HoloLens 2 Sensor Streaming

Juan Carlos Dibene
Stevens Institute of Technology

Enrique Dunn
Stevens Institute of Technology

Abstract

We present a HoloLens 2 server application for streaming device data via TCP in real time. The server can stream data from the four grayscale cameras, depth sensor, IMU, front RGB camera, microphone, head tracking, eye tracking, and hand tracking. Each sent data frame has a timestamp and, optionally, the instantaneous pose of the device in 3D space. The server allows downloading device calibration data, such as camera intrinsics, and can be integrated into Unity projects as a plugin, with support for basic upstream capabilities. To achieve real time video streaming at full frame rate, we leverage the video encoding capabilities of the HoloLens 2. Finally, we present a Python library for receiving and decoding the data, which includes utilities that facilitate passing the data to other libraries. The source code, Python demos, and precompiled binaries are available at `https://github.com/jdibenes/hl2ss`.

1. Introduction

In this report, we present a real time system for streaming HoloLens 2 sensor data over TCP. The system consists of a server application that runs on the HoloLens 2 and a Python client library that receives and decodes the data. The client library works on Windows, Linux, and OS X systems. The server can also be integrated into Unity projects as a plugin and has support for receiving messages from the client. This allows leveraging the compute power of the client and the powerful capabilities of the Unity Engine on HoloLens 2.

The purpose of this system is to enable real time online experiments with HoloLens 2 data on other systems, with potentially more compute capabilities and third-party library support. The Windows Device Portal [19] can stream data from the HoloLens 2 front RGB camera and the microphone, but does not provide access to the four side-view grayscale cameras, the depth sensor, and the IMU. The Research Mode API [26] provides access to these sensors, but the given tools only allow recording the data to files and then downloading them from the HoloLens 2.

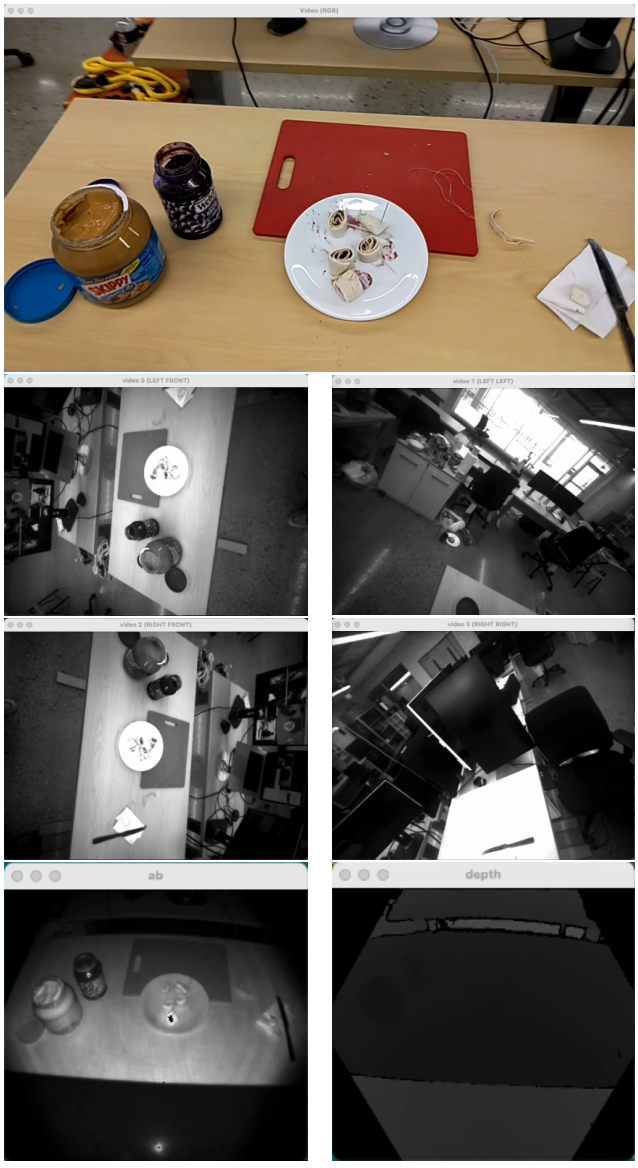

Figure 1. Our HoloLens 2 Sensor Streaming system allows to stream HoloLens 2 sensor data over Wi-Fi, enabling real time, online experiments on Windows, Linux, and OS X systems.

Our system fills this gap by providing access to all of these sensors, and to head pose, eye tracking, and hand tracking data over Wi-Fi. Figure 1 shows simultaneous capture from all of the HoloLens 2 cameras using our system. Data is transferred in real time at full framerate with low latency. For 1080p 30 FPS video from the RGB camera, we measured a latency of approximately 270 ms in our setup.

The rest of this report is organized as follows. Section 2 describes the details of our HoloLens 2 server application. Section 3 describes the features of the Unity plugin. Section 4 introduces our Python library for data reception.

2. HoloLens 2 Server Application

Our HoloLens 2 server is a C++ Universal Windows Platform (UWP) application that exposes the following data streams at specific TCP ports as shown in Table 1:

- RM Visible Light Cameras (VLC) - Four grayscale cameras operating at 640x480 @ 30 FPS. H264 or HEVC encoded.
- RM Depth Long-throw - 16-bit depth (in millimeters) and 16-bit active brightness (AB) image of 320x288 @ 1-5 FPS. Encoded as a single 32-bit PNG. RM Depth AHaT is not supported.
- RM IMU - Accelerometer (m/s^2), gyroscope (deg/s), and magnetometer.
- Photo/Video RGB camera (PV) - 1920x1080 @ 30 FPS with configurable resolution, framerate, exposure, focus, and white balance. H264 or HEVC encoded.
- Microphone - 2 channels and 48000 Hz sample rate. Encoded as AAC ADTS.
- Spatial input - Head pose, gaze ray (origin and direction), and hand tracking (2 hands with 26 joints each).

Data from the grayscale cameras, depth sensor, and IMU is acquired using the Research Mode (RM) API [26]. PV camera video is obtained using the MediaCapture class [10]. Microphone audio is captured using WASAPI [1]. Head pose and eye tracking data is acquired using the SpatialPointerPose class [18]. Hand tracking data is obtained using the SpatialInteractionManager class [16].

2.1. Data Compression

To reduce network bandwidth requirements and achieve real time data streaming, audio and video are compressed using the Microsoft Media Foundation SDK [11]. For video, the client can select one of four video encoding profiles and directly set the bitrate. For audio, the client can select one of four bitrate presets. All of the available encoding profiles are shown in Table 2 and all of them are lossy.

Stream	TCP Port
RM VLC Left-front	3800
RM VLC Left-left	3801
RM VLC Right-front	3802
RM VLC Right-right	3803
RM Depth Long-throw	3805
RM IMU Accelerometer	3806
RM IMU Gyroscope	3807
RM IMU Magnetometer	3808
PV	3810
Microphone	3811
Spatial Input	3812

Table 1. TCP Ports for each data stream exposed by the HoloLens 2 server application.

ID	Video encoding profiles	Audio encoding presets
0	H264 Baseline	AAC, 12000 bytes/s
1	H264 Main	AAC, 16000 bytes/s
2	H264 High	AAC, 20000 bytes/s
3	H265 Main (HEVC)	AAC, 24000 bytes/s

Table 2. Available audio and video encoding profiles. All of these encoding formats are lossy.

Stream	Bandwidth
Single RM VLC	1 Mbit/s (default)
RM Depth Long-throw	6.5 Mbit/s
RM IMU Accelerometer	250 Kbit/s
RM IMU Gyroscope	1.5 Mbit/s
RM IMU Magnetometer	12 Kbit/s
PV	5 Mbit/s (default)
Microphone	192 Kbit/s (default)
Spatial Input	1 Mbit/s
Total	19 Mbit/s

Table 3. Average bandwidth for each stream. The bitrate of the audio and video streams is configurable.

For depth, the invalid pixels in the depth image, as indicated by the 8-bit 320x288 sigma channel, are set to zero. Then, the depth and AB images are interleaved using NEON [7] and encoded as PNG (lossless) using the BitmapEncoder class [3]. This is 1) to avoid sending the sigma channel, and 2) interleaving the images gives better compression results than concatenating them, although this is scene-dependent. The average bandwidth for each stream is shown in Table 3.

Field	Type	Length in bytes
Timestamp	u64	8
Size of payload	u32	4
Payload	bytes	Size of payload
Pose (optional)	4x4 float	64

Table 4. Common data frame structure. The pose field is optional, and if not included, its length is zero. Data frames are unpacked from the stream by a simple FSM.

Mode	Description
0	Continuous transfer of device data
1	Continuous transfer of device data and pose
2	Single transfer of calibration data

Table 5. Stream operating modes. The client configures the operating mode when opening the stream.

2.2. Data Transfer

Data is transferred over TCP using Windows Sockets [24]. All data is transferred in little-endian format and matrices in row-major order. The structure of a data frame, shown in Table 4, is the same for all streams. The timestamps are expressed in hundreds of nanoseconds and are in the same domain (QPC [2]), so they can be used to correlate the data of different streams. The device pose field is optional and its presence is controlled by the client. The pose is expressed as a 4x4 float matrix. If the pose is valid, the last element of the matrix is 1. Otherwise, if the HoloLens 2 tracking is lost for that frame, the last element is 0.

Each stream runs on its own thread, so multiple streams can be active simultaneously. However, there is no provision for the stream-client synchronization required for data frame unpacking other than the start of the stream. Therefore, only one client per stream is allowed. The streams support up to three different operating modes, as shown in Table 5. The client sets the stream operating mode when opening the stream and persists until the stream is closed. In Mode 0, only the device data is transferred. In Mode 1, the corresponding device pose for each frame is included. Mode 2 is used by the client to download calibration data, which is device dependent, and the stream is automatically closed at the end of the transfer. The operating modes supported by each stream are shown in Table 6.

2.3. Mode 0 and Mode 1 Transfers

For all but the Spatial Input stream, the server waits for the client to send configuration data and does not begin streaming until it is received. Configuration data is sent over the stream's TCP port in little-endian format. The configuration parameters and their order for each stream are:

Stream	Supported Modes
RM VLC	0, 1, 2
RM Depth	0, 1, 2
RM IMU Accelerometer	0, 1, 2
RM IMU Gyroscope	0, 1, 2
RM IMU Magnetometer	0, 1
PV	0, 1, 2
Microphone	0
Spatial Input	0

Table 6. Stream operating modes supported by each stream. Not all streams support all operating modes.

- RM VLC: operating mode (u8), width (u16), height (u16), framerate (u8), video encoding profile (u8), bitrate (u32). Width, height, and framerate must be 640, 480, and 30, respectively.
- RM Depth: operating mode (u8).
- RM IMU: operating mode (u8).
- PV: operating mode (u8), width (u16), height (u16), framerate (u8), video encoding profile (u8), bitrate (u32). Width, height, and framerate must be one of the configurations belonging to the VideoConferencing profile shown in the locatable camera overview [9].
- Microphone: audio encoding preset (u8).

Data streaming begins immediately after the server finalizes processing the configuration data. For the RM VLC and PV streams, the payload consists of encoded video frames. For the RM Depth stream, the payload is the PNG containing both the depth (in the first 2 channels) and AB (in the last 2 channels) images. For the RM IMU streams, the payload contains batches of samples. The structure of each sample is shown in Table 7. RM IMU Accelerometer, Gyroscope, and Magnetometer frames contain 93, 315, and 11 samples, respectively. For the Microphone stream, the payload consists of encoded audio samples.

The payload structure for the Spatial Input stream is shown in Table 8. The set bits of the valid field indicate whether the data of the Head Pose (bit 0), Eye Ray (bit 1), Left hand (bit 2), and Right Hand (bit 3) fields are valid. The structure of the Head Pose and Eye Ray fields are shown in Table 9 and Table 10, respectively. For the Head Pose, up corresponds to $+Y$ and forward corresponds to $-Z$. The Left Hand and Right Hand fields contain the pose of the 26 joints of each hand. See the HandJointKind Enum documentation [6] for a list of hand joints. The structure of each hand joint pose is shown in Table 11. Orientation is expressed as a quaternion. For more details, see the documentation for the JointPose struct [8].

Field	Type	Length in bytes
Sensor timestamp in ns	u64	8
Data frame timestamp	u64	8
X-axis measurement	float	4
Y-axis measurement	float	4
Z-axis measurement	float	4

Table 7. RM IMU sample structure. The data frame timestamp is the same for all of the samples in the batch.

Field	Type	Length in bytes
Valid	u8	1
Head Pose	bytes	36
Eye Ray	bytes	24
Left Hand	bytes	26x36
Right Hand	bytes	26x36

Table 8. Spatial Input payload structure. The first 4 bits of the valid field indicate the validity of the Head Pose, Eye Ray, Left Hand, and Right Hand fields.

Field	Type	Length in bytes
Position	1x3 float	12
Forward direction	1x3 float	12
Up direction	1x3 float	12

Table 9. Head Pose structure. Vector element order is X, Y, and Z. Right direction can be obtained as cross(up, -forward).

Field	Type	Length in bytes
Position	1x3 float	12
Direction	1x3 float	12

Table 10. Eye Ray structure. Vector element order is X, Y, and Z. Ray length is not provided.

For Mode 1 streams, the pose of the device is included in each data frame. For RM streams, the pose corresponds to a device-defined coordinate frame defined as the *rigNode*. See the HoloLens 2 Research Mode report [26] for details. The extrinsics that express the transform of each sensor to the *rigNode* can be obtained from a Mode 2 transfer. The poses are acquired using the SpatialLocator class [17]. For the PV stream, the pose corresponds to the camera and comes bundled with the acquired video frames.

2.4. Mode 2 Transfers

The calibration data for each sensor can be obtained from a single Mode 2 transfer. Like Mode 0 and Mode 1 transfers, the client must send the configuration data first. For RM streams, only the operating mode byte must be sent. For the PV stream, the whole configuration string, as defined in subsection 2.3, must be sent. The server automatically closes the connection after all the calibration data has been transferred.

Field	Type	Length in bytes
Orientation	1x4 float	16
Position	1x3 float	12
Radius	float	4
Accuracy	u32	4

Table 11. Hand Joint structure. Quaternion element order is X, Y, Z, and W. Vector element order is X, Y, and Z.

Field	Type	Length in bytes
LUT uv2xy	480x640x2 float	2457600
Extrinsics	4x4 float	64

Table 12. RM VLC calibration data.

Field	Type	Length in bytes
LUT uv2xy	288x320x2 float	737280
Extrinsics	4x4 float	64
Scale	float	4

Table 13. RM Depth Long-throw calibration data.

Field	Type	Length in bytes
Extrinsics	4x4 float	64

Table 14. RM IMU calibration data. Not available for RM IMU Magnetometer.

The structure of the calibration data for the RM VLC, RM Depth Long-throw, and RM IMU streams is shown in Table 12, Table 13, and Table 14, respectively. The uv2xy LUT converts image coordinates to normalized coordinates on the camera unit plane. First channel corresponds to x, second channel to y. The extrinsics matrix for the sensor is the transform to the *rigNode*. The scale value is used to convert depth units to meters. This data is obtained using the Research Mode API [26], and more details can be found in its documentation. The structure of the calibration data for the PV stream is shown in Table 15. This data is embedded in every video frame. However, since the server only exposes this data through Mode 2 transfers, the autofocus function of the PV camera must be disabled so the downloaded calibration data remains valid.

Field	Type	Length in bytes
Focal length	1x2 float	8
Principal point	1x2 float	8
Radial distortion	1x3 float	12
Tangential distortion	1x2 float	8
Projection	4x4 float	64

Table 15. PV calibration data.

2.5. Remote Configuration Port

The server exposes an interface at TCP port 3809 that the client can use to send secondary configuration data and query information. All data is in little-endian format. The available commands and their parameters are as follows:

- Set PV display marker state: 0 (u8), enable (u8). Controls the display marker used to show to the user the lower boundary of the field of view of the PV camera. This marker is used to help the user keep objects of interest within PV camera video frames.
- Set PV camera focus: 1 (u8), focus mode (u32), autofocus range (u32), manual focus distance (u32), focus value (u32), driver fallback (u32). See the FocusControl class [5] for details.
- Set PV camera video temporal denoising: 2 (u8), mode (u32). See the VideoTemporalDenoisingControl class [20] for details.
- Set PV camera white balance preset: 3 (u8), preset (u32). See the WhiteBalanceControl class [23].
- Set PV camera white balance value: 4 (u8), value (u32). See the WhiteBalanceControl class [23].
- Set PV camera exposure: 5 (u8), mode (u32), value (32). See the ExposureControl class [4] for details.
- Get server version: 6 (u8). Returns the server version as 1x4 vector of u16.

The PV camera can be configured even if the PV stream is transferring data to the client. For the white balance value and exposure commands, the values must be divided by 25 and 10, respectively. This makes the step size equal to 1. The server undoes this transformation so the values are in the expected range. Like the data streams, only one connection at a time is allowed.

3. Unity Integration

Our HoloLens 2 server can be integrated into Unity projects as a plugin. All the streams are supported. However, Spatial Input support requires that the plugin is initialized from the UI thread. The PV display marker functionality is not supported. Unlike the standalone server, the plugin has basic upstream capabilities.

Field	Type	Length in bytes
Command ID	u32	4
Size of parameters	u32	4
Parameters	bytes	Size of parameters

Table 16. Unity plugin IPC message structure. Command ID $FFFFFFFF is reserved and should not be used by the client.

3.1. Unity IPC Port

The plugin exposes an additional interface at TCP port 3816 that allows the client to send messages to a Unity application. The message structure is shown in Table 16. The interface is message-agnostic, so the meanings of the commands and their parameters are defined by the user. This allows the user to add support for new commands to their Unity application without modifying the plugin. However, command $FFFFFFFF is reserved and should not be used for new commands, as it is used to notify the Unity application that the client has disconnected from the IPC port. The plugin also supports sending responses to the client in the form of 4-byte integers. All data is in little-endian format. Only one connection at a time is allowed.

3.2. Remote Unity Scene

As an example, we created a Unity project with the Mixed Reality Toolkit (MRTK) [22] and implemented a set of commands to allow the client to create basic Unity objects, which the user can see in augmented reality (AR). The commands and their parameters are:

- Create primitive (0): type (u32). Creates a Unity primitive in the Unity scene. See Table 17. Returns key (u32), which can be used to modify the properties of the primitive.
- Set active (1): key (u32), state (u32). Activates or deactivates the object associated with key.
- Set world transform (2): key (u32), position (1x3 float), rotation (1x4 float), scale (1x3 float). Sets the world transform of the object associated with key. Rotation is a quaternion.
- Set color (4): key (u32), rgba (1x4 float). Sets the color of the primitive associated with key. The elements of rgba are in $[0, 1]$, and semi-transparency is supported.
- Set texture (5): key (u32), texture (JPG or PNG file). Sets the texture of the primitive associated with key.

Type	Primitive
0	Sphere
1	Capsule
2	Cylinder
3	Cube
4	Plane
5	Quad

Table 17. Available Unity primitives.

- Create text (6). Creates a TextMeshPro object. Returns key (u32), which can be used to modify the properties of the object.
- Set text (7): key (u32), font size (float), rgba (1x4 float), string (utf-8). Sets the text, font size, and color of the TextMeshPro object associated with key.
- Remove (16): key (u32). Destroys the object associated with key.
- Remove all (17). Destroys all objects created using the plugin.
- Begin display list (18). Hint for the Unity application to process the next commands in the same pass.
- End display list (19).
- Set target mode (20): mode (u32). Changes the behavior of commands that modify the properties of objects. If mode is 0, property changes apply to the object associated with key. If mode is 1, the key parameter is ignored and property changes apply to the last object created. This is to allow the client to create objects and set their properties immediately without having to wait for the server to return the key.

All of the commands return a 4-byte value. For the commands that do not return a key, the value 0 denotes failure, and 1 denotes success.

4. Python Library

We offer a Python library that facilitates connecting to HoloLens 2 server, sending configuration data and commands, receiving data and decoding it, and preprocessing the data for use with other libraries. For H264 and AAC decoding we use the PyAV [13] library, and PNG decoding is performed using OpenCV [12].

The library abstracts most of the communication details and presents a simple interface to the user. Here is an example of acquiring 1080p 30 FPS video from the PV camera and displaying it in a window in real time.

```
import cv2
import hl2ss
import hl2ss_utilities

# HoloLens 2 IP address
host = '192.168.1.15'

# Stream settings
port = hl2ss.StreamPort.PERSONAL_VIDEO
chunk = hl2ss.ChunkSize.PERSONAL_VIDEO
mode = hl2ss.StreamMode.MODE_1
width = 1920
height = 1080
framerate = 30

# Video encoding settings
profile = hl2ss.VideoProfile.H265_MAIN
bitrate = 5*1024*1024

# Video decoding settings
output_format = 'bgr24'

client = hl2ss_utilities.rx_decoded_pv(
            host,
            port,
            chunk,
            mode,
            width,
            height,
            framerate,
            profile,
            bitrate,
            output_format)

client.open()

while (enable):
    # Get next video frame, blocks
    # until the next frame is received
    data = client.get_next_packet()

    # data fields are:
    # data.timestamp
    # data.payload
    # data.pose (None for Mode 0)

    # Display frame using OpenCV
    # Decoded payload is a 1080x1920x3
    # NumPy array of u8
    cv2.imshow('Video', data.payload)
    cv2.waitKey(1)

client.close()
```

The procedure to acquire data from other sensors is very similar: 1) create a hl2ss rx or hl2ss_utilities rx_decoded (for the encoded streams) object with the desired configuration, 2) call the open() method to connect to the HoloLens 2 server, 3) continuously call the get_next_packet() method to receive data frames, and 4) call the close() method to close the connection. The get_next_packet() method must be called as soon and as often as possible to avoid dropping frames. Note that when using a hl2ss rx object with an encoded stream, the payload contains encoded frames.

One important limitation is that the get_next_packet() method blocks until the next frame is received, which complicates working with multiple streams. To overcome this, we provide a library extension based on Python's multiprocessing library.

4.1. Multiprocessing Extension

We extend our library using Python's multiprocessing capabilities to facilitate working with multiple streams. This extension abstracts HoloLens 2 data acquisition and data transfer between processes into four components:

- Source (process): continuously receives data from a single HoloLens 2 stream by calling get_next_packet() and sends every data frame to the interconnect.
- Interconnect (process): receives data frames from a single source and stores them into a ring buffer. Multiple sinks can request data from the interconnect.
- Sink (interface): Retrieves data frames from the interconnect. A process can manage multiple sinks, or each sink can be contained in its own process, or a combination of both.
- Control (process): Responsible for creating, attaching, and terminating sources, interconnects, and sinks. It is usually the main process.

The interconnect processes messages from the source, control, and sinks in a round-robin fashion. All communications between the components are signaled by semaphores, which obviates polling mechanisms and avoids wasting CPU time busy-waiting.

The interconnect exposes the following interface to the sink objects:

- get_nearest(timestamp): returns the data frame that is closest in time to the input timestamp.
- get_frame_stamp(): returns the frame stamp of the most recent data frame.
- get_most_recent_frame(): returns the most recent data frame.
- get_buffered_frame(frame_stamp): returns the data frame corresponding to the input frame stamp.

The frame stamp corresponds to the global position of the data frame in the stream (0 corresponds to the first frame at the beginning of the stream). Since the ring buffer has limited capacity, older frames will become unavailable.

The extension abstracts most of the multiprocessing details and presents the user with a simple interface. Here is an example that acquires data from the PV and RM Depth Long-throw streams.

```
import multiprocessing as mp
import hl2ss
import hl2ss_mp
import hl2ss_utilities

# HoloLens 2 IP address
host = '192.168.1.15'

......

# A producer is a source-interconnect
# pair and the hl2ss_utilities producer
# manages the collection of producers
# for the HoloLens 2 streams
producer = hl2ss_utilities.producer()

# Initialize PV
producer.initialize_decoded_pv(
  30 * 5, # Space for 5 seconds of data
  host,
  hl2ss.StreamPort.PERSONAL_VIDEO,
  hl2ss.ChunkSize.PERSONAL_VIDEO,
  hl2ss.StreamMode.MODE_0,
  640,
  360,
  30,
  hl2ss.VideoProfile.H265_MAIN,
  1*1024*1024,
  'rgb24')

# Initialize RM Depth Long-throw
producer.initialize_decoded_rm_depth(
  5 * 5, # Space for 5 seconds of data
  host,
  hl2ss.StreamPort.RM_DEPTH_LONGTHROW,
  hl2ss.ChunkSize.RM_DEPTH_LONGTHROW,
  hl2ss.StreamMode.MODE_1)

# The sources start acquiring data and
# the interconnects add it to their
# ring buffers
producer.start()
```

```
# We use this manager to create a
# semaphore that signals when new
# data has been received
manager = mp.Manager()

# This consumer object manages a
# collection of sinks
consumer = hl2ss_utilities.consumer()

# Create a sink to read from the PV
# producer, but don't create a
# semaphore
sink_pv = consumer.create_sink(
  producer,
  hl2ss.StreamPort.PERSONAL_VIDEO,
  manager,
  None) # No semaphore

# Create a sink to read from the depth
# producer and create a semaphore
# because we want to know when a new
# depth frame arrives
sink_depth = consumer.create_sink(
  producer,
  hl2ss.StreamPort.RM_DEPTH_LONGTHROW,
  manager,
  ...) # Create new semaphore

# Semaphores can be shared by sinks of
# the same consumer, but then the user
# is responsible for checking which of
# the producers received new data

# Get the frame stamp of when the sink
# was attached to the producer. Used to
# synchronize producer and sink 1-to-1
# (not shown in this example). Must be
# called before any other sink method
sink_pv.get_attach_response()
sink_depth.get_attach_response()

while (enable):
  # Wait for new data
  sink_depth.acquire()

  # Get depth data and check that the
  # pose is valid
  data_depth =
    sink_depth.get_most_recent_frame()
  if (not data_depth.is_valid_pose()):
    continue

  # Get the closest RGB frame
  _, data_pv =
    sink_pv.get_nearest(
      data_depth.timestamp)
  if (data_pv is None):
    continue

  # At this point we have an RGB-depth
  # image pair with pose we can use for
  # 3D reconstruction

......

# Detach sinks from producer
sink_pv.detach()
sink_depth.detach()

# Terminate producer
producer.stop()
```

We use a variation of this example to perform real time TSDF integration on a client machine using Open3D [27].

4.2. Interoperability

All decoded data is returned as NumPy arrays that can be used as is with other libraries:

- RM VLC: 480x640 of u8.
- RM Depth: 288x320 of u16 for both depth and AB.
- PV: height x width x 3 of u8.
- Microphone: 2x1024 of float.

Additionally, our library provides utility functions for data preprocessing and storage:

- Image undistort for the RM VLC and RM Depth streams (PV images have no distortion).
- RGBD image alignment: RM VLC + Depth or PV + Depth (AB + Depth are always aligned). See Figure 2.
- Associate data from different streams.
- Transforms between reference frames.
- Write/Read timestamps, data, and poses to file and save encoded audio / video to MP4.

Undistorting and aligning is performed using OpenCV [12] and writing to MP4 uses PyAV [13]. Since decoding, undistorting, and aligning are performed in real time, our framework can be integrated into real time, online systems.

As examples, Figure 3 shows a point cloud created on a client machine using Open3D [27] from HoloLens 2 rgbd-aligned data, and Figure 4 shows panoptic segmentation results on an undistorted RM VLC image using MMDetection [25] which is then used to segment RM Depth data.

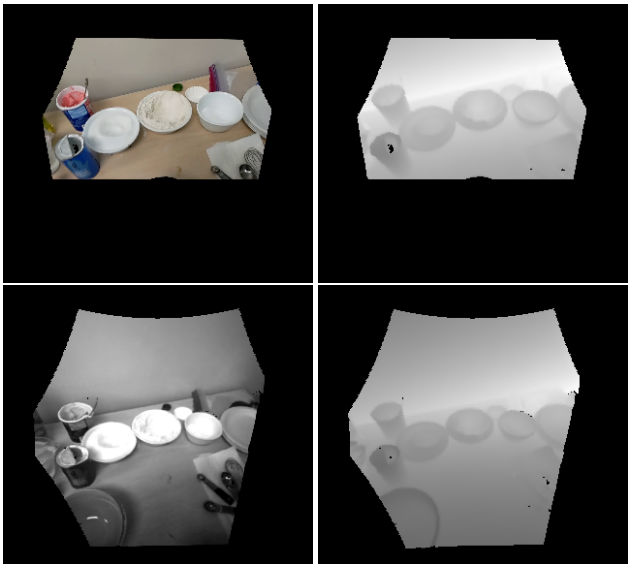

Figure 2. RGBD image alignment. Color can be from either the RM VLC (top) or the PV (bottom) stream.

Figure 3. Point cloud generated from HoloLens 2 data using Open3D [27].

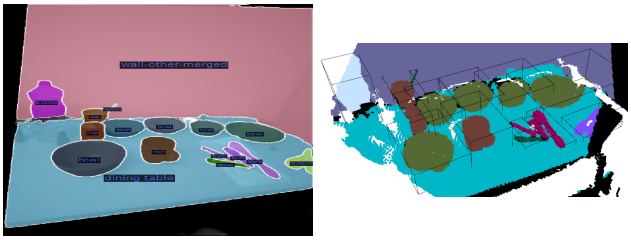

Figure 4. Panoptic segmentation on RM VLC data using MMDetection [25] and used to 3D segment RM Depth data.

5. Conclusions

We have presented a system to stream HoloLens 2 data over Wi-Fi at full framerate in real time, that enables real time online experiments on Windows, Linux, and OS X systems. In future work, we aim to include streams for Spatial Mapping data [15], Scene Understanding [14], and to give the client access to Voice Input [21].

6. Acknowledgments

This work was sponsored by the Defense Advanced Research Projects Agency, the content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

[1] About WASAPI. https://learn.microsoft.com/en-us/windows/win32/coreaudio/wasapi. Accessed: 2010-09-30. 2

[2] Acquiring high-resolution time stamps. https://learn.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps. Accessed: 2010-09-30. 3

[3] BitmapEncoder Class. https://learn.microsoft.com/en-us/dotnet/api/system.windows.media.imaging.bitmapencoder?view=windowsdesktop-6.0. Accessed: 2010-09-30. 2

[4] ExposureControl Class. https://learn.microsoft.com/en-us/uwp/api/windows.media.devices.exposurecontrol?view=winrt-22621. Accessed: 2010-09-30. 5

[5] FocusControl Class. https://learn.microsoft.com/en-us/uwp/api/windows.media.devices.focuscontrol?view=winrt-22621. Accessed: 2010-09-30. 5

[6] HandJointKind Enum. https://learn.microsoft.com/en-us/uwp/api/windows.perception.people.handjointkind?view=winrt-22621. Accessed: 2010-09-30. 3

[7] Intrinsics. https://developer.arm.com/architectures/instruction-sets/intrinsics/. Accessed: 2010-09-30. 2

[8] JointPose Struct. https://learn.microsoft.com/en-us/uwp/api/windows.perception.people.jointpose?view=winrt-22621. Accessed: 2010-09-30. 3

[9] Locatable camera overview. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/locatable-camera-overview. Accessed: 2010-09-30. 3

[10] MediaCapture Class. https://learn.microsoft.com/en-us/uwp/api/windows.media.capture.mediacapture?view=winrt-22621. Accessed: 2010-09-30. 2

[11] Microsoft Media Foundation. https://learn.microsoft.com/en-us/windows/win32/medfound/microsoft-media-foundation-sdk. Accessed: 2010-09-30. 2

[12] opencv-python. https://github.com/opencv/opencv-python. Accessed: 2010-09-30. 6, 8

[13] PyAV. https://github.com/PyAV-Org/PyAV. Accessed: 2010-09-30. 6, 8

[14] Scene understanding. https://learn.microsoft.com/en-us/windows/mixed-reality/design/scene-understanding. Accessed: 2010-09-30. 9

[15] Spatial mapping. https://learn.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping. Accessed: 2010-09-30. 9

[16] SpatialInteractionManager Class. https://learn.microsoft.com/en-us/uwp/api/windows.ui.input.spatial.spatialinteractionmanager?view=winrt-22621. Accessed: 2010-09-30. 2

[17] SpatialLocator Class. https://learn.microsoft.com/en-us/uwp/api/windows.perception.spatial.spatiallocator?view=winrt-22621. Accessed: 2010-09-30. 4

[18] SpatialPointerPose Class. https://learn.microsoft.com/en-us/uwp/api/windows.ui.input.spatial.spatialpointerpose?view=winrt-22621. Accessed: 2010-09-30. 2

[19] Using the Windows Device Portal. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/using-the-windows-device-portal. Accessed: 2010-09-30. 1

[20] VideoTemporalDenoisingControl Class. https://learn.microsoft.com/en-us/uwp/api/windows.media.devices.videotemporaldenoisingcontrol?view=winrt-22621. Accessed: 2010-09-30. 5

[21] Voice input. https://learn.microsoft.com/en-us/windows/mixed-reality/design/voice-input. Accessed: 2010-09-30. 9

[22] What is Mixed Reality Toolkit 2? https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05. Accessed: 2010-09-30. 5

[23] WhiteBalanceControl Class. https://learn.microsoft.com/en-us/uwp/api/windows.media.devices.whitebalancecontrol?view=winrt-22621. Accessed: 2010-09-30. 5

[24] Windows Sockets 2. https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2. Accessed: 2010-09-30. 3

[25] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 8, 9

[26] Dorin Ungureanu, Federica Bogo, Silvano Galliani, Pooja Sama, Xin Duan, Casey Meekhof, Jan Stühmer, Thomas J. Cashman, Bugra Tekin, Johannes L. Schönberger, Bugra Tekin, Pawel Olszta, and Marc Pollefeys. HoloLens 2 Research Mode as a Tool for Computer Vision Research. *arXiv:2008.11239*, 2020. 1, 2, 4

[27] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 8, 9