AEP Summer 15: Final report

Shared Robo-Vision

Team: Alberto Vaccari, Dapeng Lan and Yu Liu.

Introduction

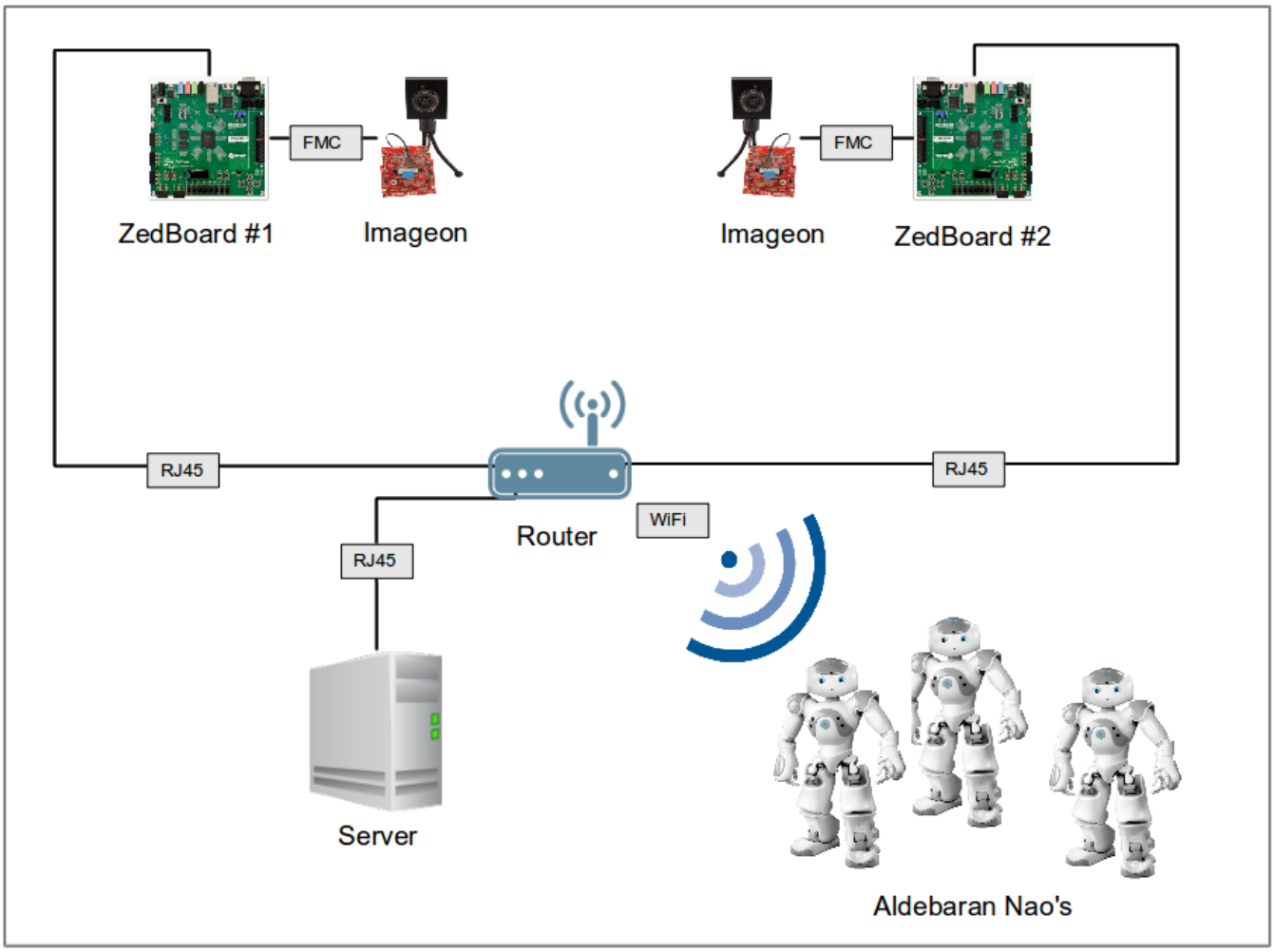
RoboCup is an annual international robotics competition with the aim to promote robotics and AI research.

The AOT department of TU-Berlin has been working and participating in this competition since 2003.
After discussing with some people involved in the project, an interest in a “tool for logging matches and the behaviour of the robots” was shown.
This project was chosen as a way to improve their workflow and support their useful contribution in the field of robotics and artificial intelligence.

The main idea is to position 2 cameras overseeing the 2 sides of the football pitch, where the match will be held. Each robot will have a marker on their head to identify them, also used for tracking.
The 2 camera will be connected to a server for tracking and logging. The server will also be able to communicate with the robots through the network.

The operator will be able to see the current position of the ball and of each robot in the field, record the match and directly control a specific robot.

System Diagram



Objectives

The objectives of the project are the following:

- Transmit HD images (1920*1080) from 2 ZedBoards to a computer
- Perform some degree of image processing directly on the ZedBoards
- Track the position of the robots on the field
- Track the position of the ball on the field
- Record a match
- Replay a recorded match
- Control the movements of the robots from the server application

Roles

Alberto:

- Tracking of the robots
- Tracking of the ball
- Recording a match
- Replaying a saved match
- Communication and control of the robots
- Writing progress and final reports
- Preparing Demo and Presentation

Dapeng:

- Fetching HD images from FMC-Imageon V2000C camera
- Transmission of HD images from 2 ZedBoards to the server
- Image pre-processing on the ZedBoards

Yu:

- Receiving images from the ZedBoard
- Image pre-processing on the ZedBoards

Current Status

Here is the current status of the tasks:

- ☑ Tracking of the robots
- ☑ Tracking of the ball
- ☑ Recording a match
- ☑ Replaying a saved match
- ☑ Communication and control of the robots
- ☑ Fetching HD images from FMC-Imageon V2000C camera
- ☐ Transmission of HD images from 2 ZedBoards to the server
- ☐ Image pre-processing on the ZedBoards
- ☑ Receiving images from the ZedBoard

Completed | Partially Completed | Started | Not Started

Image Processing – Overview

This section covers the image processing, and the reasons behind some of the choices made.

Initially, it was thought that using already existing patterns on the robots (Aldebaran Nao's) for their localization would be the best choice.

By simply colouring the almost semi-circular front area on the head of the Nao, it is possible to get both orientation and rotation of the robot in the field.

Image 1: Original

Image 2: With coloured front, used for testing

Locating the shape was a fairly easy task, although calculating its difference in rotation from the template shape ended up being a fairly challenging task, as OpenCV doesn't provide rotation information when performing a template matching (searching an image template within another image).

Since the task was not only to detect the position and the rotation of each robot on the field but also to identify it, the suggestion of looking into ArUco markers was followed.

Image 3: Template for shape

ArUco markers have been created to facilitate Augmented Reality applications, providing an easy and light-weight library for detecting 2D markers and returning their 3D coordinates in the real world. Even if they were initially created for AR projects, their use rapidly extended to robotics and computer vision ones as well.

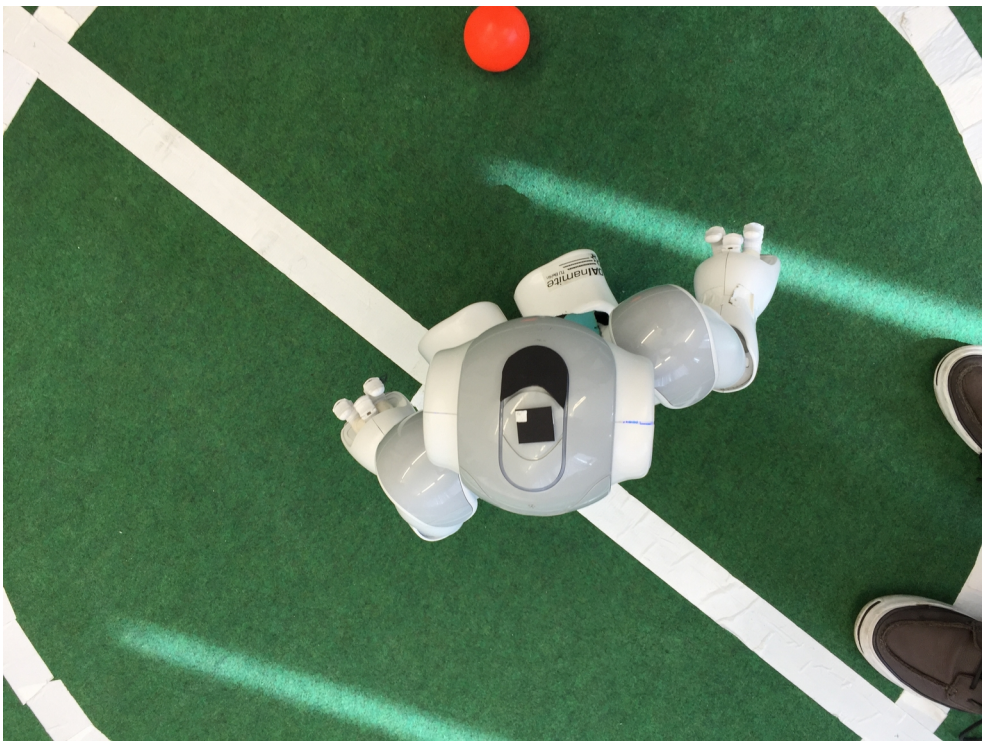


Image 4: Proposed, later discarded, solution for the robot identification

ArUco markers have the characteristic to also contain an ID, meaning that a detected marker contains its 3D coordinates
(in regards to the camera) and an identification number, making it easy to map an ID to a specific robot.

The component locating the ball relies on the assumption that the ball has a regular (by rules) and constant size and it is red in colour.

The following section will cover the components performing image processing, presenting and explaining their workflow for a better understanding.

Image 5: Example of ArUco marker

Image Processing – System Workflow

1. Main.cpp

– Clean 'field' image

– Fetch image from Left camera

– Fetch image from Right camera

– Make a copy of the camera images (for showing the unchanged version as stream, for debugging)

On both images:

- Detect ArUco markers (***Markerdetector.cpp**)
- Detect ball (***BallFinder.cpp**)
- Draw ball on the 'field' image
- Go through each marker found
- Get center and rotation of each marker
- Retrieve the team of the current player (dictionary lookup)
- Retrieve the team colour and the player's number
- Calculate and draw the player on the field
- If a movement command has been issued to a specific robot:
    * Check if target location has been reached
    * If not: show target location and line connecting it to the player
    * Send command to robot, if it has not already been sent

2. Markerdetector.cpp

– If image is not a grayscale (8UC1), convert it to grayscale (BGR2GRAY)

– Threshold image:

 # Using an adaptive threshold (ADAPTIVE_THRESHOLD_MEAN_C): the threshold value is the mean of the block neighborhood minus a given C value.

– Detect all the rectangles in the image:

- Calculate min/max size of a valid contour
- Find contours:

    # Get absolutely all the contour points (no compression/approximation)

- Check that each contour found (e.g. candidate markers):
    - Has 4 sizes (points)
    - Is convex
    - Its min size is greather than 1x10^-10
    - Its min size is greather than 10 (remove points that are too close to each other)
- Sort the points of each contour in anti-clockwise order
- Remove candidates that are too small (checking the perimeter)
- Identify each marker (get its ID) (***Arucofidmarkers.cpp**)
- Remove duplicates (if any)

3. Arucofidmarkers.cpp

– If image is not a gray-scale (8UC1), convert it to gray-scale (BGR2GRAY)

– Threshold image:

 # Uses a binary threshold (THRESH_BINARY) together with the Otsu's algorithm (THRESH_OTSU) to determine the optimal threshold value

– Check if border is black

 # Markers are 7x7 blocks with 1 black block as border (quiet zone)

– Check if inner blocks are black or white

– Check every possible rotation

 # Get the hamming distance to the nearest possible word (calculating the minimum number of errors that could have transformed a word into another)

– Convert bits (black/white) into an integer

4. BallFinder.cpp

– Convert image to HSV (Hue, Saturation, Value)
– Threshold the image
    # *Removing anything not red*
– Remove small objects from the foreground
    # *Morphological opening (erode-dilate)*
– Fill small holes in the foreground
    # *Morphological closing (dilate-erode)*
– Apply Gaussian blur (to avoid false circle detection)
– Find circles within a certain range
    # *Using HoughCircles*