ChatGPT

Camera-Based Race Timing System Overview

A camera-based timing system uses video to detect finish-line crossings and identify athletes by their bib numbers. It must capture participants in varying outdoor light and weather, timestamp their finish to ~0.01s precision, and scale to arbitrarily many racers. This report surveys **hardware** (cameras, lighting, computers, triggers), **software** (detection, OCR, timestamping), and existing solutions, and proposes an end-to-end architecture. We focus on approaches compatible with DIY or open-source implementation, while also noting professional systems for context.

1. Hardware Options

- **Cameras:** A key choice is frame rate vs. resolution. High-speed photofinish cameras (e.g. FinishLynx's EtherLynx) capture 1,000–40,000 fps via *strip* imaging [1], achieving 1 ms timing. These are expensive. Consumer alternatives include:
- **Industrial/Academic:** GigE or USB3 cameras (e.g. Basler, FLIR) can capture 60–240 fps at megapixel resolution. For instance, ALGE's IDCam captures up to 60 fps at 5 MP [2]. These can be triggered by a finish sensor.
- **Action Cameras/Smartphones:** Many phones (particularly iPhones) support 120–240 fps video [3]. At 240 fps, timing error can be ~20–30 ms [4]. GoPro/HERO cameras also offer high-speed modes (120–240 fps at 1080p). These are low-cost but require manual triggering or continuous recording and later processing.
- **DIY Boards:** Raspberry Pi cameras or ESP32-Cam modules are low-cost (e.g. 30–90 fps) and can run on battery, but typically have limited frame rates and low-light sensitivity. They may need auxiliary illumination. The open-source Tachikoma87 project, for example, uses ESP32-Cam boards as "photoelectric" barriers [5].

In practice, 60–120 fps is a minimum to approach 0.01–0.02s resolution (each frame ~8–17 ms), higher (200–240 fps) for sub-10 ms. High shutter speeds or global shutters reduce motion blur. Weatherproof housings and lens hoods protect against rain/sun.

- **Lighting:** Outdoor lighting varies from bright sun to dusk. To ensure clear bib capture:
- **Ambient Light:** Position cameras to avoid facing the sun. Use fast lenses (large aperture) and high ISO if needed.
- **Fill Lighting:** For low light, high-power LED floodlights (24–36 V) can illuminate the finish line. Infrared (IR) LEDs/LED bars coupled with IR-sensitive cameras can work invisibly even at night. Tachikoma's design uses IR emitters and a synchronized camera to detect beam breaks [5].

- **Sun Glare:** Polarizing filters or narrow FOV lenses can reduce glare.

- **Computing:** Video processing can be done on:

- **Embedded devices:** A GPU-enabled board (e.g. NVIDIA Jetson Nano/Xavier) can run neural networks (YOLO, OCR) on-site. Pi/Arduino can handle simple triggers but might struggle with full video OCR.

- **Laptops/PCs:** A laptop (e.g. Core i7) with a GPU can capture high-FPS video (e.g. via USB3 or capture card) and process it. US Sports Timing suggests an i7 with 8 GB RAM for their 60 fps Capture system [6].
- **Cloud/Server:** If network allows, video can be streamed for remote processing, though latency and bandwidth are concerns for high-FPS.

- **Triggering Mechanisms:** To timestamp accurately, cameras often rely on sensors to mark crossing instants:

- **Photoelectric Beams:** A laser or IR beam across the finish line can trigger frame capture. Tachikoma's ESP32-Cam barrier is essentially a camera-based beam break [5].
- **Pressure/Mat Sensors:** A physical mat or pad at the line can trigger cameras when stepped on. These are simple but can fail if mis-stepped.
- **Acoustic or RFID Start Sync:** A sound (gun) or an RFID/chip-lap crossing at start can sync clocks (if start isn't a camera event).
- **Software Triggers:** Continuous video can be post-processed to detect motion crossing a virtual line. This avoids extra hardware but can be less precise and requires analysis.

Table 1. Camera hardware comparison (illustrative, prices indicative)

Camera Type	Max FPS (approx)	Resolution	Notes	Example/Ref.
FinishLynx (EtherLynx)	up to 40,000 (line-scan)	N/A (slit image)	1 ms accuracy; **very** expensive (~$10k+) [1]	FinishLynx [1]
ALGE IDCam	60 (burst)	2592×1944 (5 MP)	Ethernet trigger; pairs with photocell [2]	ALGE IDCam [2]
DSLR/Video	30–60 (cont.)	~1080p or 4K	Sharp but limited FPS; good daylight, heavy	–
GoPro / Action	120–240 (1080p)	1080p–4K	Waterproof; mobile; battery power; sliding scale FPS	–
Smartphone	30–240 (device-dependent)	~720p–1080p	iOS: up to 240fps [3], Android often 30fps; cheap	MySprint App study [4]
Industrial Cam	60–240 (global shutter)	up to 4K or higher	High sensitivity, low-latency; PoE or USB3	–
Raspberry Pi Cam	~30–90	1080p	Inexpensive; flexible; may need light	Tachikoma (ESP32) [5]

2. Software Components

A complete system typically involves:

- **Video Capture & Timing:** Acquire frames with known timestamps. High-speed cameras often embed timecode, or the capture software can note system time on each frame. It's critical to synchronize the camera clock with the official race clock. Consumer cameras may not expose exact frame times, so many systems use the frame index (FPS) to infer time (e.g. frame N at N×(1/FPS) after start trigger). For sub-0.01s accuracy, a hardware timestamp (e.g. from a trigger) is preferable.

- **Event Detection:** (Optional) Determine when runners cross the line. This can be done via:

- **Hardware triggers:** As above, a sensor triggers the capture or marks frames (e.g. photocell starts high-speed capture for a brief period).

- **Motion Detection:** In software, detect when any runner's motion intersects a designated finish line in the image. OpenCV can compute frame differences or use background subtraction to flag motion across a line region.

- **Athlete/Bib Localization:** Once a crossing event is identified, the system should find each athlete's bib in the frame. Approaches include:

- **Person/Bib Detection Models:** Use a neural network (e.g. YOLOv5/YOLOv8) trained to detect "person" and/or "bib" regions. For example, one open project trained YOLO to find bib patches on runners [7]. These detectors can quickly localize each bib among multiple finishers.
- **Classical Methods:** Face or pose detectors can hypothesize bib location (e.g. below the chin) and then apply a text detection (Stroke Width Transform) to find numbers [8] [9]. Several academic systems use SWT or contour analysis to isolate bib text [8] [9].

- **Segmentation:** A Mask R-CNN could segment the bib area if labeled data is available.

- **OCR/Text Recognition:** Once a bib region is cropped, apply OCR to read the number:

- **General OCR Engines:** Tesseract (open-source) or EasyOCR can recognize printed digits. However, they may struggle with varied fonts or occlusions. Preprocessing (grayscale, thresholding, despeckle) can improve results [9].
- **Custom Digit Models:** Training a CNN (e.g. on the SVHN street-number dataset) to recognize multi-digit bibs, as in the bib-detector project [10], can yield higher accuracy. The project by Bayless et al. used YOLO for digit boxes and SVHN-trained CNN for recognition. Similarly, Kapil Varshney's marathon project found that object-detecting individual digits and then classifying them worked well [11].

- **End-to-End:** A sequence OCR like CRNN or PaddleOCR can read the number string directly from the bib image, avoiding separate detection.

- **Timestamping & Logging:** For each detected bib number, record the frame's timestamp as the finish time. Store results in a database or CSV. Often, the first frame where the bib crosses the line is

the official finish moment. The system should log “[Bib, Time]” entries, and flag any ambiguities (e.g. unreadable bib). A simple architecture flow is:

- **Capture Frame** (time `t`)
- **Detect Movers:** Is an athlete crossing the finish-line region? If not, skip.
- **For each athlete in frame:** crop bib image.
- **OCR Bib Number:** get digits (or blank if unreadable).
- **Record (bib, t).**

- **User Interface:** A GUI or command-line tool can review detections. The US Sports system, for example, lets officials “review video by bib number” [6]. Kinovea (open-source) provides manual/automatic frame-by-frame analysis of sports video [12], but has no built-in OCR. A custom app could display the bounding box and recognized number for confirmation.

- **Error Handling:** Cases of tied finishes or unreadable bibs may need human override. The system might flag low-confidence OCR or overlapping athletes for manual review.

3. Libraries and SDKs

A DIY system can leverage many mature libraries:

Component	Example Libraries/Tools	Notes/Citations
Image I/O	OpenCV (VideoCapture, GStreamer), libuvc	Capture frames from USB/PCI cameras
Object Detection	YOLO (Darknet, ultralytics YOLOv5/8), TensorFlow Object Detection, MobileNet-SSD, MediaPipe	YOLO has pretrained models; YOLOv4/8 detect bibs [7]
Text Detection	OpenCV (SWT, EAST), PaddleOCR	SWT + Tesseract used in bibnumber [9]; EAST text detector mention [13]
OCR	Tesseract OCR, EasyOCR, PaddleOCR, Amazon Textract (API), Google Vision (API)	Tesseract/EasyOCR open-source; Google/AWS are commercial.
Machine Learning	PyTorch, TensorFlow, Keras, ONNX Runtime, TensorRT	For training/inference on models (digits, detection).
Data Handling	Python (pandas, sqlite3), InfluxDB, CSV logs	Store and query results.
Trigger I/O	Arduino IDE (for IR beams), ESP-IDF (ESP32), GPIO libraries (RPi)	Interface sensors or LED triggers.
Time Sync	NTP/PTP protocols, GPS PPS	Synchronize camera/computer clock.

Component	Example Libraries/Tools	Notes/Citations
Misc Hardware	libgphoto2 (DSLR), Pylon SDK (Basler), Spinnaker (FLIR)	Camera manufacturer SDKs (often LGPL or commercial).

Open-source projects also offer ready frameworks. For example, Lwhieldon's BibObjectDetection uses YOLOv4 (cuDNN) for bib detection [7]. The bibnumber C++ project uses OpenCV+Tesseract pipeline [9]. The Tachikoma project provides ESP32 firmware and Android app for timing barriers [5].

Commercial SDKs:
- **Vision APIs:** Google Vision and AWS Rekognition can read text in images but require internet and cost per use.
- **Sport Timing SDKs:** Vendors like ChronoTrack or FlashTiming do not generally publish APIs for DIY.
- **Pro Camera SDKs:** Basler Pylon, FLIR Spinnaker allow control of industrial cameras (temperature, timestamp features).

4. System Architecture Overview

A typical workflow is shown below (illustrative):

1. Camera Capture → 2. Frame Processing → 3. Bib/Number Detection → 4. OCR & Timestamp → 5. Result Logging

1. **Camera Capture:** Video streams in real time. Each frame `F_i` has an associated timestamp `t_i` (either from the camera or system clock). If a hardware trigger (e.g. IR beam) signals, the system notes that trigger time.
2. **Frame Processing:** For each new frame (or triggered block of frames), optionally run motion detection. If no one is near the line, skip to save computation.
3. **Bib Detection:** Run an object detection model on the frame to find all bib regions (e.g. bounding boxes around numbers) [7]. Alternatively, detect humans and then localize bib within each person (e.g. below torso).
4. **OCR & Timestamp:** For each detected bib box: crop the region and run OCR. If the text passes confidence checks, output `[bib_number, t_i]`. If the athlete is mid-run and not yet at the line, hold the result until final crossing frame.
5. **Result Logging:** Store finishes in a table/file. Optionally, match to pre-registered participant list. For simultaneous finishes (multiple detections in one frame), all get the same timestamp. Any ambiguous cases can be flagged.

This modular flow can be parallelized: e.g. multiple CPU threads or using the GPU. The pipeline needs to be fast enough to keep up with frame rate; lower-priority steps (like saving images) can run asynchronously.

5. Existing Systems and Case Studies

- **FinishLynx (Videtic):** Industry-standard photo-finish. Uses a *line-scan* camera at the finish line, capturing a 1D "strip" image at up to 40,000 fps [1]. It automatically determines finish order and

times to 0.001 s. An optional *IdentiLynx* camera provides a front-facing high-res video (30–100 fps) for visual bib verification [14]. FinishLynx is proprietary and expensive (~$10k+).

- **ALGE IDCam:** A 2D camera triggered by a finish sensor. It captures bursts (up to 60 fps) of full-frame images when the timing photocell fires [2]. Each photo is time-stamped and can be reviewed to correct or verify chip times. No automated OCR is provided – an operator checks the images.
- **US Sports Capture Camera:** A 60 fps video camera system sold for $900 [6]. It integrates with chip-timing; after the race, officials review the video by bib number. (This suggests OCR can index videos.) It notes that 30 fps suffices for verification, 60 fps recommended for finishes [6].
- **Smartphone Apps:** The "MySprint" iOS app uses the device's 240 fps camera to time sprints. A study found its timing was accurate to ~28 ms compared to photocells [4]. Other apps like "SprintTimer" (iOS) use strip-photography from 120–240 fps video. Most such apps lack automatic bib reading, requiring manual marking.
- **OpenPhotoFinish (2024):** An open-source workflow where a race is recorded on video, then processed offline into a photo-finish image. Operators mark start/finish times on the composite image to get results [15] [16]. This demonstrates using ordinary cameras (even a phone on a tripod) to time races, though it is manual.
- **Tachikoma/CBPhotoelectricBarrier (2023):** A research project (Chemnitz Univ.) that built *camera-based photoelectric barriers* using ESP32-Cam modules [5]. The open-source design (MIT license) includes schematics and code. Tested on 30 m flying sprints, it achieved ~62 ms accuracy against professional IR gates [17]. All hardware (WiFi module, IR LED, smartphone app) is consumer-grade. This shows a DIY video-timing approach can rival pros.
- **Academic Bib Recognition:** MIT's RBNR system combined face detection and stroke-width transform to locate bib text, then OCR [8]. Eric Bayless's "bib-detector" (GitHub) used a YOLO bib detector and an SVHN-trained CNN to read digits [10], aimed at ordering high school cross-country results. Lwhieldon's BibObjectDetection (MIT license) trained YOLOv4 on public datasets, achieving ~99% mAP on bib detection [7]. These examples illustrate feasibility of open bib-recognition tools.
- **Commercial Photo-Tagging Services:** Companies like Tagily or ScoutFoto use proprietary AI to tag race photos by bib number [18]. While not timing systems per se, they validate that automated bib OCR on action images is practical at scale.
- **RFID Chips (for context):** While outside our camera-only scope, many races use RFID chips on bibs or shoes. These systems are mature and accurate (~ms), but require issuing chip tags. Video-timing aims to avoid this hardware.

6. Reliability, Scalability, and Accuracy Considerations

- **Lighting & Weather:** Extreme sunlight or glare can blow out images; use shading or HDR settings. Rain/snow may obscure lens – a waterproof housing with windshield wiper or hydrophobic coating helps. Night races need strong lighting or IR setups. Changing shadows (clouds) may confuse motion detection, so parameter tuning or adaptive thresholds are needed.
- **Occlusion & Overlap:** In mass finishes, bibs may be partially hidden by other runners or arms. Multi-camera setups (from the front and/or side) can capture different angles. Overlapping athletes with the same frame timestamp may tie; the system should either report a tie or allow an official to review frame-by-frame (high FPS can reveal micro-gaps).
- **Timing Precision:** Achieving true 0.001 s precision is challenging without specialized line-scan cameras. At 240 fps, each frame is ~4.17 ms apart; interpolating between frames is uncertain. Most consumer solutions can promise ~0.01–0.02 s at best [4]. Professional track rules accept 0.01 s timing, but Olympic photo-finish uses 1 ms. Decide if 1 ms is required or 10 ms suffices.

- **System Load & Scalability:** Unlimited participants implies potentially hundreds finishing per minute. The system must process each frame quickly. Using a GPU for detection/OCR and batching frames will scale better. If too slow, one could drop to triggering only around race end. For very large events, multiple identical setups (lanes) could share load and then merge results.
- **Resilience:** Use redundant storage (save video locally and backup) and battery/UPS backups. A backup timing (e.g. manual stopwatch or RFID mat) is wise in case of failure. Open-source code should be thoroughly tested on practice runs. Monitoring logs/health in real-time helps catch issues early.
- **Data Accuracy:** Pre-load expected bib numbers (from registration) so the OCR can validate against known range. Implement checksum or character validation (digits only, certain fonts) to reduce misreads [9]. Flag improbable readings (e.g. "BIA" instead of "814") for manual correction.
- **Regulatory Compliance:** If the times are for official results, the system should have timestamp synchronization (e.g. NTP or GPS) and ideally meet timing standards (USATF Rule 163 for auto timing). Provide audit logs (video and data) in case of disputes.

Overall, a DIY camera timing system can be **accurate to ~0.01–0.03 s** under good conditions, as demonstrated by smartphone studies [4]. While it may not match pro photo-finish in every respect, it offers a low-cost, flexible solution for many outdoor races. Combining proven hardware (60–240 fps cameras) with robust CV libraries (OpenCV, YOLO, Tesseract) and careful engineering (lighting, triggers, fail-safes) can yield reliable results [5] [19]. The references below provide further technical details and examples.

References: We cite key resources throughout, including product datasheets and research papers [1] [2] [6] [8] [10] [13] [7] [9] [19] [4]. Each provides insight into specific hardware, software, or real-world trials in camera-based timing.

---

1 **Sports Timing Systems & Photo-Finish Camera Packages - FinishLynx**
https://finishlynx.com/packages/

2 **ALGE-TIMING - IDCAM**
https://alge-timing.com/en/product/2311/idcam

3 4 12 17 19 **A Novel Inexpensive Camera-Based Photoelectric Barrier System for Accurate Flying Sprint Time Measurement**
https://www.mdpi.com/1424-8220/23/17/7339

5 **GitHub - Tachikoma87/CBPhotoelectricBarrier: An implementation of camera-based photoelectric barriers with ESP32-Cam modules.**
https://github.com/Tachikoma87/CBPhotoelectricBarrier

6 14 **Capture Camera System | US SPORTS TIMING**
https://www.ussportstiming.com/product-page/capture-camera-system

7 **GitHub - Lwhieldon/BibObjectDetection: Using OpenCV, CUDA, & YOLOv4 to detect numbers on racing bibs found in natural image scene**
https://github.com/Lwhieldon/BibObjectDetection

8 **RBNR - Racing Bib Number Recognition**
https://people.csail.mit.edu/talidekel/RBNR.html

9 **GitHub - gheinrich/bibnumber: Recognize bib numbers from racing photos**
https://github.com/gheinrich/bibnumber

10 **GitHub - ericBayless/bib-detector: Object detection project to identify race bib numbers for order of finish**
https://github.com/ericBayless/bib-detector

11 13 **Marathon Bib Identification and Recognition | by Kapil Varshney | TDS Archive | Medium**
https://medium.com/data-science/marathon-bib-identification-and-recognition-25ee7e08d118

15 16 **On The Track**
https://openphotofinish.blogspot.com/2024/09/the-race-has-finished-and-you-have.html

18 **ScoutFoto - Find your perfect race photo**
https://find.scoutfoto.com/