Rugby Vision - Complete Project Plan (Phases 1-13)

Overview

This document provides the complete roadmap for Rugby Vision development from initial POC through production deployment. Each phase builds incrementally on previous work, following agile principles with clear deliverables and success criteria.

Project Timeline

- **Phase 1-3**: Foundation (4-6 weeks) - ✅ COMPLETE
- **Phase 4-5**: Detection, Tracking & 3D Reconstruction (6-8 weeks) - ✅ COMPLETE
- **Phase 6**: Forward Pass Decision Engine (2-3 weeks)
- **Phase 7-9**: Backend Integration & Testing (6-8 weeks)
- **Phase 10-11**: Performance & Real-time (4-6 weeks)
- **Phase 12-13**: Documentation & Production Readiness (3-4 weeks)

Total Estimated Duration: 25-34 weeks (~6-8 months)

Phase 1: Project Bootstrap ✅ COMPLETE

Status: COMPLETED

Goal: Establish project structure and development standards.

Deliverables

- [x] Monorepo structure (`/frontend` , `/backend` , `/ml` , `/infra`)
- [x] React + TypeScript frontend scaffolding
- [x] Python FastAPI backend scaffolding
- [x] ARCHITECTURE_OVERVIEW.md
- [x] CONTRIBUTING.md with coding rules
- [x] Root README.md

Success Criteria

- [x] All directories created with proper structure
- [x] Build systems configured (package.json, pyproject.toml)
- [x] Documentation complete and reviewed
- [x] Coding standards defined

Phase 2: Requirements and Use Case Spec ✅ COMPLETE

Status: COMPLETED

Goal: Define precise, implementable specifications.

Deliverables

- [x] RUGBY_VISION_REQUIREMENTS.md
- [x] Primary user workflows documented
- [x] Key interfaces between components defined
- [x] Phase boundaries clearly marked
- [x] Constraints and success metrics documented

Success Criteria

- [x] Requirements cover all functional needs
- [x] Non-functional requirements (latency, accuracy) quantified
- [x] Stakeholder review completed

Phase 3: Multi-Camera Ingestion and Synchronization ✅ COMPLETE

Status: COMPLETED

Goal: Design and scaffold video ingestion + synchronization layer.

Deliverables

- [x] VIDEO_INGEST_DESIGN.md
- [x] `backend/video_ingest.py` - VideoIngestor class
- [x] `backend/video_sync.py` - VideoSynchronizer class
- [x] `ml/mock_data_generator.py` - Synthetic video generator
- [x] Unit test stubs in `backend/tests/`

Success Criteria

- [x] Can load multiple video files
- [x] Synchronizes frames within ±33ms tolerance
- [x] Handles missing frames gracefully
- [x] Unit tests pass for core logic
- [x] Synthetic data generation works

Phase 4: Player and Ball Detection and Tracking ✅ COMPLETE

Status: COMPLETED

Duration: 3-4 weeks

Goal: Implement detection and tracking subsystem for players and ball.

Tasks

1. **Model Selection and Setup**
   - Research and select YOLO variant (YOLOv8 recommended)
   - Set up model loading infrastructure in `/ml/detector.py`
   - Download pre-trained models or adapt existing ones

2. **Player Detection**
   - Implement `PlayerDetector` class
   - Fine-tune model on rugby-specific data (if available)
   - Achieve 80%+ precision on test data

3. **Ball Detection**
   - Implement `BallDetector` class (separate or combined with player detector)
   - Handle challenges: small size, motion blur, occlusion
   - Accept lower initial accuracy (60%+), improve iteratively

4. **Object Tracking**
   - Integrate DeepSORT or ByteTrack
   - Implement `Tracker` class in `/ml/tracker.py`
   - Maintain consistent track IDs across frames

5. **Data Structures**
   - Define `Detection` dataclass (bbox, class, confidence, camera_id, frame_id)
   - Define `Track` dataclass (track_id, class, detections list)
   - Document data flow in `DETECTION_TRACKING_OVERVIEW.md`

6. **Testing**
   - Create synthetic test data with known ground truth
   - Unit tests for detection and tracking logic
   - Measure precision/recall on test set

Deliverables

- `/ml/detector.py` - Detection module
- `/ml/tracker.py` - Tracking module
- `/ml/models/` - Model weights directory
- `DETECTION_TRACKING_OVERVIEW.md`
- Unit tests in `/ml/tests/`

Success Criteria

- Player detection: 80% precision, 85% recall
- Ball detection: 60% precision, 50% recall
- Tracking maintains IDs for 3+ seconds
- Processing speed: 15+ fps equivalent
- All tests pass

Dependencies

- Pre-trained YOLO models

- Synthetic test data (from Phase 3)

Phase 5: 3D Reconstruction and Field Reference Frame ✅ COMPLETE

Status: COMPLETED

Duration: 3-4 weeks

Goal: Reconstruct 3D positions of players and ball from multi-camera detections.

Tasks

1. **Camera Calibration Management**
   - Define calibration data format (JSON/YAML)
   - Implement `calibration.py` to load intrinsic/extrinsic parameters
   - Create sample calibration files for testing

2. **Triangulation Logic**
   - Implement `triangulation.py` with multi-view geometry
   - Use OpenCV's triangulation functions or custom implementation
   - Handle cases with 2, 3, or more cameras

3. **Field Coordinate System**
   - Implement `field_coords.py`
   - Define origin (try line), axes (x=width, y=length, z=height)
   - Transform 3D scene coords to field coords

4. **Frame State Computation**
   - Create `FrameState` dataclass (timestamp, ball_pos_3d, players_pos_3d)
   - Implement pipeline: detections → 3D points → field coords
   - Handle uncertainty and missing detections

5. **Validation and Error Checking**
   - Sanity checks: ball above ground plane, players on field
   - Compute reprojection error as quality metric
   - Log warnings for suspicious 3D positions

6. **Documentation**
   - Write `SPATIAL_MODEL_AND_COORDINATES.md`
   - Document coordinate system precisely
   - Explain error sources and mitigation

Deliverables

- `/ml/calibration.py` - Calibration management
- `/ml/triangulation.py` - 3D reconstruction
- `/ml/field_coords.py` - Coordinate transforms
- `SPATIAL_MODEL_AND_COORDINATES.md`
- Sample calibration files in `/ml/calibration_data/`
- Unit tests for geometry functions

Success Criteria

- 3D position accuracy: ±0.5m horizontal, ±0.2m vertical
- Handles 2-8 cameras
- Reprojection error < 5 pixels average
- All geometry tests pass
- Documentation complete

Dependencies

- Detection and tracking (Phase 4)
- Camera calibration data (can use synthetic initially)

Phase 6: Forward Pass Decision Engine - Physics Core

Duration: 2-3 weeks

Goal: Implement forward-pass decision engine using 3D data and physics.

Tasks

1. **Decision Criteria Definition**
   - Define mathematical criteria for forward pass
   - Phase 1: Simple displacement-based model
   - Document in `FORWARD_PASS_PHYSICS_MODEL.md`

2. **Pass Event Detection**
   - Identify pass start time (ball leaves passer's hands)
   - Identify pass end time (ball caught/grounded)
   - Use velocity and position changes as heuristics

3. **Ball Trajectory Analysis**
   - Compute ball velocity vector from 3D positions
   - Smooth trajectory (Kalman filter or polynomial fit)
   - Handle noisy measurements

4. **Decision Logic**
   - Implement `decision_engine.py`
   - Compute ball displacement along field axis
   - Account for player momentum (simplified for Phase 1)
   - Return `DecisionResult` (is_forward, confidence, explanation)

5. **Confidence Scoring**
   - Factor in: detection confidence, 3D reconstruction quality, trajectory smoothness
   - Provide actionable confidence (0-1 scale)

6. **Extensibility for Future Physics**
   - Abstract geometry into interfaces
   - Prepare for metric tensor model (Phase 2+)
   - Document extension points

7. **Testing**
   - Create synthetic scenarios: clearly forward, clearly backward, borderline

- Unit tests for all decision logic
- Validate with known ground truth

Deliverables

- `/ml/decision_engine.py` - Decision logic
- `FORWARD_PASS_PHYSICS_MODEL.md`
- Unit tests with synthetic scenarios
- Integration with 3D reconstruction

Success Criteria

- 100% correct on obvious test cases (10/10)
- 70%+ correct on borderline cases
- Confidence scores align with accuracy
- <1 second computation time per pass
- All tests pass

Dependencies

- 3D reconstruction (Phase 5)
- Synthetic pass scenarios

Phase 7: Backend API and Pipeline Glue

Duration: 2-3 weeks

Goal: Wire all components into coherent backend API.

Tasks

1. **API Endpoint Implementation**
   - Implement `POST /api/clip/analyse-pass`
   - Implement `GET /api/clip/{id}/debug-data`
   - Update `main.py` with orchestration logic

2. **Pipeline Orchestration**
   - Wire: ingestion → detection → tracking → 3D → decision
   - Guard clauses at each stage
   - Clear error handling with HTTP status codes

3. **Request/Response Models**
   - Refine `AnalysePassRequest` model
   - Refine `DecisionResult` model
   - Add validation (Pydantic)

4. **Logging and Metrics**
   - Add structured logging per stage
   - Track latency per component
   - Count failures and warnings

5. **Error Handling**
   - Graceful degradation (e.g., missing ball detection → low confidence)

   - Return actionable error messages
   - Log detailed errors for debugging

6. **Documentation**
   - Write `BACKEND_API_SPEC.md`
   - JSON schemas for requests/responses
   - Usage examples with curl/Postman

7. **Integration Testing**
   - End-to-end tests with synthetic data
   - Mock lower-level components for unit tests
   - Validate full pipeline flow

Deliverables

- Updated `/backend/main.py` with full orchestration
- `BACKEND_API_SPEC.md`
- Integration tests in `/backend/tests/`
- Logging and metrics infrastructure

Success Criteria

- API endpoints functional and documented
- End-to-end test passes with synthetic data
- Latency < 10 seconds for 5-second clip
- Error handling tested and robust
- All integration tests pass

Dependencies

- Phases 3-6 (all pipeline components)

Phase 8: Frontend UI for Referees and Analysts

Duration: 3-4 weeks

Goal: Build clean, intuitive UI for TMOs and analysts.

Tasks

1. **Core UI Components**
   - Video player component (multi-view or switchable)
   - Decision indicator (prominent, color-coded)
   - Confidence display
   - Explanation text panel

2. **Timeline Component**
   - Timeline with pass event markers
   - Scrubbing capability
   - Frame-by-frame controls

3. **Analysis Control Panel**
   - Clip selection/upload

   - Camera selection (checkboxes)
   - Time window controls (start/end time)
   - “Analyse Pass” button

4. **Results Display**
   - FORWARD / NOT FORWARD indicator
   - Confidence percentage
   - Explanation text (human-readable)
   - Optional: 2D field view overlay

5. **Debug View (Analysts)**
   - Per-frame detections overlay
   - 3D trajectory visualization (top-down view)
   - Export debug data (JSON download)

6. **State Management**
   - Use React hooks or context for state
   - Handle loading states
   - Error state display (clear, non-technical)

7. **Design and UX**
   - High contrast, accessible design
   - Large buttons for TMO use case
   - Responsive layout (desktop and tablet)
   - Simple, uncluttered interface

8. **Testing**
   - Component tests (Jest + React Testing Library)
   - E2E tests (Playwright/Cypress) for main workflow
   - Usability testing with target users

Deliverables

- Updated `/frontend/src/` with full UI
- `UI_OVERVIEW.md` documenting components
- Component tests
- E2E test suite
- Usability test report

Success Criteria

- TMO can use interface with <5 minutes training
- All core workflows functional
- UI tests pass
- Positive usability feedback
- Responsive on common screen sizes

Dependencies

- Backend API (Phase 7)

Phase 9: Testing Strategy - Unit, Integration, E2E

Duration: 2-3 weeks

Goal: Define and implement comprehensive test strategy.

Tasks

1. **Test Strategy Document**
   - Write `TEST_STRATEGY.md`
   - Coverage: unit, integration, E2E
   - Test data management
   - CI integration plan

2. **Backend Unit Tests**
   - Complete coverage for all modules
   - Pytest with fixtures
   - Target: 80%+ code coverage

3. **ML/CV Unit Tests**
   - Geometry functions (triangulation, transforms)
   - Decision logic with known scenarios
   - Mock detection/tracking outputs

4. **Integration Tests**
   - Full pipeline with synthetic data
   - API endpoint tests
   - Database (if any) integration tests

5. **Frontend Unit Tests**
   - Component tests for all UI components
   - Service/API client tests
   - State management tests

6. **E2E Tests**
   - Main workflow: load clip → analyse → view result
   - Error scenarios (e.g., invalid clip)
   - Multiple pass analyses

7. **Test Fixtures and Data**
   - Synthetic video fixtures
   - Mock calibration data
   - Expected output fixtures

8. **CI Integration**
   - Configure test runs in CI pipeline
   - Fail builds on test failures
   - Coverage reporting

Deliverables

- `TEST_STRATEGY.md`
- Complete test suites (unit, integration, E2E)
- Test fixtures in `/tests/fixtures/`

- CI test configuration
- Coverage reports

Success Criteria

- 80%+ code coverage (backend and ML)
- 70%+ code coverage (frontend)
- All tests pass in CI
- Test execution < 5 minutes
- Documentation complete

Dependencies

- All previous phases (comprehensive testing requires full system)

Phase 10: CI/CD and Deployment Pipeline

Duration: 1-2 weeks

Goal: Set up continuous integration and deployment infrastructure.

Tasks

1. **CI Configuration**
   - GitHub Actions (or GitLab CI) workflow
   - Steps: lint, type-check, test, build
   - Separate jobs for frontend and backend
   - Caching for faster builds

2. **Linting and Type Checking**
   - Black, flake8, mypy for Python
   - ESLint, Prettier, tsc for TypeScript
   - Fail builds on violations

3. **Docker Images**
   - Dockerfile for backend service
   - Dockerfile for frontend (Nginx)
   - Multi-stage builds for optimization
   - Push to container registry (Docker Hub, GCR, ECR)

4. **Docker Compose**
   - Compose file for local development
   - Services: frontend, backend, (optional) database
   - Volume mounts for hot-reload

5. **Deployment Scripts**
   - Deploy to staging environment
   - Deploy to production (manual approval)
   - Environment variable management
   - Secrets handling (API keys, credentials)

6. **Health Checks**
   - `/health` endpoint for backend

   - Kubernetes liveness/readiness probes (if K8s)
   - Simple uptime monitoring

7. **Documentation**
   - `CI_CD_SETUP.md`
   - Deployment procedures
   - Rollback procedures

Deliverables

- `.github/workflows/` or equivalent CI config
- `Dockerfile` for backend and frontend
- `docker-compose.yml` for local dev
- Deployment scripts in `/infra/`
- `CI_CD_SETUP.md`

Success Criteria

- CI runs on every push/PR
- All checks pass (lint, type, test)
- Docker images build successfully
- One-command deployment to staging
- Documentation complete

Dependencies

- Test suite (Phase 9)

Phase 11: Datasets, Labeling, and Model Training

Duration: 3-4 weeks (ongoing)

Goal: Plan and scaffold data collection, labeling, and model improvement.

Tasks

1. **Dataset Schema Design**
   - Define structure: videos + metadata + annotations
   - Annotation format: pass events (start/end time, forward/not forward)
   - Storage: local files or cloud (S3, GCS)

2. **Data Ingestion Pipeline**
   - Scripts to ingest match footage
   - Extract metadata (camera IDs, timestamps)
   - Organize into dataset structure

3. **Labeling Tool**
   - Simple web UI for labeling passes
   - Mark pass start/end times
   - Label as FORWARD / NOT FORWARD
   - Store annotations in JSON/CSV

4. **Labeling Workflow**
   - Assign clips to human labelers
   - Quality control (inter-rater agreement)
   - Export labeled dataset

5. **Model Training Pipeline**
   - Scripts in `/ml/training/`
   - Train/fine-tune detectors (players, ball)
   - Optional: train learned decision model
   - Track experiments (MLflow, Weights & Biases)

6. **Model Evaluation**
   - Hold-out test set
   - Compute precision/recall, F1 score
   - Compare to baseline

7. **Documentation**
   - `DATASET_AND_LABELING_PLAN.md`
   - Data collection guidelines
   - Labeling instructions for humans

Deliverables

- `DATASET_AND_LABELING_PLAN.md`
- Labeling UI (basic web interface)
- Data ingestion scripts
- Training scripts in `/ml/training/`
- Initial labeled dataset (synthetic + real if available)

Success Criteria

- Labeling tool functional
- 100+ passes labeled (synthetic + real)
- Training pipeline runs successfully
- Model performance tracked
- Documentation complete

Dependencies

- Detection/tracking infrastructure (Phase 4)
- Real match footage (may not be available initially)

Phase 12: Performance, Latency, and Real-Time (Phase 2)

Duration: 3-4 weeks

Goal: Prepare system for near-real-time operation (semi-real-time, Phase 2).

Tasks

1. **Performance Profiling**
   - Identify latency bottlenecks (detection, 3D, I/O)
   - Use profilers (cProfile, line_profiler for Python)
   - Measure per-stage latency

2. **Optimization Strategies**
   - Model quantization (FP16, INT8)
   - Frame subsampling (process every Nth frame)
   - Parallel processing (per-camera, per-stage)
   - GPU acceleration (CUDA, TensorRT)

3. **Live Stream Support**
   - Implement RTSP/RTMP ingestion
   - Handle buffering and jitter
   - Update `video_ingest.py` for streams

4. **Incremental Processing**
   - Process frames as they arrive (streaming mode)
   - Maintain sliding window buffer
   - Emit results incrementally

5. **Latency Targets**
   - Reduce total latency from 10s to 5s
   - Detection: <2s, 3D: <1s, Decision: <0.5s

6. **Performance Monitoring**
   - Add timing logs per stage
   - Dashboard for latency metrics
   - Alert on performance degradation

7. **Documentation**
   - `PERFORMANCE_AND_REALTIME_PLAN.md`
   - Optimization techniques applied
   - Trade-offs (latency vs accuracy)

Deliverables

- `PERFORMANCE_AND_REALTIME_PLAN.md`
- Optimized pipeline (GPU, quantization)
- Stream ingestion support
- Performance monitoring dashboard
- Latency < 5 seconds for 3-second clip

Success Criteria

- Latency reduced by 50%
- Supports RTSP streams
- GPU utilization > 70%
- Maintains 85%+ accuracy
- Documentation complete

Dependencies

- Full pipeline (Phases 1-9)

Phase 13: Documentation, Demo, and Pitch Material

Duration: 1-2 weeks

Goal: Package Rugby Vision for stakeholders, customers, and users.

Tasks

1. **Technical Documentation**
   - Update `README.md` with latest info
   - Create `RUGBY_VISION_TECH_OVERVIEW.md`
   - Ensure all design docs are current

2. **User Guide**
   - Write `RUGBY_VISION_USER_GUIDE.md` for TMOs/analysts
   - Step-by-step instructions
   - Screenshots and diagrams
   - Troubleshooting section

3. **Demo Script**
   - Prepare demo scenario (2-3 example passes)
   - Clear forward pass
   - Clear legal pass
   - Borderline case
   - Record demo video

4. **Demo Video**
   - Screen recording of full workflow
   - Voiceover explaining each step
   - Show decision results
   - Highlight key features

5. **One-Pager / Pitch Deck**
   - Problem statement
   - Solution overview
   - Key benefits (accuracy, speed, transparency)
   - Target users
   - Technology highlights

6. **API Documentation**
   - Generate OpenAPI/Swagger docs
   - Host at `/docs` endpoint
   - Include examples for all endpoints

7. **Deployment Guide**
   - Installation instructions
   - System requirements

- Configuration options
- Troubleshooting

Deliverables

- `RUGBY_VISION_USER_GUIDE.md`
- `RUGBY_VISION_TECH_OVERVIEW.md`
- Demo script and video
- One-pager (PDF)
- Pitch deck (slides)
- Hosted API documentation
- Deployment guide

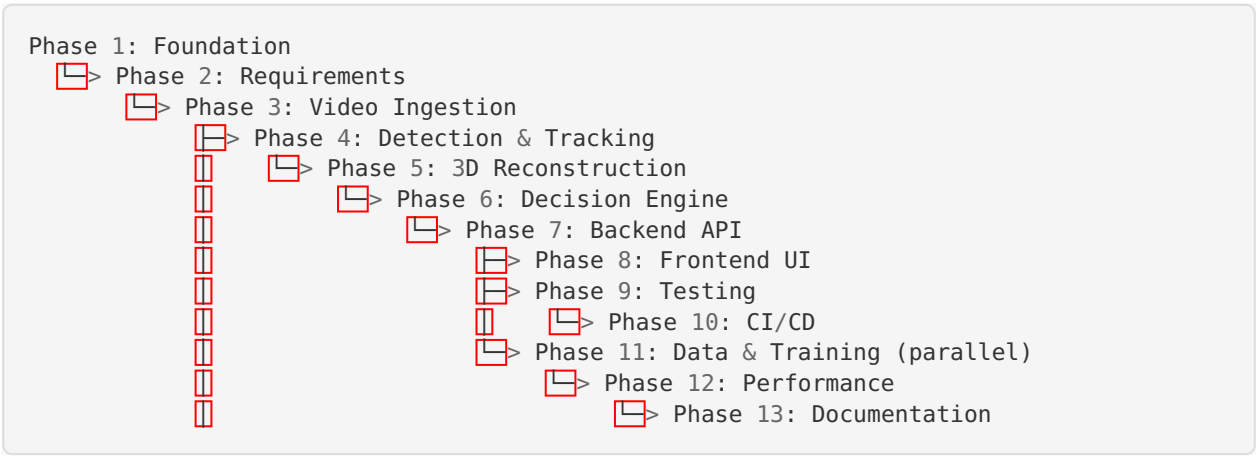
Success Criteria

- All documentation reviewed and polished
- Demo video < 5 minutes, engaging
- Non-technical stakeholders understand the system
- One-pager ready for distribution
- API docs auto-generated and current

Dependencies

- Complete system (all previous phases)

Summary of Phase Dependencies



Parallelizable Phases:
- Phase 11 (Data & Training) can run in parallel with Phases 7-9
- Phase 8 (Frontend) can partially overlap with Phase 7 (Backend API)

Risk Management

Phase	Risk	Mitigation
4	Ball detection too inaccurate	Use ensemble models, multiple cameras, iterative improvement
5	3D reconstruction errors	Validate with synthetic data, provide confidence scores
6	Physics model too simplistic	Design for extensibility, Phase 2 will improve
7	Integration complexity	Incremental integration, thorough testing
8	UI too complex for TMOs	Early user testing, iterative design
11	Lack of real match footage	Start with synthetic, partner with rugby unions
12	Latency targets not met	Early profiling, GPU acceleration, frame subsampling

---

Success Metrics by Phase

Phase 1-3 (Foundation)

- All scaffolding in place
- Documentation complete
- Team aligned on standards

Phase 4-6 (ML/CV Pipeline)

- Detection accuracy: 80%+ (players), 60%+ (ball)
- 3D accuracy: ±0.5m
- Decision accuracy: 85%+ on test cases

Phase 7-9 (Integration & Testing)

- API functional
- End-to-end tests pass
- Code coverage > 80%

Phase 10-13 (Production Readiness)

- CI/CD operational
- Latency < 5 seconds

- Documentation complete
- Demo ready

Future Work (Beyond Phase 13)

Phase 3 Enhancements (Fully Integrated Live System)

- Sub-3 second latency
- TMO hardware integration (custom panels, buttons)
- Broadcast graphics overlay
- Advanced metric tensor physics model
- Player identification/jersey numbers
- Multi-pass analysis in single clip
- Offside detection
- Knock-on detection
- Historical analytics and trends

Commercial Features

- SaaS deployment for multiple clients
- Multi-tenant architecture
- Subscription/licensing model
- White-label UI options
- Integration with broadcast systems
- Mobile app for coaches

Conclusion

This plan provides a clear roadmap from POC to production for Rugby Vision. Each phase is achievable with defined deliverables and success criteria. The modular design allows for flexibility, and the iterative approach ensures we can adapt based on learnings from each phase.

Current Status: Phases 1-3 COMPLETE. Ready to proceed with Phase 4 (Detection & Tracking).

Next Steps:
1. Review and approve this plan
2. Allocate resources for Phase 4
3. Set up project tracking (Jira, GitHub Projects, etc.)
4. Begin Phase 4 implementation

Total Project Duration: 6-8 months to production-ready system.