Final Report

Autonomous Person Following with an iRobot Create 2 and Intel RGB-D Camera

Mykal Bakker-Westeinde 96213699

Ryan Stanley 88977897

Kim Nebe 94377108

Abstract

The first industrial robot was invented in 1954 [1]. By now, industrial robotics is a huge research field with many applications in several fields. Beyond this, the application of robotics in households has grown significantly - Robotic vacuum cleaners made their way into our houses years ago [2]. This project is our first approach to create a robot that can support a person at home.

In this project for our MECH 464: Industrial Robotics course, we turned an iRobot Create2 in combination with an Intel RealSense Depth Camera into an autonomous person follower, inspired by a project done by JetsonHacks [3]. Our robot is capable of following an individual with red pants around any room, so long as there are no obstacles between the person and the robot. A video of our fully implemented robot can be seen here.

In this report, we will explain the details of our successful project. We will comprehensively cover the process and reasoning behind our project, as well as the means we used to get to our final product. This includes our code, the challenges we faced, the steps we undertook, the results, our conclusion and our future ambitions.

Table Of Contents

1 Motivation

We were tasked to use an iRobot Create2 Robot and an Intel RealSense Depth Camera to develop a robot that could fulfill a specific task that could be useful in real life. We came up with several ideas, such as SLAM mapping a room and detecting a certain object in a room. We eventually decided to explore a topic related to collaborative robots, and developed a cobot that would follow a person at a set distance. To achieve this, we divided the project into three main parts as the robot has to be able to detect a target, localize this target, and adjust its own position according to the position of the target.

We can also divide this project into hardware, vision and control. The main goal regarding the hardware part was to mount the computer and the depth camera on the robot. Besides having a secure mounting, it was also important to achieve a repeatable set up with a constant field of view. After achieving an input from the camera in our desired set-up, the second part was to detect our target person. This person detection should be executed in real-time. In this part, the overall objective was to not only detect the target but to localize it in reference to the robot. After obtaining the local position of the target, the robot should be able to reach a certain distance to the target and maintain this distance even when the target is moving, which comprises the control portion of the project.

2 Implementation

2.1 Person Recognition

The first step in our implementation was to detect a person in front of the robot and read how far away the person is from the robot. We needed a method that would do this in real-time, without losing track of the person. To do so, we first explored a few methods to determine exactly where the person was in the image:

1. Person detection with OpenCV's HOG (Histograms of Oriented Gradients) method. HOG is a built-in OpenCV function that can detect pedestrians quickly. This method only worked when the entire person was in the camera frame. When the person was close to the camera only half their body was in the frame, and HOG no longer worked.
2. Train a convolutional neural network (CNN) to identify the person's pants. While this might have been an effective strategy, we would have had to collect and label thousands of images. This would have taken too much time for us to implement.

Finally, we decided to use color masking to isolate the pants of the person to be followed in our camera image. Our implementation focused on detecting bright red pants, but it could easily be adjusted to detect other colours that stand out from the surroundings.

There are two image streams from the RGB-D camera we used: an RGB image, and a depth image. These streams can be easily accessed and read frame by frame in python through the intel real-sense library [4]. We first read the RGB image from the Intel depth camera, then converted it to an OpenCV HSV (Hue Saturation Value) image so that we could apply a mask more efficiently. To detect the red pants, we

applied a mask to isolate the pant colour from the background. To determine the optimal HSV filter values, we created a simple trackbar shown in Figure 2.1.1 to adjust the values in real-time [5].

Figure 2.1.1: Trackbar to adjust HSV mask parameters

Through trial and error, we chose maximum and minimum hue, saturation and value thresholds that would isolate the pants in the image and remove the rest of the pixels. This mask was not perfect and some errant pixels were detected in the threshold. To remove these pixels, we applied an erosion, then dilation filter with OpenCV [6]. Once the pants were isolated in the frame, we calculated the centroid of the masked image to determine the angle of the person relative to the camera.

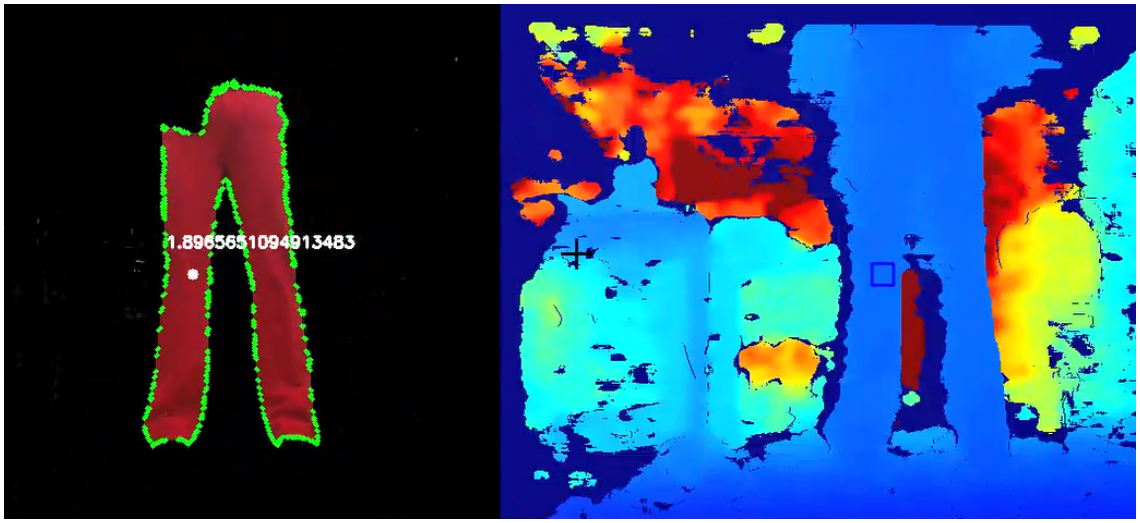


Figure 2.1.2: Masked RGB image on the left and distance image on the right. The RGB image has the centroid labeled with the average distance of the pants to the camera. The distance image is represented by a colour scale corresponding to the distance from the camera.

The distance image stream is the same shape as the RGB stream, with distance values for each pixel. To determine the distance of the person, we averaged the distance of all the masked pixels, shown above in Figure 2.1.2. We repeated this process in real time for every frame. With the centroid and the distance to the person determined, we could then pass these values onto our control function to set the iRobot wheel speeds. Our complete person detection code is implemented in *person_detection.py* in the Appendix.

2.2 Control

Once we determined the location of the person with respect to the robot, we then needed to direct the iRobot to drive toward the person and stay at a set distance. We used the iRobot pycreate2 library [7,8] to interface with the robot through python. This library allowed us to send velocity commands to the two iRobot wheels, 0 (stationary)-500(max speed).

With the centroid data from the camera, we first created a state space of 0-10 value for its x-coordinate. If the x-component of the centroid was in the first 9% of the frame, we would assign it a 0, last 9% a 10 and 1-9 for all the states in between. When the robot was oriented directly at the person, the state would be 5. If the person was not in the frame and exited on the left, the state would be -1 and on the right 11. If in a given frame the state was not 5, we sent commands to the wheels to turn so that the robot would be realigned. The states are shown below in Table 1.

Table 1: State commands for robot rotation

Case, Based on Bin Return [0-10]	Robot Response
-1	Turn left in place until person detected
11	Turn right in place until person detected
If bin < 2	Turn hard left
If 2 < bin < 4	Turn slight left
If 6 < bin < 8	Turn slight right
If bin > 8	Turn hard right
When bin ~ 5 (object approximately centered)	Object is directly ahead and implement distance control

If in a given frame the robot was centered with the person, we then drove the robot forward or backwards to maintain the set distance from the person. We used proportional control with limits for low speed and maximum speed. We set the distance to be 1m for our tests. The state space responses are shown below in Table 2.

Table 2: State command for robot distance

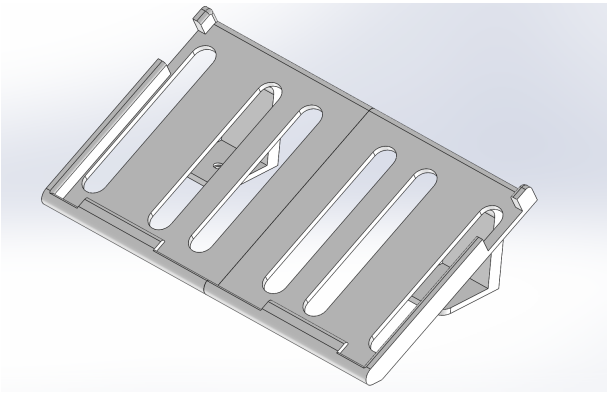
Case, Based on Distance [meters]	Robot Response
When distance > set distance + 0.1	Drives motors at a speed of 200 (kp) times the error
When kp times error > 500	Drives motors at max speed of 500 RPM

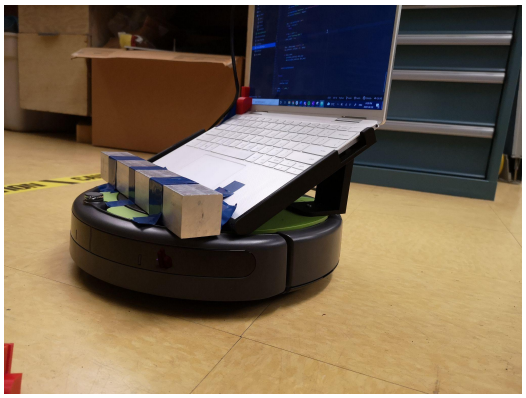
When kp times error < 150	Drives motors at a speed of 150 RPM
At set distance ± 0.1	The robot will remain stationary, but will continue to monitor for object changes and adjust accordingly
When distance < set distance - 0.1	Robot will back away from the target with a speed of 150 RPM

2.3 Mounting

The third main component of our project was to mount the camera and computer onto the iRobot. Although we had access to an Odroid single board computer, we decided to use a laptop to interface both the iRobot and the camera as there was less set-up involved. We used a Dell XPS 13 due to its relative small size and mounted the camera directly to the laptop, as discussed below.

First, we designed the laptop mount. We were limited in how we could attach the mount to the robot because there were specific locations on the iRobot we could drill into. Given this, we designed a stand in OnShape that we could bolt into two holes on the back of the robot.





Figures 2.3.1 and 2.3.2: Laptop mount CAD and printed piece.

The stand is shown above in Figures 2.3.1 and 2.3.2. The laptop slides into the stand from the top and is held in place by tabs along the side of the stand. It was 3D printed in Dr. Salcudean's lab at UBC. We had to print our stand in two pieces as it was too large for the printer. With this setup, the robot was back heavy so we added a makeshift aluminum block to the front of the robot to keep an even weight distribution.

We then focused on mounting the camera for the robot. We chose to mount it directly on to the laptop as it would decrease the amount of holes we needed to drill into the iRobot.

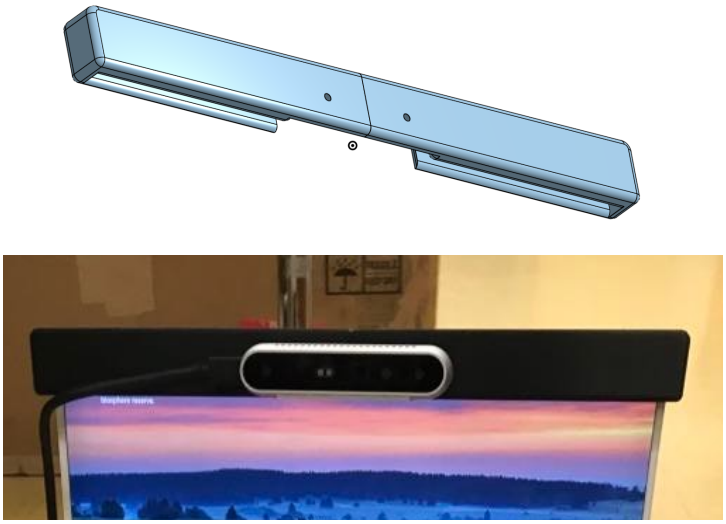


Figure 2.3.3 and 2.3.4: Camera mounting CAD and printed part

The camera mount is shown in Figures 2.3.3 and 2.3.4. As with the laptop mount, we had to print it in two separate pieces due to printer size constraints.

3 Challenges

In this section, we will outline the challenges we faced throughout the project and the steps we took to overcome them.

3.1 Brightness adjustment

During the first test of our vision algorithm and the trial to determine the HSV filter values, we recognized a huge difference between the optimal parameters for the filter if the light conditions changed. We realized that the filter will not work properly if we tune the parameters in one light and then apply this filter under different light conditions. As every room has different light conditions and the goal was to use the robot in households, we had to come up with a solution to stabilize the performance of the vision algorithm regardless of the lighting conditions (or at least to a certain degree, as the vision would obviously not work in the dark).

In order to solve this problem, we decided to add a pre-processing step to the image, before reaching the filtering. We set a fixed goal brightness value of all images and adjusted the HSV filter values accordingly. To obtain this brightness value for each image being processed, we implemented a brightness adjustment algorithm as follows:

Compute the overall brightness of the given image

1. Compute a beta value with beta = goal brightness - actual brightness.
2. Use the *cv.convertScaleAbs* function with the computed beta value to adjust the overall brightness of the image by adding/subtracting the offset to the goal brightness to each pixel.

[For further information see GitHub *adjust_brightness.py*]

Even though this might be a simple solution for our problem, the brightness adjustment proved to be sufficient and we were able to keep the preprocessing pipeline real-time capable.

3.2 Angle Conservation

During our first complete test run, we noticed that the movement transitions of the robot were occasionally very rapid regarding the direction of the movement. This caused the laptop with the mounted camera to be forced closed and hence led to the robot losing vision capabilities. This problem was solved in two ways. Firstly, the oscillation was smoothed out (covered in subsequent sections). Secondly, we designed an additional 3D-printed part to keep a consistent view angle from the laptop, as well as to keep the laptop from accidentally closing if the robot executes a fast change of direction.

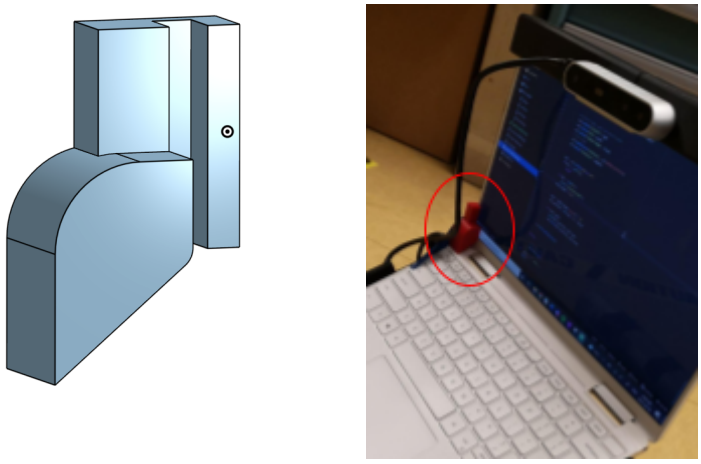


Figure 3.2.1, 3.2.2: Laptop screen holder CAD design (left) and actual piece on the laptop (right).

The part is pictured above in figures 3.2.1 and 3.3.2, and is used as a block which slides onto the laptop to keep the screen at a set angle of 120 degrees with the keyboard. This angle conservation also contributes to a repeatable setup of the camera and achieves a constant angle of view - horizontally as well as vertically.

3.3 Robot Oscillation

With our first implementation, our robot oscillated about the distance set point very quickly. This was partially due to our control code having a single point as the set distance, and partially due to the camera readout not returning a consistent value.

We solved the control issue by having a range of 0.1m about the distance set point and introducing a proportional controller. The camera readout still occasionally returns an incorrect distance from the target

due to noise and the robot will oscillate. This could potentially be fixed by averaging the camera readouts over time using a median filtering method, but was not implemented in this project due to time constraints.

4 Results

In this section, we will present results regarding the prior mentioned parts of the implementation. The structural realization of the Hardware is pictured in figures 4.1 and 4.2. Furthermore, the performance of the collective work on person recognition and control is shown in the appended videos below.

As shown in these videos, if the robot does not recognize a target, a search algorithm will take place in which the robot will rotate until a target is in sight. If a target is in field of view, the robot will adjust its orientation until it is facing the object of interest. If the target is not in the desired distance range, it will adjust its distance to the target by either driving forward towards it, or backwards away from it.

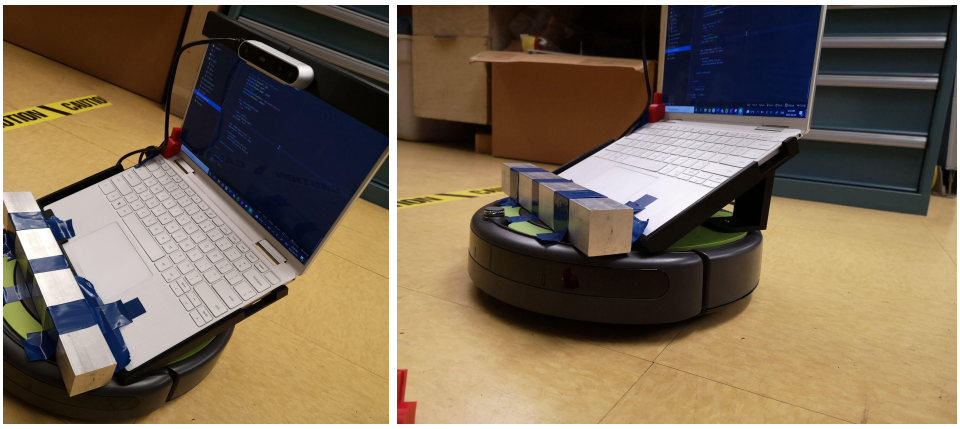


Figure 4.1 and 4.2: Top and Side View of Hardware

Final Implementation videos of robot:

Final Video 1.mp4
If that does not work, access video here

Final Video 2.mp4
If that does not work, access video here

Validation of results:
The desired result was recreated a number of times in a few different environments to ensure consistency. Further testing could take place once additional features, such as those outlined in Section 5, are covered.

5 Conclusion, Limitations and Future Development

In this project, we successfully implemented a person-following collaborative robot. We first segmented an image from an Intel RealSense depth camera to determine the position and distance of a person in relation to the robot. We then used state space control to direct the robot to be within a constant set distance of the person. To implement these methods, we mounted a computer and camera onto the iRobot with 3D printed parts.

There are several avenues for future development if we were to expand upon this project. These are discussed in detail below.

Speed up our robot to follow the person more quickly. To decrease the transient response time, we could increase the gain parameter.

Active obstacle avoidance with both vision and onboard hardware. For use of onboard sensors, impact detection indexes were not reading sensors properly. The PyCreate2 library did not properly accommodate the use of bump sensors, but instead added extra functionality for light bumpers (the onboard IR proximity sensors), which is what was used in our case. It proved difficult to cross reference these sensor values for obstacle avoidance since they typically coincided with vision commands The main problems associated with this is the fact that the light bumpers are very sensitive to the proximity of objects, as well as colors/light, but provide little to no feedback on location of the objects. Beyond this, their results vary based on robot orientation.

We also hope to include premeditated path planning, whether through use of ROS, SLAM or some other method. This would allow for a much more refined solution with more possibilities for users.

For consumer use, it is crucial we also develop advanced target detection. At this time, the project scope was limited, but in future iterations it will need to be capable of tracking targets other than those with red pants.

Finally, we can add a number of customizable roles for the robot. These can take the form of a table mounting, or other apparatus etc. in order to customize the function of the robot to suit the user's needs.

6 References

[1] The History of the Industrial Robot; Wallén, Johanna, 2008, Linköping University:
https://www.diva-portal.org/smash/get/diva2:316930/FULLTEXT01.pdf

[2] Vacuum Cleaner Market Size, Share & Trends Analysis Report, 2021
https://www.grandviewresearch.com/industry-analysis/vacuum-cleaner-market

[3] Jetsonbot person following example:
https://www.jetsonhacks.com/2015/07/22/jetsonbot-part-7-autonomous-following-vision-robot-with-a-create-2-base/
https://www.youtube.com/watch?v=Q_c8TS0p-jQ&ab_channel=JetsonHacks

[4] Real sense camera quick start:
https://github.com/IntelRealSense/librealsense/tree/development#quick-start

[5] HSV trackbar code:
https://analyticsindiamag.com/real-time-gui-interactions-with-opencv-in-python

[6] Erosion/Dilation
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[7] Pycreate2 library: https://github.com/MomsFriendlyRobotCompany/pycreate2

[8] Python Tethered Driving for Create 2:
https://edu.irobot.com/learning-library/python-tethered-driving-for-create-2

7 Appendix

First test Video:
First Video (good).mp4 (shows color sensitivity with door)
Irobot vid.mp4 (Shows initial progress with slow speeds and twitchy movement)

Code:
All of our code can be found here:
https://github.com/IRobot-Create-2-Follower-Project/Control-Code

Our class presentation slides can be viewed here:
https://docs.google.com/presentation/d/1eXXrwPCP3U9NESx14YNEbcOjKKeX-79x1IeY3zLuvhk/edit?usp=sharing