An IoT-based system of mechanizing sport competition motion for perception improvement

Vadim V. Romanuke, Serhii Y. Dementiev and Svitlana A. Yaremko

Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, 87 Soborna Str., Vinnytsia, 21050, Ukraine

Abstract

Human perception heavily relies as on seeing and hearing, as well as on possibility to view different angles and continuity of motion. Obviously, modern video broadcasting and displaying do ensure high-resolution content, but they nonetheless still lack that subtlety of perceiving objects live, by which one is still capable to distinguish watching screen picture from watching natural picture. In sport competitions, events are perceived much better when they are observed up close, with stronger capability of estimating dynamics and general mechanics of motion. This concerns the coaching staff and analysts, as well as fans and spectators. A special case is the use of scaled-down or miniaturized models similar to tabletop games, where on-field sports players are represented by model figures. Such models, additionally allowing tangibility, are especially precious for people with visual impairments. We suggest a conceptual model of an IoT-based embedded system, which allows visually, tangibly, and acoustically perceiving location and motion of a sports player within sports field. We consider three approaches to technical implementation of motion mechanism for providing speed and smoothness. Motion is controlled by stepper motors and specialized drivers TMC22xx/DRV88xx. Acceleration and deceleration are computed by a special pattern. An algorithm is developed for automatic calibration of the system mechanics, and also for automatic positioning. Motion control, data acquisition from player-mounted sensors, and data exchange among servers are provided by Raspberry Pi Compute Module 5. We optimize internal protocols for data exchange among processes, as well as external protocols for real-time data acquisition, based on neglecting states that have relatively low effect on the overall model response. This renders our system power consumption to almost linear with respect to stepper motors, which are the most power-consuming in the system.

Keywords

wearable sensors, real-time data acquisition, mechanical presentation of information, perception improvement, stepper motor, TMC2209, Raspberry Pi Compute Module, Python server, UART, calibration

1. Introduction

Nowadays, huge variety of portable displays, tablets, electronic boards, and other devices for visual reproduction of information allows covering nearly any household, industrial, and entertainment tasks. However, almost all of these tools ultimately produce a 2D image on the screen plane, which a person “reads” with one’s eyes and eventually transforms into an image or object as a factual perception of information. Techniques of perceptual augmentation to enhance perception in vision are well-known. Thus, the contrast between the white background and black characters is used to highlight the text, certain color palettes are used to enhance part of the image, special dynamic video sequences are generated, which actually allow enhancing perceptibility of 3D scenes and objects on 2D screen, etc. Meanwhile, large manufacturers of electronic visual displays continue to develop. There are already many concepts of flexible displays [1], curved screens [2], 3D LED displays [3], 3D hologram spinning fan LED displays [4], and others, but reproduction of information by mechanical pointers will always be relevant also. The matter is human perception heavily relies as on seeing and hearing, as well as on possibility to view different angles and continuity of motion [5, 6]. Despite modern video broadcasting and displaying unquestionably ensure high-resolution content, mechanical reproduction of a system object (element or parameter) is visible at any side and angle, it is not limited in size and is often much cheaper, more reliable and informative than electronic displaying. A real mechanical object is tangible

doors-2025: 5th Edge Computing Workshop, April 4, 2025, Zhytomyr, Ukraine

romanukevadimv@gmail.com (V. V. Romanuke); s.dementiev@vtei.edu.ua (S. Y. Dementiev); svitlana_yaremko@ukr.net (S. A. Yaremko)

0000-0001-9638-9572 (V. V. Romanuke); 0009-0006-1322-6756 (S. Y. Dementiev); 0000-0002-0605-9324 (S. A. Yaremko)

and even can be felt (perceived) by its nearby vibration. Simple examples are such mechanical means of displaying as a wall clock with hands, a tachometer or a fuel level indicator on the dashboard of a car, a weather vane. Mechanization of information presentation provides that subtle feature of augmented perception that is lacked in watching screen picture, whichever quality it has.

In sport competitions, events are perceived much better when they are observed up close, with stronger capability of estimating dynamics and general mechanics of motion. The coaching staff and analysts can view detailed peculiarities of tactical movements and mistakes of players [7, 8]. Fans and spectators get aesthetic pleasure from watching details of movements, dribbling, struggling, confrontation moments. A special case is the use of scaled-down or miniaturized models similar to tabletop games, where on-field sports players are represented by model figures. Such models, additionally allowing tangibility in addition to audio commentary of the game, are especially precious for people with visual impairments who receive possibility to estimate tactilely location and motion of an individually taken player. Moreover, coaches get miniaturized yet enhanced views of analyzing individual player positioning and tactical adherence [9]. The player can review subtler details of one's performance and improve spatial awareness [10].

Our purpose is to develop a conceptual model of an IoT-based embedded system, which allows visually, tangibly, and acoustically perceiving location and motion of a sports player within sports field. Positioning is to be realized by a moving marker, whose shape and size could be changed according to the type of game or sport. Player's data must be aggregated straightforwardly from sensors via stadium network gateways, but also we ought to make it possible to download data of previous games to simulate performance of a chosen player. The system, which is presumed to be portable and energy efficient, is believed to be a more convenient tool for visual presentation of tactics and strategies in sports games by the team's coaching and teaching staff. Besides, unlike electronic applications, it should allow visually impaired people to better perceive information about events in the game.

2. Sports data collection

Sports data, and, in particular, soccer data, are continuously collected during the game. Data acquisition is the first step in the data processing pipeline. It involves capturing raw data and converting it into a format that can be easily analyzed. Data are acquired either from sensors usually mounted in vests and cleats or by optical tracking systems and event data providers. When players wear enhanced-GPS devices [11], BLE beacons [12], accelerometers, gyroscopes, magnetometers, and other wearable sensors, embedded in vests and cleats, these devices collect location data, speed, and movement patterns through gaming time [13]. Optical tracking systems have cameras placed around the stadium that track the players' movements in real time [14, 15]. Systems like Hawk-Eye [16] or ChyronHego [17] use multiple high-resolution cameras to record player positions at a high frame rate. Event data providers involve technical personnel for collecting sports data. Companies like StatsBomb [18], Opta Sports [19], and Wyscout [20] collect player data during matches using a combination of manual tagging and automated systems.

Once acquired, the raw telemetry data from tracking systems and wearables is preprocessed to determine the player's position on the field and other temporal characteristics at any given moment. This positional data is recorded in the abscissa-and-ordinate-axes coordinates with respect to the field dimensions. Besides, these are sensors inside the ball itself, which record its motion speed and the position in 3D coordinates with respect to the field to identify in-flight characteristics of the ball [16]. To maintain a continuous record of movement, the data is typically sampled at high frequencies (e. g., 25 frames per second or so) [17, 18].

The preprocessed data is used for generation of heatmaps [21]. The field is divided into a grid of small cells, where each cell represents a specific area of the field. The system calculates the time a player spends in each grid cell. More time spent in a cell results in higher intensity for that cell on the heatmap. Other heatmaps show such activity levels as sprints, touches, interceptions, headkicks, etc. Heatmaps use gradient color coding (e. g., blue for low activity, red for high activity) to represent the intensity of a player's presence and activity in different areas. Static heatmaps show the whole match, whereas dynamic heatmaps show gradual changes over time. Nevertheless, heatmaps cannot show player's geometrical twists and turns, which are observable only either on screen or, what is at least no way worse, on a miniaturized field.

3. Structure of the system

All telemetry data is collected on wireless gateways in the stadium and transmitted to a server where the telemetry data is filtered and processed. Most of the game's data is not available to the general public, and it is mostly not of interest to common spectators or fans. But basic information about the game process is delivered to providers of such services, which give the end user the opportunity to receive it, whether for a fee or free. Examples of such telemetry data available on the match broadcast screens and tabulated statistics on the Internet are ball possession percentage, pass accuracy, amount of kilometers covered by the player, maximum impact force, etc.

Data providers deliver data from their servers to users online, or they may provide recordings of previous games. Data formats can be different, but as a rule, they can be obtained in CSV, XML, or JSON and similar formats, which are easy to parse, convert, and transfer for further processing. Recorded games, as well as interesting educational game situations, can be stored on own separate "On demand" server in an optimized format that allows for faster access to data and makes data classification better.

The center of the system is the Raspberry Pi Compute Module (figure 1). This is a powerful mini-computer based on the Linux OS operating system [22], which has powerful peripherals for simultaneously processing input sensors, data exchange with servers, controlling motors, displaying required information on the screen and processing other related information. Raspberry Pi products are the most rational choice today in terms of price-to-quality ratio. With the price of the basic model starting at $35, the Raspberry Pi 5 operation speed is about 25 Gflops [23], which is an excellent indicator. Of course, competitors such as Orange Pi or Banana Pi are not far behind, but the quality and support of their products is much worse.

In our system, the Raspberry Pi uses the following peripherals:

1. Ethernet and Wi-Fi to access the Internet to exchange data with servers, and to be able to configure the system and collect logs.
2. GPIO, UART, I^{2}C, I^{2}S for controlling stepper motor drivers, receiving data from sensors, controlling the vibrator and transmitting sound to user's headphones.
3. Bluetooth as an alternative audio channel for a wireless headset.
4. HDMI or MIPI for outputting the broadcast video signal to an optional screen.

The system runs on Raspberry Pi OS, which is a Unix-like operating system based on the Debian Linux distribution for the Raspberry Pi family of compact single-board computers. Raspberry Pi OS is very stable, allows multithreading, and also supports all popular software development tools like C/C++, Python, Bash script, Java, PHP.

The system converts the received data about the position of the player on the sports field, his speed and current acceleration into a 2D coordinate on the miniaturized field and controls the position of the carriage with the help of two motors and belts. The carriage moves under the miniaturized field with a magnet attached to it. A magnetic marker as the player's model figure is placed on top of the miniaturized field. This marker completely repeats the movement of the carriage. The calibration of the carriage position, as well as the permanent control of its position, is performed by four optical sensors on each side of the miniaturized field, which makes mechanical malfunctions and slips impossible. The optical sensors also help position the marker more accurately, whereas positioning is quite repeatable. The carriage is moved along the conditional abscissa and ordinate axes by two 24 V stepper motors with a maximum peak current of 2 A. The accuracy, smoothness, and other dynamic parameters of the motors are controlled by driver boards TMC22xx, which are controlled from the Raspberry Pi via GPIO and UART interfaces. The system has a vibration motor that generates a vibration signal of a certain

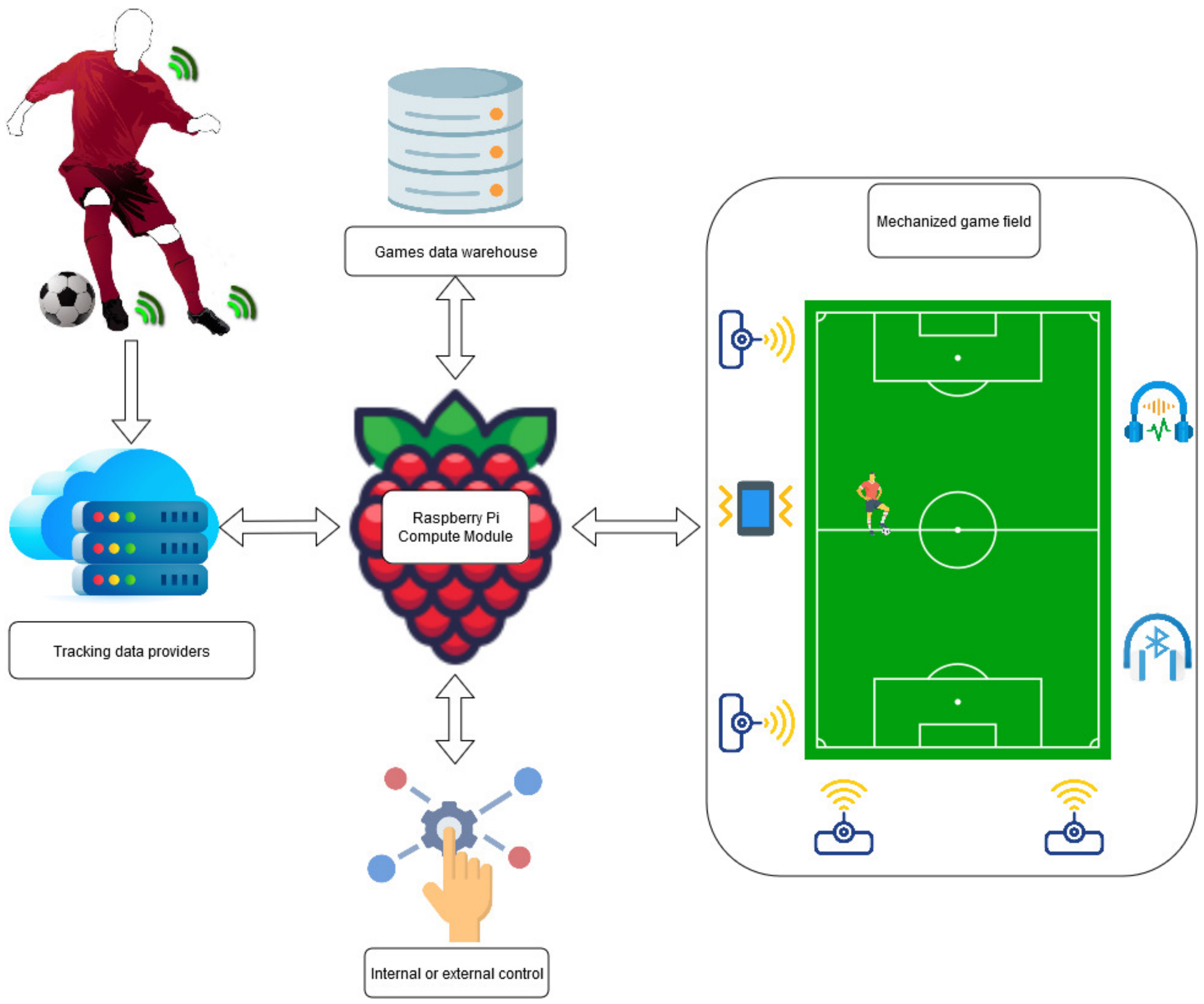

Figure 1: Structure of the IoT-based system of mechanizing sport competition motion for perception improvement, where the computing module is intended to deliver quicker responses to the player on the miniaturized field during real-time game (data acquisition).

amplitude and frequency, by the type of which the user can distinguish one or another event that is currently happening on the field.

Basic (or custom) control of the system is possible using several buttons and switches. Full control of the system is carried out using a command protocol through an internal or external (Internet) connection.

4. Mechanics of the system

The mechanical part of the system consists of motors that move the belts, which in turn move the carriage (figure 2). The carriage moves on linear bearings along linear guides in the abscissa-and-ordinate-axes plane in the direction specified by the logic. Gears and tension devices allow the belt to move smoothly and have a stable tension. All elements are fixed on a strong aluminum frame, which is the base.

The choice of the design is determined by the main requirements: maximum movement speed, simplicity, and maximum reliability of the system. We consider three versions of the mechanical design. Version “A” has the appearance of a classic scheme, which is used in the manufacture of computer-numerical-control machines (figure 3). It has two (cheaper schemes have one) lower motors that move the main beam with two running screws and determine the position of the ordinate axis (Y axis). Another motor is placed on the beam, which moves the carriage along the abscissa axis (X axis) with another running screw. Scheme "A" has such advantages as good positioning and excellent holding power and movement of the carriage, which is relevant for computer-numerical-control machines. However, it is not fast and requires much more additional relatively expensive components than schemes on belts. Belt schemes (figure 2) are often used in the construction of 3D printers of the cheap and middle segments of the market, so there are no problems with components and auxiliary mechanisms.

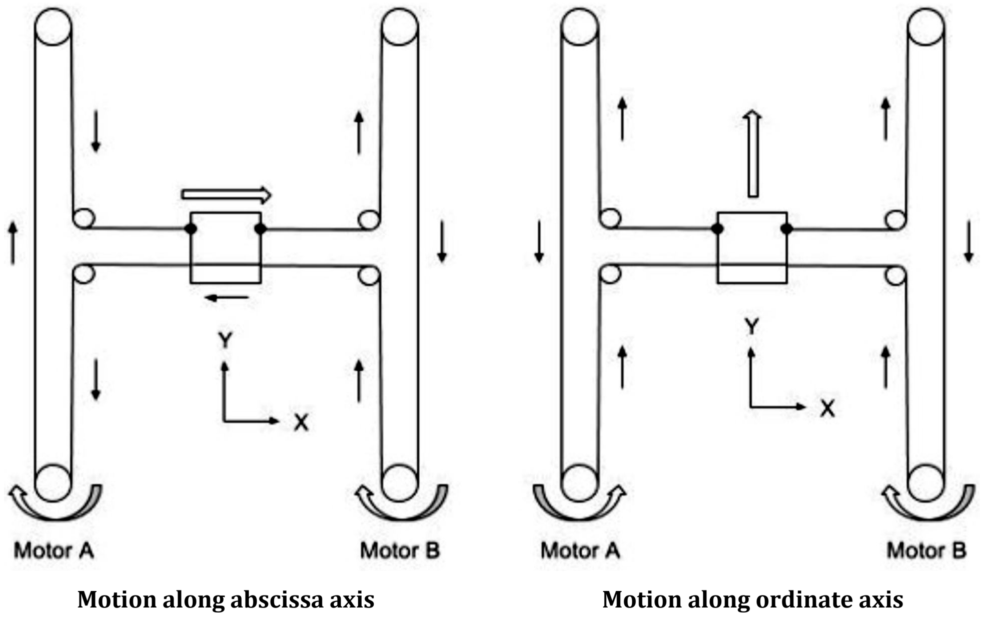

Figure 2: The general structure and components of the system mechanics [24].

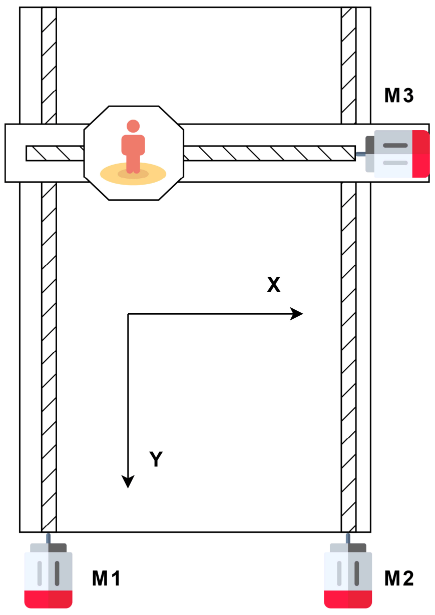

Figure 3: Scheme "A" of the system mechanics with good positioning and excellent holding power and movement of the carriage, but the non-belted scheme is not fast and requires much more additional relatively expensive components.

Scheme "B" is a family of 2-axis mechanisms called H-Bots [25]. They use two motors on one belt to create 2-axis motion (figure 4). The simplest type is the classic H-Bot [26]. Letter "H" comes from the contour of the belts that form this letter. The motors rotate in one direction for movement along one axis, for movement along another axis the motors must be rotated in opposite directions. If only one motor is rotated, both axes will move evenly (diagonally). All other angles can be realized with the corresponding gear ratio.

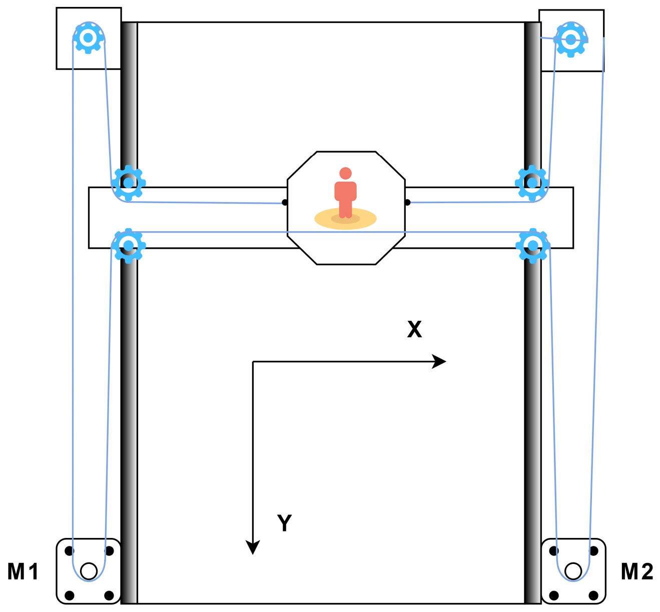

Figure 4: Scheme "B" of the system mechanics, which is simple and reliable under conditions of non-heavy loads.

The H-Bot is simple and very reliable, but adds some torque on the mechanism under heavy loads. For example, when the belt is pulled down from the left side of letter "H" and up from the right side, the force will try to twist the frame if there is too much resistance of the carriage in X. Scheme "C" called CoreXY [27] solves this nuance (figure 5), but it has crossed belts on one side that complicates the construction and lowers its reliability upon intensive usage. In addition, a longer belt increases the likelihood of problems with its stretching and carriage positioning. There are also other belt routing schemes, e. g., T-Bots, but they are less popular.

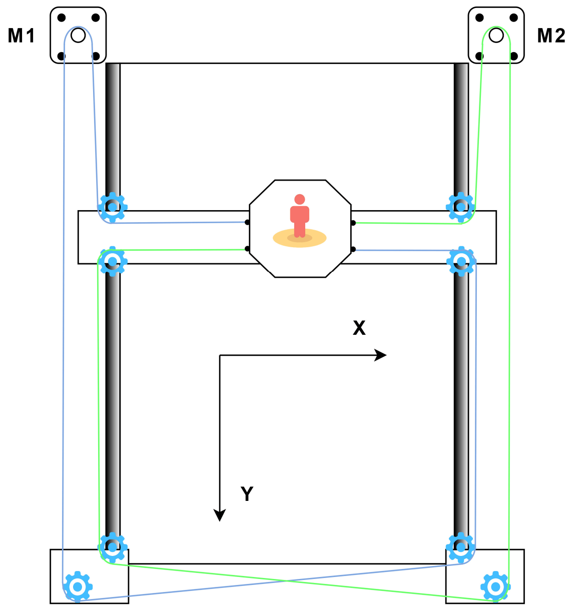

Figure 5: Scheme "C" of the system mechanics with crossed belts complicating the construction and lowering its reliability upon intensive usage.

Therefore, we select the classic H-Bot layout to ensure the maximum speed of movement and reliability of the system under conditions of not applying excessive loads to the carriage, which in principle is not required for this task. The rating of stepper motors is determined based on the NEMA17 standard size with a torque of 3-4 kg/cm at a current of up to 2 A. They are usually used in 3D printers, extruders and small computer-numerical-control machines. For example, the NEMA17 JK42HS40 motor is quite suitable for such tasks by owing to that the motor has acceptable characteristics (table 1).

Table 1
NEMA17 JK42HS40 motor characteristics [28].

Parameter	Value
Typical turning angle per step	1.8°
Shaft diameter	5 mm
Shaft length	24 mm
Motor length	40 mm
Current per winding	1.7 A
Voltage	24 V
Winding resistance	1.65 Ohm
Winding inductance	3.2 mH
Holding torque	4.2 kg/cm
Number of contacts on the connector	4
Weight	280 g

The driver for controlling the stepper motor is selected based on popularity, ease of connection to the Raspberry Pi Compute Module, and energy efficiency. The DRV88xx series of drivers [29] from the Pololu company are considered, in particular the DRV8825 model [30], as well as the TMC2209 model from Trinamic [31]. The latter is eventually used in the system. It differs from other drivers in that it receives information about rotation parameters from back EMF signals and winding currents. The obtained data allow to achieve precise and noiseless control of the engine. In addition, this driver allows a maximum motor current of 2 A and supports fine-tuning parameters via the UART interface.

The PCM5102 module serves for the sound I^2S DAC [32]. It produces high-quality 24-bit sound and is intended for use with Raspberry Pi or Orange Pi mini-computers. This is an inexpensive module, and for its size it has excellent characteristics both in headphones and through an additional power amplifier.

A laser distance sensor on the VL53L0X module serves for calibrating and adjusting the position of the carriage. This is a miniature ToF (Time-of-Flight) distance sensor module that allows quickly and accurately measuring distances up to 2 m. The module is connected via a common I^2C serial interface for device control and data transfer. The VL53L0X sensor is equipped with an advanced matrix based on highly sensitive single-photon avalanche diodes. The principle of its operation is based on STMicroelectronics' patented FlightSense technology [33]. A VCSEL surface-emitting laser with a wavelength of 940 nm acts as an optical signal source in the VL53L0X distance sensor. It is equipped with a built-in infrared filter. Its glow is completely invisible to human eye, and provides a longer measurement distance with less sensitivity to ambient light levels and is more resistant to crosstalk such as glass surfaces. The distance measurement range is from 1 cm to 2 m.

5. Logic of system functioning

After powering on, the analog part of the system is ready to work almost immediately. The Raspberry Pi digital controller loads the operating system and starts its all related services within 10 seconds. The main internal logic of the system is shown in figure 6. If the system is idle without action for 60 seconds, it automatically goes into sleep mode to save power resources, which turns off all loads and puts the controller into a reduced power saving mode. The transition to operating mode occurs when the user presses a button on the panel, or activity on the Ethernet port. There is a light-emitting diode that indicates the current mode.

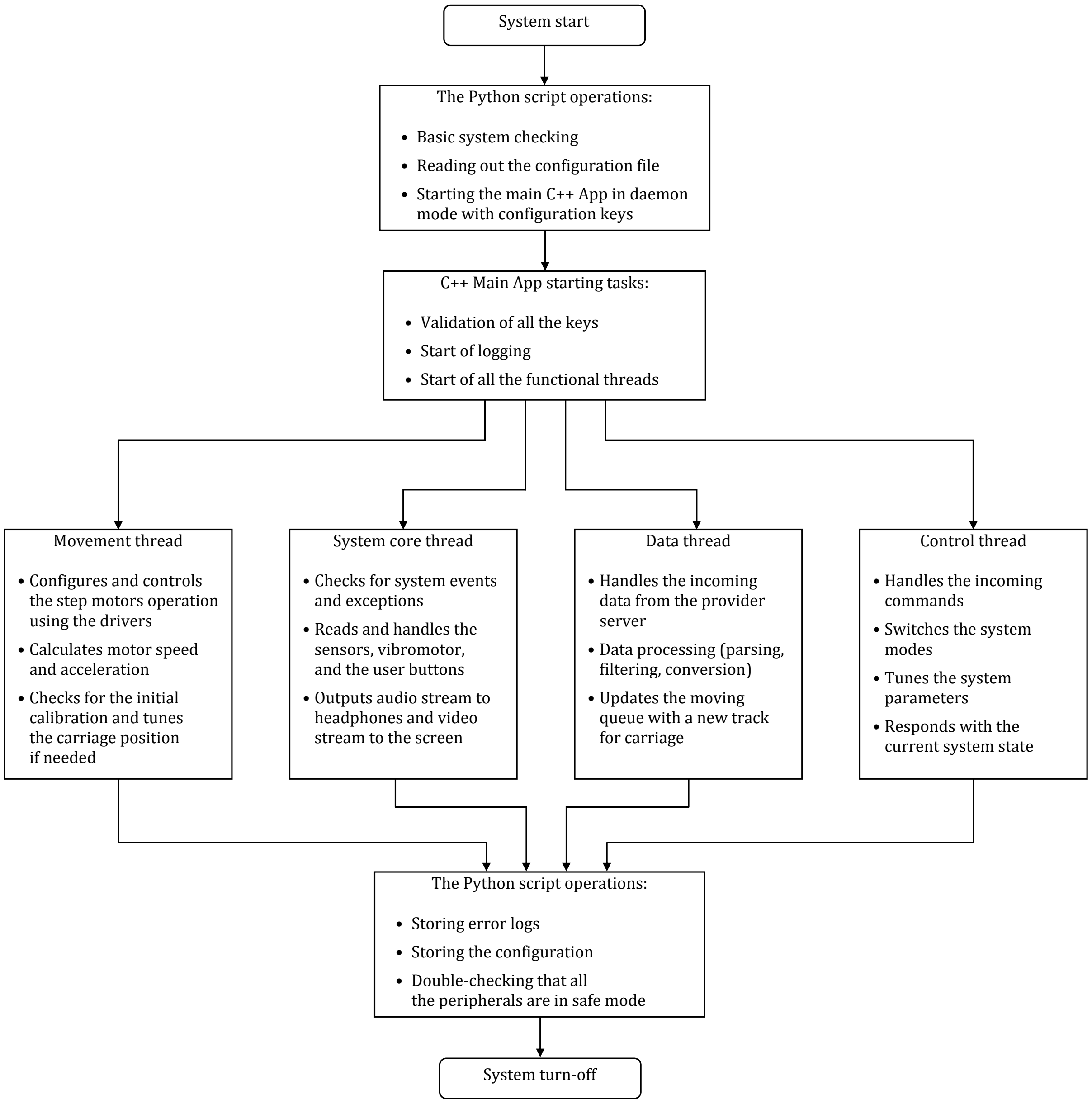
System start
The Python script operations:
• Basic system checking
• Reading out the configuration file
• Starting the main C++ App in daemon mode with configuration keys
C++ Main App starting tasks:
• Validation of all the keys
• Start of logging
• Start of all the functional threads
Movement thread
• Configures and controls the step motors operation using the drivers
• Calculates motor speed and acceleration
• Checks for the initial calibration and tunes the carriage position if needed
System core thread
• Checks for system events and exceptions
• Reads and handles the sensors, vibromotor, and the user buttons
• Outputs audio stream to headphones and video stream to the screen
Data thread
• Handles the incoming data from the provider server
• Data processing (parsing, filtering, conversion)
• Updates the moving queue with a new track for carriage
Control thread
• Handles the incoming commands
• Switches the system modes
• Tunes the system parameters
• Responds with the current system state
The Python script operations:
• Storing error logs
• Storing the configuration
• Double-checking that all the peripherals are in safe mode
System turn-off

Figure 6: The main internal logic and its components explaining how the system functions.

The system starts by running a Python script that checks whether all necessary services are running in the operating system and performs basic settings. The script reads the configuration file that contains the saved data of the previous system session and runs a daemon program written in C++ with the appropriate configuration keys. For the main task, C++ is chosen because it provides maximum code execution speed and, accordingly, the operation of all system processes. Less demanding processes for execution time, such as system start, shutdown, and other related tasks, are written in Python, as they are easy to adjust and add new functionality if necessary.

The C++ program starts with a basic task that checks the settings, initiates the start of logging, and launches four main processes, which are responsible for the main operation of the system. Dividing the work of this IoT-based system into four separate processes allows distributing tasks more logically and functionally. This eventually allows parallelizing the system work without noticeable delays, overhead, or issues of non-synchronization.

The first process is responsible for movement. It performs the initial initialization of the TMC2209 driver parameters for optimal and smooth operation of the motors. For this, two system UART ports are used. The movement control itself is performed via the GPIO interface. At the beginning of the work, the carriage position is calibrated: the carriage is first moved along the entire Y axis until the optical sensor is triggered, then a similar operation is performed along the X axis and the carriage is placed exactly in the center of the playing field, which starts the main functionality. To prevent emergency situations and mechanical failure, the system provides a movement timeout without triggering the edge sensors, after which the motor movement stops and the system signals an error. Over time, a small position drift is possible caused by belt slippage or accidental skips in motor steps. However, the system constantly calculates the carriage position in the field, so when the marker and, accordingly, the carriage approaches an edge, it automatically adjusts its real position to the expected one. The first process also calculates the optimal motor speed and acceleration when the direction of the marker changes. In fact, the movement parameters are calculated after receiving data from the server (a tracking data provider), but before transferring control to the motor drivers.

The second process is the core of the IoT-based embedded system. This process is responsible for processing system events such as user pressing buttons, handling system exceptions, and terminating work. The process also polls the state of edge sensors via the I^2C interface, controls signals to the vibration motor, controls the output of the audio stream via I^2S/BLE to headphones and the output of the video stream to the optional HDMI/MIPI screen. This process is also an integral part of edge computing, since it provides the user with actually converted 2D game data before it is transferred to a games data warehouse.

The third process is responsible for obtaining data on the player's movement on the field. It receives and processes data from the selected provider server (network gateway), but also it may receive data from storage. It performs data preprocessing, such as parsing, filtering, and converting into binary format, which is then added to the queue of the carriage movement trajectory, which is further handled by the process that monitors the movement. In simplified terms, the data stream is received in CSV format, strip by strip, where the belonging of each "column" is determined by the header (the first strip). The numbers in the specified columns are the coordinates of a particular player on whom a conditional marker is placed and whose movement is then monitored.

The control process is responsible for selecting a data column (i. e., the player on whom the marker is placed). This is the fourth, no less important, process in the system. The control process receives external commands from the user and responds to user's requests. The commands are used to configure the system, select a server with data, start, stop the game, command to recalibrate the system, etc. With the help of the commands, a remote operator can find out the current state of the system. The command system itself is as simple as possible, being actually the command itself in a string form and several parameters that are added to it. Before each command, a unique ID is transmitted – a key number that constantly increases by one with each new command, which allows to clearly separate one command from another. A response with the same ID is generated for each incoming command. For example, command "12345,RECALL" does not require additional parameters and will recalibrate the mechanics, and next command "12346,DATASRVADDR,192.168.0.100" will set a new IP address for the data server. The response to the first command will be "12345,0", where the status zero means the command has been executed without errors. This method allows to flexibly add new commands for controlling the system and makes it impossible to lose understanding of which previous command the response was received for.

To read field boundaries data from VL53L0X sensors, the I^2C interface is used. In total, there are four identical sensors on each of the lines. All four sensors are hardware-connected to one two-wire line, and each has its own unique address on the bus in order to be able to query a specific sensor. The VL53L0X has a default I^2C address of 0x29. But there is a software option to change it to any in the range 0x29...0x7F. Accordingly, the sensors that work in the system have addresses 0x2A, 0x2B, 0x2C, and 0x2D. Below (figure 7) is a commented example of a squeezed C++ code that initializes the interface, queries the sensors, performs averaging with outlier filtering. The result of the work is a conclusion whether the carriage is within the field; if not, then the current coordinates are corrected. This happens imperceptibly for the user and the corrected system continues to function.

```
#include <Wire.h>
#include <VL53L0X.h>
#include <iostream>
#include <algorithm>
#include <vector>
// Borders and the sensors index
enum { SENSOR_UP = 0, SENSOR_DOWN, SENSOR_RIGHT, SENSOR_LEFT, SENSOR_COUNT };
// The Class implements reading of four VL53L0X sensors using Raspberry Pi I2C
// Wire library and a driver for VL53L0X are used
class DistanceSensors {
public:
    DistanceSensors() { // Init sensor address
        sensors[SENSOR_UP].setAddress(0x2A); sensors[SENSOR_DOWN].setAddress(0x2B);
        sensors[SENSOR_RIGHT].setAddress(0x2C); sensors[SENSOR_LEFT].setAddress(0x2D); }
    bool initialize() { // Init I2C and the default sensors settings
        Wire.begin();
        for (int i = 0; i < SENSOR_COUNT; i++) {
            if (!sensors[i].init()) {
                std::cerr << "Err init at: " << std::hex << sensors[i].getAddress() << std::endl;
                return false; }
            sensors[i].setTimeout(500); sensors[i].startContinuous();
        }; return true; }
    // Readout the sensor value in mm
    int getAverageDistance(int sensorIndex) {
        std::vector<int> distances;
        // Reading more than once to have filtration and averaging
        for (int i = 0; i < kAverageCounts; i++) {
            int distance = sensors[sensorIndex].readRangeContinuousMillimeters();
            if (sensors[sensorIndex].timeoutOccurred()) {
                std::cerr << "Timeout occurred reading from sensor " << sensorIndex << std::endl;
                return -1; }
            distances.push_back(distance); }
        // We sort the values to discard the smallest and largest one
        std::sort(distances.begin(), distances.end());
        // Calculate the average value from remained
        int sum = 0;
        for (int i = 1; i < (kAverageCounts - 2); i++) { // Skip the first and last values
            sum += distances[i]; }
        return sum / ((kAverageCounts - 2))); } // Average value
private:
    const uint8_t kAverageCounts = 6; VL53L0X sensors[4]; // Sensors items
};
// The subthread that controls the playing field borders
DistanceSensors distanceSensors;
// Field border size (mm)
const int kDistanceLimits[SENSOR_COUNT] = {400, 400, 800, 800};
// Will initiate a coordinate correction if one of the borders is crossed
int borderCheckerInit() { if (!distanceSensors.initialize()) return -1; return 0; }
// The subthread that controls the playing field borders
// Will initiate a coordinate correction if one of the borders is crossed
void borderCheckerLoop() {
    for (int i = 0; i < SENSOR_COUNT; i++) {
        int distance = distanceSensors.getAverageDistance(i);
        if (distance != -1) {
            std::cout << "Sensor " << i << " aver dist:" << averageDistance << "mm" << std::endl;
            // Border cross check
            // Initiate X or Y coordinate correction in movements thread
        } else { if (distance < kDistanceLimits[i]) coordCorrection(i); } }
}
```

Figure 7: Interface initialization, queries of VL53L0X sensors, averaging, and filtering of outliers.

After receiving the command to terminate the operation, the Python script again takes control, which records the operation log, saves the current system configuration, checks whether all peripherals are in normal mode, and safely parks the operating system.

6. System operation and power characteristics

The IoT-based system normally functions in two modes: operation and sleep. The main system operation characteristics are summarized in table 2 along with commentaries and explanations. The computing module boot time depends on settings and used services just like in the case of desktop operating

Table 2
The main characteristics of the system operation.

Characteristic, process, state	Time of operation or execution	Commentary or explanation
Raspberry Pi Compute Module boot time	10...25 s	Depends on settings and used services
Carriage calibration time	3...10 s	Depends on an initial carriage position. The carriage moves to the center if this is an initial calibration (initialization) or backwards to its previous position if this is an on-demand calibration
Motor release timeout	60 s	The motors will not hold the carriage (configurable)
Sleep mode timeout	5 min	All the peripherals will be off. The computing module switches to the sleep mode (configurable)
System turn-off timeout	20 min	Shuts down the system completely (configurable)
Wake-up from the sleep mode	up to 1 s	Non-configurable
Server initial connection time	up to 2 s	Depends on server settings, location, latency
Respond to the command time	up to 5 ms	Non-configurable
Delay before motor moves the player upon setting its new position	up to 5 ms	Non-configurable
Application boot time	up to 50 ms	Includes motor driver and sensor initial set-up
Sensor readout time	up to 5 ms	Non-configurable, although depends on the tracking data provider and how data are acquired (whether it is received from optical tracking systems and/or player-mounted wearable sensors)

systems. The range of carriage calibration time is estimated by uniformly covering the miniaturized field of a 105-by-70 cm size with a step of 5 cm. Motor release timeout is set to a minute due to the player's inactivity (standstill, in fact) on a real pitch through a minute usually implies an injury or pending substitution. Sleep mode timeout is intended to save power. It is intended to be during the 15-minute break after the first half. By this very reason, system turn-off timeout is set to 20 minutes: if the player is still inactive following the 15-minute break for another 5 minutes, this means that he will not participate in the current match further.

The main system power characteristics are summarized in table 3 along with commentaries and explanations. The computing module consumes no more than 10 W, but this is about 22 % of what the stepper motor may consume. Meanwhile, our real-time experiments reveal that average system consumption is 19.3 to 22.1 W (this includes both time halves, while the 15-minute break is not included for obvious reason). A peak in 103 W occurs when both motors start moving at the same time at maximum speed. The vibration motor, drivers, the ToF distance sensor, and other peripherals consume at most 2.09 W. This power part, obviously, cannot be reduced.

Owing to internal protocols for data exchange among processes, as well as external protocols for real-time data acquisition, are optimized based on neglecting states that have relatively low effect on the overall model response [34, 35], the system consumes power almost linearly with respect to stepper motors. Therefore, the system can be expanded to an IoT network, where data from sensors are aggregated in a hierarchical manner with a branching factor of 2. For instance, if a is a power amount required to aggregate data at a network node (in the case of a single node considered above, it is already included into those 103 W), then level $k+1$ of the network tree will consume an additional amount

$$(93 + a) \cdot 2^{k-1} \tag{1}$$

Table 3
The main power characteristics of the system.

Component, process, state	Estimated power	Commentary or explanation
Raspberry Pi Compute Module	2...10 W	Depends on operations and settings
Stepper motor	up to 45 W	Depends on movement phase and driver settings
Vibration motor	1 W	In continuous vibration mode
TMC2209 drivers	0.03 W	Typical consumption
VL53L0X sensor	0.06 W	Average power consumption at 10 Hz with 33 ms ranging sequence
Other peripherals	up to 1 W	LEDs, interfaces, circuit passive elements
Motor release	up to 10 W	Drivers stop to hold the motors, while the other peripherals still work
Sleep mode	2.5 W	The computing module is in minimum load mode, the other peripherals are turned off
System peak consumption	up to 103 W	A 103 W peak is possible when both motors start moving at the same time at maximum speed; this corresponds to a situation, when the player starts accelerating either horizontally or vertically (diagonal acceleration is realized with a single stepper motor)
Average system consumption	19.3...22.1 W	The carriage is moved about 50 % time at most, which corresponds to the averaged player who covers a distance of 10.08 km

of power compared to level k, $k = 1,\ 2,\ \ldots$, where the amount of 93 W in (1) is the result of adding a couple of stepper motors (including consumption of the vibration motor, drivers, the ToF distance sensor, and other peripherals). Hence, the grand total of power $P_2(k+1)$ consumed by such a hierarchical network consisting of $k+1$ levels can be expressed recursively:

$$P_2(k+1) = P_2(k) + (93+a)\cdot 2^{k-1},\ \ k = 1,\ 2,\ \ldots \tag{2}$$

It is easy to prove that the power consumed by such a hierarchical IoT network consisting of N levels (to show performance of 2^{N-1} players) is

$$P_2(N) = 10 - a + (93+a)\cdot 2^{N-1},\ \ N = 1,\ 2,\ \ldots \tag{3}$$

Indeed, equality (3) can be proved by induction. The base case is when $N = 1$, i. e. we have just a single node similar to the considered above:

$$P_2(1) = 10 - a + (93+a)\cdot 2^{1-1} = 10 - a + 93 + a = 103, \tag{4}$$

which is true (table 3). By the inductive hypothesis we assume that equality (3) holds for any $N = k$:

$$P_2(k) = 10 - a + (93+a)\cdot 2^{k-1}. \tag{5}$$

By the inductive step, we are about to show that equality (3) holds for $N = k+1$. Here,

$$\begin{aligned} P_2(k+1) &= 10 - a + (93+a)\cdot 2^{k+1-1} = 10 - a + (93+a)\cdot 2^{k} = \\ &= 10 - a + (93+a)\cdot 2^{k-1} + (93+a)\cdot 2^{k-1} = P_2(k) + (93+a)\cdot 2^{k-1}. \end{aligned} \tag{6}$$

The last term in (6) is the right side of true recursion (2), and this proves equality (3) by induction with (4) and (5).

Alternatively, the system can be expanded to an IoT network, where data from sensors are distributed among M miniaturized fields without constructing a hierarchy. If b is a power amount required to route data to every newly added player (in the case of a single node considered above, it is already included into those 103 W), then it will consume an additional amount $93 + b$ of power. Hence, the grand total of power $P_{\text{par}}(k+1)$ consumed by such a parallelized network consisting of $k+1$ players can be expressed recursively:

$$P_{\text{par}}(k+1) = P_{\text{par}}(k) + 93 + b, \quad k = 1,\ 2,\ \ldots \tag{7}$$

It is easy (even trivially, to some extent) to prove that the power consumed by such a one-level-split IoT network showing performance of M players is

$$P_{\text{par}}(M) = 10 - b + (93 + b) \cdot M, \quad M = 1,\ 2,\ \ldots \tag{8}$$

Indeed, equality (8) can be proved by induction as well. The base case is when $M = 1$, i. e. we have just a single player:

$$P_{\text{par}}(1) = 10 - b + (93 + b) \cdot 1 = 10 - b + 93 + b = 103, \tag{9}$$

which is true (table 3) coinciding also with (4). By the inductive hypothesis we assume that equality (8) holds for any $M = k$:

$$P_{\text{par}}(k) = 10 - b + (93 + b) \cdot k. \tag{10}$$

By the inductive step, we are about to show that equality (8) holds for $M = k + 1$:

$$\begin{aligned} P_{\text{par}}(k+1) = 10 - b + (93 + b) \cdot (k+1) = 10 - b + (93 + b) \cdot k + 93 + b = \\ = P_{\text{par}}(k) + 93 + b. \end{aligned} \tag{11}$$

The last term in (11) is the right side of true recursion (7), and this proves equality (8) by induction with (9) and (10).

Nevertheless, it is worth noting that the proved power equalities (3) and (8) make sense for situations, when the soccer player starts abruptly accelerating along either axis X or axis Y (i. e., horizontally or vertically, respectively). So, the amount of consumed power in equalities (3) and (8) is a close-to-worst-case scenario. In other situations, power consumption is less.

7. Discussion

Compared to the performance of electronic applications, the developed system has a unique feature in its performance – tangibility. This feature is based on miniaturized mechanization of motion. Along with possibility to watch sports events closely enough, without necessarily touching the player's model figure, acoustics of the events additionally builds up an effect of presence, although being simulated. Such a mechanical, non-virtual, reality simulation further improves perception and forms positive interest in the game. Such models providing useful information and aesthetically positive impression are especially precious for people with visual impairments, who become capable to include tactility into their perception of the game, if viewing detailed peculiarities of player's motion is burdensome.

The computing module is edged as possible close to sensors with a purpose to further intensify data exchange without any delays during real-time game. Since 2010s, amounts of sports data acquired, collected, and recorded during matches have been gradually growing, becoming factually Big Data instances. So, this is crucial to further edge and parallelize computational systems in order to respond to the known Big Data challenges. Our system complies with those challenges by providing almost linear power consumption with respect to stepper motors, which are the most power-consuming in the system, and by neglecting data of non-effective states in the system.

The developed system is not just an amusement – replays of recorded games serve as an alternative approach for coaches and analysts to review player's consistency and vision of pitch in struggling situations. Besides, in soccer, often conceded goals from free kicks and corner kicks are results of improper individual actions of a definite player (say, defender). In this way, along with watching screen motion, miniaturized mechanical replay of the player's mistaken behavior helps better understand how to rectify individual and team tactics during free kicks and corner kicks.

Obviously, exposing performance of only one player on the miniaturized field seems to be a demerit. Indeed, our model cannot show the entire miniaturized match. Putting two and more players on the same field would require much complicated mechanics, which at the moment looks implausible due to physical obstacles of multiple belts and likely interference from magnets. A possible solution might be in using multilayered H-Bots by placing each carriage magnet on its own layer, but then each magnet would require to have specific calibration and amperage to hold tightly to the corresponding magnetic marker of the player.

Another drawback is the carriage calibration time. The carriage position is calibrated automatically, but it takes 3 to 10 seconds to complete calibration. Calibration during a real-time match takes about 3 to 5 seconds, which may incur significant delays, despite sensor readout time is just up to 5 milliseconds. Calibration during a real-time match is caused mainly by inertia of stepper motors, belt slippage, and accidental skips in motor steps [25, 26, 36]. However, our experiments reveal that soccer player's position drift does not exceed 3 mm, which is 3 meters on a real pitch. Moreover, calibration within a 105-by-70 cm miniaturized field is executed hardly noticeable for human eye.

8. Conclusion

For the purposes of perception improvement in sport analysis and entertainment, we have suggested a conceptual model of an IoT-based embedded system of mechanizing sport competition motion on a miniaturized sports field. The system operated by a Raspberry Pi Compute Module allows visually, tangibly, and acoustically perceiving location and motion of a player within sports field in accordance with his real-time motion. Recorded motion is reproducible as well. Motion data, acquired either from wearable sensors or by optical tracking systems and event data providers on wireless gateways in the stadium, are transmitted to the computing module for further filtering, truncating, and processing. The mechanical part of the system is based on the classic H-Bot layout with two stepper motors whose torques, determined by the real player's location and motion information, impart corresponding motion to the miniature player.

Based on the system tests for soccer with Raspberry Pi Compute Module 5, our system consumes no more than 103 W at peak, while average electric power consumption does not exceed 23 W. Besides, system power consumption is almost linear with respect to the 90 W consuming stepper motors, which makes this system expandable and scalable both hierarchically and linearly. Power consumption linearization is additionally facilitated by neglecting data of non-effective states in the soccer-exemplified system. In general, such optimization of internal protocols for data exchange among processes, as well as external protocols for real-time data acquisition, speeds up and rectifies the system response to actual player's motion on the sports field. The soccer-exemplified system has a 5 millisecond response, although the mechanical part with the two-belted carriage of the miniature player may incur a-few-seconds delays of accurately reproducing player's motion. The inaccuracy within a 105-by-70 cm miniaturized field is 3 mm at most, though, which is unlikely to be very noticeable.

Carriage calibration is still an open question yet the system boot requires special consideration. The matter is the worst boot case including initial carriage calibration is about 35 seconds, which seems to be pretty long. Another open question remaining to be studied is how the computing module must handle data frames with missed data and probable outliers. Handling the latter successfully is expected to have a strong impact on shortening the carriage calibration time.

Author Contributions: V. Romanuke and S. Dementiev conceived the system. S. Dementiev designed the system architecture and selected its components by supervision of V. Romanuke and S. Yaremko. S. Dementiev wrote most of the codes for the system functioning, while V. Romanuke measured timing and energy characteristics of the completed system. V. Romanuke wrote drafts for sections 1, 3, 8; S. Dementiev wrote drafts for Sections 2, 4, 5; sections 6 and 7 were drafted by V. Romanuke and S. Yaremko. V. Romanuke and S. Yaremko made the finalized layout of the paper.

Funding: This research received no external funding.

Data Availability Statement: Data of this research are available upon request.

Declaration on Generative AI: The authors have not employed any generative AI tools.

References

[1] L. Huang, D. Liao, W. Peng, Y. Zhou, K. Chen, Study on interface fatigue failure of flexible OLED display modules based on cyclic cohesive zone model, Displays 87 (2025) 102949. doi:10.1016/j.displa.2024.102949.

[2] X. Hao, G. Zhang, Numerical prediction of glass molding process for 3D curved screen, Heliyon 9 (2023) e19693. doi:10.1016/j.heliyon.2023.e19693.

[3] Z. Bian, Q. Chen, X. Chang, H. Wang, Y. Fang, H. Lu, Z. Wang, M. Xu, Fabrication of microlens arrays on curved substrates for large viewing angle integral imaging 3D display, Displays 84 (2024) 102755. doi:10.1016/j.displa.2024.102755.

[4] 3D Hologram Fans, 2025. URL: https://www.display-innovations.com/3d-hologram-fans.

[5] S.-C. Li, E. Muschter, J. Limanowski, A. Hatzipanayioti, Chapter 9 – Human perception and neurocognitive development across the lifespan, in: F. H. Fitzek, S.-C. Li, S. Speidel, T. Strufe, M. Simsek, M. Reisslein (Eds.), Tactile Internet, Academic Press, 2021, pp. 199–221. doi:10.1016/B978-0-12-821343-8.00021-6.

[6] S. Duyck, A. I. Costantino, S. Bracci, H. O. de Beeck, A computational deep learning investigation of animacy perception in the human brain, Communications Biology 7 (2024) 1718. doi:10.1038/s42003-024-07415-8.

[7] D. Memmert, B. Strauss, D. Theweleit, Perception and deception, in: Mind Match Soccer: The Final Step to Become a Champion, Springer Berlin Heidelberg, Berlin, Heidelberg, 2023, pp. 123–137. doi:10.1007/978-3-662-68035-3_7.

[8] C. Vater, U. Schnyder, D. Müller, That was a foul! How viewing angles, viewing distances, and visualization methods influence football referees' decision-making, German Journal of Exercise and Sport Research 54 (2024) 476–485. doi:10.1007/s12662-024-00947-5.

[9] V. Machado, R. Leite, F. Moura, S. Cunha, F. Sadlo, J. L. Comba, Visual soccer match analysis using spatiotemporal positions of players, Computers & Graphics 68 (2017) 84–95. doi:10.1016/j.cag.2017.08.006.

[10] R. Grazioli, M. L. H. Soares, P. Schons, A. Preissler, F. Veeck, S. Benítez-Flores, R. S. Pinto, E. L. Cadore, Curve sprint performance and speed-related capabilities in professional soccer players, Journal of Bodywork and Movement Therapies 40 (2024) 1034–1040. doi:10.1016/j.jbmt.2024.07.018.

[11] Y. Wang, H. Mao, Intelligent soccer system based on biosensor network technology, Measurement 173 (2021) 108564. doi:10.1016/j.measurement.2020.108564.

[12] C. Gupta, G. Varshney, An improved authentication scheme for BLE devices with no I/O capabilities, Computer Communications 200 (2023) 42–53. doi:10.1016/j.comcom.2023.01.001.

[13] B. J. C. Bastiaansen, R. J. K. Vegter, E. Wilmes, C. J. de Ruiter, E. A. Goedhart, K. A. P. M. Lemmink, M. S. Brink, Biomechanical load quantification of national and regional soccer players with an inertial sensor setup during a jump, kick, and sprint task: assessment of discriminative validity, Sports Engineering 27 (2024) 17. doi:10.1007/s12283-024-00458-4.

[14] C. Yang, M. Yang, H. Li, L. Jiang, X. Suo, L. Mao, W. Meng, Z. Li, A survey on soccer player detection and tracking with videos, The Visual Computer (2024) 1–15. doi:10.1007/s00371-024-03367-6.

[15] C. Yang, M. Yang, H. Li, L. Jiang, X. Suo, Z. Li, W. Meng, L. Mao, Soccer player tracking and data correction based on attention with full-field videos, The Visual Computer 40 (2024) 9141–9153. doi:10.1007/s00371-024-03300-x.

[16] Hawk-Eye | A global leader in the live sports arena, 2025. URL: https://www.hawkeyeinnovations.com.

[17] ChyronHego, 2025. URL: https://chyronhego.com.

[18] Hudl Statsbomb | Data Champions, 2025. URL: https://statsbomb.com.

[19] Opta Sports, 2025. URL: https://www.statsperform.com/opta.

[20] Wyscout – Hudl, 2025. URL: https://wyscout.hudl.com.

[21] D. Garrido, B. Burriel, R. Resta, R. L. del Campo, J. M. Buldú, Heatmaps in soccer: Event vs tracking datasets, Chaos, Solitons & Fractals 165 (2022) 112827. doi:10.1016/j.chaos.2022.112827.

[22] E. Shoop, S. J. Matthews, R. Brown, J. C. Adams, Hands-on parallel & distributed computing with Raspberry Pi devices and clusters, Journal of Parallel and Distributed Computing 196 (2025) 104996. doi:10.1016/j.jpdc.2024.104996.

[23] G. Guillen, Sensor Projects with Raspberry Pi. Internet of Things and Digital Image Processing, Apress Berkeley, CA, USA, 2024. doi:10.1007/979-8-8688-0464-9.

[24] H-Bot simulation, 2021. URL: https://discourse.mcneel.com/t/h-bot-simulation/127563.

[25] K. Craig, Mechatronic Model-Based Design Applied to an H-Bot Robot, Mechatronics and Applications: An International Journal (MECHATROJ) 2 (2021). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3839216.

[26] S. Weikert, R. Ratnaweera, O. Zirn, K. Wegener, Modeling and Measurement of H-Bot Kinematic Systems, 2011. URL: https://api.semanticscholar.org/CorpusID:66950480.

[27] R. Yu, Y. Qin, J. Peng, T. Guo, X. Tang, The Design and Implementation of Simple Corexy Structure Writing Robot, in: Y. Jia, J. Du, W. Zhang (Eds.), Proceedings of 2019 Chinese Intelligent Systems Conference CISC 2019, volume 593 of *Lecture Notes in Electrical Engineering*, Springer, Singapore, 2019, pp. 201–213. doi:10.1007/978-981-32-9686-2_25.

[28] Electronicos Caldas, 1.8° 42mm Hybrid Stepper Motor-NEMA17, 2025. URL: https://www.electronicoscaldas.com/datasheet/JK42HSxx-Series_Jkong-Motor.pdf.

[29] Texas Instruments, DRV88xx EVM GUI (Rev. C), 2025. URL: https://www.ti.com/lit/ug/slvu361c/slvu361c.pdf.

[30] Pololu, DRV8825 Stepper Motor Driver Carrier, High Current, 2025. URL: https://www.pololu.com/product/2133.

[31] Analog devices, TMC2209, 2025. URL: https://www.analog.com/en/products/tmc2209.html.

[32] PCM5102, 112dB Stereo DAC with 2VRMS output and Integrated Audio PLL, 2025. URL: https://www.ti.com/product/PCM5102.

[33] VL53L0X, Time-of-Flight (ToF) ranging sensor, 2025. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html.

[34] S. E. Mathe, H. K. Kondaveeti, S. Vappangi, S. D. Vanambathina, N. K. Kumaravelu, A comprehensive review on applications of Raspberry Pi, Computer Science Review 52 (2024) 100636. doi:10.1016/j.cosrev.2024.100636.

[35] A. Goel, R. Bhatia, Exhaustive Theoretical Study of Practical Free Space Optical Cooperative Relaying Technology: New Trends in IoT Communication, in: A. Prasad, T. P. Singh, S. Dwivedi Sharma (Eds.), Communication Technologies and Security Challenges in IoT: Present and Future, Springer Nature Singapore, Singapore, 2024, pp. 529–560. doi:10.1007/978-981-97-0052-3_26.

[36] N. Edoimioya, K. S. Ramani, C. E. Okwudire, Software compensation of undesirable racking motion of H-frame 3D printers using filtered B-splines, Additive Manufacturing 47 (2021) 102290. doi:10.1016/j.addma.2021.102290.