Low-Speed Cargo Bicycle Balance

Design, Implementation, and Validation of an Active Kickstand Stabilization Mechanism for Low-Speed Cargo Bicycle Balance

B. de Vries

Delft University of Technology

Low-Speed Cargo Bicycle Balance

Design, Implementation, and Validation of an Active Kickstand Stabilization Mechanism for Low-Speed Cargo Bicycle Balance

by

B. de Vries

Supervisor:	J.K. Moore
Committee Member:	L. Marchal Crespo
Graduation date:	15 July 2025
Faculty:	Faculty of Mechanical Engineering, Delft

Cover: Photo taken and edited by Daniël Stiekema

TU Delft

Contents

Design, Implementation, and Validation of an Active Kickstand Stabilization Mechanism for Low-Speed Cargo Bicycle Balance

Bart de Vries, *Master of Mechanical Engineering, Biomechanical Design*, Delft University of Technology

***Abstract*—As cities increasingly transition to car-free and low-emission zones, cargo bicycles have become a popular alternative for both companies and families. They allow for the efficient transport of goods and children while navigating busy urban areas without emissions. However, single-track cargo bicycles still have some limitations. Stability issues, especially at low and zero speeds, and the effort required to lift a heavy cargo bicycle onto its kickstand, reduce usability and safety. These problems are amplified by uneven load distribution and heavy cargo.**

A prototype for an active kickstand stabilization mechanism was developed. A model was made where a simple spring-damper feedback system was placed on the roll degree of freedom of the cargo bicycle. After formulating design criteria, a design was created and manufactured, and the system was assembled on a cargo bicycle. The prototype was then tested to validate the design specifications.

The results show a measurable improvement in stability, especially when cycling off from a standstill. The system also stabilized the bicycle with a cargo of 50 kilograms on the luggage rack in addition to an 80-kilogram rider. These results show that an active kickstand stabilization mechanism is suitable for everyday use on cargo bicycles and can increase the safety and usability of the cargo bicycle.

***Index Terms*—Cargo bicycle, low-speed stability, bicycle dynamics, bicycle stability, mechatronic design**

I. Introduction

OVER the past decade, how we travel has changed a lot. Climate goals, personal and public health goals, and cities transitioning to low-emission and car-free zones have contributed to these changes. These car-free and low-emission zones push companies to find an alternative for their diesel and petrol vehicles. Not only are companies seeking alternatives to cars, but families are also reducing their car usage. [1].

Cargo bicycles have become a popular alternative due to their ability to transport packages, children, or other cargo while avoiding traffic and parking issues and complying with emission rules. The number of e-cargo bicycles sold in Europe has increased significantly in the past years, with a yearly revenue increase of over 200% between 2018 and 2024 and a projected revenue increase of 300% between 2024 and 2030 [2]. Germany shows an even greater increase in commercial cargo bicycle sales of 104% in 2022 compared to the previous year [3].

Despite their potential, conventional long-john cargo bicycles, which are two-wheeled single-track cargo bicycles with a cargo bay between the rider and the front wheel (Figure 1), also have several practical limitations. For regular bicycles, one study showed that 76.8% of accidents are single-sided accidents, and between 13.3% and 19.2% are while standing still or mounting or dismounting the bicycle [4]. While similar results are expected for cargo bicycles, there is limited data available on cargo bicycle accidents. However, it is reasonable to assume that the risks are even greater with a cargo bicycle. In one study, cyclists mentioned that the added weight of their e-bicycle was why they fell over while dismounting [5]. Not only are cargo bicycles heavier than e-bicycles, but they can also have an uneven load distribution, increasing the risk of falling over when coming to a stop further.

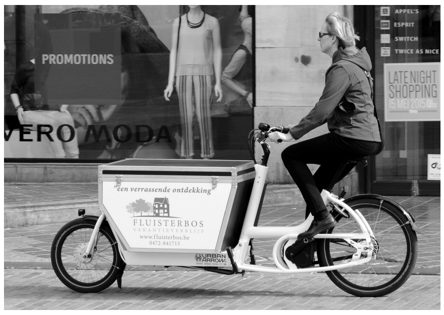


Fig. 1: Conventional single-track long-john cargo bicycle (image used under Pixabay license).

Additionally, lifting a cargo bicycle onto its kickstand can be difficult, especially when the bicycle is heavily loaded. This makes parking impractical for short stops and potentially unsafe, particularly on inclined or uneven surfaces.

These limitations ask for a solution that can help to increase the stability of a single-track cargo bicycle, especially at low speeds and when standing still, and to potentially replace or improve a traditional kickstand.

Previous research on low-speed stability for single-track vehicles shows multiple solutions. These can be categorized into five methods: control moment gyroscopes, reaction wheels, balancers, steering control & others.

The control moment gyroscope (CMG) was found to be the most studied method for stability. The CMG can be divided into separate control categories. Although the passive CMG works for stabilization, a CMG with an actively controlled gimbal delivers better performance [6]. The CMGs can also be divided by the quantity of gyroscopes and the orientation of the CMG: single horizontal, double horizontal, single vertical, and double vertical. When using a double CMG, the flywheels are combined as a ”scissored pair”, which means that both flywheels spin in a different direction, and the gimbals are also coupled and rotate in opposite directions. This ensures that the gyroscopic torque from both gimbals is in the same direction, while unwanted torques are canceled [7]. Although no study showed that the orientation of the wheels, either horizontal or vertical, changed the performance of the CMG, multiple studies showed that the scissored-pair double CMG is superior to a single CMG [8]–[10]. Wardle et al. [11] use a single horizontal CMG to help balance a rider on a stationary bicycle, but were unable to balance for more than 40 seconds.

Six studies used a reaction wheel with a prototype that could balance their single-track vehicle at zero speed. Two additional studies showcased a working model, one of which had no prototype and the other had a mechanical failure during testing [12], [13]. Notably, the reaction wheels add significant mass to the vehicles, enabling them to achieve balance.

Studies using balancers could also balance at zero speed. The balancer can be divided into different categories: laterally moving mass, pendulum, and inverted pendulum. Griese et al. [14] used a balancer, in addition to their CMG, to compensate for the center of gravity of their system not being in the same plane as the wheels, thus increasing the robustness.

Studies found on steering control show less promising results for stabilization at low and zero speed. These studies were on both two-wheel steering control and one-wheel steering control. Two studies could only balance their bicycle at zero speed using a balancer next to their steering control [15], [16]. Xiong et al. [17] could balance at zero speed by rotating the front wheel 90 degrees, but could not stabilize their vehicle while driving at low speeds. Yang et al. [18], [19] published two studies on the same bicycle, comparing different modes of steering control and accomplishing zero-speed stabilization.

Five studies show solutions that do not fall within the categories mentioned above. Three of these studies show an experimental motorcycle developed by Yamaha, which rotates itself around a swivel axis to achieve stabilization at zero and low speeds [20]–[22]. Honda also showcases an experimental motorcycle that can laterally move its rear wheel and adjust the steering axis angle to achieve low-speed and zero-speed stabilization [23]. Finally, Huang et al. [24] proposed a bicycle with legs to help stabilize on rough terrain.

While the literature research revealed multiple methods that could sustain stability at low and zero speeds, many of these approaches lack convincing evidence of robustness in real-life situations, where uneven surfaces, moving or removing cargo, and variable rider inputs can significantly affect balance. Although the concept of using legs has received little attention in existing research, it offers a promising combination of mechanical simplicity, energy efficiency, and robustness. Unlike systems found in the literature that require continuous sensing and actuation, a leg-based approach can provide reliable passive support at a standstill and controlled assistance during low-speed maneuvers. For these reasons, the concept of an active kickstand mechanism was selected for further development. The research goal of this study is therefore:

“To design, implement, and validate a novel active kickstand stabilization mechanism for a single-track cargo bicycle, which is aimed at low-speed stability.”

II. Modeling and Simulation

A bicycle has two degrees of freedom, roll and steer. Balancing a bicycle can be done by steering or by applying a roll torque. To approximate the roll torques required for stabilization, the stability of the cargo bicycle was simulated using the method and equations by Meijaard et al. [25]. The Carvallo-Whipple model, used by Meijaard et al., uses the equations of motion of the bicycle, which are linearized about the upright position at a constant speed (Equation 1).

$$M\ddot{q} + vC_1\dot{q} + \left(gK_0 + v^2K_2\right)q = f \tag{1}$$

With generalized coordinates q and external forces f:

$$q = \begin{bmatrix}\phi \\ \delta\end{bmatrix},\ f = \begin{bmatrix}T_\phi \\ T_\delta\end{bmatrix}$$

Where ϕ and δ are the roll and steering angle of the bicycle, respectively, and where T_ϕ and T_δ are the roll and steering torques of the bicycle. M, C_1, K_0, and K_2 are the mass, damping, and two stiffness matrices, respectively, which are defined by the bicycle's parameters. The gravitational acceleration is g and the bicycle's speed is v.

The parameters (Appendix A) of the cargo bicycle without a rider, which were determined in previous research, were used to generate the mass, damping, and stiffness matrices of the cargo bicycle. Inertia data for a rider was generated and added to the cargo bicycle parameters using the parallel axis theorem [26]. Consequently, the stable speed regions of the cargo bicycle with rider could be determined with no external forces acting on the system, so $f = [0\ 0]^T$. The linearized equations of motion were converted to state-space form, and the eigenvalues of the state matrix were calculated. Stability is indicated when all real parts of the eigenvalues are negative, corresponding to the exponential decay of the system's modes. The eigenvalues were calculated for a range of speeds and plotted to visualize the bicycle's stability. As shown in Figure 2, the cargo bicycle showed no self-stability at low speeds, while it is self-stable between 4.38 and 6.3 m/s.

A simple spring-damper feedback system was added to the system as a simplified way to express the legs on the cargo bicycle. The feedback system generated an active torque between the ground and the bicycle, tending to move it toward the upright position. A small spring and damper on the steering angle were also required to stop the handlebars from getting to unstable angles. The torques were added to the equations of motion of the system as shown in Appendix B.

The coefficients were as follows:

$$\begin{aligned} k_{roll} &= 1030\ N/rad, \\ c_{roll} &= 20\ Ns/rad, \\ k_{steer} &= 7\ N/rad, \\ c_{steer} &= 1\ Ns/rad \end{aligned}$$

With this feedback system, the cargo bicycle with rider model could be made self-stable at all speeds, but most importantly, it was stable at zero and low speeds (Figure 3).

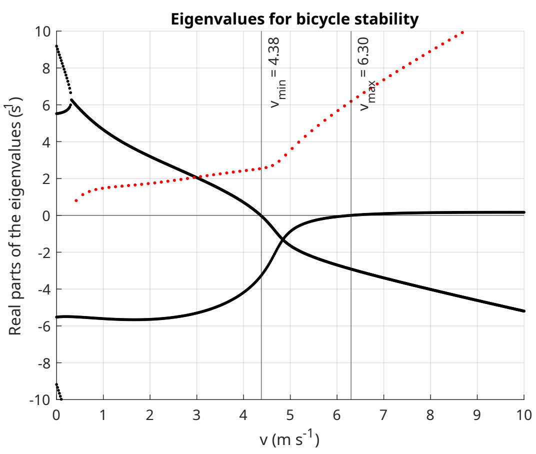


Fig. 2: Eigenvalues of the cargo bicycle with rider over speed. The real parts of the eigenvalues are in black. The positive imaginary parts of the eigenvalues are in red. The system shows no stability at low speeds.

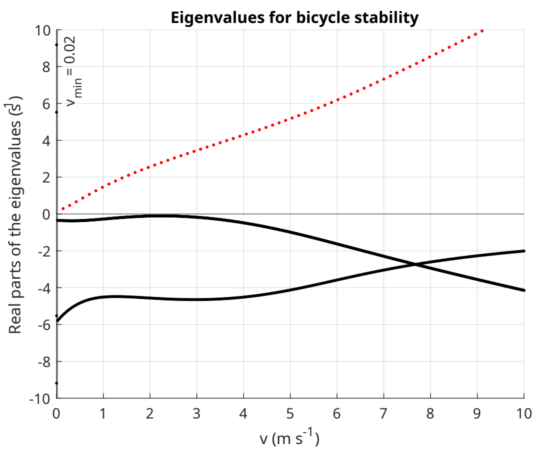


Fig. 3: Eigenvalues of the cargo bicycle with rider and spring-damper feedback system. The real parts of the eigenvalues are in black. The positive imaginary parts of the eigenvalues are in red. The system shows stability at all speeds.

A simulation was performed to evaluate the linearized roll and steer dynamics of the cargo bicycle model at a constant forward speed. The system's equations of motion were integrated over time using initial values for the angles and rates, yielding the roll angle, steering angle, and their respective rates over the simulation period to assess stability behavior. The initial values and simulation parameters were as follows:

$$v = 1.0\ m/s,$$
$$t = 10\ s,$$
$$x_0 = \begin{bmatrix} \phi_0 \\ \delta_0 \\ \dot{\phi}_0 \\ \dot{\delta}_0 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 0 \\ 0 \end{bmatrix}$$

With roll angle (ϕ) and steering angle (δ) in degrees, where ϕ_0 and δ_0 are the initial angles, and $\dot{\phi}_0$ and $\dot{\delta}_0$ are the initial angular rates. The bicycle speed in the simulation is v, and the simulation time is t.

From the simulation, the roll torque can be calculated with:

$$T_{roll} = -k_{roll}\phi - c_{roll}\dot{\phi} \tag{2}$$

The results of the simulation are shown in Figure 4. An absolute maximum roll torque of 180 Nm is required to maintain stability.

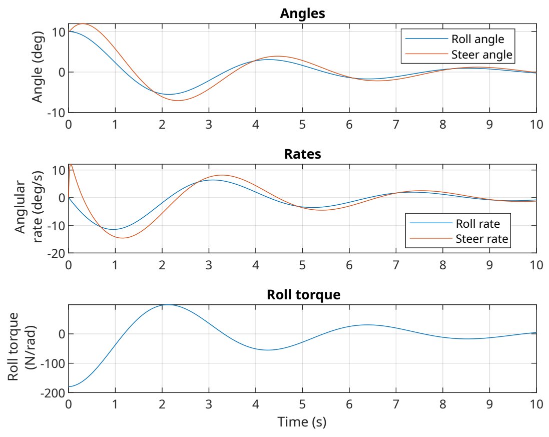


Fig. 4: Plot of the simulation of the cargo bicycle with an 80-kilogram rider and spring-damper feedback system. The initial roll and steering angles are both 10 degrees. The simulation shows stable behavior.

Additionally, another set of parameters was generated by adding an 80-kilogram point mass cargo to the model (Table VII, Appendix A). For these parameters, new stiffness and damping coefficients were determined, which made the bicycle model stable at all speeds.

$$k_{roll} = 1620\ N/rad,$$
$$c_{roll} = 90\ Ns/rad,$$
$$k_{steer} = 7\ N/rad,$$
$$c_{steer} = 1\ Ns/rad$$

With the stable model of the cargo bicycle including rider and cargo, another simulation was run to determine the torques required for stabilization. The simulation parameters were as follows:

$$v = 1.0\ m/s,$$
$$t = 10\ s,$$
$$x_0 = \begin{bmatrix} \phi_0 \\ \delta_0 \\ \dot{\phi}_0 \\ \dot{\delta}_0 \end{bmatrix} = \begin{bmatrix} 7.5 \\ 10 \\ 0 \\ 0 \end{bmatrix}$$

This resulted in an absolute maximum roll torque of 215 Nm required to keep the bicycle stable.

III. Design Process

In addition to the constraint that the system would stabilize the cargo bicycle using an active kickstand mechanism, a set of design requirements was formulated to further guide the design process. These requirements are listed in Table I.

TABLE I: The design requirements formulated for the new solution.

#	Need
1	Activates when the bicycle goes below a certain speed
2	Stabilizes the bicycle when at low speed and when stationary
3	When the system is active, the bicycle cannot fall over due to disturbances
4	Does not add too much weight to the system
5	When the system is active, the motion of the bicycle feels the same to the rider
6	System can be turned on and off
7	The power consumption of the system is only up to 20% of average e-cargo bicycle power consumption
8	The bicycle can take corners at normal speed when the legs are retracted
9	The bicycle stabilizes to a vertical position, even on sloped or uneven ground

From these requirements, and by using the results from the simulations in Section II, metrics were derived to assess the design after manufacturing and implementation (Table II).

A morphology-based design process was employed to create multiple concept solutions, which were evaluated, and the best solution was chosen based on weighted criteria. Appendix C shows the full design process.

As described in Section II, a spring-damper feedback model was used in the simulation to analyze the roll dynamics and estimate the required torque, stiffness, and damping to stabilize the cargo bicycle at low speeds. While this helped to determine these values, the physical design ultimately used a rigid mechanism. A compliant solution, such as concept B.2 using a ball screw, was considered but dismissed due to significantly higher cost. Additionally, the simulation showed that the required stiffness increases with rider weight and cargo load, which would have required the system's stiffness to be adjustable, adding further complexity. For these reasons, a rigid design was selected as a simpler and cost-effective alternative.

IV. Final Design and Implementation

Concept B.1 was further developed into a design that could be manufactured. For this project, a Workcycles Kr8 cargo bicycle (Workcycles, Amsterdam, The Netherlands) was used, which was taken into account when making the final design for the prototype. The final design was manufactured and assembled on the cargo bicycle. Since the system was mechatronic, it consisted of three parts. The **mechanical part** converts the motor's rotational output to the movement of the legs, which stabilizes the cargo bicycle. The **electronic part** powers the system and does the sensing. Finally, the **control part** processes the user inputs and sensor data, and through logic and a state machine, outputs signals to control the motors.

A render of the final design for the mechanism can be seen in Figure 5.

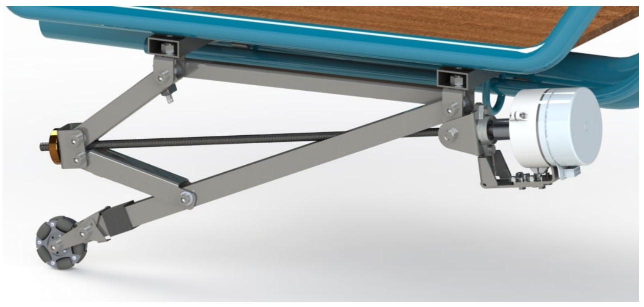


Fig. 5: Solidworks render of the final design of the mechanism.

A. Functional overview

The final design, based on concept B.1, consists of two separately actuated legs mounted on either side of the underside of the cargo bicycle. The legs are composed of a system of linkages and a trapezoidal spindle, similar to a scissor jack, which allows them to retract upwards and extend downwards. On the ends of the legs, wheels are placed to allow the cargo bicycle to keep moving forward, while the legs are down and touching the ground. The wheels are omni-wheels, which can also roll sideways, enabling the cargo bicycle to take turns while the wheels are on the ground. Brushless DC motors drive the spindles. To increase the torque from the motors, a custom planetary gearbox is used.

The system automatically lowers the legs when the bicycle's speed drops below a certain threshold, and raises them again once the bicycle speeds up. When walking slowly, the wheels slightly lift off the ground to allow for greater maneuverability. Limit switches are used to sense when the legs come in contact with the ground and when the legs are fully retracted. To sense the speed of the cargo bicycle, the output voltage of the front-wheel dynamo is used. The system can be turned on or off with a switch below the handlebars. When the system is turned off, the legs always retract and remain retracted, regardless of the speed of the cargo bicycle. An emergency stop switch is located next to the on/off switch, which shuts down the system and brakes the motors when pressed. The accelerometer on a 9-degree-of-freedom IMU is used to measure the lateral (X) and vertical (Z) accelerations, from which the roll angle of the bicycle is calculated.

An Arduino Uno R4 WiFi processes all signals. The Arduino runs the control logic and a PID controller that outputs direction and speed commands to the motor controllers.

The functioning of the system is visualized in Figures 6, 7, and 8.

B. Mechanical overview

As mentioned before, the system uses linkages and a spindle to move the legs. The motors, which are Maxon 500267 brushless DC motors (Maxon, Sachseln, Switzerland), have their torque increased by a custom 3D-printed PLA planetary gearbox with a reduction ratio of 1:6.7. The nominal torque

TABLE II: The design metrics formulated for the design process. The metrics refer to the design requirements to which they are linked.

#	Need #	Metric	Value	Unit
1	1,2,5	Speed sensor frequency	≥ 50	Hz
2	1,2,5	Speed sensor resolution	≤ 0.5	m/s
3	1,2,3	Activation time	≤ 1	s
4	2,3	Rider of 80 kg and cargo of 80 kg can be stabilized at 7.5 degrees	215	Nm
5	2,3	Rider of 80 kg can be stabilized at 10 degrees	180	Nm
6	4	Total weight of the system (without battery)	≤ 15	kg
7	5	Time increase to do 90-degree turn (compared with system off)	≤ 10	%
8	5	Time increase to do 180-degree turn (compared with system off)	≤ 10	%
9	7	Power consumption	≤ 2.7	Wh/km
10	8	Roll angle with legs up	≥ 10	deg
11	9	Maximum slope at which bicycle is vertical	10	deg

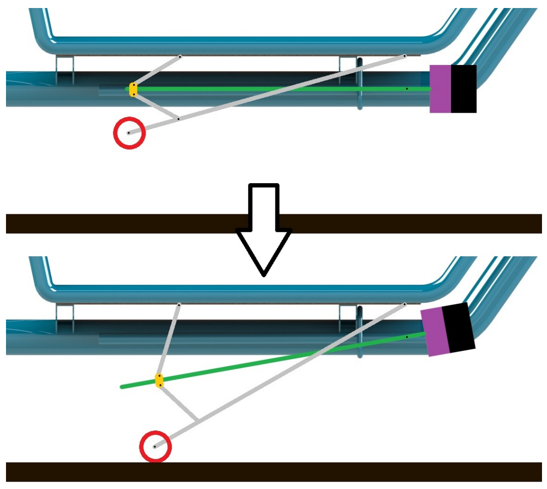


Fig. 6: Side view of the system activating. When the speed goes below the threshold, the legs lower. The motors turn the trapezoidal spindles, which pull the linkages toward the motor and extend the legs. When the omni-wheels touch the ground, the limit switches are activated, and a signal is sent to the Arduino to stop lowering the legs. The components from right to left are: **Black**: motor, purple: gearbox, green: spindle, gold: spindle nut, red: omni-wheel. The black dots are the hinges.

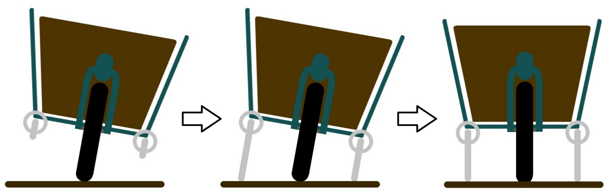


Fig. 7: Frontal view of the legs lowering when the cargo bicycle is leaning to the side. The legs touch the ground, stabilizing the cargo bicycle in a vertical position.

of 0.964 Nm of the motors is increased to 6.46 Nm. A 10-millimeter steel trapezoidal spindle with a 2-millimeter pitch converts this torque to a linear force. While a 16-millimeter spindle would have been more suitable for this project, a 10-millimeter spindle was chosen to limit the costs of the prototype. The spindle exerts a pull force of approximately 6.43 kN on the bronze spindle nut. To restrict the axial

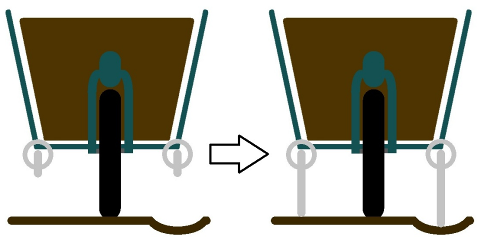


Fig. 8: Frontal view of the legs lowering when the cargo bicycle is on an uneven surface. The legs lower until they touch the ground and keep the cargo bicycle vertical.

movement of the spindle, a flange bearing was placed near the hinge of the motor, and a steel trapezoidal nut was placed against it.

The spindle crosses the long linkage to which the wheel is attached (the leg), as can be seen in Figures 5 & 6, for two reasons. This enables a lower placement of the motor, which ensures the motor does not collide with the frame of the cargo bicycle when the legs are fully extended. Additionally, the spindle and hinge of the leg do not go through the same point, simplifying the design of the hinge.

For this prototype, all linkages and hinges are made from 304 stainless steel, since this was the strongest material readily available. Except for the leg and the upper linkage, which were made from square tubes, all parts are made of sheet metal, which was laser-cut and bent with a press brake.

A detailed overview of the assembled system can be seen in Figure 9.

C. Electronic overview

The prototype is powered by a 48 V 13 Ah electric bicycle battery. The 48 V is directed through two Maxon DRS 70/30 shunt regulators, one for each motor, to the Maxon ESCON 70/10 motor controllers (Maxon, Sachseln, Switzerland). An XL7015 DC-DC converter converts the 48 V to 12 V to supply power to the Arduino Uno R4 WiFi. The Arduino provides the power to all sensors and switches.

Two limit switches are used to sense when the legs are fully retracted. Another set of limit switches is used to sense when the omni-wheels are in contact with the ground. The wheels are placed on hinges, and when the wheels touch the

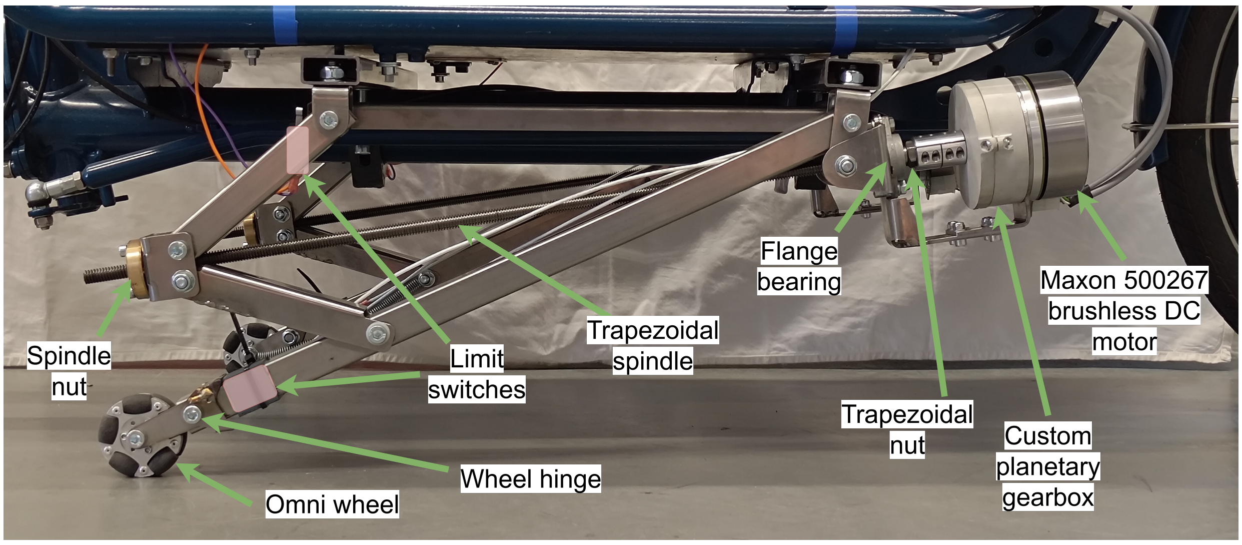


Fig. 9: Prototype of the final design. Important mechanical parts are highlighted. The limit switches are on the rear side of the linkages, and therefore their locations are indicated with orange boxes.

ground, the hinges rotate slightly, pressing the limit switches. All switches, including the emergency stop switch and the on/off switch, are in a custom circuit with a pull-down resistor for each switch.

The system uses the Shimano DH-3D32-NT dynamo in the front-wheel hub of the cargo bicycle as a speed sensor. A diode and RC filter rectify the AC output, and a 4.7 V Zener diode limits the voltage. An analog pin on the Arduino then measures the resulting signal.

An overview of the electronic components can be seen in Figure 19 in Appendix D. A schematic overview of the wiring of all components is shown in Figure 21 in Appendix E.

D. Control overview

The Arduino processes all sensor and user inputs to determine the output to the motors. This is done with control logic and by using a state machine. The states are:

1) Deactivated

The system is switched off. The system has power, but the legs are retracted and will not engage when the speed gets below the threshold.

2) Activated

The system is switched on. The system will engage when the speed gets below the threshold.

3) Engaging

The speed went below the threshold while activated, so the legs will move down.

4) Disengaging

The speed went above the threshold while activated, or the system was deactivated. The legs will retract.

5) Walking

The system is activated, and the speed is below the threshold, but you are slowly moving. The legs will slightly move up to allow more maneuverability of the cargo bicycle while walking.

6) Controlling

Both legs are down, and the speed is below the threshold to go to the walking state. The system will ensure the cargo bicycle is vertical.

7) Emergency

The emergency switch is activated. The motors brake and the Arduino cannot switch state until the switch is released.

The Arduino code can be seen in Appendix F. A state diagram, which shows the working of the state machine, can be seen in Figure 23 in Appendix G.

While in the controlling state, the Arduino controls the angle of the bicycle by using a PID controller that uses the roll angle, which is calculated from the filtered X and Z accelerations, as input. A deadband of ± 1 degrees was used to prevent oscillatory behavior at small angles.

A block diagram showing how the system operates is shown in Figure 24 in Appendix H.

E. Signal processing

Although an RC filter is used to convert the AC output of the speed sensor to a DC signal, residual sinusoidal fluctuations remain. To further smooth the signal, a digital exponential moving average filter is applied. This filter uses a smoothing factor of $\alpha = 0.05$.

A digital exponential moving average filter also filters the IMU data. The smoothing factor α is calculated using Equation 3 to approximate a cutoff frequency of $f_c = 1$ Hz, which filters out all vibrations from the road, while the rider input and bicycle roll remain.

$$\alpha = \frac{2\pi f_c}{2\pi f_c + f_s} = \frac{2\pi}{2\pi + 200} = 0.0305 \tag{3}$$

V. Testing and Validation

To validate the system and to ensure the system satisfies the design requirements as formulated in Section III, multiple tests were formulated. This section elaborates upon those tests and shows the results of each test.

TABLE III: Tests done to validate the system, showing the metric that they are intended to measure.

Metric #	Test
1,2	Speed sensor test
3	Activation time
4,5	Torque test
6	Mass measurement
7,8	Maneuverability test
9	Power consumption
10	Roll angle and inclination test
-	Stability test

A. Speed sensor test

To determine the resolution of the speed sensor, which the system requires to accurately set the activation threshold, the cargo bicycle was placed on a treadmill. The speed was gradually increased from 0 to 18 km/h in steps of 1 km/h. The output signal from the dynamo was measured as an analog input to the Arduino. Since the analog input of the Arduino gives a value between 0 and 1023, the resolution of the speed sensor can be determined. Considering the Zener diode limits the input voltage to the Arduino to 4.7 V, the maximum value the Arduino can read is theoretically 962.

The frequency of the speed sensor is determined by the Arduino by setting the sampling frequency in the code. The frequency needs to be sufficient to avoid introducing a delay between the speed decreasing below the threshold and the system measuring that the speed decreased below the threshold.

The results from the speed sensor test can be seen in Figure 10. The measurements are linear between approximately 2 km/h and 8 km/h. This shows the speed sensor has a resolution of 0.0125 km/h/div, or 0.003 m/s/div, in the linear part of the measurement range.

Since the Arduino runs the code at 200 Hz, the sampling frequency of the speed sensor is also 200 Hz.

B. Activation time

The time between turning the system on and the legs touching the ground is measured. This was done multiple times, and an average was taken.

Table IV shows that the average activation time for the system is 2.16 seconds.

C. Torque test

The cargo bicycle was placed at a roll angle of 10 degrees with the system turned off and both legs down and touching the ground. An 80-kilogram rider was seated on the bicycle,

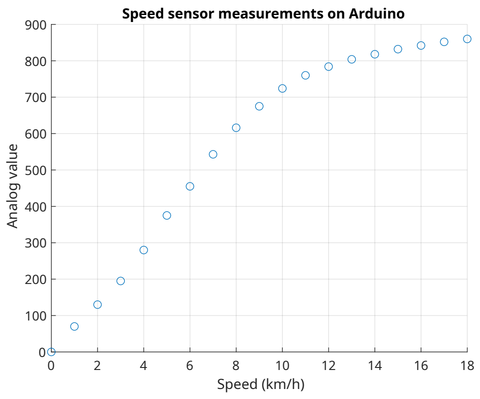


Fig. 10: Values measured by the analog input of the Arduino over speeds.

TABLE IV: Measured activation times.

Measurement	Time (s)
1	2.32
2	2.06
3	2.16
4	2.14
5	2.11
Average	**2.16**

and the system was activated to record if the system would have enough roll torque to stabilize the bicycle back to vertical.

The same was repeated with an additional weight at an initial roll angle of 7.5 degrees. Since the cargo bay of the bicycle was being used for the electronics of the system, a weight of 50 kilograms was placed on the luggage rack (Figure 11). Since the luggage rack of the bicycle was higher than the cargo bay, the longer moment arm resulted in approximately the same roll torque as when placing 80 kilograms in the cargo bay.

The system was only tested at the maximum weights as defined in the design requirements as not to damage the prototype for further testing. The system was able to stabilize both the rider at 10 degrees roll and the rider plus cargo at 7.5 degrees roll.

D. Mass measurement

The mass of the system, excluding the cargo bicycle, was determined using the design in Solidworks and the documentation of all the electronics. Additionally, the mass of the system, including the cargo bicycle, was also determined by placing the cargo bicycle on scales.

The weight of the system was determined to be approximately 11.5 kilograms using SolidWorks and part documentation. The weight of the cargo bicycle, including the system, was measured at approximately 58 kilograms.

E. Maneuverability test

To test the maneuverability, 90-degree and 180-degree turns were performed at walking speed. The 180-degree test was

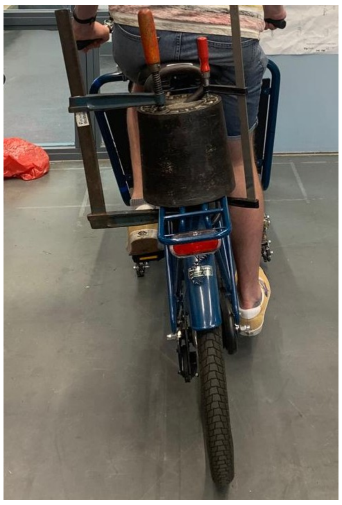


Fig. 11: Torque test with an 80-kilogram rider and 50 kilograms of weight on the luggage rack. The initial roll angle of the bicycle is 7.5 degrees.

done by performing a three-point turn (Figure 12). Multiple different ways of maneuvering the bicycle were tested. For both the 90-degree and 180-degree tests, each maneuver was repeated five times. The maneuvers were:

- Turning the bicycle to the **left** while standing next to the bicycle and **walking** along
- Turning the bicycle to the **right** while standing next to the bicycle and **walking** along
- Turning the bicycle to the **left** while **sitting** on the saddle and scooting the bicycle forwards
- Turning the bicycle to the **right** while **sitting** on the saddle and scooting the bicycle forwards

All these maneuvers were performed with the system turned on and with the system turned off. Each maneuver was timed by the rider to determine how much the time to take the maneuver would increase when the system was turned on and the legs were on the ground.

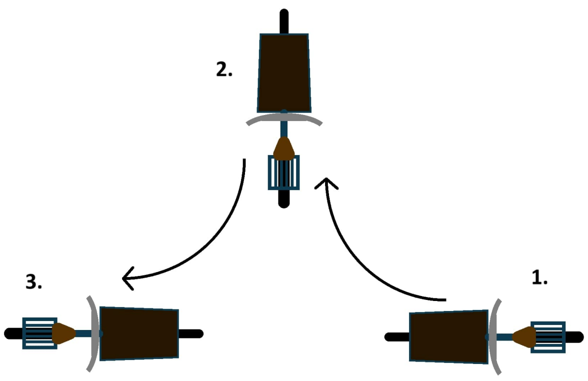


Fig. 12: The three-point turn maneuver. The cyclist is not in the figure, but can be either sitting on the bicycle or walking next to the bicycle. In this figure, the maneuver is performed to the right.

The results of the maneuverability test can be seen in Table V and Table VI, with more detailed versions tables in Appendix I.

The results show an average increase in maneuvering time of 9.28% for the 90-degree turn and a 7.13% increase for the 180-degree three-point turn.

TABLE V: Results of the maneuverability test for the 90-degree turn. Each cell shows the time in seconds for each maneuver. The table is color-scaled, where red cells are slower than average and green cells are faster than average.

Standing				Sitting			
System off		System on		System off		System on	
Right	Left	Right	Left	Right	Left	Right	Left
4.60	4.77	4.60	4.97	4.76	4.96	5.53	4.86
4.20	4.63	4.94	4.62	4.70	4.81	5.04	5.41
4.03	4.61	4.51	4.75	4.26	4.91	5.28	5.29
4.10	4.67	4.58	4.95	4.79	4.80	5.19	5.57
4.70	4.64	5.05	4.85	4.32	4.83	5.17	5.54

TABLE VI: Results of the maneuverability test for the 180-degree three-point-turn. Each cell shows the time in seconds for each maneuver. The table is color-scaled, where red cells are slower than average and green cells are faster than average.

Standing				Sitting			
System off		System on		System off		System on	
Right	Left	Right	Left	Right	Left	Right	Left
8.66	8.48	9.43	9.65	10.16	9.46	10.78	10.63
8.39	8.90	8.71	9.57	10.03	10.30	11.09	11.59
8.70	8.65	9.09	9.41	9.73	9.00	10.32	9.03
8.51	9.40	9.26	9.48	10.40	10.22	11.53	11.24
8.34	8.33	9.14	9.15	10.22	9.36	9.98	9.31

F. Power consumption

To determine the system's power consumption, a test ride was performed. A watt meter (EXTRON Modellbau, Eggenfelden, Germany) was added between the battery and the system to measure the power consumption during the test ride. A route of 6.4 kilometers was planned through the city of Delft in the Netherlands. The route consisted of multiple busy intersections, traffic lights, and bridges to simulate everyday usage. The test ride was done by an 80-kilogram rider, without any additional cargo, at an outside temperature of 25 degrees Celsius.

During the 30-minute ride, a total power consumption of 15.5 Wh was recorded. This results in a system power consumption of 2.42 Wh/km, or an average power of 31.0 W, for urban areas. During the ride, a total of 13 stops in which the system was activated were recorded. This results in an average power consumption of 1.2 Wh per activation.

G. Roll angle & inclination test

First, the system was deactivated, and with the legs fully retracted, the cargo bicycle was rolled over until the wheel on the leg hit the ground. The roll angle of the bicycle was measured at the farthest point. The same was done while rolling the bicycle to the other side. The measured angles indicate how much roll the bicycle can have while cornering when the legs are fully retracted.

Additionally, the cargo bicycle was placed on a wooden board, which was jacked up on one side to increase the angle (Figure 13). A rider weighing approximately 100 kilograms was seated on the bicycle, and the legs were put down with the system turned off, to resemble a normal kickstand. The angle was slowly increased until the bicycle would fall over to the side. This was repeated with the system turned on.

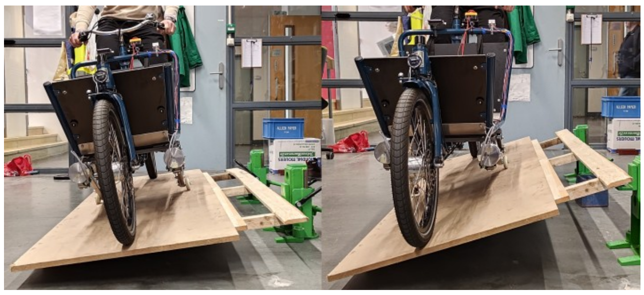


Fig. 13: Incline test with the system turned off on the left and the system turned on on the right.

The maximum roll angle of the cargo bicycle with the legs fully retracted was measured to be 24.7 degrees to the right and 23.5 degrees to the left.

The inclination test showed that the cargo bicycle would fall over at an angle of 7.7 degrees when the system was turned off. When the system was turned on, the cargo bicycle would remain perfectly level up to an angle of 15.5 degrees. At 23.2 degrees, the bicycle would fall over with the system turned on.

H. Stability test

To test whether the system would increase the stability of the cargo bicycle during take-off and coming to a stop, a test was conducted in which the system was activated and the linear accelerations were measured while cycling off from a standstill and then coming to a stop again. A total of 10 take-offs and stops were done with the system turned on, and 10 more with the system turned off. The stability was measured using the accelerometer on a Nokia 3.4 smartphone, which was mounted right below the handlebars of the cargo bicycle (Figure 20, Appendix D). The data was recorded with the phyphox app from RWTH Aachen University.

The data from the stability test were filtered to isolate the accelerations that are due to the human input and natural movement of the bicycle's roll from vibrations and other noise. This was done using a second-order low-pass Butterworth filter, chosen for its good balance between signal preservation and noise reduction, with a cutoff frequency of $f_c = 1$ Hz. The filtered X and Z accelerations were used to calculate the roll angle of the cargo bicycle. The trials are cut into pieces of 3 seconds for taking off, and 2 seconds for stopping, to ensure only the part of the trial where the system influences the stability of the cargo bicycle is considered.

The filtered roll angle of the bicycle is plotted in Figure 14 to visualize the stability during taking off. The root mean square of all trials yields 1.2493 degrees for the system when turned off and 1.0251 degrees for the system when turned on. To assess statistical significance, a t-test was performed, yielding $p = 0.0902$, which indicates that the observed increase in stability was not statistically significant.

The results of the stability during stopping showed nearly identical results of the root mean square of the roll angles; 1.2479 degrees when the system was turned off and 1.2802 degrees when the system was turned on.

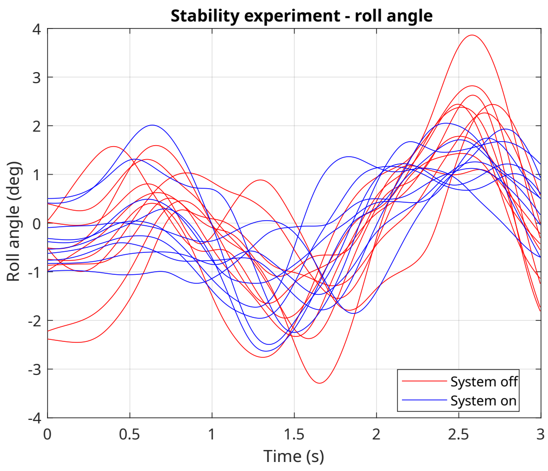


Fig. 14: Roll angle during takeoff during the stability test. Each line is a different trial.

VI. Discussion

In this section, the results from the tests, as described in Section V, are evaluated. Observations related to user experience are included, and potential improvements to the system are identified.

A. User experience

Although not part of the validation process, several test rides showed that the system "felt" natural during use. It did not interfere with cycling away from a standstill, even when turning immediately after takeoff. When coming to a slow stop, the legs were often able to fully extend and touch the ground before the rider needed to place a foot down.

B. Performance

The speed sensor, based on the voltage output from the front-wheel dynamo, provided sufficient resolution to reliably detect low speeds. The sampling frequency was high enough to allow triggering of the system when the speed dropped below the threshold on time.

Although the activation time of the system was more than twice the desired value, it was improved by increasing the speed threshold at which the legs deploy. This adjustment ensured the legs were nearly in contact with the ground by the time the rider would come to a stop, effectively compensating for the slower deployment mechanism.

The relatively slow actuation time was a result of prioritizing torque over speed during the design process. However, internal friction limited the motors to a speed of approximately 1850 RPM, which was lower than the manufacturer's no-load specification of 1960 RPM [27]. Changing to a motor with a higher speed or reducing the gear ratio could significantly improve deployment time but at the cost of torque. Alternatively, a more advanced design could incorporate two separate stages, one motor for rapid extension to the ground and another for stabilization. Another option is a gearbox capable of shifting between high-speed and high-torque modes during deployment.

The torque performance was satisfactory. Although the torque test was only performed at 7.5 degrees and 10 degrees with a set weight, the results indicated that the system could withstand more roll torque than was used during testing. Additionally, the motor current was limited to 3.5 A during tests, while the rated nominal current is 4.06 A. Increasing the current limit would further improve stabilization torque.

C. *Weight and power consumption*

The total weight of the system, excluding the battery, remained below the design requirement of 15 kg. This assumes that the system is being used on an electric cargo bicycle. However, if the system is installed on a non-electric cargo bicycle, an additional battery would be required. To obtain a more accurate total weight, the system should be disassembled and weighed directly on a scale.

The power consumption test showed an energy use of 2.42 Wh/km during a 25-minute urban test ride. This is well below the typical energy consumption range of e-cargo bicycles (9–18 Wh/km) [28], meaning the system contributes an additional 14% to 27% in energy consumption. This is within acceptable limits for integration into existing electric cargo bicycles. For parcel delivery use, which involves frequent stops, the system would consume approximately 30 Wh for 25 deliveries. Assuming a trip length of up to 1.5 hours, the additional power used during riding would be 46.5 Wh, resulting in a total consumption of 76.5 Wh per trip. While this is within the system's design specifications, it was on the higher end of what was initially expected.

D. *Low-speed behavior*

The maneuverability test indicated that the system slightly increased the time required to perform walking maneuvers, although the increase was relatively small and within design specifications. Potential bias may have been introduced by the position of the person (always on the left side of the bicycle) and the phone timer (on the right side of the handlebars). Additionally, the test was performed on an uneven, tiled parking lot instead of smooth tarmac. Maneuvers performed downhill were faster than those performed uphill.

E. *Stability*

The incline test demonstrated that the cargo bicycle remained stable at higher roll angles when the system was activated. The difference between the measured 23.5 degrees maximum roll angle and the 15.5 degrees angle at which the system remained level is likely due to early activation of the limit switches under lateral loading, which occurs at greater inclinations. The roll angle test confirmed that the bicycle had sufficient lean clearance to safely take corners at normal cycling speeds.

During the stability test, the system showed, while not significant, benefits during takeoff. It allowed the rider to begin cycling from a stable, upright position, without the need to balance manually. However, no improvement in stability was observed during stopping. This may be due to the way the rider uses the system in practice: when stopping, riders tend to put their foot down before the legs have fully extended. As a result, the system cannot actively assist in stopping, but it can passively help prevent tipping once stationary.

VII. Conclusion

In this paper, a novel active stabilization system was developed to increase low-speed stability for single-track cargo bicycles. The system consists of two deployable support legs that extend when the bicycle slows down, offering additional support during stopping, starting, or walking with the bicycle. A PID controller regulates the actuation based on the filtered roll angle, and the complete system was implemented, tested, and validated on a cargo bicycle.

Test results showed that the system improved stability in multiple real-life scenarios. It increased the maximum stable inclination at which the cargo bicycle would tip over by 15.5 degrees, provided greater confidence during takeoff, and did not interfere with maneuverability or normal riding. Test rides showed that the bicycle retained a natural and intuitive feel during operation with the system active. Compared to more complex balancing methods found in literature, which often involve continuous sensing and actuation, or complex steering adjustments, this solution is mechanically simpler, energy-efficient, and better suited to urban use cases with frequent stopping and starting, such as parcel delivery or transporting children.

While some design requirements, such as activation time, were not fully met, practical workarounds proved effective during testing. Slight improvements, such as using faster motors or a two-stage actuation mechanism, could further increase the system's performance. The results show that this stabilization approach is not only feasible but also offers a robust and practical solution to a real problem, making it a good addition to future cargo bicycle designs.

Acknowledgments

As mentioned earlier in the paper, the code for the communications between the Arduino and the IMU was generated by an artificial intelligence tool, Claude.ai. Additionally, spelling and grammar were checked using artificial intelligence tools Grammarly and ChatGPT.

References

[1] W. Riggs, "Cargo bikes as a growth area for bicycle vs. auto trips: Exploring the potential for mode substitution behavior," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 43, pp. 48–55, Nov. 2016.

[2] Horizon Grand View Research, "Europe Electric Cargo Bikes Market Size & Outlook, 2030," https://www.grandviewresearch.com/horizon/outlook/electric-cargo-bikes-market/europe.

[3] Zweirad Industrie Verband, "2022 Market Data – Bicycles and E-Bikes," Zweirad Industrie Verband, Tech. Rep., 2022.

[4] E. M. J. Verstappen, D. T. Vy, H. M. Janzing, L. Janssen, R. Vos, M. G. J. Versteegen, and D. G. Barten, "Bicycle-related injuries in the emergency department: A comparison between E-bikes and conventional bicycles: A prospective observational study," *European Journal of Trauma and Emergency Surgery*, vol. 47, no. 6, pp. 1853–1860, Dec. 2021.

[5] S. Haustein and M. Møller, "E-bike safety: Individual-level factors and incident characteristics," *Journal of Transport & Health*, vol. 3, no. 3, pp. 386–394, Sep. 2016.

[6] R. Lot and J. Fleming, "Gyroscopic stabilisers for powered two-wheeled vehicles," *Vehicle System Dynamics*, vol. 57, no. 9, pp. 1381–1406, Sep. 2019.

[7] D. Brown and M. A. Peck, "Scissored-Pair Control-Moment Gyros: A Mechanical Constraint Saves Power," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 6, pp. 1823–1826, Nov. 2008.

[8] R. Kusumardana, E. Pitowarno, A. Darmawan, and E. Kusumawati, "A Propose of PID Stability Control in A Gyro-Disc Actuator System," in *2019 International Electronics Symposium (IES)*, 0027/2019-09-28, pp. 341–349.

[9] S.-H. Park and S.-Y. Yi, "Active Balancing Control for Unmanned Bicycle Using Scissored-pair Control Moment Gyroscope," *International Journal of Control, Automation and Systems*, vol. 18, no. 1, pp. 217–224, Jan. 2020.

[10] S. Spry and A. Girard, "Gyroscopic Stabilization of Unstable Vehicles: Configurations, dynamics and control," *Vehicle System Dynamics*, vol. 46, May 2008.

[11] D. Wardle, T. Gregory, and B. Cazzolato, "Electronic training wheels: An automated cycling track stand," in *Proceedings of Australasian Conference on Robotics and Automation*, 2014.

[12] N. Vu, T. Nguyen, H. Dao, P. Nguyen, and H. Nguyen, "Robust Optimal Controller for Two-wheel Self-Balancing Vehicles Using Particle Swarm Optimization," *International Journal of Mechanical Engineering and Robotics Research*, vol. 12, no. 1, pp. 16–22, 2023.

[13] J. K. Moore, J. K. Cherian, B. Andersson, O. Lee, and A. Ranheim, "Modeling and Implementation of a Reaction Wheel Stabilization System for Low Speed Balance of a Cargo Bicycle," in *The Evolving Scholar - BMD 2023, 5th Edition*, Mar. 2023.

[14] M. Griese, F. Kottmeier, and T. Schulte, "Modeling the Vertical Dynamics of a Self-stabilizing Monorail Vehicle," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, 0013/2021-10-16, pp. 1–6.

[15] Z. Wang, Y. Wang, B. Zhang, G. Wang, T. Liu, J. Yi, and M. Han, "Development of a two-wheel steering unmanned bicycle: Simulation and experimental study," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, vol. 2020-July, 2020, pp. 119–124.

[16] L. Keo and M. Yamakita, "Controlling balancer and steering for bicycle stabilization," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, 2009, pp. 4541–4546.

[17] C. Xiong, Z. Huang, W. Gu, Q. Pan, Y. Liu, X. Li, and E. X. Wang, "Static Balancing of Robotic Bicycle through Nonlinear Modeling and Control," in *2018 3rd International Conference on Robotics and Automation Engineering (ICRAE)*, 0017/2018-11-19, pp. 24–28.

[18] C. Yang, S. Kim, T. Nozaki, and T. Murakami, "A self-balancing performance comparison of three modes of handleless electric motorcycles," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015-07-22/2015-07-24, pp. 352–357.

[19] C. Yang and T. Murakami, "Full-Speed Range Self-Balancing Electric Motorcycles Without the Handlebar," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 3, pp. 1911–1922, Mar. 2016.

[20] S. Hara, K. Nakagami, K. Miyata, M. Tsuchiya, and E. Tsujii, "Robust control system design for self-standable motorcycle," *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, vol. 14, no. 3, pp. JAMDSM0030–JAMDSM0030, 2020.

[21] S. Hara, M. Tsuchiya, and T. Kimura, "Robust Control of Automatic Low-Speed Driving Motorcycle "mOTOROiD"," in *2021 IEEE 10th Global Conference on Consumer Electronics, GCCE 2021*, 2021, pp. 645–646.

[22] M. Tsuchiya, S. Hara, T. Kimura, and N. Tsurumi, "Robust Control Strategy for Robotic Motorcycle Without Falling Down at Low-Speed Driving," *International Journal of Automotive Engineering*, vol. 13, no. 4, pp. 188–195, 2022.

[23] T. Sumioka, K. Akimoto, T. Tsujimura, S. Takayanagi, K. Fukushima, and T. Nose, "Rider Cooperative Control of Rear-Wheel-Swing Motorcycle Based on Divergent Component of Motion," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 223–230, 2024.

[24] X. Huang, F. Han, Y. Han, S. Wang, T. Liu, and J. Yi, "Motion Control of an Autonomous Wheel-Leg Bikebot," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 0020/2022-08-24, pp. 2341–2346.

[25] J. Meijaard, J. M. Papadopoulos, A. Ruina, and A. Schwab, "Linearized dynamics equations for the balance and steer of a bicycle: A benchmark and review," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 463, no. 2084, pp. 1955–1982, Jun. 2007.

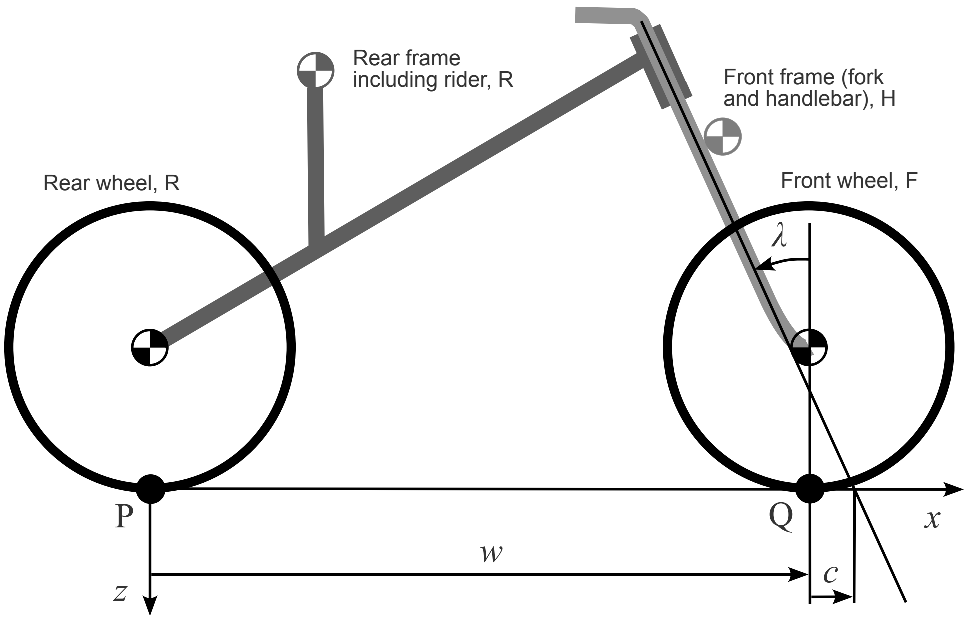
[26] J. Ronné, "Quantification de la maniabilité des vélos : évaluations et contributions aux approches expérimentales et théoriques," Ph.D. dissertation, Université Claude Bernard - Lyon I, Dec. 2024.

[27] Maxon, "Maxon 500267 EC 90 flat brushless, 260 W, with Hall sensors," https://www.maxongroup.com/maxon/view/product/500267.

[28] S. Narayanan and C. Antoniou, "Electric cargo cycles - A comprehensive review," *Transport Policy*, vol. 116, pp. 278–303, Feb. 2022.

[29] Anthropic. (2025) Claude ai. Accessed: May 20, 2025. [Online]. Available: https://claude.ai

Appendix A: Model parameters



Rear frame including rider, R
Front frame (fork and handlebar), H
Rear wheel, R
Front wheel, F
λ
P
Q
x
w
c
z

Fig. 15: Simplified bicycle visualizing the bicycle parameters and the coordinate system. Labeled parameters correspond with the parameters in Table VII.

TABLE VII: Parameters of the cargo bicycle with and without rider. The front assembly A consists of the handlebar and fork assembly and the front wheel.

Parameter	Symbol	Cargo bike	Cargo bike with rider	Cargo bike with rider & cargo	Unit
Gravitational acceleration	g	9.81	9.81	9.81	m/s^2
Steer axis tilt	λ	0.262	0.262	0.262	rad
Trail	c	0.0241	0.0241	0.0241	m
Wheel base	w	1.97	1.97	1.97	m
Front assembly (A)					
COM perpendicular distance from steering axis	u_A	-0.00364	-0.00364	-0.00364	m
Mass	m_A	8.5	8.5	8.5	kg
Horizontal position center of mass	x_A	1.85	1.85	1.85	m
Vertical position center of mass	z_A	-0.557	-0.557	-0.557	m
Mass moments of inertia	$\begin{bmatrix} I_{A\lambda\lambda} \\ I_{A\lambda x} \\ I_{A\lambda z} \end{bmatrix}$	$\begin{bmatrix} 0.0145 \\ -0.0511 \\ -0.0241 \end{bmatrix}$	$\begin{bmatrix} 0.0145 \\ -0.0511 \\ -0.0241 \end{bmatrix}$	$\begin{bmatrix} 0.0145 \\ -0.0511 \\ -0.0241 \end{bmatrix}$	kgm^2
Rear wheel (R)					
Mass	m_R	4.62	4.62	4.62	kg
Radius	r_R	0.305	0.305	0.305	kg
Mass moment of inertia	I_{Ryy}	0.166	0.166	0.166	kgm^2
Front wheel (F)					
Mass	m_F	2.67	2.67	2.67	kg
Radius	r_F	0.230	0.230	0.230	m
Mass moment of inertia	I_{Fyy}	0.09	0.09	0.09	kgm^2
Total system (T)					
Mass	m_T	49.0	129	209	kg
Horizontal position center of mass	x_T	0.811	0.5085	0.7885	m
Vertical position center of mass	z_T	-0.395	-0.801	-0.705	m
Mass moments of inertia	$\begin{bmatrix} I_{Txx} & 0 & I_{Txz} \\ 0 & 0 & 0 \\ I_{Txz} & 0 & I_{Tzz} \end{bmatrix}$	$\begin{bmatrix} 8.38 & 0 & 16.9 \\ 0 & 0 & 0 \\ 16.9 & 0 & 50.2 \end{bmatrix}$	$\begin{bmatrix} 32.6 & 0 & 10.3 \\ 0 & 0 & 0 \\ 10.3 & 0 & 60.9 \end{bmatrix}$	$\begin{bmatrix} 35.7 & 0 & 1.25 \\ 0 & 0 & 0 \\ 1.25 & 0 & 87.3 \end{bmatrix}$	kgm^2

Appendix B: Equations of motion feedback system

Filling in q and f in the equations of motion results in:

$$M\begin{bmatrix}\ddot{\phi}\\ \ddot{\delta}\end{bmatrix} + vC_1\begin{bmatrix}\dot{\phi}\\ \dot{\delta}\end{bmatrix} + \left(gK_0 + v^2K_2\right)\begin{bmatrix}\phi\\ \delta\end{bmatrix} = \begin{bmatrix}T_\phi\\ T_\delta\end{bmatrix} \tag{4}$$

Since the torque is modeled as a spring-damper system, it can be expressed as:

$$\begin{bmatrix}T_\phi\\ T_\delta\end{bmatrix} = \begin{bmatrix}-k_{roll}\phi - c_{roll}\dot{\phi}\\ -k_{steer}\delta - c_{steer}\dot{\delta}\end{bmatrix} \tag{5}$$

Where k_{roll} and k_{steer} are the stiffness coefficients on the roll and steer angle, respectively, and c_{roll} and c_{steer} are the damping coefficients on the roll and steer angle, respectively. This leads to the equations of motion:

$$\begin{bmatrix}M_{\phi\phi} & M_{\phi\delta}\\ M_{\delta\phi} & M_{\delta\delta}\end{bmatrix}\begin{bmatrix}\ddot{\phi}\\ \ddot{\delta}\end{bmatrix} + v\begin{bmatrix}C_{1\phi\phi} & C_{1\phi\delta}\\ C_{1\delta\phi} & C_{1\delta\delta}\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\delta}\end{bmatrix} + \left(g\begin{bmatrix}K_{0\phi\phi} & K_{0\phi\delta}\\ K_{0\delta\phi} & K_{0\delta\delta}\end{bmatrix} + v^2\begin{bmatrix}K_{2\phi\phi} & K_{2\phi\delta}\\ K_{2\delta\phi} & K_{2\delta\delta}\end{bmatrix}\right)\begin{bmatrix}\phi\\ \delta\end{bmatrix} = \begin{bmatrix}-k_{roll}\phi - c_{roll}\dot{\phi}\\ -k_{steer}\delta - c_{steer}\dot{\delta}\end{bmatrix} \tag{6}$$

Matrix multiplication gives:

$$\begin{bmatrix}M_{\phi\phi}\ddot{\phi} + M_{\phi\delta}\ddot{\delta}\\ M_{\delta\phi}\ddot{\phi} + M_{\delta\delta}\ddot{\delta}\end{bmatrix} + v\begin{bmatrix}C_{1\phi\phi}\dot{\phi} + C_{1\phi\delta}\dot{\delta}\\ C_{1\delta\phi}\dot{\phi} + C_{1\delta\delta}\dot{\delta}\end{bmatrix} + g\begin{bmatrix}K_{0\phi\phi}\phi + K_{0\phi\delta}\delta\\ K_{0\delta\phi}\phi + K_{0\delta\delta}\delta\end{bmatrix} + v^2\begin{bmatrix}K_{2\phi\phi}\phi + K_{2\phi\delta}\delta\\ K_{2\delta\phi}\phi + K_{2\delta\delta}\delta\end{bmatrix} = \begin{bmatrix}-k_{roll}\phi - c_{roll}\dot{\phi}\\ -k_{steer}\delta - c_{steer}\dot{\delta}\end{bmatrix} \tag{7}$$

From here, the feedback coefficients can be moved to the left-hand side of the equation. The stiffness coefficients are added to the K_2 matrix, but could also have been added to the K_0 matrix:

$$\begin{bmatrix}M_{\phi\phi}\ddot{\phi} + M_{\phi\delta}\ddot{\delta}\\ M_{\delta\phi}\ddot{\phi} + M_{\delta\delta}\ddot{\delta}\end{bmatrix} + v\begin{bmatrix}\left(C_{1\phi\phi} + \frac{\mathbf{c_{roll}}}{\mathbf{v}}\right)\dot{\phi} + C_{1\phi\delta}\dot{\delta}\\ C_{1\delta\phi}\dot{\phi} + \left(C_{1\delta\delta} + \frac{\mathbf{c_{steer}}}{\mathbf{v}}\right)\dot{\delta}\end{bmatrix} + g\begin{bmatrix}K_{0\phi\phi}\phi + K_{0\phi\delta}\delta\\ K_{0\delta\phi}\phi + K_{0\delta\delta}\delta\end{bmatrix} + v^2\begin{bmatrix}\left(K_{2\phi\phi} + \frac{\mathbf{k_{roll}}}{\mathbf{v^2}}\right)\phi + K_{2\phi\delta}\delta\\ K_{2\delta\phi}\phi + \left(K_{2\delta\delta} + \frac{\mathbf{k_{steer}}}{\mathbf{v^2}}\right)\delta\end{bmatrix} = 0 \tag{8}$$

Resulting in the coefficients affecting only four matrix components of the mass, damping, and stiffness matrices. With this knowledge, the stiffness and damping coefficients can be added to the C_1 and K_2 matrices, before working out the equations of motion:

$$C_1 = \begin{bmatrix}C_{1\phi\phi} + \frac{c_{roll}}{v} & C_{1\phi\delta}\\ C_{1\delta\phi} & C_{1\delta\delta} + \frac{c_{steer}}{v}\end{bmatrix},\ K_2 = \begin{bmatrix}K_{2\phi\phi} + \frac{k_{roll}}{v^2} & K_{2\phi\delta}\\ K_{2\delta\phi} & K_{2\delta\delta} + \frac{k_{steer}}{v^2}\end{bmatrix} \tag{9}$$

Appendix C: Design process

Using the design constraints and requirements, the design was broken down into 13 subproblems. These subproblems were:

- A way to move the system
- A way to control the system
- A way to tell the system what direction to move
- Compliance of the system
- A way the system makes contact with the ground
- A way the system knows the bike speed
- A way the system knows the legs are touching the ground
- A way the system knows the roll angle of the bike
- A way the system knows the legs are at the end of their movement range
- A way rotational movement is transferred to linear movement (if required)
- A way to increase the system's force/torque (if required)
- The materials used for the system
- The way the system is attached to the bike

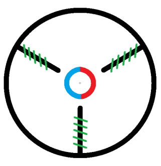
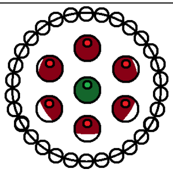
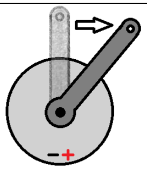
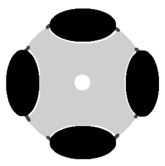
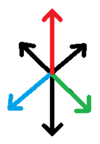
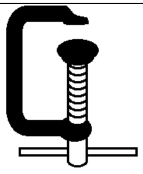
These subproblems could be categorized as different functions to help the design process. Using these functions, a morphological chart was made where each function had multiple solutions for each subproblem (Table XII). From the morphological chart, multiple concepts could be made.

Since the system would activate below a certain speed, all concepts require a speed sensor, which is therefore not added to the morphological chart. Materials are also not added to the morphological chart, since the final decision on the material was made during the final design after the concept had been chosen.

Concepts

Concept A

TABLE VIII: Solutions used for Concept A

					
Brushless motor	Cycloidal drive	Motor compliance	Omni-wheels	Accelerometer	Clamp

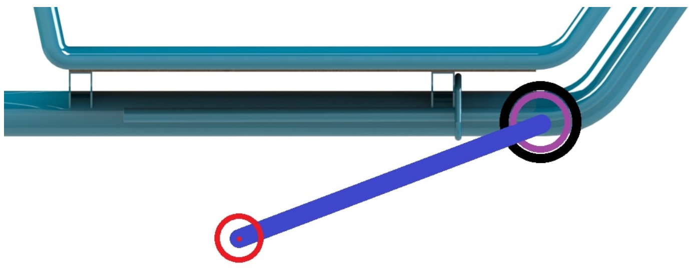


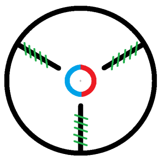
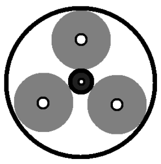
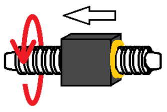
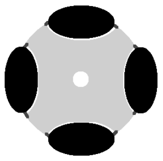
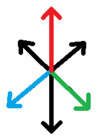
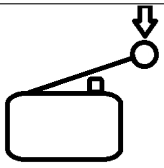
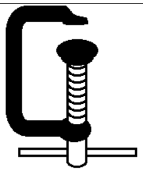
Fig. 16: Sketch of Concept A. The motor is shown in black, the gearbox is purple, the leg is blue, and the wheel is red.

Concept A uses a brushless DC motor with a cycloidal drive to move the legs. The motor shaft is perpendicular to the main frame tube, and the leg is directly attached to the output shaft of the gearbox. Since the cycloidal drive is back-drivable, the system can use motor compliance to achieve a compliant system, so that the bicycle dynamics can remain. The results from the simulations in Section II can be used to find the motor torque and the cycloidal drive gear ratio. Omni-wheels allow for lateral rolling of the wheels. The system uses an accelerometer to find the roll angle of the bike. Attachment to the cargo bike is done with clamps.

Concept B

Since two similar concepts were made, Concept B consists of Concept B.1 and Concept B.2. Figure 17 shows a sketch of Concept B.

TABLE IX: Solutions used for Concept B. Concept B.1 uses a lead screw, while Concept B.2 uses a ball screw.

 Brushless motor	 Planetary gearbox	 Lead screw/ ball screw	 Omni-wheels	 Accelerometer	 Limit switch	 Clamp

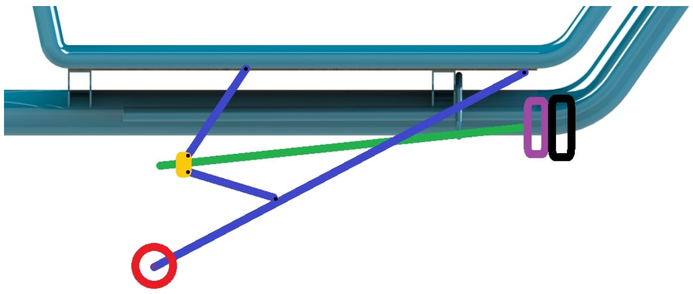


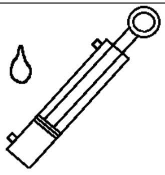
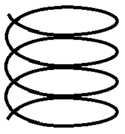
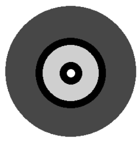
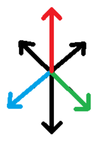
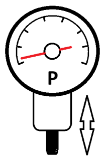
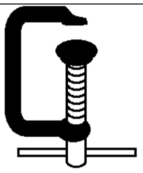
Fig. 17: Sketch of Concept B. The motor is shown in black, the gearbox in purple, the leg and linkages in blue, the spindle is green, and the spindle nut is gold.

Concept B uses a brushless DC motor with a planetary gearbox. The rotational movement is transferred to a linear movement using a spindle and a spindle nut. Concept B.1 uses a trapezoidal lead screw, while Concept B.2 uses a ball screw. The linkages transfer the movement from the spindle nut to the leg. Like in Concept A, omni-wheels are used. An accelerometer determines the roll angle of the bike. Additionally, limit switches trigger when the legs touch the ground or are at the end of their movement range.

Since a ball screw is back-drivable, Concept B.2 can use motor compliance to improve bicycle dynamics while the system is active and the legs are touching the ground. However, ball screw assemblies are significantly more expensive than lead screw assemblies and therefore Concept B.1 was also considered, which are not back-drivable.

Concept C

TABLE X: Solutions used for Concept C.

 Hydraulic motor	 Springs	 Wheels	 Accelerometer	 Pressure sensor	 Clamps

Concept C uses a hydraulic cylinder as the actuator. The wheels are attached to the cylinder through springs, which add compliance to the system, improving bicycle dynamics. The wheels caster on the cylinder ends, so they can also move sideways when the bike is taking a turn. Pressure sensors are used to determine when the wheels make contact with the ground.

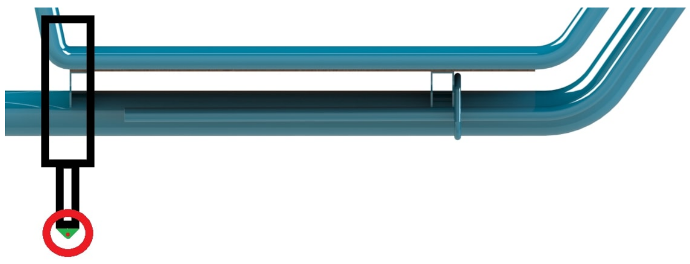


Fig. 18: Sketch of Concept C. The hydraulic cylinder is shown in black, the wheel in red, and the spring is green.

Concept selection

To select the most suitable concept for the final design, assessment criteria were formulated. The criteria were as follows:

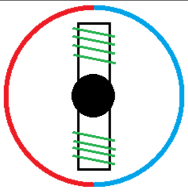
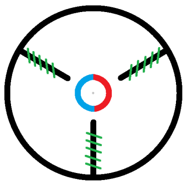
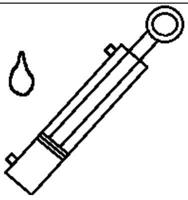
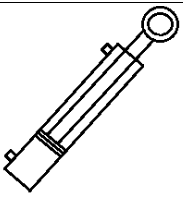
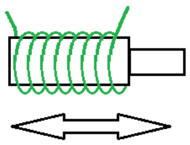
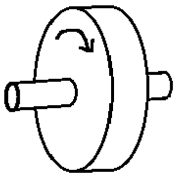
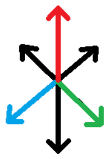
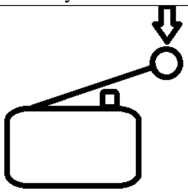
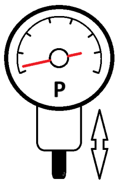
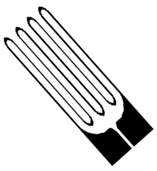
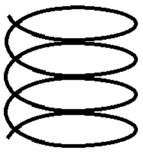
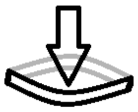
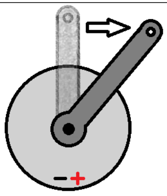
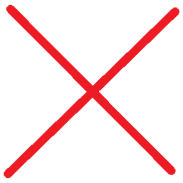
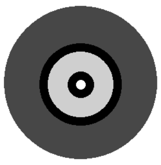
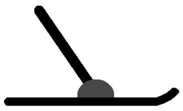
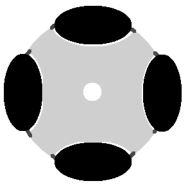
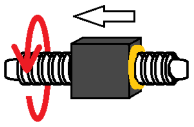
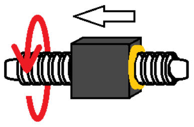
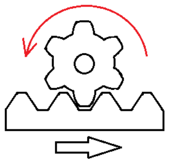
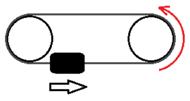
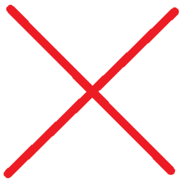
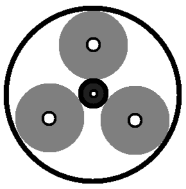
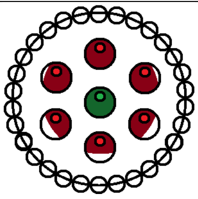
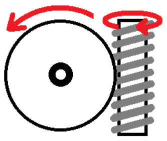
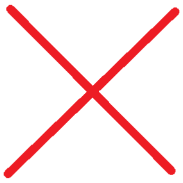
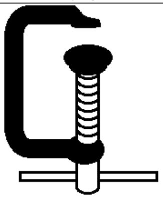
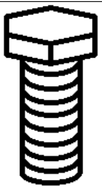
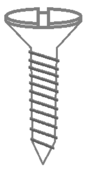
- Cost
- Dynamics
- Stability
- Weight

The criteria were ordered from important to least important, and then were given a weight so they could be used for concept evaluation. Each concept was given a score on a 1-5 scale according to its expected performance for each criterion. The scores were multiplied by the weights, and the sum of the weighted scores gave a final score that indicated which concept would perform best. Table XI shows that Concept B.1 scores the highest during the evaluation. Therefore, Concept B.1 will be further worked out into a final design.

TABLE XI: Weighted criteria and concept evaluation.

Criterion	Weight	Concept A	Concept B.1	Concept B.2	Concept C
Stability	0.4	1	4	4	4
Cost	0.3	3	4	2	2
Dynamics	0.2	5	1	3	2
Mass	0.1	5	4	4	1
Weighted score		2.8	3.4	3.2	2.7

TABLE XII: Morphological chart used for the design process. The first column shows the functions in which the solutions can be categorized.

Actuation	 Brushed	 Brushless	 Hydraulic	 Pneumatic	 Solenoid
Sensing	 Gyroscope	 Accelerometer	 Limit switch	 Pressure sensor	 Strain gauge
Compliance	 Springs	 Compliant materials	 Motor compliance	 None	
Ground contact	 Wheels	 Sliders	 Omni-wheels		
Linear transmission	 Lead screw	 Ball screw	 Rack & pinion	 Belts/pulleys	 None
Rotational transmission	 Planetary gearbox	 Cycloidal drive	 Worm drive	 None	
Attachment	 Clamp	 Bolt	 Screw		

Appendix D: Electronics overview and user interface

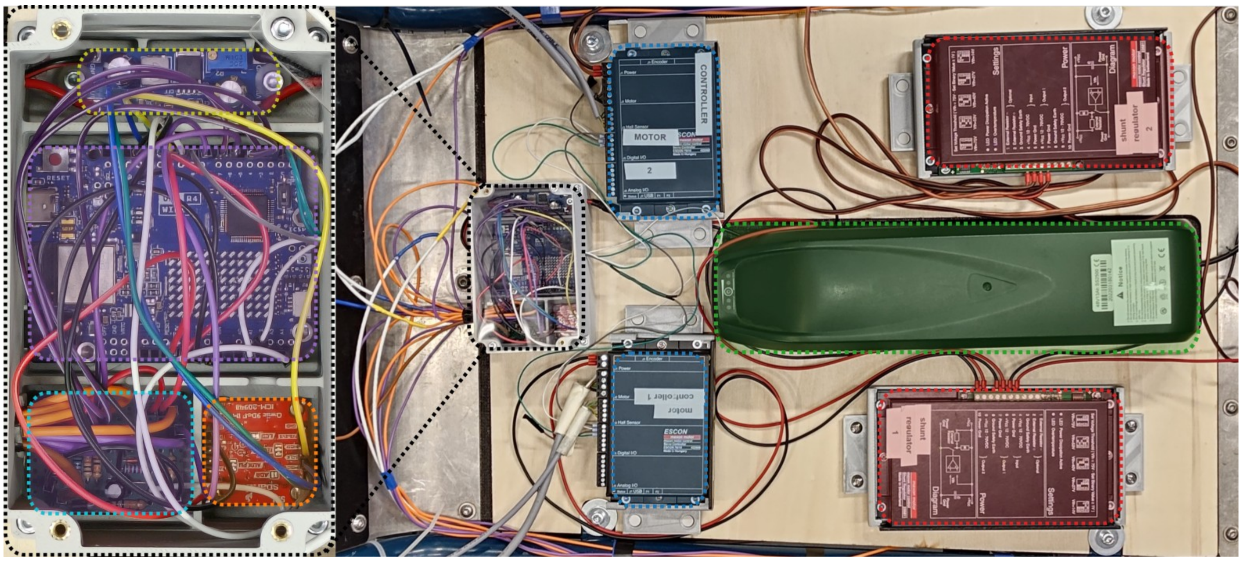


Fig. 19: Detailed overview of the electronics of the system. Top right to bottom left: Red: Shunt regulators, Green: Battery, Blue: motor controllers, Yellow: DC-DC converter, Purple: Arduino UNO R4 WiFi, Orange: Sparkfun ICM-20948 IMU, Turquoise: custom circuit board (shown in Figure 22 in Appendix E).

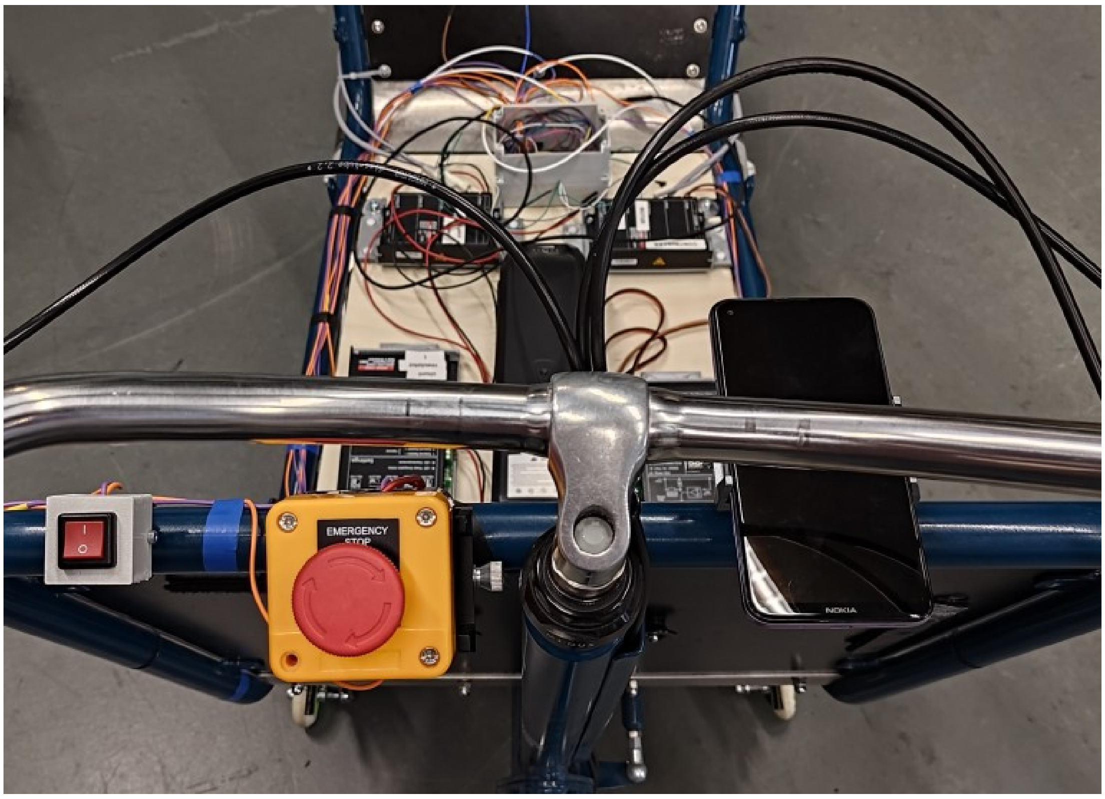


Fig. 20: The cargo bike as seen from the driver's point of view. Under the handlebar are the on/off switch, the emergency switch, and the phone that was used for cloud connection and for gathering accelerometer data during testing.

Appendix E: Wiring diagram

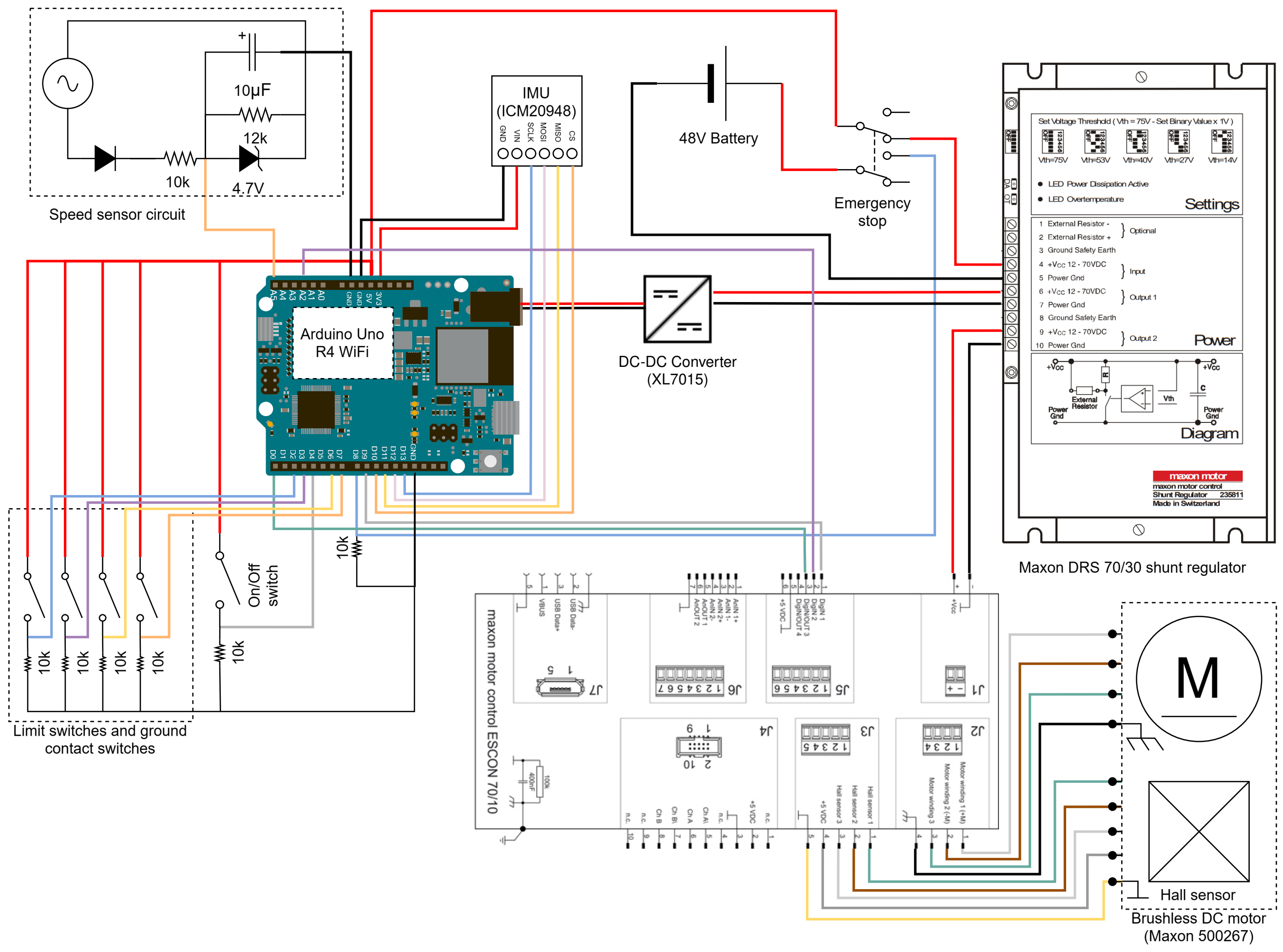


Fig. 21: Wiring diagram of the system. For simplicity, only one shunt regulator, motor controller, and motor are shown. Red wires indicate the positive terminal of the DC supply, while black wires represent the negative terminal (ground/GND). Other color wires are used for logic signals or data transfer.

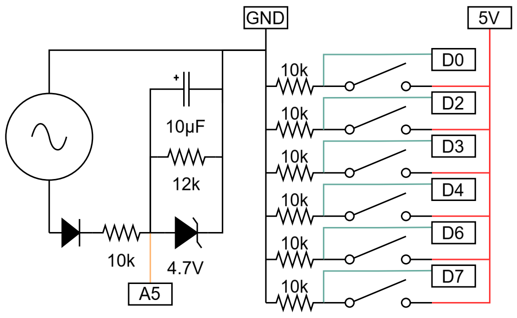


Fig. 22: Schematic of the custom circuit board. The custom circuit board includes the speed sensor circuit and the circuitry for the limit switches, ground contact switches, on/off switch, and emergency stop. This wiring is also present in Figure 21. The boxes represent the inputs and outputs of the circuit board. Red wires indicate the positive terminal of the DC supply, while black wires represent the negative terminal (ground/GND). Other color wires are used for logic signals or data transfer.

Appendix F: Arduino code

The Arduino code consists of three different files.

A. *Main code*

The main code runs the state machine, controls the motors, and receives input from the (limit) switches. The motor controller settings, which are set in ESCON Studio, are incorporated as constants in the Arduino code to set the motors' RPM correctly. The controller settings can be changed as desired. The controller runs at 200 Hz, but the rate can be changed as desired.

At first, cloud connectivity was enabled in the code to get live readings from the system and to set the speed threshold while cycling. However, this introduced a problem where the system would freeze for 10 seconds whenever the Arduino lost internet connection, so it was removed in the latest version. The code can be seen in Listing 1.

B. *IMU_Handler library*

The IMU_Handler communicates with the IMU over SPI. The library was generated using Claude.ai [29]. The low-pass filter for the IMU data is integrated in the library. The filtered X and Z accelerations are used to calculate the roll angle. The dt, which is determined by the controller rate of the main code, is required as input for the filter. The code can be seen in Listing 2.

C. *Header file*

The header file for the IMU_Handler. The header file was also generated by Claude.ai. The code can be seen in Listing 3.

Listing 1: Main code running on the Arduino.

```
/*
Cargo bike balance assist (CBBA) Arduino code

by Bart de Vries

as part of the graduation project for the master of Mechanical Engineering, department of
    Biomechanical Engineering

Latest version: 17-5-2025 21:34
*/

// LIBRARIES

#include <SPI.h>
#include <IMU_Handler.h>

// CONSTANTS & VARIABLES

const bool down = 1;  // Value controller accepts to turn leg down
const bool up = 0;    // Value controller accepts to turn leg up

int check = 0;  // Check is used to not switch states due to false positives

const float angleThreshold = 15.0;    // Angle threshold. Any angle above this will not
    activate the system
const float engagingTolerance = 1.5;  // Angle tolerance to control motors during engaging (see
     engagingHandler)

// Speed thresholds for engaging and disengaging
int cyclingThreshold = 450;
int walkingThreshold = 50;

// Stores values of switch signals:
bool touchL;
bool touchR;
bool limitL;
bool limitR;

// Motor RPM and PWM values
const int maxPWM = 230;  // PWM  signal to get max motor RPM
const int minPWM = 25;   // PWM signal to get 0 RPM on the motor
const int minmaxPWM = maxPWM - minPWM;
const int maxRPM = 2000;                                                    // Maximum RPM of the motor (As set in the controller: ESCON Studio)
const int disengagingRPM = 1670;                                            // RPM for disengaging the system (change as desired)
const int maxControllingRPM = 1670;                                         // Max RPM for controlling
const int minControllingRPM = 500;                                          // Min RPM for controlling
const int disengagingPWM = int(minmaxPWM * disengagingRPM / maxRPM) + minPWM;  // PWM value for disengaging the system
const int maxControllingPWM = int(minmaxPWM * maxControllingRPM / maxRPM) + minPWM; // PWM for maximum controlling RPM
const int minControllingPWM = int(minmaxPWM * minControllingRPM / maxRPM) + minPWM; // PWM for minimum controlling RPM

// Filter parameters
const float fc = 1.0;           // Cutoff frequency
const float fs = 200.0;         // Sampling frequency and frequency at which the system runs
const float speedAlpha = 0.05;  // Filter constant for speed
int previousSpeed = 0;          // Store previous filtered speed value
int bikeSpeed;
float dt;                       // dt value used for IMU accelerations and integral and derivative errors for PID

// Controller constants
const float targetAngle = 0.0;  // Target angle
const float tolerance = 1.0;    // Tolerance on target angle
float currentAngle;

// PID parameters
const float Kp = 0.8;
const float Ki = 0.1;
const float Kd = 0.01;
const float maxIntegralError = 5.0;

// Controller variables
float integralError;                                  // Integral error for integral calculation
float previousError;                                  // Previous error for derivative calculation
const unsigned long controlInterval = 1000000 / fs;  // Control interval to set the rate of the controller using microseconds
int motorSpeed;

// Stores values for timers
unsigned long lastTime = 0;     // Timer for
unsigned long motorTimer = 0;  // Timer for walking state motor control

// States:
int currentState;
int previousState;
enum State {
  DEACTIVATED,
  ACTIVATED,
  ENGAGING,
  DISENGAGING,
  WALKING,
  CONTROLLING,
  EMERGENCY
};

// PINS
// Input pins:
const int OnOff = 4;  // Turn the system on and off
const int eStop = 8;  // Emergency stop

const int LimitL = 3;  // Sensor to check end of travel of leg 1 (wired to button)
const int LimitR = 2;  // Sensor to check end of travel of leg 2 (wired to button)

const int touchdownL = 6;  // Sensor to check if the wheel touches the ground (wired to button)
const int touchdownR = 7;  // Sensor to check if the wheel touches the ground (wired to button)

const int speedPin = A5;  // Speed sensor (voltage sensor from dynamo) (bike speed)

// Output pins:
const int OnL = A2;  // Turn on/off left motor
const int DirL = 0;  // Direction of left motor
const int OnR = A1;  // Turn on/off of right motor
const int DirR = 1;  // Direction of right motor

const int PWMpin = 9;  // Sets motor speed (for both motors)

// Other pins:
IMU_Handler imu(10);  // Set CS pin for SPI communication between IMU and Arduino



void setup() {
  Serial.begin(115200);  // Initialize Serial
  log("Setup starting.");

  SPI.begin();  // Initialize SPI

  // Initialize the IMU
  log("Attempting to connect to IMU...");
  if (imu.begin()) {
    log("IMU initialized successfully");
  } else {
    log("Failed to initialize IMU");
    while (1)
      ;                              // Infinite loop the program if IMU initialization fails
  }

  String filterMsg = "Lowpass filter used for IMU data. Using cutoff frequency fc = " + String(
    fc) + " Hz, sampling frequency fs = " + String(fs);
  log(filterMsg);

  imu.setFilterFrequency(fc, fs);  // Set cutoff frequency and sampling frequency for the
    filter (calculates alpha in .cpp)

  // Setting pins to OUTPUT or INPUT
  pinMode(OnL, OUTPUT);
  pinMode(DirL, OUTPUT);
  pinMode(OnR, OUTPUT);
  pinMode(DirR, OUTPUT);
  pinMode(LimitL, INPUT);
  pinMode(LimitR, INPUT);
  pinMode(OnOff, INPUT);
  pinMode(eStop, INPUT);
  pinMode(touchdownL, INPUT);
  pinMode(touchdownR, INPUT);

  // Attach interrupts to limit switches
  attachInterrupt(digitalPinToInterrupt(LimitL), limitLInterrupt, RISING);
  attachInterrupt(digitalPinToInterrupt(LimitR), limitRInterrupt, RISING);

  // Set initial state
  if ((digitalRead(OnOff) == 0) && (digitalRead(LimitL) == 0 || digitalRead(LimitR) == 0)) {
    log("Current state is Deactivated: Disengaging system.");
    currentState = DISENGAGING;
  } else if (digitalRead(OnOff) == 1) {
    log("Current state is Activated.");
    currentState = ACTIVATED;
  } else {
    log("Current state is Deactivated.");
    currentState = DEACTIVATED;
  }
}
```

```

void loop() {
  // Emergency stop
  if (digitalRead(eStop) == 1) {
    if (currentState != EMERGENCY) {
      log("Emergency stop activated!");
    }
    currentState = EMERGENCY;
  }

  // Continue if emergency stop is released
  if (digitalRead(eStop) == 0 && currentState == EMERGENCY) {
    log("System deactivated.");
    currentState = DEACTIVATED;
  }

  // Run system at defined rate
  unsigned long currentTime = micros();
  if ((currentTime - lastTime) < controlInterval) {
    return;  // Exit if not enough time has passed
  }

  dt = (currentTime - lastTime) / 1000000.0;  // dt in seconds
  lastTime = currentTime;

  // Update IMU data
  imu.update(dt);
  currentAngle = imu.getRoll();

  // Update bike speed
  bikeSpeed = readSpeed();

  // Check angle threshold
  if (abs(currentAngle) > angleThreshold) {
    log("Angle too large, stopping system.");
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
    return;
  }

  // Main state machine
  switch (currentState) {
    case DEACTIVATED:
      handleDeactivated();
      break;

    case ACTIVATED:
      handleActivated();
      break;

    case ENGAGING:
      handleEngaging();
      break;

    case DISENGAGING:
      handleDisengaging();
      break;

    case WALKING:
      handleWalking();
      break;

    case CONTROLLING:
      handleControlling();
      break;

    case EMERGENCY:
      emergencyBrake();
      break;

    default:
      log("Error: entered undefined state.");
      currentState = DEACTIVATED;
      break;
  }
}

void log(const String& message) {  // Function to log messages to Arduino Cloud. Replaces
    Serial.print
  Serial.println(message);
  String serialOutput = message;
}

void limitLInterrupt() {                      // Interrupt for left limit switch
  digitalWrite(DirL, !digitalRead(DirL));   // Change direction for some braking
  digitalWrite(OnL, 0);
}

void limitRInterrupt() {                      // Interrupt for right limit switch
  digitalWrite(DirR, !digitalRead(DirR));   // Change direction for some braking
  digitalWrite(OnR, 0);
}

void emergencyBrake() {
  digitalWrite(DirL, !digitalRead(DirL));   // Reverse direction of left motor for braking
  digitalWrite(DirR, !digitalRead(DirR));   // Reverse direction of right motor for braking
  analogWrite(PWMpin, 230);                 // Increase PWM to maximum to increase braking force

  delay(100);

  digitalWrite(OnL, 0);
  digitalWrite(OnR, 0);
}

int readSpeed() {                            // Function that reads the speed of the bike and
    filters it
  int rawSpeed = analogRead(speedPin);
  bikeSpeed = (speedAlpha * rawSpeed) + ((1 - speedAlpha) * previousSpeed);   // Exponential
    filter to smooth out the speed reading
  previousSpeed = bikeSpeed;
  return bikeSpeed;
}

void handleDeactivated() {
  if (digitalRead(OnOff) == 1) {  // Checks to see if on/off switch is activated
    currentState = ACTIVATED;
    log("Going to state: Activated");
  } else if (digitalRead(LimitL) == 0 || digitalRead(LimitR) == 0) {  // Recheck if legs are
    really up
    log("System is deactivated, but legs are not up: Disengaging.");
    currentState = DISENGAGING;
  }
}

void handleActivated() {
  if (digitalRead(OnOff) == 0) {  // Checks to see if on/off switch is activated
    currentState = DEACTIVATED;
    log("Going to state: Deactivated");
    return;
  }

  if (bikeSpeed < cyclingThreshold) {        // Checks speed with threshold
    log("Speed under threshold: Engaging.");
    check = 0;
    currentState = ENGAGING;
  } else if (digitalRead(LimitL) != 1 || digitalRead(LimitR) != 1) {    // Checks to see if
    legs are up when they should be
    log("System is activated, but speed is above threshold and legs are not up: Disengaging.");
    currentState = DISENGAGING;
  }
}

void handleEngaging() {
  touchL = digitalRead(touchdownL);
  touchR = digitalRead(touchdownR);

  if (touchL == 1 || touchR == 1) {  // Sets motor RPM to max, but when a leg touches it is slowed down to max Controlling RPM
    analogWrite(PWMpin, maxControllingPWM);
  } else {
    analogWrite(PWMpin, 230);  // Set speed to max for engaging before touchdown
  }

  if (digitalRead(OnOff) == 0) {    // Checks to see if on/off switch is activated
    currentState = DISENGAGING;
    log("System deactivated: Disengaging.");
    return;
  }

  if (bikeSpeed > cyclingThreshold) {  // If the bike is moving too fast, disengage
    currentState = DISENGAGING;
    String message = "Speed (" + String(bikeSpeed) + ") above threshold: Disengaging.";
    log(message);
    return;
  }

  // Setting direction for both motors down
  digitalWrite(DirL, down);
  digitalWrite(DirR, down);

  if (touchL == 0 && touchR == 0) {  // Logic to control motors (tilted left is postive, tilted right is negative)
    digitalWrite(OnL, 1);
    digitalWrite(OnR, 1);
  } else if (touchL == 1 && currentAngle > engagingTolerance) {  // Left touches and bike is tilted left > keep moving both legs
    digitalWrite(OnL, 1);
    digitalWrite(OnR, 1);
  } else if (touchR == 1 && currentAngle < -engagingTolerance) {  // Right touches and bike is tilted right > keep moving both legs
    digitalWrite(OnL, 1);
    digitalWrite(OnR, 1);
  } else if (touchL == 1 && currentAngle < -engagingTolerance) {  // Left touches, but bike is tilted right > only move right
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 1);
  } else if (touchR == 1 && currentAngle > engagingTolerance) {  // Right touches, but bike is tilted left > only move left
    digitalWrite(OnL, 1);
    digitalWrite(OnR, 0);
  }

  if (touchR == 1 && touchL == 1) {   // If both legs touch down, go the controlling state
    check += 1;
    if (check > 20) {  // Stop system from going to CONTROLLING from false true button values
      digitalWrite(OnR, 0);
      digitalWrite(OnL, 0);
      currentState = CONTROLLING;
      log("Both legs are down: Controlling");
    }
  }
}

void handleDisengaging() {
  analogWrite(PWMpin, disengagingPWM);  // Set speed for disengaging

  limitL = digitalRead(LimitL);
  limitR = digitalRead(LimitR);

  digitalWrite(DirL, up);
  digitalWrite(DirR, up);

  if (digitalRead(OnOff) == 1 && bikeSpeed < cyclingThreshold) {  // If the bike slows down while activated, engage
    String message = "Speed (" + String(bikeSpeed) + ") under threshold while disengaging: Engaging.";
    log(message);
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
    currentState = ENGAGING;
    return;
  }

  if (limitL == 0) {      // Turn off left motor if limit switch is activated
    digitalWrite(OnL, 1);
    check = 0;
  } else {
    digitalWrite(OnL, 0);
  }

  if (limitR == 0) {    // Turn off right motor if limit switch is activated
    digitalWrite(OnR, 1);
    check = 0;
  } else {
    digitalWrite(OnR, 0);
  }

  if (limitL == 1 && limitR == 1) {   // If both legs are up, go to deactivated state
    check += 1;
    if (check > 2) {
      log("Both legs are up: Deactivating.");
      currentState = DEACTIVATED;
    }
  }
}

void handleWalking() {    // Turns on motor for X seconds to slightly raise the wheels off the ground
  analogWrite(PWMpin, disengagingPWM);  // Set motor speed

  touchL = digitalRead(touchdownL);
  touchR = digitalRead(touchdownR);
  limitL = digitalRead(LimitL);
  limitR = digitalRead(LimitR);

  digitalWrite(DirL, up);   // Legs can only move up while in Walking mode
  digitalWrite(DirR, up);

  if (bikeSpeed > cyclingThreshold || digitalRead(OnOff) == 0) {    // If the bike is moving too fast or the system is deactivated, disengage
    String message = "Speed (" + String(bikeSpeed) + ") above threshold while walking: Disengaging.";
    log(message);
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
    currentState = DISENGAGING;
    return;
  } else if (bikeSpeed < (walkingThreshold - 30)) {       // If the bike is moving too slow, engage
    String message = "Speed (" + String(bikeSpeed) + ") under threshold while walking: Engaging.";
    log(message);
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
    currentState = ENGAGING;
    return;
  }

  // Motor control
  if (millis() - motorTimer >= 500) {  // Check if 0.5 seconds have passed to turn off the motors
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
  } else {
    digitalWrite(OnL, 1);
    digitalWrite(OnR, 1);
  }

  if (previousState != currentState) {  // Reset timer when entering walking state
    motorTimer = millis();
  }

  previousState = currentState;
}

void handleControlling() {
  previousState = currentState;
  touchL = digitalRead(touchdownL);
  touchR = digitalRead(touchdownR);

  if (bikeSpeed > (walkingThreshold + 30)) {    // If the bike is above walking speed, go to walking state
    currentState = WALKING;
    String message = "Speed (" + String(bikeSpeed) + ") above threshold while controlling: Walking.";
    log(message);
    return;
  }

  if (touchL == 0 || touchR == 0) {       // If either wheel is not touching the ground, go to engaging state
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
    currentState = ENGAGING;
    log("Legs lost contact with the ground while controlling: Engaging.");
    return;
  }

  if (digitalRead(OnOff) == 0) {         // Disengage if the system is deactivated
    currentState = DISENGAGING;
    log("System deactivated while controlling: Disengaging.");
    return;
  }

  // Function that actually controls the motors:
  angleController();

}

void angleController() {
  limitL = digitalRead(LimitL);
  limitR = digitalRead(LimitR);
  touchL = digitalRead(touchdownL);
  touchR = digitalRead(touchdownR);

  // Calculate error
  float error = targetAngle - currentAngle;

  // Update integral term
  integralError += error * dt;
  integralError = constrain(integralError, -maxIntegralError, maxIntegralError);

  // Calculate derivative term
  float derivativeError = (error - previousError) / dt;
  previousError = error;

  // Calculate control effort using PID controller
  float controlEffort = Kp * error + Ki * integralError + Kd * derivativeError;

  // Map absolute control effort to PWM value (10% to 90% -> values motor controller accepts)
  motorSpeed = constrain(map(abs(controlEffort), 0, maxIntegralError, minControllingPWM,
    maxControllingPWM), minControllingPWM, maxControllingPWM);

  // Determine motor actions based on control effort
  if (abs(error) <= tolerance) {
    // Within tolerance band, stop both motors
    digitalWrite(OnL, 0);
    digitalWrite(OnR, 0);
  } else if (controlEffort > 0) {
    // Angle is positive (tilted left), need to tilt right
    digitalWrite(DirL, up);
    digitalWrite(DirR, down);

    // Check to see if the left limit switch is activated, it cannot move up if it is
    if (limitL == 1) {
      digitalWrite(OnL, 0);
      digitalWrite(OnR, 0);
    } else {
      if (touchL == 0) {        // If statement checks if left wheel is still on the ground
        digitalWrite(OnL, 0);   // Legs do not move up if the wheel is not touching
      } else {
        digitalWrite(OnL, 1);
      }
      digitalWrite(OnR, 1);
      analogWrite(PWMpin, motorSpeed);
    }
  } else {
    // Angle is negative (tilted right), need to tilt left
    digitalWrite(DirL, down);
    digitalWrite(DirR, up);

    // Check to see if the right limit switch is activated
    if (limitR == 1) {
      digitalWrite(OnL, 0);
      digitalWrite(OnR, 0);
    } else {
      if (touchR == 0) {        // If statement checks if right wheel is still on the ground
        digitalWrite(OnR, 0);   // Leg does not move up if wheel is not touching
      } else {
        digitalWrite(OnR, 1);
      }
      digitalWrite(OnL, 1);
      analogWrite(PWMpin, motorSpeed);
    }
  }
}
```

Listing 2: Library written by Claude.ai for communications with the Sparkfun ICM20948 IMU.

```
#include "IMU_Handler.h"
#include <math.h>

// ICM20948 SPI registers
#define ICM20948_REG_BANK_SEL    0x7F
#define ICM20948_PWR_MGMT_1      0x06    // Bank 0
#define ICM20948_ACCEL_XOUT_H    0x2D    // Bank 0
#define ICM20948_ACCEL_CONFIG    0x14    // Bank 2

// Constructor
IMU_Handler::IMU_Handler(int csPin) :
  CS_PIN(csPin),
  spiSettings(1000000, MSBFIRST, SPI_MODE0),
  alpha(0.1f) // Default filter value, will be recalculated in setFilterFrequency
{
  // Initialize filtered values
  filteredAccelX = 0.0f;
  filteredAccelZ = 0.0f;
  prevFilteredAccelX = 0.0f;
  prevFilteredAccelZ = 0.0f;
}

bool IMU_Handler::begin() {
  // Initialize SPI and CS pin
  pinMode(CS_PIN, OUTPUT);
  digitalWrite(CS_PIN, HIGH);
  SPI.begin();

  // Reset the device
  writeRegister(ICM20948_REG_BANK_SEL, 0x00); // Select Bank 0
  writeRegister(ICM20948_PWR_MGMT_1, 0x80);   // Reset device
  delay(100);

  // Wake up the device
  writeRegister(ICM20948_REG_BANK_SEL, 0x00); // Select Bank 0
  writeRegister(ICM20948_PWR_MGMT_1, 0x01);   // Auto select best available clock
  delay(10);

  // Configure accelerometer (Bank 2)
  writeRegister(ICM20948_REG_BANK_SEL, 0x20); // Select Bank 2
  writeRegister(ICM20948_ACCEL_CONFIG, 0x00); // Set accel to +/- 2g

  // Return to Bank 0 for normal operation
  writeRegister(ICM20948_REG_BANK_SEL, 0x00);

  // Setup default low-pass filter (1Hz with 100Hz sample rate)
  setFilterFrequency(1.0f, 100.0f);

  return true; // Add error checking in a real implementation
}

void IMU_Handler::writeRegister(uint8_t reg, uint8_t data) {
  SPI.beginTransaction(spiSettings);
  digitalWrite(CS_PIN, LOW);
  SPI.transfer(reg & 0x7F); // Bit 7 low for write
  SPI.transfer(data);
  digitalWrite(CS_PIN, HIGH);
  SPI.endTransaction();
}

uint8_t IMU_Handler::readRegister(uint8_t reg) {
  uint8_t data;
  SPI.beginTransaction(spiSettings);
  digitalWrite(CS_PIN, LOW);
  SPI.transfer(reg | 0x80); // Bit 7 high for read
  data = SPI.transfer(0);
  digitalWrite(CS_PIN, HIGH);
  SPI.endTransaction();
  return data;
}

void IMU_Handler::readXZAccelData(int16_t* accelX, int16_t* accelZ) {
  uint8_t rawDataX[2]; // Buffer for X accelerometer data
  uint8_t rawDataZ[2]; // Buffer for Z accelerometer data

  // Switch to Bank 0 where accelerometer data registers are located
  writeRegister(ICM20948_REG_BANK_SEL, 0x00);

  // Read X accelerometer data
  SPI.beginTransaction(spiSettings);
  digitalWrite(CS_PIN, LOW);
  // ACCEL_XOUT_H register address
  SPI.transfer(ICM20948_ACCEL_XOUT_H | 0x80); // | 0x80 for read operation

  // Read 2 bytes for X accelerometer data
  rawDataX[0] = SPI.transfer(0); // High byte
  rawDataX[1] = SPI.transfer(0); // Low byte

  digitalWrite(CS_PIN, HIGH);
  SPI.endTransaction();

  // Read Z accelerometer data
  SPI.beginTransaction(spiSettings);
  digitalWrite(CS_PIN, LOW);
  // ACCEL_ZOUT_H register address (ACCEL_XOUT_H + 4)
  SPI.transfer((ICM20948_ACCEL_XOUT_H + 4) | 0x80); // | 0x80 for read operation

  // Read 2 bytes for Z accelerometer data
  rawDataZ[0] = SPI.transfer(0); // High byte
  rawDataZ[1] = SPI.transfer(0); // Low byte

  digitalWrite(CS_PIN, HIGH);
  SPI.endTransaction();

  // Convert accelerometer data to 16-bit signed values
  *accelX = (int16_t)((rawDataX[0] << 8) | rawDataX[1]);
  *accelZ = (int16_t)((rawDataZ[0] << 8) | rawDataZ[1]);
}

void IMU_Handler::update(float dt) {
  int16_t rawAccelX, rawAccelZ;
  float accelXG, accelZG;

  // Read X and Z accelerometer data from IMU
  readXZAccelData(&rawAccelX, &rawAccelZ);

  // Convert to G forces
  const float accelScale = 1.0f / 16384.0f; // For +/- 2g range
  accelXG = rawAccelX * accelScale;
  accelZG = rawAccelZ * accelScale;

  // Apply low-pass filter to accelerometer data
  filteredAccelX = alpha * accelXG + (1.0f - alpha) * prevFilteredAccelX;
  filteredAccelZ = alpha * accelZG + (1.0f - alpha) * prevFilteredAccelZ;

  // Update previous values for next iteration
  prevFilteredAccelX = filteredAccelX;
  prevFilteredAccelZ = filteredAccelZ;
}

float IMU_Handler::getRoll() {
  // Calculate roll angle using filtered X and Z acceleration
  float roll = atan2(filteredAccelZ, filteredAccelX);

  // Convert to degrees
  return roll * 180.0f / M_PI + 90.0;
}
```

```

void IMU_Handler::setFilterFrequency(float cutoffFreq, float sampleFreq) {
  // Calculate filter coefficient
  float RC = 1.0f / (2.0f * M_PI * cutoffFreq);
  float dt = 1.0f / sampleFreq;
  alpha = dt / (RC + dt);
}

float IMU_Handler::getFilteredAccelX() {
  return filteredAccelX;
}

float IMU_Handler::getFilteredAccelZ() {
  return filteredAccelZ;
}
```

Listing 3: Header file for the IMU_Handler library

```cpp
#ifndef IMU_HANDLER_H
#define IMU_HANDLER_H

#include <Arduino.h>
#include <SPI.h>

class IMU_Handler {
  private:
    // SPI settings
    const int CS_PIN;
    SPISettings spiSettings;

    // Low-pass filter variables
    float filteredAccelX;      // Filtered X accelerometer data
    float filteredAccelZ;      // Filtered Z accelerometer data
    float prevFilteredAccelX;  // Previous filtered X value
    float prevFilteredAccelZ;  // Previous filtered Z value
    float alpha;               // Filter coefficient

    // Private methods
    void writeRegister(uint8_t reg, uint8_t data);
    uint8_t readRegister(uint8_t reg);
    void readXZAccelData(int16_t* accelX, int16_t* accelZ);

  public:
    // Constructor
    IMU_Handler(int csPin);

    // Public methods
    bool begin();
    void update(float dt);
    float getRoll();           // Calculates roll from filtered acceleration
    float getFilteredAccelX(); // Returns filtered X acceleration
    float getFilteredAccelZ(); // Returns filtered Z acceleration
    void setFilterFrequency(float cutoffFreq, float sampleFreq);
};

#endif
```

Appendix G: Arduino state machine diagram

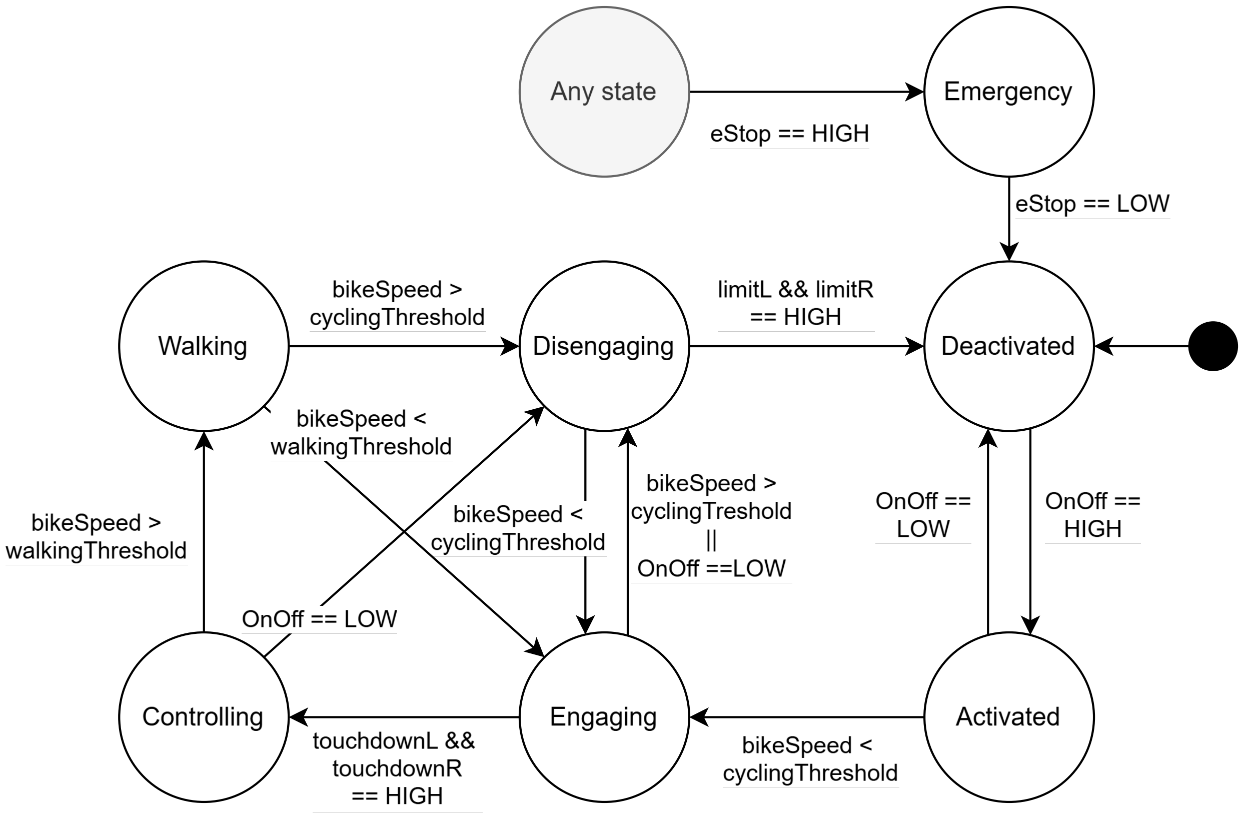

Any state
Emergency
eStop == HIGH
eStop == LOW
Walking
Disengaging
Deactivated
bikeSpeed > cyclingThreshold
limitL && limitR == HIGH
bikeSpeed < walkingThreshold
bikeSpeed > walkingThreshold
bikeSpeed < cyclingThreshold
bikeSpeed > cyclingTreshold || OnOff ==LOW
OnOff == LOW
OnOff == HIGH
OnOff == LOW
Controlling
Engaging
Activated
touchdownL && touchdownR == HIGH
bikeSpeed < cyclingThreshold

Fig. 23: State diagram of the Arduino state machine. The circles represent the states, and the arrows show the transitions between states and their conditions. The black dot represents the start of the system when it is powered up or reset. The ”Any state” circle replaces arrows running from all states to the Emergency state for clarity.

Appendix H: Block diagram

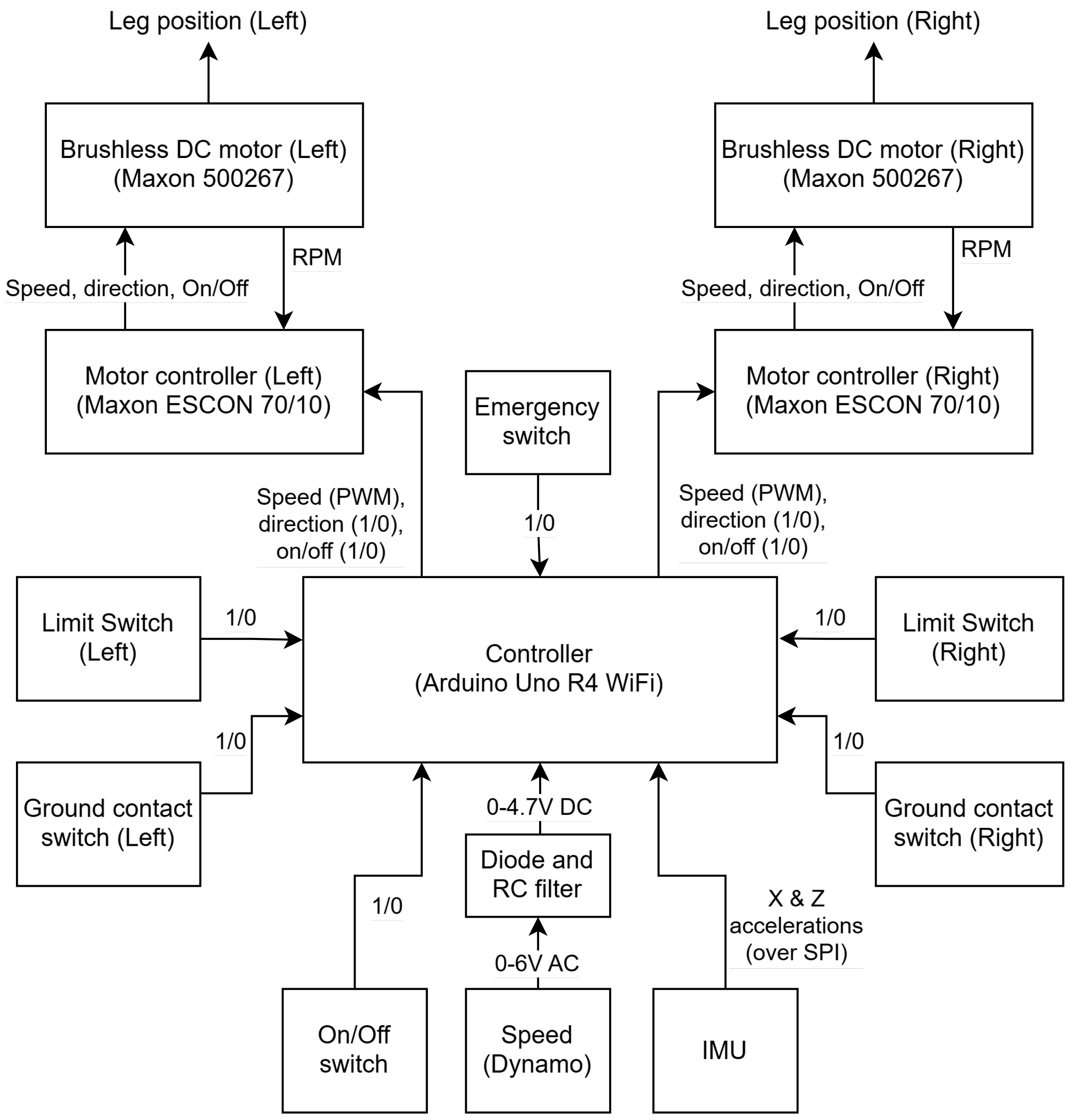

Leg position (Left)
Leg position (Right)
Brushless DC motor (Left) (Maxon 500267)
Brushless DC motor (Right) (Maxon 500267)
Speed, direction, On/Off
RPM
Speed, direction, On/Off
RPM
Motor controller (Left) (Maxon ESCON 70/10)
Emergency switch
Motor controller (Right) (Maxon ESCON 70/10)
Speed (PWM), direction (1/0), on/off (1/0)
1/0
Speed (PWM), direction (1/0), on/off (1/0)
Limit Switch (Left)
1/0
Controller (Arduino Uno R4 WiFi)
1/0
Limit Switch (Right)
1/0
1/0
Ground contact switch (Left)
0-4.7V DC
Ground contact switch (Right)
Diode and RC filter
1/0
X & Z accelerations (over SPI)
0-6V AC
On/Off switch
Speed (Dynamo)
IMU

Fig. 24: Block diagram shows the functioning of the system and signals between components. The blocks are the components of the system, and the lines represent the logic signals being sent between the components. For clarity, the components providing power to the system are left out, and only the logic signals are considered.

Appendix I: Maneuverability experiment results

TABLE XIII: Results of the maneuverability test for the 90-degree turn. All numbers are the measured times in seconds. The table is color-scaled to indicate which maneuvers were faster. Green colored cells indicate maneuvers that were faster than average, while red colored cells indicate maneuvers that were slower than average. The colored cells show each timed maneuver. The white cells with numbers indicate the average time colored cells above them. The white cells with percentages indicate the percentage increase in time between the system on and off.

Standing				Sitting			
System off		System on		System off		System on	
Right	Left	Right	Left	Right	Left	Right	Left
4.60	4.77	4.60	4.97	4.76	4.96	5.53	4.86
4.20	4.63	4.94	4.62	4.70	4.81	5.04	5.41
4.03	4.61	4.51	4.75	4.26	4.91	5.28	5.29
4.10	4.67	4.58	4.95	4.79	4.80	5.19	5.57
4.70	4.64	5.05	4.85	4.32	4.83	5.17	5.54
4.33	4.66	4.74	4.83	4.57	4.86	5.24	5.33
4.50		4.78		4.71		5.29	
6.38%				12.18%			
9.28%							

TABLE XIV: Results of the maneuverability test for the 180-degree three-point-turn. All numbers are the measured times in seconds. The table is color-scaled to indicate which maneuvers were faster. Green colored cells indicate maneuvers that were faster than average, while red colored cells indicate maneuvers that were slower than average. The colored cells show each timed maneuver. The white cells with numbers indicate the average time of the colored cells above them. The white cells with percentages indicate the percentage increase in time between the system on and off.

Standing				Sitting			
System off		System on		System off		System on	
Right	Left	Right	Left	Right	Left	Right	Left
8.66	8.48	9.43	9.65	10.16	9.46	10.78	10.63
8.39	8.90	8.71	9.57	10.03	10.30	11.09	11.59
8.70	8.65	9.09	9.41	9.73	9.00	10.32	9.03
8.51	9.40	9.26	9.48	10.40	10.22	11.53	11.24
8.34	8.33	9.14	9.15	10.22	9.36	9.98	9.31
8.52	8.75	9.13	9.45	10.11	9.67	10.74	10.36
8.64		9.29		9.89		10.55	
7.56%				6.69%			
7.13%							