[TME290]
Project report

Alexander Radne
Anna Petersson
Jan Schiffeler

May 2020

1 Introduction

In this project, the assignment is to develop a program which can autonomously drive a small car (called a Kiwi car). The environment for this car is a closed race track with yellow, blue, and orange cones marking the boundary. The blue cones are to the right and the yellow cones to the left, whereas the orange cones indicate an intersection or stopping point. This assignment is split into three tasks. First, to drive around the track as fast as possible without any other cars in the path. Second, to drive around the track without hitting other cars. Third, to pass an intersection without collision with another car coming sideways. In all tasks, no cones should be hit. These robot behaviours are developed and tested in a simulation environment and with pre-recorded data, acquired in real-life tests.

2 Method

The robot control is split into three microservices written in C++. Two microservices handle the perception of the robot: one detects cones[1] and the other detects Kiwi cars[2]. The last microservice handles the actuation[3] of the car, that is, the control of the steering angle and ground speed based on the perception. Additionally, a fourth microservice[4] has been introduced for development and visualisation purposes. The information flow between the microservices is shown in Figure 1.

During the development, the code has been under version control using Gitlab. Moreover, the project has used Gitlab's CI/CD pipeline and container registry. Additionally, unit testing was used to check each commit. Working with a Kanban board helped with organisation and issue tracking.

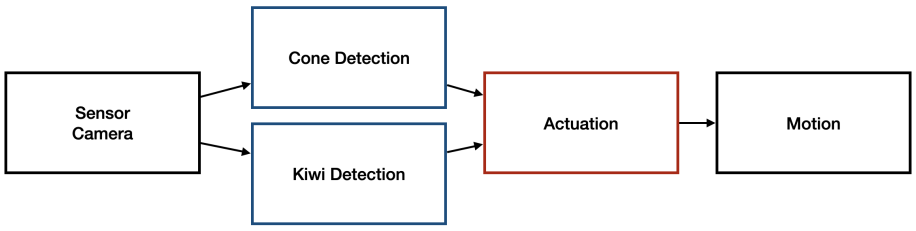


Figure 1: *Information flow in the system of microservices.*

[1]https://git.chalmers.se/courses/tme290/2020/group8/opendlv-perception-cone-detection
[2]https://git.chalmers.se/courses/tme290/2020/group8/opendlv-perception-kiwi-detection
[3]https://git.chalmers.se/courses/tme290/2020/group8/opendlv-actuation
[4]https://git.chalmers.se/courses/tme290/2020/group8/opendlv-development

2.1 Perception

The perception of the Kiwi car is separated into two image based detection parts, namely the detection of cones and other Kiwi cars.

2.1.1 Cone Detection

The cone detection uses the fact that there are cones of three different colours: blue, yellow, and orange. The input is the video feed of the Kiwi car. First, the image's top and bottom is cropped to reduce the search space and rule out possible false positives. Then, the image is segmented by the use of thresholds of different colours. Several morphological operations are performed to remove noise in the segmentation: open, close, and dilate. This method of segmentation is used since it is fast.

The next step is to find bounding boxes of the cones. These are used to filter out segments that are not cones based on the shape as well as to make it easy to find positions of the cones, which will be the output. Thus, the contours of the binary images from the segmentation are found. Then, the bounding box is let to be the rectangle of the minimum area that encloses the contour. Using the bounding box, shapes that are wider than they are high are removed as well as shapes that have an area that is too large or too small. These criteria were chosen rather strict to not include any false positives but this also means fewer cones are detected overall. Finally, the positions of the cones are let to be the middle base point of each rectangle.

2.1.2 Kiwi Detection

To avoiding collision with another Kiwi car, it is essential to detect them reliably. Since ultrasonic sensors have trouble picking up small objects, and especially those with a rough surface, the detection was chosen to be image-based. Two methods where tried: one based on classical computer vision methods and one using a neural network.

The first approach uses edge detection and colour thresholding to find a Kiwi car. This method was very unreliable, especially under difficult lighting conditions. Further improvements could be made, for example, by masking the whole image with a very generously chosen colour threshold before edge detecting, to rule out some obvious misinterpretations. However, since the second method was found to be highly reliable, no further investigation proceeded.

Neural networks with various architectures have been dominating object detection and classification problems. However, deploying such algorithms usually requires lots of extensive matrix calculations which are computationally heavy, especially in the absence of an appropriate graphics card. Since this project aims to be computed on a Raspberry Pi, the network must run sufficiently fast. While the classic detector R-CNN (regional convolutional neural network) requires several iterations over each image to estimate its content, single shot detectors only iterate the image once. Here, we used a "you only look once" (YOLO) network, or more precise YOLOv3-tiny [1]. The network was trained on the framework called darknet. The training used the provided sample images as well as annotated images gathered from the simulation environment. The provided images have a different resolution than the Kiwi camera, but this is no problem for the YOLO algorithm since all incoming images are resized. The Kiwi car got detected correctly using this approach. Still, it sometimes got lost for a few frames in between, as seen in supplementary video one[5]. A Kalman filter was introduced to solve this issue. The physical model used in the filter estimates the motion to be linear. This is not completely true in reality, but the results with this filter are better than assuming no motion between frames or no filter at all.

2.2 Actuation

The foundation of the actuation is based on a behaviour based robotics structure. The control consists of four main behaviours: emergency brake, no cones, crossing, and regular motion. The flow is illustrated in Figure 2. If the Kiwi is too close to something in front, then it should brake

[5]Supplementary video one - Kiwi detection without Kalman filter: https://youtu.be/C6JvfYd1Ykc

immediately according to *emergency brake*. If it cannot see any cones, then it should move forward slowly as defined in *no cones*. If the robot perceives an intersection, then the *crossing* behaviour is applied, where it should follow the right-hand rule, see further in Section 2.2.3. Otherwise, the car should follow its *regular motion* as described in Sections 2.2.1 and 2.2.2.

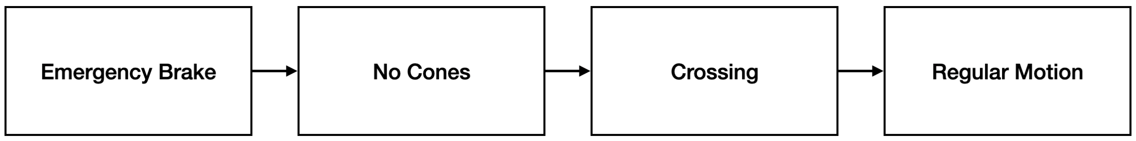


Figure 2: *Actuation behaviour-based structure. The first applicable behaviour from the left will be used.*

2.2.1 Path Following (Task 1)

The Kiwi car should be able to follow the track of the cones for task one. Thus, an aim point is computed, and then the ground steering angle is set proportionally to the angle to this aim point. The angle to the aim point is defined as shown in Figure 3. In a first version of the path follower, the aim point was set to the midpoint of the average positions of the yellow and the blue cones respectively. In a later version, the steering angle was set to a weighted average where points closer to the bottom of the image — that is, closer to the Kiwi — were more influential. Moreover, a PID control of the steering angle was implemented to stabilise the output steering angle further.

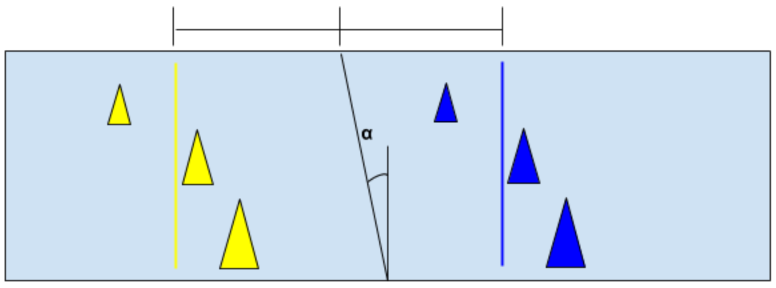


Figure 3: *Steering angle is proportional to α, where α is the angle to the mid-point of the mean position of the observed cones.*

The pedal position is set inversely proportional to the steering angle. This will speed up the Kiwi on straights and slow it down in curves.

2.2.2 Distance Keeping (Task 2)

For task two, our Kiwi car should be able to run in the same track as another Kiwi track without colliding. Moreover, if another Kiwi car is in front with a slower speed, our Kiwi car should adapt its speed and keep an equal distance. The distance to the Kiwi car in front is calculated based on the bounding box, which is returned by the kiwi detector, using the formula $D = F\frac{H}{h}$. Here, D is the distance to the kiwi car in meter, H is the height of the Kiwi in meter, h is the height of the Kiwi in the image in pixels, and F is the perceived focal length in pixel. Since the width of the bounding box varies with changing yaw angles of the car, while the detected height stays rather constant, only the latter is used.

The perceived focal length is calculated by a set of known values using the provided calibration image. Image width and height of each cone was measured, and jointly with the known distance a focal length for each cone was calculated. The different focal lengths vary a lot since the correlation between appeared size and distance is not linear. However, a linear estimation was used since we know in what range to expect and react to a Kiwi car. This approach provided reliable results at around and below 0.5m.

Then, the value of the pedal position is set depending on the distance to the Kiwi car in front. If the distance is larger than a specified maximum braking distance, then the pedal position is set to the default speed. If it is between the maximum braking distance and a minimum braking distance, the speed is proportionally lowered. If the distance is below the minimum braking distance, the pedal position is set to zero.

2.2.3 Intersections (Task 3)

For task three, the right-hand rule is implemented. This behaviour is based on the number of detected orange cones, if another Kiwi car is detected in front to the right, and the distance to the detected Kiwi car. If these criteria are fulfilled, the car stops.

2.2.4 Testing the Actuation

Unit tests were added early in the development in order to check that the actuation behaves as expected. These tested, for example, the following: the Kiwi should stop when the front distance sensor output is very short, and the Kiwi should turn right if it only observes cones to the left. With the use of unit tests, it was made sure that we did not impair the code when implementing further improvements.

Figure 4 shows a snapshot of a development toolbox microservice that plots the Kiwi and the cones in realtime in a simulation mini-map. This was useful when testing and improving the actuation.

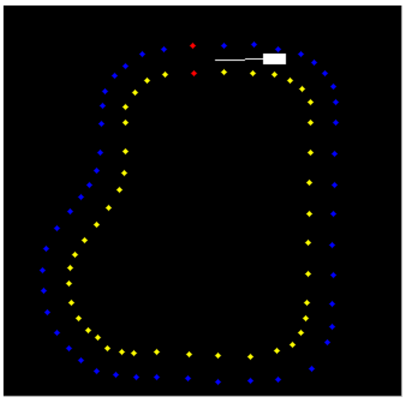


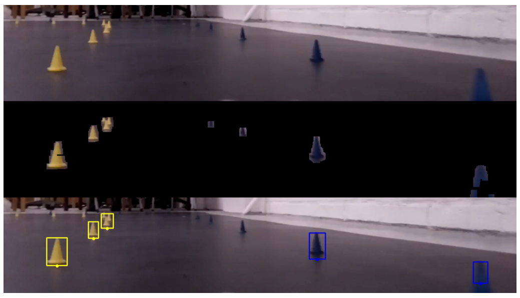
Figure 4: *Mission overview showing the position of the car and the cones*

3 Results and Discussion

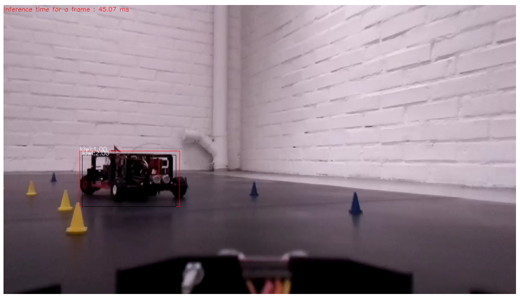
In this section, results from replay, simulation, and the demonstration are presented and discussed. Results using replay of data can be found in supplementary video two[6], showing the cone detection, and in supplementary video three[7], which shows the Kiwi detection. Further, examples showing the cone and Kiwi detection are showed in Figure 5. As can be seen in the supplementary video one, the cones closest to the Kiwi car are well detected. Moreover, there are few or none false positives. However, many cones are missed, especially those further ahead, which probably is due to different lighting. Keeping memory of detected cones between frames could possibly improve this. Supplementary video two shows that the Kiwi detection with the YOLOv3-tiny network works very reliably. The execution time seems to be appropriate but was not tested on the Raspberry Pi.

[6]Supplementary video two - Cone detection: https://youtu.be/z5hkflZnpCA
[7]Supplementary video three - Kiwi detection: https://youtu.be/pLId5WH1GxE

Improvements could possibly be made by switching to object tracking after a successful detection instead of trying to detect the Kiwi car in every frame. This could improve the execution time and smooth the bounding box motion. Introducing a Kalman filter was found to improve the detection results and helps to ensure to bridge frames without a successful detection. Additionally, the weights for the real world deploy were trained on image sets not containing a lot of side views on the Kiwi car and thus might have trouble detecting it accurately.



(a) *An example of the cone detection. Top: original image, only cropped. Middle: segmented image after open, close, and dilate. Bottom: bounding boxes and centre base point of the box.*



(b) *An example of kiwi detection. Top left corner shows the execution time. The red frame is the bounding box drawn by the detector. The black rectangle shows the estimated bounding box by the Kalman filter.*

Figure 5: *Examples of the perception from replay of data.*

The simulation environment was used to test the actuation. Task one is shown in supplementary video four[8]. The white line and dot shows the aim point and the steering angle whereas the coloured dots shows the perceived positions of the cones. Since the cone positions are not plotted on the same frame as they are detected, they are a bit off. The steering works well, but it sometimes hits the yellow cones. Task two, where the car identifies and keeps the distance to another Kiwi, is shown in the simulation recording five[9]. Supplementary video six[10] shows scenario three where the car stops before the intersection and then starts driving when the other car has passed. In supplementary video seven[11], the same scenario can be seen but where the other car is moving throughout the whole simulation. There were some difficulties to detect the other Kiwi car from the side at the crossing, probably due to obscuring by cones. This problem was diminished by the introduction of the Kalman filter.

In the real world trial the software performed generally quite well. The perception was able to identify cones and the actuation steered in the right direction. The speed did, however, need to be higher and more stable. With some tweaking of the PID parameters the steering performed better. Full recordings of the Kiwi's perception from the demonstration event can be seen in supplementary video eight[12], nine[13], and ten[14].

4 How to Reproduce the Results

To run the simulation for the different scenarios, first run run-task1-simulation.yml, run-task2-simulation.yml, or run-task3-simulation.yml. For task two and three, open up a web browser, go to `http://localhost:8082/`, and activate the joystick to steer the second Kiwi car. Then, for all scenarios, in another terminal, run run-perception.yml and, when the two perception microservices have attached to shared memory, run run-actuation.yml.

[8]Supplementary video four - Path following: https://youtu.be/2wfHUMjSjqQ
[9]Supplementary video five - Kiwi following: https://youtu.be/t8Jtmg1M5jc
[10]Supplementary video six - Crossing 1: https://youtu.be/zyebwKDiwek
[11]Supplementary video seven - Crossing 2: https://youtu.be/vvyRg1DMmxY
[12]Supplementary video eight - Steering Visualisation at Demo Event 1: https://youtu.be/m3PFZHwfXAc
[13]Supplementary video nine - Steering Visualisation at Demo Event 2: https://youtu.be/ITo420ZLc7Y
[14]Supplementary video ten - Steering Visualisation at Demo Event 3: https://youtu.be/3n8ScaNHz5I

References

[1] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *ArXiv*, vol. 1804.02767, 2018.