Maksim Holly - 403985
Finn Harms - 400387
Sebastian Eberlein - 374794

Project Report RoboCup SoSe 2021

Project Description

Our goal was, to build a software, that allows a simulated nao-robot to find the goal on a RoboCup soccer field and navigate towards it.

Therefor we had to implement a vision-agent, that can detect the goal in the picture, given to us by the simulated camera, an interface to control the robot in the simulator, and a navigation-agent that takes the data given by vision-agent and turns it into movement-decisions.

Installation

To use our software, you first must install webots, an open-source robot simulator. You can download it using following link: https://www.cyberbotics.com/

Next you must install the naoqisim-library by following the instructions provided at this link: https://github.com/cyberbotics/naoqisim

After a restart of the webots-program you can load the *naoqisim_robocup.wbt*-world by using *File -> Open World*. You can find this file in the directory you just downloaded with the library.

After you loaded the world, you must start the Docker-Container in the naoqi-folder of our project-files. It establishes a bridge to the simulator

All that is left now, is to start our program by launching *main.py* in python 3.8.

USED LIBRARIES AND THIRD-PARTY COMPONENTS

OpenCV

OpenCV is a powerful library for computer vision. We used it to analyse the pictures we got from our camera. OpenCV can be used in different languages, including python, so we had no problem integrating it in our program.

https://opencv.org/

Webots

For the reasons which we will explain in detail in the next section, we decided to switch to webots as our nao-simulator.

https://www.cyberbotics.com/

Pynaoqi

We use the official pynaoqi library to communicate with the simulated nao robot. It also provides us with a toolset to make the robot turn and walk and to retrieve images from its camera.

https://developer.softbankrobotics.com/nao6/naoqi-developer-guide/sdks/python-sdk/python-sdk-installation-guide

Use of Webots instead of simspark

In our project we wanted to focus on the part of detecting the goal and wanted to make a little bit of research in the field of computer vision.

It turned out that the simspark-agent does not supply us with an image we could analyse, instead we would get the positions of the ball and the goal directly. That conflicted with our plan to write a program to analyse the camera-images ourselves.

So, we decided to discard our work we've done so far and switch to a different simulator, namely the Webots-simulator. It is an open-source robot simulator, for which Aldebran wrote a nao-library for. That enabled us to write a computer vision agent ourselves, as we planned in the beginning of the project.

Software-Components

Naoqi-RPC-Server

The naoqi-rpc-server runs as a separate process and acts as a wrapper for the pynaoqi library, because the library requires Python2.7, but all our other code uses Python3.

It connects to the Simulator or to a real nao robot.

Walking-Agent

The walking agent acts as a client for the naoqi-rpc-server and provides the `walk_to()` function which calls the `walk_to()` function of the naoqi-rpc-server which then calls the `moveTo()` function of the pynaoqi library.

Navigation-Agent

The navigation agent is responsible for triggering the right motion sequences of the Nao in order to turn towards the goal and walk in the right direction afterwards. The agent initializes a CVAgent and a WalkingAgent. It contains the two methods `turn_to_goal()` and `walk_to_goal()` which are called in the right order in the `run()` method. With the constant `TARGET_COLOR` it is possible to specify the designated goal color, towards the robot should walk and turn.

In `turn_to_goal()` the agent triggers the `update()` method of the CVAgent which recalculates the goal coordinates inside CVAgent by taking an image from the simulated camera. Afterwards it executes a looped procedure and turns the robot to the right direction by setting the theta angle according to the calculated goal center and executes the `moveTo()` function of WalkingAgent. It stops when the goal center is approximately 0 (with tolerance of +/- 10 pixel).

Following the previous method the NavigationAgent executes `walk_to_goal()`. In this method the goal size is being used as a measure of how far the distance between robot and goal is. The method loops through updating the CVAgent via `update()` and afterwards moves one meter forward at a time before re-calling the `update()` method. The loop stops if the goal size inside CVAgent is being set to 'None', as at this point the robot stands right on the goal line and can't see the goal barriers anymore.

The movement of walking to the goal is hereby finished.

CV-Agent

The cv-agent (cv short for computer vision) grabs the picture from the simulator and searches for the ball and both goals.

The algorithm searches for a match to a given colour by calculating the difference to the target-colour using the HSV-System. The HSV-System represents a colour using a Hue-, a Saturation- and a Value-channel instead of the typical red-, green- and blue-channel of the RGB-System.

With those three channels we can calculate the distance to a given colour for each channel so that the result will be three grayscale images, where darker colours mean, the image is closer to the targes-colour in that spot. Now those three images are added up with different weights for the H-, S- and V-channels, so we have one resulting grayscale-image with darker spots meaning, they are closer to the target-colour.

We now threshold this image to get white areas, where the similarity to the colour is over a given threshold. The rest of the picture is black.

Finally, we search for the minimum surrounding circle or rectangle (depending on if we are searching for a ball or a goal) and we get the coordinates of our Objects.