UiO : Department of Technology Systems
The Faculty of Mathematics and Natural Sciences

Adrian Anderson Bergflødt, Jørgen Petlund, Stephen Hangluah

Object tracking vehicle

TEK5030 — Computer Vision
Obligatory assignment spring 2021

Supervisors: Trym Haavardsholm, Idar Dyrdal, Thomas Olsvik Opsahl , Sigmund Johannes Ljosvoll Rolfsjord and Ragnar Smestad

2021

Contents

1 Introduction

Object following vehicles have many useful applications, such as transportation of goods and automating filming during movie production. The goal of this project is to grant the participants with experience in working with computer vision systems.

This project is based on an object following toy car, in-which the car is seeking to understand an object and be able to follow it autonomously. To accomplish this task the project was initially divided into three phases; phase 1 collision detection, phase 2 object following, and phase 3 implementation on hardware. However, phase 3 was omitted due to the importance of the collision detection behaving properly. To acquire this a method for segmenting out the ground plane was needed. The project therefore consist of the two phases, collision detection and object following. The first step is to segment out the ground plane and then check for possible collisions. This is done by virtue of the RealSense Camera, which contains depth readings. Secondly, the the object following is implemented by tracking of an ArUco marker. Finally the controllers for tracking the marker are implemented and visualized.

The report is mainly divided into the three sections; method, result and discussion. All of the work done is presented and discussed in detail, and is a shared effort by all group members.

2 Method

Execution of this project was split into two phases.

- Phase 1: Collision detection
- Phase 2: Object following

Phase 1 consists of utilizing the Realsense 435d camera to register if an object is too close to the car. Phase 2 involves using the realsense camera to register the position of an object, in this case an ArUco marker, and its distance from the car.

2.1 Phase 1 Collision Detection

Phase 1 consists of utilizing the Intel Realsense 435d camera to register if an object is too close to the car. The Realsense camera provides an RGBD feed that can be exploited for this purpose. A visualization of the depth data from the RGBD feed is shown in Figure 1.

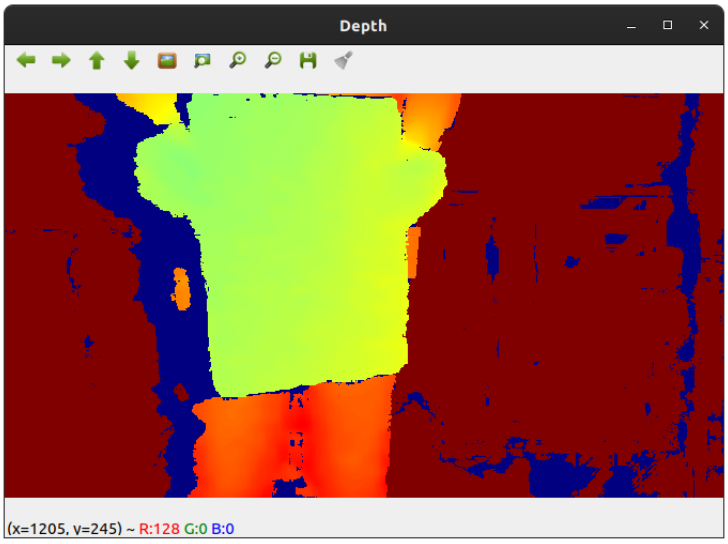


Figure 1: Visualization of depth data.

Segmentation

This RGBD feed does however not ignore the ground. Meaning that if we scanned the image directly for points within 0.4 meters, it would detect the ground as being to close and tell the car to stop. Thus a method for segmenting out the ground had to be implemented. To do this a rectangle, in the area where the ground is thought to be, is sampled of the RGB image. This is used to train a multivariate normal model of the colour of the ground. This model is then used to create a Mahalanobis image of the RGB image. This Mahalanobis image is thresholded using Otsu's method, creating a mask.

Finally, the mask is used to set the distance in the RGBD feed on the parts detected as ground to a large number. In effect the ground is now seen as an object that is far away from the camera. Figure 2 show the sampling region and the segmented image.

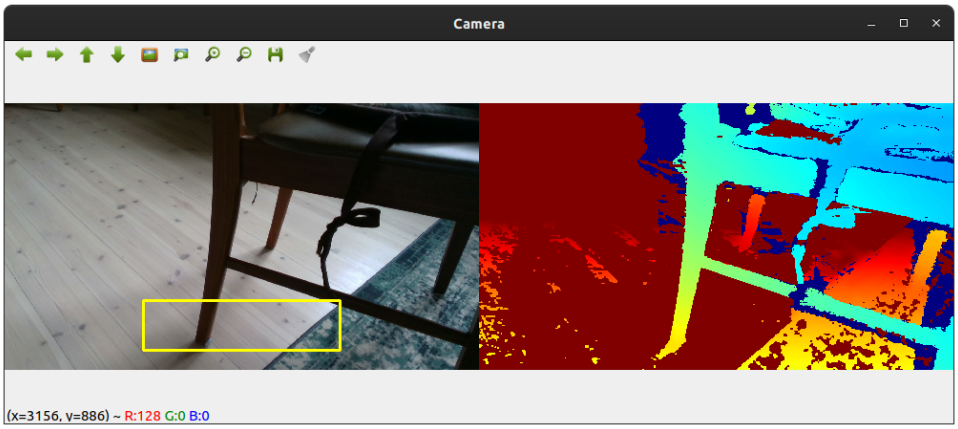


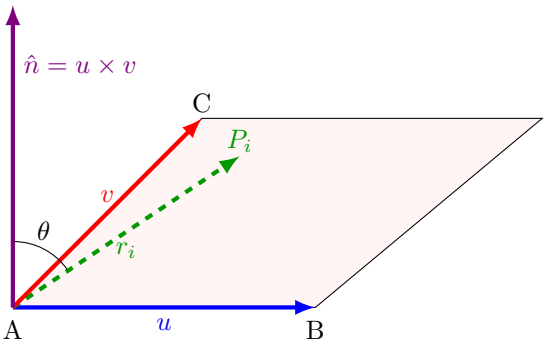
Figure 2: The RGBD camera feed with the sampling region denoted by the yellow rectangle, and the segmented image output. The segmentation sets the detected ground plane as far away, indicated by the deep red colorization.

Detection condition

The RGBD feed contains zeros wherever it was unable to calculate depth, it is also noisy. Thus an algorithm for deciding when an object is to close had to be robust to noise, and false zeros. This algorithm iterates through all the pixels in the image and counts the amount of pixel depths between 0.1 and 0.4. if this exceeds 2500 pixels, it reports this as an collision.

Ground Plane Estimation

In some cases color segmentation will not detect the ground plane to a satisfactory degree. Another mask may be required, so that the ground can be properly filtered. One possible approach is to use the depth image to calculate the ground plane from three points.



The mathematical foundation for plane estimation is given by the fact that any plane can be represented by two distinct but intersecting lines. Which in turn can be be represented by three arbitrary points, A, B and C. Thus the two vectors $u = B - A$ and $v = C - A$ represent the plane. The cross product of the two vectors u and v give the normal of the plane. Any point can then be tested by creating a vector with the base point A, and then checking the dot product of this vector and the normal. If the two lines are perpendicular to each other, $\theta = 90$, the point lies in the plane, and the dot product will be zero.

The implementation of the ground plane estimation is done by:

1. Choosing three random points from the sampling region. Points A, B and C which contains pixel location and depth, $Point(pixel\,x,\ pixel\,y,\ depth)$.
2. Create the vectors $u = B - A$ and $v = C - A$. Do the cross product of these to create the normal of the plane, $\hat{n} = u \times v$.
3. Check for all the points in the sampling region the dot product of the normal and the vector $r_i = P_i - A$. If the dot product is close to zero the point P_i lies in in the plane, $|\hat{n} \cdot r_i| < Tolerance$, where $Tolerance$ is a tuning parameter.
4. Check the fit of the plane estimation by getting the percentage of points in the plane. If enough points in the sampling region are in the plane, $Threshold < Fit$, set this plane as the ground plane, if not, go to step 1. and repeat. Here $Threshold$ is a tuning parameter and $Fit = \frac{Points\ within\ the\ plane}{Points\ in\ sampling\ area}$, both given in percentage.
5. Do the same dot product approach as in step 3. for all the points in the image. Set the points that match as a mask of where the ground plane is detected.

However, before this estimation is done the depth image is down-sampled to a suitable size. This is done to save computation time and to filter out noise. Applying the mask is done in the same way as for the color segmentation, by setting the filtered values to a sufficiently large number in the depth image.

This estimation method is susceptible to variation in camera orientation, especially pitch, and the altitude from the ground. When these change the tuning parameters needs to be tuned again for the new positioning. Thus, it is not used for the collision detection in this phase, where a more simplified test environment is envisioned and the color segmentation proves sufficient. The color segmentation is also less prone to necessary re-tuning given different orientations.

Collision detection pipeline

The collision detection pipeline is then as follows. An RGBD image is taken, the ground is segmented out of this image, and an algorithm calculates if there is an object within 0.4 meters of the camera and sets a flag if this is the case.

2.2 Phase 2 Object Following

Phase 2 involves using the realsense camera to register the position of an object, in this case an ArUco marker, and its distance from the car. The ArUco marker in question is shown in Figure 3.

Figure 3: ArUco marker

The object following consists of two controllers, one for the depth distance to the ArUco marker, and one for the horizontal distance between the center of the RGBD feed, and the center of the marker.

ArUco detection

The detection of this ArUco module is done by using the OpenCV ArUco module. The module returns the position of the corners of all the ArUco markers in a picture. Depth distance to the marker is then extracted from the depth image. Horizontal error is calculated by subtracting the markers x position in the image frame, from the x position of the center of the image.

Depth controller

Since the motor responsible of driving the car forward is either on or off, the distance controller is a on-off controller with dead-zone. Meaning that if the absolute value of the depth distance to the marker is smaller than 0.5 meters then the controller tells car to stop moving, and if it is greater than 0.7 meters the controller tells the car to move forward.

Horizontal controller

The steering of the car is done by a servo, thus on-off control for the horizontal error would be inefficient. Instead a P-controller was implemented for its simplicity and because it is good enough for this purpose. The formula for the P-controller is shown in Equation 1.

$$u_k = Pe_k \tag{1}$$

where u_k is the control variable, e_k is the horizontal error, and P is the controller gain.

3 Results

The results are divided by the two implemented phases; collision detection and object following.

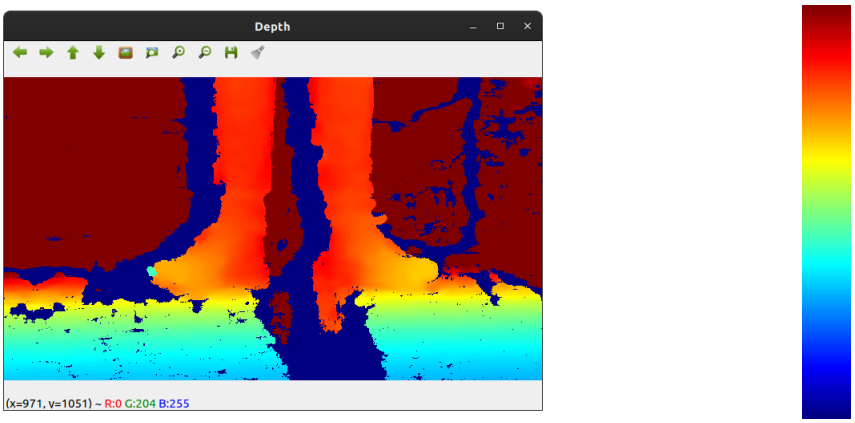
Phase 1: Collision detection show tests for the color segmentation and the ground plane estimation. Different scenarios where the two segmentation methods may be suitable, either alone or together, are presented. Finally, the test of the collision detection is presented.

Phase 2: Object following presents the results of the testing of the; ArUco marker detection, depth controller and, finally, the horizontal controller. In the end all controllers perform as expected.

3.1 Phase 1 Collision Detection

Segmentation

Figure 4a shows a visualization of the depth image before the ground is segmented. While Figure 4b show the colormap. If this image is not segmented, the controller will detect the ground as a collision. Another interesting aspect is the blue patch in the middle. The deep blue colorization indicates zero depth, which in this case corresponds to a faulty depth read. Distances closer than approximately 17 centimeters will not be read due to limitations of the camera. Though this is not the case here, glare obstructs the depth reading, as can be seen from Figure 5. Although the collision detection threshold is set to 10 cm to filter these readings, it also means that a reflective object can subvert the depth measurement and collision detection.



(a) (b)

Figure 4: (a) Depth image before segmentation and (b) the color scale used to visualize the depth. Blue indicates a short depth measurement and red indicates a far away depth measurement.

Figure 5 shows a visualization of the depth image after it has been segmented using Otsu's method. In this image the ground is successfully segmented out. The controller will not detect this as a collision. There are some errors occurring due to the glare. However, most of it is segmented away. The troublesome faulty depth read of the glare, indicated by the deep blue from Figure 4a, is no longer prevalent.

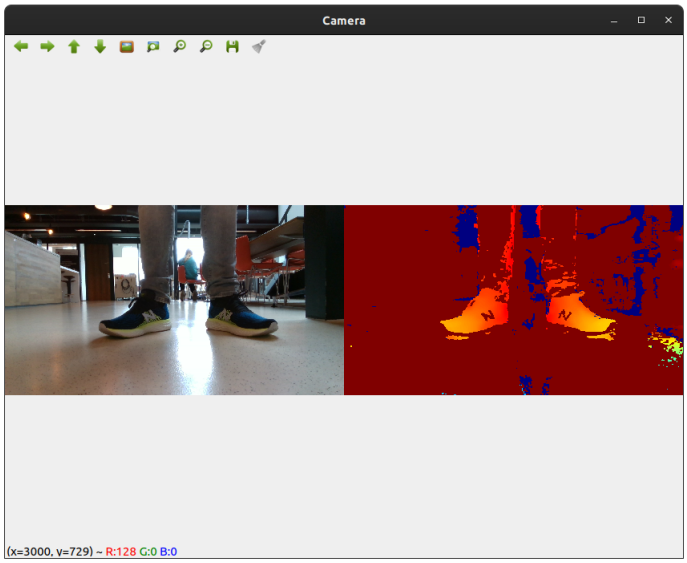


Figure 5: The camera feed and depth image after applying the segmentation mask.

Ground Plane Detection

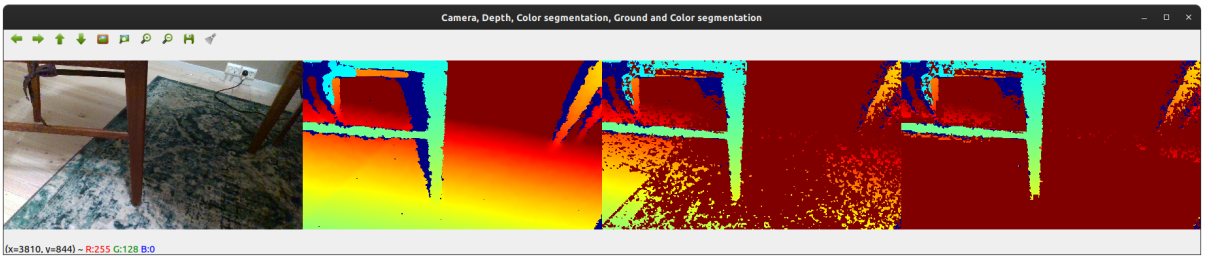
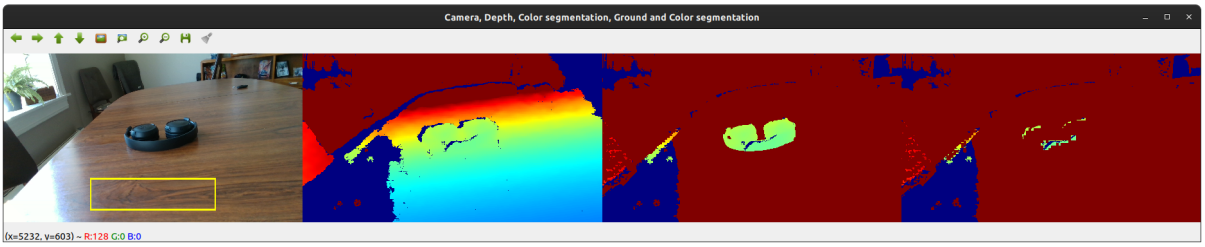
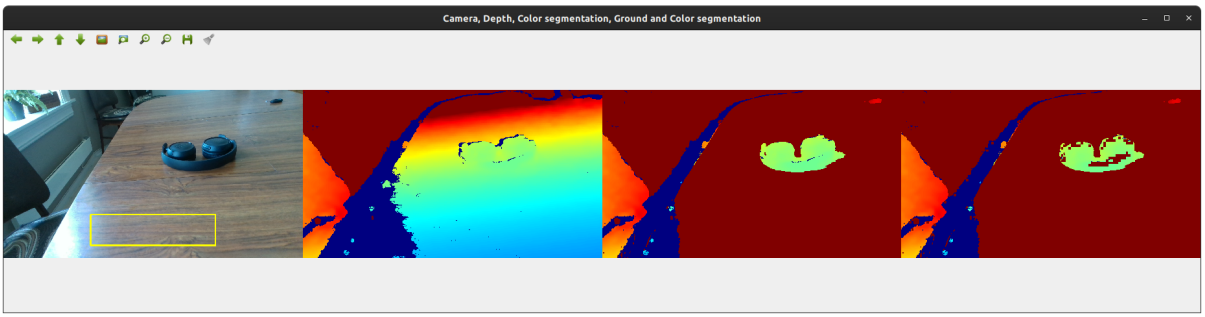


Figure 6: The four image views are from left to right: (1.) The camera feed, (2.) Depth image, (3.) The color segmented depth image and (4.) The ground plane detection mask applied to the segmented depth image.

Figure 6 show a problem scenario where the color segmentation alone is not sufficient. The ground has uneven color, and the segmented depth image includes readings that clearly lies in the ground plane. This includes the wooden floor and several patches in the carpet. With the ground plane estimation mask applied these readings are filtered out. In the end, only the chair legs are present as possible collision objects, which is the desired output.



(a) Low altitude and a small pitch rotation from the ground plane.



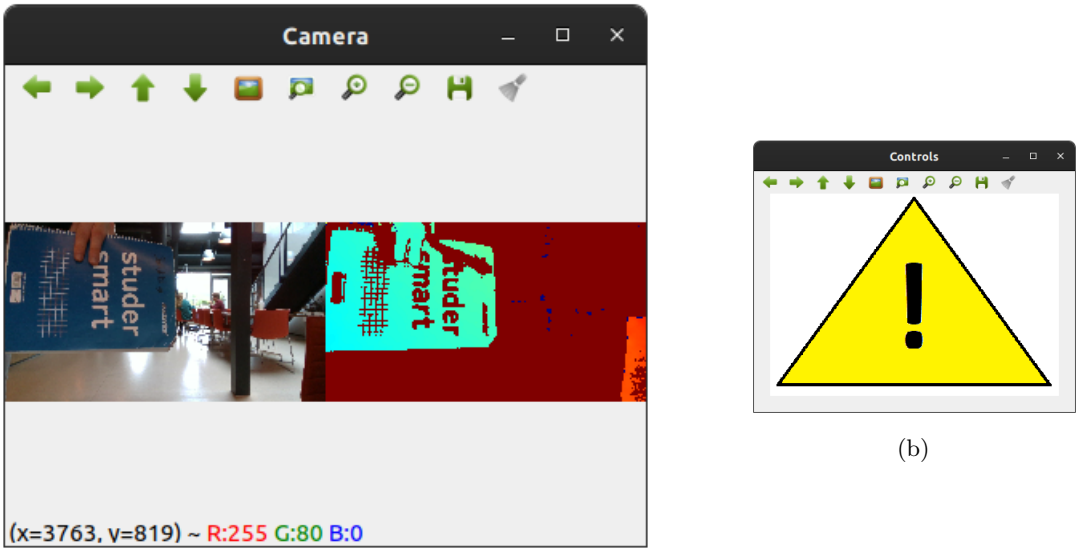
(b) High altitude and a large pitch rotation from the ground plane.

Figure 7: The four image views for the two figures are from left to right: (1.) The camera feed with sampling region, (2.) Depth image, (3.) The color segmented depth image and (4.) The ground plane detection mask applied to the segmented depth image. In (a) the headset is almost completely filtered out with the ground plane estimation mask applied. While in (b), with a different camera orientation, the headset remains after the segmentation.

Figure 7 show a scenario where the ground plane estimation filters away possible collision objects depending on camera orientation. This is not wanted and the color segmentation here provides a sufficient image without the additional mask. Difference in pitch and altitude plays a large part in whether or not the object becomes filtered out. Contrary, the color segmentation can, more or less, obtain an optimal setting that applies in many cases, irregardless of orientation.

Collision Detection

Figure 8a show an object that is close to the camera, and the segmented depth image of it. This should be close enough to be registered as a collision. Which, in turn can be seen from Figure 8b. The control output from the system is a warning sign that means a collision is imminent.



(a)

(b)

Figure 8: (a) Object close to the camera and (b) Output from the controller.

3.2 Phase 2 Object Following

ArUco Detection

Figure 9 shows the system detecting the position of an ArUco marker during testing. The system had no difficulties detecting markers.

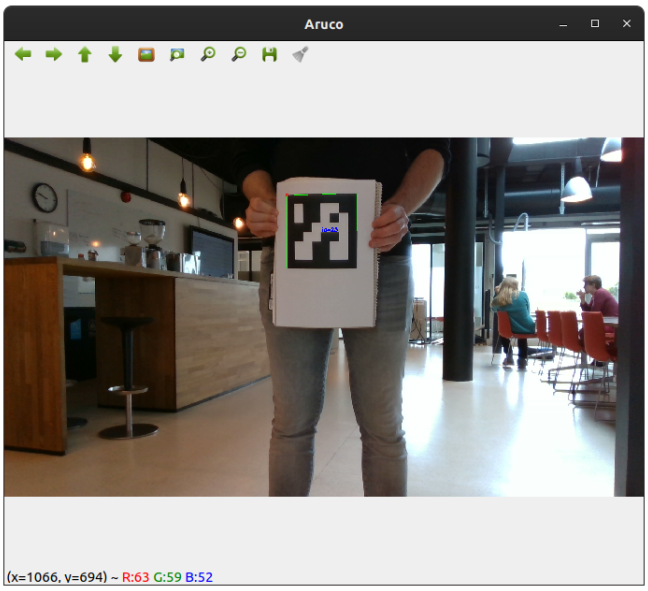
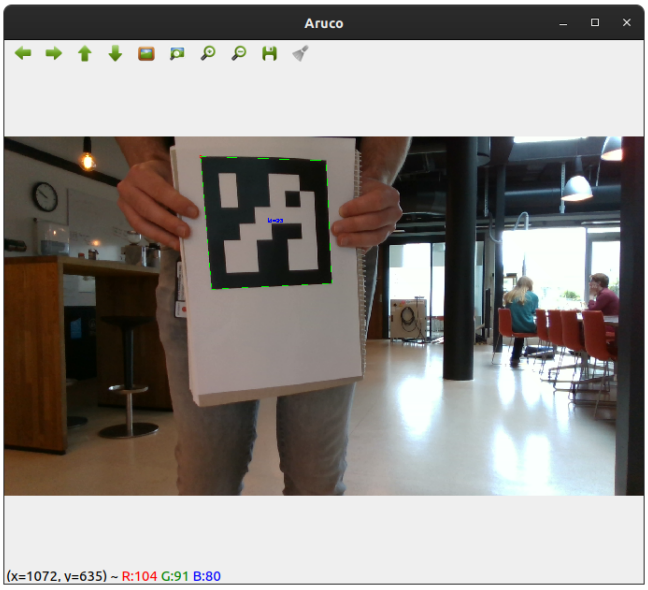


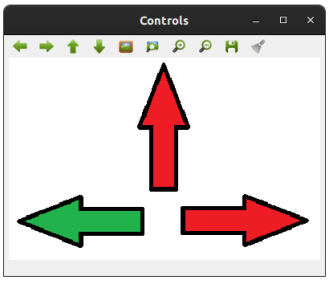
Figure 9: Marker being detected.

Depth Controller

Figure 10a show an ArUco marker being detected close to the camera. This should result in the controller telling the car to stop. Figure 10b show the output from the controller. Where the arrows indicate that the car should indeed stop, while also turning left.



(a)

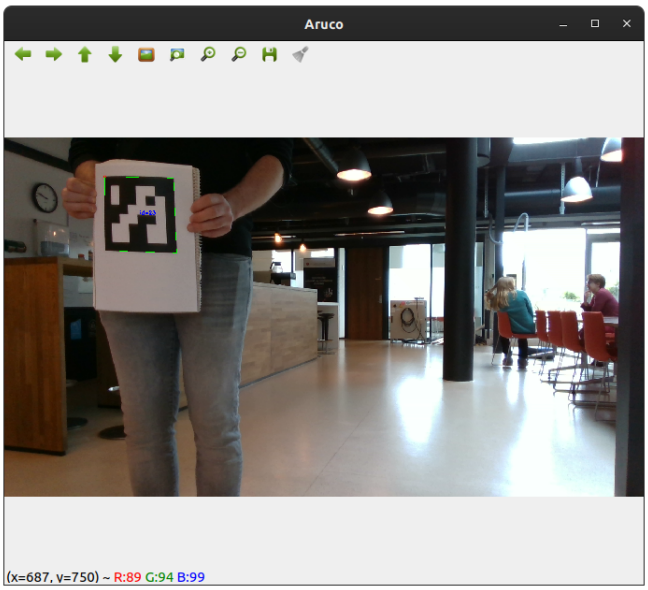


(b)

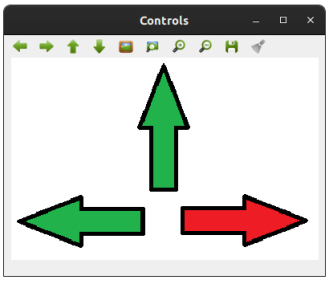
Figure 10: (a) ArUco marker detected close to the camera and (b) Control output.

Horizontal Controller

Figure 11a shows a marker being detected to the left of the center of the camera, and some distance away. This should result in the controller telling the car to turn left and drive forward. And, can be verified by looking at Figure 11b, which show the output from the controller. The arrows indicate that the car should, as assumed, drive forward while turning left.



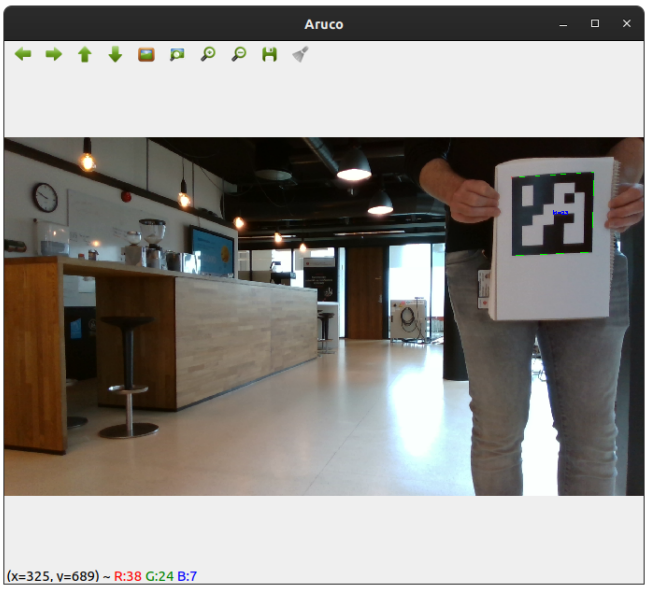
(a)



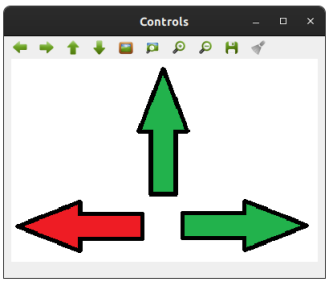
(b)

Figure 11: (a) ArUco marker detected left and (b) control output.

Figure 12a shows a marker being detected to the right of the center of the camera, and some distance away, this should result in the controller telling the car to turn right and drive forward. Again, this can be verified by looking at Figure 12b. The output from the controller, given by the arrow symbols, means that the car should drive forward while turning right.



(a)



(b)

Figure 12: (a) ArUco marker detected right and (b) control output.

4 Discussion

The project sought to create a solution for a toy car to follow a predefined object. Such a solution has many large scale real life applications, be it as transportation of goods or for automating filming during movie production. The group participants saw this as an opportunity to gain experience in working with computer vision systems and its possible applications and challenges.

The scope of the project provided unexpected challenges, particularly the ground segmentation. For this purpose two different solutions where implemented and tested, color segmentation and ground plane estimation. The color segmentation method worked great in environments with uniformly coloured surfaces, but it encounter difficulties in areas with multicoloured surfaces. It also results in the collision detection ignoring objects with the same colour as the ground. The ground plane estimation does not have these issues. It is, however, prone to errors caused by change of orientation of the camera, and therefore requires a lot of fine tuning. It is also prone to errors from the depth reading propagating through to the applied mask. This was not the case for the color segmentation, which proved more robust to changes in orientation.

An inherent problem with the ground segmentation is the tendency for over filtering. The reason behind adding the two masks together is that they, by themselves, do not detect the ground to a satisfactory degree. It is paradoxical then, that it is the opposite operation that is needed to counter over filtering. By finding the difference between the two masks it would be possible to account for their individual shortcomings, by detecting areas where over filtering is happening. Naturally, this requires that they properly detect the ground plane on their own. Reaching a stage where this is accomplished would require a lot of tuning, be it from the tuning parameters to the sampling region, and may also warrant different ways to implement the methods. However, the sort of sensitivity analysis needed to attain this, lies outside the scope of this project. Yet, it highlights a central issue in computer vision and the problems associated with segmentation.

A controller was implemented to allow a car to follow an ArUco marker. The controller worked as intended, being able to display control inputs to a car to navigate it towards the ArUco marker. The solution provides a p-controller for horizontal movement which was reduced to a digital controller for illustration purposes.

The solution created in this project works perfectly in a controlled environment, but may be prone to error in a real world application, the most volatile component being the segmentation of the ground.

The project was successful in its goal to create a controller that follows an object. It was also successful in providing its participants a greater understanding of computer vision, and different challenges when implementing computer vision for real world applications.

The next step is to implement the procedures created on the vehicle and enabling continuous control for forward movement. Further work on the control system may include adding evasive maneuvers in response to detected collisions, including functionality to follow humans and other objects through the use of classifiers and further improvement of the ground segmentation.

A Github

A repository with the source code created for this project can be found at:
https://github.uio.no/jorgepe/TEK5030-Project