0.0.1 Quadrature Encoder

In order to understand the design of the Quadrature Decoder a small explanation of how a Quadrature Encoder works is needed.
A Quadrature Encoder uses two channels to sense the position and direction of, typically, a rotating disk/shaft or a linear strip. The disk or strip has two paths on it, positioned 90 degresse out of phase with each other, see figure 1 for a rotary encoder and figure 2 for a strip encoder. Statinoary sensores are typically placed on top of the encoder, so when a track moves in relation to a sensor, it outputs a logic high or low output signal depending on what part of the track is visible to the sensor. An encoder has two output signals, one for each channel, typically called A and B, see figure 3 for a representation of those for both encoder types. This two signals, A and B, is what allows the encoder to determine the position and direction of the encoder

Placeres sensorere oven pp encoder eller er det en del af d færdige encode

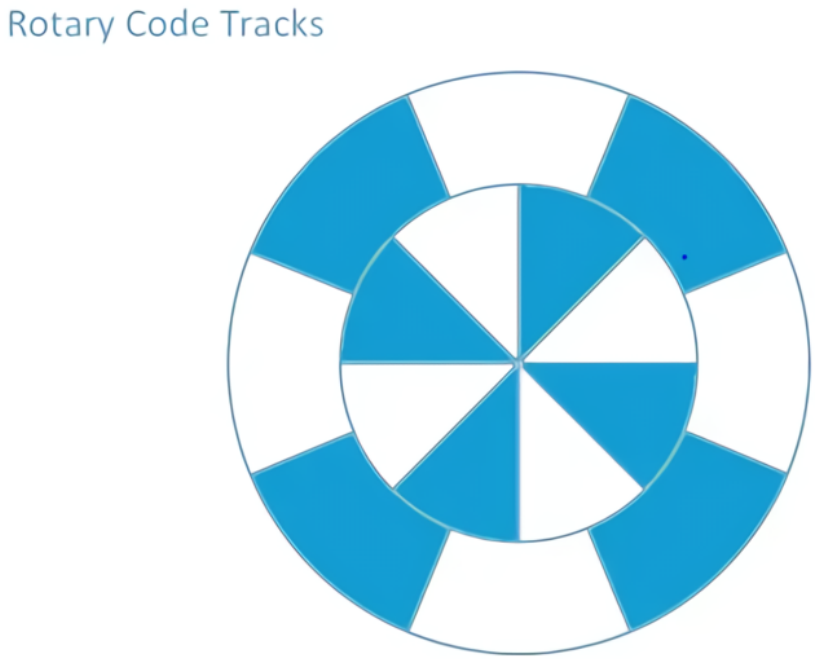

Figure 1: *Tracks of a rotary encoder.*

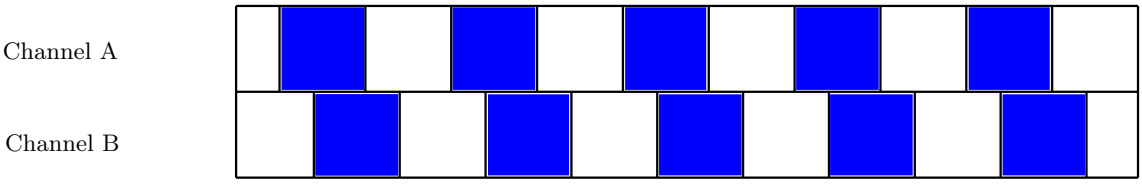

Figure 2: *Tracks of a Linear encoder*

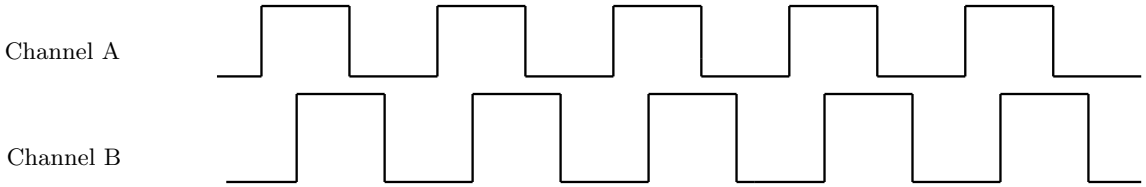

Figure 3: *The logic output of an encoder*

0.0.2 Decoder principle

The founding principle of a Quadratur decoder relies on decoding the relationship between the two outputs and deduce a position change. The idea is to observe both encoder outputs. By counting the transitions from high to low and low to high on just one of the outputs, it can be determined how far the encoder has rotated. However, by adding the second output, the direction of the encoder can be computed as well as the resolution to the position. The encoder used in the project is a rotary encoder with a resolution of 360[1], which means that the Quadrature Decoder will count 360 "ticks" per shaft turn. This resolution and rotation can not be mapped directly to the system itself as its motors are geared. The motors do three full rotations of their shafts before the system itself has completed a full rotation. Hence the resolution for the system is 1080 ticks. This gives a $\frac{1}{3}$ of a degrees position precision, allowing a very precice measurement.

0.0.3 Statemachine

To use the transitions between encoder output A and output B, a truth table of every possible combination of the two singals must be computed. It is necessary to include a third signal called reset to allow for resetting of the position counter when a new home position is desired. The truth table computed from all three signals is shown in figure 4.

A_prev	A_new	B_prev	B_new	Reset	Direction	Position
0	1	0	0	1	Forward	+ 1
1	1	0	1	1	Forward	+ 1
1	0	1	1	1	Forward	+ 1
0	0	1	0	1	Forward	+ 1
0	0	0	1	1	Backward	- 1
1	0	0	0	1	Backward	- 1
0	1	1	1	1	Backward	- 1
1	1	1	0	1	Backward	- 1
x	x	x	x	0	No change	0

Figure 4: *This figure shows the truth table for the Quadrature decoder*

The truth table shows nine possible scenarios that can occur from tracking the transistions of the outputs. Four of these results in a forward direction and the counter will increment, four scenarios results in a backwards direction and the counter will decrement. The last scenario occurs if the reset flag is set low[2], and the counter is set to zero regardless of what it was on before. From these observations a state machine containing 6 states has

[1] See datasheet x

[2] Reset is implemented as active low, as can be observed in the truth table 4.

been computed. The states are: AB_low with the gray code 001[3], AB_high with gray code 111, A_high with gray code 101, B_high with gray code 011, undefined and reset $xx0$. The undefined state is purely for debugging purposes used if an error with the signal A and B ocurs. A potential error could be if the state is currently AB_low the gray code can not be 111 in the next instance. This would mean that both signals changed at the same time. The state machine diagram is shown below in figure **??**.
The decoder works by first initialalizing and go to a state depending on the level of signal A and B. The decoder will then use the debounced signals to determine the next state. If the starting state is AB_low, there is three correct outcomes: the reset signal goes low and the counter will go to zero, the signal A goes high and the state changes to A_high or the last possible outcome, that does not give an error, is B_high. By looking at the truth table from the above figure 4, it can be observed that if the state goes from AB_low to A_high, the direction will be forward and the counter will increment where the counter would decrement and direction set to backwards if the next state is B_high.
The counter keeps track of the position of the encoder. The counter counts position zero as the angle the system was in, as the encoder was initialized. It is thereforth considered useful to add a reset function, so a new home position can be added as desired. The Statemachine approach has been implemented and the process described in section **??**

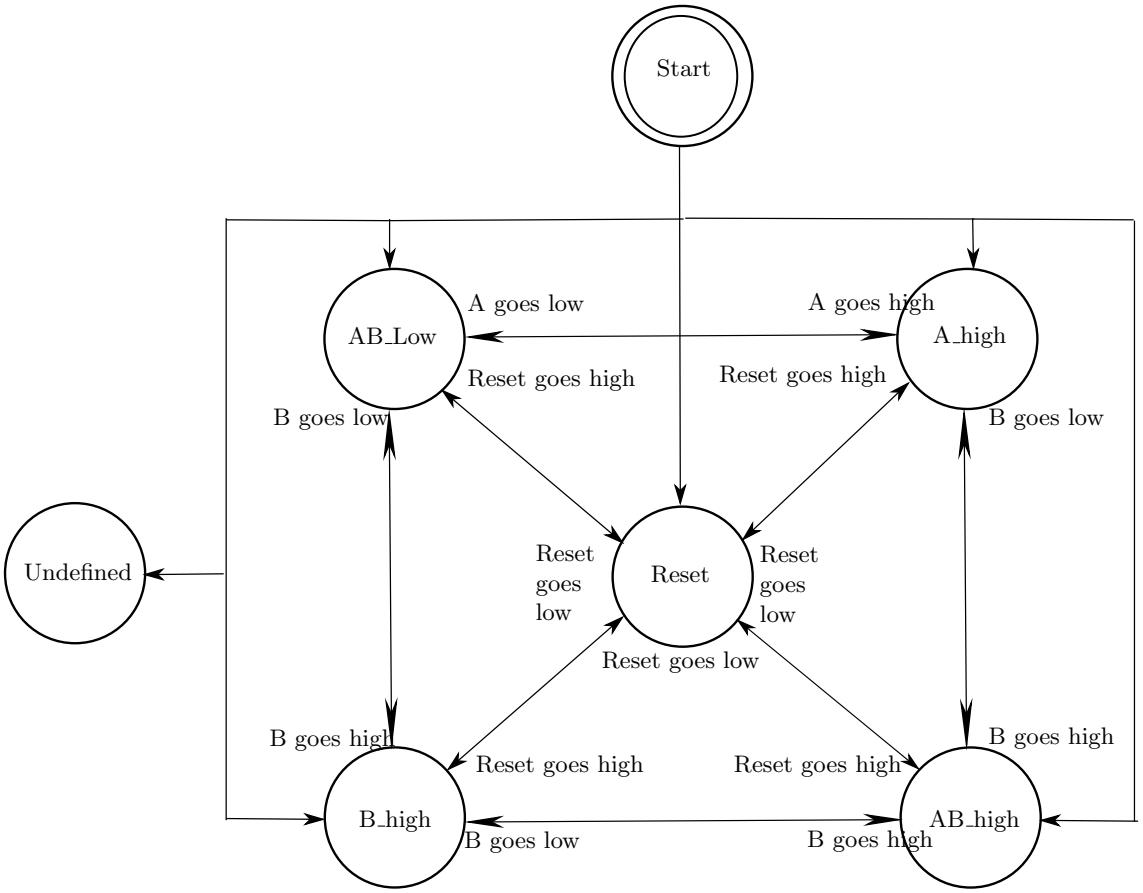

Figure 5

As mentioned above, a reset is needed for the application. This allows for a new home

[3]The first digit is signal A, second is signal B and thrid is the reset signal

position to be set as desired. The state machine does not make use of a clock singal so the reset is required to be a asyncron reset to avoid adding more complications to the application.

0.0.4 Velocity measure

The decoder is not only responsible for computing the position of the system but also the velocity. The simplest way to estimate the velocity is by diving the change in position with the change in time. The most classic options are either to keep the Δt fixed, measure the position between the interval and then take the difference between x(n) and x(n-1) as shown in equation 0.1. The other is to keep the position fixed and compute how long it takes to reach the fixed Δx like in equation 0.2.

$$v(n) = \frac{x(n) - x(n-1)}{T} \tag{0.1}$$

$$v(n) = \frac{X}{t(n) - t(n-1)} \tag{0.2}$$

The second approach 0.2 is considered the most reliable at slow speeds, which arguably the system does fall under. However, this approach depends on the output to be a "fixed interval pulse train". The system does not fit this criteria, because it generates different frequency waves depending on the supplied PWM value[4].

The first appraoch using equation 0.1 is not hindred by the differnce in frequency it has been chosen.

As the chosen approach requires a fixed time interval one such must be decided. The issue to take into account here is to make sure the interval is neither too short or too long. A too short interval might yield scenario where the velocity seems to be zero even if the system does move. However, on the opposite side, a too long interval could yield a scenario that gives a wrong velocity, if the system changes velocity a lot. If the interval is say 1 second, the velocity at the first $10ms$ be 50 ticks, but it slows down considerably at the $20ms$ mark and hardly moves at $70ms$ mark. This will give an average velocity that might not be as useful and downright misleading as it shows the system to have travelled with a low speed instead of a showing that it was fast at first but slow towards the end. In this case it is important to the system speed up and slow down, so a low interval time of $10ms$ has been chosen.

Forstår hvad d mener men det er mudret

[4] https://www.embeddedrelated.com/showarticle/158.php