Intelligent cycling assistance system
Design report

Team 7

Ąžuolas Arlauskas (s2758555)
Bart Westenenk (s28048916)
Sandro Vrieling (s2769859)
Julius Kuijf (s2514001)

Table of contents

System description

E-bikes and regular bikes alike have the capability to go at very fast speeds. This could result in reckless behavior or even road accidents. Our system is made to incentivize cyclists to slow down in crowded areas and be more aware of their surroundings.

The system takes into account people in front of the user, the user's speed and acceleration to safely slow down the bike without setting a speed limit or stopping the bike.

Requirement specifications

After the first meeting with our supervisor we quickly worked on the requirement specification of our project to get a clear understanding of what the finished product should adhere to. After some time we had come up with three main goals we thought were important to our project:

- Remove the dependency of the phone the previous prototype had, including the app.
- Miniaturize the arduino subsystem to a speed management system.
- Use the bicycles power system to feed power to our system.

To accompany these goals we also made the following functional and non-functional requirements:

Requirements version 1

Functional requirements

- The system controls the speed limit based on its environment and predetermined thresholds.
- The system does not brake the bike in a way that could harm its user.
- The system shall make sure the bike stays within a predetermined speed limit.
- The system uses the power system of the E-bike.
- The system will be able to detect cyclists and pedestrians.

Non-functional requirements

- The system follows the recommendations set forth in the mentioned EU document
- The system does not rely on user provided devices.
- The system reduces the bike speed within 5 seconds to the speed limit.
- The system will take a region of interest into account.
- The system will detect an object of interest (pedestrian or cyclist) within 2 seconds.
- The system will detect an object of interest within 5 meters.
- The system will fit within the frame bag.

Requirements version 2

Later on, we received some feedback on these requirements. Out of the three goals we set at the beginning only the first was left. Based on that we revised the requirements.

Functional requirements

- The system will use a camera and pytorch to determine the density of people in the area.
- Based on the density of people in the area and the speed of the user a braking force is applied to incentivize the user to slow down. Where a higher speed or a denser area will apply a higher braking force.
- The system shall apply the friction in a consistent and predictable manner.
- The system will be able to detect cyclists and pedestrians.

Non-functional requirements

- The system follows the recommendations set forth in the study on braking behavior and safe braking speeds. https://doi.org/10.1080/15389588.2019.1643015
- The system does not rely on user provided devices.
- The system applies the braking force within 2 seconds of detection.
- The system will take a region of interest into account.
- The system will detect an object of interest (pedestrian or cyclist) within 2 seconds.
- The system will detect an object of interest within 5 meters.
- The system will fit within the frame bag.
- The system will not take the road conditions into account.

Requirements version 3

A third meeting clarified what kind of object detection is required. This slightly changed the requirements once again.

Functional requirements

- The system will use a camera and pytorch to determine the motion of all pedestrians and bicycles in the field of view.
- Based on the motion of pedestrians and bicycles in the area and the speed and trajectory of the user a braking force is applied to incentivize the user to slow down. Where a higher speed or a higher likelihood of a collision will apply a higher braking force.
- The system shall apply the friction in a consistent and predictable manner.
- The system will be able to detect cyclists and pedestrians.

Non-functional requirements

- The system follows the recommendations set forth in the study on braking behavior and safe braking speeds. https://doi.org/10.1080/15389588.2019.1643015
- The system does not rely on user provided devices.
- The system applies the braking force within 2 seconds of detection.
- The system will detect an object of interest (pedestrian or cyclist) within 2 seconds.
- The system will detect an object of interest within 5 meters.
- The system will fit within the frame bag.
- The system will not take the road conditions into account.

“Nice to have” or out of scope requirements

Whilst coming up with a list of goals and requirements, we had to prioritize which requirements were most important and necessary to have for the system to function. This meant leaving out a few requirements which were not central to the application.

User interface

Originally the program was on a phone so it was really simple to introduce a user interface. Since our job was to remove the dependency on a phone the user was left with no interface. After discussing this with our stakeholder, we came to the conclusion that a display is not necessary for the braking system to function and thus should be placed on the back burner for the time being.

Speed zones

The project we are working on is a part of a larger initiative to add regulation and more control on bike speeds. We discussed with the client the possibility of adding speed zones. This would entail mapping parts of the city by their busyness to then apply a greater or lesser braking force when in said area. This was also left for if we have time at the end since it would be an addition to the speed control and not a central part of it.

Miniaturize Arduino subsystem

When we initially got to look at the bike, the Arduino subsystem was quite large, barely fitting in the bag attached to the bike. We thought that there could be some space optimization we could do by using an Arduino Micro instead of an UNO, for example. After some feedback by the client, we realized that this is not our priority and thus it got moved to “nice to have”.

Use the bicycles power system to feed power to our system

Initially, the system was hooked up to a power bank. We thought that we could make more space in the bag and get rid of the need for a power bank by hooking the entire system up to the e-bike’s battery. However we decided that this is simply outside of the scope of our project.

Taking external factors into account

During brainstorming and peer review sessions the thought of external variables came into mind quite often. Most commonly brought up was taking into account the material/texture you are driving on, the reaction time of the user and the weather conditions. We decided to rule these out as out of scope as it is very difficult to reliably and dynamically detect these things.

Road detection

Another thing we considered is detecting the edges of the road to give us an area of interest – the bike path. This would help us distinguish what detections are actually important and which detections are in which lane. However this idea only came about mid way through the project and to not convolute an already complex system we decided to keep this out.

RPi Camera

Originally we were supposed to develop this application to run using a Picamera 3 NoIR however we could not find a proper way to secure it on the bike. Since we lack the skill to make the housing ourselves we decided to replace the RPi camera with a GoPro adjacent camera.

Dynamic braking angle range detection

We also considered detecting the appropriate angle range for a bike dynamically. However we realized that this would require us to know the wear and the type of bike that is being used which is outside the scope of our project.

Prototyping

To begin we created separate proof of concept applications to figure out how we wanted to implement the project.

Software

HumanDetector

The HumanDetector went through a lot of prototyping. We decided on YOLOv8n as soon as we got the details of the project as it was a newer version of the model already being used in the previous version. However we still wanted to run a test to see if it is reliable. We wrote a short program that performed the whole detection process and sent a signal via serial to the Arduino to turn on the onboard LED. The prototype was successful so we chose to move forward with implementing YOLOv8n in our system.

YOLOv8n was then implemented into the system according to the initial design but around this time a field of interest idea was introduced to classify detected pedestrians and bikes into being on the left, on the right or in the center of the feed, leaving out the detections on the sides.

However after implementing that, we had another pivot as we decided to also track the direction of movement of the detections. This helps us tell apart what is moving, what is stationary and if the detections on the sides are also a threat.

Around week 7 we realized that the detections are not as accurate as we would like them to be. For a moment we considered changing the model entirely. After searching for a model better fit to our needs we realized that we would need to train one ourselves. Since this was quite late in the project we decided that there is not enough time to create a quality model ourselves so we continued working with the model we had, trying to account for the distance error.

Moreover, during week 7 we realized that the Raspberry Pi may not be strong enough for our purpose so we started optimizing the processes. This started by changing to YOLOv8n NCNN which is the version for embedded devices.

CoordinatorService

The coordinator service was initially supposed to handle the entire decision making process – collect the sensor data, have a lot of logic for when the braking force should be applied and calculate said braking force. This changed in week 4 when we redesigned the coordinator class to include policies for ease of maintainability and modularity. Now we have a few different scenarios (or policies) that we want to cover that get checked separately and the correct outcome is chosen and applied in the CoordinatorService by checking for the most severe braking angle as that means that the worst situation is considered and will be mitigated. In a sense, that makes the most severe option the safest option.

Hardware

Camera

In the beginning, we were asked to use a Pi Camera 3 NoIR. However, around the middle of the project we started using a GoPro for testing purposes instead and later decided to stick with the GoPro. We did this because it was difficult to reliably set up the PiCamera for testing before we had a proper case for it. Since we could not find a case for purchase or to 3D print and did not have the skills to model our own, we ended up sticking with the GoPro.

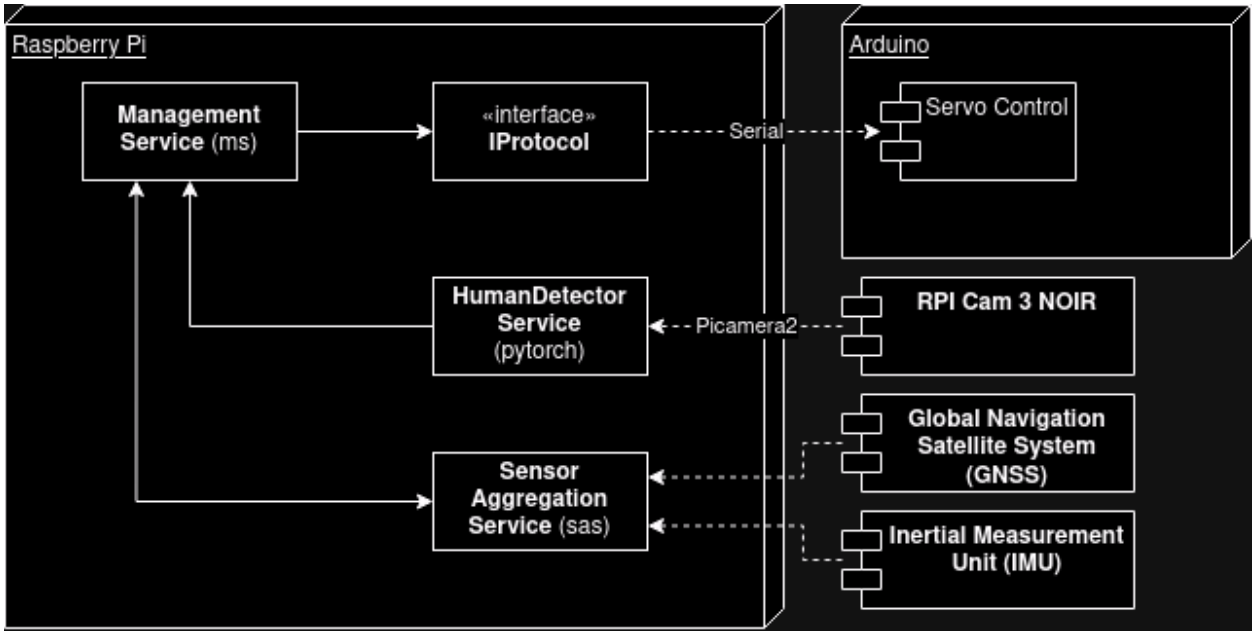
SBC

Originally, we were using a Raspberry Pi 4 as we were more familiar with it. The Jetson Nano was offered to us from the very beginning but since we had not worked with it before we chose to try the Pi first. However, as we found through integration testing in week 8 – the Pi was the bottleneck for the performance of our project and since the Nano is made specifically for AI use, we thought that it might be a better fit and an upgrade to the current system.

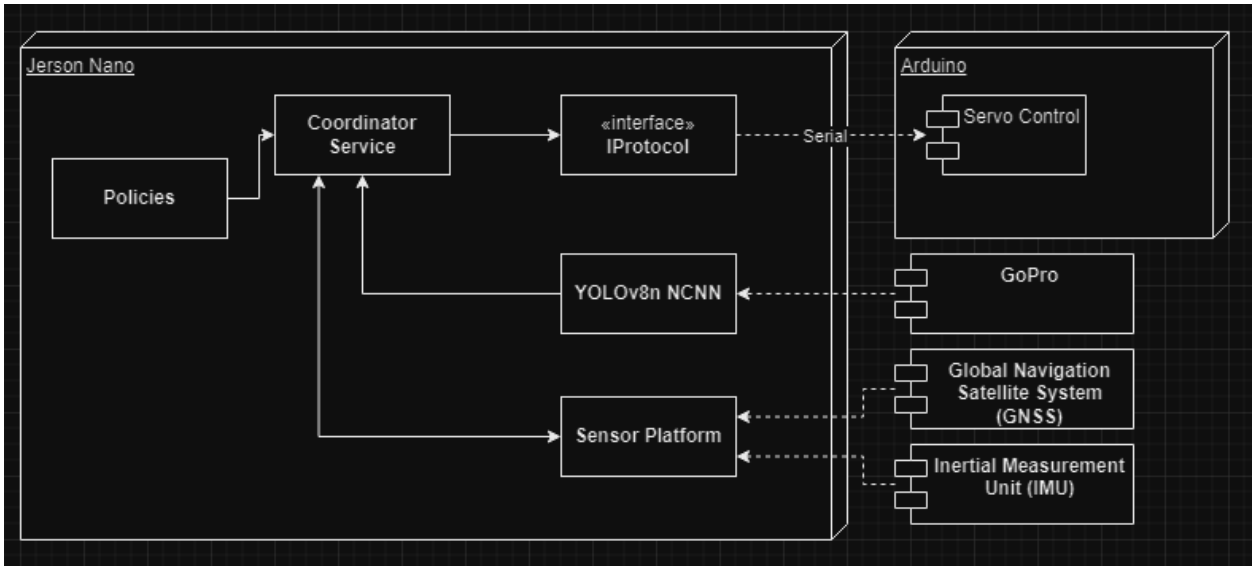
Global & detailed designs

Physical design

For our project we were given access to a few hardware components. These initially included a Raspberry Pi 4, an Arduino UNO, a servo motor, a Pi Camera 3 NoIR, a GNSS, and an IMU. Our initial hardware design was as follows:



Later, we also got access to the Jetson Nano. With further updates came an updated version of this diagram:



Jetson Nano

The Jetson Nano spearheads the entire system. There used to be a Raspberry Pi 4 in its place but we decided to switch to the Nano towards the end of the project. More about that later.

It hosts a Python system that collects data from sensors and uses the data alongside calculations designed through testing to calculate the angle to be applied to the e-bike's front tire.

Arduino Nano

The Arduino Nano has one responsibility – listen to the Jetson Nano and activate a servomotor at a specified angle.

Servomotor

As described above, the servo is connected to the Arduino to brake a specific amount. This motor is directly connected to the brakes and applies force to activate them.

Camera

The camera is implemented using a GoPro. The camera is used to detect incoming pedestrians and bicycles. While implementing the HumanDetector we used three different cameras for testing purposes – the PiCamera 3 NoIR we were initially supposed to use, a laptop webcam, and a GoPro. More about this in a later section.

GNSS

The GNSS is implemented using a ZED-F9P GPS-RTK HAT. The GNSS is used for positioning and serves as a speedometer. Knowing the speed of the bike allows the system to determine

where the bike will head and from that we can determine if any pedestrian or bike will be in its path.

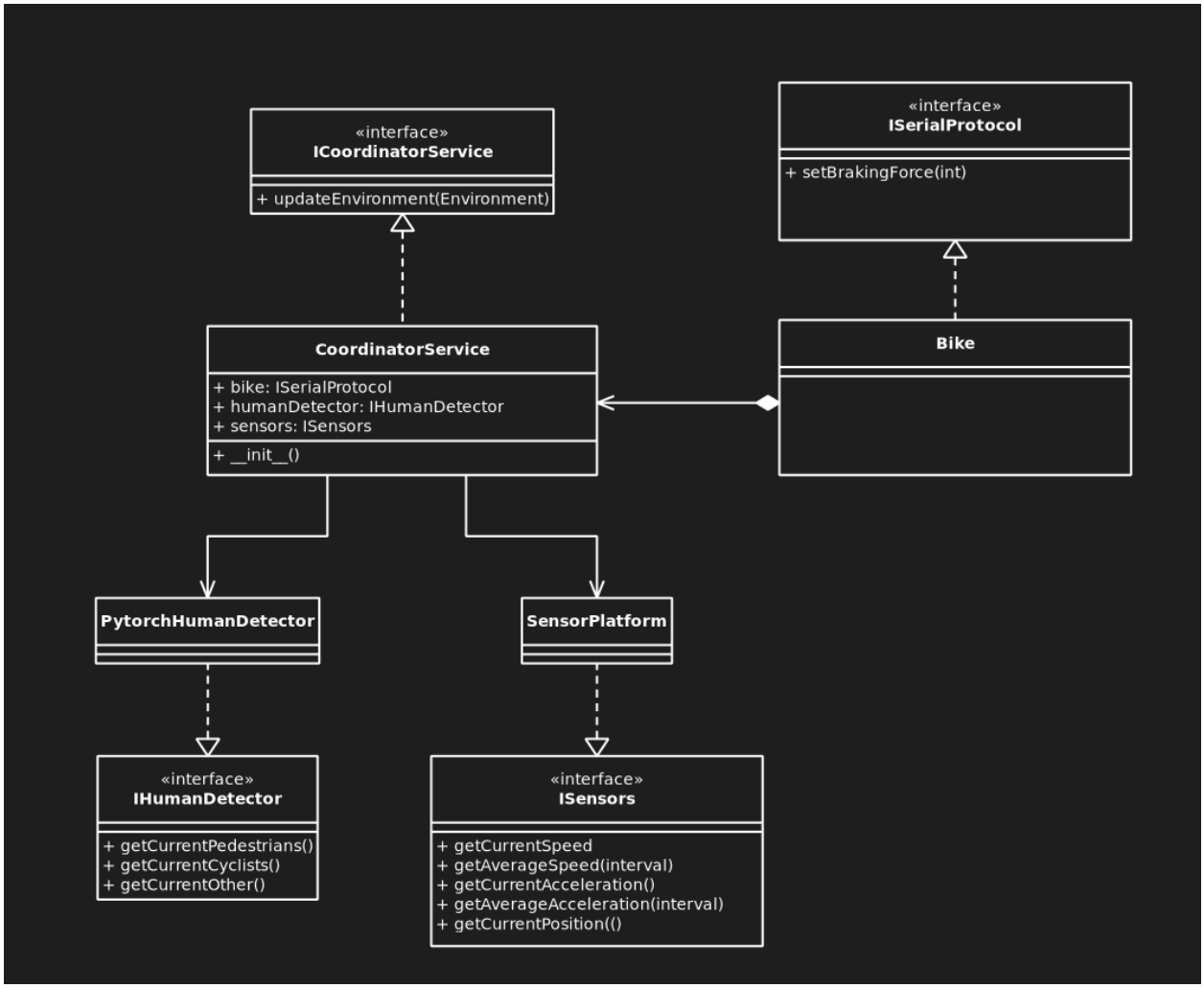
We were also asked to implement RTK for the GNSS for more precise location tracking. We got access to a base station and managed to get corrections of the data on our laptops but were not able to find a way to connect to the base station on the Jetson Nano.

IMU

The IMU is implemented using an MMA8452Q. It currently serves no purpose but has been added as it will be essential if this project is ever expanded upon. A future version of the system could implement trajectory prediction in which case it is essential to know the acceleration, rotation and direction of the bike which the IMU would provide.

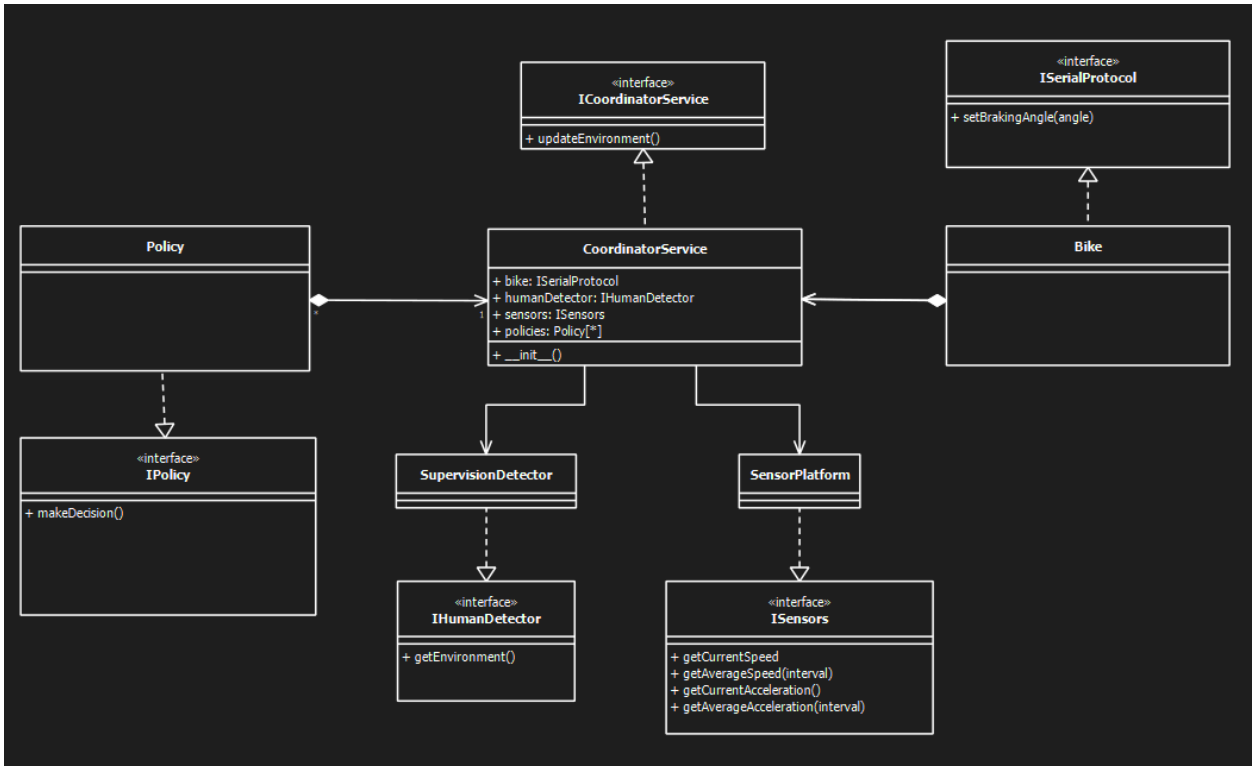
Class design

All of our classes are implementations of interfaces to add modularity to the project, making it easier to replace and test separate parts.



This is what our initial design looked like, the tasks the final system needed to perform were split up over multiple classes.

During the project we realized that the Coordinator service would need to handle a lot of different scenarios. To split up these tasks we have implemented a Policy class that handles the break level for a certain scenario and sends its response back to the coordinator service. This led to the following design:



SensorPlatform

The SensorPlatform, implementing the interface ISensors, is a service that receives data from the GNSS and IMU. It calculates the following:

- Current speed
- Average speed (over given interval)
- Current acceleration
- Average acceleration (over given interval)
- Location (latitude and longitude)

SupervisionDetector

The SupervisionDetector, implementation of the IHumanDetector interface, uses the camera to determine the motion of all pedestrians and bicycles in its field of view. We decided to implement an already existing and efficient model for this purpose. We chose YOLOv8, more specifically the Nano model as it is the fastest and best fits a real time program like our own. We later changed this to an NCNN model which is made for embedded devices. By doing some processing we are able to figure out the depth and track the relative distance and relative velocities of the detections.

For each detection, we collect the following:

- An ID number
- The type of detection

- The relative position of the detection in the field of view
- The relative velocity of the detection for both x and y axis

CoordinatorService

The CoordinatorService, implementing the ICoordinatorService, is the main service that receives information from the SensorPlatform and SupervisionDetector and uses their data to make the decision on how much braking power should be applied. The decision is made by polling different scenarios we call policies. The most severe decision is then selected from all the policies as it is the safest scenario. It will then send this value to the Arduino over serial.

Bike

This class is an implementation of the ISerialProtocol interface and is used for communication between the Jetson Nano and the Arduino. It is used to establish a connection and using a certain protocol will relay to the arduino what angle to apply.

Policy

A policy in our system is a scenario where the user should slow down. The most basic example is the SimpleDistance policy which activates the brakes to slow down if there are detections in front of the user within a specified range. Policies return a braking level from 0 to 10 where 0 is no braking force, 1 is a 150 degree braking angle and 10 is a 170 degree braking angle. The angles are easily adjustable in main.py as arguments for the coordinator service. These numbers may differ bike to bike depending on the type and wear of the brakes but dynamically detecting it is, as discussed previously, outside the scope of our project.

We have implemented the following policies:

- SimpleDistance: described above
- SimpleCrowd: takes the amount of relevant detections and the user's speed into account to decide the braking power when approaching a crowd.
- UnsafeOvertake: will look if there is a person in front the user wants to overtake, if there is something blocking the way it will brake (this has been disabled due to issues detecting cyclists)
- TimeToCrash: will look if the user is headed into oncoming traffic, brakes accordingly

Testing

Testing the relationship between braking angle and deceleration

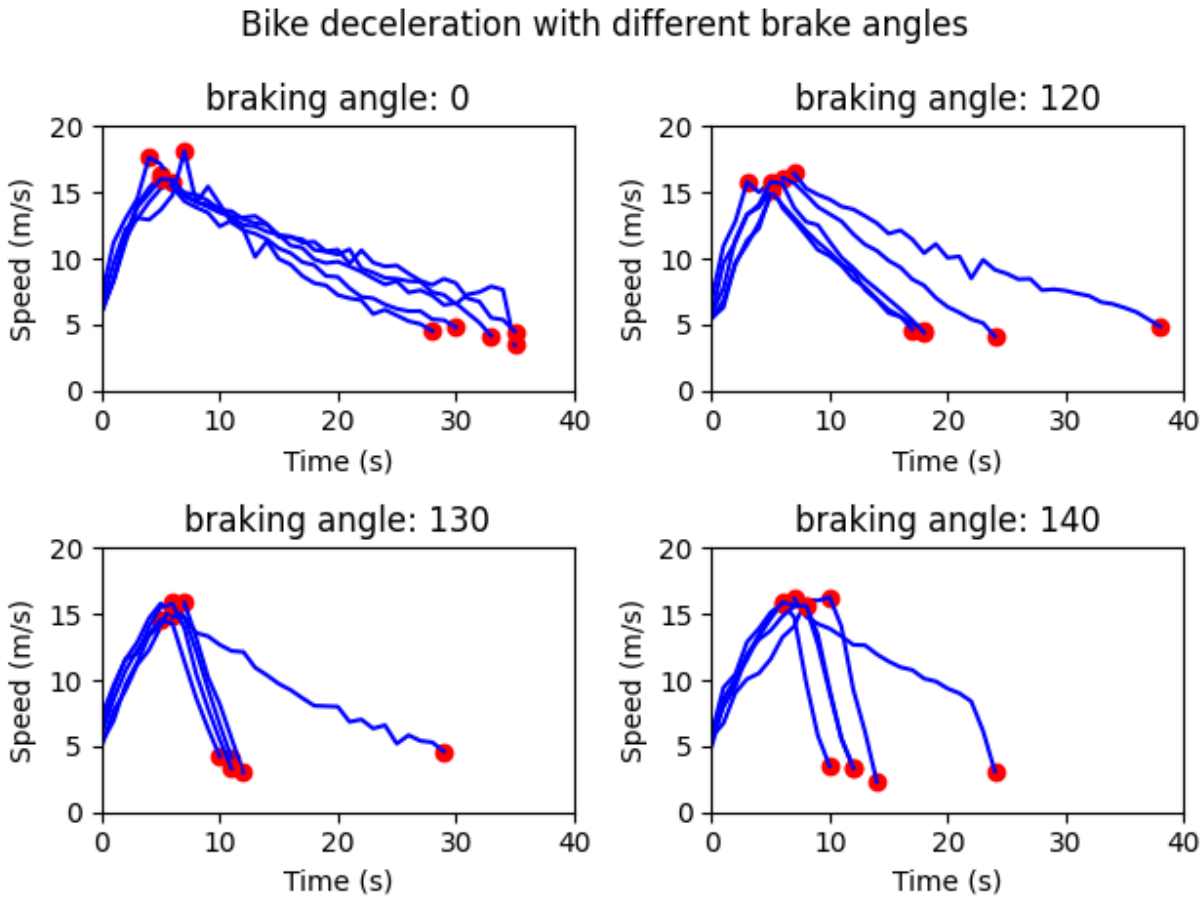
Plan

It is very important for the proper functioning of the bike that we know how fast the bike decelerates at certain brake angles. Braking using a servo motor like in this project is not done very commonly so it was necessary to do some tests.

To perform these tests a program was written so the angle of the servo motor could be changed on the spot. A button would be pushed to activate the servo, therefore enacting a braking force. For all testing scenarios the bike will be driven up to about 15km/h and then one of five braking angles would be applied whilst simultaneously halting the peddling. The time between the braking angle activation and the bike coming to a standstill would be measured to determine the deceleration. A control will also be performed where the servo will not be activated and the normal deceleration without any additional force will be tested. Each scenario will be tested 5 times to gain enough data. The relationship between the velocity and deceleration is linear under normal conditions so only one bike speed will be tested.

Results

The average deceleration of each test was calculated between the highest speed and the end time of the run, just before standstill. Both the start and endpoints for the interval are noted with a red dot in the graph below. The average deceleration for each of the braking angles is given in the table below.


Bike deceleration with different brake angles
braking angle: 0
braking angle: 120
braking angle: 130
braking angle: 140
Speed (m/s)
Time (s)
0
5
10
15
20
0
10
20
30
40

		0°	120°	130°	140°
acceleration (m/s^2)	**Test 1**	-0.117	-0.104	-0.123	-0.214
	Test 2	-0.180	-0.221	-0.576	-0.970
	Test 3	-0.110	-0.225	-0.652	-0.855
	Test 4	-0.137	-0.185	-0.716	-0.577
	Test 5	-0.118	-0.244	-0.692	-0.855
	Average	-0.132	-0.196	-0.552	-0.694

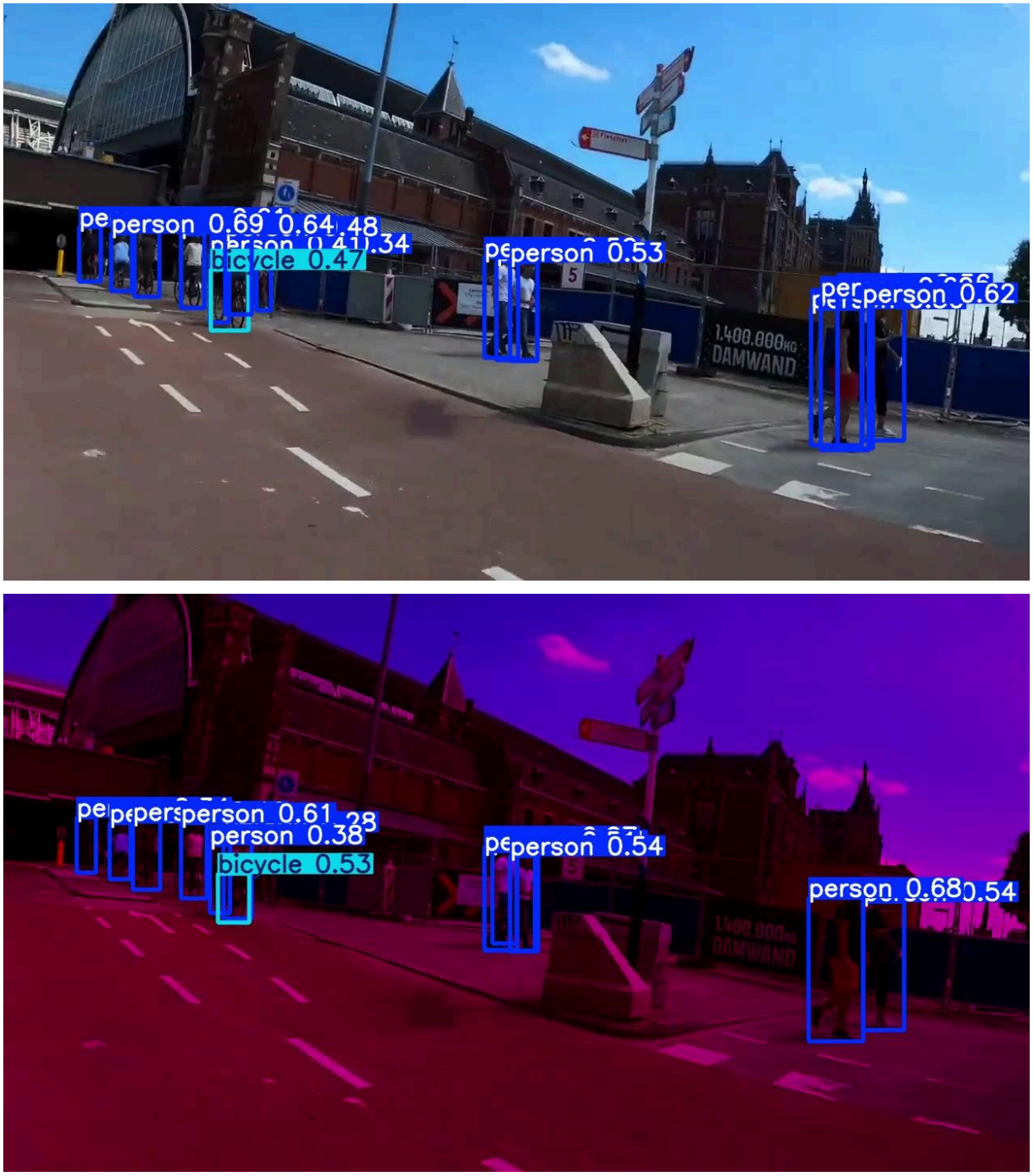
Testing the error of the NoIR camera

Plan

Originally we were supposed to use a Pi camera 3 NoIR, the feed of which is purple. We were unsure if the fact that the video feed is tinted magenta would cause problems with pedestrian and bike detection. We decided to test this by taking a video and passing it through the model twice – once with regular colors and once with a magenta tint. Both files were passed through the same model without any corrections.

Results

The test resulted in a clear answer – the color tint does not really matter. Here are the first frames of the normal and the tinted video:



As you can see the detections are almost identical with almost identical confidences.

Integration testing session 1

Plan

For the first integration testing session we wanted to test 2 of the core policies – SimpleDistance and SimpleCrowd. Both of these needed the separate parts to work together well so we thought this would be a good starting point.

We were planning to run the program and drive towards a person in an empty space to test the SimpleDistance policy and then drive near a crowded area to test the SimpleCrowd policy. Both of the tests are quite dangerous as they require us to trust the braking system but we tried to execute them with the utmost attention and care to make sure an accident does not occur.

Results

From the initial testing we learned quickly that the system was too slow to react in practical situations. We first tested the SimpleDistance policy without the bike moving – just by standing too close to the bike the system should activate. There was however a significant delay between a person entering the frame and the system reacting accordingly. From this we learned that the system would need to be faster. We discovered that the processing of the images was what was taking the longest amount of time. To solve this we decided to switch from the Raspberry Pi to a Jetson Nano. The Jetson Nano has a GPU, which means it is more optimal to handle AI like we use it, and is therefore quicker.

Testing lower resolutions and lower FPS

Plan

As the Pi was too slow, we first decided to see if image size and the amount of frames it has to process made a difference. We did this by resizing every frame as the camera did not have a function to lower resolution itself. We then also tried lowering the FPS cap.

Results

With the lowered FPS, the Pi was able to process way more efficiently as before it would skip frames anyways as it was taking too long. However, even taking a lower resolution into account, the change was negligible and thus we continued porting the system to the Jetson Nano.

Integration testing session 2

Plan

After switching to the Jetson Nano it was time to redo the integration testing. In these tests we would have one person stand still and the other person ride towards them at a moderate speed (~10 km/h). The person standing still would move out of the way right before a collision would occur. We recorded the data using csv and also asked the person riding the bike at what point the braking became significant enough that it drew attention.

Results

From this test we have determined that switching to the Jetson Nano has been a major success, reducing the inference time from between .5 and 1 seconds to around 50 milliseconds. From the users perspective it also becomes clear that their bike is braking well before a collision would happen, giving them ample time to either steer the bike out of the way or use the manual brakes. However, we did have some issues during testing as well. Since we were not sure whether or not this system was final, wires were taped together unreliably and these connections would often fall apart. From this we determined that the next course of action was to actually secure the devices to increase reliability.

Integration testing session 3

Plan

After realizing the improvement of the Jetson Nano, we continued testing to make sure that the policies are calibrated well. We mainly focused on SimpleDistance once again as it was the most accurate policy at the time. We once again would run the program and drive towards a person at differing speeds to see what were the optimal settings.

Results

This test brought to our attention the massive difference in braking effectiveness depending on the speed of the bike. We decided to take a more aggressive approach when calibrating SimpleDistance to account for this and also started testing the TimeToCrash policy more as it took the speed into account. We tested the policy by running towards the bike as it started raining and we had to move inside. This helped us narrow down the parameters to have a pretty responsive TimeToCrash policy.

Reflection

Towards the end of the project we had a lot to reflect on both on the teamwork and the project side of things.

Team communication and managing expectations

As the time went on it became clear that the team was not on the same page about the quality we were aiming for, the workload and the communication within the team. We had very different expectations going into the project which caused some disputes within the team. Looking back, all three of these points should have been discussed at the very beginning of the project so everyone was on the same page. Unfortunately, since that was not the case we ended up disagreeing on the quality we wanted the product to have (in the sense of the grade), the amount of hours we work weekly and how to communicate clearly. These issues ended up causing a few setbacks but we managed to keep composure and figure out our differences with a group discussion.

Project management

When it came to project work, we realized how important it was to keep the Trello board tidy and up to date. Many times we would wander and waste time thinking of things to do because the Trello was empty or all the tasks were taken. Usually this was caused by the fact that the board was not properly organized or updated. Moreover, we realized the importance of landing on an appropriate design as soon as possible. As described before, there was an almost constant flow of changes due to unforeseen circumstances that lead to certain work being scrapped or unused, resulting in wasted time and effort. This goes hand in hand with the previous paragraph because there was a lack of communication in the team about the design, sometimes we understood it in different ways and only realized that when it was too late.

Requirements specification

We did not have full knowledge of the achievability of the requirements nor how realistic they were when we first set out to do the project. Here we reflect on how appropriate the requirements were and whether or not they were achieved

Functional requirements

- **The system will use a camera and pytorch to determine the motion of all pedestrians and bicycles in the field of view.** There were some problems mostly with the occlusion of bicycles, where the bounding box of the bicycle would be cut short either by some of the bike being out of view or because a person was sitting on the bike. PyTorch however has been very accurate at detecting pedestrians. We also discussed a potential fix for this problem using META's SAM2 but it was brought up too late in the project and we did not have time to implement it. The feasibility of using SAM2 was also in question as our system was already quite complex and heavy on the Jetson Nano.
- **Based on the motion of pedestrians and bicycles in the area and the speed and trajectory of the user a braking force is applied to incentivize the user to slow down.**

Where a higher speed or a higher likelihood of a collision will apply a higher braking force. This has been mostly successful. We have used a set of policies, one of which accounts for the user's relative speed of the pedestrians and bicycles around them and will apply a braking force relative to that speed.

- **The system shall apply the friction in a consistent and predictable manner.** There has been no issue with this. The servo motor is very precise but does not turn extremely fast which would result in sudden friction.
- **The system will be able to detect cyclists and pedestrians.** As stated before, the system has some trouble detecting bicycles and people on bicycles. A different model that would detect pedestrians and cyclists rather than bikes and people would be a major improvement. We tried looking for such a model however most were ineffective and we did not have the appropriate time to train our own model when we discovered this issue. As discussed prior, the SAM2 model could be used alongside our current model as a work around but there was not enough time.

Non-functional requirements

- **The system follows the recommendations set forth in the study on braking behavior and safe braking speeds.** Angles up until 140 degrees are completely safe however anything above has not been properly documented.
- **The system does not rely on user provided devices.** This has been fully achieved.
- **The system applies the braking force within 2 seconds of detection.** Although this requirement has been fulfilled it was also not as strict as it should have been. We had previously underestimated just how far a bike can get within 2 seconds and an application of the braking force in 2 seconds would not achieve much in imminent crash scenarios.
- **The system will detect an object of interest (pedestrian or cyclist) within 2 seconds.** The system recognises objects of interest much faster than 2 seconds in relevant scenarios.
- **The system will detect an object of interest within 5 meters.** The system is very reliable between 2 and 8 meters but becomes unreliable outside of that range. This is occasionally an issue when an object is extremely close.
- **The system will fit within the frame bag.** This requirement has not been fulfilled. It was not very realistic as we need to use a lot more parts than the previous team. Designing and printing a new holder for the system was outside of the scope of this project.
- **The system will not take the road conditions into account.** No efforts have been made to take road conditions into account.

Testing

First of all we should have taken a more structured approach to integration testing. Even though we would plan out what we would do, we were not exactly sure what we were looking to find. We would often just tinker with parameters of the system until it worked to a level we thought

was acceptable and find results as a consequence of that instead of setting concrete questions we needed answers to.

Moreover, the braking angle and deceleration testing needs to be redone. It was originally limited to angles up until 140 degrees due to the brake wire bending when going above said threshold. We originally thought that the bend would make the system unreliable. However, integration testing showed that the 140 degrees were simply not enough to make a proper impact for the user. Because of this, we ended up turning the angle way above the tested threshold. The braking still proved to be reliable with the higher angles. This does not invalidate the test per se but the test lacks coverage.

Alternatives

Braking

In our project, we used the front brakes to slow down the bike. This is due to the fact that the people who worked on this system earlier did so themselves so we trusted that they made an informed decision. We did not test replacing the back brakes but it would be interesting to experiment with.

Other sensors

Even though we used a camera, a point could be made that we should have used another sensor for depth instead of post processing. Using a camera was a requirement for our system thus we could not have replaced it with a depth sensor but we could have accompanied the camera with it. Also, the camera can be used to do roadside detection and other similar tasks which could improve the system in the future thus we still believe that a camera is a must have.

Work division

We tracked the tasks and who is working on what via Trello.

Ąžuolas Arlauskas

- Design report
- Reflection plan
- Proof of concept implementations for the YOLOv8n on RaspberryPi
- Initial designed implementation of the HumanDetector (including field of interest)
- Testing the error of the NoIR camera
- CoordinatorService implementation
- SimpleDistancePolicy implementation
- SimpleDistancePolicy unit testing

- Reflection report
- ConfigChecker
- Integration testing
- Console display
- Final presentation

Bart Westenenk

- System design
- Servo proof of concept program
- Serial communication (Bike class) implementation
- SupervisionDetector with speed and movement detection implementation
- Debug display
- CoordinatorService implementation
- Integration testing
- Jetson Nano porting

Sandro Vrieling

- Design report
- Reflection plan
- Project proposal
- Testing the relationship between braking angle and deceleration
- GNSS implementation
- GNSS implementation unit testing
- Reflection report
- TimeToCrashPolicy implementation
- TimeToCrashPolicy unit testing
- Integration testing
- Poster

Julius Kuijf

- Design report
- Testing the relationship between braking angle and deceleration
- IMU implementation
- IMU implementation testing
- SimpleCrowdPolicy implementation
- UnsafeOvertakePolicy implementation
- Integration testing
- CSV display