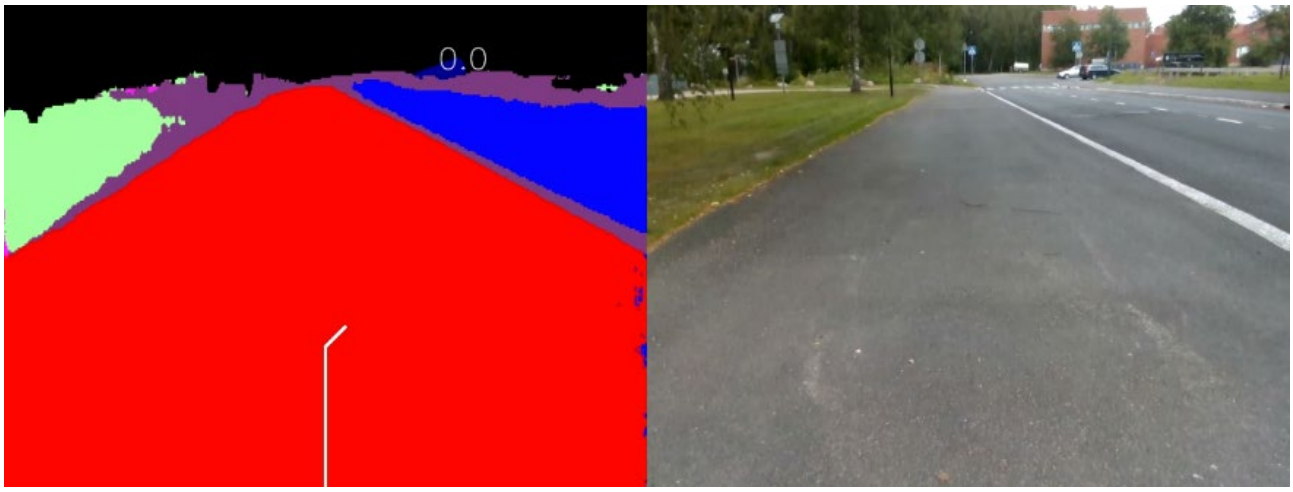
Aalto-yliopisto, Sähkötekniikan korkeakoulu
ELEC-D0301 Protopaja
2020

Final report

Last Mile Autonomous Delivery Robot Computer Vision System



Date: 31.08.2020

Student 1: Quan T. Nguyen
Student 2: Vinh Q. Nguyen
Student 3: Sakari Burda
Student 4: Tuan N. Nguyen
Student 5: Justus Ojala

Information page

Students

Quan T. Nguyen

Vinh Q. Nguyen

Sakari Burda

Tuan N. Nguyen

Justus Ojala

Project manager

Justus Ojala

Sponsoring Company

Futurice Oy

Starting date

01.06.2020

Submitted date

31.08.2020

Abstract

In this project, we devised a computer vision system that can be integrated to robots to perform last-mile delivery autonomously. The constraint on the robots is such that they should travel on the sidewalk at the speed of human pacing. Accordingly, we reckon that there are three objectives for the system to achieve: path segmentation, path planning and obstacle detection.

For the first problem, state-of-the-art deep learning techniques were used, producing an image with the path and other objects of interest colored. Next, path planning was attained by using the A-star search algorithm, while obstacle detection utilized edge detection from the depth color map. All of this was accomplished using the setup of an Intel Realsense Depth Camera D435i and a Nvidia Jetson Xavier NX Developer Kit.

The results are promising in ideal conditions. The path is clearly delineated from other areas that the robot should not go; path finding provides reliable guidance; obstacle detection performs with adequate accuracy. However, when put in an environment with divergent conditions ranging from intense shadows or insufficient exposure to disparate road materials, the accuracy of the whole system starts to decrease significantly. That said, there is still much to be done in order to achieve full automation in the real world.

Table of Contents

1. Introduction

The rapid growth of the e-commerce space has pushed customer expectations higher than ever before. As a result, speedy fulfilment is becoming a standard for the market, with companies in intense competition to reduce shipping time. Unfortunately, within such an operation, one phase poses many big challenges, taking up a disproportionately large amount of time and expenses: last-mile delivery.

It is the last few miles of the shipping process, where packages are sent from fulfilment centers to customers. Its most notable characteristics are having multiple stops with low drop size, implying a significant number of labor hours doing trivial tasks. To minimize such cost, one way forward is to move towards automation, specifically to delivery robots.

The aim of this project is to deploy a computer vision system using which a delivery robot can navigate the world. It is based on 3 central components: path segmentation, path planning and obstacle detection. Path segmentation is achieved by a high-performance deep learning model that labels the path and objects of interest. After receiving the segmentation, path planning is done by running the A-star algorithm that traverses on the colored path. Finally, obstacle detection is attained by using edge detection on the depth colormap of the environment.

The report begins with a detailed account of the setup and implementation of the project. Then, the results of the system are demonstrated visually. Eventually, reflections upon the project are contemplated, with limitations discussed, and recommendations made for future work.

2. Objective

The robot should be able to segment the path that it is supposed to follow. That segmentation should be clear and have limited noise. In addition, the segmentation of other objects of interest should also be of adequate quality. The model should work under different conditions in terms of location, weather, path curvature, .etc.

With regards to obstacle detection, the system should be able to detect obstacles with different sizes. The localization of detected actors should be accurate (colored or with a bounding box drawn over them) and is accompanied with rich information.

Finally, path planning should behave reliably, showing a result that would be in agreement with what a human would come up with. It also has to be realistic, avoiding sharp turns that the robot could not complete with reasonable speed.

As a whole, the system should work with consistency at a sufficient speed, allowing the robot to make timely maneuvers under complex situations.

3. User Manual

3.1. Hardware Setup

1. Connect 11.1 V LiPo battery to the robot
2. Wait until the ip address will be shown on the PiOLED display
3. Open any internet browser and write <ip>:8888 to the search
4. I will be connected to the robot and will be able to give it commands and see video output

5. Run a ready teleoperation code to define the controller if you want to use it to control the robot

3.2. Software Setup

1. Install Pytorch
2. Install OpenCV
3. Install Librealsense
4. Clone from LMAD repository
5. Run file "run.py" for demo video

4. Methods

4.1. Path Segmentation

As we discussed, in order for the robot to sense its surroundings, it needs to understand the environment around it. After that, there should be some mechanisms to isolate different subjects in the scene, so that the robot can pay more attention to meaningful signals and remove uninteresting information

4.1.1. Semantic Segmentation

This problem can be addressed with semantic segmentation, which is capable of separating semantically similar subjects into the same category by coloring them with the same color

This task is an area of active research, where there are tremendous sources of research papers, open-source code, tutorials, as well as public datasets. Despite all those advantages, the working process still faced many challenges such as lack of available computing resources, experiences, and misunderstandings of materials, but we were still able to overcome these difficulties and produce satisfactory results.

At the moment there are many models that show SOTA results on semantic segmentation, but for this project, we chose popular U-Net for its simplicity and efficiency. We also chose Pytorch as the framework to implement the model. We used both Cityscapes and Berkeley Deep Drive datasets for training and validating. The trained model is capable of recognizing and separating roads, sidewalk, cars, grass, and people.

We thought that this model would work well when deployed to real-world settings. However, when we test on real scenes around Otaniemi, we realize that the model cannot distinguish the road and sidewalk because sidewalks in Otaniemi (and in Finland in general) are made of asphalt, which is the same as the roads, but the roads and sidewalks in the training data usually have different surfaces. This is a critical problem as the requirement of the project is that the robot should drive on the sidewalk, not on the road, and that at some places the sidewalk is higher than the road, so there is a high chance that the robot would break itself if it cannot recognize the boundary.

We then tried many solutions but none of them showed promising results. However, at last we came up with an idea and it actually worked. We will discuss this in the next section

4.1.2. Drivable Area Segmentation

At first when the first model failed to distinguish the road and sidewalk, we thought that we cannot do anything about this issue unless we collect the training data ourselves, which is

obviously laborious and time-consuming. Fortunately, we come up with the idea that if the road and sidewalk look similar, then we can treat the sidewalk as a lane of the road. Furthermore, Berkeley Deep Drive has a dataset that contains labels of separate drivable areas of the road. We are now able to distinguish the road and sidewalk.

For this task, we also used U-Net with attention, with additional attention mechanism, and trained on Berkeley Deep Drive drivable area dataset. The results were satisfactory, and we are confident that the combination of the two models would serve as a great perception module for the robot.

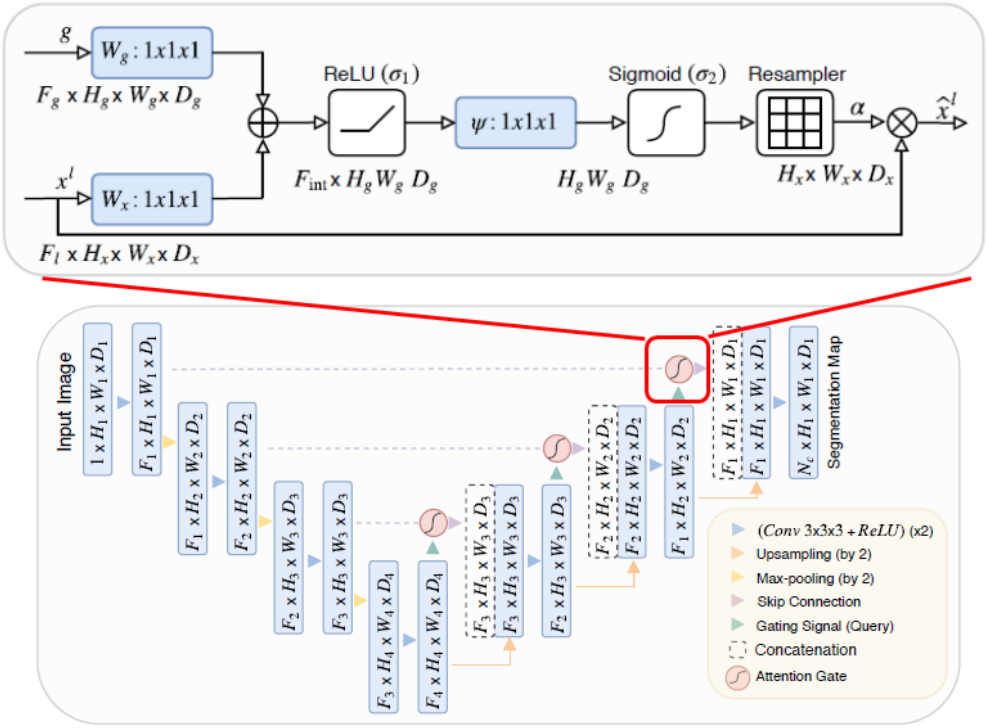


Fig. 1. Unet with attention gate.

4.1.3. Obstacle Detection

Obstacle detection is one of the most important components of any autonomous vehicle. It prevents the robot from damage itself and other people around it while in operation, ensuring smooth deployment. Hence, the domain has been a very actively researched area within computer vision, with many breakthroughs happening in recent years. For our project, we opt for a simple implementation of the system which comprises only one class of obstacle, regardless of it being static or dynamic. The sequence of steps implemented is as follow:

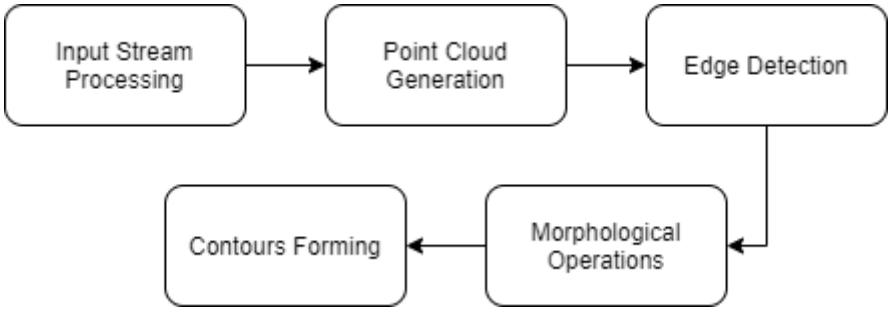


Fig. 2. Overview of our pipeline

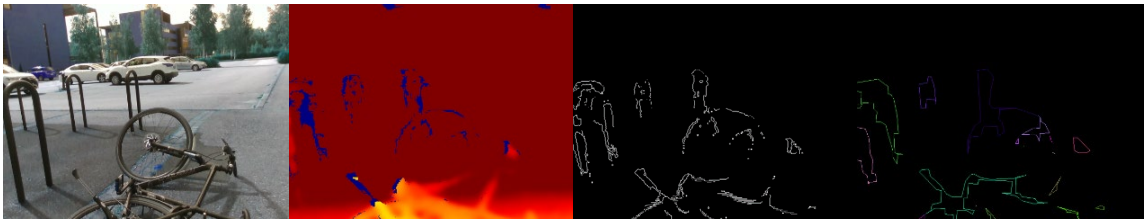


Fig. 3. The output of each steps in the process. In order from the left to right: input image, depth colormap, edge detection, contours.

First, the input image I_p needs to be scaled and taken an absolute value, then converted to an unsigned 8-bit type (OpenCV, n.d.):

$$O_p = saturate\_cast < uchar > \left(\left|I_p \cdot \alpha\right|\right) \text{ where } \alpha = 0.12$$

The value of α was chosen only by practice where we tried to maximize the occurrence of objects in the depth colormap later. Such a map was created by assigning RGB values to corresponding ranges of processed depth value (Ware, 1988). Within the representation, we cropped the point cloud at the depth limit l_d, up to which the measurements can be trusted, and deployed a bilateral filter to reduce background noise and artefacts.

Next, the tool chosen for edge detection was the Canny filter, with parameters refined according to the output of the program. The reason for this decision is because of the renowned performance of the filter (Vijayakumar & Durai, 2017) and its trivial computational cost. When calibrating the thresholds that the filter depends on, we strived to get the most of obstacles in the output while minimizing the terrain that got caught up in the process. In a sense, we were doing ground plane detection and removal by figuring out its slope through those parameters. The whole process will leave a byproduct that is the border between the depth limit l_d and the points further than that. This could be easily fixed by filtering out points that are further than the depth limit subtracted by a certain threshold $l_d - \delta l$.

The two notable operations performed at the next step are dilation and erosion. Both use structuring elements for expanding and reducing the edges, respectively (Haralick, et al., 1987). The result is that the edges are significantly more consistent, forming a smooth outline. Now, we need to recognize the edges as object instances from all these outlines. The methodology for such an operation is the active contour model or Snakes (Kass, et al., 1988). The algorithm uses an energy minimizing, deformable spline that, under the influences of the object contours and inner forces, resist deformation:

$$E^*_{snake} = \int_0^1 E_{snake}\big(\boldsymbol{v}(s)\big)\, ds = \int_0^1 \left(E_{snake}(\boldsymbol{v}(s)) + E_{image}\big(\boldsymbol{v}(s)\big) + E_{con}(\boldsymbol{v}(s))\right) ds$$

The output of all these steps is the coordinates of each object instances, in addition to an image with the contours of such objects colored.

4.2. Path Planning

In this module, we identify a path through which the robot can travel without colliding into any detected obstacles or straying from the segmented path. There are many approaches to this

problem including graph traversal algorithms, sampling-based methods or using artificial potential field. The technique that we specifically chose was the Weighted A-star path search algorithm, a variant of the famous A-star, because of its simplicity and efficiency.

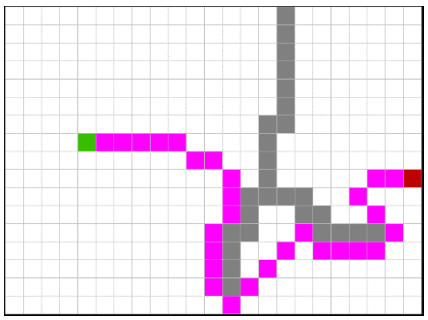


Fig. 4. Path-planning illustration

4.2.1. The A-Star Algorithm

There are two classes of graph traversal algorithm: uniformed and informed. A-star belongs to the latter class in that it needs a heuristic function to guide the exploration of the search space. Specifically, informed algorithms try to find a path to the goal that has the smallest cost according to the heuristic function.

With regards to the A-star, it selects the path that minimizes

$$f(n) = g(n) + h(n)$$

where n is the next node on the path, g(n) is the cost of the already explored path from the start node to n, and $h(n)$ is the cost of traversing to that node. Thus, $f(n)$ can be understood as the priority value of the node n. By adding the element of existing cost, the algorithm is guaranteed to give an optimal path (Russell & Norvig, 2020).

To get the Weighted version of the algorithm, we simply add a multiplier ε to $h(n)$. The result is a less optimal program but comes with a far better execution time.

4.2.2. Implementation

In the program, the Weighted A-star operates on the output of both the segmentation and the obstacle detection module. The nodes that it needs to connect are the pixels which has the value assigned to the path by the deep learning model and is not too close to any detected obstacle. The goal is such a pixel that has the lowest y-coordinate, i.e. the end of the path. For the heuristics function, the actual distance measured by the stereo camera from the node to the goal is measured. Most importantly, the 'weighted' aspect of the algorithm is exhibited by a multiplier ε = 2 based on the tradeoff between speed and optimality.

In details, when expanding a node m, the algorithm looks for neighboring nodes n to the 8 compass directions at a step of 20 pixel-Euclidean-distance away. There has to be steps as not doing so increase the number of nodes to expand logarithmically and lead to very sharp turns. Cost for actions, traversing from one node to another, is calculated based on the actual distance between the points that the pixels display.

Differing from a typical graph traversing algorithm, our Weighted A-star does not have to reach the goal node g to terminate, but the condition is relaxed to improve runtime with the terminating condition as

$$h(n) < \epsilon \text{ where } \epsilon = 1 \text{ (meter)}.$$

The result is connected lines painted on the output that leads the robot towards the end of the path.

4.3. Testing robot

We created a test robot for testing the functionality of the software. We started out with a robot controlled by a NVidia Jetson nano controlling the robot's motors through an Arduino and a generic motor controller. It was to be built into a 3D-printed hull, which was based on an earlier project of one of the team members and went through rapid iterations before being dropped entirely in favor of first a ready car platform and later a laser cut acrylic base. The depth camera was attached to the robot with a threaded rod. The Arduino was also dropped and the motor controller exchanged for an Adafruit Featherwing unit, as we moved from directly controlling the motors with an addition to the software to integrating our software with Jetbot, an existing platform which supported the Featherwing. For power, we used a 3s lithium-polymer RC battery which provided 11,1 V to the motors and powered the logic through a 5 V dc-dc converter. We also experimented with some other implementations for the robot, but these were short-lived.

5. Results

Overall, every module can perform its own task with reasonable accuracy in ideal conditions. However, when put in an environment with divergent conditions ranging from intense shadows or insufficient exposure to disparate road materials, the accuracy of the whole system starts to decrease significantly. Regarding performance, the system runs at an acceptable frame rate of around 3 FPS on the Nvidia Jetson Xavier NX, or 5 FPS on any machine with i7 Intel Processor equivalent and a Nvidia GTX 1650 equivalent.

5.1. Path segmentation

With the methods discussed in section 3.1, the final model shows decent results in several settings and environment. The model is capable of correctly distinguishing objects of different categories. Most importantly, the model can show a clear border between the main road and the sidewalk, which facilitates the development of further path planning algorithms. These two models can enhance each other's strengths and complement their weaknesses in order to improve the perception of the robot about the environment.

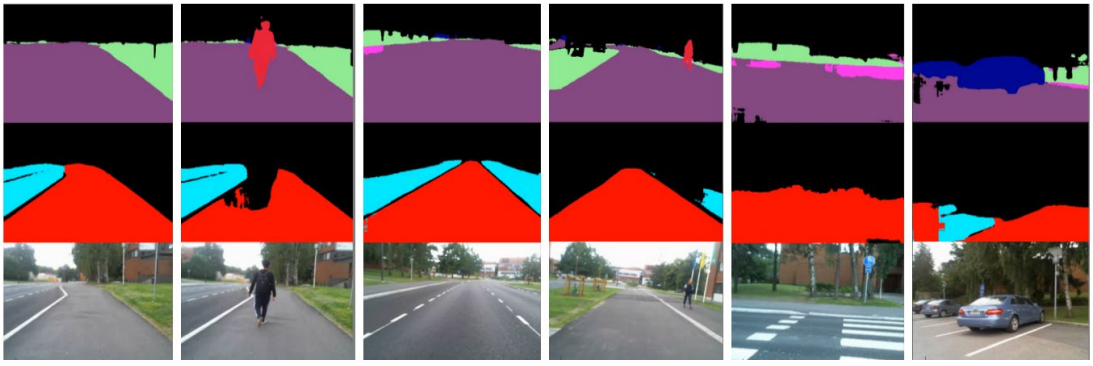


Fig. 2. Results from the two models of path segmentation. The top row shows the segmentation of different objects including road, sidewalk, terrain, car, and person. The second row shows the segmentation of different drivable lanes of the road (this ensures the separation of the main road and the sidewalk). The last row shows the images from the camera

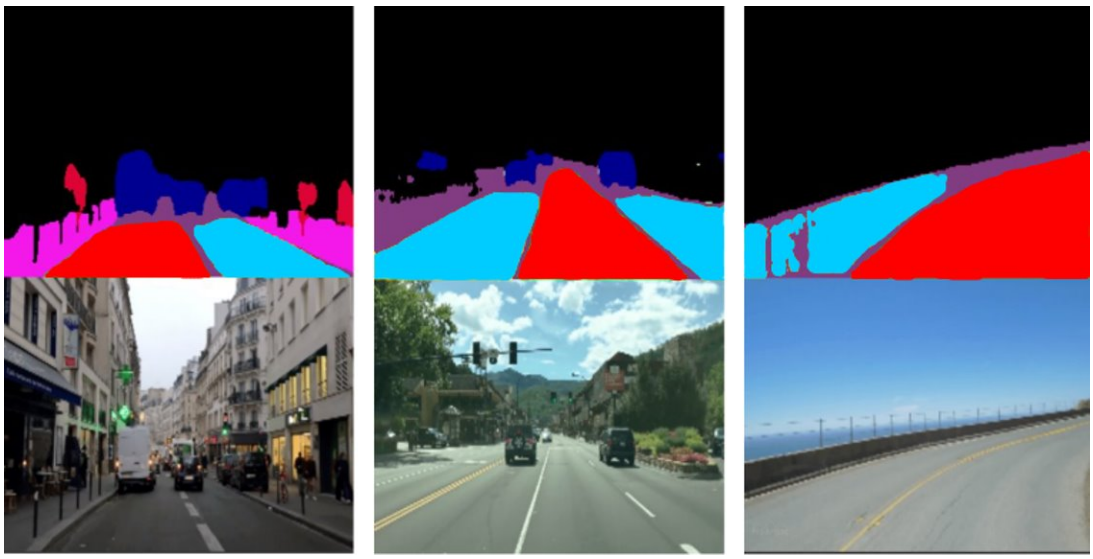


Fig. 3. The combination of the outputs of both models

These models work well in their own domain. When we tested in real environments, the final model only worked if the path was made of asphalt, and it failed most of the time if the path was made of different materials, such as soil, concrete, and paving. This problem could be addressed to the datasets that the models were trained on. Those datasets were collected and designed specifically for self-driving cars in urban areas, which means that there was no variation in driving surfaces as cars only drove on asphalt roads.

This could pose a huge problem in the scalability of the method, in the sense that the service can only be deployed in areas where all sidewalks are covered with asphalt. This condition is hardly met in reality. To tackle the issue, a dataset that contains various road surfaces is required for the model to learn and perform well in different driving scenes. However, collecting and annotating such dataset could be expensive and time-consuming. Other solutions such as domain adaptation or synthetic dataset could also be experimented to minimize the need for an extra-large dataset.

Another direction for this is to use LIDAR to help the robot to sense its surrounding environment. This could be more suitable for autonomous robots when they need the ability to handle different types of driving surfaces.

5.2. *Obstacle Detection*

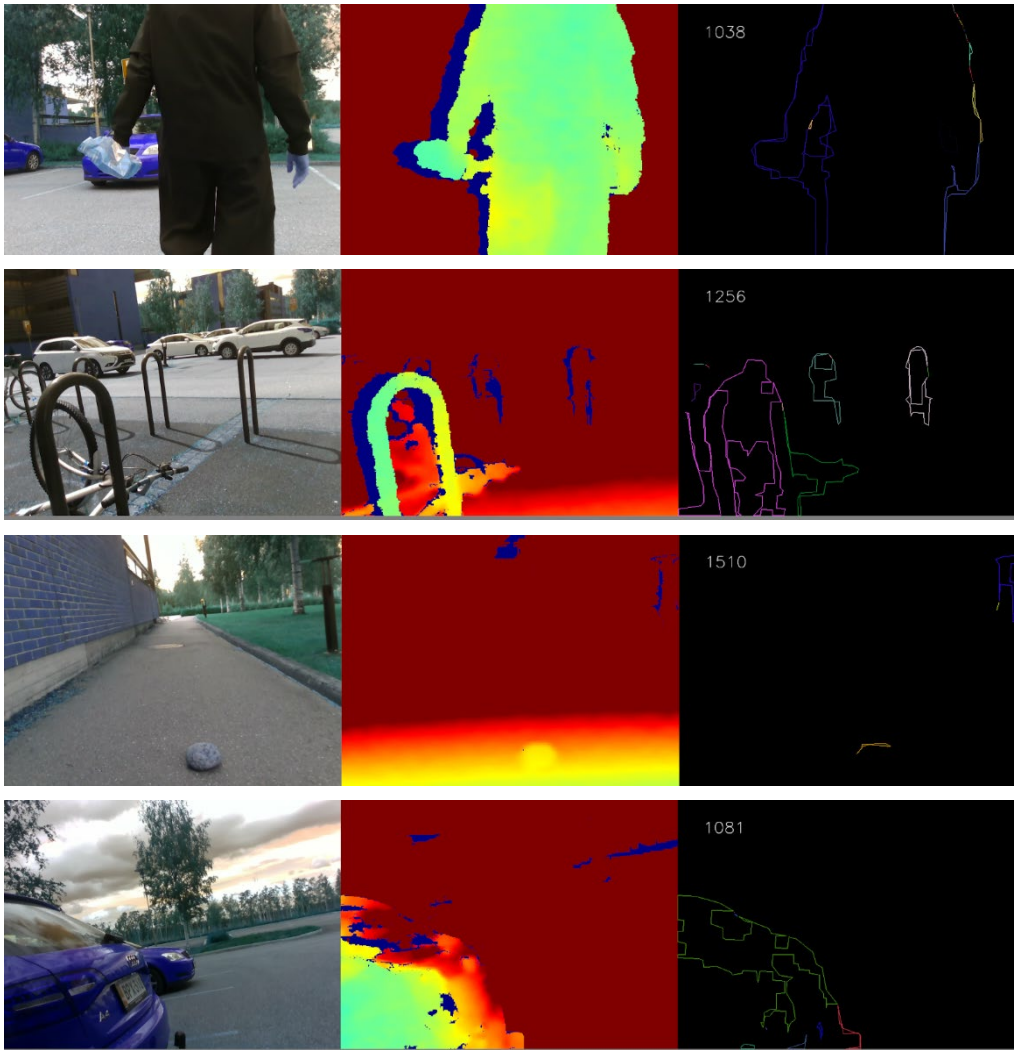


Fig. 7. Images at each level of process

Left: input image. Middle: depth map of the image. Right: result of obstacle detection.

Through careful choices of thresholds for the edge detection filter, the program managed to achieve a reasonable accuracy when separating the obstacle and the background. It works best when given a medium-sized non-reflective object, as shown in these images. In the first one, the person is clearly separated from the background and all the outlines are well connected. For the second image, all the bicycle parking stands outlines are localized together with some extra detail.

On the other hand, small objects such as a rock or reflective one like a car may not provide good results. This is a partly because of the limitation of the program, and because of how the stereo camera operate. In the image featuring the rock, the difference in terms of depth value between the ground and the lower part of the rock is close to unnoticeable from the perspective of the Canny filter, thus rendering the contour of the lower region of the rock visible.

This can be resolved by increasing the upper threshold of the Canny filter, However, unfortunately, the approach makes not only the rock outlines appear in more detail in the final contour image, but also the ground plane, posing a dilemma. Concerning the car image, we can see that depth cameras generally do not work well with reflective surfaces, evident by the broken depth image of the car. This problem trickled down to the contour image, but the algorithm can still handle surprisingly well. That said, all these problems can be overcome later in the pipeline

by creating a bounded region around the detected outlines with enlarged perimeter, minimizing the margin of error.

Regarding measurement, the program demonstrated both high accuracy and reliability. It can consistently measure distance to objects with a very thin margin of error and minor fluctuation. This can be attributed to the quality of the stereo camera.

5.3. Path Planning

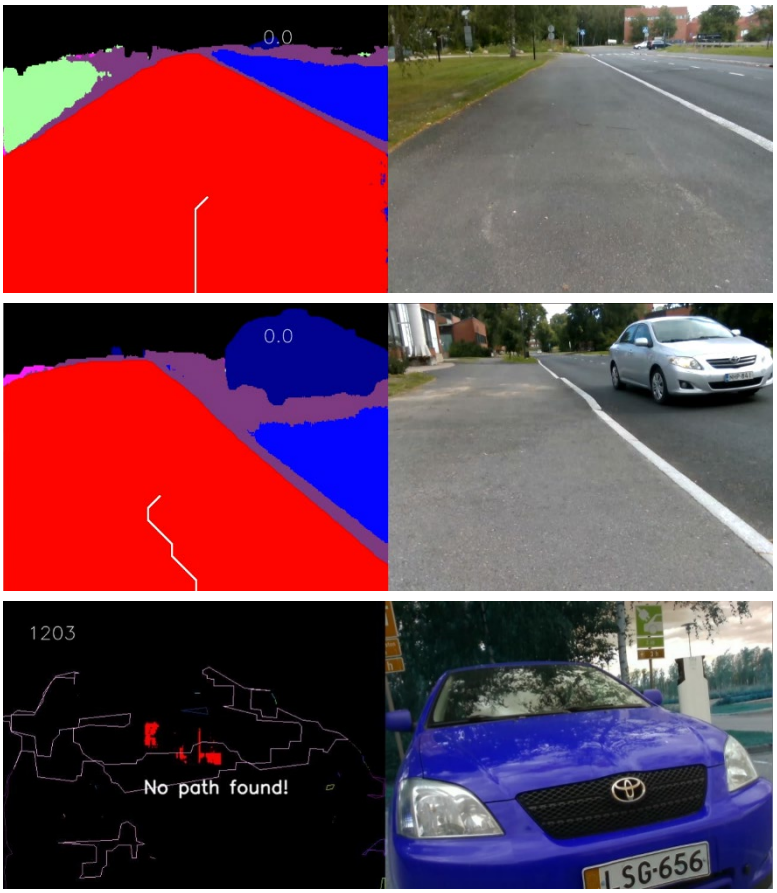


Fig. 7. Path planning in different scenarios

Right: RGB image. Left: output of path planning.

The module performs its task capably, is highly adaptive to different scenarios in the figure: path, skewed path, no path. However, its performance greatly depends on the output of the other components, as we can see from this image from an earlier version of the software:

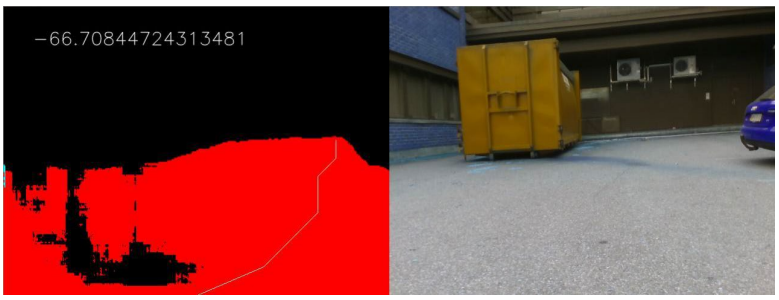


Fig. 7. Path planning when other components does not function well

Right: RGB image. Left: Output of path planning.

In this version, the path is not clearly notated, and thus is difficult to see. Nonetheless, if we look closely at the details, we can see that the path line is quite removed from what we think it

should be, due to the broken segmentation. The solution for this seems quite straight forward, that is to improve the other parts of the program.

Regarding the processing cost of this module, under ideal conditions, the execution time would be almost instantly. However, if the goal is unreachable, there would be a noticeable drop in frame rate, possibly down below the 1 frame-rate mark.

5.4. Testing Robot

Hardware used for the first prototype:

- Jetson Nano
- 3D printed frame
- 5V power bank as power supply
- 2 x TT 5V DC motors + wheels
- Feather wing 5-12V motor driver
- PiOLED display
- Raspberry pi camera

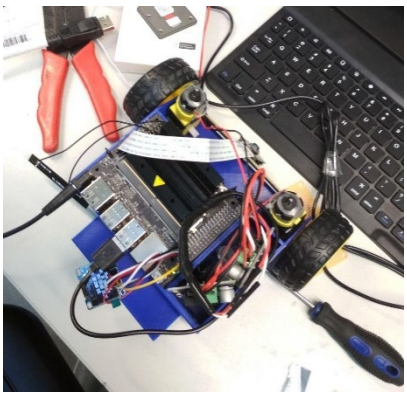


Fig. 8. First prototype of the test robot

The working voltage of the first prototype was 5V and computational power was based on the Jetson Nano's computational power. To make the robot working it was needed to install the Jetbot library to the Jetson. If the robot is built right, from the PiOLED display you can see the IP address of the connected network when you turn on the robot. You can see memory and Ram of the robot from the PiOLED as well. To connect a laptop or computer to the robot the given IP should be used. We can see the camera's video output and give commands when the robot is connected to a laptop.

The problem of this build was that it wasn't powerful enough to function on other surfaces than flat ground inside a building. While working on obstacle detection problems we decided that we should get a depth camera to be able to effectively do obstacle detection and get the best sufficiency possible. Also, Jetson Nano was sometimes freezing or performing badly with the code we had so we needed to upgrade computational power as well.

Hardware used for the second prototype:

- Jetson Xavier
- Acrylic sheet as frame
- 11,1V LiPo battery as power supply
- 12v to 5v converter
- 2 x 11V motors + wheels
- Feather wing 5-12V motor driver
- PiOLED display
- Intel Realsense Depth Camera
- Xbox controller

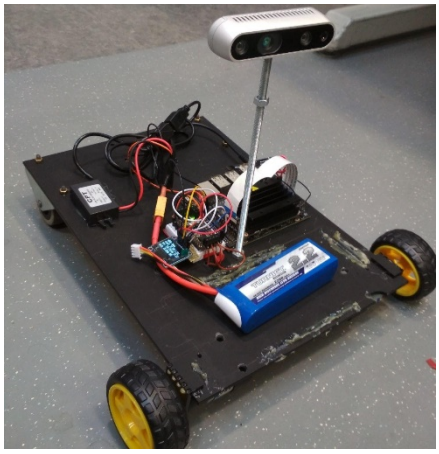


Fig. 9. Second prototype of the test robot

The working voltage of the second prototype was already 11.1V which increased its efficiency outside a building a lot even though it still isn't able to deliver heavy things to a destination. Now we are able to run our segmentation, obstacle detection and path planning code more smoothly and get better results with computational power of Jetson Xavier. While using the robot we stuck to the problem that it's working range was not wide enough because the robot and the laptop were always connected to the same wi-fi. With some research we understood that it is solvable and that we can control our robot from any location with the wi-fi connection while the robot is connected to another wi-fi network. Unfortunately, we couldn't test it in the university due to specific internet settings, which the university has. Also, this feature might not be working with smartphone's wi-fi hotspot. Also purchasing a Realsense Depth camera resulted in getting obstacle detection code to perform well in the tasks we used it for. With a xbox controller we got the opportunity to control the robot without writing any commands. This increased working speed of the robot because there is much less needed to make any corrections to the direction of the robot.

6. Reflection of the Project

In general, the final result resembles very much what we set out to do in the plan. However, there was some inevitable tweaks and refinement, reflecting the new knowledge that we continually gathered throughout the project

6.1. Reaching objective

For the most part, the project achieved the expected output defined in the project plan, with the software being able to segment the road, detect obstacles and partly navigate. Nonetheless, there were some secondary elements that were dropped due to various reasons.

Motion prediction was such a component. Initially, it was determined that the software should anticipate the movement of dynamic actors in the environment then make maneuvers accordingly. However, it was later deemed to be too time consuming and cumbersome to implement, as the team has very little knowledge of this sophisticated technique, which may lead to significant risk.

Another software unit that did not make it in the final version of the software is remote-control. Although it is important for the operator to observe everything from the perspective of the robot, this is still the outer layer of the core program, therefore regarded as less crucial among

other components: a task we only do when every other task was done. Unfortunately, such an occasion never arrived.

Regarding the testing phase of the system, due to the specialized nature of this program, we realized that we had to build our own robot in the middle of the project. However, the robot was not completed in time, so we had to resorted to manual testing, which comprises attaching the camera to a trolley and simulated the route of a test robot. Although this approach might be viewed as unsatisfactory, we still managed to obtain footages of what a robot would see – an adequate demonstration of the final work.

6.2. Timetable

Surprisingly, most of the main phase of the project was completed before the schedule. This could be partly attributed to the decision to do many tasks in parallel, especially lane segmentation and obstacle detection. Nevertheless, the time it takes to optimize the system offset all the advantages of getting ahead.

Unfortunately, some tasks were unexpectedly difficult and waste a substantial amount of time. This is especially true regarding the hardware, with the library for the stereo camera took precious weeks to be installed. Problems also occurred when building the test robot, which was often disrupted by lack of understanding what parts we might need when code will grow bigger and more power consuming and ordering/delivering problems during the pandemic. This reason as well as the consecutive breakdowns of the hardware rendered this subtask abandoned.

Eventually, all the work was already done before the deadline, thus we considered the project as being on time.

6.3. Risk analysis

The foremost risk we considered was that a robot, when traveling in traffic, might cause an accident. Because of that no testing took place in traffic, helping us avoid any possible accidents. We also anticipated the prospect of not meeting the deadline, and how it happened was elaborated in the previous subsection.

One hidden risk that manifested only while we are working on the project is setting up of hardware and software wasting a significant amount of time. In retrospect, the installation of the Intel Realsense library took us more time than actually building path-planning – a core component in the project.

Another risk we didn't consider was the delays in hardware parts ordering and delivery. It was complicated to predict what all the hardware we need because often the need was revealed only while testing the code and hardware we had. This problem is intensified in the context of a pandemic, resulting in ordering delays which sometimes was detrimental to some areas of the project, notably in building the test robot.

7. Discussion and Conclusions

7.1. Discussion

In general, the outcome of the project matches with our goal and expectation that we made in the beginning of the course. We managed to implement three software submodules performing

three important tasks, with all of them showing adequate results in their own domain. However, there remain many points to consider when we put our project in the broader context.

Firstly, the program only act as a base upon which further development can happen to move towards full autonomy. Although the path segmentation module brought about great result for the project, its weakness of breaking down under varying conditions prompts the need for more extensive data to train the deep learning model. Obstacle detection module is satisfactory in that it can produce a result, but the real world demands much more sophisticated techniques with 3D localization to decrease the margin of error. A more suitable approach could be using the RANSAC algorithm to remove the ground plane then approximate depth data by voxels.

Later in the pipeline, the representation of the surrounding should be improved so as it is not what the robot sees, but as an occupancy grid with regions of interest colored. Ultimately, a motion planning unit, in addition to the path planning, is indispensable. To move efficiently, the robot should not only go in straight line but also in natural curves to improve efficiency. We recommend the Conformal Spatiotemporal Lattice planner which is pioneered by planetary rovers and is demonstrated to be highly effective.

For the robot deployment, a testing robot could be built to test our current modules and examine their effectiveness in navigating the robot in real sidewalks as well as ensuring the safety of nearby people as well as the robot itself. Furthermore, a remote-control module can also be implemented, allowing flexible control for the operators. With the current pipeline, the robot can only navigate and move with local information obtained from the surrounding environment. Global information about current location and destination are required to make a successful delivery. Moreover, a logistic module to optimize sourcing and shipping is needed to ensure smooth operation, maximize customer satisfaction, and minimize cost.

Secondly, even after implementing such improvement, there remains many problems not only for this specific system, but also in the domain of building autonomous vehicles as a whole. For instance, human movement prediction is an open problem in the field. This is due to the only fact: humans are complicated, requiring not only mathematics and computer science, but also psychology, biology and sociology. Nevertheless, the landscape is very exciting. New ideas are continually arising in the literature, and companies are working tirelessly to improve on their autonomous products. Thus, expects major improvements coming soon.

Through this project, we learned many aspects of product development. With respect to management, we learned how to work in an international team where differences in culture and language are major barriers. We also learned about how to effectively communicate internally with team members and externally with company representatives. Project management, planning, and discipline was also an important skill. In the technical aspect, we acquired many experiences while working on the project. We got to know how deep learning works and how to implement deep learning models in Pytorch. Different public datasets were studied together with data preparation, training tricks, and cloud resource utilization. We also acquired many computer vision skills including depth map processing, edge detection via contouring, and path planning with searching algorithm. Most importantly, by researching, implementing, and evaluating, we sharpened our critical thinking, that we had to think out of the box to solve problems.

7.2. Conclusion

In this project we presented a vision-based system that can understand its surrounding environment. Path segmentation deep learning model helps the robot know where it can go; path planning shows where it should go; obstacle detection tells it where to avoid. We showed that our system is real-time capable on a computationally constrained platform such as the Jetson Xavier NX

Autonomous robots, especially vehicles, are exciting innovations. They opened the way for a wide range of applications that tremendously benefit people. By transitioning to autonomous systems, innumerable number of business processes would be improved, countless lives would be saved. Realizing the tremendous opportunity of autonomous systems, we expect this project could be served as a proof of concept for the feasibility of adopting such technology to improve lives in Finland.

Working on this project gives us the opportunity to apply our knowledge in practice and push our skillset to a new level, but also prepare us with the empathy and discipline to work in an international team. We got to know the current advancement in computer vision, AI, and robotic hardware; we learned about their strengths and weaknesses when applied to practical settings.

List of Appendixes

- Project plan
- User manual
- Demo videos

References

- Haralick, R. M., Sternberg, S. R. & Zhuang, X., 1987. Image Analysis Using Mathematical Morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* PAMI-9(4), p. 532.
- Kass, M., Witkin, A. & Terzopoulos, D., 1988. Snakes: Active Contour Models. *International Journal of Computer Vision,* pp. 321-331.
- OpenCV, n.d. *OpenCV 2.4.13.7 documentation.* [Online]
  Available at:
  https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#void%20convertScaleAbs(InputArray%20src,%20OutputArray%20dst,%20double%20alpha,%20double%20beta)
- Russell, S. J. & Norvig, P., 2020. *Artificial intelligence a modern approach.* 4th ed. Boston: Pearson.
- Vijayakumar, J. & Durai, L. J., 2017. A Review and Performance Analysis of Image Edge Detection Algorithms. *International Journal on Future Revolution in Computer Science & Communication Engineering,* III(12), p. 397 – 401.
- Ware, C., 1988. Color Sequences for Univariate Maps: Theory, Experiments, and Principles. *IEEE Computer Graphics and Applications,* pp. 41-49.

- Oktay, O., Schlemper, J., Le Folgoc, M., Heinrich, M., Misawa, K., McDonagh, S., Y Hammerla, B., Glocker, B., & Rueckert, D. (2018). Attention U-Net: Learning Where to Look for the Pancreas. *arXiv e-prints*, arXiv:1804.03999.