MARIO: Modular and Extensible Architecture for Computing Visual Statistics in RoboCup SPL

Domenico D. Bloisi, Andrea Pennisi,
Cristian Zampino, Flavio Biancospino,
Francesco Laus, Gianluca Di Stefano,
Michele Brienza, and Rocchina Romano

Abstract

This technical report describes a Modular and Extensible Architecture for Computing Visual Statistics in RoboCup SPL (MARIO), presented during the SPL Open Research Challenge at RoboCup 2022, held in Bangkok (Thailand). MARIO is an open-source, ready-to-use software application whose final goal is to contribute to the growth of the RoboCup SPL community. MARIO integrates multiple machine learning and computer vision techniques to perform automatic camera calibration, background subtraction, homography computation, player and ball tracking and localization, NAO pose estimation, and fall detection using an intuitive graphical unit interface (GUI). MARIO has been ranked no. 1 in the Open Research Challenge.

Contents

1 Introduction

1.1 RoboCup

RoboCup [1] is an annual international robotics initiative founded in 1996 by a group of university professors. The main goal of the competition is to promote robotics and AI research by offering an appealing and formidable challenge. To this end, the competition set a long term goal to create a fully autonomous humanoid robot team capable of competing and winning a soccer game, in compliance with the official rules of FIFA, by 2050.

1.2 Open Research Challenge

Since its foundation, one of the objectives of RoboCup was to push the boundaries of the research by offering high-level challenges. For the year 2022, the Open Research Challenge [2] proposed by the RoboCup Standard Platform League (often referred as RoboCup SPL) focused at generating statistics of the matches from external videos captured by using a GoPro-like camera. Mellmann *et al.*[3] tried to extract statistics using only the *GameController/TeamCom* data without success. They realized that the data was not sufficient to extract consistent statistics from a match. For such a reason, the Open Research Challenge focuses on the use of a camera for extracting match statistics. Such a challenge has 2 main goals: 1) to calculate extrinsic camera parameters (camera matrix) from the camera feed and to locate/track all moving objects (i.e.: ball and robots) on the field (short-term goals); and 2) to create game statistics (e.g.: time under control, successful, unsuccessful shots on goal, passes, etc.) based on the located objects and positions (long-term goals).

1.3 MARIO

MARIO is an end-to-end architecture for computing visual statistics in RoboCup SPL. One of the features of MARIO is the *modular architecture*, which allows the user to customize the system for extracting statistics.
SPQR team of Sapienza University of Rome in collaboration with UNIBAS WOLVES team of the University of Basilicata took part in the Open Research Challenge at RoboCup 2022 held in Bangkok (Thailand) where they have been ranked as no. 1 ex-aequo with the B-Human team (Table 1).

Rank	**Team**	**Score**
1	B-Human & SPQR	25 pt
3	NomadZ	21.9 pt
4	Nao Devils	18.1 pt
5	RoboEireann	13.5 pt
6	R-ZWEI KICKERS	5 pt

Table 1: Open-Research Challange 2022 ranking.

The final rank has been decided using a vote among all the SPL teams, which had the possibility to choose the five best teams through a Borda counting mechanism. The vote has been evaluated according to the following criteria:

- *Achievement of long/short term goals*;
- *Execution time and hardware requirements*;
- *Metrics (accuracy/precision/recall)*;
- *Technical strength*;
- *Novelty.*

1.4 Report Structure

The following sections provide the description of the system, the experiments, and the results. The remainder of the report is organized as follows:

- `MARIO System Overview` which describes the architecture of the system;
- `Initialization` containing an overview of the preliminary operations performed by the software, such as camera calibration, background subtraction, and homography computation;
- `Detection, Tracking and Localization` presents the detection and tracking systems;
- `Pose Estimation and Fall Detection` and `Illegal Defender` show how the pose estimation systems works, and how the player and the illegal defender fouls are detected;
- `Data Association and Statistics` describing how the visual data is merged with the data from the *GameController*/*TeamCom*;
- `GUI` and `Experimental Results`: give an overview of the graphical interface and the obtained results;
- `Conclusions`: draws the conclusions and and future work.

2 MARIO System Overview

2.1 Architecture

MARIO has a modular architecture where each module can perform a specific task independently. Figure 1 shows the architecture diagram of MARIO.

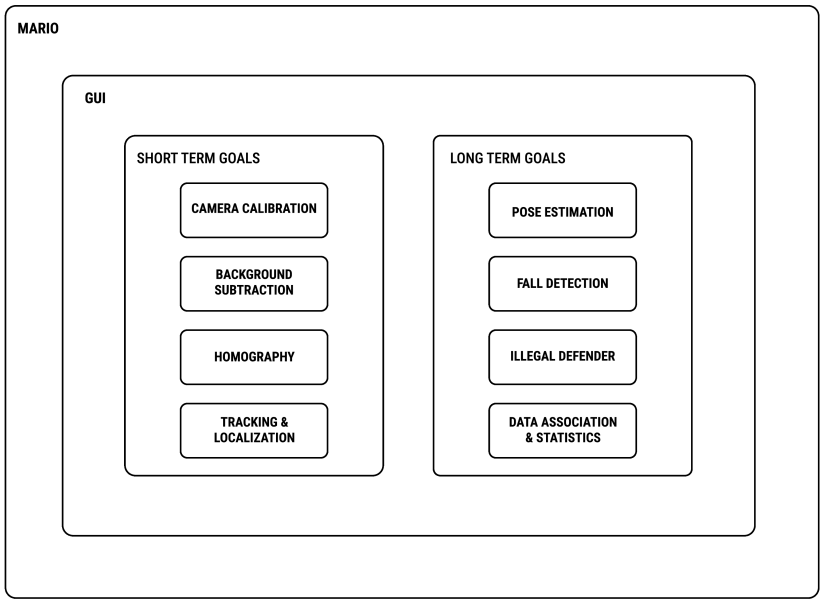


Figure 1: Architecture diagram of MARIO.

Each module is built according to the goal pursued, whether belonging to one of the short-term or long-term goals. The short-term modules can be grouped as follows:

- *Camera Calibration.* A preliminary camera calibration is performed in order to remove the camera lens distortion;
- *Background Subtraction.* A background subtraction technique is applied for extracting the moving objects;
- *Homography.* Homography is performed to compute a plan view of the field;
- *Tracking and Localization.* YOLOv5 and StrongSORT models are used to track and localize the players and the ball.

While the long-term modules are:

- *Pose Estimation.* A Convolutional Neural Network (CNN) model used to compute the robot pose. A custom data-set is created specifically for such a task;
- *Fall Detection.* Given the skeleton information, a Spatial-Temporal Graph Convolutional Networks (ST-GCN) model is used to detect the fall detection;
- *Illegal Defender.* The tracking results are used to check if no more than 3 players from same team are in the same penalty area;
- *Data Association and Statistics.* Game data containing player and ball information is extracted and used to compute the statistics of the game.

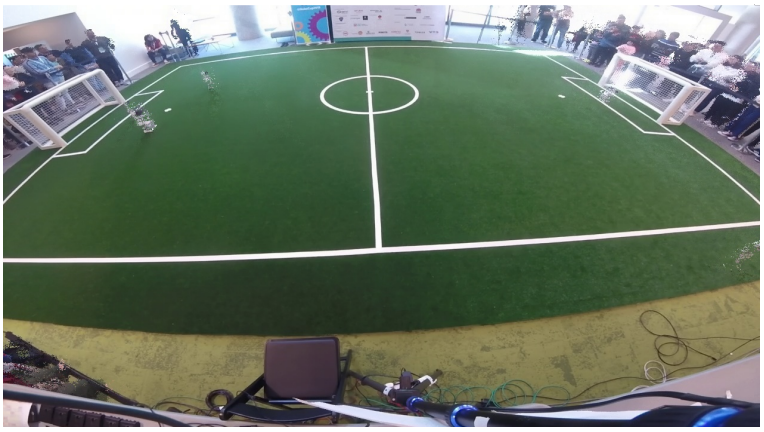
3 Initialization

3.1 Camera Calibration

A *camera calibration* procedure is performed in order to remove the distortion of the camera lens. In order to undistort the image, the *intrinsic parameters* are required. Such parameters include information such as the *focal length* (f_x, f_y) and the *optical center* (o_x, o_y), which are used to calculate the *camera matrix*. Such a matrix is a unique 3×3 matrix used to remove the distortion of the camera:

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}.$$

In general, the camera calibration process uses images of a 3D object with a geometrical pattern (i.e., a checker board), which is called *calibration grid*, for matching the 3D coordinates of the pattern to the 2D image points of the same pattern. Then, the matches are used to calculate the camera parameters.
For the open research challenge, we could rely only on the pitch information, because no calibration grid has been provided. For such a reason, we performed an initial setup aiming at finding patterns such as corners, intersections between the penalty areas and the goal lines, intersections between the midfield line and the sidelines, penalty area corners, and so on. We used the 2D points of those patterns and the corresponding 3D real-world coordinates of the same patterns to compute the camera matrix, the distortion coefficients, and the rotation and translation vectors. Such information has been saved to a calibration file that is used to undistort the video images. Figure 3 show an image prior and after the calibration process.



(a) Image prior the calibration process.

(b) Image after the calibration process.

Figure 2: Image prior and after the calibration process.

3.2 Background Subtraction

Background Subtraction (*BS*) is a popular and widely used technique that represents a fundamental building block for different Computer Vision applications, ranging from automatic monitoring of public spaces to augmented reality. The BS process is carried out by comparing the current input frame with the model of the background scene and considering as foreground points the pixels that differ from the model. So, the main problem is to generate a consistent background model that is reliable with the observed scene. BS has been largely studied and many techniques have been developed for tackling the different aspects of the problem. In particular, we used a method called Independent Multimodal Background Subtraction (IMBS).

3.2.1 Independent Multimodal Background Subtraction

Independent Multimodal Background Subtraction (IMBS) [4] is a BS method designed for dealing with highly dynamic scenarios characterized by non-regular and high frequency noise. IMBS is a per-pixel, non-recursive, and non-predictive BS method, meaning that:

- each pixel signal is an independent process (*per-pixel*);
- a set of input frames is analysed to estimate the background model based on a statistical analysis of those frames (*non-recursive*);
- the order of the input frames is considered not significant (*non-predictive*).

The above listed design choices are necessary in order to achieve a very fast computation, since working at pixel level and considering each background model as independent from the previous computed ones allow to carry out the BS process in parallel.

IMBS-MT. *Independent Multimodal Background Subtraction Multi-Thread* (*IMBS-MT*) [5] is an enhanced version of IMBS. IMBS-MT differs from the original IMBS in two aspects:

1. The background formation and foreground extraction processes are carried out in parallel on a disjoint set of sub-images from the original input frame;
2. The background model is initialized incrementally, i.e., the quality of the model is increased as soon as more frame samples are available.

IMBS-MT is designed for performing an accurate foreground extraction in real-time for full-HD images. IMBS-MT can deal with illumination changes, camera jitter, movements of small background elements, and changes in the background geometry. Figure 3 shows an example of an image processed by IMBS-MT.

(a) Image prior the application of IMBS-MT.

(b) Image after the application of IMBS-MT.

Figure 3: Image prior and after the application of IMBS-MT.

3.3 Homography

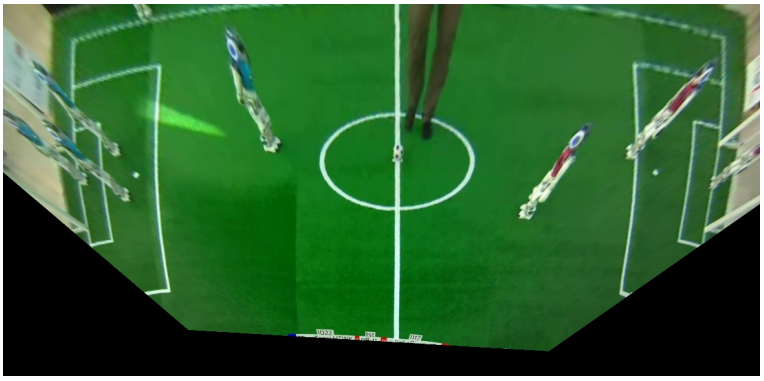
According to Wikipedia[6]: " *a homography is an isomorphism of projective spaces, induced by an isomorphism of the vector spaces from which the projective spaces derive. It is a bijection that maps lines to lines, and thus a collineation. In general, some collineations are not homographies, but the fundamental theorem of projective geometry asserts that is not so in the case of real projective spaces of dimension at least two. Synonyms include projectivity, projective transformation, and projective collineation*".
We used the homography matrix to apply a perspective transformation of the soccer field in order to obtain a bird-view of the pitch. To this end, we used an image of the soccer field as source and a reference image of the field as destination. In order to find the corresponding points between the two images,

we trained a neural network to recognize patterns like corners, intersections between the penalty areas and the goal lines, intersections between the midfield line and the sidelines, the penalty area corners, etc. The network model uses *YOLOv5* (a more detailed explanation of this model will be given in Sec. 4.1) and a proper data-set has been created for accomplishing the task. Figure 4 shows the patterns identified by the detector.

Figure 4: Patterns identified by the detector. T labels identify intersection points and L labels identify joining points. Other labels are used to identifies the penalty areas and the goal areas - such labels are used later to infer the field's version.

Using such a model, we can figure out the version of the pitch used in the match by analysing if the penalty areas and the goal areas are detected. Moreover, we can calculate the set of the source points used for the calculating the homography, which correspond to the intersection and the joining points of the field lines. Since, the set of the destination points is fixed, the homography and the pitch version are automatically computed. The system gives the possibility to manually choose the destination points by using the mouse. This method is usually used when the *re-projection error* of the automatic homography falls above a threshold that does not guarantee a good projection result. Figure 5 shows an image of the soccer field as seen from the camera point of view and the same image after the application of the perspective transformation.

(a) Shot captured by the camera.



(b) Perspective transformation of the shot captured by the camera.

Figure 5: Soccer field as seen from the camera point of view and the same image after the application of the perspective transformation.

4 Detection, Tracking, and Localization

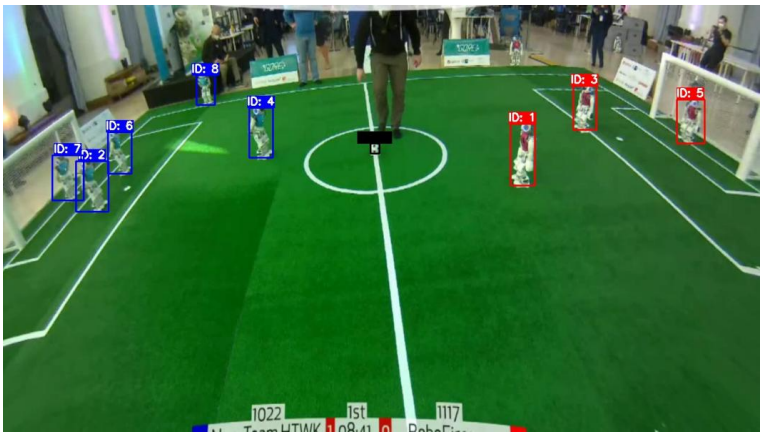
4.1 Detection

Object detection is a technique for locating instances of objects in images or videos. In the state-of-the-art, there exists many techniques for detecting objects [7] with deep learning methods. In this report, we focus on a widely used technique called YOLO.

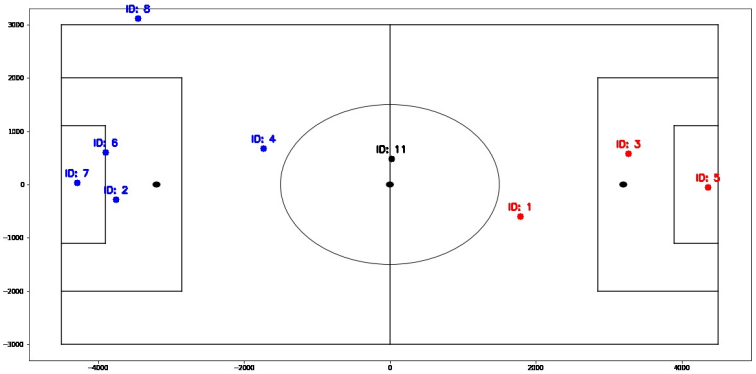
4.1.1 YOLO

You Only Look Once (YOLO) [8] is an object detection algorithm that is part of the one-stage detectors and uses a single network for performing both the tasks of detection and classification. It uses an end-to-end neural network on the whole image and divides the image into grid regions while predicting the rectangular boxes in each region of the grid, treating the object detection problem as a regression problem where the model only needs to perform one operation on the input. Each grid is responsible only for the targets whose centroids are within the grid, and each grid predicts the coordinates of several rectangular boxes as well as their scores. Each rectangular box corresponds to a five-dimensional output, i.e., coordinates and confidence level. Such an approach is fast in inference but less accurate.

YOLOv5. *YOLOv5* is one the most recent version of YOLO that increases the detection speed by 50% compared to the previous generation YOLOv4, with a model size only of 1/10 of that of YOLOv4. The adaptive anchor frame calculation and the use of a focus structure enhance the accuracy of the model for small target recognition. At the same time, the model has four network models with different depths, thus allowing the best balance between detection accuracy and recognition speed. Figure 6 shows an image of the detection of the robots and the ball, and an image of the perspective transformation of the positions of the robots and the ball into a 2D field view.



(a) Detections of robots and ball.



(b) Perspective tranformation of the positions of robots and ball into a 2D field view.

Figure 6: Detections and perspective transformation of the detections.

4.2 Tracking

Tracking is the task of predicting the positions of objects throughout a video using their spatial and temporal features. In other words, it takes the initial set of detections, assigns unique IDs, and tracks them throughout the frames of a video feed while maintaining the assigned IDs. Tracking is a two-step process that involves the use of a detection module for target localization and a motion predictor.

The tracking systems can be classified according to the number of objects to be tracked: *Single Object Tracking* (SOT) where the systems tracks only a single object even if there are many other objects in the same scene; *Multiple Object Tracking* (MOT) where the system tracks multiple objects. The tracker used in this report belongs to the MOT category. In particular, we used a tracker called

StrongSORT [9] [10], which is an enhanced version of the popular *DeepSORT* [11] algorithm, which in turn is an extension of *SORT* [12].

4.2.1 SORT

Simple Online Realtime Tracking (SORT) is an approach that uses classical tracking techniques such as *Kalman filters* [13] and *Hungarian algorithms* [14] to track objects. SORT is made of 4 key components which are as follows:

- *Detection.* An object detector detects the objects to be tracked. Detectors like FrRCNN, YOLO, and more are the most frequently used;
- *Estimation.* Then, the detections are propagated from the current frame to the next one using a constant velocity model. When a detection is associated with a target, the detected bounding box is used to update the target state while the velocity components are calculated using a Kalman filter framework;
- *Data association.* A cost matrix is computed as the Intersection-over-Union (IoU) distance between each detection and all the predicted bounding boxes from the existing targets. The assignment is solved using a Hungarian algorithm. If the IoU of the detection and the target is less than a certain threshold the assignment is rejected. This technique copes with the occlusion problem and helps in maintaining the correct IDs;
- *Creation and Deletion of Track Identities.* This module is responsible for the creation and deletion of IDs. Unique identities are created and destroyed according to the threshold defined in the previous step. If the overlap between the detection and the target is less than a threshold the object is marked as *untracked*. Tracks are terminated if they are not detected for a certain number of frames. If an old object appears again in the scene, the tracking system assigns a new ID to it.

SORT performs very well in terms of tracking precision and accuracy, but it is affected from a high number of ID switches and failures in case of occlusion. This is due to the association matrix used for data association that is calculated only using the eucldidean distance between the detections and the tracks. DeepSORT uses a better association metric that combines both motion and appearance descriptors. The network is trained on a large-scale person re-identification data-set, making it suitable for tracking context. The association metric model used in DeepSort is based on the cosine distance metric [15], which is suitable for calculating the similarity between two feature vectors.
In this work, we used an enhanced version of DeepSORT: StrongSORT. Which improves the standard DeepSORT in the following points:

- the use of a stronger appearance feature extractor in place of the simple CNN used by DeepSORT. It uses a ResNet50 model as backbone network that can extract much more discriminative features;

- the use of a feature updating strategy that involves the exponential moving average (EMA). The EMA updating strategy enhances the matching quality and reduces the time consumption;
- the use of a more robust Kalman filter in place of the vanilla one implemented in DeepSORT.

5 Pose Estimation and Fall Detection

5.1 Pose Estimation

Multi-person pose estimation is an important task that can bee used in different domains, such as action recognition, motion capture, sports, etc. The task predicts a pose skeleton for every person in an image. The skeleton consists of key points, or joints, that identifies specific body parts such as ankles, knees, hips, and elbows. Multi-person pose estimation problem can be approached in two ways: 1) *top-down*, which first applies a person detector and then runs a pose estimation algorithm for every detected person, and 2) *bottom-up*, where all the key-points are detected in a given image and then grouped by the entity instances. The bottom-up approach is usually faster than the top-down one, since it finds key points once and does not rerun pose estimation for each person.

5.1.1 OpenPose

OpenPose [16] is the first real-time multi-person system to jointly detect human body, feet, hands, and facial key points on a single image. OpenPose uses a CNN model such as VGG-19 for feature map extraction that then is used for a multi-stage CNN pipeline to generate the Part Confidence Maps (PCM) and the Part Affinity Fields (PAF). In the last step, the PCM and the PAF are processed by a greedy bipartite matching algorithm to obtain the poses for each entity in the image.

Lightweight OpenPose. In our system, we used a slightly optimized version of OpenPose, called *Lightweight OpenPose* [17]. Such a method allows real-time inference on CPU with a negligible accuracy drop. Some of the improvements achieved by the Lightweight OpenPose method are described in the following points:

- *Lightweight Backbone.* A lighter but still effective network is used. The common used network is MobileNet v1 [18];
- *Lightweight Refinement Stage.* To produce new estimations of PCMs and PAFs, the refinement stage takes features from the backbone network concatenated with the previous estimation of PCMs and PAFs. In order to share the computation between PCMs and PAFs and realize a consistent speed-up, a single prediction branch is used in the initial and the refinement stages. The last two layers are not shared between PCMs and PAFs and produce the model outpt;
- *Fast Post-processing.* Key points extraction is performed in a parallel way.

In figure 7, we show an example of pose estimation performed on a single frame of a RoboCup soccer game.
In the next subsection, we will explain how we created the training data-set and how the Lightweight OpenPose method has been adapted to the NAO robots.

Figure 7: Pose estimation for NAO robots.

5.2 Fall Detection

Human action recognition has become an active research area in the recent years, as it plays a significant role in video understanding. In general, there exists many methods for understanding human actions, and one of these, involves the use of dynamic skeletal information. The dynamic skeletal information can be represented by a time series of human joint locations, in the form of 2D or 3D coordinates. Human actions can then be recognized by analyzing the motion patterns of such coordinates. The skeleton and the joint trajectories of the human bodies are robust to illumination changes and scene variations, and they are easy to obtain using depth sensors or 3D pose estimation algorithms. In general, the approaches can be categorized into:

- *Feature Based methods*, that design several handcrafted features to capture the dynamics of the joint motion. The most common features are: covariance matrices of joint trajectories, relative positions of joints, and rotations and translations between body parts;
- *Deep Learning methods*, that involve the use of recurrent neural networks (RNN) and temporal CNNs.

By using the skeletal information retrieved through the Lightweight OpenPose method and observing the temporal dynamic of a single skeleton, it is possible to infer the type of action that is being performed by the robots. In particular, we are interested in capturing the fall-down action of the robots.

5.2.1 ST-GCN

The *Spatial-Temporal Graph Convolutional Network* (ST-GCN) [19] is a *graph convolutional network* (*GCN*) able to estimate actions from the spatio-temporal graph of a skeleton sequence. An example of spatio-temporal graph is shown in figure 8.

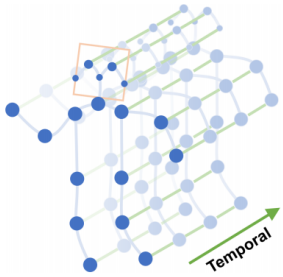


Figure 8: Spatio-Temporal graph of a skeleton sequence. Blue dots denote the body joints and blue edges between body joints defines natural connections in human bodies. [Image Source]

In general, there are two ways to perform convolutions on graphs. The first approach performs convolutions in the spatial domain, while the second one in the spectral domain. The ST-GCN method uses convolution in the spatial domain, where the data is represented in the form of graph nodes and connections between them. Since, we are dealing with graphs, the standard convolution cannot be applied to them, so, we use a modified convolutional kernel that only operates on the direct neighbors of the node. Moreover, it divides the neighbor set into several subsets and applies different weights to these subsets during the convolution. ST-GCN proposes three partition strategies to divide the neighbor set:

1. *Uni-labeling.* All nodes in a neighborhood are in the same subset;
2. *Distance partitioning.* The root node is inserted in a subset (distance 0) and the remaining neighbors into another subset (distance 1).
3. *Spatial configuration.* The nodes are partitioned according to their distances to the skeleton gravity center and compared with the root node. Three subset are then created: one subset for the root node; one subset for the centripetal nodes, that have a shorter distance than the root node; and one subset for the centrifugal nodes, that have longer distance than the root node.

Figure 9 shows the partitioning methods just described.

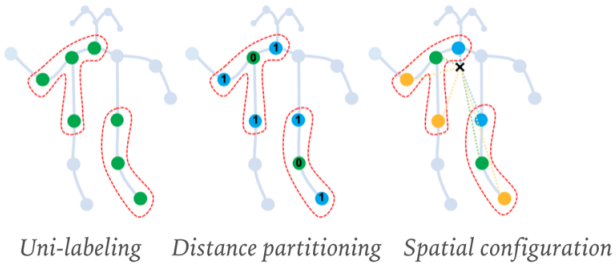


Figure 9: Partitioning methods. [Image Source]

The network contains several spatio-temporal convolution blocks. Each of these blocks performs three actions: (1) temporal convolution, partition, (2) graph convolution and, in order to optimize the results, (3) a second temporal convolution. Figure 10 shows the ST-GCN architectural scheme.

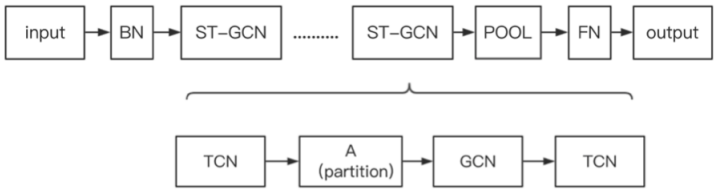
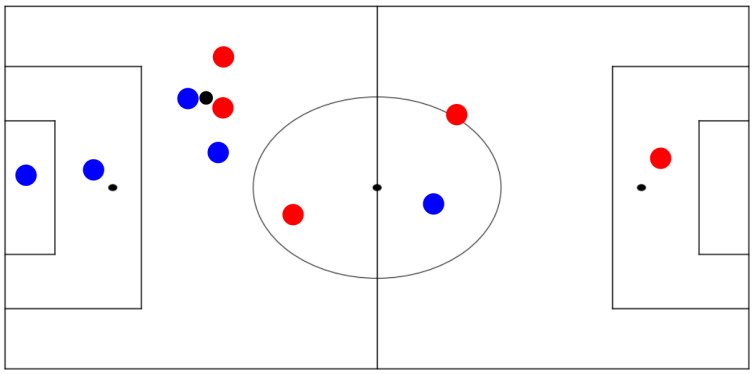


Figure 10: ST-GCN architectural scheme.

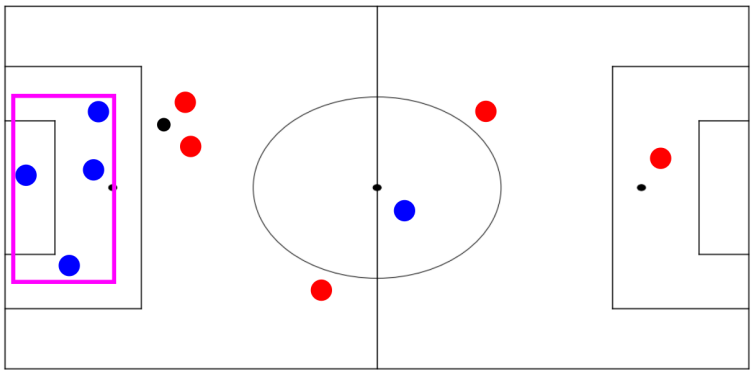
In this work, the ST-GCN method is used to identify the fallen robots. Due to the low-resolution images, the robot skeleton may be missing some of its limbs. To solve this issue, an additional fallback method has been implemented to detect falls. In particular, the shape of the bounding box obtained in the robot detection phase is used: if the aspect ratio of the bounding is below a specific threshold, the robot is considered is marked as "fallen".

6 Illegal Defender

The tracking results are used to check if the *illegal defender* foul is detected. An illegal defender context happens when three players from the same team are in the same penalty area. To be more specific, the *illegal defender* works as follows: the positions of the robots in the field is extracted from the tracking data, then, the robots of each team are identified and their positions are taken into account. If more than 3 robots from the same team are in the range of coordinates representing the penalty area, then an *illegal defender* foul is detected and a counter is increased by 1. At the end, the total number of illegal defender fouls per team is returned. Figure 11 shows two images that visually explain when an illegal defender foul occurs and when not.



(a) No illegal defender scenario.



(b) Illegal defender scenario. Purple square identifies the group of players in foul.

Figure 11: Visual difference between an illegal defender scenario and a no illegal defender scenario.

7 Data Association and Statistics

7.1 Data Association

Data association consists of performing the fusion between multiple homogeneous or heterogeneous sources in order to have a unique reference system from which extracting global information about the monitored environment. In our case, it consists in aligning the player tracking data with the data coming from the *GameController*. Such a step improves the accuracy of the available data and obtains additional information about the players, such as the team membership, the jersey number, whether the robot is leaving the field or not, whether the robot is dropped or inactive, whether it committed a foul, etc.

In our system, the data association process occurs in the first few frames of the game. The robot positions extracted from the tracking system and the positions from the *GameController* are associated using a K-Means approach. In some cases, such an association process turns out to be inaccurate due to the imprecision of the position data in the *GameController*. In fact, such positions are calculated by the robots themselves during the game, hence with a limited perspective.

7.2 Statistics

Match statistics and analysis computation is one of the long term goals of the Open Research Challenge. The statistics module focuses at estimating the **heatmaps** and the **trackmaps** of robots and ball, **pass and shot maps**, and **ball possession**. All the statistics have been computed using the *game_data* file converted into a data frame on which we carry out the data analysis operation filtering to obtain new data frames.

7.2.1 Heatmaps of robots and ball

Heatmap shows the most occupied locations by each robot and ball in the 2D plan representing the pitch top view. The system allows to select a specific robot by its id or to choose the ball (id 11). A new data frame called **player_data** is created from the main one by filtering all the rows with different ids from the chosen one. Then, all the data about the player or the ball is plotted on a graph whose density depends on the most occupied area by the chosen robot or ball.

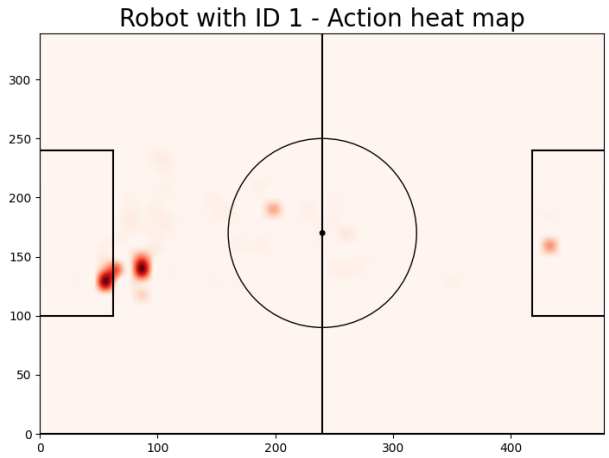


Figure 12: Heatmap of robot with ID 1.

7.2.2 Trackmaps of robots and ball

A trackmap shows all the points touched by each robot and ball in the 2D plan. It is estimated in a similar way to the heatmap, but instead of using a density graph, all the positions

$$(x, y)$$

of the chosen robot or ball are projected on the 2D field model.

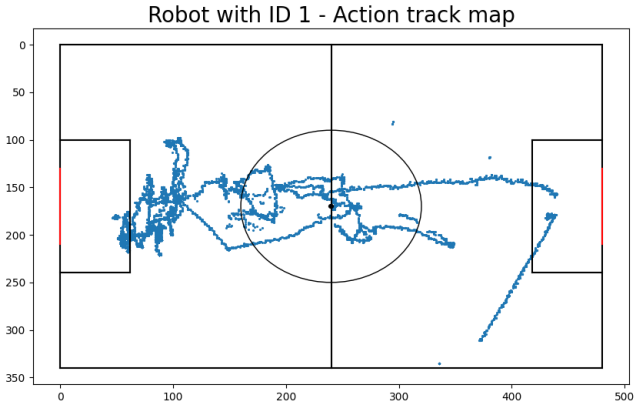


Figure 13: Trackmap of robot with ID 1.

7.2.3 Pass and shot map

Pass and shot map shows all the passes, shots, shots on target, and goals of each team in different colors. A new filtered data frame called *ball* is obtained from the main one and includes all the information about the ball. Then, other 2 data frames are created from *ball*: the first one called *ball0* stores the information about ball at the current frame n, while the second one, *ball1* stores information about the ball at frame $n + 5$. Then, the euclidean distance between *ball0* and *ball1* is calculated for extracting the number of passages. If this distance is greater than 50 and less than 70 a pass counter is increased. If the ball position at frame n is outside the area and it is inside at frame $n + 5$, a shot counter is increased depending on which area is located. In order to calculate the shots on target, we consider only the y-coordinates related to the goal area. To calculate the goals, if the ball position at frame n is outside the goal and goes over it at frame $n + 5$, a goal count is increased by 1.

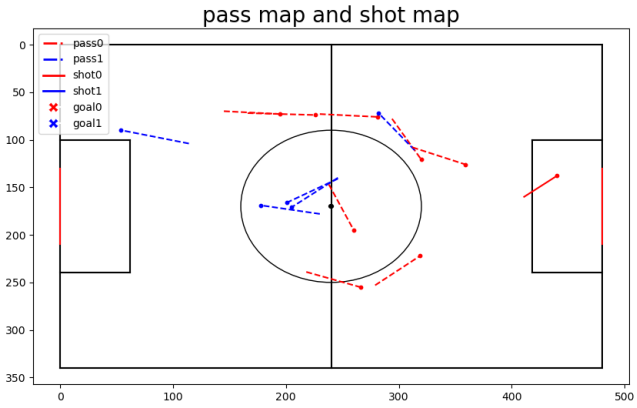


Figure 14: Map of pass and shot.

7.2.4 Ball possession

This section calculates the numerical values for the ball possession of each team. Three filtered data frames are created from the main one, namely: *team0*, *team1*, and *ball*. For each frame in which the ball is detected and tracked, the euclidean distance between the ball position and *team1* and *team0* is calculated. If the distance between ball and *team0* is less than the distance between ball and *team1*, the possession counter for the team 0 is increased, otherwise we increase the possession counter for the team 1. At the end of the game, the percentage for the ball possession for both the teams is calculated.

8 GUI

A graphical user interface has been implemented using Python with the graphic module *tkinter*. From the GUI, you can run all the MARIO flow of execution, from calibration to final stats. It consists of 3 windows (*Configuration, Tracking and Analysis*) accessible by using the buttons.

8.1 Configuration

The main window, called *MARIO* is the configuration window where you can choose the video of the match to analyze, the extrinsic parameters (optional), the game controller (optional), and the calibration file (if your video is not calibrated); then, you can run the calibration and the background subtraction modules or directly go to the tracking step. In addition there are 3 icon buttons which link to UNIBAS WOLVES, SPQR team and UNIBAS web pages.

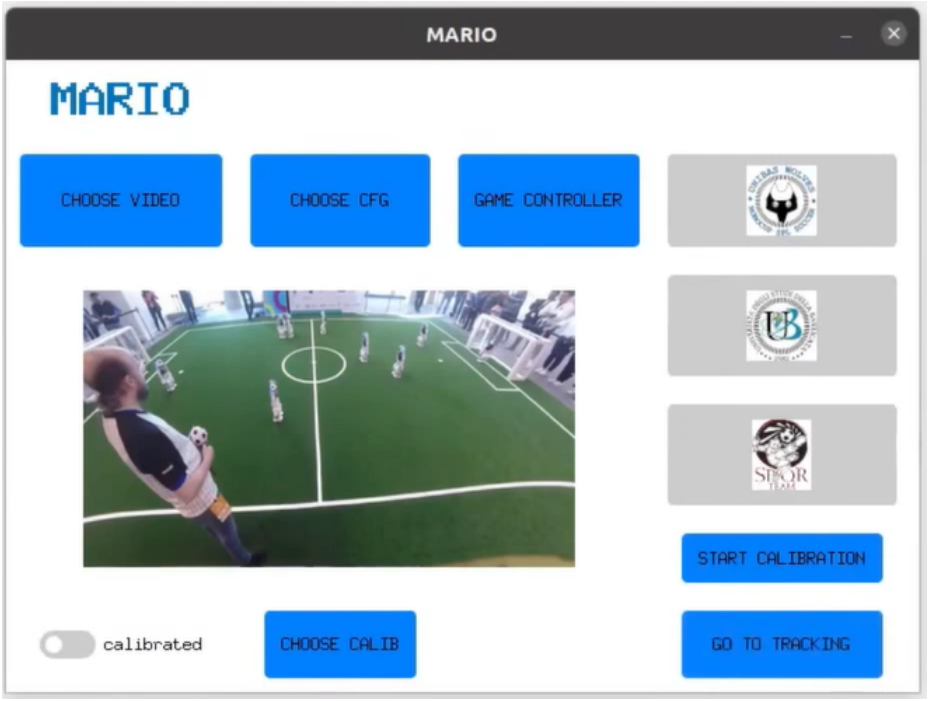


Figure 15: Main window of MARIO application.

8.2 Tracking

After the calibration phase that outputs a calibrated video, a second window called *MARIO-Tracking* can be used to evalute the Homography. Homography

is computer before starting the tracking. In this window, 2 videos are showed: the first one is the tracking video with bounding boxes and pose estimation (only for robots) for each robot, the second one is the tracking on the 2D model of the pitch. At the end of this phase, a file (game_data) is created (in csv format) containing all the data about the position, the team membership, the jersey number, the id, and the frame of each robot.

Figure 16: Window for displaying tracking.

8.3 Analysis

You can subsequently open the analysis window by pressing the *GO TO ANALYSIS* button. In this third window, called *MARIO-Analysis*, you can estimate the robot heatmaps and trakmaps, pass and shot map, and calculate the numerical stats of the match, such as goals, ball possession, total attempts, attempts on target and total passes that are showed in a window with a soccer match scoreboard design.

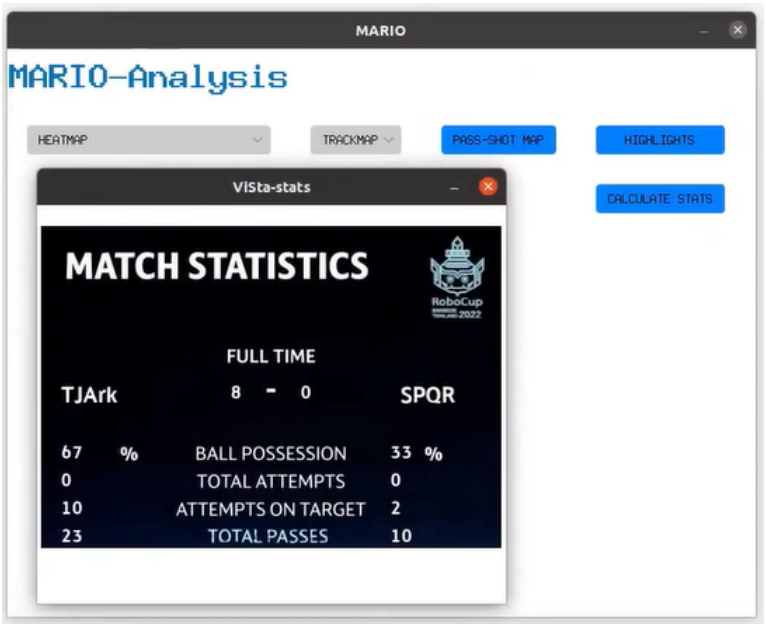


Figure 17: Window for displaying analysis.

9 Data-sets

In this section, we present 2 data-sets used for training the networks for accomplishing the tasks of detection and pose estimation. In particular, the pose estimation data-set is a novelty in the scientific community and it is publicly available.

9.1 Detection data-set

To carry out robots and ball detection, the database of images provided by the RoboCup SPL has been used. Each team provided 5000 images captured from the matches of RoboCup SPL 2019, and for each one they carried out the labelling as explained in the rule-book [2]. The data-set consists of 35000 images, and we used 24000 images as training set, 3000 images as validation set, and the remaining 8000 images as testing set.

The data-set used for the detection is distributed as-is at the following link.

9.2 UNIBAS NAO Pose data-set

NAO robots share the same body structure as human beings. However, from a model point of view, the PCMs and PAFs calculated by this method show the existence of substantial differences between robots and humans. For such a reason, we built a new data-set called *UNIBAS NAO Pose Data-set*.

To create the data-set, we used the COCO Annotator tool [20], a web-based image annotation tool designed for efficiently labeling images for image localization and object detection. All annotations share the same basic data structure. The pose is made of up to 18 key-points: ears, eyes, nose, neck, shoulders, elbows, wrists, hips, knees, and ankles. The annotations are stored using a JSON structure.

The data-set is distributed as-is at the following link.

10 Experimental Results

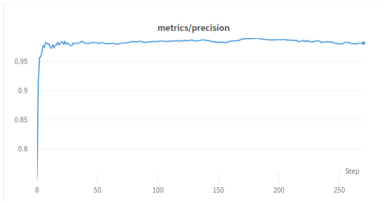
A video which highlights the main features of MARIO is available at the following link.

10.1 YOLO Results

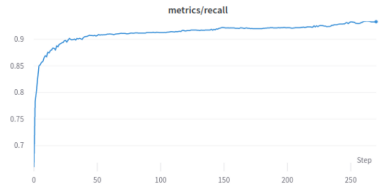
To train YOLOv5, the data-set described in Sec. 9.1 has been converted into the YOLO format. Then, YOLOv5 has been trained for 270 epochs obtaining the following results:

- *Precision*: 0.982;
- *Recall*: 0.932;
- *Mean Average Precision (mAP) 0.5*: 0.953;
- *Mean Average Precision (mAP) 0.5:0.95*: 0.822.

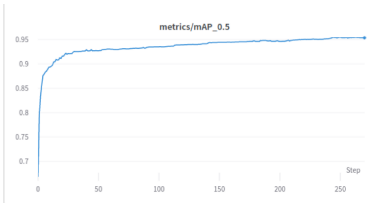
Figure 18 shows all the metrics scored by the YOLOv5 model used.



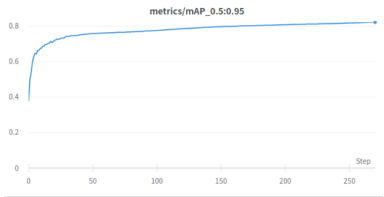
(a) Precision.



(b) Recall.



(c) mAP 0.5.



(d) mAP 0.5:0.95.

Figure 18: Metrics of the YOLOv5 model.

10.2 Run-time Performance

Run-time performance is evaluated on three different type of hardware:

- a *low-end hardware* with CPU Intel i3-6006U, GPU Intel Graphics HD 520, and 8 GB of RAM;

- a *middle-end hardware* with CPU Ryzen 7 5700U, GPU NVidia RTX 3050 Mobile with 4 GB of vRAM, and 16 GB of RAM;
- an *high-end hardware* with CPU Intel Xeon W-2135, GPU Nvidia Quadro P4000 with 8 GB of vRAM, and 32 GB of RAM.

The performance has been evaluated with and without the pose estimation system. Pose estimation is a very computationally expensive operation and it significantly degrades the performance of the system. Table 2 shows the performance of MARIO, in term of average frames per second, with all the tested hardware scenarios, with and without pose estimation.

	w/ Pose Estimation	**w/o Pose Estimation**
Low-End HW	3	7
Middle-End HW	6	16
High-End HW	10	20

Table 2: Performance of MARIO - in term of average frames per second - with all the tested hardware scenarios, with and without pose estimation.

11 Conclusions

In this technical report, we described MARIO, a modular end-to-end architecture for computing visual statistics in RoboCup SPL. Each module of MARIO performs a specific task, namely:

- camera calibration;
- background subtraction;
- homography;
- tracking and localization;
- pose estimation;
- fall detection;
- illegal defender;
- data association and statistics.

All these modules are wrapped around a simple and intuitive graphical user interface (GUI).

The system includes many novelties such as: the automatic homography computation, the pose estimation task, the creation of a custom data-set. From the experimental results, MARIO appears to be able to fulfill all the major goals set by the Open Research Challange of the year 2022.

11.1 Future Directions

Some of the possible improvements that can be implemented in the future are:

- the augmentation of the data-set created for the 2D pose estimation and the extraction of the 3D poses from the 2D poses;
- the detection of other actions such as walking, passing and kicking;
- computation in real-time of the illegal defender foul;
- add more statistics;
- improve the performance on low-end hardware.

A Environment Setup and Codebase

Prerequisities. MARIO has been tested on Ubuntu 20.04 LTS.

System Dependencies. In order to resolve all the system dependendies, it is possibile to execute the following commands:

```
$ sudo apt install git python3-pip zlib1g-dev libjpeg-dev \
libpng-dev
```

OpenCV. In order to install OpenCV, you can execute the following command:

```
$ pip install opencv-python==4.6.0.66
```

To install OpenCV from source, it is necessary to follow these steps:

- install the build tools and the dependencies:

```
$ sudo apt install build-essential cmake \
pkg-config libgtk-3-dev libavcodec-dev \
libavformat-dev libswscale-dev libv4l-dev \
libxvidcore-dev libx264-dev libjpeg-dev libpng-dev \
libtiff-dev gfortran openexr libatlas-base-dev \
python3-dev python3-numpy libtbb2 libtbb-dev \
libdc1394-22-dev libopenexr-dev \
libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
```

- clone the OpenCV's and OpenCV contrib repositories:

```
$ mkdir $HOME/opencv_build && cd $HOME/opencv_build

$ git clone https://github.com/opencv/opencv.git

$ git clone https://github.com/opencv/opencv_contrib.git
```

- create a temporary build directory, and navigate into it:

```
$ cd $HOME/opencv_build/opencv

$ mkdir -p build && cd build
```

- setup the OpenCV build:

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D INSTALL_C_EXAMPLES=ON \
  -D INSTALL_PYTHON_EXAMPLES=ON \
  -D OPENCV_GENERATE_PKGCONFIG=ON \
  -D OPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules \
  -D BUILD_EXAMPLES=ON ..
```

- start the compilation process:

```
$ make -j<number_of_cores>
```

- install OpenCV:

```
$ sudo make install
```

- verify the installation:

```
$  python3 -c "import cv2; print(cv2.__version__)"
```

Python libraries. Python modules dependencies are the following:

```
$ requests==2.28.0

$ torch==1.11.0 to11.0 torchvision==0.12.0

$ pyyaml==6.0

$ tqdm==4.64.0

$ matplotlib==3.5.2

$ seaborn==0.11.2

$ gdown==4.4.0

$ cython==0.29.30

$ tensorboard==2.9

$ easydict==1.9
```

```
$ scikit-learn==1.1.1

$ protobuf==3.20.0

$ https://github.com/KaiyangZhou/deep-person-reid/archive/master.zip
```

Clone and run the project. In order to run the project, clone the repository with the following command:

```
$ sudo git clone https://github.com/unibas-wolves/MARIO.git
```

and insert:

- the content of this folder in `MARIO/detectionT`;
- the content of this folder in `MARIO/data`;
- the content of this folder in `MARIO/video`;

To start the project, run the following command:

```
$ python3 ./gui/mario.py
```

Know problems. Possible problems that may be encountered during a run of the application:

- `FileNotFoundError: No such file or directory: 'python'`
  - **Solution**: change the value `python` in `python3` in `Preparation.py`, at line 70.
- `FileExistsError: File exists: './MARIO/imbs-mt/images'`
  - **Solution**: delete `images` folder in `MARIO/imbs-mt` folder.

A.1 Docker file

Prerequisites. Prerequisites includes Ubuntu 20.04 and Docker.

System Dependencies. In order to resolve all the system dependencies, it is possible to execute the following commands:

```
$ sudo apt-get install ca-certificates curl gnupg \
lsb-release
```

Import Docker's official GPG key. Import Docker GPG key using the following commands:

```
$ sudo mkdir -p /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Setup Docker repository. In order to setup the Docker repository, run the following commands:

```
$ echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.gpg]  \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee \
/etc/apt/sources.list.d/docker.list > /dev/null
```

Install Docker Engine. To install Docker Engine, run the following commands:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io \
docker-compose-plugin
```

Folder setup. Clone the following repositories

```
$ sudo git clone --branch MARIO-docker \
https://github.com/unibas-wolves/MARIO.git

$ sudo git clone https://github.com/unibas-wolves/MARIO.git

$ mv ./MARIO/ ./MARIO-docker/
```

and insert:

- the content of this folder in `MARIO-docker/MARIO/detectionT`;
- the content of this folder in `MARIO-docker/MARIO/data`;
- the content of this folder in `MARIO-docker/MARIO/video`;

Build and Setup Docker image. In order to build and setup the Docker image, run the following commands:

```
$ sudo docker build -t robocup2022 .

$ xhost +

$ sudo docker run -it --volume=/tmp/.X11-unix:/tmp/.X11-unix \
--device=/dev/dri:/dev/dri --env="DISPLAY=$DISPLAY" robocup2022

$ cd src
```

To start the project, run the following command:

```
$ python3 ./gui/mario.py
```

Known problems. Possible problems that may be encountered when running the application include the following:

- `FileNotFoundError: No such file or directory: 'python'`
  - **Solution**: change the value `python` in `python3` in `Preparation.py`, at line 70.

References

[1] RoboCup. *Objective*. URL: https://www.robocup.org/objective.

[2] RoboCup Technical Committee. *RoboCup Standard Platform League (NAO) Rule Book*. 2022.

[3] Benjamin Schlotter Heinrich Mellmann and Philipp Strobel. *Data Driven Research and Development in RoboCup - Collection, Organization and Analysis of RoboCup Game Data*. 2018.

[4] Domenico Bloisi and Luca Iocchi. "Independent multimodal background subtraction." In: *CompIMAGE*. 2012, pp. 39–44.

[5] Domenico D Bloisi, Andrea Pennisi, and Luca Iocchi. "Parallel multi-modal background modeling". In: *Pattern Recognition Letters* 96 (2017), pp. 45–54.

[6] Wikipedia. *Homography*. URL: https://en.wikipedia.org/wiki/Homography.

[7] Syed Sahil Abbas Zaidi et al. "A Survey of Modern Deep Learning based Object Detection Models". In: *CoRR* abs/2104.11892 (2021).

[8] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[9] Yunhao Du et al. *StrongSORT: Make DeepSORT Great Again*. 2022. eprint: arXiv:2202.13514.

[10] dyhBUPT. *StrongSORT*. URL: https://github.com/dyhBUPT/StrongSORT.

[11] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[12] Alex Bewley et al. "Simple Online and Realtime Tracking". In: *CoRR* abs/1602.00763 (2016). arXiv: 1602.00763. URL: http://arxiv.org/abs/1602.00763.

[13] Gregory F. Welch. "Kalman Filter". In: *Computer Vision: A Reference Guide*. Cham: Springer International Publishing, 2020, pp. 1–3. ISBN: 978-3-030-03243-2. DOI: 10.1007/978-3-030-03243-2_716-1. URL: https://doi.org/10.1007/978-3-030-03243-2_716-1.

[14] N. P. Arun Kumar et al. "Performance Study of Multi-target Tracking Using Kalman Filter and Hungarian Algorithm". In: *Security in Computing and Communications*. Ed. by Sabu M. Thampi et al. Singapore: Springer Singapore, 2021, pp. 213–227. ISBN: 978-981-16-0422-5.

[15] Nicolai Wojke and Alex Bewley. "Deep Cosine Metric Learning for Person Re-identification". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087.

[16] Zhe Cao et al. “OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (July 2019), pp. 1–1. DOI: 10.1109/TPAMI.2019.2929257.

[17] Daniil Osokin. “Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose”. In: (Nov. 2018).

[18] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. DOI: 10.48550/ARXIV.1704.04861.

[19] Sijie Yan, Yuanjun Xiong, and Dahua Lin. “Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition”. In: *CoRR* abs/1801.07455 (2018). arXiv: 1801.07455. URL: http://arxiv.org/abs/1801.07455.

[20] Justin Brooks. *COCO Annotator*. URL: https://github.com/jsbroks/coco-annotator/.