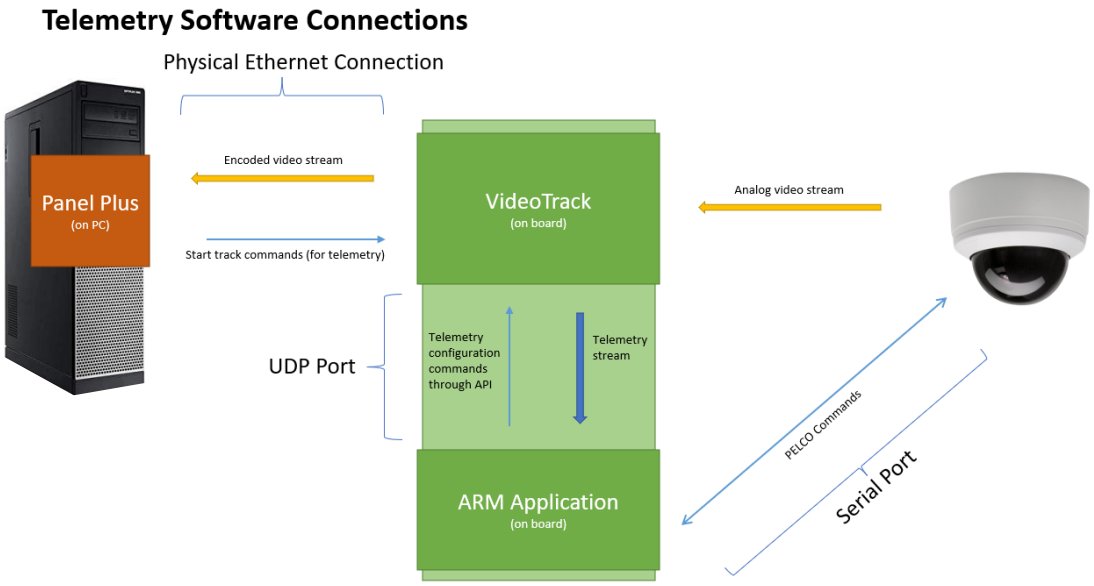

Figure 2: Software / Hardware Data Flow Diagram

The host PC (through Panel Plus) receives the encoded video stream through a physical Ethernet connection to VideoTrack. Through Panel Plus, tracks can be initiated that VideoTrack will process and optionally add graphical overlay indicators on the video stream.

The ARM application opens a serial port to send Pelco D commands to the camera, and a UDP port to send commands to VideoTrack. This UDP port is also responsible for receiving the packets from the VideoTrack telemetry stream. Note that UDP packets between the ARM app and VideoTrack are internal to the 4000 OEM.

3.1 Running the Tracking Loop

The main logic loop of the application performs the following sequence of actions:

1. Reads in a telemetry packet from the telemetry screen.
2. Feeds this value into the PID controller, which outputs a speed command.
3. Calculates from the speed sign whether to pan left or right or tilt up or down.
4. Plots the offset and output speed onto the overlay graphs. See Appendix B for more information.

5. Sends a single Pelco D command with the pan and tilt command and their respective speeds to the camera.
6. The camera movement causes a change in scene detected by VideoTrack, which updates the telemetry.

By minimizing the X and Y offsets from the screen center, the PID Controller will attempt to keep the target in the center of the scene. When a track is started, the camera will move to center it. As the target moves the camera position will update to keep it in the center of the frame.

ⓘ **IMPORTANT**: When there is no track selected or the track disappears, the default behavior uses scene tracking. This means any movement by the camera causes the camera to adjust so that it continues to point at the same scene.

4 SightLine Protocol Background

4.1 VideoTrack

VideoTrack is the main application running on the SightLine hardware that performs the bulk of the image processing including object tracking and adding overlays to the video stream. Interaction with VideoTrack is through the SightLine Command and Control Protocol (SVP).

SightLine commands are sent using formatted messages called SVP packets. SVP packets consist of a header, a length, a message ID, a payload, and a checksum. SVP packets can have either a 1-byte length field or a 2-byte length field (called extended packets).

SVP command payloads are represented as structures (structs) with specific formats, lengths, byte orders, and types. All current SVP commands, the formatting of their payloads, and much more can be found in the IDD. SightLine provides a C++ API as example code to help develop applications that use SVP commands to communicate with VideoTrack.

4.2 Panel Plus

Panel Plus is a graphical engineering tool developed by SightLine for communicating with VideoTrack. Panel Plus can be installed on a PC that is connected directly to the board through either Ethernet or a serial port.

Send and receive SVP commands to and from VideoTrack with Panel Plus to exercise all the features in the SightLine protocol.

For more information on how to use Panel Plus see the Pane Plus User Guide located in the Help menu of the Panel Plus application.

To evaluate the SVP commands formed by this project, SightLine recommends using Panel Plus to generate the same commands and compare them to the project commands. Raw SVP commands in Panel Plus are shown in the verbose area of the Panel Plus GUI when an SVP command is sent or received by Panel Plus as shown in Figure 3.

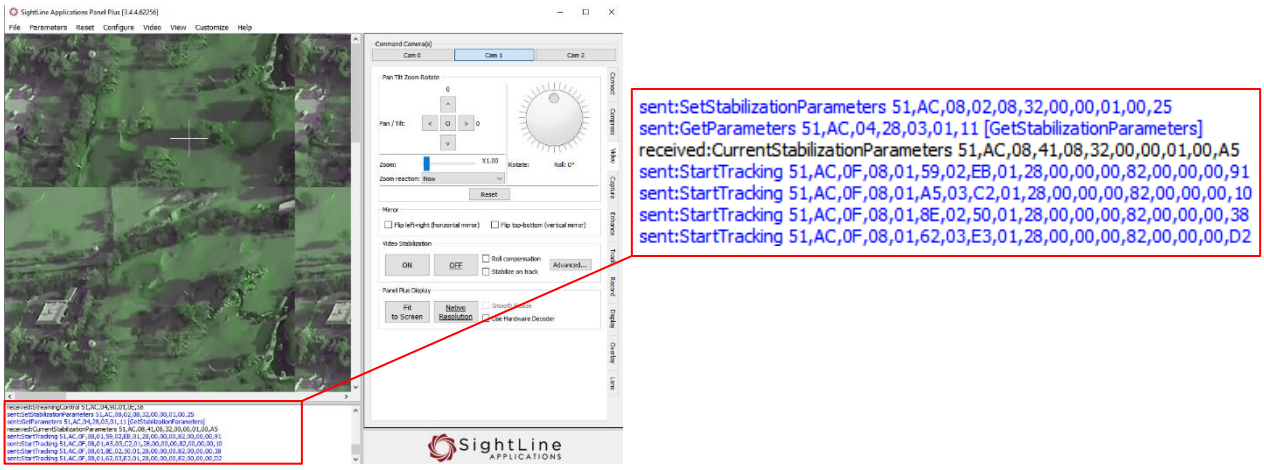

Figure 3: Verbose Area in Panel Plus

4.3 Telemetry

Telemetry includes reports from multiple video processing software functions. See the Telemetry page of the IDD for a complete list of telemetry information available.

Telemetry is received as a stream of data sent to a specified IP address. It is also available on the serial port, but this usage is not addressed in this guide.

4.3.1 Viewing Telemetry in Panel Plus

Telemetry can be viewed in Panel Plus by clicking the *Show Telemetry* button at the bottom of the *Track* tab or through the Panel Plus main menu » *Configure Telemetry*.

The *Telemetry* window gives options for viewing the telemetry data that SightLine hardware records. To start receiving telemetry in Panel Plus, check the *Send Telemetry 2Me* and *Registration and Primary Track* checkboxes.

Use the *Send Telemetry Every [n] Frames* to set the frame reporting period in Panel Plus.

Any combination of telemetry data can be recorded to a text file on the PC by clicking the *Start Recording* button and specifying a file location in the file save dialog. Clicking *Stop Recording* stops the recording.

When the options have been checked, the telemetry will display in the window as shown in Figure 4.

It can be beneficial to collect telemetry data about tracks. To see track data in Panel Plus, make sure a primary track is selected in the *Track* tab. Tracking data will then appear above the Registration information in the telemetry window.

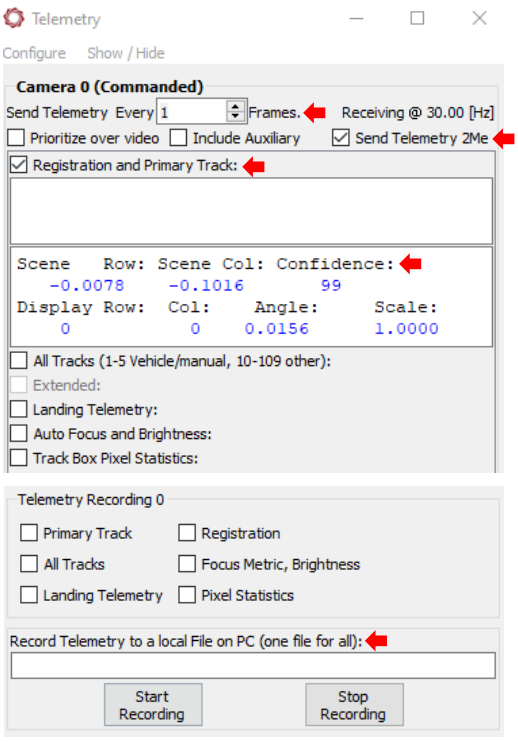

Figure 4: Show Telemetry

4.3.2 Telemetry Stream Setup Using the C++ API

In an application, start the telemetry stream by sending the following SVP commands to VideoTrack:

- SetTelemetryDestination (0x64)
- CoordinateReportingMode (0x0B)

The C++ API implements the following functions in *slfip.cpp* that make building and sending SVP commands to VideoTrack easier:

- *SLFIPSetPacketDestination*
- *SLFIPSetCoordinateReportingPeriod*

The *SLFIPSetPacketDestination* function tells VideoTrack which address and port to send the telemetry to. When developing an ARM application running on the board alongside VideoTrack, the address should be the localhost (127.0.0.1).

If development and testing is on a different PC, the address will be the IP of that device. The port number is the read port specified when VideoTrack is initialized. Replies from VideoTrack are automatically sent to this port. See Instantiating a Serial port and a UDP port for more information.

The *SLFIPSetCoordinateReportingPeriod* function specifies to VideoTrack the type and how often the specified telemetry information is shown.

Telemetry can be provided for every video frame. For example, if the video stream is running at 30 Hz, telemetry can be provided once every ~33ms, which corresponds to a frame count of 1. To receive every other frame the frame count would be 2.

4.3.3 Reading from the Telemetry Stream

To receive packets from the telemetry stream, read from the socket port set above. SightLine recommends using the *FIPReadPacket* function in *slfipport.cpp*. This looks for the SVP header and converts the packet sent by VideoTrack into a byte array.

Ensure this byte array is a telemetry packet by verifying the message ID, confirming the legal SVP syntax, and verifying the checksum. The easiest way to do this is by using the code provided in the examples. Once it is verified, unpack it using the appropriate unpacking function provided in *slfip.cpp*. This function takes as an argument the telemetry data structure (*SLTelemetryData* in *slfip.h*) and populates that struct with the data from the SVP telemetry packet. Telemetry data can then be used by accessing the struct members.

4.3.4 Understanding Scene Translation

The translation of the scene since the last telemetry report, measured in pixels, is shown in *Scene Row* and *Scene Col*.

The *Scene Col* indicates the number of pixels the scene moved right or left (negative indicates left, positive indicates right). The *Scene Row* indicates the number of pixels the scene moved up or down (negative indicates up, positive indicated down).

Track Row and *Track Col* represent the location of the primary track in camera source coordinates when a track has been selected.

4.4 Pelco D

Pelco D is a protocol for communicating with Pan Tile Zoom (PTZ) cameras. The protocol supports a number of commands, but individual manufacturers can choose which commands a given camera responds to. Pelco D commands are sent as 7-byte packets from a device to the camera through a port. The camera used in this example has an RS-485 serial port.

Pelco D protocol and additional information on how to build Pelco D commands can be found by referencing online resources.

Further explanation of how to build and debug Pelco D commands is beyond the scope of this guide. If there is further interest in Pelco D, SightLine recommends exploring the protocol prior to implementing the example code.

5 Setup

To set up the camera, follow the guidelines in the 4000-OEM Startup Guide. The connector descriptions and their location on the OEM can be found in the ICD-4000-OEM. This document also contains further details of each connector, including pin functions and descriptions. It is left to the user to verify their selected camera compatibility with SightLine products.

Since multiple cameras can be connected to SightLine hardware, and communications with VideoTrack often require the index of the camera, it is important to note the camera index in the setup. See the Files and Classes section below for more information.

5.1 Serial Ports Connections

Available serial ports are shown in the *Connector Descriptions* section of the ICD-4000-OEM. Depending on the adaptor board being used on the 4000-OEM, not all the serial ports may be available.

4000 serial ports are 3.3V TTL. Select an appropriate voltage level adapter to connect to the camera.

5.2 Verification and Initial System Exploration

Before writing any ARM applications, make sure the camera is correctly set up and able to receive Pelco D commands. This can be tested using the send command function in Panel Plus. CommandPassThrough (0x3D) will send the Pelco D command to VideoTrack, which in turn sends the command to the camera.

The example in Figure 5 shows the command to tilt the camera up when connected through serial port 0.

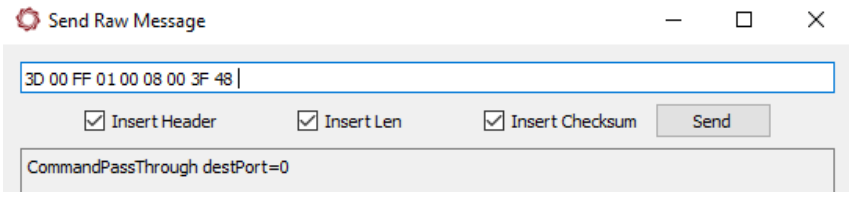

Figure 5: Tilt Camera Up Command Example

1. From the main menu » *File* » *Send Command*. This allows raw SVP commands to be sent directly to VideoTrack.
2. Check the boxes to automatically insert the SVP header bytes, checksum, and length in the *Send Raw Message* dialog window.
3. Enter the SVP Message ID byte. In this example it is CommandPassThrough (0x3D). This command routes its payload to a specified port.
4. Enter the serial port number that VideoTrack will send the command to (in this example it is serial port 0).
5. Enter the full Pelco D packet to send to the camera. Panel Plus is indifferent to the payload that is sent through this function. The Pelco D checksum must be calculated, and any required Pelco D headers must be manually inserted.

There are other ways to send a Pelco D command to a camera directly. This method utilizes the connections already set up for this example project.

6 Code Overview

6.1 PID Controllers

In this startup guide, constant refers to the constant number that a variable is multiplied by. A term refers to the portion of an equation where a constant is multiplied by a variable for the purpose of creating an effect on the result with an intuitive meaning. A value refers to the actual number of what is being discussed.

A PID controller is a feedback control system that calculates a correction to a control variable based on the offset between a center of the frame (the desired value) and the measured value. This difference is also referred to as the error. The controller calculates the output based on the current error (the proportional term), the summation of past errors (the integral term) and the change in error between the current and previous error over time (the derivative term).

Not all these terms may be needed so their value may be set to 0. In this example project the PID controller takes in the difference between the current location of the track, and the desired location (the center or half the screen pixel size) and tries to reduce that error to 0. It does this by calculating the speed at which the camera should move based on the three terms.

6.2 Files and Classes

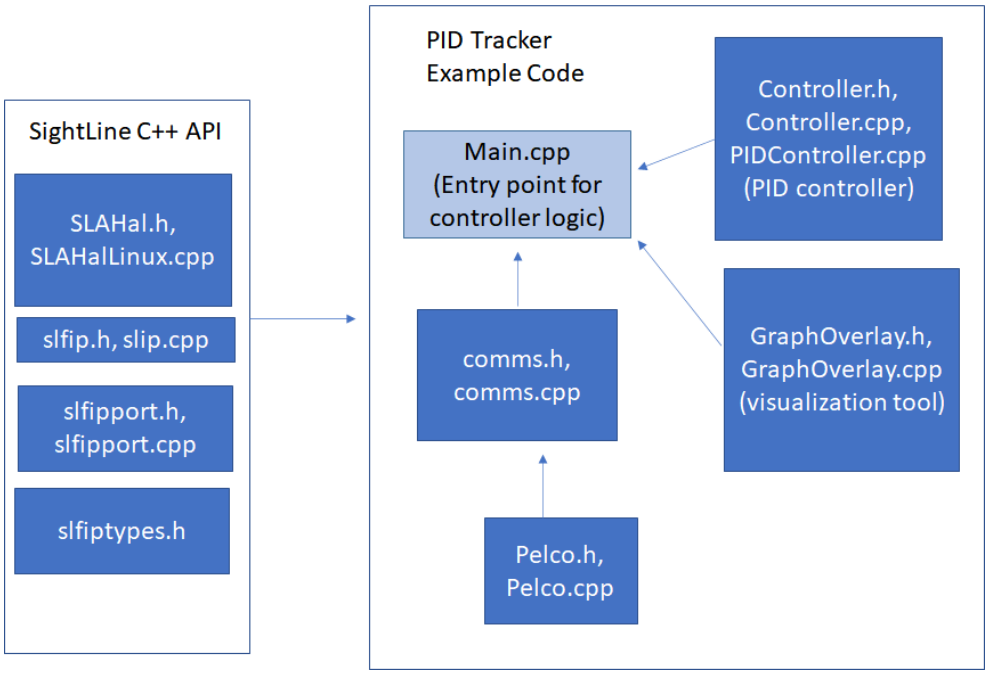

Figure 6: File Relationship

The main file (*Main.cpp*) contains the entry point for the program and performs the tracking. It also specifies near the top of the file constants that are system-dependent, such as the baud rate and the camera index.

In this example, the baud rate (defined as *BAUD_RATE* in the code) is 9600, and the camera index (defined as *CAMERA_INDEX*) is 2. It may be different depending on each setup.

The communications file (*comms.cpp*) contains the code that initializes ports and sends and receives information from those ports.

The remaining files include the PID Controller (*PIDController.cpp*) and a utility for overlaying real-time graphs on top of VideoTrack (*GraphOverlay.cpp*).

It is good practice to avoid putting the whole project into a single file. Multiple files make it easier to evaluate, increase portability, improve readability, and enforce the principle of abstraction.

6.3 Building the Project

During development, an SSH connection can be established directly into the 4000-OEM to build and test the application. The build files can be copied into the 4000-OEM using a remote file management application such as WinSCP (Windows only) or FileZilla. A command-line tool such as SCP on MacOS and Linux can also be used. An SSH session into the board can be established using the command line on Linux, Terminal on Mac, or Tera Term on Windows.

Run the bash file to build the project, and then run the executable. If needed change the permissions of the build script into an executable using *chmod*.

```
> chmod +x ./buildPelco.sh
```

To see the camera responding to the application, Panel Plus should be running on the PC while the program is running on the OEM. To stop the application, use the bash quit command (*Ctrl-c*).

It is better to use one text editor during development since line ending encodings are often managed differently by different applications. Use print statements sparingly and print from a calling routine rather than in member functions.

Printing to the shell is computationally expensive and has the potential to cause performance degradation if overused.

6.4 GDB Debugging Tool

GDB comes pre-installed on the 4000-OEM. It is a useful tool for debugging C/C++ applications in real-time. To use GDB, compile with the -g flag. Further information can be found online.

6.5 Instantiating Serial Port and UDP Port

1. Using the Sightline API, open a serial port to communicate with the camera by instantiating and initializing a *SLARs232* object found in *SLAHal.h*.
2. Make sure to use the correct the baud rate and serial port for the connected camera. In this example, it is 9600 and serial port 0.
3. Instantiate and initialize a *SLASockUDP* port to connect to VideoTrack. Use the following initialization parameters:
   - The address of the port that is being opened. Because the application is connecting to VideoTrack, which is on the OEM that is also running the program, the local loopback IP address (127.0.0.1) is used.
   - The port that is being written to. Some ports that are used by VideoTrack are used for specific cases.
   - A list of network ports and their purposes is detailed in the IDD. Because this internal ARM program will run alongside VideoTrack, use 14003 since it automatically sends its replies to the sending port and the sending IP.

     The 14001 receiving port sends replies to 14002 at the sending IP for Panel Plus.
   - Read port number. In this example, it is 16003. Telemetry data will be read from this port.
   - Broadcast flag- defaults to false.

6.6 Sending SVP Commands

Before the tracker can run, the height and width of the video frames will need to be determined. This information will be used to calculate the center of the frame and determine how large the overlays should be in pixels. The code for this logic is found in the project example code at *comms.cpp*.

6.6.1 Determining Video Frame Height and Width

1. Using the guidelines in the C++ API section, find the API function that builds an SVP command request. For height and width of the scene use *SLFIPGetImageSize*.
2. Write this command to the VideoTrack port. This tells VideoTrack to send an SVP command containing the current image size back to the VideoTrack port. Continue to read from the port until a packet arrives with the message ID for the image size (0x4E).
3. Since the image size message is sent in the SVP protocol format, its payload must be unpacked. This SVP command does not have a SightLine API unpacking function, so the struct associated with the payload - CurrentImageSize (0x4E) - must be implemented in the example program. To do this, use *memcpy* to copy the non-header portion of the SVP packet into an instance of the image size struct.
4. Start the telemetry stream to the VideoTrack port by sending the two SVP commands described in the Telemetry section. Once the SVP commands have been sent to the VideoTrack port, the VideoTrack port will be continuously receiving telemetry packets. Do not expect to receive any other packets from VideoTrack after this step.

6.7 Initializing PID Controller

The example code allows the PID constants to be set for both the x and y movement from the command-line. To do this, all six constants must be filled, or it will revert to a simple P controller. The PID controller automatically internally caps the speed at its highest value (in Pelco D, this is 0x3f).

Pelco D also supports a turbo mode where the speed is 0xff, but some Pelco D compliant cameras may not support it. Pelco D also does not guarantee a smooth speed transition between the value 0x3f and 0xff. For the simple PID controller, turbo mode would have complicated the controller. For more information see the Pelco D documentation.

There are different methods to tune a PID Controller, e.g., the Ziegler-Nichols method or the iterative tuning procedure described in the Wikipedia article on PID Controllers.

The following procedure is used for this example, which starts by determining an acceptable proportional term and then fine-tuning it with derivative term and integral term.

1. Start with the proportional term and set the other terms to 0. A good starting point for the P constant is the maximum speed that Pelco D supports (0x3f) divided by the total possible offset (the width or height of the frame divided by two). This evenly maps the fastest speed to the farthest location from the center of the frame, and the slowest possible speed to the location closest to the center of the frame.
2. Raise the P constant until the camera response starts oscillating markedly around the center of the frame. Note that this leads to the speed always being faster. The camera is now reaching its maximum speed closer to the center of the frame than before.
3. Raise the derivative term until it dampens the effect of the oscillation. At this point, the controller will output high speeds when the track is far away from the center but will slow down due to the derivative term closer to the center, reversing that wobble of the high proportional term.

The PID controller for this example does not need the integral term. Adding an integral term may result in a buildup of error that causes undesirable oscillation around the center of the frame depending on the camera.

6.8 Select Primary Track

For the telemetry packet to give useful data, there must be a primary track selected. This is done by clicking on an object in the video feed through Panel Plus, which will send tracking SVP commands to VideoTrack.

Since this application does not support multiple tracks, SightLine recommends selecting One track only and Primary in the Manual section of the Track tab. This will result in sending an SVP message with the x and y coordinates of the desired object in the frame, as well as the size of the initial track box.

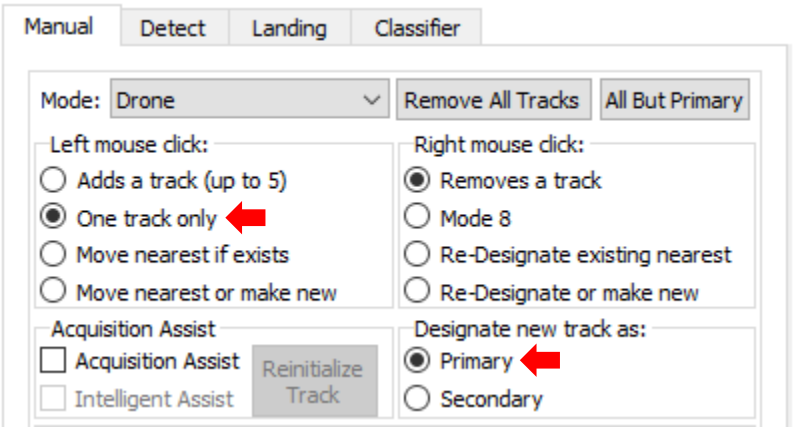

Figure 7: Primary Track

6.9 Expected Behavior and Scene Stabilization

With the track initiated and the PID controller tuned, the camera should move to keep the target in the center of the scene.

This program also uses scene tracking. When there is no track selected, the camera should keep the scene fixed in place.

7 Questions and Additional Support

For questions and additional support, please contact Support. Additional support documentation and Engineering Application Notes (EANs) can be found on the Documentation page of the SightLine Applications website.

Appendix A: Potential Issues

Zoom causing changes to the Field of View (FOV) :

If the zoom level changes while the controller is running, the tuning parameters may not have satisfactory performance at a different zoom level. This is because the change of FOV makes previously reasonable camera movements too fast for smooth movement if zoomed in, or too slow if the FOV widens.

One way to accommodate for this is to calculate the ratio between the new FOV and the previous one, and then adjust the parameter constants by the same factor.

For example, the FOV was 90° when the camera was fully zoomed out, and ~33° when zoomed in (camera is 3X zoom). The maximum speed of the camera was calculated to be 42 °/s. When zoomed out, a movement of 300 pixels corresponds to the camera rotating ~14°, so moving at the fastest possible speed of 42 °/s is reasonable.

However, when fully zoomed in, 300 pixels of scene movement would require the camera to move only 5°. If the parameters are not changed, the camera still moves at 42 °/s, which is an unreasonable speed given the number of degrees the camera needs to move. If the proportional constant for horizontal motion was 0.1 when the camera was fully zoomed out, 0.03 should be used when the camera is fully zoomed in. This results in the fastest speed being about 14 °/s when fully zoomed in.

Baud Rate: It is important to be aware of the baud rate of the camera when using this program since it may be necessary to rework the code if the baud rate is too low. If the amount of time it takes to send a packet to the camera is greater than the amount of time it takes VideoTrack to update telemetry data, use a separate thread to send commands to the camera. If this applies, be sure to include a call to *SLASleep* to send the correct number of packets per second.

The camera used in this example had a baud rate of 9600, and together with the byte length of the commands that needed to be sent the baud rate was not an issue. Otherwise, the writes to the serial port would not have succeeded.

Thread Safety: If multiple threads are needed, it is important to pay attention to thread safety since the PID controller output in VideoTrack will need to be shared with the camera command thread through a context struct.

If the camera command thread iterates faster than the thread reading from the telemetry stream, telemetry may be acting on old data.

Other physical considerations:

The performance of the track and the PID is heavily dependent upon the distance between the target and the camera and the speed of the target. The PID controller acts upon the scene in a two-dimensional pixel space, but the speed of an object in real space may appear slow or fast in relation to the two-dimensional scene. This depends on how far the object is from the camera.

The performance of the PID controller and the VideoTrack tracking capability will be better if an object is moving slowly in pixel space. See the *Target Tracking Best Practices* section in EAN-Target-Tracking for more information.

Appendix B: Overlay Tool

Graphic overlays are a convenient tool to see how the PID controller is performing in real-time. In the example code in the *GraphOverlay.cpp* and *.h* files, the top graph shows the x-direction pixel offset from the center of the frame over time, and the bottom graph shows the calculated speed along the pan axis (the output of the PID controller). They are created by drawing overlays. Overlays are a filled rectangle in the background, with short line segments as points displayed over the filled rectangle and a text field for displaying the numbers.

To use these overlays, first calculate where the graph should go in the scene in pixels. Take note of any other VideoTrack overlays present to set the graphical object IDs correctly.

Calculate how big the graph should be. Calculate what the maximum possible values are that may be expected so the graph can normalize these values in its coordinate space properly.

The graph assumes its inputs are signed. The middle of the graph is zero. As a convenience, a function is included to copy a graph. This new graph appears immediately below the original, and with the same dimensions. It calculates new object IDs based on the current graph overlay IDs, so it will override any object IDs that were created before the copy function was called.

To use the plot function of the graph, input an array with values that will be graphed. The function expects the array to be of the same length as was specified in the initialization of the graph. An option is given to specify which array index the graph should plot first. This makes it easier to show real-time updates. For this to work correctly it is important to keep track of the current array index and increment it to add new values and remove old ones as needed.

Appendix C: Windows Development

The ARM processor example code provides a Visual Studio solution file for development on Windows. If the Visual Studio solution is used for development, the application will run on a Windows machine and not directly on the OEM alongside VideoTrack.

The software and hardware relationships when developing on a PC are shown in Figure C1. This diagram is similar to the Software / Hardware Data Flow Diagram shown in Figure 2 in the Integration Design and Project Objectives section, but the application is no longer on the OEM.

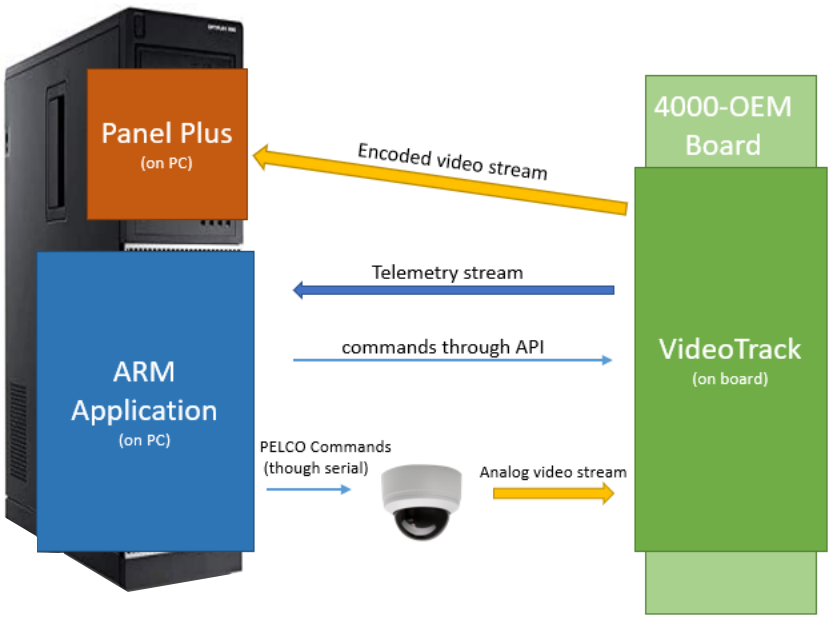

Figure C1: Software / Hardware Data Flow Diagram on Windows

C1 Camera Setup on Windows

The hardware difference between developing on the PC and developing on the board is that the serial connection to the camera is on the PC, rather than the board.

Since most computers have USB serial capabilities, use a USB serial adapter along with the appropriate seral converter to convert the serial interface from the PC to the serial interface on the camera. For example, if the USB serial adapter plugged into the computer outputs RS-232 serial signals, and the camera uses RS-485, use an RS-232 to RS-485 converter and a USB serial adapter.

PC to camera communication needs the physical port number that the camera is connected on to open that port. This port number will depend on the PC. To find this port Click WIN + R and enter *devmgmt.msc* to open the *Device Manager*. Go to *Ports COM & LPT* and expand the drop-down.

If there are USB serial comm ports available on the PC, they will be listed by COM and number, e.g., COM3.

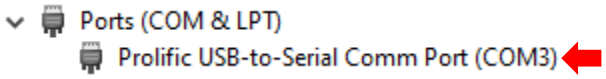

Figure C2: Windows Device Manager COM Port

All other connections described are the same as for development on the board. For example, the video output must still be connected to the board for VideoTrack to process.

C2 Port Initialization

The comm port described in the previous section must be changed in *Main.cpp* by setting the MICROCTRL_SERIALPORT definition as shown in Figure C3.

The IP addresses used for communication will be different when the application is running on the PC instead of the OEM board. These are defined in the example code in *Main.cpp* as shown in Figure C3.

```
 //Note: all settings that should be changed according to your system are marked with **
#ifdef _WIN32
 #define MICROCTRL_SERIALPORT 3  //replace with your PC COM PORT**
 #define IPADDR_VIDEOTRACK   "192.168.1.128" //replace with your videotrack ip**
 #define IPADDR_LISTEN       "0.0.0.0"
#else
 #define MICROCTRL_SERIALPORT  0 //SLA-Board Serial Port**
 #define IPADDR_VIDEOTRACK     "127.0.0.1"
 #define IPADDR_LISTEN         "0.0.0.0"
 #endif

#ifdef _WIN32
 const char *IPADDR = "192.168.3.219"; //If running from windows this is your PC IP address**
#else
 const  char *IPADDR = "127.0.0.1";
```

Figure C3: Communication Definitions

When the application is running on the OEM communication with VideoTrack happens locally. On Windows the application must initialize the port to VideoTrack with the IP address of the OEM board through the definition IPADDR_VIDEOTRACK. This is the same IP address used to connect to an OEM board on Panel Plus and as to establish an SSH session.

Change IPADDR to the PC IP address. This is the IP address that telemetry is sent to.

C3 Running the Visual Studio Solution

Open the *PID.sln* file in the Windows File Explorer. Make the necessary changes as described in the section above.

To run the application, click on *Local Windows Debugger*. The application can be run in *Release* mode or in *Debug* mode. Debug mode has additional options for setting breakpoint and inspecting variables but is significantly slower than *Release* mode.

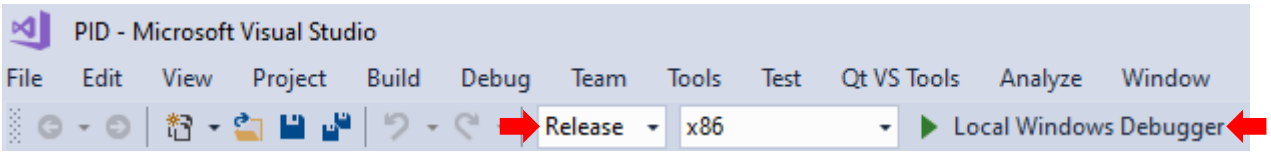

Figure C4: Running the Application