Robotics

BounceBOT

PiCar-X Kicks A Ball

Team Lost & Found

Jelmer Prins
s2773058

Julius Hendrix
s1825216

Julia Huber
s3291987

Ernest Vanmosuinck
s3210359

Daniel Siegmund
s2999404

Taught by
Erwin M. Bakker

Teaching Assistant:
Hainan Yu

May 4, 2023

Abstract

Interaction with robots is no longer a rarity. Often the main purpose for Human Robot Interaction is simply entertainment, which can be a useful tool to promote a healthy lifestyle, for example by encouraging stress relief and socialization. The robot presented here was developed for entertainment by applying playfulness. The so-called *BounceBOT* is a modified PiCar-X that plays ball with the interacting person. The BounceBOT detects a ball rolling in its direction, intercepts the ball by moving in front of its trajectory and pushing the ball back to a marker. The ball is detected with a camera and a simple color-based detection algorithm using OpenCV. The BounceBOT intercepts the ball by driving forwards and backwards, trying to keep the ball horizontally centered in the camera frame. The ball is pushed using a solenoid-powered, extending plate which is attached to a servo. This servo is directed towards an *ArUco* marker, which is also detected by the camera.

The BounceBOT is able to consistently position itself in front of a brightly colored rolling ball and push it towards an ArUco marker. Limitations of the robot's performance include the narrow field-of-view of the PiCar-X camera module, timing of the push due to the short stroke of the solenoid and varying ball speeds, and deviation from a straight path when moving due to the construction of the PiCar-X.

Contents

1 Introduction

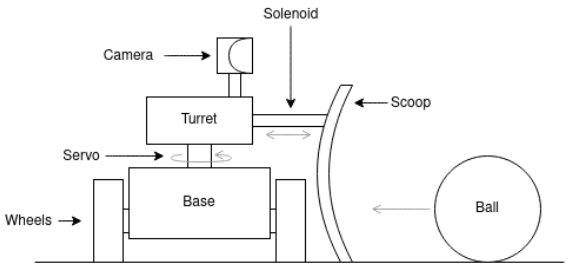
Human-robot interaction is playing an increasingly important role, and in the process, societal demands and expectations for human-interactive robots have increased in recent years [1]. One of the most important values for participants' attitudes toward robots is entertainment and thus playfulness [2]. Perceived playfulness is described as "the feeling of curiosity and pleasure" and therefore associated with more relaxation [3]. This report presents a modified PiCar-X robot whose main function is to play a ball game with an interacting human. It is able to detect a ball rolling in its direction, intercept the ball by moving in front of its trajectory and push the ball back to a marker.

First, the design and construction of the BounceBOT is discussed in Section 2. Second, the implementation of the ball- and marker detection algorithms, as well as the decision making algorithm is described in Section 3. The performance of the BounceBOT is presented in Section 4, which are then discussed in Section 5. Finally, some conclusions are drawn in Section 6.

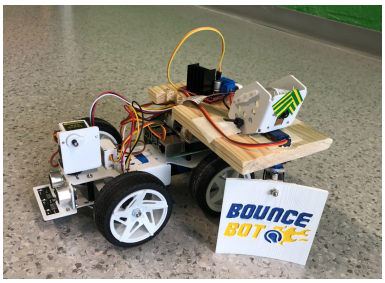
2 Design

A concept schematic of the BounceBOT can be seen in Figure 1a. It shows a movable base driven by wheels, on top of which sits a rotating turret. On this turret, a camera and a solenoid-powered scoop are mounted. In operation, the robot would move forwards or backwards to position itself in front of a ball rolling towards it, rotate the turret to aim at a target and kick the ball in the target's direction.

The PiCar-X robot[4] is used as the basis for the construction of the BounceBOT. The relevant features it offers are: a Raspberry Pi Camera Module V1 mounted on a 2-axis movable head, rear-wheel drive, and a Raspberry Pi 4 with custom HAT to interface with the hardware. The original design had to be adapted to work with the PiCar-X. The most stable place to mount the turret to is on top of the Raspberry Pi and HAT. However, because the solenoid used to actuate the scoop has a very small reach, it was decided to move the servo and solenoid from the center of the robot to the left; this way the scoop does not run into the rear wheels of the robot when it turns. The servo and solenoid, which together with the scoop form the turret, are mounted on an extending piece of wood which had holes drilled to interface with the screw holes that hold the Raspberry Pi and HAT. It was also decided to mount the camera module on top of the wooden extension. This was done because the camera does not need to



(a) First schematic drawing of the BouceBOT.



(b) Assembled robot

Figure 1: The design of BounceBOT

rotate with the turret for the robot to function, making for easier construction. The solenoid is powered directly by the batteries of the robot, and is triggered by a relay controlled by the Raspberry Pi. The scoop was printed with a 3D printer and has slight rounding on the edges in order to deflect the ball to the center of the scoop in case the turret is not perfectly aligned with the trajectory of the ball. To give the robot some character, the cup contains the logo of the BounceBOT, painted in yellow and blue.

3 Implementation

3.1 Ball Detection

For the ball detection a simple color-based detection using OpenCV was chosen. This worked well for our situation since we were able to choose the ball our self and thus were able to use colors that were not otherwise present in the environment. For a previous assignment we worked with the same robot with a MobileNetV3 implementation for COCO and we knew this would result in a worse framerate and possibly some false positives or false negatives. This further incentivised us to use a more simple color detection.

The color detection was performed by taking the frame from the camera and turning it into a HSV format, as this color format is able to detect different color more easily. We then create a mask to filter out the color we want to detect using two color bounds. Those colors are used to see if any section of the frame falls between those bounds. Once all the colors have been filtered out by the color mask, we can get all the bounding boxes in the image using function `findCountours`. We then simply assume the largest bounding box to be our ball, thus giving us its size and position in the frame. This is required as there can still be noise in the image detection, especially since color is so easily affected by light and camera perception.

The lower and upper bounds were determined using the HSV colormap shown in Figure 2. Our implementation tested 8 different colors (one for each of the balls we possessed); red, purple, pink, blue, green and yellow, with pink being the most efficient detection. We realized that the color detection was very sensitive not only to light, but also the environment, which made it hard to key out certain colors. In a perfect

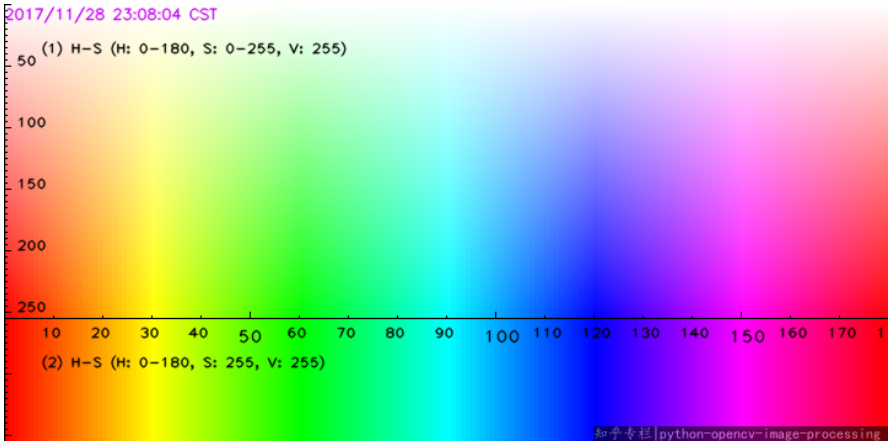
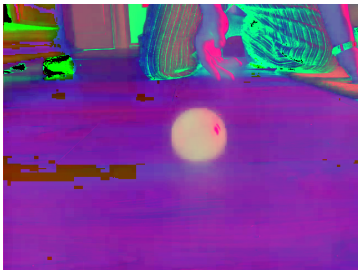
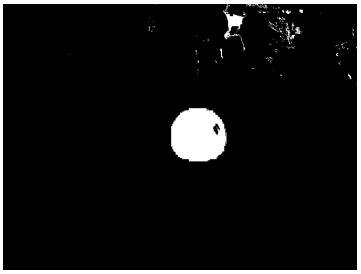


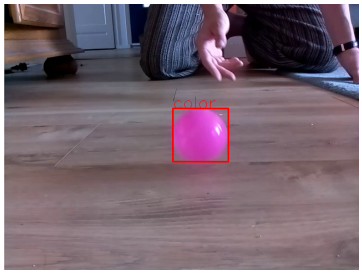
Figure 2: HSV colormap. OpenCV caps the saturation to 180 (normally 360).



(a) HSV Transformation



(b) Color mask on frame



(c) Ball bounding box

Figure 3: Frames at each step of the color detection procedure

environment (e.g.: uni-color/white background), the color detection would find the ball more easily. The different steps of the color detection can be seen in Figure 3.

3.2 Target Detection

The target for the BounceBOT to aim at is an ArUco marker [5]; an easy to detect binary fiducial square. More specifically, the marker chosen for the demonstration of the BounceBOT is the 4×4 marker corresponding to the number 6. This marker was chosen because it is fast to detect, and easy to estimate the pose of the ArUco marker with respect to the camera. ArUco detection and pose estimation is easily achieved using OpenCV, and only required the calibration of the camera module using a series of images of a chessboard pattern taken with the camera module.
Once the pose of the ArUco marker is estimated, the horizontal offset of the center of the marker from the center of the camera frame, together with the estimated distance is used to calculate the angle that points the turret towards the ArUco marker. This is so easily achieved because the camera module is mounted directly on top of the rotation axis of the turret. A screenshot of the BounceBOT aiming at the ArUco marker can be seen in Figure 4

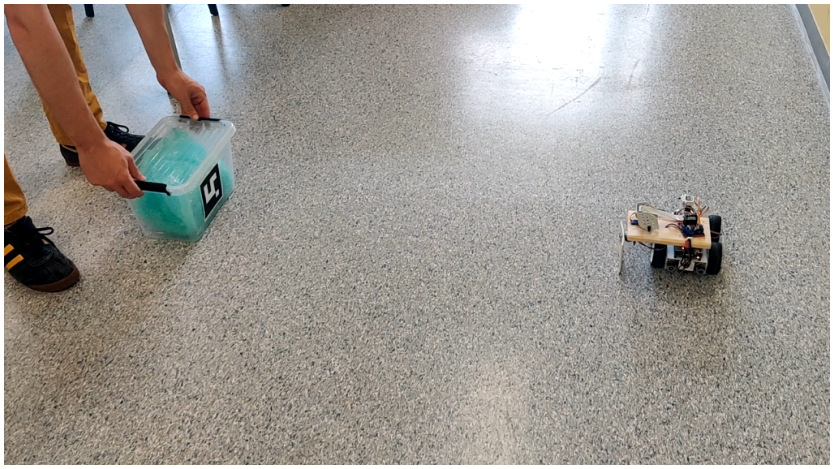


Figure 4: BounceBOT camera catching focus point

3.3 Decision Making

Based on the position of the ball in the image-space we had to make a decision on what the robot should do. Because we were able to mount the camera in-line with the cup this was a fairly simple task. If the ball was more to the left of the image we had to move to the left, if the ball was more to the right we had to move to the right. We did add some simple offsets and a "deadzone" in the middle of the image for calibration and a slightly more smooth experience.

The decision for when to fire the solenoid also came from the position of the ball in the image. When the ball reached a certain threshold from the bottom of the image we send a fire command after a certain set delay. This threshold and fire delay were calibrated to work as well as possible in most situations but for optimal performance they should have been dependent on the speed of the ball but we were unable to measure this.

4 Results

It is hard to quantify the quality of our final robot design since we are unable to get a consistent test environment and since we don't have anything to compare to. The best way to see the result is in the video we included with the report. A screenshot of the BounceBOT in action can be seen in Figure 5.

In general, the robot is able to consistently position itself in the trajectory of the ball, with the exception of the case when the ball moves outside of the field-of-view of the camera module. The turret is consistently aimed at the ArUco marker, as long as it also is withing the field-of-view of the camera module. The robot has some difficulty with the timing of the push, because this is dependent on the velocity of the ball.

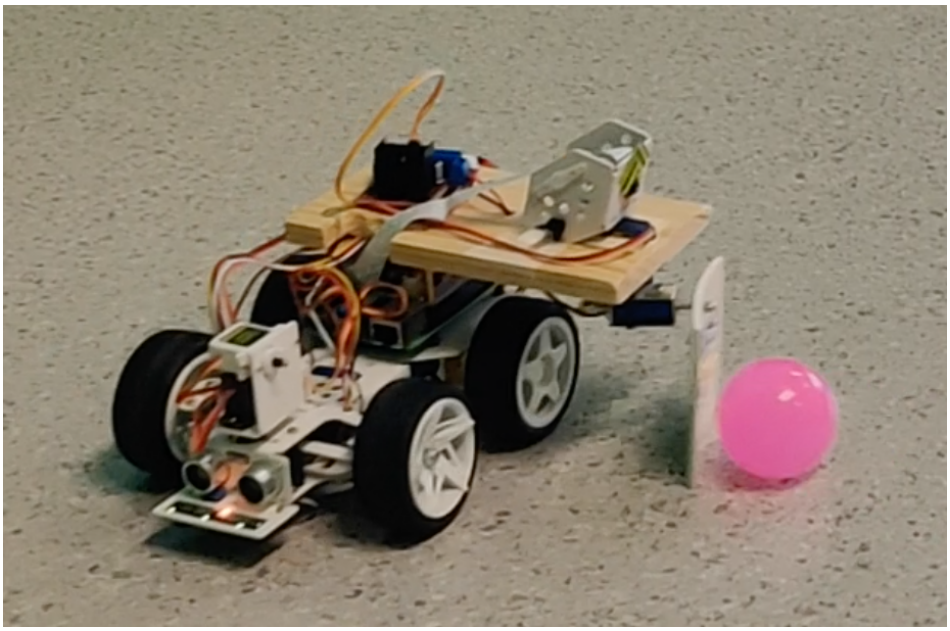


Figure 5: BounceBOT kicking ball with cup

5 Discussion

Overall there are many things we could improve on. These are mostly hardware limitations but would then of course also be paired with software improvements.

The simplest upgrade would be a camera with a wider angle lens. This would allow the robot to see more to the sides meaning it would be able to intercept balls that follow more extreme trajectories. This would however change the angular size of the ball quite drastically based on position so we would have to find a balance between not too narrow and not too wide. The current camera however was clearly way too narrow.

Another solution that would be interesting to investigate would be the addition of another camera on the robot, perhaps positioned on the other side. Adding another camera feed would give the ability for "accurate" depth calculation, providing better calculation for triggering the solenoid. This feature could however encounter processing limitations with the current Raspberry Pi installed on the PiCar-X.

To more consistently push the ball with the correct timing, velocity estimation could be done in the software using the position of the ball in subsequent camera frames. This is however not straight-forward, as the position of the ball in the camera frame is not only dependent on its velocity, but also on the velocity of the robot, which cannot directly be measured.

To further improve the push of the ball, a solenoid with a longer reach could be used. An alternative is a gearing system which would increase the reach of the current solenoid. With more range, the accurate timing of the push becomes less critical to successfully push the ball. It may also allow the ball to be pushed further, as it will accelerate more during a longer push.

Currently, we have found that the PiCar-X is not very precise when it comes to driving and turning, mostly caused by slipping from the tyres and large tolerances in the steering construction of the front wheels. A solution would be to mount the system on a rail where the robot would only move only that axis. This would constrain the movement of the robot, and also allow for larger acceleration of the robot.

6 Conclusion

The BounceBOT is able to consistently position itself in front of a brightly colored rolling ball and push it towards an ArUco marker. The main limitation for the robot is the narrow field-of-view of the PiCar-X camera module. The short stroke of the solenoid and varying ball speeds make it difficult for the robot to time the push correctly. It also deviates from a straight path when moving due to the construction of the PiCar-X.

References

[1] N. Yamaguchi and H. Mizoguchi, "Robot vision to recognize both face and object for human-robot ball playing," in *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, vol. 2, pp. 999–1004, IEEE, 2003.

[2] A. Chowdhury, "Exploring the user needs and experience of the university guidance robot," Master's thesis, Tampere University, 2019.

[3] Y. Song, M. Zhang, J. Hu, and X. Cao, "Dancing with service robots: The impacts of employee-robot collaboration on hotel employees' job crafting," *International Journal of Hospitality Management*, vol. 103, p. 103220, 2022.

[4] Sunfounder, "Play with python." https://docs.sunfounder.com/projects/picar-x/en/latest/python/play_with_python.html.

[5] OpenCV, "Aruco marker detection." https://docs.opencv.org/4.x/d9/d6d/tutorial_table_of_content_aruco.html.

[6] Sunfounder, "Download and run the code." https://docs.sunfounder.com/projects/picar-x/en/latest/python/python_start/download_and_run_code.html.

[7] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[8] "Numpy." https://numpy.org/.

[9] P. Community, "Pygame." https://www.pygame.org/news.

Appendix A: Links

- Github repository: https://github.com/Phobos97/RoboticsObjectDetection
- Demo video: https://drive.google.com/file/d/1hH8Ehn9sVmJ9oHzR1NyiAWUUBIOkXqkb/view?usp=sharing

Appendix B: Build and run instructions

The algorithms implemented in this are solely dependent on the standard packages required by and provided by SunFounder's Picar-X. Installation instructions for the Picar-X setup can be found on Sunfounder's website [6].

Notable packages are OpenCV [7], NumPy [8] and Pygame [9] (for manual control).

To run the main script with a red ball, simply use the following command:

```
python Main/bounce.py --rendering --ball_color red
```

The command posses two parameters. If you would like a live camera feed of the robot camera, use `--rendering`, but we do not advise using rendering when actually testing the robot, as it makes the object detection much slower. Additionally, you can specify a different ball color using `--ball_color "color"`, replacing "color" which one of the available options (blue/green/yellow/red/pink/purple).