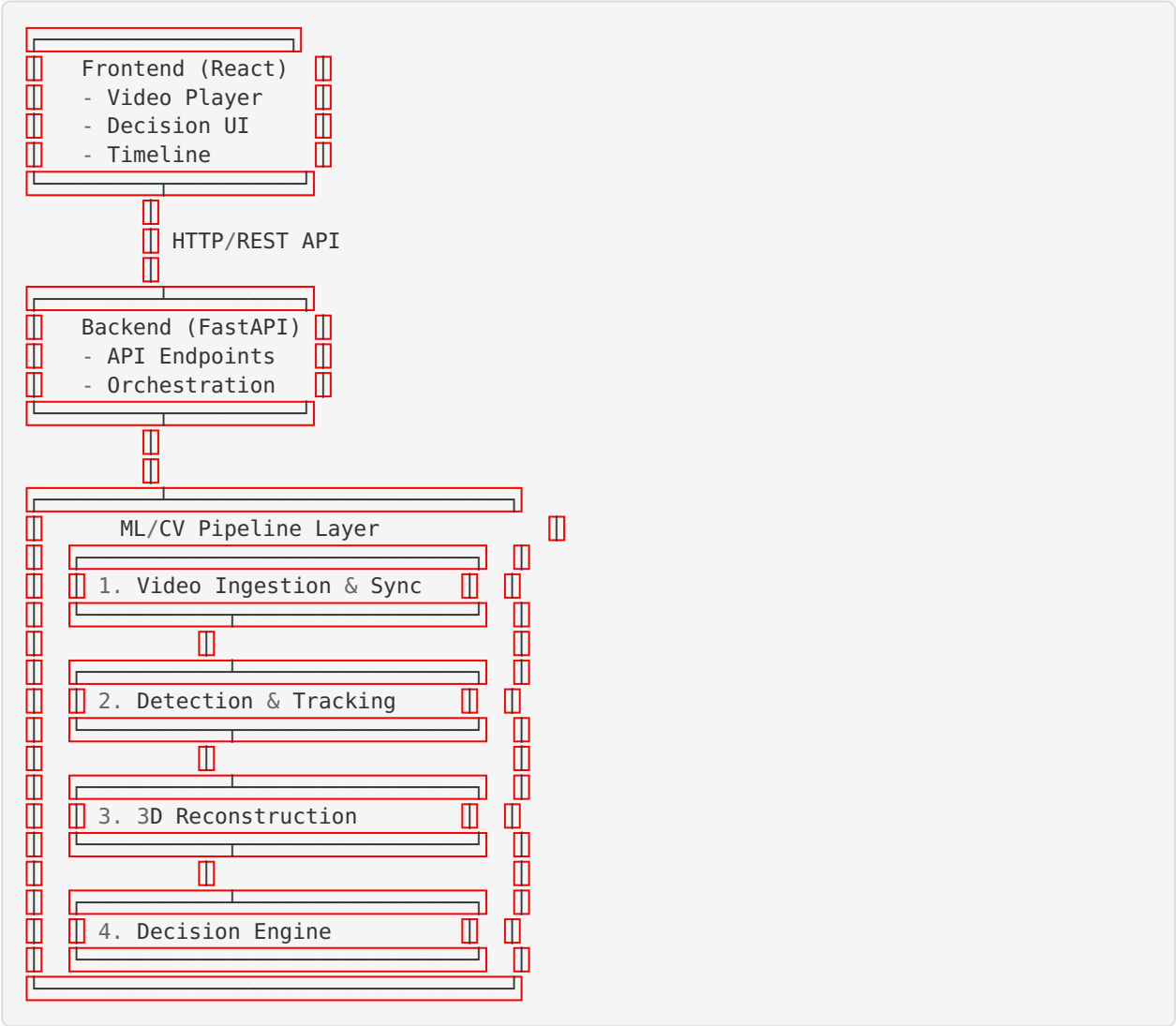
Rugby Vision - Architecture Overview

System Purpose

Rugby Vision is a multi-camera 3D forward pass detection system designed for Television Match Officials (TMOs), referees, and analysts. The system processes synchronized video from multiple camera angles to reconstruct 3D positions of players and the ball, then determines whether a pass is forward according to rugby laws.

High-Level Architecture



Technology Stack

Frontend

- **Framework**: React 18+ with TypeScript
- **Build Tool**: Vite
- **Key Libraries**:

- React Router for navigation
- Axios for API communication
- CSS Modules for styling

Backend

- **Framework**: FastAPI (Python 3.11+)
- **Server**: Uvicorn with ASGI
- **Validation**: Pydantic v2
- **Key Features**:
- Type-safe API definitions
- Automatic OpenAPI documentation
- CORS middleware for frontend integration

ML/CV Layer

- **Core Libraries**:
- OpenCV for video processing
- NumPy for numerical operations
- **Future Extensions**:
- PyTorch/TensorFlow for advanced detection models
- YOLO variants for player/ball detection
- DeepSORT/ByteTrack for object tracking

Infrastructure

- **Containerization**: Docker + Docker Compose
- **CI/CD**: GitHub Actions
- **Deployment**: Cloud-agnostic (AWS/GCP/Azure ready)

Data Flow

Primary Workflow: Analyse Pass

1. **User Interaction (Frontend)**
   - User loads clip with multiple camera angles
   - Selects pass event or time window
   - Clicks “Analyse Pass” button

2. **API Request (Frontend → Backend)**
   ```typescript
   POST /api/clip/analyse-pass
   {
     clip_id: string,
     cameras: string[],
     start_time: number,
     end_time: number
   }
   ```

3. **Backend Orchestration**
   - Validates request (guard clauses)
   - Triggers pipeline stages sequentially:

4. **Stage 1: Video Ingestion & Synchronization**
   - Load video streams from multiple cameras
   - Synchronize frames using timestamps
   - Normalize frame rates and resolutions
   - Output: Synchronized frame batches

5. **Stage 2: Detection & Tracking**
   - Detect players and ball in each frame per camera
   - Track objects across frames (maintain IDs)
   - Output: Per-frame detections with bounding boxes and confidence

6. **Stage 3: 3D Reconstruction**
   - Apply camera calibration parameters
   - Triangulate 3D positions from 2D detections
   - Transform to field coordinate system
   - Output: 3D positions of ball and players over time

7. **Stage 4: Decision Engine**
   - Identify pass event (passer, receiver, timestamps)
   - Compute ball trajectory and velocities
   - Apply physics-based criteria:
     - Ball displacement relative to field
     - Player momentum considerations
     - Output: Boolean decision + confidence + explanation

8. **API Response (Backend → Frontend)**

```typescript
  {
    is_forward: boolean,
    confidence: number,
    explanation: string,
    metadata: object
  }
```

9. **Display Results (Frontend)**
   - Show decision indicator (FORWARD / NOT FORWARD)
   - Display confidence percentage
   - Render explanation text
   - Optional: Visualize 3D trajectory on field overlay

Key Components

Backend Components

`/backend/main.py`

- FastAPI application entry point
- API endpoint definitions
- Request/response models
- Orchestration logic

`/backend/video_ingest.py`

- `VideoIngestor` class
- Handles multiple video source loading
- Video file/stream management

`/backend/video_sync.py`

- `VideoSynchronizer` class
- Frame alignment across cameras
- Timestamp synchronization
- Missing frame handling

ML Components

`/ml/detector.py` (Future)

- Player and ball detection
- Model loading and inference
- Detection data structures

`/ml/tracker.py` (Future)

- Object tracking across frames
- Track ID management
- Association algorithms

`/ml/calibration.py` (Future)

- Camera intrinsic/extrinsic parameters
- Calibration data loading

`/ml/triangulation.py` (Future)

- Multi-view 3D reconstruction
- Point triangulation algorithms

`/ml/field_coords.py` (Future)

- Field coordinate system transforms
- Spatial reference frame management

`/ml/decision_engine.py` (Future)

- Forward pass physics model
- Decision logic and confidence scoring

Frontend Components

`/frontend/src/App.tsx`

- Main application component
- State management
- API integration

`/frontend/src/components/` (Future)

- `VideoPlayer`: Multi-camera video display
- `DecisionIndicator`: Forward/not forward status
- `Timeline`: Pass event markers
- `FieldView`: 2D top-down visualization

Design Principles

1. Guard Clauses First

- Validate inputs early
- Fail fast with clear error messages
- Avoid deep nesting in validation logic

2. Explicit Types Everywhere

- TypeScript: No `any` types
- Python: Full type hints with mypy strict mode
- Pydantic models for API contracts

3. Minimal Nesting (Max 2 Levels)

- Use guard clauses to exit early
- Extract complex logic into named functions
- Avoid `else` blocks when possible

4. Compact, Readable Functions

- Single responsibility principle
- Clear function names
- Maximum 50 lines per function (guideline)

5. Error Handling

- Structured error types
- Meaningful error messages for users
- Detailed logging for debugging

Phase Boundaries

Phase 1: Offline POC (Current)

- Recorded video clips only
- Mock/synthetic data for testing
- Manual pass event selection
- Target: Sub-10 second analysis latency
- Focus: Prove core concept and architecture

Phase 2: Semi-Real-Time

- Near-live stream processing
- Small delay (5-15 seconds)
- Automatic pass detection
- Performance optimizations
- Enhanced accuracy

Phase 3: Fully Integrated

- Live match integration
- Minimal latency (<3 seconds)
- TMO workflow integration

- Production-ready reliability
- Advanced visualization

Scalability Considerations

Horizontal Scaling

- Stateless backend design
- Each clip analysis is independent
- Can distribute across multiple workers

Performance Bottlenecks

- Video decoding (I/O bound)
- Detection inference (GPU bound)
- 3D reconstruction (CPU bound)

Optimization Strategies

- Frame subsampling (process every Nth frame)
- Model quantization (faster inference)
- Parallel processing per camera
- Caching calibration data

Security & Privacy

Data Protection

- No storage of video content (process and discard)
- Temporary file cleanup
- Secure API authentication (future)

Access Control

- Role-based permissions (future)
- Audit logging of decisions

Monitoring & Observability

Metrics to Track

- Analysis latency per stage
- Success/error rates
- Model inference times
- API response times

Logging

- Structured JSON logs
- Trace IDs for request tracking
- Error stack traces

Future Extensions

Advanced Features

- Metric tensor-based physics model
- Multi-pass analysis in single clip
- Player identification/jersey numbers
- Offside detection
- Knock-on detection

Integration Possibilities

- Broadcast overlay graphics
- Referee communication systems
- Statistical analysis platforms
- Training and coaching tools

References

- See `RUGBY_VISION_REQUIREMENTS.md` for detailed requirements
- See `CONTRIBUTING.md` for coding standards
- See `VIDEO_INGEST_DESIGN.md` for video processing details
- See `PLAN_RUGBY_VISION.md` for full project roadmap