Product: Railly Clean
Team: Dalektable

Abstract

Railly Clean is a train sanitising robot that clears and sanitises train tables while the train is in, or out, of transit to save waiting time at stations and reduce risk to employees and customers.

Development on Railly Clean for the first demo focused on the robot's aisle navigation. In particular, the robot was able to autonomously move through a simulated train carriage without colliding into seats on both sides, trying to center itself in the aisle. As Railly Clean moved, it would stop upon detecting an object in close proximity in front of it, and recognize whether it was at the end of carriage by detecting an identifying blue sticker. For the purpose of the demo, upon recognizing the end of the carriage, the robot would move slightly backward before stopping.

1. Project plan update

- Carriage modelling [Partly achieved]

  We did not have sufficient knowledge to create bounding objects for collision test with the robot in time for the demo. Carriage's details have been abstracted to include an aisle and two blocks modelling seating areas on both sides.

- Hardware (robot) modelling [Achieved]
- Tables modelling [Achieved]
- Robot stability [Achieved]
- Robot movement [Partly achieved]

  We wanted the robot to make accurate right-angled turns in confined space like train's aisle, but did not have sufficient experience with sensors to implement proper feedback control. The code so far only enables crude turning.

- Carriage end recognition [Achieved]
- Demo testing [Partly achieved]

  We were not able to create a simulated train-in-motion environment to test robot stability as it moved forward. We tested that the robot in its current design could stably move around on a flat, non-moving surface, so the design was feasible.

- Demo report preparation [Achieved]

Arnav and Daniel handled modelling tasks and robot stability. Carriage end detection was split into two sub-tasks: Caitlin and Suhas handled object recognition software using cameras, while Apurv and Máté handled sensing distances to objects. Austin and Sean worked on basic back and forth movement and stopping of the robot, as well as integrating with recognition software to stop at end of carriage together with Anh. Arnav, Daniel and Sean also tested robot stability in simulated train motion, as well as robot's moving through carriage without colliding with seat blocks and stopping upon detecting end of carriage. Besides the shared chat for the entire group, each team also had its own Messenger group chat where most discussions happened. Caitlin as the project manager participated in all the group chats to overlook discrepancies and answer questions. At the start of development, under 'Aisle Navigation' milestone, an issue listing the sub-tasks was opened for each team in the project's repository and assigned to team's members. A branch was created for the milestone, on which work was done throughout the week, with commits being referenced to respective open issues. Each team also had its own folder to separate workflows. The weekly meeting was used to discuss code integration for testing, with testing happening after. Finally, Anh and Suhas wrote the demo report and prepared the demo video.

So far, because the project focuses on simulations of the robot's functionalities, the £200 budget has not been used. A 30 minute consultation on January 30 with Thomas Corberes to discuss right-angled turning of the robot was booked via sdp-rt@inf.ed.ac.uk.

Since most of the goals for this demo have been at least partially achieved, the existing goals for next demo do not need to be modified and will be the group's focus from week 4 to week 6. Simulated testing environment has been moved to week 7 as part of the 'Public Avoidance' milestone.

2. Technical details

2.1. Hardware

Figure 1 shows our first and current design for Railly Clean using existing Webots nodes. The current model aligns with our intended design, which we will build using more fundamental components in Webots in the coming weeks.

2.1.1. Distance Sensors

The robot has 4 sonar distance sensors on 4 sides of body which are used for the detection of objects on the aisle and is situated 0.626m above the ground (height may be reconsidered during 'Public Avoidance' milestone). We

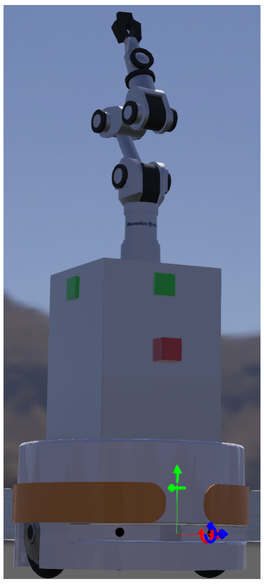

Figure 1. Railly Clean's first design, created in Webots

Figure 2. The Phidgets distance sensor we modelled the distance sensor on

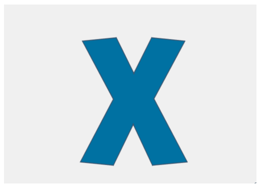

Figure 3. The template for the sticker we will be using

have modelled the output of these sensors on Sonar Phidget (see figure 2) distance sensors and they have a noise of 0.1 meters (Phidgets).

2.1.2. Camera

Our robot uses two cameras for object detection. They are placed at a height of 0.476m from the ground but will be subject to change in later milestones, as the cameras will also have to be at an optimal distance for detecting objects on tables.

2.1.3. Wheels

The IPR arm comes with 7 rotational motors (Cyberbotics, a). The TIAGo base has 2 motorized rotational wheels (Cyberbotics, b). It also has 4 caster wheels that we will probably not make use of.

2.1.4. Other Sensors

The TIAGo base comes with a wide array of sensors including a: gyroscope, accelerometer, 2 position sensors, and a touch sensor. The IPR arm also comes with 4 touch sensors and position sensors.

2.2. Software

2.2.1. Robot Movement

For each time step that the robot moves along the aisle of the train carriage, it will move forward at a constant speed of 3.9 radians/second. As it moves, distance sensors on two sides are used to check on which side the robot has more space for it to make a turn. The robot then turns in that direction to avoid obstacles as well as to center itself along the aisle. Turning is accomplished by making the wheel closer to the desired direction turn backward, while the other wheel turns forward.

If the objects on two sides are more than 4m away, too far away to be in the way of the robot, the robot will not make any adjustment. The robot considers itself centered if the difference between the distances on two sides is less than 1cm, to avoid making too many minute changes.

While moving, the robot uses its front distance sensor to know to stop if an object is within 60cm of the robot. Since that object may be the wall at the end of the carriage, to distinguish carriage's wall from other objects, as the robot stops, the front camera is used to detect the presence of a blue cross sticker that identifies the end of the carriage. As cleaning door buttons and crossing carriages have not been implemented, for the purpose of the demo, the robot will simply move backwards a bit before stopping completely.

2.2.2. Aisle Navigation and Object Detection

The main task with aisle navigation for this milestone was identifying if any object was obstructing the robot and then checking whether that obstruction was the carriage door or not.

Our robot navigates down the aisle by using the distance sensor to detect if an object is within 60cm. If it detects an object it uses computer vision to look for a blue sticker that would be placed on the carriage door to indicate whether the object is the carriage door or not. In future milestones, we expect the robot to be able to clean the door buttons as well as to cross carriages, so identifying end of carriage is an important task. We will go a bit into detail about the software we used in computer vision for object detection that helped identifying end of carriage.

For the computer vision coding we used Python's openCV library along with the Numpy library for array operations as they are the standard pair used for computer vision processing in Python. When the controller receives the image it normalizes it to reduce the affect of illumination by dividing each rgb value in every pixel by the sum of the rgb values. After this we filter all pixels of a particular shade of blue (rgb(0, 168, 243)) using the openCV `cv2.inRange()` function for the particular shade of blue we use for the sticker. We run a dilation on the image we receive from the function to reduce noise and make the image clearer using the `cv2.dilate()` function. Then we get the structuring element of the figure we have detected and run a morphological opening function on it using openCV's `morphologyEx()` function as so `cv2.morphologyEx(image, cv2.MORPH_OPEN, structuring_element, iterations = 2)`, this helps reduce the noise even more so we can get clearer contours that will greatly help with shape recognition. Contours can be found in openCV using the standard `cv2.findContours()` function. Since we expect the sticker to be of a regular shape most of the boundary points would be redundant, so we used the `cv2.CHAIN_APPROX_SIMPLE` parameter for finding the contours which would reduce redundant points and compress the contour thereby saving memory. Once we have the contours from the image we have received we call a function to match the contours we found to the contours of a template of the sticker. We match the contours using `cv2.matchShapes()` which uses the hu moments of the shapes to return a score with their similarity (OpenCV; Ming-Kuei Hu, 1962). We determined a score of less than 0.3 indicated a contour similar to the template through some testing and would thus classify the object ahead of the robot as a carriage door.

3. Evaluation

For this run of testing we decided to go for a manual logging methodology with repeated runs where we interchanged the variables according to different situations the robot might find itself in.

3.1. Upright Stabilisation

This set of tests concerned whether the robot could remain upright at different levels of extension of the robot arm.

Test	Arm Extension	Success
1	None	√
2	Minimum	√
3	Maximum	√

Table 1. Results for test case 3.1

3.2. Object Detection

This set of tests concerned whether the robot could, while moving from different starting distances, detect the presence of nearby objects and stop.

Test	Starting Distance	Success
1	1m	√
2	3m	√

Table 2. Results for test case 3.2

3.3. Carriage-End Identification (Distance Varied)

This set of tests concerned whether the robot could differentiate the end of the carriage from other objects at different distances from the carriage end.

Test	Distance	Success
1	1m	√
2	3m	√
3	Carriage length	√

Table 3. Results for test case 3.3

3.4. Carriage-End Identification (Orientation Varied)

This set of tests concerned whether the robot could differentiate the end of the carriage from other objects starting from different angles towards the carriage door. With 0°indicating the direction of the carriage door that the robot is currently moving towards.

Test	Angle	Success
1	90°	√
2	180°	×

Table 4. Results for test case 3.4

4. Budget

Table 5 estimates the total budget for the components in our system we can account for at the moment, which does not include battery as further development is needed to determine:
1. Whether charging capability can be implemented.
2. If so, what type of battery and what battery capacity is needed to complete cleaning one or more carriages.

5. Video

https://uoe.sharepoint.com/:v:/s/SDP2021-Group-10/EZl6K02c29hLr0UVnXZj9E0B52eCAum0I_WxptlsL7gluQ

Component model	Price	Count	Note	Total
Raspberry PI Camera	£20.00	2		£40.00
Raspberry PI 3	£30.00	1		£30.00
EV3 Ultrasonic Sensor	£32.39	4		£129.56
EV3 Touch Sensor	£19.19	2		£38.38
EV3 Large Motor	£32.39	2	Comes with rotation sensor; for the base	£64.78
EV3 Medium Motor	£25.79	2	Comes with rotation sensor; for the arm	£51.58
DFRobot - Makeblock Robot Gripper	£36.70	1	robosavvy.com	£36.70
Wickes.co.uk 10mm Twinwall Polycarbonate Sheet 700x2500mm	£29.00	3	Rectangular body chassis: 0.3x0.4 m^2 Cylindrical arm: 0.03(r)x0.03(r)x1.47(l) m^3 Cylindrical base: 0.265(r)x0.265(r)x0.276(h) m^3 (assume 1/3 empty space for wheels)	£87.00
LSM6DS3TR STMicroelectronics	£4.26	1	uk.rs-online.com; accelerometer + gyroscope	£4.26
Misc.	£10.00	1		£10.00
			Total estimated cost for system	£492.26

Table 5. Estimated budget for the system up until demo 1

References

Cyberbotics. Neuronics' IPR, a. URL https://cyberbotics.com/doc/guide/ipr.

Cyberbotics. PAL Robotics' TIAGo Base, b. URL https://cyberbotics.com/doc/guide/tiago-base.

Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 1962. doi: 10.1109/TIT.1962.1057692. URL https://doi.org/10.1109/TIT.1962.1057692.

OpenCV. Contours : More Functions. URL https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_more_functions/py_contours_more_functions.html#contours-more-functions.

Phidgets. Sonar Phidget. URL https://www.phidgets.com/?&prodid=973&fbclid=IwAR2g9Eh52x4JenH_R6fuDCPKEXSU06HXqxqjbHcfwcVYcsS03rI1PVnq2t8.