Teknologiprosjekt

Autumn 2020

Final Report

Group 8

By

Kim Andre Bech, Ali Isa Hamou,
Abdelaziz Qassi, Husam Albozid,
Jamal Aljajan, Kristoffer R Pinås

Table of Contents

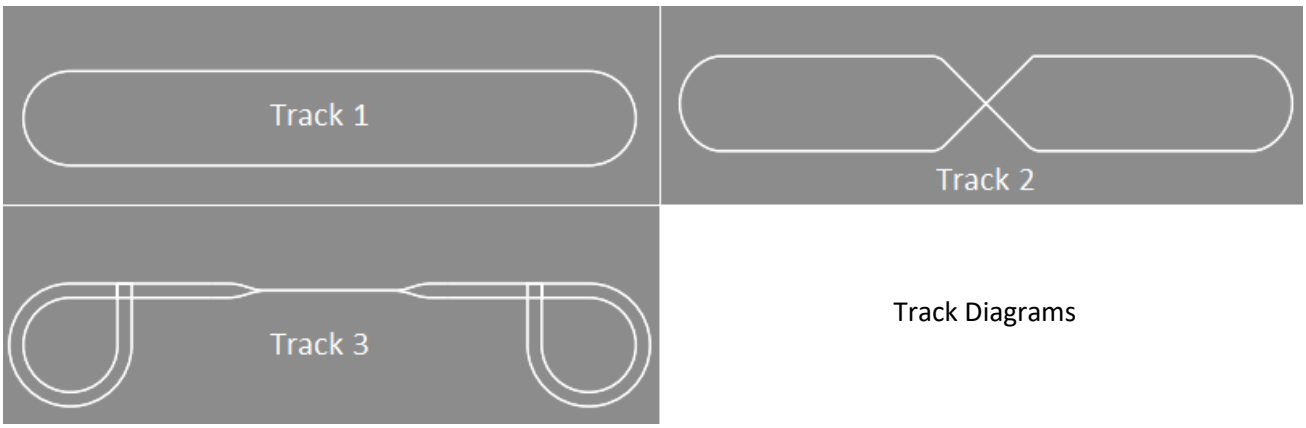
Part 1: Technical Part

1. Closing summary

The requirements for this project are to develop software for a gopigo3 robot, with the following details:

- The robot shall be able to follow three given tracks. The first track is a simple oval shape, the second track is in the shape of an infinity symbol, the third track is more complex with branches, double lines and intersections.
- The robot shall stick to the driving line.
- If there are more than one lane, then drive on the right-hand side lane.
- At every intersection, turn right.

All aims have been met. The robot is able to follow all given tracks. The robot's sticking to the driving lane is ensured by a PD controller as well as multiple image processing methods. The robot is following the rightmost edge of the lane, making it always drive on the right-hand side lane when multiple lanes. At intersections there is implemented a system that allows choosing between turning right and driving straight. This can easily be extended with more choices.



Track Diagrams

2. Benefits and significance

Automation technology has seen massive advancements in the last 20 years and have transformed many aspects of modern society. Today, automation technology is used in a wide range of different commercial and industrial applications. The basic principle of automation is to preform procedures and processes with little to no human assistance, and the methods of which this is achieved have evolved greatly over the years and varies depending on the task in question. There are multiple motivational factors for increased automation, but the biggest and most universal factors are usually efficiency and cost. As mentioned, varying methods of automation are used depending on the application, but something they all have in common is that automation is achieved using control systems.

Various forms of low-level automation have existed in commercial applications for some time, industrial applications have been by far one of the biggest drivers for this technology. Using technologies like PLS, robotics and computer vision, large parts of modern industrial prosses have been automated. However, one commercial field that have gotten a lot of attention in the last few years have been automated vehicles or self-driving cars. While low-level automation features have been present in cars for some time like cruise control, self-driving cars take it to the next level.

The general idea is that, using methods like image analysis and machine learning technologies, onboard software algorithms can use the provided sensory data to sense the current surroundings of the vehicle. Using this information, the onboard software can make decisions that have normally been made by the human driver.

Presently, only partially autonomous vehicles are available, where automated systems can perform many of the driver related responsibilities, but human oversight and assistance are still required. In the feature however there is a possibility that fully autonomous vehicles will be present on public roads, without any form of human assistance. In this project we are making a line follower robot using an Arduino based GoPiGo kit. Our main task is creating a software solution that will prosses the raw sensory data from the robot mounted camera and use this information to control the robot. While on a much smaller and simplified scale, we are employing multiple fundamental principles used in automated vehicles. Through image analysis the camera is used to sense the robot's surroundings, and in order to control the robot, we must design a control algorithm that will process this sensory information. This project will give us, the students, knowledge and capabilities to participate in similar projects. This is beneficial since there is a huge further potential within this field.

3. Methods

3.1. Overall solution architecture

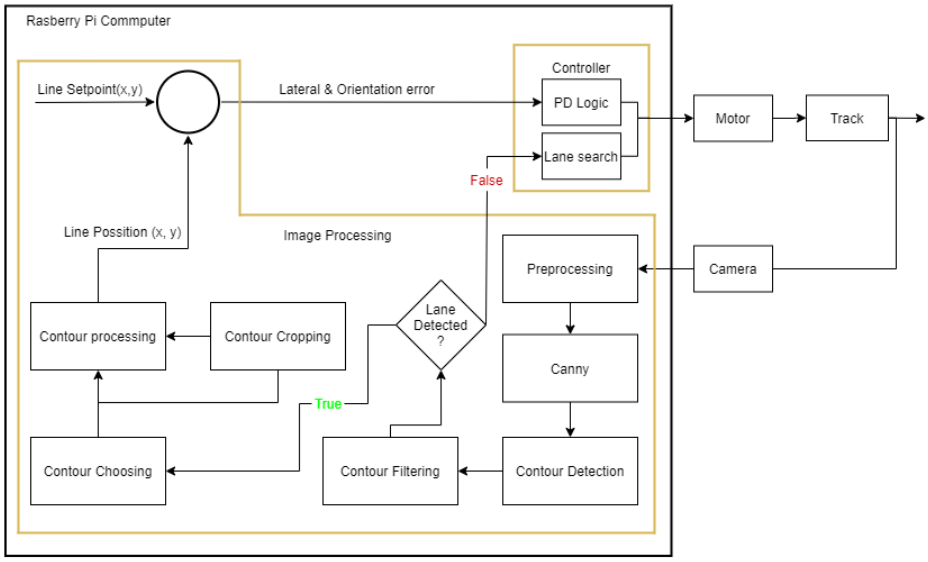
The processing is done with functions from Intel's OpenCV library. The Canny edge detection is used to detect the edges in the image, and Contours for joining all continuous points outputting a list of arrays with coordinates.

To choose the correct contour to follow, the contours extremal points are extracted, making it possible to sort the contours in various ways, depending on the track the robot is following. On lane 3 we chose to follow the rightmost contour, resulting in the robot always turning right at intersections or branches. On lane 2, the infinity shape, the contours are sorted in a way that make the robot drive straight in an intersection.

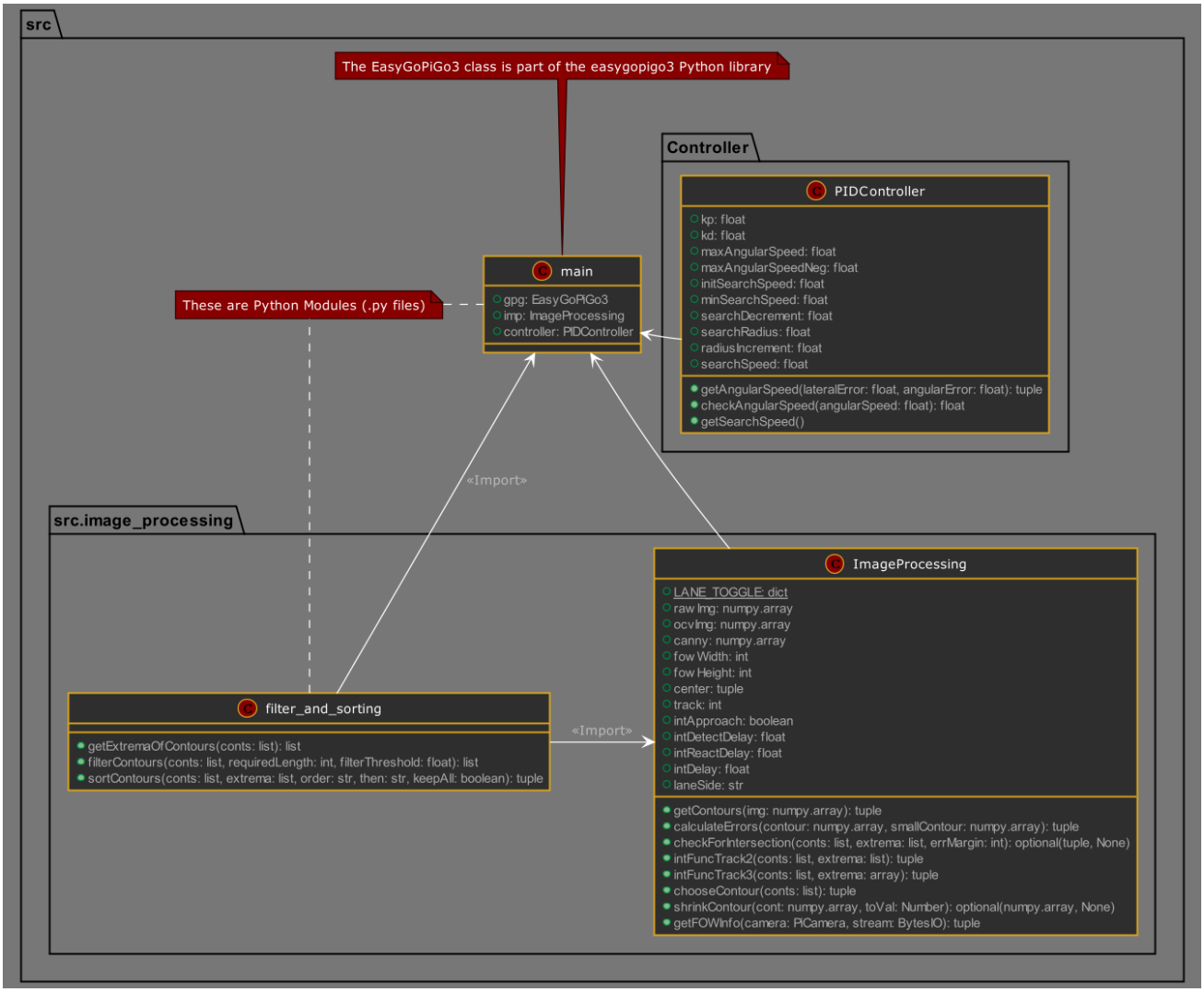
The chosen contour is fitted with a bounding rectangle for calculation of lateral error between center of field of view and the center of the contours bounding rectangle. The contour is also fitted with a minimum-area rectangle outputting angle used to calculate orientation error.

The controller used is a proportional derivative PD controller, outputting angular velocity. This is then fed into a differential-drive based unicycle model, calculating speed for each wheel.

Line follow robot block diagram and class diagram:



Preprocessing Components: Down sampling, Grayscale, Thresholding
Contour Choosing Components: Intersection detection and Contour sorting
Contour Processing Components: Applying BoundingAreaRect and MinAreaRect on contour/s.



Camera and image processing

The camera stream is established using the BytesIO library. Images from the camera is only held in memory, not stored locally. There as JPEG format. Testing showed that storing the images at the hard drive, or with PNG format, would result in the framerate decreasing noticeably. The image is converted to a PIL-format (Python Imaging Library) and then to a OpenCV-format by converting the PIL-image to a NumPy array.

The processing is mainly done with Intel's OpenCV library. OpenCV is a well-established computer vision and machine learning library, with more than 2500 optimized algorithms.

Down sampling

Due to the Raspberry Pi's limited CPU and memory, we encountered some issues during development. This was expected. The problems experienced was mainly memory related, resulting in system crashes and low framerate. It was here necessary to focus on code optimization and removal of any unnecessary information from the images captured. The resolution was lowered, and the image was transformed to a black and white binary image. A good framerate is crucial for controlling the robot's movement. For further development it is recommended upgrading to Raspberry PI 4.

Initial filtering

The raw image is first converted to a grayscale image. Thresholding is applied, comparing pixel value against threshold value. Pixels with value below the threshold value is set to 0, and pixels with value above threshold value is set to 255. This will return a binary image. Testing showed that this severely reduce noise when applied before the Canny Edge detection algorithms.

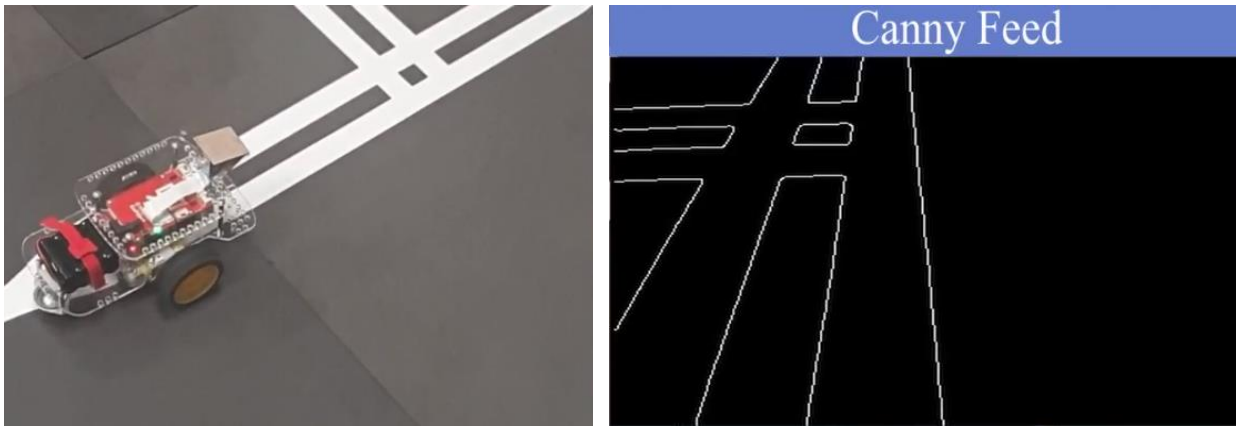
Edge detection and contours

OpenCV Canny edge detection is a multi-stage algorithm consisting of noise reduction, gradient calculation, thinning of edges, thresholding and edge tracking. The canny function takes four input parameters. First the image, second and third parameters are the low and high threshold values for step 4. The fourth parameter is kernel size, set too 3 (3x3) by default.

Canny edge detection steps:

- Noise reduction is done by a Gaussian filter that reduces noise and smooth the image.
- The gradient calculation detects the edge intensity and direction of the edge, and output two matrices.
- Thinning of edges is then done with an algorithm that goes through the edge intensity matrix provided by the gradient calculation step. The algorithm scan, finds and keep all max value pixels in edge direction.
- Thresholding is done by classifying pixels in three groups after intensity (strong, week and non-relevant). The strong we are sure contribute, non-relevant are discarded and week is taken care of in the next step.
- Edge tracking is transforming week pixels into strong if at least one pixel next to it is strong.

Canny will return a binary image with inverse colors. White edges on black background.

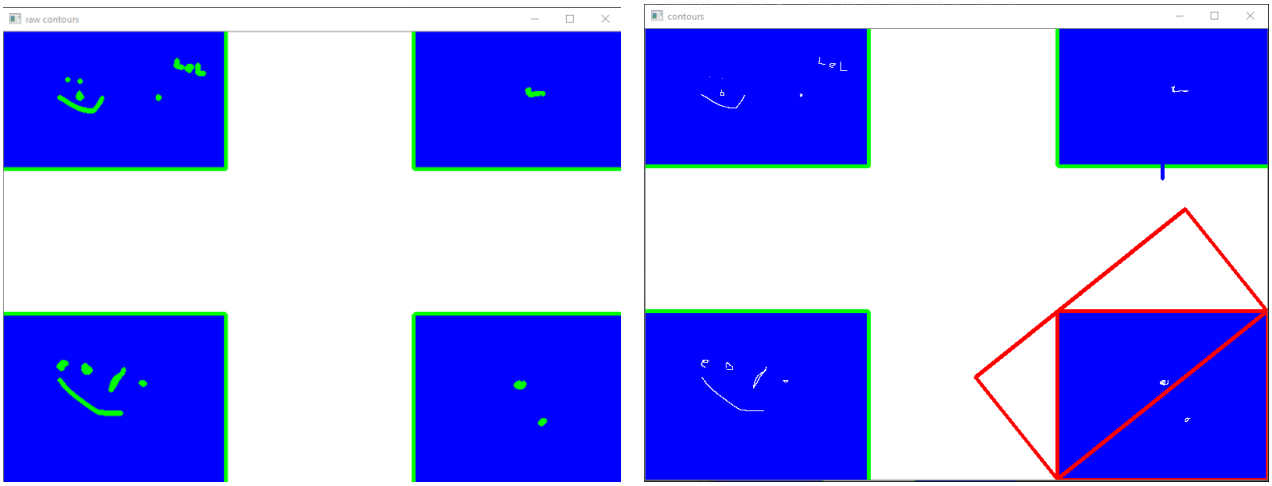


OpenCV Contours is then applied, with the canny image as input. Contours output a list of arrays, containing (x, y) coordinates of the edge's points. A contour is defined as a curve joining all the continuous points, having the same pixel value. It divides the Canny image into multiple contours. The "Canny Feed" image provided above will consist of 7 contours. Contour is usually used for finding white objects from black background.

Contour filtering

One issue we discovered quite early in the project was that noise from small contours detected in the image, made it hard to find the relevant contours that made up the track. This resulted in the robot following small non-relevant contours that were detected in the image. To fix this issue we had to implement a filtering algorithm that filters the detected contours based on their size. This was achieved using the OpenCV arch length function that calculates the perimeter of a contour using the coordinate list that defines each contour. The function finds the delta between the current and the previous coordinate in the list (dx, dy) and calculates the length between these points using the Pythagorean theorem ($h = \sqrt{dx^2 + dy^2}$). This is naturally repeated for every coordinate in the list, and when all these length gets added together, we get the approximated perimeter/length of the contour measured in pixels.

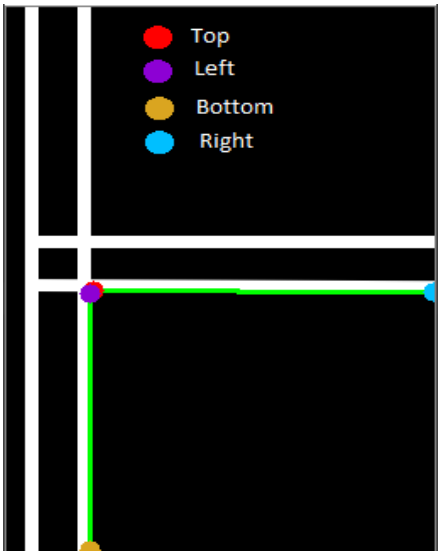
Contours detected in the image gets filtered based on a minimum length threshold, but in addition we also compare each contour with the largest contour in the image to further reduce noise. Trough testing, we ended up using a required length of 19 pixels, and a filtering threshold of 0.16 (16%). This means that all contours with a length that are less than 19 pixels, and smaller than 16% of the biggest contour in the image, gets removed. In a situation where all detected contours are smaller than 19 pixels, the lane hasn't been found so the filter algorithm will return an empty list, which will make the robot search for the lane.

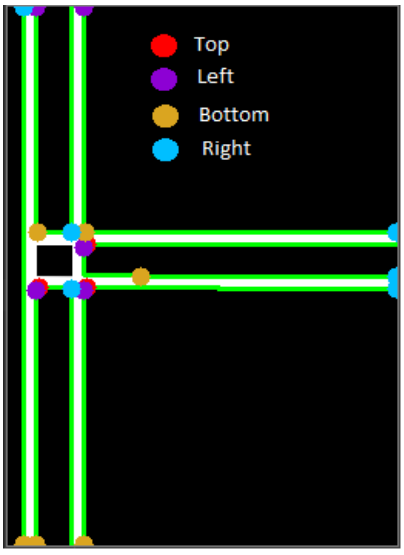


From simulation. Left image shows all detected contours in green color. Right image shows contours in green color after filtering on contour-size. The contours here is also sorted, making it possible to choose the bottom right contour. The chosen contour is also fitted with a bounding rectangle and a minimum area rectangle, used to calculate lateral and angular error.

Contour sorting

In order to find the desired contour of the image, the contours that passed the filtering stage will now have to be sorted based on their location in the image. In order to achieve this, we got the idea to use the extrema points of each contour, and by searching around we found a very helpful resource (here) to implement this feature. Since a contour fundamentally is just a NumPy list of coordinates in the image matrix. The extremal point of each contour is the coordinates with the biggest and smallest x and y positions in the list. So as a result, the extrema coordinate of each contour will be as follows **Top extrema** = smallest y, **Left extrema** = smallest x, **Bottom extrema** = biggest y and **Right extrema** = biggest x. Using these extremal points, we can deduce the location of each contour in the image and sort them based on this knowledge. This also allows us to accurately determine where each contour enters and exits the image, which makes it relatively easy to detect intersections.





The image to the left, depict the extrema point of a single contour (green) in the image. In the picture on the right all contours present in the image has been drawn, together with all extrema datapoints.

PS: It's worth noting that multiple extrema points overlap in the image, which naturally means that the extrema points share the same coordinate.

Intersections and branching

We define the track by counting how many exit-lanes there are from the field of view. And using these datapoints, the robot is able to detect upcoming intersections. Trough testing, we ended up defining intersections like this:

Track 2:

- 2 enter/exits points in each direction (top, left, bottom, right)

Track 3:

Due to the more complex track layout, we had to classify an intersection based on these characteristic combinations:

Enter and exits points combinations in each direction, that signifies an intersection			
Top	Left	Bottom	Right
3	3	3	0
3	0	3	3
0	3	3	3

When an intersection is detected, specialized logic is implemented to make the robot react in different ways depending on the track layout (track 2 or 3). The reason for doing this is to ensure that the robot follows the line as smoothly as possible. To achieve this, we used the sorting logic we already had implement, together with contour cropping, which is detailed below.

Contour Cropping

An issue we discovered during testing, was that the robot would react to quickly on upcoming turns and intersections, which caused the robot to stray of the line. This behavior was especially noticeable in sharp turns, due to the high curvature of the contour/lane. To alleviate this issue, we ended up creating a small cropped copy of the chosen contour, that went from the center of the image and down to bottom of the image (bottom extrema). This cropped contour is always used to calculate lateral error, and in intersections (track 3 more specifically) it will be used to calculate both angular and lateral error. If the cropping was unsuccessful or left out (intersections on track 2), the original contour will naturally be used instead. It's worth noting that we only cropped the contour based on the y value, if we also cropped in the x direction, we could have gotten even better results. However due to time limitations we weren't able to implement this feature.

Track 2 Intersection

Knowing the extremal points of each contour, allowed for flexible sorting that can adapt to the robots needs, which most notably made it possible to make the robot go straight ahead in intersections on track 2. This was made possible by sorting the contours from top-to-bottom instead of left to right, which resulted in the robot moving around the infinity shaped lane, instead of following the outer edges of the lane. To achieve the best possible behavior, we also used one special case of contour cropping, since during testing we found that creating a straight line from one of the bottom contours of the image (leftmost or rightmost) to the top extrema of the chosen contour, produced the best results. The reason behind this was that the lines in the intersection, would join in the middle to a continuous line. And as a result, only a single contour representing each side of the lane would be detected. A more elegant solution to this problem would have been to crop the contour along the x-axis, but we didn't have time to expand the contour cropping logic to support this functionality.

Lateral and angular error

The coordinates of the chosen contour can now be used to calculate position and angle of the line relative to the robot's position.

For calculation of the lateral error, a bounding rectangle is applied, surrounding the contour. This method returns the starting coordinates (bottom left of bounding rectangle) and the width and height of the rectangle. The x-position of the center of the bounding rectangle is calculated:

$$Process\ value\ (PV) = (starting\ xpos) + (rectangle\ width/2)$$

The setpoint (SP) is the center of the robot's field of view: FOW width/2. Consequently, the lateral error is SP-PV.

For calculation of the angular error, a minimum area rectangle is applied, surrounding the contour. This method outputs angle, however, the rectangle is rotated clockwise from 0 to -89 degrees to make it square. To calculate the angular error from this outputted angle, some logic is added to return a decreasing negative value if line is shifted one way and increasing positive value if the other way.

Lane search algorithm

When no line is detected, the robot will move in a growing spiral motion until lane is detected. This is done by starting off with a high angular velocity which is decreasing over time, and a constant linear velocity.

Motors and Controller

Controller

The proportional-integral-derivative (PID) controller is widely used in industrial control systems and is the control algorithm that is used by most motion control applications. It works by continuously calculate the error between the setpoint and the process value, and applies correction based on the PID terms. In many applications, only part of the algorithm is used. For instance, only P, PI, PD or in combination with fuzzy logic or other logic. It depends on the situation and accuracy needed.

Proportional term: Proportional to the lateral error by a "gain constant".

Integral: Accumulation of error over time.

Derivative: Cancel out or dampen the proportional and integral contribution when the robot's orientation is pointing towards the setpoint/line, making a smooth transition to the line.

The output/angular speed is then calculated by adding these terms together.

Tuning of the PID parameters was done during testing, watching the robot's behavior, with the goal of it being responsive, stable and to minimize overshoot of the track. A rule of thumb is that in loops where the process value changes quickly you should have a small proportional gain, and a higher integral gain. And opposite where the process value changes slowly. In our case the process value is the physical line to follow and will change quickly. The derivative term is then added to ensure a smooth transition to the track with little initial overshooting. Testing showed that the removal of the

integral part provided better results. This must be seen in relation to framerate and speed. With the framerate achieved on the Raspberry Pi 3, the integral part made the robot “move like a duck”.

The “unicycle” model

In this model the velocity of the wheels is calculated by the linear velocity and the angular velocity. The angular velocity is given by the controller.

The velocity of each wheel is calculated:

V = Linear velocity

ω = Angular velocity

L = Distance between wheels = 0,125m

R = Wheel radius = 0.033m

For the GoPiGo robot, the wheel speeds can be set by following functions:

gpg.set_motor_dps(gpg.MOTOR_LEFT, dps=VLeft), $VRight = 2V - \omega L/2R$

gpg.set_motor_dps(gpg.MOTOR_RIGHT, dps=VRight), $VRight = 2V + \omega L/2R$

The speed is measured in degrees per second of the robot’s wheels. The maximum DPS for these motors are 1000DPS.

3.3. Design theory

In this project we used methods from the OpenCV library to handle processing of the sensory output data. We used this functionality to analyze the output image of the mounted camera, in order to determine the position of the target line and the position of the robot relative to this line. Our reasons for using the OpenCV library, instead of implementing the imaging analysis algorithms ourselves, mainly comes down to time. If we were to create our own implantation of these algorithms, the potential point of failures and total project workload would have increased significantly. By using a library that have been well tested by the huge supporting community, we were able to focus more of our efforts on other aspects of the project, what would have been the point in trying to reinvent the wheel? The only real upside we would have had from implementing the functionality ourselves, is that we would have gotten a deeper understanding of the tools we were using. However, being able to use preexisting tools to solve a problem is just as important of a skill to have, which also required an understanding of the tools being used and their underlying principles.

The most important choices we made during this project, was choosing the image analysis and control algorithms that we were going to use. On the image possessing side, we used the canny edge operator

to find the edges in the image and then we used OpenCV contours on the edge map to detect the lines that were present. Canny edge detection is a kernel operator that takes an image and outputs a binary edge map, with the location of the edges present in the image. This is achieved using multi-stage algorithms that first emphasize the edges present, before a second algorithm thins the edges and remove weak responses, with the resulting image holding the center position of the most profound edges. Lastly, using OpenCV contours we get a collection of all the continuous edge lines, where each edge line is a list of image coordinates that defines it.

When it comes to motion control, we originally considered using PID control as the control mechanism for this project. However, we found during testing that our PID implementation produced unreliable results. The original idea behind this decision, was that by adding the integral term that sums the error over time, will negate the lack of curvature information that we had on the lane. We based this theory largely on the fact that PID controllers are widely used in industrial applications, where precise and continuous control is of paramount importance. When the PID control mechanism wasn't preforming as well as we had hoped, we started looking for other options. We ended using the PD control algorithm used on the Robotarium robot in Øving 1. In that project the curvature of the lane could be calculated and accounted for in the control algorithm, since the radius of the lane was a given constant. We originally thought that the lack curvature information on the lane would be a problem, but after implementing and testing the PD algorithm, this didn't seem to be too big of an issue.

4. Results and recommendations for further development

The goal of this project was too was to develop a solution that makes a GoPiGo3 robot move along 3 predetermined tracks, by following the lines that make up the track. We think our final solution solves the given problem in a good manner, with very promising results. There are of course areas where the results good have been better, like the robots could have moved more smoothly in intersections for example. However, we think the core aspects of our solution gives a solid foundation for further developments, that can net even better results. Our contour cropping implementation can for example be built upon further, like we detailed earlier. And theoretically it should also be possible to find the delta between the contours that make up a line in the track, given the flexible sorting algorithm that have been developed during this project. By doing this, it should be possible to translate the position of the selected contour, so that the robot follows the center of the line instead of the line edges/contours.

4.1 Document results

- The robot can search for lines, detect lines and filter noise. This is done mainly by already existing algorithms and techniques from the OpenCV library, but some custom configuration was necessary.

- The robot can choose which line to follow with a sorting algorithm.

- The robot can stick to the line detected. This is done with a PD controller.

- The robot is able to complete all 3 tracks provided, with ability to change the "turning logic" depending on the track. Always drive straight, or always turn right in an intersection. Here there is opportunities to further develop the solution with more manual alternatives or implement a solution where the robot chooses what way to turn based upon for instance a shortest path algorithm.

Result documents can be found at:

GitHub link: https://github.com/kriap139/GoPiGo_project

GitHub-repo/src → Project source code
GitHub-repo/src/testing → Code used during testing
GitHub-repo/src/testing/test_images → test images used in this project

Video-folder: https://hiof-my.sharepoint.com/:f:/g/personal/kristop_hiof_no/EnjTpY_RiOFIl-2v8CdqnxoBtDo3A6p2klAZXWnWwh3McQ
Folder-Password = robots1234

Video-folder/bane X → Unedited videos filmed during this project. $x \in \{1, 2, 3\}$
Video-folder/edits/bane X → Edited Videos and files from this project. $x \in \{1, 2, 3\}$
Video-folder/final-videos → Edited and unedited final videos, from track 2 and track 3

4.2 Explanation for results

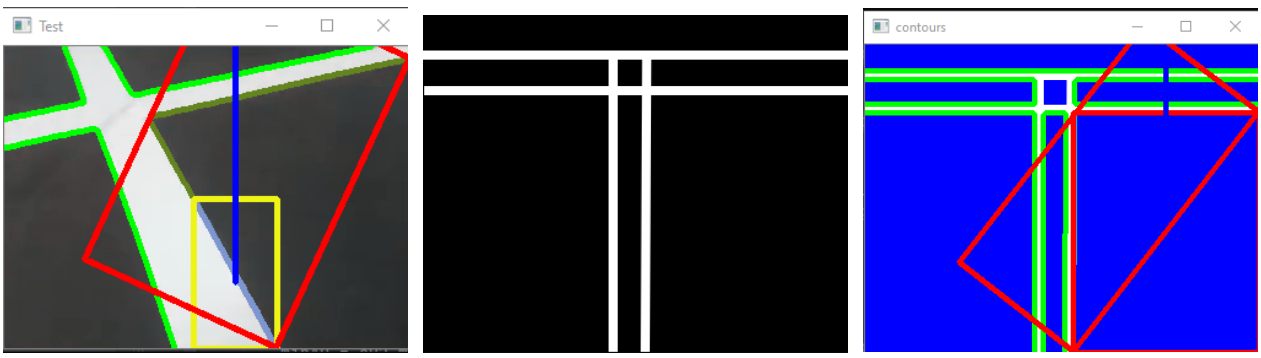
As explained under "Design Theory", the chosen strategy was to use existing solutions and python libraries if possible. Time saved from this strategy was used to further enhance the solution beyond the main goals, and still within the deadline.

Knowledge of the components of the solution and which functions and libraries to use was obtained during the planning phase. Under development we chose to mainly stick to the plan, even though multiple new ideas arose. When reflecting upon this choice now at the end of the project, we realize that this was the right choice. Entering new waters would have been too risky within this limited timespan of development. However, some parts were altered from the original plan. At those parts, the plan was either lacking, too weak, or testing showed that the planned solution didn't work as theorized. The classification of intersections/ branches, the removal of the integral term from the controller, and additional noise filtering were among the parts that was severely altered from the plan. Continuous simulation and testing during development were critical for this project's success.

Simulation

The development of the image processing part was mainly done in simulation. All individual component and functions were applied and tested in simulation before the real testing on robot and track. By allowing for experimentation and testing on the predicted behavior of the robot's components we achieved faster results. It was also allowing us to work separately and from home during the pandemic and provided insights on what to look for in the real test.

In simulation it was used both images and video. The material consisted of images and video from the robot's camera feed during testing, and custom-made images representing the various parts of the track.

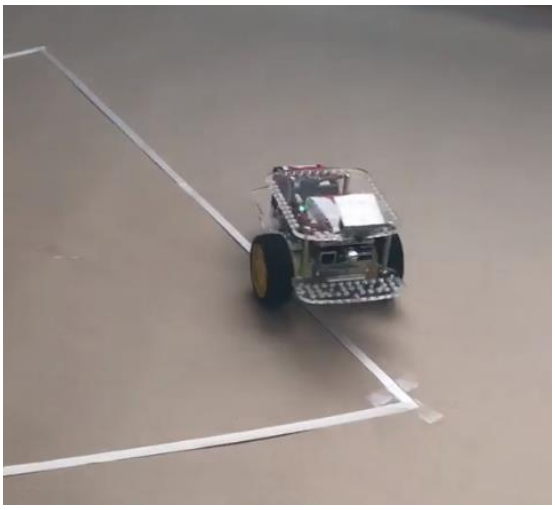


Examples from simulation

Testing

The robot testing was done at Høgskolen I Østfold Halden, with the three given tracks, and a custom track on one of the team member's living room floor. Testing on track was mainly motion control focused but would also give insights and reveal issues about other parts of the application that was not detected in simulation. This resulting in heading back to the drawing board, simulating and running another live test to see if the problem was solved.

For motion control, the controller parameters and speed were tuned during live testing, observing the robot's response to the changes.



First test performed on custom track.

Part 2: Report on business and economic case

Produkt, marked og organisasjonsformer

Presentasjon av produkt

Produktet er et Autopilot sett for gaffeltrucker, og service avtale for disse. Med Autopilot settet kan eksisterende manuelt styrte gaffeltrucker bygges om og styres helautomatisk. Settet består av en datamaskin, sensorer, kabler og programvare.

Selvkjørende gaffeltrucker, også kjent som automatiserte styringskjøretøyer, kan utføre oppgaver som en vanlig gaffeltruck, men uten en menneskelig operatør. De kan også jobbe tett med mennesker på en trygg måte. De kan løfte, stable og flytte varer trygt og raskt inne i lageret. De er programmert til å følge et spesifikt spor, utføre de nødvendige instruksjonene, og samtidig oppdage gjenstander for å unngå kollisjon. Roboten er utstyrt med sensorer for å gi et trygt arbeidsmiljø. I tillegg registrerer roboten alle oppgavene og gir en daglig rapport. Roboten kan brukes steder med både høye og lave temperaturer. Den har mange funksjoner som gir muligheten til å veie og måle volumet av pakken før transport. I tillegg kontrolleres pakkedimensjon mot hylledimensjon, før pakke plasseres for å unngå skader.

Målmarkedet

Det globale markedet for autonome gaffeltrucker forventes å nå rundt 12 162,10 millioner dollar innen 2027, og markedet vokser med cirka 7,14% i prognoseperioden 2020 til 2027. Gaffeltrucker benyttes vanligvis i lagrene og distribusjonssentrene til å løfte og flytte materialer over korte avstander. Det er bred anvendelighet av autonom gaffeltruck i forskjellige næringer som transport og logistikk, produksjon, papirindustri, treindustri osv. Dette er en viktig faktor som forventes å drive veksten i det globale autonome gaffeltruckmarkedet. Produsenter har fokus på å øke driftseffektivitet, med økende automatisering.

Blomstrende e-handelssektor over hele verden, sammen med distributører og leverandører, er tilbøyelige til å ta i bruk autonom gaffeltruck for å øke produktiviteten og den logistiske prosessen, som er forventet å øke veksten i det globale markedet. Distributører er fokusert på å øke produktiviteten på grunn av økende konkurranse og økende fragmenterte varestrømmer. Autonom gaffeltruck har en tendens til å fungere som en dyktig arbeidsstyrke, og kritisk arbeid kan utføres med høy pålitelighet og uten menneskelig inngripen.

Organisasjonsformer

Ansvarlig selskap (ANS og DA)

Et ansvarlig selskap er et selskap som kan etableres av en eller flere eiere og De har flere ulike personlig ansvar for virksomhet. En type selskap krever at hvis for eksempel selskapet går konkurs har alle eierne ansvar for gjelden. Om en av eierne ikke kan betale, må de andre betale.

Når vi snakker om ansvarlig selskap, det finnes to typer. I ANS type har alle deltakerne personlig ansvar for gjelden, mens i DA har deltaker bare ansvar opp til sin egen eierandel.

Fordel med ANS og DA

- Eierne står fritt når det gjelder kapitalinnskudd.
- Det innebærer stor avtalefrihet imellom eierne i selskapsavtalen.
- I ansvarlig selskap vil viktige beslutninger i hovedsak gjennomføres ved enighet mellom eierne.
- Alle deltakerne kan følge med i hva som foregår i selskapet.
- I ansvarlig selskap, hvis de er mer enn fem deltakere med driftsinntekter mer enn fem millioner kroner, er der revisorplikt.
- Mindre enn fem deltakere med mindre enn fem millioner kroner er der ikke revisorplikt.

Ulempene med ANS og AD

- Om en eier trer ut, vil denne fortsatt ha ansvar for gjeld, inntil kreditorene godtar at en eventuelt ny eier overtar ansvaret for gjelden.
- Det er vanskelig å selge selskapsandeler på grunn av det personlige ansvaret.
- Det er en stor personlig risiko hvis selskapet går konkurs. Eierne må dekke gjeld med personlige midler.
- Det kan bremse effektiviteten i driften fordi alle eierne må være enige når det skal tas viktige beslutninger.

Enkeltpersonforetak (ENK)

Er en bedrift for selvstendig næringsdrivende som styres av en person. Personen driver for egen regning og risiko. Dette er en vanlig type firma til de fleste prosjekter som ønsker å etablere et firma.

Fordeler med enkeltpersonforetak (ENK)

- Gratis å registrere et enkeltpersonforetak.
- Krever ikke en egen bankkonto.
- Krever ikke oppstartkapital.
- Du må ha 100 prosent kontroll og styring over aktiviteter.
- Du bestemmer overskuddet slik du ønsker.
- Du kan starte et enkeltpersonforetak ved siden av fast jobb eller skole.
- Momsfritt fram til man tjener 50 000 kroner i året.

Ulemper med enkeltpersonforetak (ENK)

- Personlig ansvar, risiko for å tape personlige eiendeler og formue hvis bedriften skylder penger.
- Høyere satser på grunn av skatt på alt overskudd og pengene regnes sammen med alle inntekter du får.
- Forventer man ikke egnet for store økonomiske investeringer.
- Har ikke krav på dagpenger, yrkesskadeforsikring, yrkesskadetrygd, pensjonsordning og redusert krav på sykepenger, fordi personen kan ikke vare er ansatt i egen bedrift.

Aksjeselskap (AS)

Er et selskap som kan styres av flere deltakere. Aksjeselskap har bestemt kapital fordelt på en eller flere andeler som kalles aksjer.

Fordeler med AS

- Man kan ansatte seg i eget selskapet. Det vil si at man kan få dagpenger, yrkesskadeforsikring, yrkesskadetrygd og pensjonsordning.
- Feriepenger og fridager med syke barn eller familie.
- Ingen personlig ansvar og mindre risiko ved konkurs.
- Minstefradrag på lønn.
- Lavere arveavgift for eiere av ikke børsnotert AS enn for andre næringsdrivende.
- krever ikke mye prosess ved forandring i kapital eller å ta inn nye eiere.
- Du får rett til å permittere deg selv.

Ulemper med AS

- Eierne til selskapet må betale arbeidsgiveravgift av egen lønn.
- Mer utgifter når gjelder regnskapsførsel i forhold til ENK, fordi regnskap skal gjennomføres i regnskapsbyrå for å unngå feil.
- Du får ikke mulighet for løpende uttak av penger gjennom året, dette må gjøres på annen måte som kravene seir. Da må eieren ta kontakt med regnskapsfører for slik prosess.

Selskapsform

Vi har bestemt å etablere et AS-selskap, noe som passer bra til vårt produkt. Årsaker til valg av AS:

- Det kreves minst 30000 kr i aksjekapital for å registre AS. Pengene kan så benyttes i selskapet.
- En annen fordel med å etablere et AS-selskap er at eieren kan ansettes i selskapet. Da får eieren i dette tilfellet like rettigheter som andre arbeidere. I tillegg til at As-selskap har ansvar for å betale lønnen til eieren.
- Eier også har mulighet til å ikke være ansatt i firma selv om han kan arbeide der. I dette tilfellet får ikke eieren lønn, men kan ta ut utbytte. Her vil eieren miste sosiale rettigheter som øvrige ansatte har.
- Kanskje den viktigste fordelen med å etablere et AS-selskap er at eieren ikke har personlig ansvar for den økonomiske biten ved konkurs, men eieren har plikt til å melde oppbud til den lokale tingretten. Etter at tingretten godkjenner saken, er eieren ikke lenger økonomisk forsvarlig og drifte firmaet. Under konkurs, vil firmaets eiendeler og utstyr selges for å dekke gjelden, mens eierens personlige eiendeler som bil, hus og klær vil ikke selges.
- En viktig fordel også er at utbetalingstidspunktet for aksjeutbytte kan gjøres innen seks måneder etter at utbyttebeslutningen ble gjort.

Årsbudsjett

Bedriften planlegger her å selge 500 enheter "Autopilot Kits" a/ 17 305.5, - eks mva. Dette vil gi planlagte inntekter på NOK 8 652 750. Det er også budsjettert inntekter fra montering og serviceavtaler for halvparten av de solgte enhetene, altså 1 250 000 kr.

Bedriften har 6 ansatte, hvor 3 er ansatt i produksjon og 3 er ansatt i administrasjon og salg. Totalt utgjør dette 4 200 000 kr i lønnskostnader, hvor halvparten er faste indirekte kostnader tilknyttet administrasjonen og den andre halvparten er variable direkte kostnader tilknyttet produksjonen (disse er knyttet mot produktet). Lønn tilknyttet administrasjonen anses som en indirekte kostnad, siden disse kostnadene ikke påvirkes av hvor mye bedriften produserer. Lønn tilknyttet produksjon anses derimot som en direkte kostnad, siden disse lønnskostnadene kan stige/synke avhengig av hvor mye bedriften produserer. For. Eks hvis salget går dårlig kan produksjonen trappes ned og lønnskostnadene kan dermed reduseres. Hvis salget øker, kan derimot flere produksjon ansatte ansettes for å øke produksjonen, noe som naturlig nok også vil øke lønnskostnadene.

Husleie er satt opp som en fast direkte kostnad, fordi lokalene hvor varen produseres kan føres tilbake til produktet ved at vi må ha et lokale for å produsere produktet. Husleien ansees som fast da det fremdeles er en kostnad ved en eventuell nedramping av produksjonen. Markedsføring er satt opp som en fast indirekte kostnad for det ofte er budsjettert en fast sum i løpet av et år til markedsføring. Det gir også mening at denne kan være variabel, ved at den for eksempel økes for å styrke salg, eller senkes ved hendelser som fører til stans i produksjonen.

Andre variable kostnader er varekostnader, svinn, strøm og frakt. Andre variable indirekte kostnader er personalkostnader I produksjon, rengjøring, avskrivninger og regnskapsfører. Totalt budsjetterte kostnader for året er 5 768 500 kr.

I balansebudsjettet for oppstart er der budsjettert 325 000 kr for investeringer i produksjonsutstyr, og 175 000 kr for investering i varer for produksjon av 100 enheter. Dette utgjør totalt 500 000 kr. Dette dekkes så ved 200 000 kr i innskutt aksjekapital fra eiere, 250 000 kr i langsiktig banklån og 50 000 kr i leverandørgjeld.

Årsbudsjett

		2021
Inntekter		
Salgsinntekter	kr	8 652 750
Inntekter montering, service	kr	1 250 000
Sum inntekter	kr	9 902 750

Kostnader			**Andel av Total**	**Fast**	**Variabel**	**Direkte**	**Indirekte**
Lønn ink sosiale kostnader	kr	4 200 000	72,81 %	50 %	50 %	50 %	50 %
Varekostnader	kr	875 000	15,17 %	0 %	100 %	100 %	0 %
Svinn	kr	17 500,0	0,30 %	0 %	100 %	100 %	0 %
Strøm	kr	48 000	0,83 %	0 %	100 %	100 %	0 %
Frakt	kr	99 000	1,72 %	0 %	100 %	100 %	0 %
Husleie	kr	240 000	4,16 %	100 %	0 %	100 %	0 %
Markedsføring	kr	50 000	0,87 %	100 %	0	0	100 %
Inndirekte personalkostnader i produksjon	kr	60 000	1,04 %	0	100 %	0	100 %
Rengjøring	kr	59 000	1,02 %	0	100 %	0	100 %
Avskrivninger	kr	100 000	1,73 %	0	100 %	0	100 %
Regnskapsfører	kr	20 000	0,35 %	0	100 %	0	100 %
Sum kostnader	kr	5 768 500					

Balansebudsjett - Oppstart (3mnd)		
Sum anleggsmidler, bygg, masiner, biler utstyr	kr	325 000
varelager	kr	175 000
kundefordringer	kr	-
Andre fordringer	kr	-
Likvider	kr	-
Sum eiendeler	kr	500 000
Aksjekapital	kr	200 000
Opptjent egenkapital	kr	-
Langsiktig gjeld, ink utsatt skatt og pensjonsforpliktelser	kr	250 000
Leverandøringjeld	kr	50 000
Annen kortstiktig gjeld	kr	-
Sum egenkappital og gjeld	kr	500 000

Beregning av kapitalbehov / drift og finansieringsplan

Til oppstarten av bedriften, så investeres det i utstyr til blant annet en testrigg. Dette er for testing, utvikling og validering av produktet. Totale investeringer utgjør 325 000 kr.

For råvarer er det budsjettert produksjon av 500 enheter for 2021. Sum råvarer for produksjon av disse utgjør 875 000 kr.

Detaljert oversikt over initiale investeringer og råvarekostnader:

Investeringer - Oppstart:	Antall		Pris		Total
Gaffeltruck				kr	150 000
Pallereoler				kr	10 000
Datautstyr				kr	65 000
Utstyr kontor				kr	50 000
Utstyr lager				kr	50 000
Sum				kr	325 000
Råvarer/ omløpsmidler:					
Sensor LiDAR	500	kr	1 000	kr	500 000
Kabler				kr	50 000
Raspberry PI	500	kr	450	kr	225 000
Misc koblinger				kr	50 000
Verktøy				kr	50 000
Sum				kr	875 000
Total inv og varer				kr	1 200 000

Fra årsbudsjettet har vi oppstarts-/driftskostnader på 4 893 000 kr og varekostnader på 875 000 kr. Dette utgjør totalt 5 768 500kr. Initiale investeringer utgjør 325 000 kr. Dekningsgraden på 61% er hentet fra dekningspunktanalysen. Følgelig 39% av salgspris er beregnet varekostnad hvor 281 214 kr er beregnet kostnad for varelager. Vi anslår et kredittsalg på 50% for dette produktet. Kredittid er 30 dager. Dette gir et kapitalbehov på 450 664 kr. Sum kapitalbehov er 6 824 878 kr.

Kapitalbehov modellen dekker et helt år, noe som gjør at større deler av kapitalbehovet dekkes av drift, totalt 6 023 360 kr. Finansiering med egenkapital og langsiktige lån fra balansebudsjett (se forrige oppgave) utgjør til sammen 450 000 kr. Dette er penger som behøves for oppstart av bedrift, inntil inntekter fra salg dekker kapitalbehov for drift. Alle råvarer handles på kreditt noe som fører til at 351 518 kr av kapitalbehovet er dekket med leverandørkreditt.

Beregning av kapitalbehov og oppsett av finansieringsplan

Inndata:

Navn/oppgavenummer:				per enhet	totalt
			Salgspris/omsetning:		8 652 750
Investeringer i utstyr:	325 000		Beregnet inntakskost/varekostnad:		3 374 573
Oppstart-/driftskostnader:	4 893 000		Manuell reg. av inntakskost/varekostn.:		
Binding av omløpsmidler:	875 000		Kostnader som påløper før salget:		
Gjennomsnittlig lagringstid varelager:	30 dg		Hvor mange dager før?:		
Gjennomsnittlig kredittid til kundene:	30 dg		Antall solgte enheter per år:	500	
Dekningsgrad/bruttofortjeneste:	61 %		Andel kredittsalg:	50 %	
Avanse:			Andel kredittkjøp:	100 %	
			Mva-%:	25 %	
Antall dager per år:	360 dg		Avrunding til nærmeste (velg tall):	1	
	Kr	%			
Finansiering med egenkapital:	200000		Evt. leverandørrabatt i prosent:		
Finansiering med langsiktige lån og kreditter:	250000		Maks. kredittid for å oppnå rabatt:		
Finansiering gjennom kortsiktige lån/driften:	6023360		Effektiv rente ved kassekreditt:		
Finansiering gjennom leverandørkreditt:	30 dg				

		Kroner	% - andel
Kapitalbehov:			
Investeringer i utstyr:		325 000	4,8 %
Oppstart-/driftskostnader:		4 893 000	71,7 %
Binding av omløpsmidler		875 000	12,8 %
Varelager	3374573*30/360	281 214	4,1 %
Kostnader som påløper før salget			0,0 %
Kundefordringer	8652750*0,50*30/360*1,25	450 664	6,6 %
Sum kapitalbehov		6 824 878	100,0 %
Finaniseringsplan:			
Finansiering med egenkapital		200 000	2,9 %
Finansiering med langsiktige lån og kreditter		250 000	3,7 %
Finansiering med gjennom kortsiktige lån/driften		6 023 360	88,3 %
Finansiering med leverandørkreditt	3374573*1,00*30/360*1,25	351 518	5,2 %
Sum finansiering		6 824 878	100,0 %
Restkapitalbehov			0,0 %

Vurdering av finansiering

Finansierings grad viser i hvilken grad anleggsmidlene er finansiert med langsiktig gjeld. Denne bør være mindre enn 1, så her ligger bedriften godt an med en grad på 0.72. Likviditetsgraden viser bedriftens til å møte betalings forpliktelser over tid. Denne bør være større enn to, så her ligger bedriften vel innenfor kravet. Egenkapitalprosenten viser bedriftens styrke til å påta seg gjeld, og bør være så høy som mulig. Denne ligger normal sett mellom 20 og 30 prosent, bedriften har derfor gode forutsetninger på å ta opp lån. Arbeidskapital må minimum være positiv og bør dekke det meste av varebeholdningen. I vårt tilfelle dekker den 71,4%. Altså 71,4 % av varebeholdningen er finansiert av langsiktig kapital.

Finansieringsgrad 1		0,72
Likviditetsgrad1		3,5
Arbeidskapital	kr	125 000
Egenkapitalprosent		40 %

Likviditetsbudsjett

Likviditet betyr at en bedrift har evnen til å betale forplikter før forfalls tid, med andre ord regninger som har betalings dato. Likviditetsbudsjettet viser forventede inn- og utbetalinger de første tre månedene.

Budsjetterte inntekter for året er 8 652 750 kr eks mva, hvorav 50% er kredittsalg. Dette blir altså 491 632 kr i de resterende 11mnd (ink mva) for både kontantsalg og kredittsalg, da det ikke forventes inntekter fra salg i januar. Inntekter fra kredittsalg er forskjøvet en måned gitt av kredittiden, hvilket vil si at 491 632 kr fra kredittsalg ikke vil innbetales før 2022, noe som avviker fra årsbudsjettet.

Bedriften kjøper alle varer på kreditt (30dager). Varekostnader for hele året er 875 000 kr, noe som gir 72 917 kr pr måned forskjøvet en måned grunnet kredittiden. Lønn for 6 personer utgjør 250 000kr pr mnd. Arbeidsgiveravgiften betales etterskuddsvis for de to foregående månedene i mars, altså 200 000 kr.

Faste kostnader er fra årsbudsjettet totale kostnader minus lønn, arbeidsgiveravgift, svinn og avskrivninger, totalt 48 000 kr pr mnd. Merverdiavgift betales av bedriften annenhver måned etterskuddsvis. 1 termin av 6 terminer totalt, betales ikke før i april. Ingen betaling av denne i februar da der ikke var drift under fjoråret. Fra likviditets budsjettet forventer vi at driften etter mars vil gjøre opp for de reduserte inntektene i januar, og februar grunnet kreditt tiden på 30 dager.

LIKVIDITETSBUDSJETT	januar	februar	mars		Sum
Innbetalinger:					
Kontantsalg		491 632	491 632		983 264
Kredittsalg			491 632		491 632
Nye lån	250 000				250 000
Ny egenkapital	200 000				200 000
Andre innbetalinger					
Sum innbetalinger	**450 000**	**491 632**	**983 264**		**1 924 896**
Utbetalinger:					
Kontantkjøp					
Kredittkjøp		72 917	72 917		145 834
Lønn	250 000	250 000	250 000		750 000
Arbeidsgiveravgift			200 000		200 000
Faste kostnader	48 000	48 000	48 000		144 000
Merverdiavgift					
Investeringer	325 000				325 000
Privatuttak					
Avdrag og renter	12 000	12 000	12 000		36 000
Sum utbetalinger	**635 000**	**382 917**	**582 917**		**1 600 834**
Innbetalingsoverskudd	-185 000	108 715	400 347		324 062
Likviditetsreserve IB		-185 000	-76 285	324 062	
Likviditetsreserve UB	-185 000	-76 285	324 062	324 062	324 062

Kalkyler – Selvkostkalkyle og dekningspunktanalyse

Ved Selvkostkalkyle finner vi prisen hver enhet må ha for at vi skal få en ønsket fortjeneste på 50%, og samtidig dekke drifts kostnadene. Med en selvkost på 5 768 500 kr og en fortjeneste på 2 884 250 (50% av selvkost) må vi ha en totalomsetning på 8 652 750 kr, noe som gir en pris per enhet på 17 305,50 kr. For kunden vil dette være en rimelig pris med tanke på besparelser i kundens lønnskostnader og effektivitet av prosess.

Dekningspunktanalysen viser hvor mange enheter bedriften må selge, for å gå i null. Med salgsinntekter på 8 652 750 kr og direkte kostnader på 3 379 500 kr, får vi et dekningsbidrag på 5 273 250 kr. Siden vi har totalt 2 389 000 kr i indirekte kostnader (se årsbudsjett), så må vi minst selge 226 enheter for å få i null (dekke de indirekte kostnadene). I kroner blir dette rundt 3 920 053 kr.

Selvkostkalkyle

Direkte kostnader	kr	3 379 500,00
Inndirekte kostnader	kr	2 389 000,00
Selvkost	kr	5 768 500,00
Fortjeneste 50%	kr	2 884 250,00
Salgspris	kr	8 652 750,00
Pris per enhet	kr	17 305,50

Dekningspunktanalyse

Inntekter	kr	8 652 750
Direkte kostnader	kr	3 379 500
Dekningsbidrag	kr	5 273 250
Degningsgrad		61 %
Inndirekte kostnader	kr	2 389 000
Fortjeneste	kr	2 884 250
Dekningsbidrag per enhet	kr	10 547
Nullpunkt i enheter		226
Nullpunkt i Kr	kr	3 920 053

Source

Part 1

General

Lectures and material provided by Maben Rabi

Significance

- **https://www.genesis-systems.com/**

Solution

Resolution:

- https://www.1stvision.com/machine-vision-solutions/2015/07/imaging-basics-calculating-resolution.html

Canny:

- https://docs.opencv.org/master/da/d22/tutorial_py_canny.html
- https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

Contours:

- https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html
- https://www.pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/

Line following robot:

- https://www.youtube.com/watch?v=tpwokAPiqfs

Bilateral filter:

- https://docs.opencv.org/master/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html

PID:

- https://www.crossco.com/resources/technical/how-to-tune-pid-loops/
- https://www.controleng.com/articles/using-pid-for-motion-control-robotics/
- http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

Unicycle:

- http://faculty.salina.k-state.edu/tim/robotics_sg/Control/kinematics/unicycle.html
- https://hackernoon.com/unicycle-to-differential-drive-courseras-control-of-mobile-robots-with-ros-and-rosbots-part-2-6d27d15f2010
- http://bert.stuy.edu/pbrooks/spring2017/materials/intro-pilot-2/programming_the_gopigo_robot.html

Part 2:

- https://hjelp.conta.no/kunnskapssidene/
- https://www.dalan.no/hvem-kan-beslutte-utsatt-utbetaling-av-aksjeutbytte/
- https://aksjeselskap.nyttiginfo.no/ansatt-i-eget-AS.htm
- https://www.infotjenester.no/artikler/8-tips-naar-du-skal-starte-egen-bedrift/
- https://www.tungt.no/logistikk/ford-har-utviklet-selvkjorende-robot-6775451
- https://www.gaffeltruck.no/
- https://folio.no/les/starte-bedrift-dette-er-de-vanligste-
- selskapsformene?gclid=EAIaIQobChMI1dSSzPH_7QIVDxd7Ch2XFQzvEAAYASAAEgI6dvD_BwE
- https://www.altinn.no/starte-og-drive/starte/valg-av-organisasjonsform/
- https://snl.no/organisasjonsform
- https://www.altinn.no/starte-og-drive/arbeidsforhold/lonn/betaling-av-forskuddstrekk-og-arbeidsgiveravgift/
- https://www.altinn.no/starte-og-drive/skatt-og-avgift/avgift/rapportering-og-betaling-av-mva/

Litteratur

- Økonomistyring / METTE HOLAN OG PER HØISETH/ 3 utgave
- Forelesninger og materiell fra timene til Tor Arne Moxheim