Personal Project Portfolio

SIF-402 HoloLens AR Monitoring System

Mixed Reality Interface for Industrial Automation

Project Information

Role: Lead XR Developer & System Integrator
Institution: Uppsala University
Program: Master of Science in Computer Science
Specialization: Industrial Analytics
Duration: 2025/10/25 - 2026/01/13
Tech Stack: Unity 2022.3 LTS, C#, MRTK 2.8.3, OpenXR, HoloLens 2, Node-RED, OPC UA

November 24, 2025

Contents

1 Executive Summary

This portfolio documents the development of an end-to-end Mixed Reality monitoring system that bridges industrial automation with augmented reality interfaces. The SIF-402 AR Station Monitor project demonstrates the complete pipeline from OPC UA industrial protocols through REST APIs to immersive AR visualization on the Microsoft HoloLens 2.

1.1 Project Overview

The application renders real-time PLC telemetry data (hopper levels, station status, alarms) onto a spatial UI in the operator's field of view, enabling hands-free machinery monitoring. This project showcases the integration of diverse technologies including industrial protocols, middleware, game engines, and AR hardware into a cohesive system.

1.2 Key Achievements

- Developed complete Unity application with 570+ lines of C# code across 7 modular scripts
- Implemented Node-RED middleware bridging OPC UA and HTTP REST APIs
- Achieved 60 FPS stable holographic rendering on HoloLens 2 with ¡100ms API latency
- Successfully resolved 7+ critical blocking issues across build, deployment, and networking
- Integrated 8 different technologies into functional full-stack AR system
- Demonstrated modular architecture enabling seamless swap between mock and live data sources
- Note that all End to End Data work has been done through and with the help of Eric Dario Márquez León

2 System Architecture

2.1 End-to-End Data Pipeline

The system follows a four-layer architecture bridging industrial automation and augmented reality:

System Architecture Layers

Layer 1: Industrial PLC (Siemens B&R)

- Automation Runtime with three signals:
- `MESPAOEE_udiRunningTime` (UInt32)
- `MESPAOEE_bAlarm` (Boolean)
- `MESPAEA_rCurrent` (Float)

Layer 2: Node-RED Middleware (Port 1880)

- OPC UA Client connecting to `opc.tcp://130.130.130.2:4840`
- JavaScript function nodes for data transformation
- HTTP server nodes exposing REST endpoints

Layer 3: Unity Application

- ApiDataManager with UnityWebRequest (3-second polling)
- JSON parsing and deserialization
- Event-driven UI updates

Layer 4: HoloLens 2 Display

- MRTK UI components with hand tracking
- World Space Canvas (scale: 0.001)
- Real-time spatial visualization

2.2 Data Flow Mapping

PLC Signal	**API Endpoint**	**Unity Display**
MESPAOEE_udiRunningTime	/mes/blue_latest	Blue Hopper Level
MESPAOEE_bAlarm	/mes/yellow_latest	Yellow Hopper + Alarm
MESPAEA_rCurrent	/mes/red_latest	Red Hopper Level

Table 1: PLC Signal to UI Mapping

3 Key Technical Contributions

3.1 Contribution 1: Full-Stack Industrial IoT Integration

3.1.1 Challenge

Industrial PLCs communicate via OPC UA protocol, which is incompatible with Unity's native capabilities. Direct integration would require complex low-level protocol implementation within Unity.

3.1.2 Solution Architecture

Architected a middleware layer using Node-RED to bridge OPC UA and HTTP REST APIs:

1. **OPC UA Connection:** Configured OPC UA client node connecting to PLC at `opc.tcp://130.130.130`

2. **Data Transformation:** Implemented JavaScript function nodes to transform PLC values into Unity-compatible JSON format
3. **HTTP Endpoints:** Created three REST API endpoints serving real-time data
4. **State Management:** Used flow variables to store latest values for immediate API responses

3.1.3 Technical Implementation

Node-RED Data Transformation Example:

```
// Transform PLC RunningTime to Hopper Level
msg.payload = {
    "station": {
        "id": "SIF-402",
        "status": "Running",
        "recipe": "StandardMix_v2.3",
        "timestamp": $now()
    },
    "hoppers": [{
        "id": "hopper_blue",
        "color": "blue",
        "level": $floor(payload.read.value.value % 100),
        "hasPellets": ($floor(payload.read.value.value % 100) > 10),
        "status": ($floor(payload.read.value.value % 100) > 50) ? "OK" : "Low"
    }],
    "alarms": ($floor(payload.read.value.value % 100) < 20) ?
        [{
            "id": "alarm_low_runtime",
            "severity": "warning",
            "message": "Runtime-derived low level",
            "timestamp": $now()
        }]
    : []
}
```

Listing 1: Blue Hopper Data Transformation

Unity API Manager Implementation:

```
IEnumerator FetchHopperData(string endpoint, int hopperIndex,
                           List<AlarmData> alarmsList,
                           Action<bool> callback)
{
    string url = baseUrl + endpoint;

    using (UnityWebRequest request = UnityWebRequest.Get(url))
    {
        request.timeout = 5; // 5 second timeout

        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
            try
            {
                string json = request.downloadHandler.text;
                HopperResponse response =
                    JsonUtility.FromJson<HopperResponse>(json);

                // Update hopper data
                currentData.hoppers[hopperIndex] = response.hoppers[0];
```

```

                    // Collect alarms from this endpoint
                    if (response.alarms != null)
                        alarmsList.AddRange(response.alarms);

                    callback?.Invoke(true);
                }
                catch (Exception e)
                {
                    Debug.LogError($"Parse error: {e.Message}");
                    callback?.Invoke(false);
                }
            }
            else
            {
                // Mark hopper as offline on connection failure
                currentData.hoppers[hopperIndex].status = "Offline";
                OnConnectionError?.Invoke($"Failed to connect to {endpoint}");
                callback?.Invoke(false);
            }
        }
}
```

Listing 2: Real-time API Data Fetching with Coroutines

3.1.4 Impact

- Enabled real-time (3-second refresh) visualization of live industrial data
- Decoupled Unity from industrial protocol complexity
- Provided flexible middleware for data transformation and validation
- Achieved ¡100ms API response time for low-latency monitoring

3.2 Contribution 2: Modular Data-UI Architecture

3.2.1 Challenge

Need flexibility to switch between mock JSON data (development) and live API data (production) without breaking existing UI bindings or requiring code changes.

3.2.2 Solution: Three-Layer Architecture

Layer 1: Data Structures (DataClasses.cs)

Centralized data structure definitions shared by all components:

```
using System;

[Serializable]
public class StationData
{
    public Station station;
    public HopperData[] hoppers;
    public AlarmData[] alarms;
}

[Serializable]
public class Station
{
    public string id;
    public string status;
    public string recipe;
    public string timestamp;
}

[Serializable]
public class HopperData
{
    public string id;
    public string color;
    public int level;
    public bool hasPellets;
    public string status;
}

[Serializable]
public class AlarmData
{
    public string id;
    public string severity;
    public string message;
    public string timestamp;
}
```

Listing 3: Shared Data Structures

Layer 2: Data Providers (Swappable)
Two implementations with identical interface:

- `MockDataManager`: Loads from JSON Resources for offline development
- `ApiDataManager`: Fetches from Node-RED REST endpoints for live deployment

Layer 3: Presentation Layer (StationInfoDisplay)
Event-driven UI updates consuming StationData regardless of source:

```
public class StationInfoDisplay : MonoBehaviour
{
    [Header("Data Manager")]
    public ApiDataManager apiDataManager;

    [Header("UI Text Fields")]
    public TextMeshProUGUI stationStatusText;
    public TextMeshProUGUI recipeText;
    public TextMeshProUGUI blueHopperData;
    public TextMeshProUGUI yellowHopperData;
    public TextMeshProUGUI redHopperData;

    void Start()
    {
        // Subscribe to data updates
        if (apiDataManager != null)
        {
            apiDataManager.OnDataUpdated += OnDataReceived;
            apiDataManager.OnConnectionError += OnError;
        }
    }

    void OnDestroy()
    {
        // Unsubscribe to prevent memory leaks
        if (apiDataManager != null)
        {
            apiDataManager.OnDataUpdated -= OnDataReceived;
            apiDataManager.OnConnectionError -= OnError;
        }
    }

    void OnDataReceived(StationData data)
    {
        UpdateDisplay(data);
    }

    void UpdateDisplay(StationData data)
    {
        if (data == null) return;

        // Color-coded status feedback
        if (stationStatusText != null && data.station != null)
        {
            stationStatusText.text =
                $"Station Status: {data.station.status}";

            switch (data.station.status)
            {
                case "Running":
                    stationStatusText.color = Color.green;
                    break;
                case "Alarm":
                    stationStatusText.color = Color.red;
                    break;
                case "Warning":
                    stationStatusText.color = Color.yellow;
                    break;
                case "Disconnected":
                case "Connecting...":
                    stationStatusText.color = Color.gray;
                    break;
            }
        }

        // Update recipe information
        if (recipeText != null && data.station != null)
        {
            recipeText.text = $"Recipe: {data.station.recipe}";
        }

        // Update all hopper displays
        if (data.hoppers != null && data.hoppers.Length >= 3)
        {
            UpdateHopperDisplay(blueHopperData, data.hoppers[0]);
            UpdateHopperDisplay(yellowHopperData, data.hoppers[1]);
            UpdateHopperDisplay(redHopperData, data.hoppers[2]);
        }
    }

    void UpdateHopperDisplay(TextMeshProUGUI textField,
                             HopperData hopper)
    {
        if (textField == null || hopper == null) return;

        textField.text = $"Level: {hopper.level}%\n" +
                        $"Present: {(hopper.hasPellets ? "Yes" : "No")}\n" +
                        $"Status: {hopper.status}";

        // Color code based on status
        if (hopper.status == "Offline")
```

```
            textField.color = Color.gray;
        else if (hopper.status == "Low" || hopper.status == "Empty")
            textField.color = new Color(1f, 0.5f, 0.5f); // Light red
        else
            textField.color = Color.white;
    }
}
```

Listing 4: Event-Driven UI Updates

3.2.3 Impact

- Developed entire UI and interaction logic using mock data
- Switched to live PLC data by swapping single component reference
- Zero UI code changes required for production deployment
- Enabled parallel development: UI team using mock data while backend team integrates PLC

3.3 Contribution 3: UWP Deployment Pipeline Mastery

3.3.1 Challenge

HoloLens 2 requires ARM64 UWP builds with strict OpenXR configuration, creating multi-layered deployment complexity spanning Unity, Visual Studio, and device settings.

3.3.2 Build Configuration Management

Critical Settings:

Setting	Required Value
Build Type	D3D Project (NOT XAML)
Architecture	ARM64
Target SDK Version	10.0.22621.0 (Latest)
Min Platform Version	10.0.10240.0
Visual Studio Version	2022
Build Configuration	Release (testing), Master (production)
Scripting Backend	IL2CPP
API Compatibility Level	.NET Standard 2.1

Table 2: Unity Build Settings for HoloLens 2

3.3.3 OpenXR Feature Configuration

Resolved multiple feature conflicts:

1. **Holographic Remoting Conflict:** Disabled "Holographic Remoting remote app" for device deployment (keep enabled only for Unity Editor testing via WiFi)
2. **Hand Tracking Setup:** Enabled Microsoft HoloLens feature group with Hand Tracking subsystem
3. **Interaction Profiles:** Added Microsoft Hand Interaction Profile and Eye Gaze Interaction Profile

4. **Initialization:** Configured "Initialize XR on Startup" for standalone operation

3.3.4 Network Permissions Configuration

Multiple layers of network permission required:

- **Unity Player Settings:** "Allow downloads over HTTP" (default: blocked for security)
- **UWP Capabilities:** InternetClient, InternetClientServer, PrivateNetworkClientServer
- **Windows Firewall:** Inbound rule for Node-RED port 1880
- **HoloLens Developer Mode:** Enabled for deployment access

3.3.5 Version Management Strategy

Implemented systematic version incrementing:

Version	Changes	Status
1.0.0.0	Initial deployment	Failed (version conflict)
1.0.2.0	Fixed 2D window issue	Success
1.0.3.0	Added API integration	Success
1.0.4.0	Fixed BACK button	Success
1.0.5.0	Final testing build	Success

Table 3: Version History and Deployment Status

3.3.6 Deployment Workflow

1. **Unity Build** (10-15 minutes): File → Build Settings → Build
2. **Visual Studio Configuration**: Open .sln, set Release/ARM64
3. **Device Connection**: USB cable or WiFi (Remote Machine with HoloLens IP)
4. **Deploy**: Ctrl+F5 (Start Without Debugging)
5. **On-Device Validation**: Launch app, verify hand tracking, test API connectivity

3.3.7 Impact

- Achieved stable 60 FPS holographic rendering
- Successfully deployed 5+ builds with zero runtime crashes
- Mastered complex multi-tool deployment pipeline
- Documented complete configuration for future projects

3.4 Contribution 4: MRTK Interaction System Implementation

3.4.1 Challenge

Unity's standard UI components don't work with HoloLens hand tracking. MRTK requires specific component combinations for spatial interaction.

3.4.2 Button Component Architecture

Each interactive button requires four essential components:

MRTK Button Component Stack

1. **Interactable (MRTK)**
   - State management (Default, Focus, Pressed, Disabled)
   - Event system (OnClick callbacks)
   - Visual theme application
2. **PressableButton (MRTK)**
   - Physical press detection
   - Button depression animation
   - Press distance configuration
3. **NearInteractionTouchable (MRTK)**
   - Hand proximity detection
   - Touch event generation
   - Bounds definition for touch area
4. **Box Collider (Unity)**
   - Physics collision boundary
   - Raycast target for far interaction
   - Size matching button dimensions

3.4.3 Event System Implementation

```
public class StartupManager : MonoBehaviour
{
    [Header("Buttons")]
    public Interactable startButton;
    public Interactable settingsButton;
    public Interactable exitButton;

    void Start()
    {
        // Connect button events with null checks
        if (startButton != null)
        {
            startButton.OnClick.RemoveAllListeners();
            startButton.OnClick.AddListener(OnStartButtonClicked);
            Debug.Log("Start button listener connected");
        }

        if (settingsButton != null)
        {
            settingsButton.OnClick.RemoveAllListeners();
            settingsButton.OnClick.AddListener(OnSettingsButtonClicked);
        }

        if (exitButton != null)
        {
            exitButton.OnClick.RemoveAllListeners();
            exitButton.OnClick.AddListener(OnExitButtonClicked);
        }
    }

    void OnStartButtonClicked()
    {
        Debug.Log("START button clicked");

        // Smooth navigation between spatial UI panels
        if (startupMenu != null)
            startupMenu.SetActive(false);

        if (mainControlPanel != null)
            mainControlPanel.SetActive(true);
    }

    void OnExitButtonClicked()
    {
        Debug.Log("EXIT button clicked");

        #if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
        #else
        Application.Quit();
        #endif
    }
}
```

Listing 5: MRTK Button Event Binding

3.4.4 Interaction Features Implemented

- **Hand Tracking:** Real-time hand skeleton detection with 26 joint points per hand
- **Near Interaction:** Direct hand poke gestures for button presses
- **Far Interaction:** Hand ray casting for distant object interaction
- **Visual Feedback:** State-based color changes (Focus: highlight, Pressed: darker)
- **Audio Feedback:** Optional sound effects on button press (not implemented in current version)

3.4.5 Menu Navigation System

Implemented smooth spatial UI transitions:

1. **Startup Menu:** Three buttons (START, SETTINGS, EXIT) with green/blue/red color coding
2. **Main Control Panel:** Station status, recipe, three hopper displays, BACK button
3. **Navigation Flow:** StartupMenu → START → MainControlPanel → BACK → StartupMenu

3.4.6 Impact

- Natural, intuitive hand-based interaction matching HoloLens UX standards
- Zero learning curve for users familiar with HoloLens shell
- Hands-free operation without requiring controllers or voice commands
- Responsive feedback system with ¡50ms interaction latency

3.5 Contribution 5: Robust JSON Data Parsing System

3.5.1 Challenge

Handle complex nested data structures (Station → multiple Hoppers → nested Alarms) with resilient error handling and type safety.

3.5.2 Solution: Type-Safe Deserialization

```csharp
public class MockDataLoader : MonoBehaviour
{
    public static StationData LoadData(string filename)
    {
        try
        {
            // Load JSON file from Resources folder
            TextAsset jsonFile = Resources.Load<TextAsset>(filename);

            if (jsonFile == null)
            {
                Debug.LogError($"Could not find file: {filename} " +
                               "in Resources folder");
                return null;
            }

            // Deserialize with Unity's JsonUtility
            StationData data =
                JsonUtility.FromJson<StationData>(jsonFile.text);

            // Validate loaded data
            if (data.station == null || data.hoppers == null)
            {
                Debug.LogError("Invalid data structure in JSON file");
                return null;
            }

            Debug.Log($"Successfully loaded data from {filename}\n" +
                      $"Station: {data.station.id}, " +
                      $"Status: {data.station.status}\n" +
                      $"Hoppers: {data.hoppers.Length}, " +
                      $"Alarms: {data.alarms.Length}");

            return data;
        }
        catch (Exception e)
        {
            Debug.LogError($"Error loading data: {e.Message}\n" +
                           $"Stack trace: {e.StackTrace}");
            return null;
        }
    }
}
```

Listing 6: JSON Parsing with Error Handling

3.5.3 Data Validation Strategy

- **Null Checks:** Validate each deserialized object before use
- **Type Safety:** Use [Serializable] attributes for JsonUtility compatibility
- **Array Bounds:** Check array lengths before accessing elements

- **Default Values:** Provide fallbacks for missing optional fields
- **Logging:** Comprehensive debug output for troubleshooting

3.5.4 JSON Schema Example

Listing 7: Station Data JSON Format

```
{
    "station": {
        "id": "SIF-402",
        "status": "Running",
        "recipe": "StandardMix_v2.3",
        "timestamp": "2024-11-11T14:30:00Z"
    },
    "hoppers": [
        {
            "id": "hopper_blue",
            "color": "blue",
            "level": 75,
            "hasPellets": true,
            "status": "OK"
        },
        {
            "id": "hopper_yellow",
            "color": "yellow",
            "level": 45,
            "hasPellets": true,
            "status": "OK"
        },
        {
            "id": "hopper_red",
            "color": "red",
            "level": 20,
            "hasPellets": true,
            "status": "Low"
        }
    ],
    "alarms": [
        {
            "id": "alarm_001",
            "severity": "warning",
            "message": "Red hopper level low",
            "timestamp": "2024-11-11T14:25:00Z"
        }
    ]
}
```

3.5.5 Impact

- Handled both mock JSON files (development) and API responses (production) with identical parsing logic

- Enabled seamless testing workflow with offline data
- Provided clear error messages for debugging data issues
- Maintained type safety throughout data pipeline

4 Technical Problem-Solving Showcase

4.1 Problem 1: The 2D Window Mystery

Symptom

Application deployed to HoloLens appeared as flat 2D slate window instead of holographic 3D interface with spatial positioning.

Investigation Process:

1. Verified OpenXR plugin installation and configuration
2. Checked scene setup for MRTK components (all present)
3. Compared working projects vs. current build settings
4. Researched HoloLens rendering requirements in Microsoft documentation

Root Cause: Build Type set to "XAML" (default for Universal Windows Platform) instead of "D3D Project." XAML creates 2D slate applications, while D3D Project enables holographic/immersive rendering.

Solution:

- File → Build Settings → Build Type → D3D Project
- Rebuilt application completely (not just incremental)
- Redeployed to HoloLens 2

Lesson Learned: Build configuration fundamentally determines rendering mode in UWP. This single setting controls whether the application runs in 2D overlay mode or full spatial computing mode.

4.2 Problem 2: The OpenXR Build Failure

Error Message

```
BuildFailedException: ‘‘Holographic Remoting remote app’’ and
‘‘Initialize XR on Startup’’ are both enabled. XR initialization should
be delayed until a specific IP address is entered.
```

Investigation Process:

1. Reviewed OpenXR feature documentation
2. Analyzed feature dependencies and conflicts
3. Tested different feature combinations

Root Cause: Two mutually exclusive OpenXR features enabled simultaneously:

- "Holographic Remoting remote app": For streaming Unity Editor content to HoloLens over WiFi (development)
- "Initialize XR on Startup": For standalone application launch (deployment)

Solution:

- Edit → Project Settings → XR Plug-in Management → OpenXR
- Under Features, disabled "Holographic Remoting remote app"
- Kept "Initialize XR on Startup" enabled for device deployment
- Note: Re-enable Holographic Remoting only for Unity Editor testing

Lesson Learned: OpenXR features have context-specific compatibility requirements. Some features are for development workflows, others for production deployment.

4.3 Problem 3: The HTTP Security Block

Error Messages

```
UriFormatException:  Invalid URI: Invalid port specified.
Non-secure network connections disabled in Player Settings
```

Investigation Process:

1. Verified Node-RED server running and accessible from browser
2. Tested API endpoints with Postman (successful)
3. Checked Unity console for network errors
4. Researched Unity network security policies

Root Causes:

1. Unity blocks HTTP connections by default for security (only HTTPS allowed)
2. Base URL format errors (missing http://, incorrect port format)

Solutions:

1. **Enable HTTP:** Edit → Project Settings → Player → Other Settings → "Allow downloads over HTTP" → "Allowed in Development builds only"
2. **Correct URL Format:** Ensured `http://192.168.1.100:1880` (no trailing slash)
3. **Network Capabilities:** Publishing Settings → Capabilities → PrivateNetworkClientServer

Lesson Learned: Local development requires explicit HTTP permission. Production deployments should always use HTTPS with proper SSL certificates.

4.4 Problem 4: The Lost References

Symptom

UI stopped responding to button clicks after Visual Studio 2022 installation. Inspector showed "None" for previously connected references.

Investigation Process:

1. Verified scene file integrity (not corrupted)
2. Checked if GameObjects still existed in Hierarchy (present)
3. Reviewed Visual Studio installation changes
4. Researched Unity project file regeneration

Root Cause: Unity regenerated project files (.sln, .csproj) during Visual Studio installation, breaking Inspector references. Scene file preserved GameObject references, but script component fields lost connections.

Solution: Systematically reconnected all references:

1. Selected GameManager in Hierarchy
2. Dragged StartupMenuCanvas → Startup Menu field
3. Dragged MainControlPanel → Main Control Panel field
4. Expanded StartupMenuCanvas, dragged each button to corresponding field
5. Repeated for MainControlPanel references
6. Verified all fields showed object names (not "None")

Lesson Learned: Unity's .sln/.csproj regeneration is normal behavior. Scene files preserve GameObject references, but Inspector field connections require manual restoration. Always save scene before major IDE changes.

4.5 Problem 5: The Invisible StationInfoDisplay

Symptom

API data fetched successfully (confirmed in Console logs), but UI showed "NULL" values for recipe and hopper information.

Investigation Process:

1. Verified API endpoints returning correct JSON data
2. Confirmed ApiDataManager logging successful data fetch
3. Checked UI text components existed in scene
4. Traced data flow: ApiDataManager → ??? → UI TextMeshPro
5. Discovered missing link: StationInfoDisplay component

Root Cause: StationInfoDisplay component was never added to MainControlPanel GameObject. Without this component, no script existed to read data from ApiDataManager and update UI text fields.

Solution:

1. Selected MainControlPanel in Hierarchy
2. Add Component → StationInfoDisplay
3. Connected all references:
   - Api Data Manager → DataManager
   - Station Status Text → StatusText
   - Recipe Text → RecipeText
   - Blue/Yellow/Red Hopper Data → respective TextMeshPro objects

Lesson Learned: Unity's component-based architecture requires explicit wiring. Missing components fail silently without error messages. Always verify complete data flow path from source to UI.

4.6 Problem 6: The MRTK Button Confusion

Symptom

BACK button had Unity's standard "Button" component with OnClick() event list showing "List is Empty." Button did not respond to hand gestures.

Investigation Process:

1. Compared BACK button components with working START button
2. Reviewed MRTK documentation on button components
3. Tested button interaction in Play mode with hand simulation

Root Cause: Button used Unity's standard UI Button component instead of MRTK's Interactable component. Standard Unity Button doesn't support spatial interaction or hand tracking.

Solution:

1. Selected BackButton in Hierarchy
2. Removed Unity Button component (Right-click → Remove Component)
3. Added MRTK components in order:
   - Add Component → Interactable (MRTK)
   - Add Component → PressableButton (MRTK)
   - Add Component → NearInteractionTouchable (MRTK)
   - Verified Box Collider present
4. Reconnected in GameManager: BackButton → Back Button field in MainControlUI

Lesson Learned: MRTK has its own UI event system incompatible with standard Unity components. Spatial applications require complete MRTK component stack for hand tracking support.

4.7 Problem 7: The Network Permission Mystery

Symptom

API connection worked in Unity Editor but failed on HoloLens device with "Access Denied" or timeout errors.

Investigation Process:

1. Verified HoloLens and PC on same WiFi network
2. Tested Node-RED endpoints from HoloLens browser (successful)
3. Checked Windows Firewall rules
4. Reviewed UWP capability requirements

Root Cause: Missing network capability in UWP manifest. HoloLens requires explicit permission for private network access.

Solution:

1. Edit → Project Settings → Player → UWP tab
2. Publishing Settings → Capabilities
3. Enabled: InternetClient, InternetClientServer, **PrivateNetworkClientServer**
4. Windows Firewall: Allowed Node-RED port 1880 (Inbound rule)
5. Rebuilt and redeployed application

Lesson Learned: UWP sandbox requires explicit capability declarations. Local network access (PrivateNetworkClientServer) is separate from internet access (InternetClient).

5 Quantitative Results and Metrics

Metric	**Achievement**	**Measurement Method**
Code Lines Written	570+ lines	LOC count across 7 C# scripts
System Latency	¡100ms	Network profiler + Debug.Log timestamps
API Refresh Rate	3 seconds	Configured polling interval
Frame Rate	60 FPS	Unity Profiler on HoloLens 2
Deployment Success	5/5 builds	Version 1.0.2.0 through 1.0.5.0
Integration Points	3 layers	PLC → Node-RED → Unity → HoloLens
API Endpoints	3 REST APIs	/mes/blue/yellow/red_latest
Technologies	8 tools	Unity, C#, MRTK, OpenXR, Node-RED, etc.
Issues Resolved	7+ critical bugs	Documented in Problem-Solving section
Component Stack	4 per button	Interactable + PressableButton + Touch + Collider

Table 4: Project Metrics and Performance Indicators

5.1 Performance Benchmarks

Unity Editor Testing:

- Scene load time: ¡2 seconds
- Button response time: ¡50ms
- JSON parsing: ¡10ms for typical payload (500 bytes)
- API call overhead: 80-100ms (local network)

HoloLens 2 Device Performance:

- Holographic frame rate: 60 FPS (stable)
- Hand tracking latency: ¡20ms (MRTK default)
- World-locked UI stability: ¡2mm drift over 5 minutes
- Memory footprint: 250MB (with MRTK overhead)

6 Skills and Competencies Demonstrated

6.1 Unity Development

- World Space Canvas configuration for AR environments (scale: 0.001 for meter-based sizing)
- MRTK 2.8.3 integration with DefaultMixedRealityToolkitConfigurationProfile
- Coroutine-based asynchronous programming for non-blocking API calls
- Scene management, prefab organization, and component-based architecture
- Event-driven programming with C# delegates and UnityEvents
- UnityWebRequest for HTTP REST API consumption
- JsonUtility for type-safe JSON serialization/deserialization

6.2 HoloLens/XR Development

- OpenXR configuration for Microsoft HoloLens feature group
- Hand tracking setup with Microsoft Hand Interaction Profile
- UWP build pipeline: Unity $\rightarrow$ Visual Studio $\rightarrow$ Device deployment
- Windows Device Portal for wireless deployment and debugging
- Spatial UI design principles: positioning, readability, interaction distances
- World-locked vs. body-locked UI considerations
- Hand ray and near-interaction gesture handling

6.3 Industrial IoT Integration

- OPC UA protocol fundamentals for PLC communication
- Node-RED visual programming for IoT data flows
- REST API design for exposing industrial data to AR clients
- Real-time data streaming with sub-second latency requirements
- Network architecture: firewalls, capabilities, HTTP/HTTPS configuration
- Data transformation between industrial protocols and application formats

6.4 Software Engineering

- Modular architecture with separation of concerns
- Dependency injection pattern for swappable data providers
- Error handling and graceful degradation strategies
- Debug logging strategies for distributed systems
- Version control and incremental deployment
- Documentation-driven development
- Code refactoring for maintainability

6.5 Problem-Solving Methodology

- Systematic debugging across multiple technology layers
- Root cause analysis using elimination methodology
- Hypothesis-driven troubleshooting
- Documentation research and synthesis
- Cross-platform compatibility testing (Unity Editor HoloLens)
- Performance profiling and optimization

7 Project Timeline and Evolution

7.1 Sprint 1 2: Foundation (Weeks 1-2)

Objectives: Establish Unity project structure and core UI components
Tasks Completed:

- Unity project creation (Version 2022.3.62f2 LTS)
- MRTK 2.8.3 installation via Mixed Reality Feature Tool
- OpenXR plugin configuration for HoloLens 2
- World Space Canvas setup (StartupMenu, MainControlPanel)
- UI element creation: buttons, text fields, panels

- Script architecture: StartupManager, MockDataManager, StationInfoDisplay
- Mock JSON data system for offline development

Deliverables:

- Functional UI in Unity Editor with button navigation
- 350+ lines of C# code across 5 scripts
- Two mock JSON files for testing scenarios

7.2 Sprint 3: Deployment (Weeks 3-4)

Objectives: Configure UWP build pipeline and deploy to HoloLens 2
Tasks Completed:

- Build settings configuration (D3D Project, ARM64, Release)
- OpenXR feature resolution (disabled Holographic Remoting)
- Version management system (1.0.0.0 → 1.0.2.0)
- Visual Studio 2022 deployment configuration
- Windows Device Portal setup for wireless deployment
- MRTK component troubleshooting on buttons

Challenges Addressed:

- 2D window issue (XAML vs D3D Project)
- OpenXR feature conflicts
- Lost Inspector references after VS installation
- BACK button MRTK component stack

Deliverables:

- Successfully deployed application on HoloLens 2
- Stable 60 FPS holographic rendering
- Functional hand tracking interaction

7.3 Sprint 4: Integration (Weeks 5-6)

Objectives: Connect to live PLC data via Node-RED middleware
Tasks Completed:

- Node-RED installation and OPC UA client configuration
- PLC connection: `opc.tcp://130.130.130.2:4840`
- Data transformation function nodes (JavaScript)
- HTTP REST API endpoint creation (3 endpoints)
- Unity ApiDataManager script (150+ lines)

- DataClasses.cs for shared data structures
- StationInfoDisplay refactoring for API integration
- Network configuration: HTTP permissions, UWP capabilities, firewall rules

Challenges Addressed:

- HTTP security block in Unity Player Settings
- Network permissions for PrivateNetworkClientServer
- Component reference issues (missing StationInfoDisplay)
- URI format validation

Deliverables:

- Complete end-to-end data pipeline: PLC → Node-RED → Unity → HoloLens
- Real-time data visualization with 3-second refresh
- ¡100ms API response latency
- 570+ total lines of C# code

7.4 Sprint 5: Validation (Week 7)

Objectives: End-to-end testing, optimization, and documentation

Tasks Completed:

- Full system integration testing (PLC → HoloLens)
- Performance profiling with Unity Profiler
- Error handling validation
- Documentation: setup guide, contribution log, self-assessment
- Portfolio creation with LaTeX formatting
- Final deployment to HoloLens 2

Deliverables:

- Production-ready application
- Comprehensive technical documentation
- Project portfolio for academic/professional use

8 Lessons Learned and Reflections

8.1 Technical Insights

8.1.1 Unity and MRTK Integration

- MRTK 2.8.3 requires careful profile configuration; default settings rarely work out-of-box
- World Space Canvas scale of 0.001 is standard for meter-based sizing in AR
- Input Data Providers must include appropriate device managers for hand tracking
- MRTK Interactable events differ fundamentally from standard Unity UI Button events
- Component references can break during IDE updates; scene files preserve structure

8.1.2 HoloLens Development

- D3D Project build type is mandatory for immersive holographic apps
- ARM64 architecture required; x86/x64 builds will not deploy
- Version numbers must increment for each deployment; no downgrades allowed
- Network capabilities must be explicitly enabled in UWP manifest
- Development builds show debug console; disable for production

8.1.3 API Integration

- UnityWebRequest is the standard for HTTP requests; avoid WWW class (deprecated)
- Coroutines enable non-blocking API calls without freezing UI
- HTTP requests require explicit permission in Player Settings (default: blocked)
- Local network APIs require PrivateNetworkClientServer capability
- Error handling essential for unreliable network connections

8.1.4 Industrial IoT

- Node-RED effectively bridges industrial protocols (OPC UA) to web APIs (HTTP)
- Function nodes enable custom JavaScript data transformation
- Flow variables enable data persistence between API calls
- Multiple endpoints can serve different data subsets for optimization
- Middleware decouples Unity from complex industrial protocol implementation

8.2 Process Learnings

8.2.1 Development Workflow

- Test in Unity Editor before building; catches 80% of issues
- On-device testing reveals problems invisible in editor (hand tracking, performance)
- Mock data enables parallel development: UI team independent of backend
- Incremental builds with version tracking prevent deployment confusion

8.2.2 Debugging Strategy

- Add Debug.Log statements liberally; remove before production
- Use systematic elimination methodology for multi-layer issues
- Test components in isolation before integration
- Documentation research before experimentation saves time
- Cross-platform testing essential: Unity Editor HoloLens behavior

8.2.3 Documentation Approach

- Configuration changes easy to forget; document immediately
- Error messages and solutions should be recorded for future reference
- System architecture diagrams clarify data flow for team collaboration
- Code comments should explain "why," not "what"

8.3 Personal Growth

8.3.1 Skills Developed

- **From:** Basic Unity knowledge, no HoloLens experience
- **To:** Can create complete Unity projects for HoloLens 2
- **From:** Limited understanding of MRTK
- **To:** Mastery of MRTK component architecture and interaction system
- **From:** No industrial IoT integration experience
- **To:** Can architect full-stack solutions bridging PLC and AR interfaces

8.3.2 Key Growth Moments

1. **First HoloLens Deployment:** Realized importance of build configuration (D3D vs XAML)
2. **Resolving 2D Window Issue:** Learned how critical single settings can fundamentally change application behavior
3. **MRTK Component Debugging:** Gained confidence in troubleshooting complex interaction systems
4. **Integrating Live PLC Data:** Connected classroom theory to real industrial applications
5. **Node-RED Pipeline:** Understood power of middleware in bridging disparate protocols

9 Future Enhancements and Roadmap

9.1 Immediate Next Steps

1. **QR Code Spatial Anchoring**
   - Implement QR code detection for precise panel placement
   - Align UI relative to physical machinery using Azure Spatial Anchors
   - Enable multi-user shared spatial reference frames
2. **Alarm Notification System**
   - Visual alerts: flashing borders, color-coded severity
   - Audio feedback: spatialized 3D sound at alarm location
   - Priority queue: critical alarms interrupt workflow
3. **Historical Data Visualization**

- Time-series line graphs using Unity UI Toolkit
- 24-hour trend display for each hopper
- Interactive scrubbing through historical data

9.2 Medium-Term Enhancements

1. **Voice Command Integration**
    - Windows Speech Recognition for hands-free navigation
    - Commands: "Show alarms," "Hide panel," "Refresh data"
    - Multi-language support (English, Swedish, German)
2. **Collaborative Multi-User**
    - Azure Spatial Anchors for shared coordinate space
    - Real-time data synchronization across multiple devices
    - User presence indicators (avatars, annotations)
3. **Offline Data Caching**
    - SQLite database for local data storage
    - Automatic sync when network reconnects
    - "Last known good" values displayed during outages

9.3 Long-Term Vision

1. **Predictive Maintenance**
    - Machine learning models for failure prediction
    - Anomaly detection on sensor trends
    - Maintenance schedule optimization
2. **Digital Twin Integration**
    - 3D CAD model overlay on physical equipment
    - Real-time component status visualization
    - Interactive parts explosion for maintenance guidance
3. **Remote Expert Assistance**
    - Video call integration with holographic annotations
    - Remote pointer from expert to guide local technician
    - AR-based step-by-step repair instructions

10 Appendices

10.1 Appendix A: Code Repository Structure

```
Assets/
 Scripts/
    DataClasses.cs                (50 lines)
    ApiDataManager.cs             (150 lines)
    MockDataManager.cs            (40 lines)
    StationInfoDisplay.cs         (80 lines)
    StartupManager.cs             (80 lines)
    MainControlUI.cs              (30 lines)
    StationAnchor.cs              (30 lines)
    PositionInFrontOfCamera.cs    (60 lines)
 Resources/
    station_data_small.json
    station_mock_data.json
 Scenes/
    SIF402_MainScene.unity
 MRTK/
    [MRTK package files]
 Samples/
     Mixed Reality Toolkit Examples/

Node-RED-Flow/
 PLC_3_signals_Out_File_OK.json

Documentation/
 SIF402_Setup_Guide.pdf
 Contribution_Log.pdf
 Self_Assessment.pdf
```

10.2 Appendix B: Build Settings Checklist

Setting	**Required Configuration**
Platform	Universal Windows Platform
Architecture	ARM64
Build Type	D3D Project (NOT XAML)
Target SDK	10.0.22621.0 or latest
Min Platform	10.0.10240.0
Visual Studio	2022
Build Configuration	Release (testing), Master (production)
Scripting Backend	IL2CPP
API Compatibility	.NET Standard 2.1
OpenXR Features	Microsoft HoloLens, Hand Tracking
OpenXR Remoting	Disabled (for device deployment)
HTTP Downloads	Allowed in Development builds
Capabilities	InternetClient, InternetClientServer, PrivateNetworkClientServer

Table 5: Complete Build Settings Checklist

10.3 Appendix C: API Endpoint Documentation

Base URL: `http://[NODE-RED-IP]:1880`

Endpoint 1: Blue Hopper

- **Path:** `/mes/blue_latest`
- **Method:** GET
- **Response:** JSON (StationData with blue hopper)
- **Data Source:** PLC signal `MESPAOEE_udiRunningTime`

Endpoint 2: Yellow Hopper

- **Path:** `/mes/yellow_latest`
- **Method:** GET
- **Response:** JSON (StationData with yellow hopper + alarm)
- **Data Source:** PLC signal `MESPAOEE_bAlarm`

Endpoint 3: Red Hopper

- **Path:** `/mes/red_latest`
- **Method:** GET
- **Response:** JSON (StationData with red hopper)
- **Data Source:** PLC signal `MESPAEA_rCurrent`

10.4 Appendix D: Network Configuration

Unity Player Settings:

- Edit → Project Settings → Player → Other Settings
- Allow downloads over HTTP: Allowed in Development builds

UWP Capabilities:

- Publishing Settings → Capabilities
- InternetClient
- InternetClientServer
- PrivateNetworkClientServer

Windows Firewall:

- Inbound Rule: Allow TCP port 1880
- Profile: Private
- Name: "Node-RED HTTP Server"

Symptom	Solution
App appears in 2D	Build Type → D3D Project
OpenXR build fails	Disable Holographic Remoting
HTTP requests blocked	Player Settings → Allow HTTP
Lost Inspector refs	Reconnect all GameObject references
UI shows NULL	Add StationInfoDisplay component
Button not responding	Add MRTK Interactable + PressableButton
Network timeout	Check PrivateNetworkClientServer capability
Version conflict	Increment version number

Table 6: Common Issues and Quick Solutions

10.5 Appendix E: Troubleshooting Quick Reference

11 Contact Information and References

11.1 Student Information

Name: Hamza Sallam
Program: Master of Science in Computer Science
Specialization: Industrial Analytics
Institution: Uppsala University, Sweden
Email: [sallamhamza77@gmail.com]
LinkedIn: [https://www.linkedin.com/in/hamza-sallam]
GitHub: [https://github.com/Sallamhamza]

11.2 Project Repository

Source Code: [https://github.com/Sallamhamza/unity-hololens-industrial-data]

11.3 Key Technologies and Resources

- **Unity:** https://unity.com/
- **MRTK:** https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/
- **HoloLens 2:** https://www.microsoft.com/en-us/hololens
- **Node-RED:** https://nodered.org/
- **OPC UA:** https://opcfoundation.org/