Table of Contents

resistance to a measurable output voltage, which is then pre- and post-processed by an Arduino Due which sends the final result to the on-board vehicle network via CAN.

Architecture

This section describes the architecture of the proposed solution and how it has been integrated and implemented in order to solve the problem in question. As already briefly touched upon in the introduction, the implemented system consists of strain gauges, a microcontroller and an electrical circuit including a load cell amplifier.

Strain Gauge

Strain gauges (as shown in fig. 2) are a well established and popular solution for measuring the strain on materials of many kinds. Their sheer accuracy and extremely low costs led to a

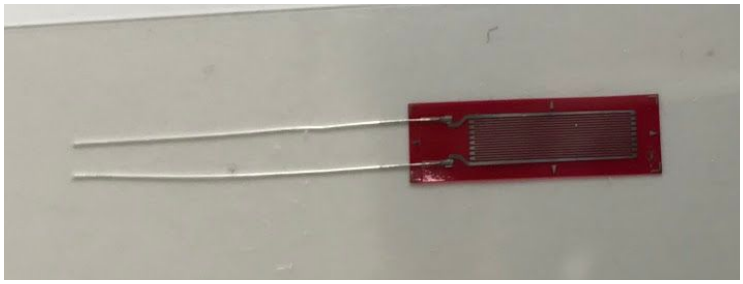

Figure 2: Strain Gauge

widespread adoption across the board. The most famous example of an implementation of strain gauges is in a normal over the counter scale for weight measurement. The combination of four sensors working together on each corner of the device can produce highly accurate measurement outputs.

Strain gauges are glued to any given object using an appropriate glue like *cyanoacrylate*. When a force is applied to an object, it causes the the metallic foil pattern in the strain gauge to be deformed in the same way as the object. This leads to a change in electrical resistance of the sensor, which can be transformed to a measurable output voltage using a *Wheatstone Bridge*, which will be discussed in the next section.

Wheatstone Bridge

A wheatstone bridge works as follows: There are four resistors in Figure 3 below. The resistance is known for R_1, R_2, and R_3 but not for R_X. The resistance of R_1, R_2 and R_3 is equal to each other. Assuming that R_X is a strain gauge and its resistance is also equal to the other three, then the bridge is balanced and the output voltage V_G is therefore equal to zero.

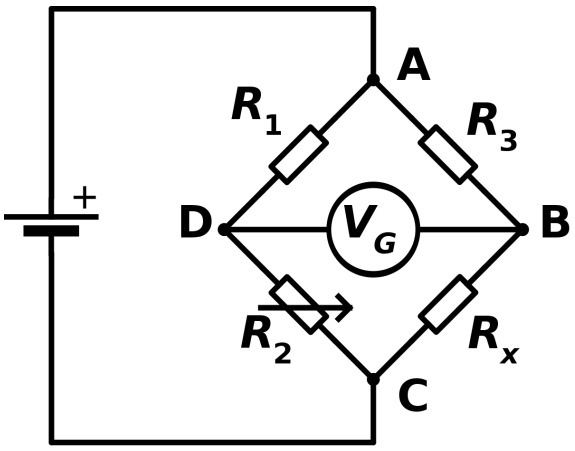

Figure 3: Wheatstone Bridge Circuit Diagram[6]

However, as soon as the resistance of the strain gauge positioned at R_X increases or decreases, it will lead V_G to also increase or decrease, which can then be forwarded to a *load cell amplifier*. Wheatstone bridges can be configured in three different ways: *Quarter Bridge, Half Bridge* and *Full Bridge*. Quarter corresponds to the circuit depicted in Figure 3 above. Half means that two out of the four possible resistors are replaced by strain gauges and in a full bridge, no resistor remains constant. The three different kinds of configurations have different properties and benefits. Although a quarter bridge might be simplest to set up, it has a major disadvantage, which is its lack of a secondary resistor that is variable and therefore allows for the cancellation of temperature variations. I.e. assuming that the temperature around the object, which the sensor is applied to varies, the resistance of the strain gauge will by definition also change. This factor has to be accounted for by the introduction of a half bridge configuration. The addition of a secondary sensor means that both sensors' resistance will vary equally much upon a change in temperature, meaning that they fully cancel each other out. The resulting output voltage will continue to remain accurate.[7] Although a full-bridge configuration sounds

[6] Wikipedia contributors. "Wheatstone bridge." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 7 Aug. 2018. Web. 11 Aug. 2018.
[7] Hoffmann, Karl. Applying the wheatstone bridge circuit. Germany: HBM, 1974.

promising, it only allows for either measuring forces along the horizontal or vertical axis, not both. Therefore the proposed solution is to use a half-bridge configuration, that is able to cancel out temperature changes, as well as measure forces in horizontal and vertical axes.
However, another decision had to be made, regarding the two possible configurations within the category of half bridges, as shown in Figure 4:

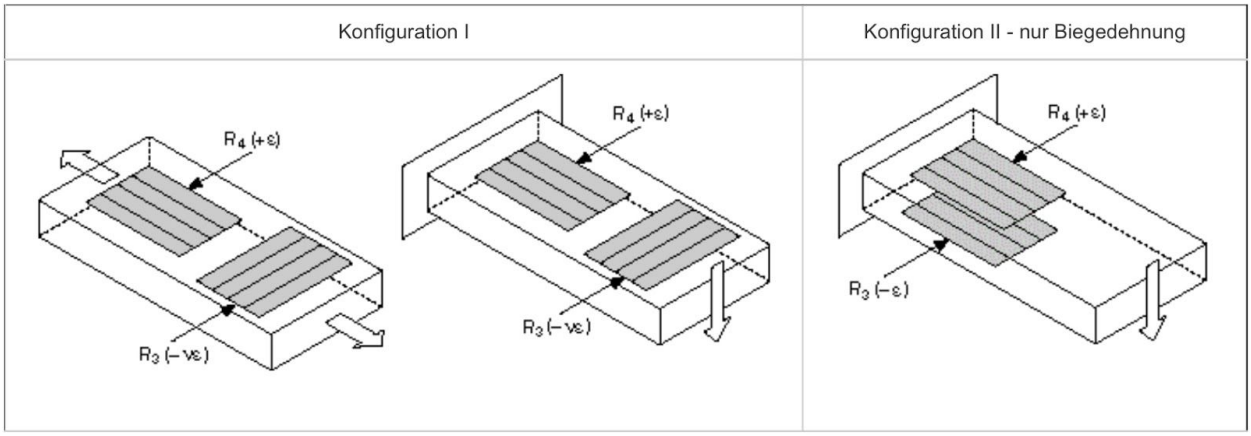

Figure 4: Wheatstone Half-Bridge configurations.[6]

The figure above shows that half-bridges can as well be configured to measure solely the forces along the vertical axis *(Configuration I)* or both *(Configuration II)*. The implementation was done using *Configuration I* in order to achieve the overall best possible results.

Load Cell Amplifier

The strain gauge sensors combined in a wheatstone bridge configuration allow for measurements in voltage difference, however the output signal is extremely small in magnitude

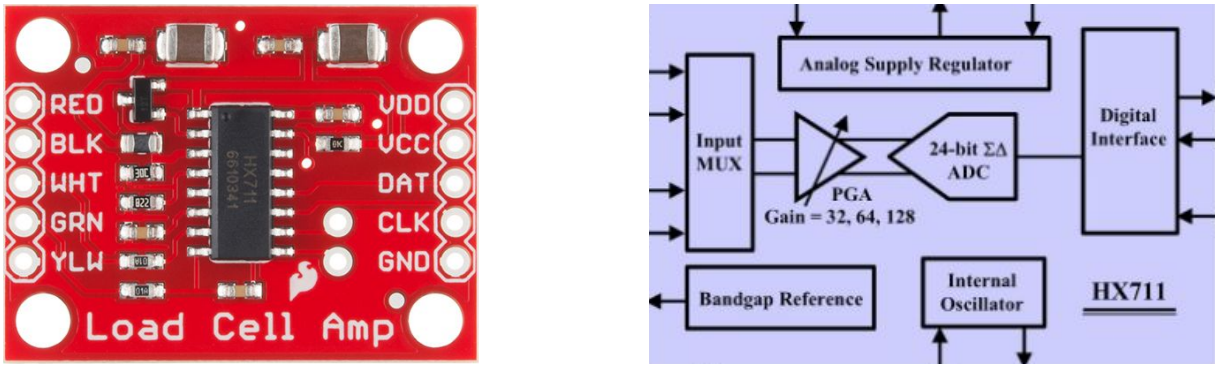

Figure 4: Load Cell Amplifier and Circuit

and therefore not measurable by a normal microcontroller, such as the Arduino Due, that was used in this solution. In order to make it work, the analog signal needs to be conditioned and

filtered first before being forwarded to an analog-to-digital converter. This is the reason why there is a need for a so called *Load Cell Amplifier*[8], as shown in Figure 4 and 5 above. Once the conditioning, filtering and conversion from analog to digital is completed, the output signal is much more accurate than prior to processing and can be forwarded to a microcontroller, which does analysation and forwarding to the on-board vehicle network.

Microcontroller and Shield

There are two obvious choices for popularly known microcontrollers: *Raspberry Pi* and *Arduino*. Research concluded that both systems are well enough suited and could potentially be used for the task, however the *Arduino* has an advantage over the *Raspberry Pi* in terms of the implementation of CAN bus and the available add-on shields. Namely the *Robot Dyn Mega*

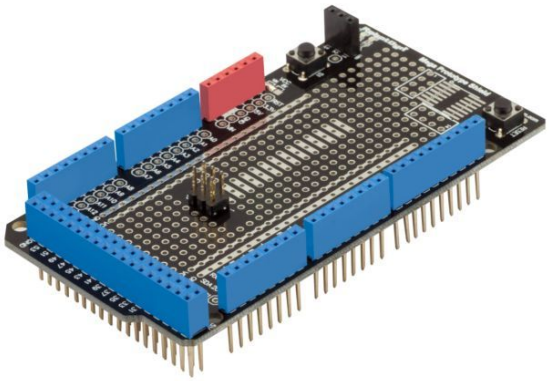

Figure 5: Robot Dyn Mega Prototype Shield[9]

Prototype Shield[9] (as shown in Figure 5 above) is particularly well suited for the problem at hand, as it gives room for plenty of custom made circuitry like the wheatstone bridge and has built-in soldering pads to easily solder the CAN bus transceiver, which will be discussed in the next subsection. Furthermore, the *Arduino Due* has built in pins for *CanTx* and *CanRx*, making the implementation significantly easier than with a *Raspberry Pi*.

CAN Bus Transceiver

Although the above mentioned prototype-shield has the available soldering pads in order to connect a CAN bus transceiver easily, it does not come with a pre-built transceiver or dub-9 port, which is the most commonly used connector port for CAN. Instead, the physical

[8] "*Introduction to Load Cell Conditioning Circuits." PREDICTABLE DESIGNS, 25 June 2018, predictabledesigns.com/introduction-to-load-cell-conditioning-circuits/.*
[9] "Prototype Shield Mega for Arduino Mega." RobotDyn, robotdyn.com/prototype-shield-mega-for-arduino-mega.html.

implementation of the CAN transceiver and its physical data link are completely up to the researchers and therefore leave room for possible modifications for future work. The CAN bus transceiver used in this system is the *MCP2551*[10] chip. "The MCP2551 is a high-speed CAN transceiver, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus."[10] The system does not need to use a lot of bandwidth on the data layer, however the chip does support speeds of up to 1Mb/s.

The Complete Architecture

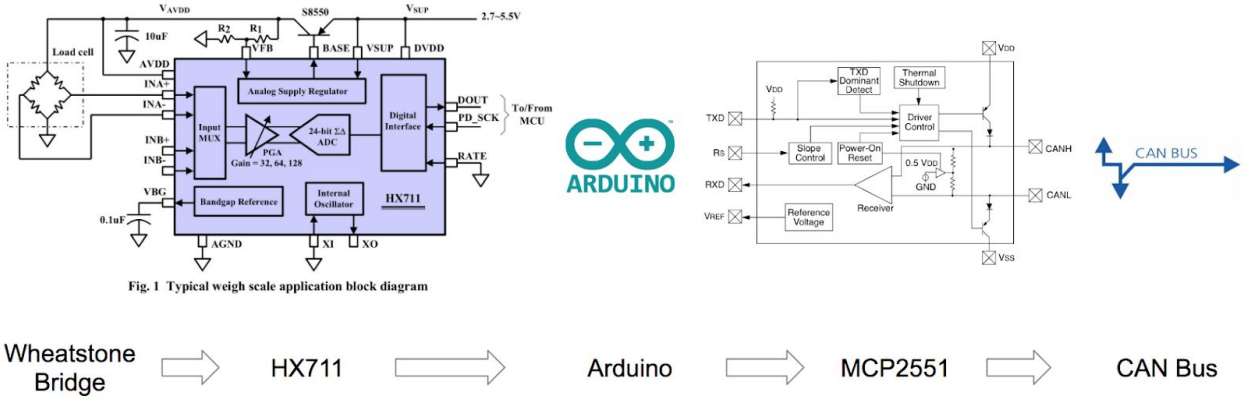

Fig. 1 Typical weigh scale application block diagram

Figure 6: Complete Architecture

Figure 6 shows the complete architecture of the hands on/off detection system implemented in this report. The signal originates in the *strain gauges*, which are part of the *wheatstone bridge* shown at the very left of the circuitry. The analog voltage measured at the output of the *wheatstone bridge* is propagated forward to the input pins of the *Load Cell Amplifier*, which has a *programmable gain amplifier (PGA)* that amplifies the signal from a low to higher magnitude. The amplified signal is then converted from *analogue-to-digital* and propagated forward to the *Arduino Due*. The code running on the microcontroller is discussed in the section *Algorithmic Solution* following. Once the software has made a new detection regarding the user's hands being put on or taken off the wheel, it makes use of the *MCP2511* chip that provides an interface between the digital and physical layers of the *CAN* protocol. The *CAN* message is now ready to be received by the *DAS* or any other components in the vehicle that have subscribed to the message with a given *CAN-ID*.

[10] "MCP2551." Microchip Technology Inc, www.microchip.com/wwwproducts/en/en010405.

Algorithmic Solution

In order to detect the hands status, the software utilizes several indications:

1. Pressure increase/decrease indication, derived from the last 2 readings.
2. Base-line indication, the base line is calculated over certain period of time (currently 5 seconds) during this time, the simulator must not be touched.
3. Special cases: other algorithms to detect special cases (like wheel turning)

Pressure increase/decrease indication

The Pressure increase/decrease indication is calculated by analyzing the difference between every two consecutive readings: *diff* = |*last_reading - current_reading*|
When the pressure on the steering wheel increases, *diff* is a negative value, and vice versa. The software defines a threshold, when *diff* is smaller than the threshold for *hands_on* or bigger than the threshold for *hands_off* then the software has detected a change of state.

Base-line indication

The software calculates a baseline, during which the simulator should not be touched. The baseline is the average of the all the readings during a certain period of time.
On each iteration, the software checks the if the current reading is above the baseline. If it is, the hands status is changed to *hands_off*.
This measure is taken in order to not rely solely on the *diff* but to also have an indication that relates to the reading value for the state.

Special cases

The *diff* and *baseline* are sufficient in order to detect the state most of the time, but there are special cases in which the detection does not perform well enough. For example, when the wheel spins for a long period of them, or when the hands are in “12 o’clock” or “6 o’clock” positions, the pressure on the wheel is considered to be too low in magnitude.
In order to deal with those scenarios, two different software approaches have been implemented, but the detection is still insufficient for those special cases:

1. Calculating the average of the last several readings: Trying to extract information from the average of the last seven to twelve readings did not seem to give an added value regarding the state in special cases.
2. Calculating the sign: Analysation of the output shows that during a long turn, the readings tends to be less random, and tend to either all increase or decrease and so the *diff*-sign tends to be all positive or negative.

Applying the methods mentioned above helped dealing with the case of a long turn, but it remains imperfect. For more information about the software, please feel do not hesitate to take a look at the source code or the *Readme*, which are both referenced in the *Appendix*.

Results

The integrated hardware has proven to be fast and sensitive enough in order to adequately analyse the signals from the strain gauge such that it can detect the current state of the system. When operating on the DCAITI simulator, without *Serial output*, the system can read and analyze a signal in ca. 180 millisecond (frequency of 5 to 6 readings/s) and it is sensitive enough to make a detection between the state of *hands_on* or *hands_off*.
When the driver simply puts his hands on or takes them off the wheel, the detection is good, as well as during normal driving with small wheel adjustments. There still is a need to improve the

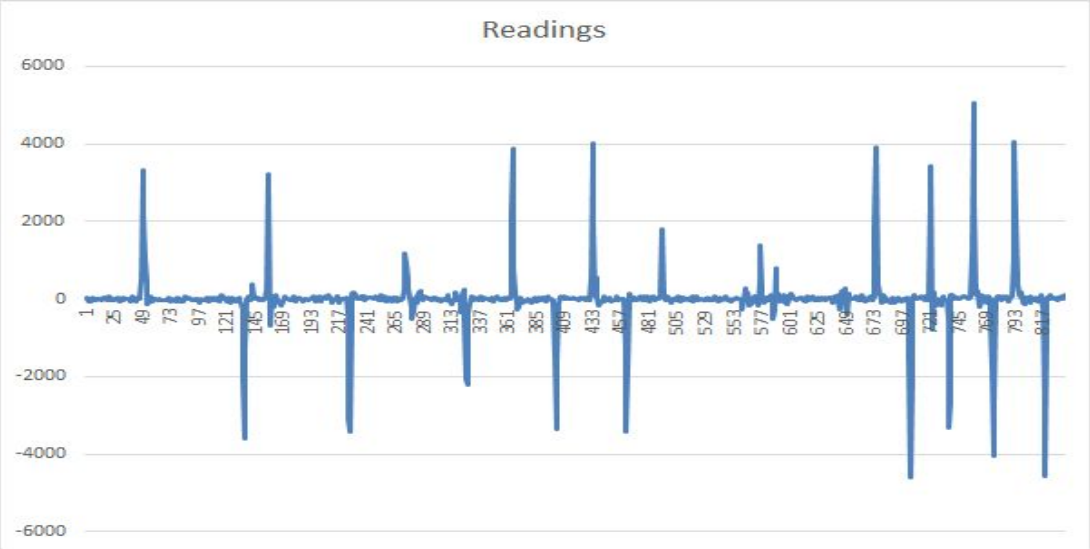

Figure 7: Visualisation of Raw Data

algorithm in order to detect the state during special cases, such as when the hands are in “12 o’clock” or “6 o’clock” positions, or during rather long wheel movements.

The recorded raw data has 2462 rows, which gives more than enough insights for simple algorithm to perform well. However, a more sophisticated solution would require large amounts of labelled data sets that can provide training for a machine learning algorithm.

Conclusions

The architectural hardware solution has shown promising results. The overall costs of both monetarial investments and time input, were kept low and affordable, which meets the first of the two specific requirements for this project mentioned in the *Introduction*, namely that the solution had to be cheap. Furthermore, the second requirement of not adding any hardware components directly onto the wheel was also fulfilled. Strain gauges are a viable option for the simulator to differentiate between DAS- and user-controlled steering. In the worst case, the system is able to read measurements five times per second, which could fulfill real-time requirements for the simulator use case, but do not simulate a production *hands on* system The software has shown its capabilities to detect and send a clear indication for simple state detection of whether the users hands are on or off the wheel and is functioning well in simple driving scenarios, but it is not accurate enough in more complex scenarios. Possible limitations are given in the *Discussions* section below.

Discussions

Although the solution worked in a controlled environment, it lacks the implementation of a more sophisticated algorithm, perhaps using vast amounts of data in order to utilise a more machine learning driven approach, such that it works as well in a less controlled environment and at some point potentially in a real world vehicle. Furthermore, the reading frequency of the solution is inadequate for real-time requirements in real-world vehicles and therefore might need to improve in order to be able to simulate production *hands on* system. The software implementation is relatively simple and therefore gives room for possible improvements: Given a much larger data set recorded from various simulated driving situations, a more sophisticated

algorithm could be implemented, which relies on machine learning algorithms. We very much encourage future researchers to take upon the challenge of improving this implementation.

Appendix

This section aims to give an overview of all physical or digital parts and components that have been handed over to the supervisor along with the final version of this report:

1. Hardware
2. Software: *read-output.ino*
3. Readme: *README.txt*
4. Raw Data: *readings.xlsx*