Technische Universität Berlin
Institut für Informatik

RoboCup

Rebeca Fernandez Niederacher -fernandezniederacher@campus.tu-berlin.de-
Julia Baumbach -jule.baumbach@hotmail.de-
Alessandro Bartsch -ale.bartsch@gmail.com-
Supervised by Yuan Xu

Version vom
July 18, 2017

Abstract

The following paper describes our project in RoboCup. We were asked to program a little demonstration of a topic we could freely choose. We decided to let our robot search for some different colored squares, take and save a picture of them. This paper gives a short overview of what we did and which smaller or bigger problems we had.

Contents

1 Introduction

Our first idea was to find two different object and then let both object in different target position. For example, we could have a ball and a Lego construction in the class. The robot had to find the first object, went there, caught it, went to the target position where we wanted to drop the object and after that did the same but with the second object.

But while working on the project we found out that this idea is a little bit too hard to implement in three weeks. One more problem for us was that at the beginning of the semester we had six students in our group. But after a couple of weeks one after one decided to leave this course. So, when we starting the real part of the project we were just three people left. Therefore, we decided to change our main Idea. The Robot has to search now for a orange square, save the photo and then search for a pink square and do the same. While doing his job the robot should tell us what he is doing and what he is seeing at the moment.

Now we go a little bit deeper in the details how we implemented this task and which problems we had to solve.

2 Vision

We knew from the start that we need some vision part in our code for detecting objects. After a small research we decided to use OpenCV, which was also presented in the lecture.

2.1 OpenCV

OpenCV is a open source computer vision library that has C++ and Python interfaces. It can be use for Windows or Ubuntu

2.2 Detection

For the detection we tried multiple formats like BGR, RGB, HSV and YUV. In the end we stayed with HSV, cause it was the most reliable for detecting colors. And because our objects are only detected by color it was only natural to take this one. In detail we are taking one picture from the bottom camera and one picture from the top camera. Afterwards we apply the specified color-filter and decide which camera take the best photo. Afterwards we determine if the found area is big enough to assume we found the object. We also detected some problems. For instance, we had to find out in which color range our pictures are. That was not so easy because we couldn't find something like a perfect RGB to HSV converter. Though, we found a lot, but every of these computed other results for our colors. But with a lot of try-and-error we were very good in color-detecting at the end. The only problem left was color reflection. If we take a picture from a very small angle beside the square it is not so easy to get the right color but the square seems to be very white. At the end we couldn't detect perfectly the squares if we are in very special positions but in most cases our color detection works very fine.

3 Motion

For the motion we thought, that this part would not bear any harm, cause it should come be mostly from the given code-base.

3.1 Walk

There are different method to walk, but the one that we implemented was MotionProxy.moveTo(). We use as input of the method, the target position of the robot. (

3.2 Fall Down

That should be the classical problem, when programming humanoid robots. Suddenly it loses its balance and falls down. Sometimes even if it does not move at the moment. We checked at most of our program, if the robot fell down already or if we can proceed normally. Our first solution was just to reset all and start from the very beginning. We did not see any fast solution in the API-description. Unfortunately, we ran out of time and we couldn't solve this problem because we had our focus on other things.

3.3 Grabbing objects

For our first idea we had to grab and transport objects, e.g. a red ball or a Lego construction. But after a while working with the robot we seemed big problems to handle the tasks. First of all, the hands of the robot are smaller than in our imagination. So, grabbing some objects would work just for very small objects. And that's not the only problem. One more is that the robot can't grab things from the floor because his arms are too short. First we thought that maybe we can solve this problem if we don't put things on the floor but on a very small platform where the robot just has to extend his arms forward to reach the object. But then we would have the problem that we have to stand at the very exact position where the length of the robot's arm are exactly the distance between him and the object to grab. While thinking about this problem we detected more and more big problems. For example, one more is that if we could hold the objects in our hands, maybe with both hands, than we can't be sure that safe walking is still possible. The robot has a bit of balance but we didn't know if his balance is good enough if he holds something in his streched out arms. Maybe we could fall down and loose our object. Finally, after detecting problems and more problems which didn't seemed easy to solvable for us we decided together to skip this part for simplifying our project. We wanted to focus on the vision part. This should work more or less perfect!

4 Distance

One more main task was to compute the distance between our actual position and the target's position. We need this to know how far we have to walk.

In the web we found a lot of solutions how people computed distances between actual positions and target positions. But all these stuff was very hard to implement and it was also said that this problem is one of the most complicated in computer vision. But we had the advantage that we didn't want to compute a distance to a real 3D object (which is much more complicated because you need to compute the center of the object

and its surface), but we want to compute the distance just to a 2D object which is just a square. From vision part we get an area which we detected. With a simple calculation using the rule of three. We did a calibrating-like step by computing the area size from the vision size while our square was exactly one meter far away. Now we had a relation between the distance to our square and the area size. Since then we could compute the distance for all area sizes which we computed in vision.

But we can't be sure that our square is directly in front of the robot. That's why walking the computed distance straight forward doesn't make really sense. But also computing the rotation angle is kind of complicated. We found a simpler solution. The robot should walk just half of the size which we computed. Then we try to find our square again. If we can't find it, it means that we walked not exactly to the right direction and we should search for the square again. But if the robot found it again means that it is on the right way and it can continue walking. If the computed distance is smaller than 30cm the robot stops and take a picture of the colored square. We need this "safety"-distance from 30cm because the calculations are not very exactly and we want to avoid crashes.

5 Concluding Words

Finally we really can say that this project was fun. It was really moving to see our little robot walk and talk. And we all were really joyful, when he does what he should. Sadly we did not have the manpower nor the time to finish the project we hoped for, but nevertheless we accomplished what we could.