RoboCup WS17/18

Daniel Kuske Clemens Kuske

May 16, 2018

Abstract

This is the documentation to our final programming exercise of the project "RoboCup". Our goal was to make the NAO-Robot detect a predefined object, calculate the position of that object and then point to it.

1 Our Approach

We solved this task in three steps, where the first was by far the most difficult and therefore biggest part:

1. Image Analysis: the detection of the object in one of the two camera-images and the calculation of the position of that object within the picture
2. Coordinate transformation: first calculating the position of the object in the coordinate system centered at NAO's chest, then calculating angles for the shoulder pitch and shoulder roll
3. Putting things together: writing a routine, that makes the robot react to objects, let's him point to the objects and then transfers him back to his initial state.

1.1 Image Analysis

We decided to take an object that looked like a pen. As it is round, it was easier for us to calculate the thickness of it in the picture and thus the distance to the camera. Further we chose a color, that would be easy to distinguish from other colors and is not too common. We ended up with a roll of orange paper glued together with a white tape, which we will call pen from now on.
To code the routine, that we describe in this part, we shot about 40 test images and tested our code on these images. As we shot all test images in one go, we had the same lightning conditions on all of them. It would have been better to have test images with various lightning conditions to improve our image analysis.
To detect the pen in the camera images, we transformed the image from BGR to HSV. In this color space it is easy to filter all orange pixels of the image via an upper and lower color range. To get better results in different lightning conditions we calculated the average brightness and average intensity of our test images (where we knew that we could detect the right colors) and then raised the intensity and brightness of the new images to match these values. After applying our filter we ended up with an image of black and white pixels, e.g. a bitmap where all pixels between our bounds were colored white and the rest were colored black. Of course some very light parts of our orange pen were not detected as orange and some other pixels (e.g. of the hand holding the object) were detected as orange.
The next step was to "clean up" the noise of the bitmap. We used three different functions which were: erode, dilate and computing the convex hull of a given area. First we applied a small

erosion, which means that we turned all white pixels that had a neighboring black pixel into black. Thus we got rid of small areas within our image that were wrongly detected as part of the pen. Next we applied a very big dilation, which is exactly the opposite of erosion, which means that we colored all black pixels within a range of 10 pixels to a white pixel into white. We did this to merge parts of the pen, that were detected as separate parts. To get rid of holes and indentations, we calculated the convex hull of the remaining connected white parts of the image. Later it turned out that these last two steps would also merge the two separate orange endings of our pen to one part, when the pen was too far away from the camera (and thus the two parts were too close together). To improve on this malfunction one could either lower the dilation and maybe not detect pens, which are far away, or make the range of the dilation dependent on the amount of white pixels e.g. the distance of the pen to the camera. The final step was then to apply another erosion to get the final white areas back to their original size.
Whenever there are exactly two separate parts detected as orange in the image, we assumed that these are the two endings of the pen. This does not take into account the shape of the pen and could be improved by matching the given areas to a virtual model of the pen. To calculate the position of the pen in the image, we decided to first calculate the centers of the detected areas, then calculate the width of the pen and then finally the coordinates of the pen in the image. We calculated the centers of the two areas as the average x and y coordinate of all border points of the areas. Define by l the line through the two centers, then the width of one of the detected areas can be calculated as the distance of the center point of that area to the borderpoint which is closest to the orthogonal line to l that goes through the same center point. By this calculation we ended up with the two center points and the width on both sides of the pen. Knowing the true width of our pen (in our case $14\,mm$), one can use the intercept theorem to compute the distance of the pen to the camera. with the information of the center points in the image and the information on the field of view of NAO's cameras one can compute the angles that the pen is off to the center of the camera view. Those values can be easily transformed to standard spherical coordinates, which then can be transformed to cartesian coordinates.
Initially we thought, that we would wait for the robot to detect the pen on about the same position for several images in a row (in our code named as self.image_count), to be sure that he really detected the pen and is not confused by some noise, but as the whole process of analyzing one image takes a lot of time, we ended up putting that value to one, so the robot would point to the first position he detects as being a pen.

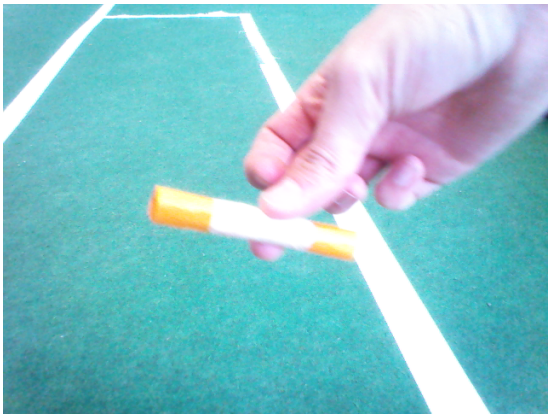
1.2 Coordinate transformation

As we learned in the lecture, we used quaternions in (4×4)-Matrix representation and simple matrix multiplication to transform the coordinates of the pen in the image to coordinates of the pen regarding to the coordinate system centered at NAO's chest. These values can then be used to calculate the position of the pen regarding to NAO's left or right shoulder. These values can then be turned back into angles for NAO's shoulder pitch and shoulder roll, to make him point to that exact position. The only difficulty is to care for the bending of NAO's arms, given by an angle of $\tan(15/105)$.

1.3 Putting things together

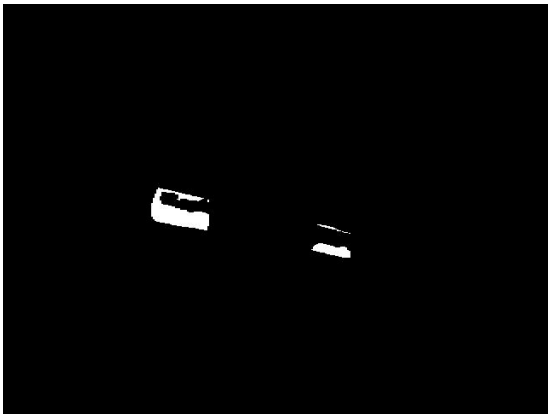
With the described parts from above it was fairly easy to write a routine, that would make the robot watch for a pen and then point to it. We initially made the NAO sit and place his arms to the side of his body, s.t. they would not be visible on either of the cameras. Next we started a loop which would always

- retrieve new images from both cameras,
- send them to the image analysis, which detects whether there is the pen in the image or not,

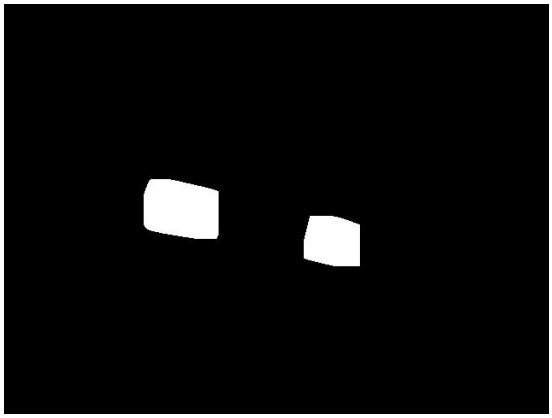


(a) The original image

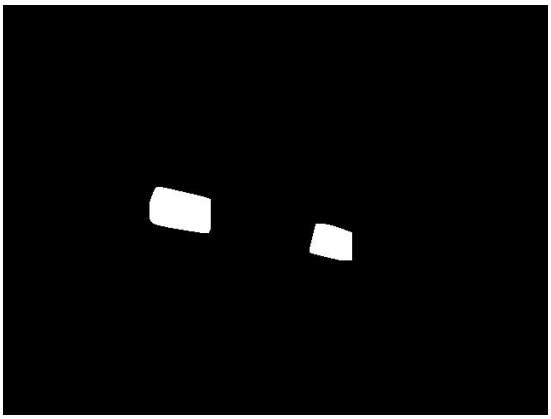
(b) The bitmap showing all pixels, that match our bounds



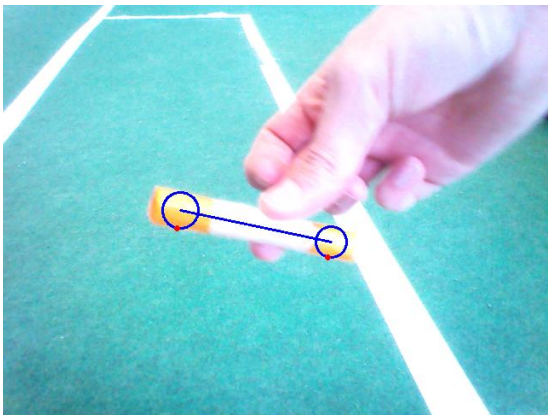
(c) Bitmap after first erosion, unwanted noise disappears



(d) By dilating and computing the convex hull, all parts of the pen are merged together

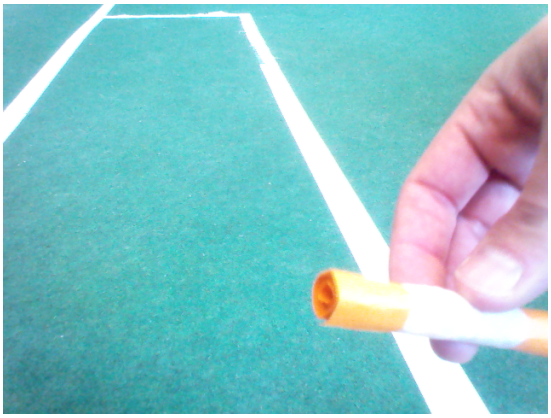


(e) Final bitmap



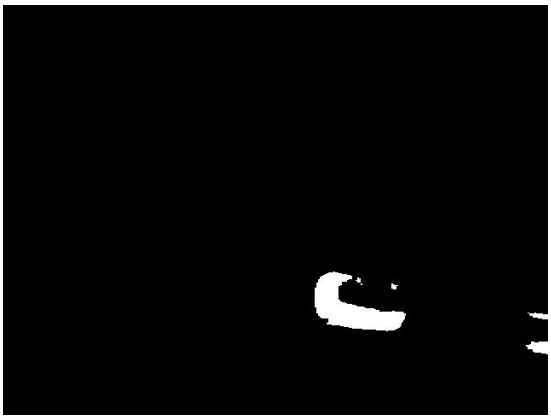
(f) The original image with the calculated values painted on it. The blue circles are centered at the calculated center points and have the radius of the calculated width. The red points denote the border points orthogonal to the blue line

Figure 1: Example of the image analysis: it manages to distinguish the pen from the white line in the background, even though the pen has a large white spot (by reflection of the sun)

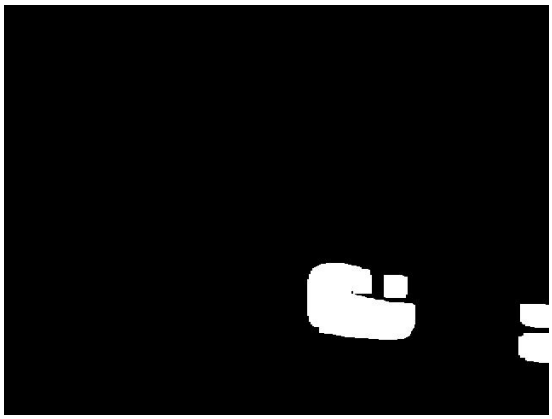


(a) The original image

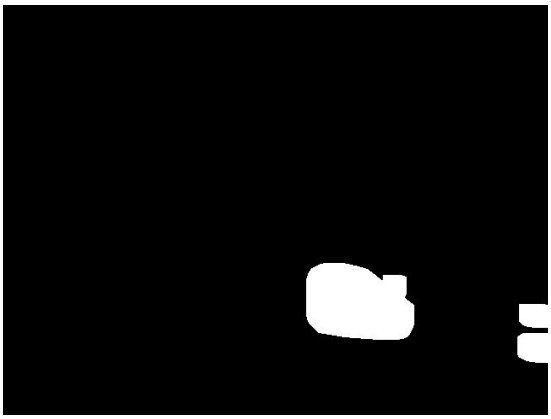
(b) The bitmap showing all pixels, that match our bounds



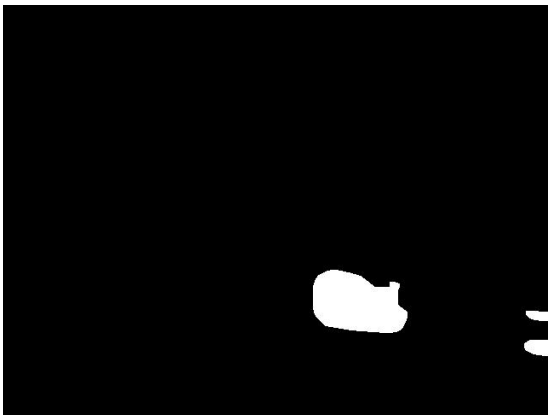
(c) Bitmap after first erosion, unwanted noise disappears



(d) The dilation is not enough to merge the seperate parts together.



(e) By computing the convex hulls of all parts, the left end of the pen gets merged to one area



(f) Final bitmap

Figure 2: Another example of the image analysis: the noise from the hand is perfectly erased, but the algorithm manages to merge only the left end of the pen. The right end is still split into two parts and thus there is no pen detected in this image.

- start from the beginning if there is no pen or the deviation to the last detected pen positions is too big, or
- get the head yaw and head pitch values and use them to calculate the coordinates of the pen to the coordinate system centered at NAO's chest.
- If the pen position is close to the former pen positions, then the robot should point to it.
- Decide whether to use left or right arm to point to the pen (is y-coordinate bigger or smaller than 0?)
- calculate the angles for the shoulder pitch and shoulder roll
- use these values to point to the pen, say that the pen has been found and go back to the initial position.

At the end of each loop we plot all data, that the image analyzer calculated as well as the original image and the bitmaps, to better understand what the robot does and why he behaves like he does. This allows us to easily debug and improve our approach.

2 Problems of our Approach and possible solutions

2.1 Lightning conditions

As described before we had problems with different lightning conditions. Our test images where all shot in one session with a specific light. The next time we tested our image analysis the used threshold included nearly the whole picture. We had some ideas on how to bypass this problem. The one we decided for is described above. We will now describe some other ideas, which we did not use in the end.

2.1.1 Movement instead of color

One big discussion after our problems with the lightning conditions was, if we should use movement to correct the result. The idea is simple, we ignore all parts of the picture where nothing has changed compared to a comparative image. Thereby we decrease the possible areas where the pen could be and thereby the area where errors could appear.
There are different problems with this approach:

Movement We implemented everything we needed to let the robot point correct even if the head moves. So if we choose to move the head we would need a comparative image that covers the whole camera space. This would be possible.

Background If we implement this function it would help if the background is static, but it would consider everything to be a pen that is moving, like some guy with an orange t-shirt, or the hand holding the pen.

The hand While the risk, that someone with an orange t-shirt is walking though the background is calculable there will most likely be a hand holding the pen. In our test images we recognized that the color of our hand is near to the color of our pen.

It's a hack It did not felt right to minimize the problem without trying to remove it. So even if we would implement this feature we would need some image analysis afterwards which still has the same problems.

2.1.2 Try different thresholds

Another idea that felt like a hack was to have a list of thresholds. For all of these we would calculate the bitmap and the possible positions of the pen. Then we could check if we find a pen with any of those thresholds and use these values. The problem of this idea is, that we most likely would always find a pen, even though there is none in the image.

2.1.3 Threshold through image analysis

This idea is a further development of the last idea and similar to what we used in the end. Instead of having hard coded threshholds, one could analyze the image and try to calculate threshholds, that would match the given lightning conditions.
We dropped this idea, because in the best case the result would not differ from changing the image itself, as we did. Additionally we can not predict if the color of our pen is related to the average values. The background could for example be black or white.

2.1.4 Make a Picture with or without the pen at the beginning of each usage and calculate a threshold for the current lightning condition

We would set up a routine on start-up, where the robot takes a picture where the robot first takes a picture of the background, then asks you to put the pen in front of him and takes another picture if something in his view changes. With these two pictures he could calculate, what values to use for the threshold.
Next to the problem, that it would be irritating to always calibrate the robot on start-up we again can not ensure that the background is not moving, therefore it is impossible to make sure that the robot sets the values according to the pen and not to the person in the background or the hand that holds the pen.

2.1.5 Machine learning

We thought about using machine learning techniques. We discarded this idea because we would have needed a lot of test images in various lightning conditions as well as a lot of time to mark the correct spots on the pictures.

2.2 Detection of the pen in image

Another problem which we had discussions on, was how to find the pen in the bitmap. Even if the threshold is calibrated perfectly to the lightning conditions there will always be some noise in the final bitmap. We had two other promising ideas next to the one we used (that did not recognizes the pen if it is to far away):

2.2.1 Pen - Model

As we described above, one could make a model of the pen and use it in combination with the raw bitmap (without any noise removal) to calculate probabilities of pen positions. One would need to calculate bitmaps for all possible rotations and positions of the pen model and compare these bitmaps with the given one. The positions, where the calculated bitmap matches the given bitmap would result in a high probability, those that do not match in a low probability. One could even improve the performance of this approach by starting with a coarse set of positions and rotations and then get less and less coarse around the spots with the highest probabilities.
The advantages of this approach is that we would directly end up with the position and rotation of the pen. The downside is, that it is extremely time intensive. Even with the described improvement one would end up with thousands of iterations. The result that we implemented has a constant running time and is therefore much more efficient.

2.2.2 Update our Implementation to draw in size of spots

An improvement to our method would be to try out different erode- and dilate-levels depending on the given bitmap. For example one could erode more if there are many separate white parts, or as described above, one could do less dilation if the amount of white pixels after the erosion is small (to avoid merging the two ends of the pen to one part).

3 Outlook

We had some other ideas, that could have been implemented in the future:

- walk to the pen if it is far away
- additionally calculate the rotation of the pen and grab it, if it is in range
- recognize pens that are divided between the upper and the lower image
- save the position of each object and ignore objects, that are on the same spot or say that it is the same object
- recognize different shapes and different colors
- detect the color of the objects and say them: "hey, i found another blue pen. That's the third blue pen."