# Designing Services and Clients for Ambient Lab Smart Space

Cristina McLaughlin
*Department of Electrical Engineering*
*University of Hawaii at Manoa*
Honolulu, United States
cemclaug@hawaii.edu

***Abstract*— Smart spaces are physical environments enhanced by computation and networking technology to create a complex ecosystem that users can interact with. Smart spaces are a blend of the Internet of Things (IoT), Cyber-physical Systems (CPS), information processing, high-speed networking, and audio and visual sensors, that work in unison to provide context aware services. Ambient Edge is a distributed platform for high performance Edge networks and is being using as the backbone infrastructure for developing the Ambient Lab Smart Space. A service is software that performs a task while also managing the underlying hardware of a sensor or device; a client connects to and uses data from one or more services. This paper summarizes the prototyping of two Smart Space services for Ambient Edge: interactive LEDs and Kinect body tracking, along with respective clients to demonstrate the operation of each service.**

***Keywords—Smart Spaces, Internet of Things, Edge Computing, user-driven service, context aware***

## I. INTRODUCTION

Smart spaces are physical environments enhanced by computation and networking technology to create a complex ecosystem that users can interact with. Devices within a space can range from small distance or noise sensors to objects that take voice commands to smart lighting solutions using projectors. The devices are coordinated to provide context-aware network services for the user.

Smart home technology is one example of smart space device integration into daily life. These devices have gained traction in helping with home monitoring, security, entertainment, and in handling regular household tasks. Predictions suggest that by 2023 there will be 70.6 million smart homes [1]. Smart spaces also have the potential to improve personal and professional productivity. Analysts from the Gartner IT Symposium/Xpo identified people-centric smart spaces as the influence for the top ten strategic technology trends of 2020 [2]. Gartner defines a strategic technology as one with "substantial disruptive potential beginning to break out of an emerging state into broader impact and use". They have identified trends like multiexperience, empowered Edge, autonomous things, and other technology shown in Fig. 1.

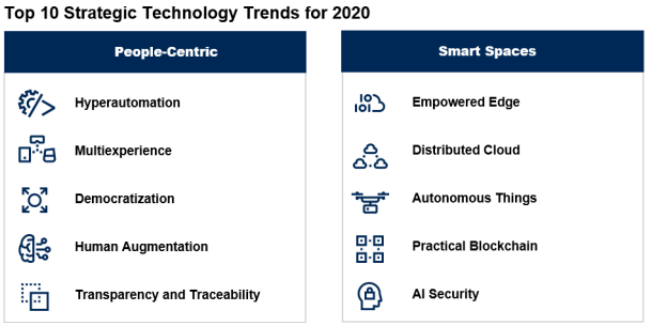


Fig. 1. 2020 Gartner IT Symposium/Xpo on people-centric technology trends [2]

Unfortunately, unlike smart homes and smart home devices, smart spaces are not one size fits all. They will be highly variant due to location, application, and user ability. For instance, a smart space used for entertainment will be designed differently from a space used by a workplace or a school. The main goal of designing a space is to meet the needs of its users and understand features necessary for each location.

The Ambient Edge project (Edge) led by Dr. Darren Carlson is a .NET based networking framework designed to enable fast creation and expansion of smart spaces through real-time low-latency messaging. It addresses the problem of high variability between each smart space by employing plug-and-play architecture, allowing designers to pick and choose the perfect devices for their ecosystem. An Ambient Edge smart space is comprised of Edge Services and Clients, all of which run on hosts throughout the space. An Edge Service is software that performs a task while managing underlying hardware of a sensor or device. An Edge Client connects and consumes one or more Edge Services. This paper summarizes the prototyping of two Smart Space services for Ambient Edge: interactive *light emitting diodes* (LEDs) and Kinect based body tracking, along with respective clients that demonstrate the operation of each service.

The remainder of the paper is structured as follows. Section 2 presents further background information on the Ambient Edge framework. Section 3 describes the hardware and software design process of the LED controller. Section 4 describes the same process in relation to the Kinect body tracking. Section 5 concludes the report.

## II. BACKGROUND

### A. *Edge Computing*

Edge computing is considered to be a descendant or extension of cloud computing. It is defined as "processing or computing close to or at the edge of a network, where the component, device, or application that produces data resides" [3-6]. To paraphrase, computing or network resources that lay between data sources and data centers.

Edge computing has become a viable solution to many challenges that have risen in the age of modern computing. The main challenge industries are facing is latency and response time in computation. In many cases, sending data to or receiving computations from a distant cloud layer is not realistic. Edge computing provides the benefit of computing within close proximity of the data source meaning latency and response time are drastically reduced [7]. Research conducted in 2015 demonstrated that a facial recognition application's response time was reduced from 900ms to 169ms by moving the computation to the edge away from the cloud [8]. In another example, Bombadier, an aerospace company, installed sensors into its jet engines [9]. Using edge computing, the sensor could instantly determine and relay the status of the engine, providing short term emergency data such as if the engine was overheating or if there was too little fuel. The second benefit of edge computing deals with bandwidth-greedy applications. With technology that generates significant amounts of data, relying on edge rather than cloud computing reduces network traffic drastically [10]. Since the computation occurs at the source of data—or very close to it—traffic loads are consequently reduced. The last benefit of edge computing is a reduction in storage and energy consumption at the device layer [4].

### B. *Ambient Edge*

Ambient Edge is a .NET based plug-and-play framework designed to handle real-time, low latency messaging and is based on the concept of edge computing that is discussed above. An Edge Network is a local area network that connects different devices within one network hop for low-latency communication. An Edge Host is a computer, virtual machine, or container that runs Edge related software such as an Edge Service. Services are logic running on host that perform tasks within the smart space such as managing sensors or high-level data output. Lastly, an Edge Client is also software running on a host that connects to and uses data from one or more Edge Services.

We can combine all the above terminology into the example that follows. An Edge Host could be a Raspberry Pi that is connected to the Edge Network. The Pi is also controlling the hardware of a distance sensor and an LED strip. The Pi would then be offering two Edge Services: i) controls and provides data from the distance sensor, and ii) control over the LED strip. An Edge Client could be a program running on another computer desktop on the network. It would subscribe to both services and tell the Pi to turn the LED strip on when someone walks within 2 feet of the distance sensor.

The Ambient Edge communication protocol is based on Representational State Transfer (REST) concepts [11]. Services are decomposed into resources that are addressed through URLs and accessed using predefined methods like GET, PUT, OBSERVE, and DELETE. GET is used to retrieve data, but it is read-only so there is no risk in mutating or corrupting the data. PUT is used to update an existing service resource. The OBSERVE method is unique to Edge; it's used to connect to a resource and stream data continuously and asynchronously. Lastly, DELETE is used to remove a resource such as an observer.

Within a service there are resources that can be accessed such as properties (exposes an attribute of the service), actions (invoke a function of the service), or events (data source that pushes data asynchronously from the service). Messages between Services and Clients are then encoded using Google Protocol Buffers (protobuf) which is used for serializing structured data [12].

Now we will extend the previous example and relate it to the communication protocol. The distance sensor service's data and configuration could be accessed using the URLs hcsr04.distance.data and hcsr04.distance.config, respectively. HCSR04 is the name of the hardware, distance is the attribute, and data and config are two ways to interact with the attribute. The Edge Client would then use GET to access distances it senses, or use PUT to configure the sampling rate of the sensor.

## III. AMBIENT EDGE LED SERVICE

This section will discuss the design and implementation of the Edge LED service. It will detail the project objectives and final design.

### A. *Design Goals*

The task was to design an LED Service and Client that could control the WS2812 Integrated Light Source—referred to as NeoPixels. NeoPixel strips are composed of individually addressable RGB color pixels using single-wire control protocol. Red, green, and blue LEDs are integrated with a driver chip into a surface mounted package that composes a single light pixel in the strip [13]. The final goal of the Edge Service was to dynamically control the NeoPixel hardware according to an Edge Client's requests. The micro-goals that comprise the foundation of this project are described in the User Story. A user story is a description of an Edge Service from the perspective of a client. The interactions with the service were decided below:

- A user can turn the LED strop on or off at any time
- A user has a choice of different light shows
- A user can run the lights for a set period of time or indefinitely
- Multiple users can make lighting requests to the same LED strop at the same time
- A user can request the current lighting information
- A user can dim up or dim down the lights over a default period of time or a specified period of time; the user can also specify how dim or how bright the final setting will be

### B. *Final Design*

#### 1) *Hardware*

This section will discuss the final hardware design. The required hardware was: Raspberry Pi 3, Arduino Uno, 100µF capacitor, 2.1mm female DC adapter, and a NeoPixels strip. Fig. 2 shows the schematic diagram. The Data In pad on the LEDs are connected to pin 5 on the Arduino. It also shows how

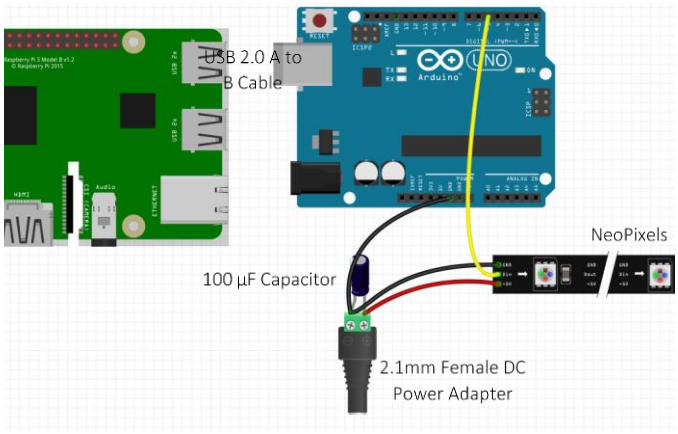


Fig. 2. Raspberry Pi to Arduino to NeoPixels hardware setup

to connect external power to the NeoPixels. External power to the Arduino was unnecessary because it drew enough through the Raspberry Pi connection.

The NeoPixel strip was not directly connected to the Raspberry Pi SPI because problem arose in previous iterations. NeoPixels use a single-wire control protocol so they can be controlled using a single *general-purpose input/output* (GPIO) pin on the Raspberry Pi. However, the control signal has very strict timing requirements that some boards (such as the Raspberry Pi) cannot achieve. The Raspberry Pi runs a multi-tasking Linux operating system and does not have real-time control over its GPIO pins [14]. To test it, *serial peripheral interface* (SPI) was enabled, and the core frequency was set to 250 in the boot file. However, running tests still showed dimness and flickering during lighting transitions on the NeoPixels. By chaining an Arduino to the Pi, timing was delegated out to a platform that could handle it.

*2) Software: Arduino*

The next step was verifying that the NeoPixels could run off of the Arduino by testing lighting programs. The FastLED Arduino library which is used for programming addressable LED strips [15]. The library deals with the low-level math, brightness settings, power usage, and performance while keeping the front-end easy to read and write. FastLED is also a well-developed and supported library meaning there were many lighting examples to pull from.

The next step was finding a way to have a .NET application communicate with the Arduino. This was more challenging than expected because an Arduino is microcontroller, and its code is a binary CPU dependent executable. Replacing or modifying existing code during runtime requires uploading and overwriting old code, which brought up questions on how the client's new requests would be handled by the Arduino. In addition, the basic structure of Arduino code executes whatever is in the loop() function continuously. This meant the Arduino would also have to asynchronously listen for requests from the .NET service to stop or run certain lighting events.

The Sharer library [16] is an Arduino/.NET serial communication library that allows a desktop application to read or write variables, and perform remote function calls on the Arduino. Fig. 3 shows how the Sharer protocol works between a .NET app and the Arduino program.

A selection of lighting examples from the FastLED library were saved into separate functions. The Sharer connection was initialized in the setup() function and the loop() function was dedicated to Sharer.run() which runs the internal kernel of Sharer that decodes the commands received over serial port. There was an early attempt to standardize the running time for each lighting function on the Arduino by having a millisecond for-loop encapsulate the lighting code, however later in the project this caused runtime and threading problems that stalled the whole system. It was decided that all timing and high-level logic should be maintained on the .NET side, rather than splitting it between both applications. The final Arduino code houses the lighting code, simply listens for commands, and immediately stops all logic when the commands cease.

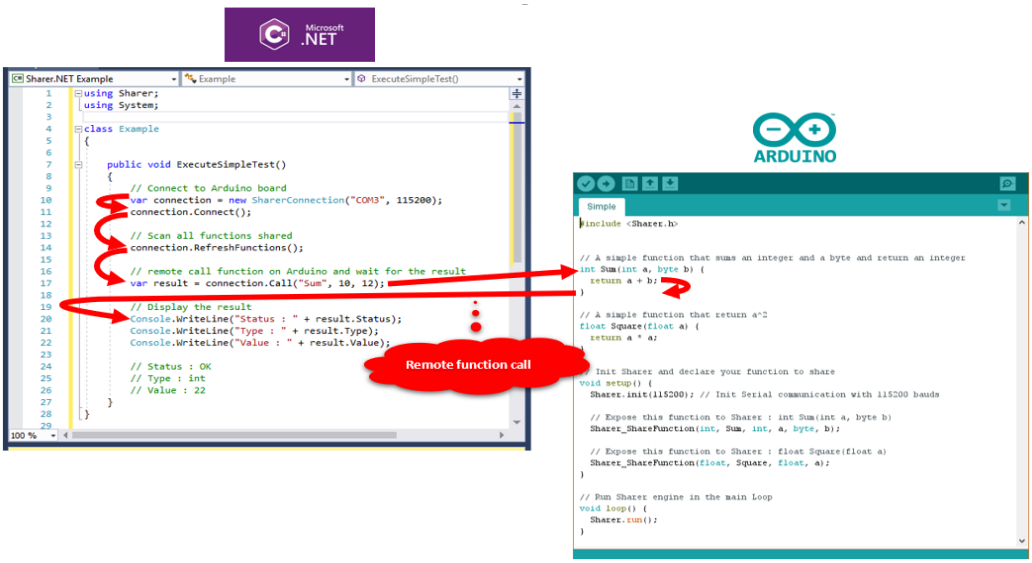


Fig. 3. Raspberry Pi to Arduino to NeoPixels hardware setup

TABLE I. LED SERVICE API

| Verb | Path | Behavior |
|---|---|---|
| GET | .lighting | Returns the current lighting config as a structured object including state, mode, and running time |
| PUT | .lighting.config | Accepts new lighting configuration data from remote clients, it changes the services lighting state, mode, and running time to match the request |

*3) Software: LED Service*

Finally, after dealing with all the low-level design problems, work started on the .NET service application. This section will be discussing the development process, important function handlers, and problems encountered along the way.

The first task was determining the Service's API based on the User Story discussed in Subsection A. The LED Service API is shown in Table 1.

The LED Service uses the Google Protobuf serialization format to structure the data processed by the service. The LightingConfig message has three different fields shown below. It describes the current state of the LEDs as mode, state, and running time. Mode indicates if the LEDs are set on a timed or untimed event. Running time is the amount of time in milliseconds a timed lighting event will run for. State indicates the lighting function the Arduino will be told to run. The fields hue, saturation, start_value, and end_value are used for dimming or brightening the LEDs over a set period of time. Lastly, the color field was added in later for ease of use. The user can specify a string color input instead of using the HSV to set the lighting.

```
message LightingConfig{

    Mode requested_mode = 1;

    enum Mode{
      TIMED = 0;
      UNTIMED = 1;
    }

    int32 running_time = 2;

    State requested_state = 3;

    enum State{
      TURNOFF = 0;
      BLINKRAINBOW = 1;
      SINELON = 2;
      RAINBOW = 3;
      RAINBOWWITHGLITTER = 4;
      CONFETTI = 5;
      BPM = 6;
      JUGGLE = 7;
      FIRE = 8;
      PACIFICA = 9;
      SINGLECOLOR = 10;
      DIM = 11;
    }

      int32 hue = 4;
      int32 saturation = 5;
      int32 start_value = 6;    //  Brightness of the
SINGLECOLOR state or starting value for DIM
      int32 end_value = 7;      // Ending value for DIM
      string color = 8;         //String input for color,
precedence over HSV values, ignored if null
  }
```

The initialization portion of the code sets up the Edge and Sharer connection. Sharer requires the serial port that the Arduino is connected to and the baud rate, which is the rate at which information is transferred over the port. For desktop, the port name is "COM3", while on the Pi it is "/dev/ttyUSB0". The baud rate was set to the maximum speed at 115200. After setting the strings, a new Sharer connection was instantiated, Sharer.Connect() was called to connect the .NET application to the Arduino application, and Sharer.Call() was used to communicate the current LED state to the Arduino.

After determining the basic service structure, next were the GET and PUT handlers. The GET handler was straightforward as it just returns the LightingConfig object back to the client. The PUT handler was more difficult; it parses the LightingConfig update and runs the new LED configuration asynchronously in the background of the main program thread. This allows the Edge service to continue listening for new PUT or GET requests while running the requested lighting event. During development there were several issues with threading timing and the Sharer connection. In addition to full task cancellation, the last Sharer.Call() had to fully finish before starting any new tasks. The timing issues were handled by sleeping the main thread before instantiating a new task to allow the last Sharer.Call() to finish.

*4) Software: LED Client*

The LED Edge Client's development was straightforward. It was a simple program to test the functionality of the LEDs. The software connects to the Edge Network and chooses to subscribe to the LED service. From there it makes GET and PUT requests from lighting information and configurations. During testing multiple Edge Clients were also instantiated to show how more than a single client can connect to the Edge Service. As soon as a new Client PUTs a new lighting configuration, the LED service would immediately show the change. This was the simplest way to handle new requests, but in the future queuing could be implemented as well. A final demonstration of the LEDs in action can be here.

## IV. AMBIENT EDGE KINECT SERVICE

This section will discuss the design and implementation of the Edge Kinect service. It will discuss the project objectives, background information, and final design. The Kinect service was more complicated to implement because there were plans to integrate Edge into Unity as a package and create a scene using data from the service as a stretch goal.

### A. *Design Goals*

The following were the objectives for this project. The first task was to design a Kinect Service that was strongly based off the Azure Kinect Developer Kit API [17]. The Kinect is a camera with sophisticated computer vision and AI sensors. It contains a 1-MP depth sensor, 7-microphones for speech and sound capture, a 12-MP RGB video camera, an accelerometer, and a gyroscope [18]. Since there are multiple data sources the Kinect can collect, this project focused solely on accessing IMU and body-tracking. The goal for the Kinect Client was to have an Edge-Integrated Unity scene run while collecting and displaying the data from the service.

The following User Story describes some of the smaller milestones for this project:

- A user can collect and IMU and body tracking data from the Kinect
- A user cannot configure the Kinect directly; it can only be configured by a smart space manager since it is a room wide service
- Multiple users can make IMU and body tracking requests at the same time
- A user can play with a simple client in Unity to visualize how the body-tracking data can be used

### B. *Azure Kinect Background Information*

The Azure Kinect Sensor Software Development Kit *application programming interface* (API) and the body tracking API were used as the foundation for the service messages [18-19]. The API is primarily in C; however, the documentation also covers the C++ wrapper. This was the first problem encountered when beginning the Kinect Service because Edge uses C# and .NET. However, there was a .NET NuGet wrapper available, which was used to complete the service [20]. Another .NET package was also found for the Body Tracking API [21]. There is no official documentation for the C# wrapper on the API page, so the C++ API was used as a reference instead to build the service functionality and .proto file.

Fig. 4. Kinect Camera Viewer

The Azure Sensor Development Kit also has a Kinect Camera Viewer that can be downloaded to see the different functionalities shown in Fig. 4. This includes an infrared camera, depth camera, color camera, and IMU data. The left-hand panel also shows the different camera configurations possible. As stated above, these configurations are set to a default in the Edge Service and can later be changed by some service manager.

The Azure Body Tracking API documents the three key components of a body frame. Each frame contains a collection of body structs, a 2D body index map, and the input capture that generated the results. The Edge Kinect service focusses on collecting data from the body structs. Each struct contains Body ID, 32 joint positions, and joint orientations. Joint position and orientation are estimates relative to the global depth sensor frame for reference. The position is specified in millimeters and the orientation is expressed as a normalized quaternion. The estimates also have a confidence level of none, low, medium, and high; these levels indicate whether a joint is out of range, predicted, or within frame. Fig. 5 shows the joint

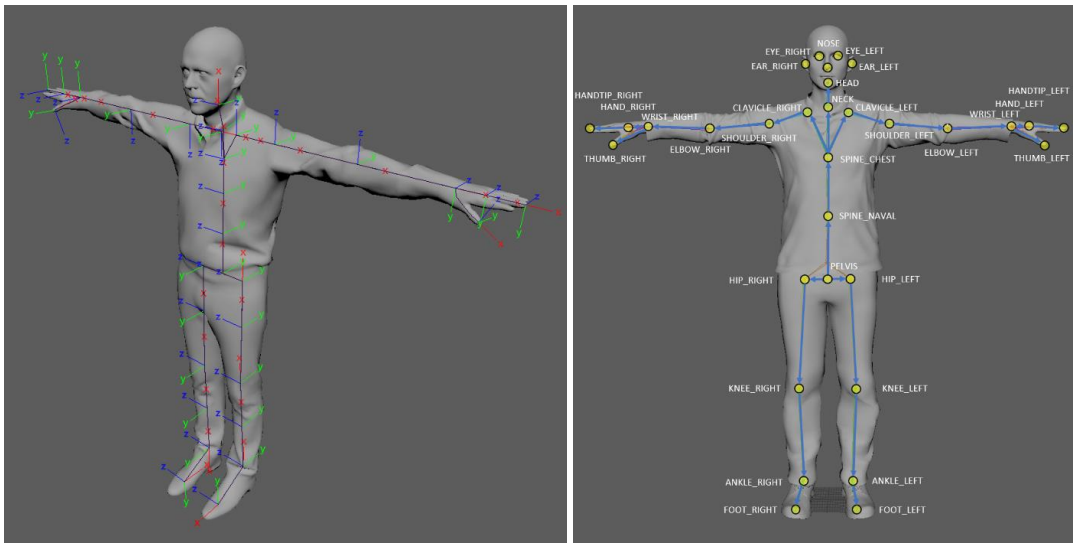


Fig. 5. Kinect bodytracking joints and axis

TABLE II. KINECT SERVICE API

| Verb | Path | Behavior |
|---|---|---|
| GET | .imu | Returns the current IMU data as a structured object including timestamp, XYZ gyroscope, XYZ accelerometer, and temperature |
| OBSERVE | imu.observers .{sub-id} | Allows an observer to connect and get a continuous stream of IMU data |
| GET | .skeleton | Returns the current skeleton data as a structured object with body ID, XYZ information on 32 different joints, and a confidence level per joint |
| OBSERVE | skeleton.observers.{sub-id} | Allows an observer to connect and get a continuous stream of bodytracking data |

axis orientation and joint hierarchy. Joint coordinates are also in axis orientations because it is widely used with commercial avatars, game engines, and rendering software. The joint data points were easily adapted to a .proto message that will be discussed in the next subsection.

### C. Final Desgin

#### 1) Software: Kinect Service

Following the same development process as the LED service; the first task was determining the Service's API based on the User Story discussed previously. The Kinect Service API is shown in Table 2. This API is more complicated than the previous service because it is accessing two different attributes from the Kinect: *inertial measurement units* (IMU) and skeletal. The previous example only accessed one set of data called "lighting". As a result, separate GET, OBSERVE, and PUT functions had to be created per attribute that was being accessed.

Again, protobuf was used to serialize the data. The code for the packaged information is shown below. There are separate messages for the IMU and skeleton attributes. The IMUEvent message has four different fields including: the timestamp of when the data was collected, gyroscopic data, accelerometer data, and ambient temperature. The SkeletonEvent message has two fields: BodyID and Joint. The Joint data is composed of another message called JointID, which contains all 32 Joints specified in the Azure Kinect API. Each Joint is described by a helper XYZ message that contains the X, Y, and Z coordinates of the Joint in space. The helper function also has Confidence which is an enum ranging from Not Applicable, None, Low, Medium, and High. This proto message was designed to match the confidence enum from the Kinect API discussed above. Confidence is a level dictated by the Kinect on how accurate the tracking of a specific joint is. High is currently not implemented by the Kinect hardware, but the Microsoft developers kept it in for future updates [19]. Medium is currently the highest level that can be returned; it indicates that the joint is directly in the camera view and not obscured by anything. If a joint has Low confidence it may be out of the camera view, but the computer vision can extrapolate its coordinates based off of the surrounding joints that are in frame. Lastly, None means that the surrounding joints are also out of frame so no predictions are made.

```
message IMUEvent{
  string timestamp = 1;       // Duration between device
turn on and message send
  XYZ gryoscope_raw = 2;
  XYZ accelerometer_raw = 3;
  float temperature = 4;
}

message SkeletonEvent{
  int32 body_id = 1;
  JointID joint = 2;
}

message JointID{
  XYZ pelvis = 1;
   XYZ ear_right = 32;
}

// Helper for XYZ values
message XYZ {
  float x = 1;
  float y = 2;
  float z = 3;
  level Confidence = 4;

  enum level{
  option allow_alias = true;
    NOT_APPLICABLE = 0;
    NONE = 0;
    LOW = 1;
    MEDIUM = 2;
    HIGH = 3;
  }
}
```

The Fall 2020 update to the Ambient Edge package included several structural changes made to Edge Services. This included a new Command Line Setup region which creates a command line prompt with options for service setup. It specifies getters and setters for clustername, servicename, and edgeurl which are variables used in creating the Edge connection that helps services and clients attach to the network. This was helpful during testing because prior to the update these values were hardcoded into the program. Within the update services also utilize a running while-loop to keep the program thread alive. This update dealt with asynchronous running issues that would time the program out if the response took too long.

The following walks through the important sections of the service code.

Within the Init() function default IMU and skeleton messages are created. This was required in order to avoid the error "System.NullReferenceException: 'Object reference not set to an instance of an object.'" which occurs if the objects are not initialized. The service will try to fill the object message, but if it is not instantiated prior, it will throw an error. Since the XYZ data for each joint is also a .proto message, each joint has to be set to a new XYZ() object as well. For example,

skeletonevent.Joint.Pelvis = new XYZ() must be instantiated before populating the pelvis data.

After the Edge connection is opened, StartCamera() is called. StartCamera() is a function in the camera control region that opens the Kinect device, configures it, and then starts the IMU capture and body tracking capture. The StopCamera() function stops the Kinect camera and IMU collection and disposes of the camera properly so that the device is not overloaded.

In the Register Routes region, the different service routes discussed above are created. Since there are also two observation routes for IMU and skeleton, two separate resource managers are also set up. These managers deal with creating and removing observers for the data streams.

The final regions of the program are the IMU Resource Handlers and the Skeleton Resource Handlers. When calling GET for IMU data, the IMUCapture() function runs. This function calls the Kinect device to get an IMU sample, and then the sample is formatted to fill the protobuf IMUEvent message. This handler is very straightforward. The GET handler for body tracking was more difficult to create. The GetHandler() function calls the SkeletonCapture() function which configures the payload and returns the Edge response. Within SkeletonCapture(), PresentSkeletonTracking() is also called. The beginning of the function flushes the Edge SkeletonEvent to default in case it was previously populated with data. Then it gets a capture from the Kinect device and enqueues the capture in the body tracker function. Finally, within a large for-loop, the function gets all of the bodies within the capture and assigns a Body ID along with populating the Kinect joint data into the protobuf message format. This required using a large switch statement to check which joint was being read from the Kinect data and then properly assigning it to the correct protobuf joint.

*2) Software: Console and Unity Client Iterations*

The development for the Kinect Client had multiple iterations leading up to a Unity Integrated Client. The first iteration was a simple console client that was used to test the service usage. The client tests GET and ASYNC GET requests for the IMU and skeleton data. It also sets up an observer for each resource for a certain amount of time, then closes the connection. In this client all of the joint data is printed to the console.

The second iteration of the console Kinect Client test using the different body tracking data. This program sets up an observer to stream the body tracking data. Using the data, it will print to console if the user's right hand, left hand, or both hands are raised. This was a simple way to test if the body coordinates were working together correctly. To see if a hand was raised the program checks if the Y coordinate of the hand joints are greater than the Y coordinate of the head. The test was simple but successful, and this type of gesture checking could be used for future Unity games.

The next step was to create a Unity client that utilized the Kinect data in a substantial way. Unity Integration was a brand-new concept, and it had a steep learning curve. The first Unity Client iteration involved modifying an example from Dr. Carlson to establish an Edge connection to the Kinect service. The scene shows a spinning sphere that displays the IMU data and a cylinder that displays the skeleton data. Originally the program was not updating the cylinder with data properly, however it turned out to be an issue of trying to access the IMU and skeleton data synchronously. This was not an issue in previous clients, so it will have to be furthered explored in the Unity setting.

The last Unity Client that was completed for this project was a scene containing a ball that moves with the players right hand movement. A skeleton observer is instantiated at the beginning of the program and runs for 30 seconds. Within the dispatcher enqueue there is logic to update a Vector3 instance on the sphere object with XYZ joint data from the right hand. Within the Sphere object there is the Vector3 instance and on Update() the instance's x and y coordinates are used to transform the sphere position. The coordinates are also transformed so that movements on screen mirrors the player's movements in front of the camera. This client was time consuming because it was difficult to determine how to transform the sphere based off the Kinect coordinates. It required a lot of trial and error to get the motion fluid. The Kinect coordinates needed to be divided by a factor of 100 so that the sphere would not immediately fly out of the scene. Inertia and acceleration also greatly affected moving the ball with simple hand gestures. All in all, it was still a successful demonstration of feasibility. This simple sphere could be further expanded into a game where the user has to collect tokens by moving across the screen. Body tracking has a variety of applications within a smart space so this service will be put to great use in the future.

## V. Conclusion

Ambient Edge has been a unique and multi-faceted platform to work with. Each project required delving into a different style of computer engineering and problem solving. Designing the LEDs required circuit setup, microcontroller programming, and exporting high-level code to a Raspberry Pi. The Kinect Service did not require hardware setup because it was a Plug and Play camera. However, it did require a thorough understanding of the Microsoft API and time to translate it to the Edge platform. Prior to even starting to program I had to understand what Ambient Edge was, how to platform worked, the messaging style, networking, etc. Both services were successfully completed despite the setbacks encountered each semester.

There is a lot of room for further development on these projects as more students join Ambient Lab. The LED service could be expanded to include more lighting displays or modify the Sharer function to have control over the speed of different states (how fast things blink, move, etc.). The Kinect service has so many different ways it could be expanded upon, but

working on more in depth Unity clients should be a priority. Developing a full-fledged Unity game based on the body tracking data sent real-time will really illustrate how robust and low-latency the Edge Network is.

## Acknowledgements

Developing services for Ambient Edge was a unique graduate experience that could only be had by working in the Ambient Computing Lab. I have learned so much about people-centric engineering, making gradual progress towards a greater vision, and having fun while doing it. Thank you, Dr. Carlson, for fostering such an inspiring and exciting environment.

## References

[1] "Smart Home - United States | Statista Market Forecast", Statista, 2020. [Online]. Available: https://www.statista.com/outlook/279/109/smart-home/united-states.

[2] K. Costello and M. Rimol, "Gartner identifies the top 10 strategic technology trends for 2020," *Gartner*, 21-Oct-2019. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2019-10-21-gartner-identifies-the-top-10-strategic-technology-trends-for-2020. [Accessed: 1-Feb-2021].

[3] S. Yi, Z. Qin, and Q. Li, "Security and Privacy Issues of Fog Computing: A Survey," Wireless Algorithms, Systems, and Applications Lecture Notes in Computer Science, pp. 685–695, 2015

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, 2016.

[5] Shi, W.; Dustdar, S. The promise of edge computing. Computer 2016, 49, 78–81.

[6] Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. IEEE Access 2017, 6, 6900–6919.

[7] Satyanarayanan, M. The emergence of edge computing. Computer 2017, 50, 30–39.

[8] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb), Washington, DC, USA, 2015, pp. 73–78.

[9] D. Linthicum, "Edge computing vs. fog computing: Definitions and enterprise uses," Cisco, 12-Dec-2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html. [Accessed: 27-Feb-2021].

[10] I. Sittón-Candanedo, R. S. Alonso, Ó. García, L. Muñoz, and S. Rodríguez-González, "Edge Computing, IoT and Social Computing in Smart Energy Scenarios," Sensors, vol. 19, no. 15, p. 3353, 2019.

[11] R. T. Fielding, "CHAPTER 5," *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed: 23-Feb-2021].

[12] Google, "API Reference | Protocol Buffers | Google Developers," *Protocol Buffers*. [Online]. Available: https://developers.google.com/protocol-buffers/docs/reference/overview. [Accessed: 23-Feb-2021].

[13] "Adafruit NeoPixel Überguide", Adafruit Learning System, 2020. [Online]. Available: https://learn.adafruit.com/adafruit-neopixel-uberguide. [Accessed: 5-Feb-2020].

[14] "NeoPixels on Raspberry Pi," *Adafruit Learning System*, 2021. [Online]. Available: https://learn.adafruit.com/neopixels-on-raspberry-pi. [Accessed: 02-Mar-2020].

[15] "FastLED/FastLED", GitHub, 2020. [Online]. Available: https://github.com/FastLED/FastLED/tree/master/examples.

[16] "Rufus31415/Sharer", GitHub, 2020. [Online]. Available: https://github.com/Rufus31415/Sharer.

[17] Microsoft, "Azure Kinect DK – Develop AI Models: Microsoft Azure," – Develop AI Models | Microsoft Azure. [Online]. Available: https://azure.microsoft.com/en-us/services/kinect-dk/. [Accessed: 18-Dec-2020].

[18] Microsfot Azure, *Azure Kinect Sensor SDK: Welcome*. [Online]. Available: https://microsoft.github.io/Azure-Kinect-Sensor-SDK/master/index.html. [Accessed: 10-Dec-2020].

[19] Microsoft, *Azure Kinect Body Tracking SDK: Welcome*. [Online]. Available: https://microsoft.github.io/Azure-Kinect-Body-Tracking/release/0.9.x/index.html. [Accessed: 24-Feb-2021].

[20] Microsoft and Azure Kinect, "Microsoft.Azure.Kinect.Sensor 1.4.1," NuGet Gallery | Microsoft.Azure.Kinect.Sensor 1.4.1. [Online]. Available: https://www.nuget.org/packages/Microsoft.Azure.Kinect.Sensor/. [Accessed: 18-Dec-2020].

[21] Microsoft and Azure Kinect, "Microsoft.Azure.Kinect.BodyTracking 1.0.1," NuGet Gallery | Microsoft.Azure.Kinect.BodyTracking 1.0.1. [Online]. Available: https://www.nuget.org/packages/Microsoft.Azure.Kinect.BodyTracking/. [Accessed: 18-Dec-2020].