The Roasted Tomato Challenge for a Humanoid Robot

Caitlin Lagrand, Michiel van der Meer and Arnoud Visser
Intelligent Robotics Lab, Universiteit van Amsterdam, The Netherlands
http://www.intelligentroboticslab.nl/

***Abstract*—Supporting humans in the kitchen is a difficult task for a humanoid robot. This study focuses on one of the tasks identified in the HUMABOT Challenge; finding a tomato in a kitchen environment and grabbing it from the table. Three detection algorithms are evaluated on their performance to find the tomato without confusion with other vegetables present in the kitchen. Once found, the location of the tomato relative to the humanoid robot is estimated. A planning algorithm is then used to grab the tomato from the table with both arms.**

1. Introduction

The use of robots has been increasing over the last years [1]. Not only in the industry, but also in service applications, such as health care or education [2]. This study focuses on a small benchmark to test how a standard humanoid robot can support humans in a domestic environment; inspired by the HUMABOT Challenge of 2014 [3]. In this challenge, a standardized kitchen is used as the environment where a robot has to perform the following tasks:

- The safety task: one of the burners in the kitchen is lit and the robot has to turn it off.
- The shopping list: identify missing objects on the shelves to make a shopping list.
- The roasted tomato: identify a tomato and put it into a pan.

This study focuses on "The roasted tomato" task. Section 2 will discuss previous research. In Section 3, the used methods will be explained. Section 4 will demonstrate the different simulations that were used. In section 5 the results will be shown. Section 6 will evaluate our findings and discuss ideas for future research. Section 7 will conclude this paper.

2. Previous research

Since the initiation of the RoboCup@Home competition [4] an extensive set of related work is created[1]. Yet, the work in the RoboCup@Home competition is not directly applicable to this challenge. The competitors of the RoboCup@Home have custom build robots with a dedicated sensor suite, while the HUMABOT Challenge the task has to be performed with a much smaller sensor suite. Instead, our research is inspired by two previous studies: the Cognitive Image Processing (CIP) approach from the Dutch Nao Team [5] and the work from the HUMABOT Challenge competitor NAO@UPC.

2.1. Previous competitors

Since the HUMABOT Challenge was held in 2014, the teams have no incentive to keep their code private any longer. One of the teams, NAO@UPC, made their implementation publicly available on GitHub[2]. In their qualification video[3], the team shows that they could localize objects using ORB descriptors, FlannBased with LSH matcher, Ransac-based PnP approach and Kalman filter [4]. Yet, because the tomato is not a textured object and has not many other features than being round and red, the tomato is in their code purely recognized on color and size.

Also the dotMEX team[5] recognized an algorithm based on color and size alone. This in contrast with the color-independent object detection explained in the next section.

2.2. Cognitive Image Processing (CIP)

One of the 2014 technical challenges of the Standard Platform League was the Any Place Challenge. In this challenge a soccer ball has to be found in an arbitrary environment. This challenge has to be performed with the same limited sensor suite as the HUMABOT Challenge and with an object (ball) with nearly the same characteristics as the tomato in the IKEA kitchen. The approach of Ras [6], member of the Dutch Nao Team [5], is inspired by the psychological theory of Recognition-By-Components [7]. In practice, the Cognitive Image Processing approach used a series of filters and detectors to maximize the accuracy when finding spherical objects. Since the detection is color invariant, the program would be able to find objects more dynamically. In short, it tries to find edges in a blurred image from the saturation channel of the original image. After detecting these edges, the user ends up with a binary image, with the outlines of objects highlighted if the recognition worked correctly. These binary images are then searched

1. https://robocup.rwth-aachen.de/athomewiki/index.php/Publications
2. https://github.com/gerardcanal/NAO-UPC
3. https://www.youtube.com/watch?v=AzZZ15W_RvM
4. http://docs.opencv.org/master/dc/d2c/tutorial_real_time_pose.html
5. https://www.youtube.com/watch?v=gwG6dQ-fNsc

978-1-5090-2255-7/16 $31.00 © 2016 IEEE
DOI 10.1109/ICARSC.2016.63

for any round shapes using blob detectors. Inspired by this approach, a variant of this approach, described in section 3.1.3, is evaluated in this study.

3. Method

The robot used in this research is the NAO from Aldebaran[6]. The NAO is a 57 cm tall humanoid robot. In this study, a H21 body and a V4 head is used. The environment is the standard play kitchen from IKEA, called DUKTIG, including all the tools and vegetables (see Figure 1). This research will focus on one of the vegetables: a red tomato with a green crown with a size of 5 centimeter.

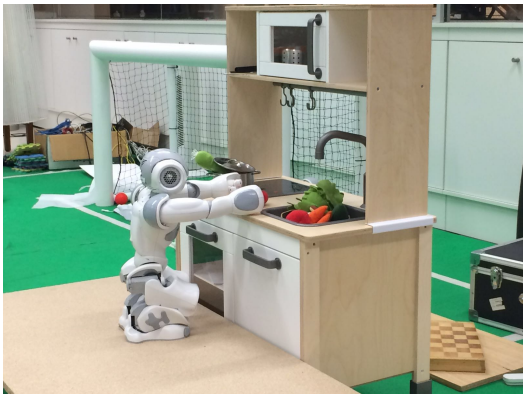


Figure 1. The kitchen environment

The ROS framework [8] was used to give access to the many tools and libraries it has. To process images, the OpenCV library [9] is used. The MoveIt library [10] is part of the ROS framework. MoveIt introduces a number of tools that aim to create developer-friendly software for mobile manipulation. This library, among other things, incorporates kinematics, motion planning and execution. All of these libraries are available in both Python and C++. The code presented with this paper is written in Python. All of our code is available at our GitHub repository[7].

The method can be described as three different phases: detecting, localizing and grabbing the tomato. This section explains the methods used for the different parts and the integration of the three phases. In section 3.1, different approaches to detect the tomato will be explained. Section 3.2 focuses on localizing the tomato and section 3.3 will explain the MoveIt implementation to grab the tomato. Section 3.4 will describe the implementation in ROS.

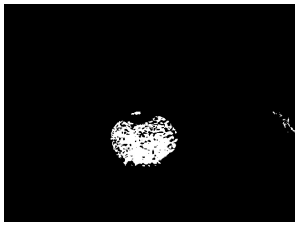
3.1. Detecting

To detect the tomato, three different approaches were used. Two approaches are based on shape and color characteristics of the tomato. The last approach is based on color-invariant object recognition.

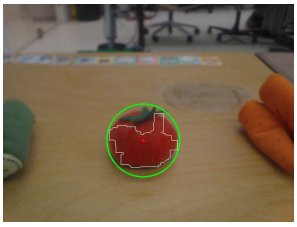
6. https://www.aldebaran.com/en/cool-robots/nao
7. https://github.com/Caiit/tomato_tracker_py

3.1.1. Color based and finding contours. The first approach focuses on the color of the tomato. Because the tomato is red, the image can be adjusted to filter out everything that is not red (see Figure 2a). This is done by transforming the image into the HSV color space and using the threshold $(0, 140, 60) - (3, 250, 250)$ to get the binary image of red. Keep in mind that this range depends on lighting as well as the camera itself, so this threshold has to be calibrated [11].

After removing the noise by using OpenCV's erode and dilate function, contours were found with `findContours()` from OpenCV. From those contours, the minimal enclosing circles were found, because a tomato is more or less a circle (see Figure 2b). From these circles, the "pixel-coordinates" of the center of the tomato and the radius were obtained. The tomato is detected when the radius is bigger than 20 pixels to filter out small red objects in the background. This limitation was put in place, because the surroundings of the robot are not filtered in any way. This means that its camera will pick up anomalies which are not related to the task at hand.



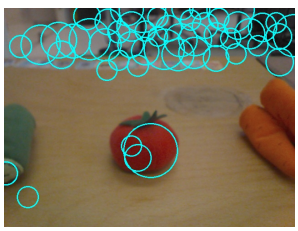
(a) Binary image of red pixels



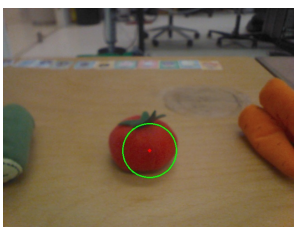
(b) Detecting the tomato with `findingContours()`

Figure 2. Color based detection in combination with finding contours

3.1.2. Circle based and average color. The second approach is based on the fact that a tomato is almost round. After transforming the image into the HSV color space and detecting edges with the Canny Edge Detector, circles were detected in the image using `HoughCircles()` (see Figure 3a). The next step is to calculate the average color of the found circles. If the average color is in the range $(20, 20, 70) - (55, 70, 200)$, the circle is considered as red and thus as the tomato (see Figure 3b).



(a) Detecting circles using `HoughCircles()`



(b) The best circle with red as average color

Figure 3. Circle based detection in combination with average color

The range for red used in this method differs from the one used in the color based approach. This is because after a circle is detected, some small areas inside this circle might not be red, but for instance part of the crown. Because the circle detector has to find a perfect circle and the tomato is not, the color of these areas is weighted into the average color. This makes it not the same shade of red as the tomato, but a slightly adjusted color.

3.1.3. Color invariant and blob detection. The last approach is color invariant. It is inspired by the CIP algorithm [6], with some simplifications (because no square objects had to be detected). It uses blob detection to detect blobs in the image. In this process, the raw image is parsed to OpenCV's `SimpleBlobDetector()`[8], which will then perform blob detection on the image. These blobs are defined as regions in the image that differ from surrounding regions. The algorithm performs four steps:

1) Thresholding: Converts several times the image to a binary image, each image containing colors that are inside the thresholds as white, and the colors outside the thresholds as black. This is repeated multiple times with slightly other thresholds, until the entire image is divided.
2) Grouping: For each binary image, the white pixels are grouped together. The centers of these blobs are calculated.
3) Merging: If blobs show up across multiple binary images and their centers are close together, they are considered the same blob and are merged together.
4) Filtering: Per blob-group, which can span multiple binary images, several characteristics are calculated. In this case filtering is performed on characteristics as area, circularity and convexity. The center and radius of the remaining blob-groups are then returned to the caller.

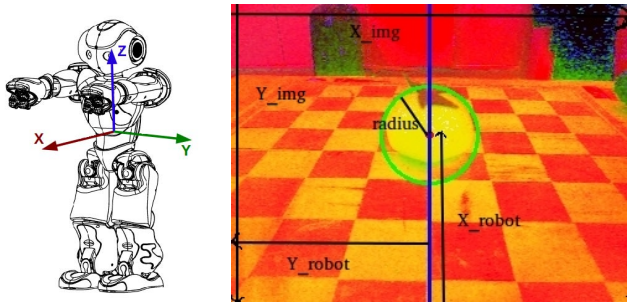
In the last step it is possible to put constraints on what the blob looks like, which is comparable with the Geon matching described in Cognitive Image Processing approach of Ras [6]. For the tomato detector, a largely circular blob would be accepted and a quadrilateral shape (quad) would be rejected.

Figure 4. Detecting the tomato using the blobDetector

3.2. Localizing

To localize the relative position of the tomato to the robot a number of coordination transformations are needed. The correct procedure is to calibrate the camera, to use a pinhole model to convert this 2D image point to a 3D vector (using the focal length found during calibration) and to transform this vector to the body

8. http://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html



(a) The robots coordinates system (b) The different orientations shown in one image

Figure 5. The different orientations from the robot and the image

frame based on the gaze message which is broadcasted by naoremote with the geometry package **tf** available in ROS (as indicated in the tutorial section of [12]).

Instead the relative position is estimated with an approximate method. Since it is known that the target object is at a fixed height in the area, a simpler method was both easier to implement and faster to work with. This could be later on replaced by a more advanced method. The robots starts out receiving images from his top camera. As long as there is no tomato detected, the robot will turn to the right. This simplified approach might cause issues in a more dynamic environment, where the table is further away from the robot's starting position ($> 1m$). At that point, even when the target object comes into vision, either the resolution of the camera would make the object too small to detect or the detector would not consider the generated output as a valid target.

If it does manage to detect the tomato in an image, the x,y-position and the radius of the tomato are found. Now this position needs to be converted to real-world coordinates with the robot as origin (see Figure 5). The z-coordinate is fixed, because the height of the table is known: 0.35m. This is 0.5m from the center of the robot coordinate system.

3.2.1. Finding the x-coordinate in the robot orientation. The x-coordinate of the robot is the distance between the robot and the tomato. This is calculated by determining what the radius of the tomato is in an image with the tomato lying on a distance of 0.3 meter. The radius was 36 pixels. The radius for a distance of 1 meter can be derived from those values as shown in eq. (1).

$$radiusToMeters = 36/0.3 \tag{1}$$

$$distanceX = radiusToMeters/radius \tag{2}$$

This constant can now be used to calculate the x-coordinate (eq. 2).

3.2.2. Finding the y-coordinate in the robot orientation. When drawing a vertical line in the middle of the image, the y-coordinate is the offset from this line to the center of the tomato. There are two possibilities: if the tomato is left of this line, the y-coordinate is positive; if it is on the right side, the y-coordinate is negative.

$$offsetInPixels = (imgWidth/2) - x \tag{3}$$

$$pixelToMeter = realWidth/(radius * 2) \tag{4}$$

$$offsetY = pixelToMeter * offsetInPixels \tag{5}$$

To get this offset in pixels, the x-position of the image is subtracted from this line (eq. 3). Now those pixels must be converted to meters. This is done using the real width of the tomato, which is 0.05m (eq. 4). Multiplying this with the offset in pixels will result in the y-coordinate of the robot (eq. 5).

3.3. Grabbing

Grabbing the tomato is done by MoveIt. As mentioned before, this library contains a number of tools for mobile manipulation. Only the left and right arm groups are used. Given the center point of the tomato, MoveIt first moves the arms of the robot 5cm to the left or to the right of the tomato, but it does not grab it yet. This is done to correct any possible localization errors. Next, both arms are moved towards each other with 2cm to squeeze the tomato to grab it. Finally, the arms are lifted up a bit and moved towards the chest to avoid hitting the table and to be more stable while moving.

3.4. Implementation in ROS

The ROS-implementation consists of four nodes (see Figure 6). The main node is the TomatoDetector node. This node is a subscriber to the Image topic and a publisher to the Point and Pose2D topics. The Camera and Walker nodes start while running the nao_bringup for the connection with the NAO. Those nodes are used for the images and to walk. The MoveIt node is used for inverse kinematics to grab the tomato.
Our ROS implementation (based on the Long Term Service release ROS Indigo Igloo) for the NAO depends on the following packages:

- nao_robot
- vision_opencv
- naoqi_bridge
- nao_moveit_config
- nao_dcm_robot
- nao_virtual

3.4.1. Component integration. The Camera node publishes images taken by the NAO. The TomatoDetector is subscribed to those images, so it receives them. Firstly, it tries to find the tomato in the image. If it does not find the tomato, it turns right and searches again. If it does find the tomato, it checks whether it is close enough to grab it. If so, it publishes a Point and the MoveIt node will start grabbing the tomato. However when it is not close enough to grab it directly (indicated by MoveIt node that no solution is possible), the robot Walker receives a Pose2D to move closer to the tomato. This Pose2D has the obtained x-coordinate of the tomato minus 0.3m as its x-coordinate, otherwise the robot will hit the table. The y-coordinate is the y-coordinate of the tomato. After moving closer to the tomato, it will search for the tomato again to check if it is now close enough to grab the tomato. This is done to compensate for possible errors during localization.

4. Simulations

Although the image processing is based on the interface with the real NAO robot, most of the planning can better be tested with simulations, to prevent the effect of real collisions. For the implementation described in this paper, the interface to two different simulators were used: Webots & RVIZ.

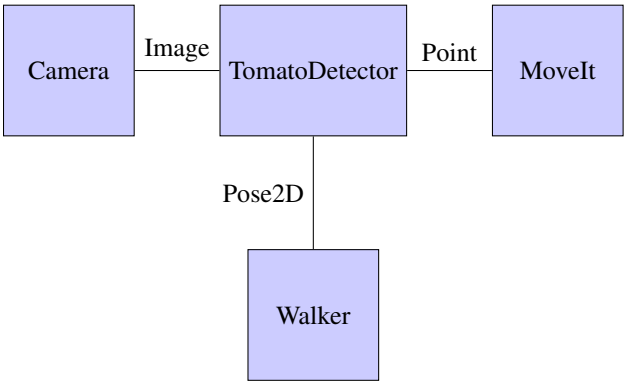


Figure 6. The implementation in ROS showing the used nodes and topics

4.1. Webots

Webots is a commercial robot simulator promoted by the HUMABOT organizers. They made the working environment available in a format that can be used in Webots, such that developers that do not have access to physical robots can still practice programming one. However, it became apparent that in order to work with Webots, a new controller for our robot would have to be written, because this application could not be done with the standard Webots ros-controller. This would make porting of the experiments directly to the NAO robots difficult. Therefore, Webots was dropped as a simulator, although the 3D environment developed by the HUMABOT organizers was ported to RVIZ.

4.2. RVIZ

RVIZ is not an actual simulator, but a 3D visualization tool integrated with ROS. The program loads configurations that define what kind of objects are in the world and gets the data from ROS nodes. By simply listening to the broadcast data it is then able to show in what way everything is moving. By sending back data, for instance two objects colliding, it sends information to the relevant nodes. The MoveIt node, with the inverse kinematics of the NAO robot, is a fully integrated plugin of RVIZ.

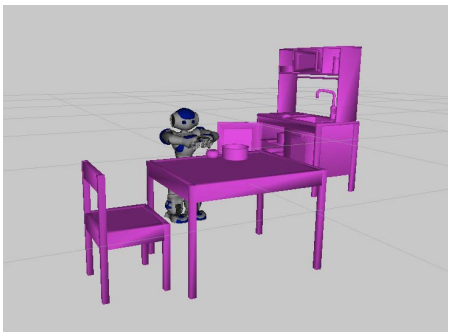


Figure 7. A screenshot of the RVIZ visualization

Due to the architecture of different nodes, controllers are not necessary. Instead, it is possible to pretend that a real robot is sending information to RVIZ. This way, while RVIZ thinks a real robot is connected, it is just receiving broadcast data from a robot that is running virtually. By design, it is then also possible to

actually plug in a physical robot, and have the program execute without significant changes, aside from recalibration.

One drawback from RVIZ was that the kitchen environment needed for the experiments was not readily available, and had to be converted manually from the Webots environment. This meant exporting all the objects to a format that was readable by RVIZ with Blender and Meshlab[9]. After converting, all object lost their original position and had to be manually readjusted. Care has been taken that also the collision models, needed to predict physics-based effects, were correctly adjusted.

With RVIZ working and the robot connected, it is now possible to simulate the robot grabbing a tomato at a specified point, since MoveIt is also controlled using the ROS system (see Figure 7). The specification of the location of this point would be combining by the detector program described in section 3.1 and with the estimation of the location described in section 3.2.

5. Results

The results described in this section are split into three parts. This is because these three parts form a core to our study. If one part was unreliable, the other would suffer from it. Firstly, the tomato detectors are evaluated in section 5.1. Then, with the best performing detector, the localization algorithm is evaluated in section 5.2 and section 5.3 will go into detail about the results of grabbing the tomato.

5.1. Tomato detector evaluation

The three detection algorithms are tested on ten different images: five images containing only one vegetable (tomato, carrot, cucumber, garlic or lettuce), three images containing all five vegetables and two images containing all vegetables without the tomato. Table 1 shows the results of the different detection algorithms. "1" means that it detected the tomato and "0" means that it did not detect anything. If something else was detected, it will say what this was. As can be seen in this table, the color based method always detects the tomato and does not detect anything else. The circle based approach does detect the tomato most of the time but it detects something in the background as a tomato once. The blobs based method detects the tomato most of the time, but detects a lot of other vegetables as the tomato as well. The color based method is the best method and is used for the rest of study.

TABLE 1. RESULTS OF THE DIFFERENT DETECTING ALGORITHMS
1 MEANS DETECTED TOMATO, 0 MEANS DETECTED NOTHING

	color based	Circle based	Blobs
tomato	1	1	1
carrot	0	0	0
cucumber	0	0	0
garlic	0	background	0
lettuce	0	0	lettuce
all1	1	1	1
all2	1	0	garlic
all3	1	1	1 & background
without1	0	0	garlic
without2	0	0	carrot

9. https://www.blender.org/ & http://meshlab.sourceforge.net/

5.2. Tomato localization evaluation

In order to create a sensible evaluation, a chessboard was used to make sure the tomato was put in the same place every time. The squares on this chessboard are each of a fixed length, making it very easy to create a discrete space on the surface. The Nao robot was on a fixed position, only the location of the tomato was varied (on the chessboard). Table 2 shows the difference between the detected point and the actual point of the tomato. As can be seen in this table the localization's estimation are not work. However when looking at the differences, the errors are so small that the can be compensated by the opening the gripper of the NAO wider during the gripper procedure.

TABLE 2. RESULTS OF THE LOCALIZATION ALGORITHMS, IN METERS

Real x	Real y	Estim. x	Estim. y	Diff. x	Diff. y
0.22	0	0.23	-0.006	0.001	-0.006
0.22	0.076	0.27	0.077	0.046	0.001
0.22	-0.076	0.27	-0.076	0.046	0.000
0.30	0	0.31	0	0.009	0
0.30	0.076	0.31	0.062	0.009	0.014
0.30	-0.076	0.39	-0.071	0.086	0.005
0.38	0	0.37	0.001	0.004	0.001
0.38	0.076	0.37	0.061	0.004	0.015
0.38	-0.076	0.54	-0.076	0.164	0.000

5.3. Tomato grabber evaluation

Getting MoveIt to run on a physical NAO proved more difficult than anticipated. Even with the ROS interface being able to easily swap between a simulation and a real NAO, the actuators that are controlled by MoveIt did not correspond directly with the name of the same actuators on the physical NAO. This resulted in being unable to run a complete test including the tomato grabbing, without a renaming scheme.

However, the simulation did show great results. The animation was able to grab the tomato after specifying the location the tomato detector thought it was at.

6. Discussion

Even though the localization seemed to work within a certain error margin, future research might want to implement a more sophisticated process. Our simple approach worked because we optimized the algorithm to our environment. The coordination transformation could be calculated with a more advanced routine, but the major drawback of our method is that distance was estimated based on the perceived size (which is sensitive to shadows at the side of the object). When looking at the future, a new method could be suggested where the x distance from the robot is calculated with a distance estimate derived from multiple viewpoints.

7. Conclusion

Multiple object recognition methods were explored, implemented and tested in this study. The color-invariant detector does not work nearly as good as the color-based detectors and was no longer considered to be integrated in the task. Using the best of the color-based detectors, the location of the tomato relative to the robot was estimated. The location was fed to an inverse kinematics

solver, which would then be able to direct the two arms to a location where they could grab the tomato; ready to put it into a pan. We view this as the first steps to a humanoid robot that can roast a tomato.

References

[1] E. Garcia, M. Jimenez, P. De Santos, and M. Armada, "The evolution of robotics research," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 90–103, March 2007.

[2] L. Iocchi, T. van der Zant *et al.*, "Advances in domestic service robots in the real world," *Journal of Intelligent & Robotic Systems*, vol. 76, no. 1, p. 3, 2014.

[3] P. J. S. Enric Cervera, Juan Carlos Garcia, "Toward the robot butler: The humabot challenge," Robotics & Automation Magazine, IEEE (Volume:22, Issue: 2), 2015.

[4] T. van der Zant and T. Wisspeintner, "Robocup X: A proposal for a new league where robocup goes real world," in *RoboCup 2005: Robot Soccer World Cup IX*, ser. Lecture Notes in Computer Science, A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, Eds., vol. 4020. Springer, 2005, pp. 166–172.

[5] P. de Kok, D. ten Velthuis, N. Backer, J. van Eck, F. Voorter, A. Visser, J. Thomas, G. D. Lopes, G. Ras, and N. Roos, "Dutch Nao Team - team description for RoboCup 2014 - João Pessoa, Brasil," June 2014.

[6] G. E. Ras, "Cognitive image processing for humanoid soccer in dynamic environments," Bachelor thesis, Maastricht University, June 2015.

[7] I. Biederman, "Recognition-by-components: a theory of human image understanding." *Psychological review*, vol. 94, no. 2, p. 115, 1987.

[8] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[9] G. Bradski *et al.*, "The opencv library," *Doctor Dobbs Journal*, vol. 25, no. 11, pp. 120–126, 2000.

[10] S. C. Sachin Chitta, Ioan Sucan, "Moveit!" Robotics & Automation Magazine, IEEE (Volume:20, Issue: 1), 2012.

[11] D. Soest, M. Greef, J. Sturm, and A. Visser, "Autonomous color learning in an artificial environment," in *8th Dutch-Belgian Artificial Intelligence Conference (BNAIC'06)*, 2006, pp. 299–306.

[12] M. Llofriu, G. Tejera, A. Barrera, and A. Weitzenfeld, "A humanoid robotic platform to evaluate spatial cognition models," in *Proceedings of 13th IEEE-RAS International Conference on Humanoid Robots*, 2013.