Simplifying Data Acquisition in Production

An Apple Vision Pro Application — RobotSelector

Apple Vision Pro (visionOS) & iPad (iPadOS) • Swift, SwiftUI, RealityKit, ARKit, Roboflow, InfluxDB

Identifying KUKA robotic arms on the shop floor via AR and surfacing their live machine telemetry in place — removing the need for a separate data-acquisition dashboard.

Student Engineering Research Program (SERP)
Yuvraj — Industrial Engineering
Purdue University

1 INTRODUCTION

- Industrial mixed reality lets a technician look at real machinery and see live, contextual data floating in space — no separate dashboard or manual lookup required.
- This SERP project builds RobotSelector: a mixed-reality app that recognizes a physical KUKA robotic arm through the camera, lets the user select it with a gaze/pinch or a tap, and displays its live telemetry (temperature, speed, load) directly above it in AR.
- The app targets two platforms from largely shared code: Apple Vision Pro (visionOS, hands-free gaze + pinch) and iPad (iPadOS, tap-to-select via ARKit raycasting), so it can be tested on hardware immediately available in the lab.
- The project grew out of an earlier iOS object-detection prototype (AVFoundation + Core ML), then moved to ARKit scene reconstruction on Vision Pro, then added a custom-trained computer-vision model and live sensor telemetry.
- Key challenge: consumer AR platforms are built around strict camera-privacy models, and general-purpose vision models don't reliably distinguish one KUKA arm from the surrounding machinery — both had to be solved from scratch.

2 PROJECT

Goal

- Detect a specific KUKA robot model in the camera view and let the user select it with a natural gesture.
- Overlay a floating panel above the selected robot with live telemetry pulled from InfluxDB.
- Share the RealityKit/ARKit selection and overlay logic across visionOS and iPadOS targets.

Technical Stack

- Swift, SwiftUI, RealityKit, ARKit (SceneReconstructionProvider, WorldTrackingProvider)
- Roboflow-hosted instance-segmentation model, custom-trained on KUKA arm images
- InfluxDB (line protocol writes, Flux queries) for live sensor telemetry
- Core ML / Vision framework (earlier iOS prototype: YOLOv3 object detection)

Code Sample — iPad Tap-to-Select (ARViewModel.swift)

```
func handleTap(at point: CGPoint) {
    let results = arView.raycast(
        from: point,
        allowing: .estimatedPlane,
        alignment: .any)
    guard let hit = results.first else {
        statusMessage = "Missed — try again"
        return
```

visionOS vs. iPadOS

Concept	Vision Pro	iPad
Entry point	ImmersiveSpace	WindowGroup only
AR view	RealityView	ARView (UIKit bridge)
Gesture	Gaze + pinch	Screen tap
Mesh access	SceneReconstructionProvider	ARWorldTrackingConfiguration
Hit detection	Mesh collision	arView.raycast()
Highlight	Material on mesh entity	Sphere at hit point

RealityKit entities, components, and materials are written once and reused unchanged on both platforms — only AR-session setup differs.

Live ML Detection Pipeline

Roboflow instance segmentation (not just bounding boxes) runs on-camera-frame every 1.5s via the Roboflow Serverless API, filtered to the KUKA class at ≥ 50% confidence — confirmed reliable between 20–88% threshold. The API key is kept out of source control via an .xcconfig file read at runtime through Bundle.main.infoDictionary.

Figure: live KUKA telemetry overlay, iPad build

Camera / AR Frame → **Roboflow Inference (every 1.5s)** → **Confidence Filter (≥ 0.5, KUKA class)** → **AR Highlight + Segmentation Overlay** → **InfluxDB Telemetry Polling (2s)**

3 DEVELOPMENT TIMELINE

01 — iOS Camera Prototype

Built a live camera app with AVFoundation (AVCaptureSession, UIViewRepresentable bridging) and Vision-framework object detection to learn the core capture pipeline before touching AR.

02 — Core ML Object Detection

Upgraded from Vision rectangle-detection (fails on curved objects) to a YOLOv3 Core ML model via VNCoreMLRequest, plus UUID-based tracking to stop bounding-box flicker.

03 — visionOS Mesh Selection

Built the first RobotSelector visionOS app: ARKit SceneReconstructionProvider meshes the room, SpatialTapGesture + collision highlights the tapped chunk orange.

04 — iPad Port

Ported selection logic to iPadOS using ARWorldTrackingConfiguration and arView.raycast(), with automatic fallback from LiDAR mesh to plane detection on non-LiDAR devices.

05 — InfluxDB Telemetry

Added an InfluxDB client (line-protocol writes, Flux queries) and a floating RobotDataOverlay panel showing live temperature, speed, and load above the selected robot.

06–07 — Custom ML via Roboflow

Replaced generic Vision segmentation with a custom-trained Roboflow instance-segmentation model for KUKA-specific detection, secured with an .xcconfig API key pattern.

08 — Camera Restriction Discovered

Found that visionOS 1.0–1.2 exposes no public API for raw camera frames — a deliberate privacy limit, not a bug — and built static-image workarounds to keep developing.

What's next →

Live camera access, multi-robot routing, and full hardware implementation on physical Apple Vision Pro (No iPad needed)

4 CHALLENGES AND LEARNINGS

1 — visionOS Camera Restriction

As of visionOS 1.0–1.2, Apple provides no public API to stream raw camera frames (CVPixelBuffer) to third-party apps — an intentional privacy decision, not a bug. This blocked the planned live Roboflow inference loop on Vision Pro hardware.

Fix: Workaround: mesh-based tap selection (geometry, not pixels) as a fallback, plus static test images cycled through the asset catalog to develop and demo the ML pipeline while awaiting a future frame-access API.

2 — Generic Segmentation Wasn't Enough

Apple's built-in VNGenerateForegroundInstanceMaskRequest segments any foreground object — it cannot tell a KUKA arm apart from other lab equipment, and offered no class labels or confidence scores.

Fix: Solution: trained a custom Roboflow instance-segmentation model on labeled KUKA images, called via the Roboflow Serverless API every 1.5s, filtering results to the KUKA robot class at ≥ 50% confidence.

3 — One Codebase, Two Interaction Models

visionOS uses gaze + pinch with mesh-entity collision; iPad has no gaze input and must raycast from a 2D tap into the 3D world. Early attempts duplicated app logic per platform.

Fix: Solution: kept ARKit session setup and gesture handling platform-specific, but shared all RealityKit entity, material, and InfluxDB telemetry code unchanged across both targets.

4 — Flickering & Camera Freezes

Early iOS prototype bounding boxes flickered because every detection shared the ID "Object," and later builds froze because Vision/Core ML work competed with the camera session on the same thread.

Fix: Solution: gave each detection a stable UUID (Identifiable), throttled inference to every 3–6 frames, capped zoom, and dropped resolution to 720p to keep the capture pipeline responsive.

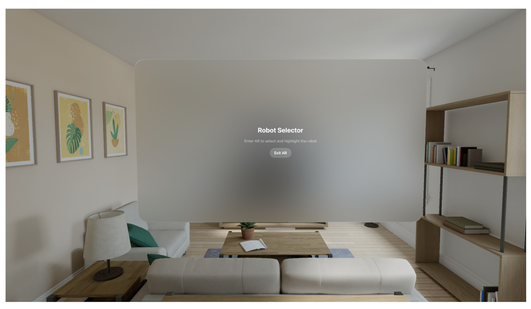

Figure: visionOS Simulator — Enter/Exit AR flow verified before physical Vision Pro access

This project built working knowledge of ARKit/RealityKit AR pipelines, cross-platform Swift architecture, custom computer-vision model training and deployment via Roboflow, and time-series telemetry integration with InfluxDB — while learning to recognize platform limitations early and design around them rather than against them. The visionOS Simulator (left) cannot stream a live camera or run scene reconstruction, but it was used throughout development to verify SwiftUI layout, the Enter/Exit AR flow, and InfluxDB polling logic without needing hardware in hand. It also confirmed that console warnings such as "Authorization requests cannot be performed on Simulator" are expected simulator behavior, not bugs.

5 NEXT STEPS

- Wire detected robot identity (kuka_1 / kuka_2) into the InfluxDB query filter for correct multi-robot telemetry routing.
- Test the full mesh-selection, segmentation-overlay, and telemetry pipeline on physical Apple Vision Pro hardware once available.
- Draw the AR overlay directly from Roboflow's segmentation polygon coordinates instead of a placeholder highlight.
- Cache the last good detection so the AR overlay persists smoothly between 1.5-second inference cycles.
- Continue labeling KUKA images in Roboflow to improve detection accuracy, recall, and performance across distances and lighting.
- Monitor upcoming visionOS releases and WWDC sessions for a supported live camera-frame API to replace the static-image workaround.
- Extract shared RealityKit material/component logic into a common Swift package used by both the visionOS and iPadOS targets.
- Add floating text annotation, proximity-based auto-highlighting, and spatial audio feedback on robot selection.