Developing a Free-Falling Robotic Cat With Righting Reflex

Jørgen Drangevåg

Supervisors
Kristian Muri Knausgård
Morten Kjeld Ebbesen

University of Agder, 2020
Faculty of Engineering and Science
Department of Engineering Sciences

Abstract

Changing the orientation of a body in free fall is a useful skill, found in cats. The applications range from zero-gravity in space to damage control of a falling object. This thesis aims to create a robot cat able to always land on its feet. The robot cat is based on the rotating cylinder concept suggested by Kane and Shere, where the front and the rear of the cat each consists of a larger outer cylinder rotating around a smaller inner cylinder, connected by a revolute joint. Each half of the robot cat consists of a PE100 outer pipe, an aluminum inner tube, a servo motor, an incremental optical encoder, and a power supply, connected by 3D printable parts. A Pixhawk 4 Mini is used as the control unit, and an Arduino nano and a quadrature counter are used to read encoder data.

The results involve concept generation and selection. Suitable components are identified and selected. A design is proposed, implementing the components to the concept. Communication between devices is achieved using SPI and UART. Hobby servo motors are modified, calibrated and linearized using the least-squares approximation. System identification is performed using blackbox frequency analysis, and a state-space velocity control system is suggested. This provides a good foundation for further development.

Contents

Chapter 1

Introduction

Background and motivation The physics of free-falling cats has been studied with scientific methods since the late 19th century. How can a free-falling body, starting in an upsidedown position, change orientation and "always" land in the right orientation with the legs down without violating the law of conservation of angular momentum? A solution was first described by Kane, T R; Scher, M P. (1969), in the paper "A dynamical explanation of the falling cat phenomenon", and is now a well known example of multibody dynamics. An optimal solution for controlling the cat's motions exist: Ge, Xin-sheng; Chen, Li-qun (2007), "Optimal control of non-holonomic motion planning for an free-falling cat".
The robot-cat shall be low-cost and easy to produce, for example using 3D-printing. An ideal solution will be simple and elegant. And potentially feature "skins", such that the "core" of the robot cat is close to the cylindrical model shown in the illustrations, but a more cat-like outer skin could be mounted for visualization purposes and robustness testing (for example providing different weight distribution). Outline of the task:

- Litterature study. Find state of the art algorithms for controlling cat motion.
- Concept design and evaluation.
- System design, including mechanical design, computing platform, sensors and actuators.
- Multibody dynamics simulation. Model the cat using a Modelica tool (Simulation X?).
- Controller development.
- Closed-loop simulation: use the Modelica model, for example using FMI, and control it using control algorithms implemented in Simulink or another rapid prototyping tool.
- Build a robotic cat prototype.
- Perform experiments in Motion Lab and evaluate results. (A Qualisys motion capture system provides accurate state estimates for cat position and orientation).
- Optional: Optimal control of cat motions, add inertial measurements on-board sensors. Land on a moving platform? Machine learning?

Chapter 2

Theory

2.1 Rotary position sensors

Sensors have many applications, measure a system output for feedback control, measure a system input for feedforward control, condition monitoring, diagnosis, or measure output for system identification [16]. Choosing the right sensor for an application is an important part of instrumenting a mechatronic system. In this section, different sensors used to measure angular position are introduced.

2.1.1 Potentiometer

The potentiometer is an analog sensor used to measure displacement [16]. The sensor consists of a high-resistive material with resistance that is proportional to its length, a slider arm who can move along the path of the resistive material, and a voltage source. The sensor has three terminals, one for voltage supply, one for ground, and one for the slider arm. By applying a constant voltage, v_{ref} $[V]$, to the resistance, the displacement of the slider arm proportional to the voltage difference between the slider arm, v_0 $[V]$, and ground.

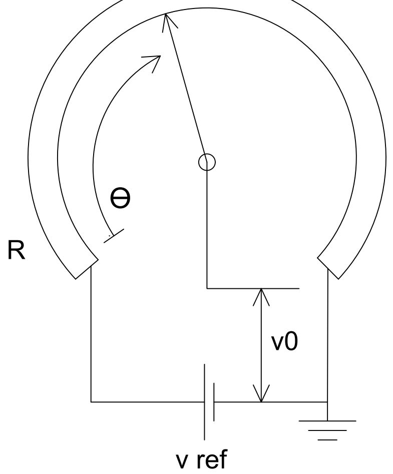


Figure 2.1: Angular potentiometer

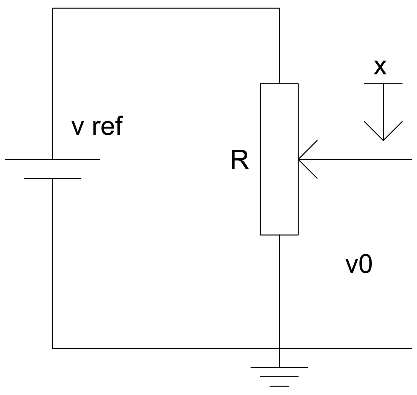


Figure 2.2: Translational approximation

$$v_0 = kx \tag{2.1}$$

Where v_0 $[V]$ is the output voltage, k $[\frac{\Omega}{m}]$ is a constant proportional to the total resistance of the potentiometer and x is the slider displacement.

$$v_0 = k\theta \tag{2.2}$$

Where v_0 $[V]$ is the output voltage, k $[\frac{\Omega}{rad}]$ is a constant proportional to the total resistance of the potentiometer and Θ is the slider displacement.

As shown in figures 2.1 and 2.2 the potentiometer can measure both linear and rotational displacement.

The potentiometer is inexpensive, produces high voltage output signals which do not require amplification in most cases[16]. There are, however, a few drawbacks: the friction force required to move the slider arm affects the reading, variation in voltage supply causes an measurement error, energy is dissipated through heat and friction, the slider arm wears the resistive material, limited resolution and performs badly at high-frequencies or high transients.

2.1.2 Rotary Encoders

Rotary encoders are digital sensors used for measuring angular position and velocity[16]. The encoder generates, represents, and transmits digital data. This removes the quantization error associated with sampling an analog signal. The digital signal is also less sensitive to noise than an analog signal. This is because the data is transmitted as a digital word, which has two identifiable states, rather than a sensitive voltage signal. Digital measuring is, therefore, preferable to analog.

There are four techniques used to measure the angle in encoders: optical (light sensors), mechanical (electrical conducting), magnetic saturation (reluctance) and proximity sensor [16]. All types generate the same type of output signal. Optical encoders are considered the most cost-effective and should be used except for particular circumstances where they are not suited, such as high temperatures [16].

Based on the method used to interpret the generated signals, encoders are generally classified into two categories: incremental encoders and absolute encoders.

Incremental Optical Encoder

Incremental optical encoders use a code disc, a light source, and a light sensor to measure the angle, as illustrated in figure 2.3. The code disc has one or more tracks with transparent windows. The light source emits light onto the code disc, depending on the position of the code wheel, the light is either blocked or passes through to the sensor. When the sensor is exposed to light, it generates a voltage signal. Rotating the wheel thus generates a series of voltage pulses used to detect the increments. The code wheel, therefore, can not detect the orientation of the disc directly, only incremental changes. The voltage pulses are converted to a digital signal whos duration can be

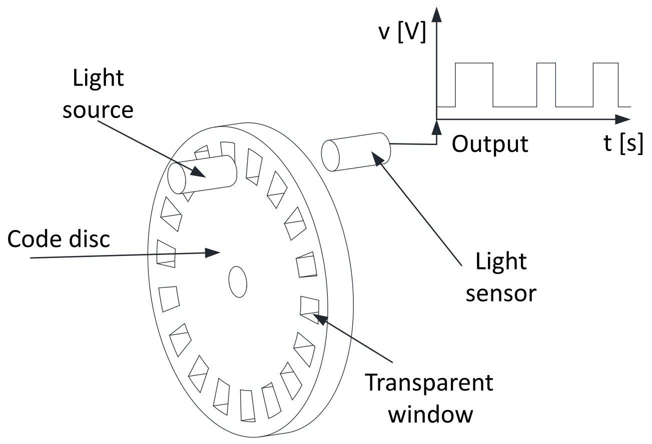


Figure 2.3: Incremental encoder [16]

used to calculate the velocity of the code wheel.

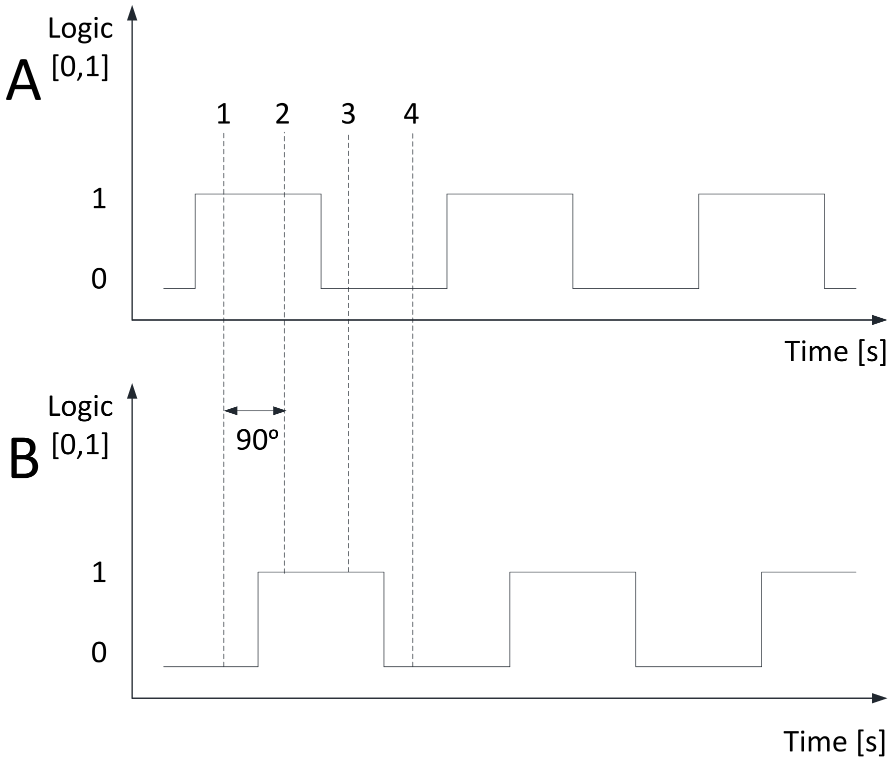


Figure 2.4: Phase difference between channels A and B [16]

The incremental encoders usually have two or three channels. When the encoder has two channels, the channels are then called A and B. The code wheel can either have one additional track, or the extra channel can use the same track. Either way, the second channel is offset from the first one by a quarter pitch, as shown in figure 2.4 [16]. This increases the number of measurable states of the encoders as moving from one window to the next now has four identifiable states: A and B high, A and B low, A high and B low, and A low and B high. The extra channel is also used to identify the direction of rotation. If B is high when A is falling, the rotation is clockwise, and if B is high when A is rising the rotation is counterclockwise. The third channel is a single-window on its own track used to indicate full rotations called the index pulse.

The physical resolution of an encoder is a measure of the smallest change in angle or velocity the encoder can measure [16]. The resolution is governed by the number of windows the encoder disc has per revolution and the sampling time.

$$\Delta\Theta = \frac{2\pi}{4N} \tag{2.3}$$

where $\Delta\Theta$ $[rad]$ is the position resolution, N $[-]$ is the number of windows and the constant 4 the four different positions between windows [16].

$$\Delta\omega = \frac{2\pi}{4NT} \tag{2.4}$$

Where $\Delta\omega$ $[\frac{rad}{s}]$ velocity resolution, T $[s]$ is the sampling interval [16].

Absolute Encoder

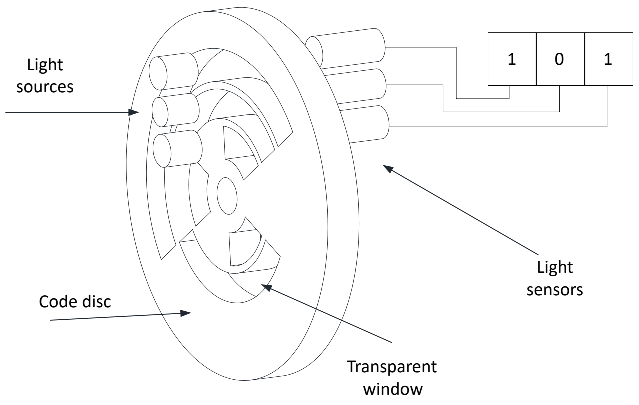


Figure 2.5: Absolute encoder [16]

Much the same way as an incremental optical encoder, the absolute optical encoder consists of a light source, light sensors, and a code disc with tracks of windows for the light to pass through [16]. The light source emits light on one side of the code disc. The light sensors are placed radially on the other side, with one sensor per track, as illustrated in figure 2.5. Each sensor is associated with a bit whos logic level is set by detecting light. Due to the structure of the code disc, each position produces a different combination of active sensors. The encoder can, at all times, directly read the position of the output shaft by evaluating the combination of active sensors. The number of tracks, therefore, determines the resolution of the encoder.

$$\Delta\theta = 2^n \tag{2.5}$$

Where $\Delta\theta$ $[rad]$ is the resolution, and n $[-]$ is the number of tracks.

2.2 Servo motor

Servo motor is a term used to describe a motor that uses a feedback signal and controller to obtain the desired motor output [16]. The feedback signal depends on availability to measure the different outputs. Typical feedback signals are position, velocity, torque, armature current, or field current.

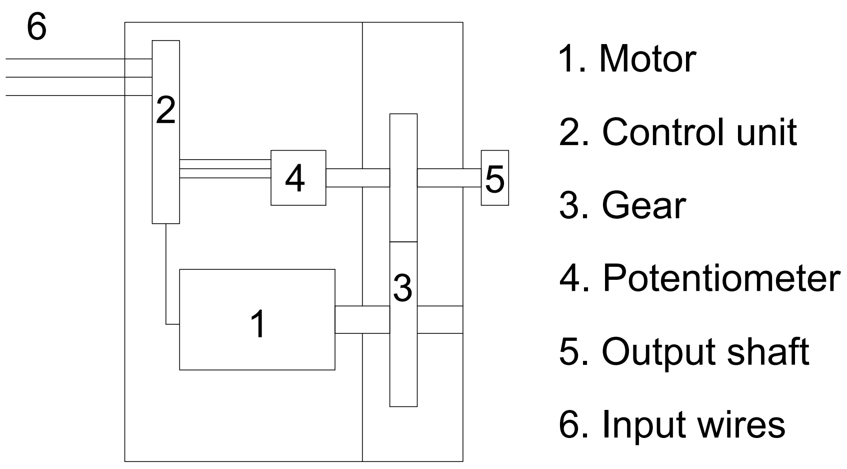


Figure 2.6: Conceptual drawing of a servo motor

Hobby servo motors are small motors with a potentiometer, gearbox, and control unit inside a compact motor case, as shown in figure 2.6 [11]. These motors have three wires; power, ground, and control signal. The motor receives a pulse-width signal, which is converted into a position reference by the control unit. The control unit uses the potentiometer feedback to correct the angle difference from the signal and measured position. The hobby servo motors are usually limited to a rotation of 180 °.

When pulse-width modulation is used to control servo motors a signal, a period of 20 ms, is often used. For the pulse to be considered an input, the pulse must be high between 1-2 ms. This corresponds to an output reference between 0-180°.

2.3 Serial communication

Serial communication is a term used for sending data over one line, one bit at a time. A byte is sent as a stream of bits, each clock pulse one bit is transmitted. Serial communication can be either synchronous or asynchronous [4]. When the communication is synchronous, one device sets the clock speed of the data transmission. The other device uses the clock signal as a reference for when to send and read data. This ensures that the data is read at the same rate that it is being sent. In asynchronous communication, each device is responsible for setting the clock time. Both devices must be, therefore, be configured to communicate at the same rate, known as the baud rate. The baud rate is a measure of how many samplings occur per second and, therefore, how fast data can be transmitted.

The transmission can be either simplex, half-duplex, or full-duplex [9]. When the communication is simplex, data can be sent from one unit to the other, not the other way. In half-duplex data can be sent in both directions, but not at the same time. In full-duplex, data can be in both directions simultaneously.

2.3.1 Serial Peripheral Interface

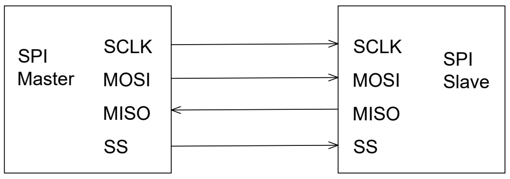


Figure 2.7: Single slave configuration [20]

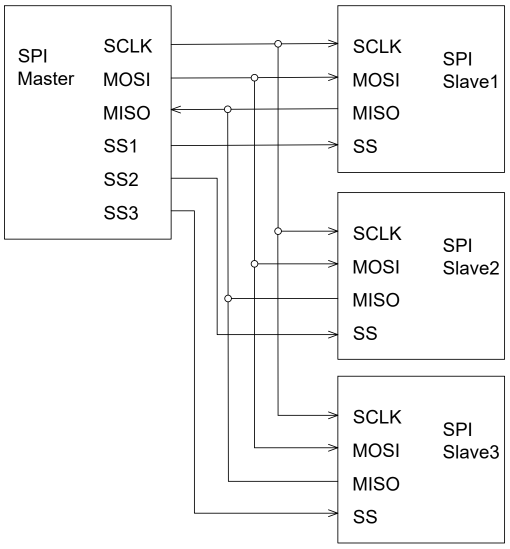


Figure 2.8: Multiple slave configuration [20]

Serial Peripheral Interface Bus (SPI) is a communication protocol used for synchronous communication between devices. The devices can be configured either as a master or a slave. The master communicates with the slave synchronously in full-duplex with speed up to 3 $[Mbit/s]$ [4]. Only one unit in a system can be configured as master, while multiple can be slaves.

SPI requires four pins to communicate; master in slave out (MISO), master out slave in (MOSI), serial clock (SCLK), and slave select (SS) [20]. The pins have different functions depending on the unit configuration. MOSI is used to transmit data for the master and to receive data for the slave. MISO is used to receive data for the master and to transmit data for the slave. SCLK is the clock speed of communication and is set by the master. The clock speed tells the slave which rate data is transferred. SS is used to initiate communication between the master and the slave. As the slaves share the MISO, MOSI, and SCLK pins, the SS pin is a way for the master to choose which slave to communicate with. If multiple slaves are connected to the master, the master must have one SS pin per slave.

Data transmission

When the master wished to communicate with one of the slaves, the master first selects a clock frequency compatible with the slave [20]. The master then pulls the SS pin of the desired slave low. This tells the slave that a transmission is beginning.

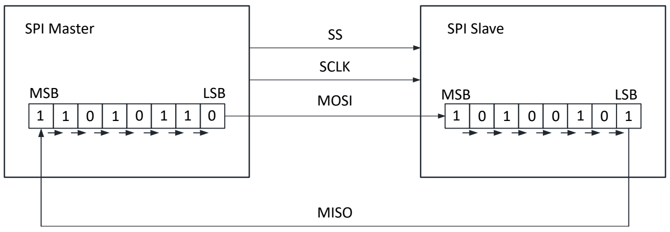


Figure 2.9: Byte transfer SPI[20]

Every clock-cycle the master and the slave each transmits and receives one bit. The communication is done between two shift registers, as shown in figure 2.9. When the transmission begins, the master shifts the least significant bit (LSB) to the MOSI pin, and the slave shifts the LSB to the MISO. The bit is represented as either high or low logic. In the next clock cycle, the master reads the MISO, and the slave reads the MOSI into the most significant bit (MSB) of their respective registers. I.e., each clock-cycle the bits are shifted one slot to the right. After eight clock cycles, each bit is in the correct place, and the two shift registers are successfully exchanged. The slave and the master can now read the value of their respective shift registers and load the next if required. This process is repeated as many times as necessary to transmit the data.

The master then pulls the SS line high, signaling to the slave that the communication is done.

2.3.2 Universal Asynchronous Receive Transmit

Universal Asynchronous Receiver/Transmitter (UART) is a hardware device, rather than a communication protocol like SPI. The UART allows full-duplex asynchronous serial communication between microcontrollers.

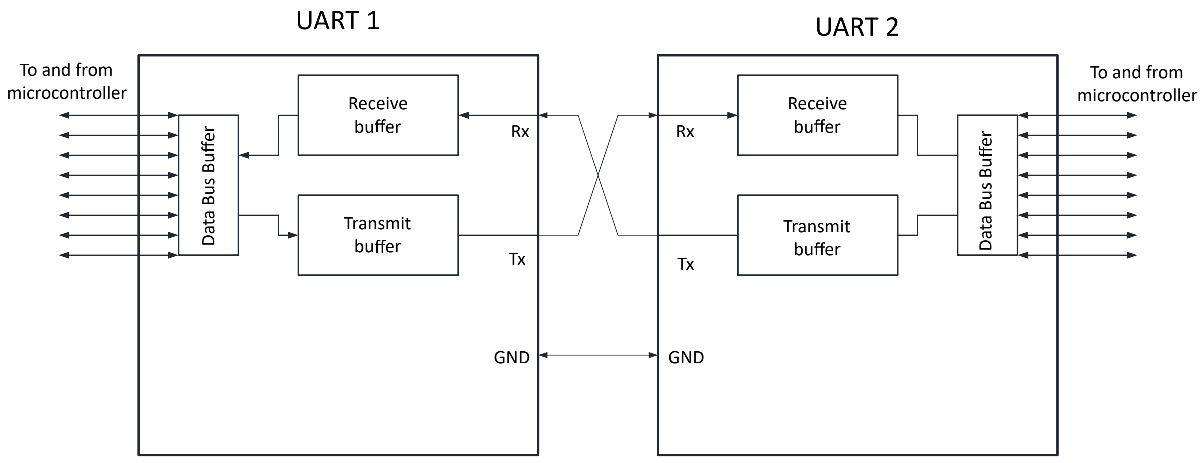


Figure 2.10: UART connection [4]

Communication between UARTs requires two wires: transmitting (Tx), receiving (Rx). But the connection should include a ground. The Tx line of one UART is connected to the Rx line of the other, and vise versa, as shown in figure 2.10.

Data transmission

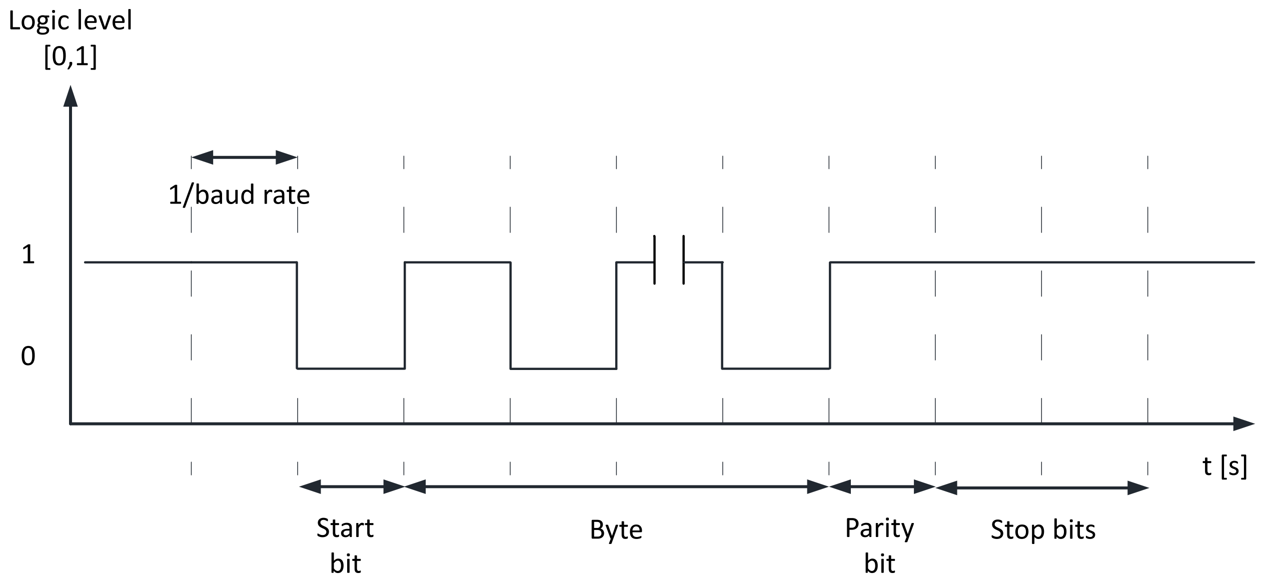


Figure 2.11: Byte transfer UART[4]

Communication between the microcontroller and the UART is done in parallel via transmission and receiving buffers[12]. When transmitting data, the UART receives parallel data from the microcontroller in the transmission buffer. As data is transmitted asynchronously, each byte needs to contain its own synchronization information. The UART, therefore, adds a start bit, a parity bit, and two stop bits to each byte, as illustrated in figure 2.11. This means that when a byte of eight bits is sent, twelve bits are transmitted. The data is then converted into a serial stream of bits, which is transmitted over the Tx line. The receiving UART reads the data, strips away the extra bits, converts it to parallel, and stores the byte in the receive buffer, where the microcontroller can read it.

2.4 Control Theory

In this section, the theory required to perform system identification and create a state-space controller for linear time-invariant (LTI) systems is introduced.

2.4.1 Transfer function

When creating a controller for a system, it is desirable to know how the system will react to the input. For linear time-invariant systems, this can be determined by the transfer function of the system. The transfer function can be obtained by Laplace transforming differential equations of the system and solve for the ratio between the output and input, assuming all-zero initial conditions [6]. The Laplace transformation transforms the equations from the time domain (t) to the frequency domain (s). The benefit of the frequency domain is that operations such as differentiation and integration can be handled with algebra [5].

The benefit of the frequency domain is that operations such as differentiation and integration can be handled with algebra [5].

$$H(s) = \frac{Y(s)}{U(s)} \tag{2.6}$$

Where H(s) is the transfer function, Y(s) is the system output and U(s) is the system input.

When an LTI system is excited with a sinusoidal input, the output response of the system will also be sinusoidal with the same frequency, only the amplitude and phase can change [5]. A logarithmic plot of the amplitude ratio in dB and the phase at different frequencies is called a bode plot. The bode plot shows the frequency response of the system. The response can be calculated using the transfer function of the system.

$$M = |H(j\omega)| \tag{2.7}$$

Where M is the ratio between the amplitude output and input signals in dB.

$$\phi = \angle H(j\omega) \tag{2.8}$$

Where phi is the phase between input and output signal

The frequency response shows how the system responds to excitations of different frequencies. For LTI systems, the superposition principles apply [5]. By knowing the frequency response of a system, and adding a controller, the output response of the system can, therefore, be estimated. This is useful when designing a controller as it does not require actual testing of the system. Analyzing the bode plot of a system also provides helpful information about what control characteristics is required in order to obtain the desired output performance.

2.4.2 Blackbox Frequency Analysis

Obtaining the transfer function is not always easy. The system can have internal components that are not accessible, unknown component values, and the differential equations may be too time-consuming to calculate [10]. In such cases, the black box frequency analysis can be used. The black box frequency analysis is an experimental way to obtain the frequency response of a system. As the transfer function can be used to describe the frequency response, the frequency response can be used to estimate the transfer function. The analysis does not require any prior information about the structure of the system, only that it is linear time-invariant and that the output value is measurable.

The analysis is done by exciting the system with a sinusoidal input of known amplitude. For LTI system, the response of the system will be a sinusoidal output with the same frequency. By calculating the resulting magnitude and phase between the input and output, one point in the bode plot is obtained. Repeating the test for a range of frequencies, the dynamics of the system becomes clear. Two points determine the range of frequencies of interest [5]. At very high frequencies, the system is unable to react to the rapid change of the input. Higher frequencies will, therefore, not provide any additional information. At lower frequencies, the output is fully able to follow the input. Lower frequencies do, therefore, not provide any additional information.

$$M = 20log_{10}\left(\frac{A_o}{A_i}\right) \tag{2.9}$$

Where M $[dB]$ is the gain, A_o [-] is the output amplitude and A_i [-] is the input amplitude[5].

$$\phi = -360ft_{\Delta} \tag{2.10}$$

Where ϕ $[°]$ is the phase, f $[Hz]$ is the frequency and t_{Δ} $[s]$ is the time difference between input and output [5].

2.4.3 Least-squares Approximation

When measuring the output of a system, the logged data is a discrete representation of the signal value at a given time. Plotting the data creates a visual representation of the results. However, it is not a suitable form for further processing the data, such as differentiation and integration, due to quantization error in the measurements. The discrete representation does not provide any information about values between the measure points either. A continuous representation is, therefore, desirable. The least-squares approximation is an approximation method that minimizes the sum of the squared error between the measured data and an input matrix.

$$\mathbf{r}_n = (\mathbf{X}'_{m\times n}\mathbf{X}_{m\times n})^{-1}\mathbf{X}'_{m\times n}\mathbf{Y}_m \tag{2.11}$$

Where **Y** is the output vector containing the measured data, **X** is the input matrix containing the approximation structure multiplied with the desired input, **r** is the coefficient vector. The subscripts m and n are used to denote the number of columns and rows in the vectors and matrix, and the apostrophe is used to denote the matrix transpose.

The **r** vector then contains the coefficients mathematical expression used in the input matrix, as shown in equation 2.11. The success of the approximation relies on that the input matrix has nearly the same structure as the measure data. Approximating a straight line to a trigonometric function, would not yield a good result, but a sine wave would. Careful consideration must, therefore, be applied when using the least-squares approximation.

2.4.4 State-space control

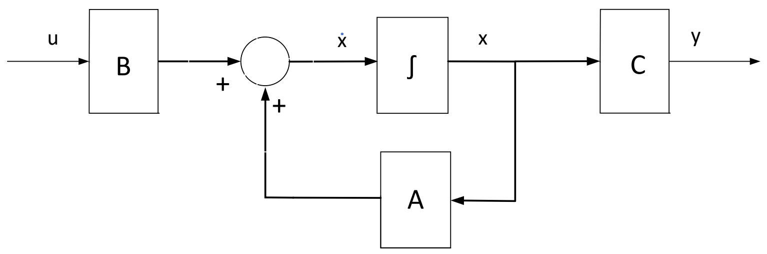


Figure 2.12: Block diagram representation of a state-space model [10]

State-space is an alternative method of describing the transfer function. The differential equations are then used in the state-variable form, where an nth-order differential equation is represented as a set of first-order differential equations, as illustrated in figure 2.12 [6]. The differential equations are ordered in matrices, whose dimensions are dictated by the number of states in the system. As the state-space control structure works directly with the states, the methods of creating a controller are very general and can, therefore, be applied to multiple systems. It also does not require the differential equations to be linear and is particularly well suited for systems with more than one input and output [10].

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.12}$$

$$y = \mathbf{C}\mathbf{x} + Du \tag{2.13}$$

Equations 2.12 and 2.13 show the system on the state space form, where **A** is the system matrix n × n, **B** is the input matrix n × 1, **C** is the output matrix 1 × n, D is a direct transmission term, $\dot{\mathbf{x}}$ is the state vector n for a nth-order system, y is the output, and u is the system input.

2.4.5 State Feedback

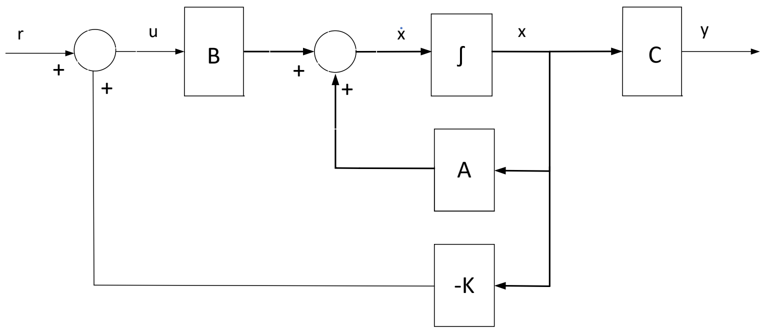


Figure 2.13: Block diagram representation of a state-space model with state feedback [10]

When designing a state-space controller, the first step is to implement state-feedback. This feeds the estimated state vector **x**, back to the control input, through a gain vector **K**, as shown in figure 2.13.

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}r \tag{2.14}$$

$$y = \mathbf{x} \tag{2.15}$$

where **K** is the state-feedback vector.

Implementing the state-feedback to the system description [6], the representation of the **A** matrix changes, as seen from equation 2.14. The eigen behavior of the system is, therefore, changed. As the **K** vector can be set to any value, the closed loop poles of the system to be placed arbitrarily.

2.4.6 Prefilter

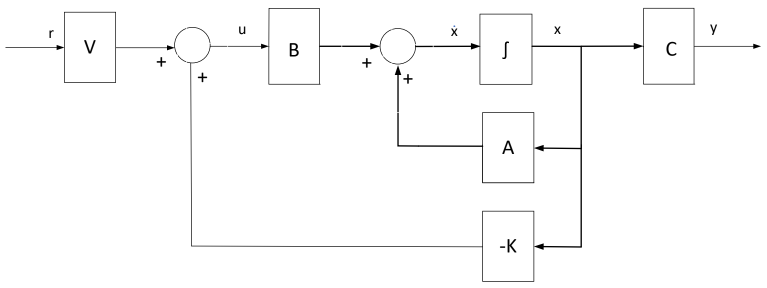


Figure 2.14: Block diagram representation of a state-space model with state feedback and prefilter [6]

Including the state feedback does not provide any steady-state accuracy. An input gain relating the input value to the corresponding output value does, therefore, need to be calculated. This is called the reference input, or prefilter [6], as can be seen in 2.14. This is done by calculating the value for the input u, such that the reference input matches the system output y.

$$\mathbf{V} = (\mathbf{C}(\mathbf{BK} - \mathbf{A})^{-1}\mathbf{B})^{-1} \tag{2.16}$$

Where **V** is the prefilter gain [15].

2.4.7 Integral Control

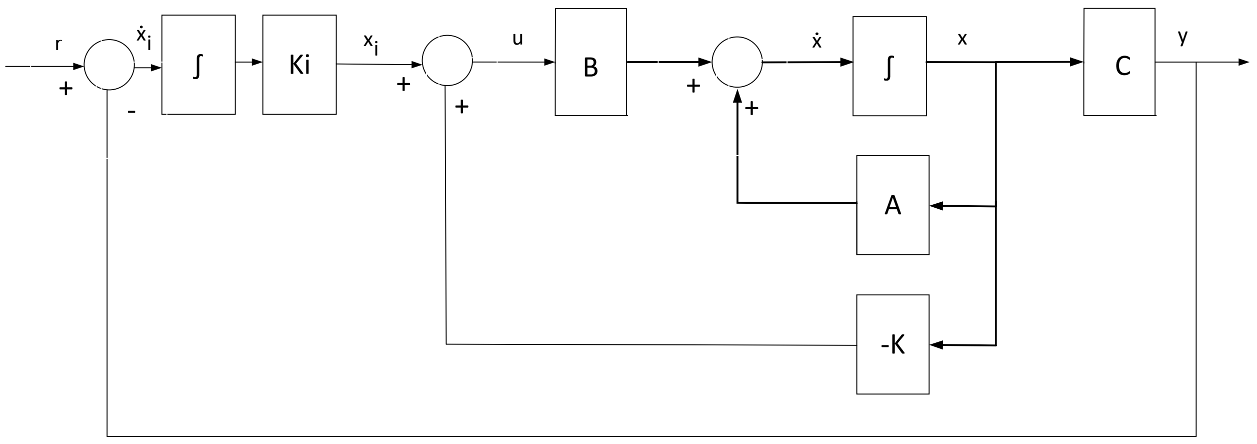


Figure 2.15: Block diagram representation of a state-space model with output feedback and integral control [6]

The state feedback and prefilter do not account for model inaccuracies or process noise[6]. Output accuracy can, therefore, not be guaranteed. A feedback path is for the system output is therefore added, as shown in figure 2.15. The feedback is subtracted from the reference signal to produce a negative error. The error is fed forward through an integrator, reducing the steady-state error of the output to zero. The integrator requires that an extra state is added to the system.

$$\begin{bmatrix} \dot{x_i} \\ \dot{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} 0 & \textbf{-C} \\ \mathbf{B}k_i & \mathbf{A}-\mathbf{BK} \end{bmatrix} \begin{bmatrix} x_i \\ \mathbf{x} \end{bmatrix} - \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} r \tag{2.17}$$

$$y = \begin{bmatrix} 0 & \mathbf{C} \end{bmatrix} \begin{bmatrix} x_i \\ \mathbf{x} \end{bmatrix} \tag{2.18}$$

Where x$_i$ is the integrator state, $\dot{x_i}$ is the time differentiated integrator state, k$_i$ is the feedback gain for the integrator

2.4.8 Observer

The theory described so far relies on all the states variables being measured. However, it is not always practically possible to measure all the states of a system, not desirable to buy the extra sensor required. In such cases, an observer, also called an estimator, can be used. The observer is a model of the plant which receives the same input as the actual system and uses feedback from the measured states to produce an error. The error is used by the model to correct the states to match the output.

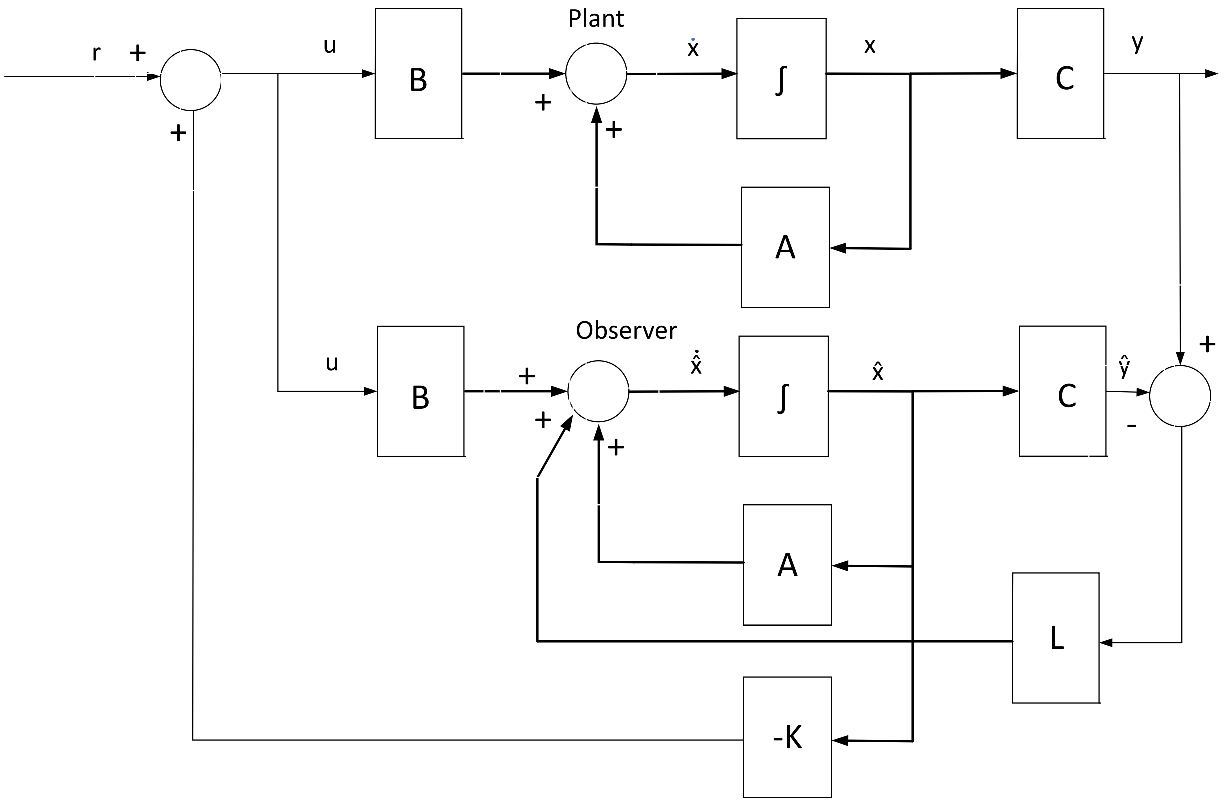


Figure 2.16: Block diagram representation of a state-space model with an observer

$$\dot{\hat{\mathbf{X}}} = (\mathbf{A} - \mathbf{LC})\hat{\mathbf{X}} + \mathbf{B}u + \mathbf{L}y \tag{2.19}$$

$$\hat{y} = \mathbf{C}\hat{\mathbf{x}} \tag{2.20}$$

$$\dot{\epsilon} = (\mathbf{A} - \mathbf{LC})\epsilon \tag{2.21}$$

Where **L** is the observer gain, ϵ is the error between measured and esitmated state values and hat is used to denote estimated values.

From equations 2.19 to 2.21 that both the estimated states and the observer error is dependant on the observer gains. Thus, by chosing choosing faster poles for the observer than for the measured system will drive the error to zero.

Chapter 3

Method

3.1 Concept selection

In this section, different concepts are introduced and evaluated before finally, one concept is selected. The chosen concept is to be the basis for further development of the cat robot.

3.1.1 Selection criteria

In this subsection, a list of criteria is created based on the problem statement. The concepts are scored individually based on the chosen criteria and weighed against each other until one concept is chosen.

Each criterion is given a score from one to ten and a . The criterion score is then multiplied with a weighting factor, which signifies the relative importance of the criterion. The sum of the weighting factors is one. The final score of the concept is the sum of all the weighted scores and is, therefore, a number between one and ten. The finale score is calculated according to equation 3.1.

$$X_i = \sum_{j=1}^{N} x_{i,j} w_j \tag{3.1}$$

Where X is the final score, x is the criterion score, w is the weigth factor, subscript i is the concept number and subscript j is the criterion number.

Catlikeness Catlikeness is a measure of how similarly the motion of the concept is to that of a cat. As this is a cat-robot, it should not be arbitrary how it moves. Catlikeness is, for this reason, a criterion that is weighted quite heavily, at 0,6 of the final score. The criterion score is given from one to ten, where ten is associated with high similarity to a cat.

Level of theoretical complexity Most of the suggested concepts involve rotation about more than one axis, it is fair to assume that the differential equations governing the motion of the cat are non-linear. The added constraints of zero angular momentum and non-holonomic motion further complicates the solution. The final product includes the design of a control algorithm for the robot, which accounts for these non-linearities. It is, therefore, desirable to limit the theoretical complexity of the concept. The criterion is scored from one to ten, where ten is associated with low complexity. Level of theoretical complexity is weighed at 0,1.

Level of implementational complexity

Level of implementational difficulty is a measure of the level of difficulty expected in implementing the sensors and actuators in the design. If a sensor is required on a part that rotates relative to the part the MCU is on, the wires can tangle. Special considerations must therefore be made to prevent this, which complicates the design.

Unrealistic expectation of actuator strength, sensor resolution or MCU computing power is also considered.

Level of implementational difficulty is therefore a measure of how easily a design can be acheived. A score of one is considered very difficult, and a score of ten is considered easy.

Price As states in the problem description, the prototype should be low-cost. High-performance actuators, high resolution of sensors, and the number of components contribute to driving the price of the prototype up. Components such as these should be avoided, as there is a limited budget for this project.
It is hard to estimate the cost of a product based on a concept, but the mentioned factors are used as an indicator. The price is scored from one to ten, ten being cost beneficial. The criterion is weighed at 0,2.

3.1.2 Concepts and scoring

In this subsection, different concepts are introduced. The primary focus in the concept phase is to generate ideas of how can be rotated in the air while conserving zero angular momentum.

Concept 1:

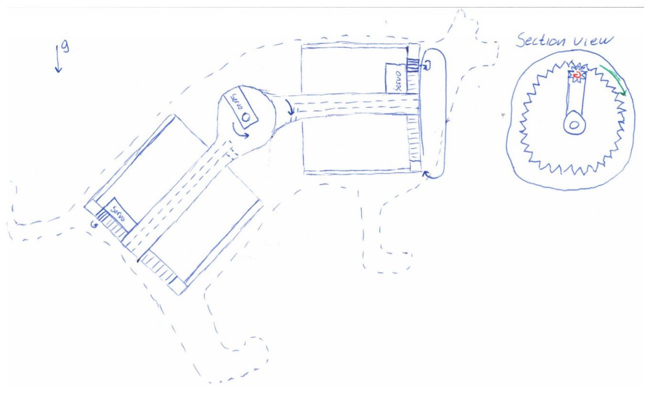


Figure 3.1: Concept one

Concept one consists of two rods joined together in a revolute joint, as can be seen in figure 3.1. The two rods serve as an axis of revolution for two cylinders mounted on the rods by roller bearings. The bearings allow the cylinder to rotate relative to the rods. The concept requires three actuators, one for rotating the revolute joint and two for rotating the cylinders about the rods. Two legs are mounted on each cylinder. Rotating the cylinders will, therefore, change the orientation of the cat.

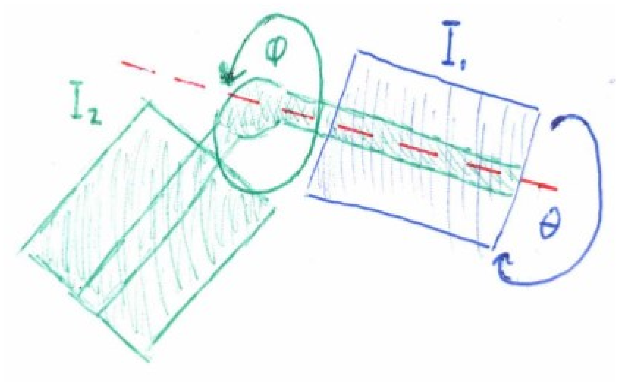


Figure 3.2: Change in inertia

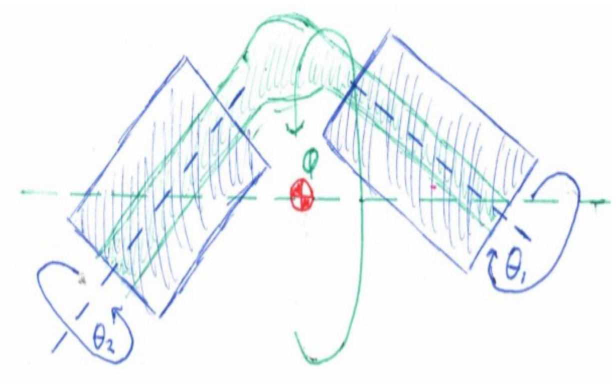


Figure 3.3: Compound effect

The concept is based on the tuck-and-turn model suggested by Schere and Kane[18]. The basic principle is that by bending the spine, the two cylinders no longer share a common axis of revolution. As a result, the relative inertia between the two parts is changed, as shown in figure 3.2. Thus rotating one cylinder (I_1), will cause a smaller proportional rotation in the rest of the body (I_2), as shown in figure 3.2. Turning the two cylinders in opposite directions creates a compound effect on the entire cat to conserve the angular momentum, as shown in figure 3.3. Thus the entire cat starts turning about an axis dictated by the center of gravity for the front and the rear body. By rotating the cylinders 360° and controlling the angle between the front and the rear, the cat should be able to reach an upright position from any initial condition. The Schere and Kane model is, however, based on a purely mathematical description of the problem. Ideal cylinders are used to rotate freely around a shared point with no regard for the physical system required to perform such movements. Therefore, it does not provide a complete system description.

Catlikeness: Concept one is based on a model used to describe the movement of a cat in free-fall. However, in order to create a physical model from the theoretical one, the spine would need to rotate the opposite direction from the front and the rear body. This reduces the resemblance of a cat.

Level of theoretical complexity: The bending of the spine causes the rotation of the cat to be about multiple axes. The axis of revolution for the cat is dictated by the center of gravity of the front and the rear body. The inertia changes quadratically as a function of the distance from the axis of revolution, and the distance changes trigonometrically as a function of the spine angle. This makes the governing differential equations highly non-linear. The roll angle of the cat is not directly controllable, only indirectly by rotating the cylinders: requiring high accuracy in the system model, and the control system.

Level of implementational difficulty: The concept requires rotation between the rear body and the rear spine, the front body and the front spine, and the rear and front spine. The spacer between the spine and the rod makes it hard to pass wires required for sensors and actuators between the front and the rear body. This makes it challenging to place sensors and actuators opposite body to that of the microcontroller.

Price: The concept requires three actuators and at least two angular measuring sensors. The rest of the required components should not be difficult to obtain.

Table 3.1: Score concept 1

Criterion	**Score**
Catlikeness	7
Level of theoretical complexity	3
Level of implementational difficulty	4
Price	7
Finale score	**6,3**

Concept 2:

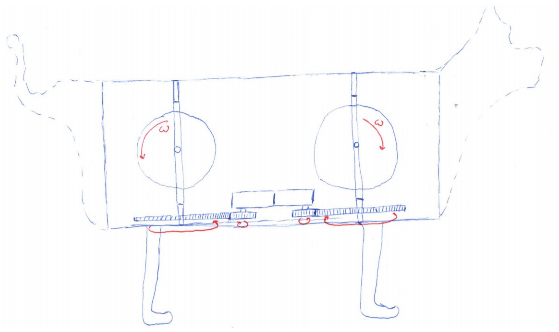


Figure 3.4: Concept two

Concept two consists of two flywheels rotation inside one body in opposite directions with the same moment of inertia and angular speed, as can be seen in figure 3.4. The flywheels are supported at the top and the bottom of the body. The support can be rotated freely through a gearing mechanism causing the angular momentum to change direction.

In the default position, the angular momentums caused by the flywheels cancel about the axis perpendicular to the spine. Changing the angle of the support will cause the angular momentum to change direction. The angular momentum of the two flywheels, therefore, no longer completely cancels each other, and the cat starts to turn, seemingly without moving.

Catlikeness: Utilizing stored energy to perform a rotation is far from how a feline operates. There are very few similarities between this concept and the movement of a cat.

Level of theoretical complexity: Not considering the flywheels, the cat consists of one rotating part; which decreases the complexity. Assuming the angular velocity of the flywheels is constant, there is one clear control input: the angle of the rotation of the support of the flywheels.

Level of implementational difficulty: The angular velocity of the flywheels must be equal, or else the cat is not in steady-state by default. The actuators and feedback signal, therefore, need to be precise. Sensors and actuators are mounted on the same part, which decreases the complexity.

Price: A prototype of the concept can be created with three motors with feedback and a gyroscope. The rest of the parts can be 3D-printed without complications. This is very cost beneficial.

Table 3.2: Score concept 2

Criterion	**Score**
Catlikeness	2
Level of theoretical complexity	7
Level of implementational difficulty	7
Price	8
Finale score	**4,2**

Concept 3:

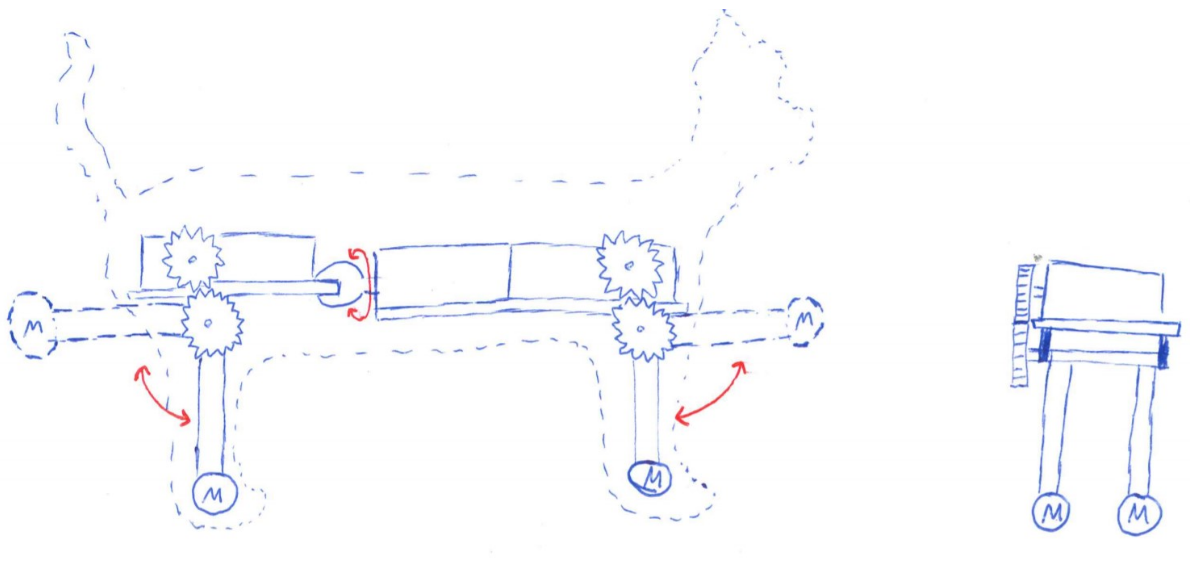


Figure 3.5: Concept three

Concept three is based on the legs-in/legs-out model suggested by J. R. Benton [2]. The concept consists of two bodies able to rotate relative to each other along the spine axis. Both of the two bodies are connected to a pair of legs. The legs can be extended and retracted, as shown in figure 3.5.
The idea is to change the relative inertia between the front and the rear body about the spine. This is done by extending and retracting the legs of the cat. Keeping one pair of legs extended, and the other retracted causes a more significant rotation in the retracted part than in the extended by an equal torque.
Altering the orientation of the cat is done by first rotate the two bodies with one pair of feet retracted and the other extended, change which feet are extended and retracted, then rotate back.

Catlikeness: Extending and retracting legs to manipulate inertia, aswell as the front and the rear body rotating in a different direction, is observed in the righting reflex of the cat. However, for this to work as the only mechanism for turning would require a significant mass placed in the legs.

Level of theoretical complexity: For the legs to end up in the correct angle, the system model and the control system has to be very accurate. Time is very limited when dropping an object from a reasonable height to it reaches the ground. For this reason, the legs would need to rotate at the same time as the spine. The rotation of multiple bodies increases the complexity of the governing equations for the movement significantly.

Level of implementational difficulty: There are many individual steps required to perform the rotation for this concept, which likely requires high-speed actuators. Tangling wires might become an issue, as bodies are rotating relative to each other. Mass located in the feet significantly increases the inertia of the rotations, which might require high torque from the actuators.

Price: The concept is likely to require high-performance actuators, which increases the cost. The rest of the prototype should be possible to 3D print.

Table 3.3: Score concept 3

Criterion	**Score**
Catlikeness	7
Level of theoretical complexity	5
Level of implementational difficulty	4
Price	4
Finale score	**5,9**

Concept 4:

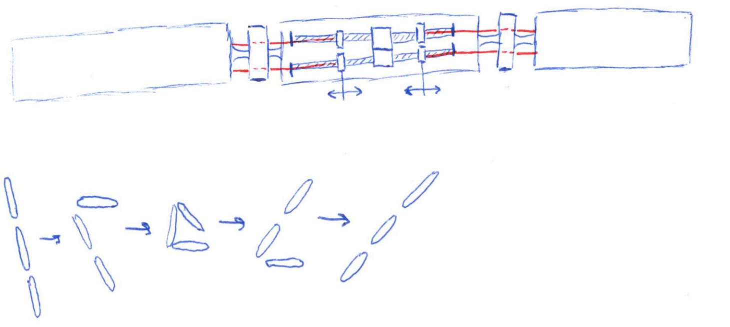


Figure 3.6: Concept four

Concept four is based on a concept suggested by J.T Bingham to reduce impact after a fall[3]. The concept consists of three bodies joined by actuated revolute joints. The joints can be rotated individually in both directions, as can be seen in figure 3.6. The concept takes advantage of the change of inertia caused by individually rotating each joint. One body moves relative to the two others, thus experiencing a more significant angular displacement by an equally large torque. Whether the two bodies are colinear or in an angle significantly affects their combined inertia, and thereby their displacement from a torque. By first bending one joint, then the other causes a more significant displacement than retracting the first joint and then the other. Ergo the sequence in which the bodies are rotated affects their ability to return to their original state. After performing the described sequences, the bodies return to their original configuration, but with a different angle. Repeating these sequences changes the pitch angle of the cat.

Catlikeness: Altering the pitch angle of the cat is not as well documented as the roll angle. For this reason, it is hard to grade this concept.

Level of theoretical complexity: The concept would require multiple iterations of the described sequence of movements in order to alter the pitch angle adequately. Time optimization of the motion will, therefore, be essential. It is also likely that is it is necessary to actuate both joints at the same time. This would significantly increase the difficulty of the task of solving the differential equations.

Level of implementational difficulty: In order to complete several iterations in a drop from a reasonable height, the actuators would have to be fast, perhaps unrealistically. The sensors would have high-performance requirements to measure the robot's angle in such a high-speed adequately.

Price: The price of actuators and sensors required for this concept is assumed to be very high.

Table 3.4: Score concept 5

Criterion	**Score**
Catlikeness	6
Level of theoretical complexity	5
Level of implementational difficulty	2
Price	1
Finale score	**4,2**

Concept 5:

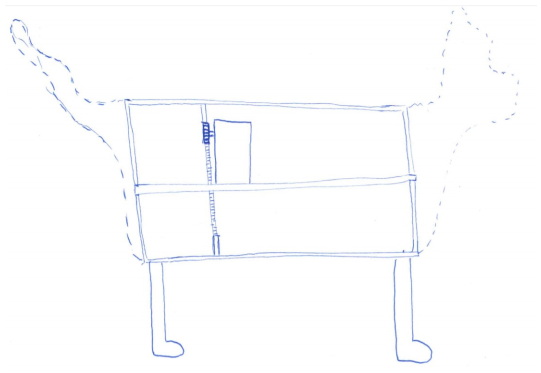


Figure 3.7: Concept five

Concept five consists of two bodies, one located inside the other, rotating relative to each other about a common axis of revolution. The concept can be seen in figure 3.7. When torque is generated in the inner body, the outer shell experiences an equal but oppositely directed torque. The torque causes the outer shell to rotate in the opposite direction to the inner part. The rotation stops when the outer shell has reached the desired angle.

Catlikeness: The concept utilizes an "invisible" inner part to rotate a visible outer part. This does not resemble the turning of a cat.

Level of theoretical complexity: Both bodies turn about the same axis with one actuator, which decreases the complexity. There is only one unknown parameter of interest, the angle of the outer shell relative to gravity, and one part which require actuation, the relative rotation between the two components.

Level of implementational difficulty: The concept does not require sensors or actuators to be mounted on more than one part, simplifying the implementation of parts on the concept.

Price: The parts required to create a prototype of this concept should be easily obtainable. The inner part and the outer shell can both be 3D printed, and the actuator can be geared freely to meet any torque or velocity demand. The concept would require one gyroscope and one angular position sensor to measure the position between the two parts.

Table 3.5: Score concept 6

Criterion	**Score**
Catlikeness	2
Level of theoretical complexity	9
Level of implementational difficulty	7
Price	8
Finale score	**4,4**

3.1.3 Concept choice

Table 3.6: Concept scores

Concept	**Catlikeness**	**Theoretical**	**Implementation**	**Price**	**Score**
1	7	3	4	7	**6,3**
2	2	7	7	8	**4,2**
3	7	5	4	4	**5,9**
4	6	5	2	2	**4,2**
5	2	9	7	8	**4,4**

As can be seen from table 3.6, concept one and three scored the highest based on the chosen criteria. As there are many assumptions involved in the scoring, the difference of 0,4 is insignificant. The choice was, therefore, taken based on personal preference. Concept one is the concept chosen for further development of a prototype.

3.1.4 Preliminary simulation

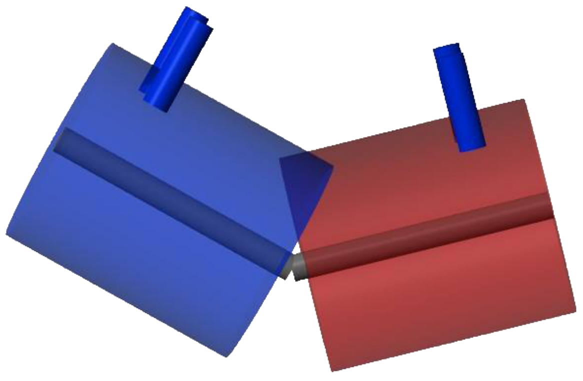


Figure 3.8: Preliminary simulation model side view

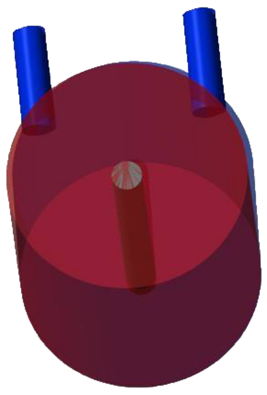


Figure 3.9: Preliminary simulation model front view

In order to test the chosen concept, a preliminary simulation is performed. The simulation simulates two cylinder-pairs, each sharing a common axis of revolution. Each pair consists of one cylinder with a small diameter, but longer, and one cylinder with a larger diameter, but shorter. The two-cylinder pairs are connected at the end of the small radius cylinder, as shown in figure 3.8 and 3.9.

The legs are kept massless through the simulations and are for illustration purposes only.

This simulation made it possible to gain a better understanding of the nature of the system. Multiple simulations were performed on the same model, changing the mass, diameter, length, and angle between the cylinders while keeping the input constant. The input was a trapezoidal velocity profile for the speed of the revolute joint between the small and large cylinder, integrating to a rotation of 360° per simulation.

The simulation shows no matter the angle or relative inertia between the two-cylinder pairs, completing a 360° rotation aligns the bend with the feet and changes roll angle of the system.

3.1.5 Project Plan

The progress of the project is divided into three parts. This is done to prevent building a complete robot only to find out that the motors do not produce enough torque. Or wasting much time creating a system model who is no longer valid because components have to be implemented in a different way, which completely changes the robot's design and dynamics.
Step one: In step one, the task is to design and select components for a prototype where the angle of the waist can be fixed without actuation. The parts should be designed to be able to assemble and disassemble the robot easily. Communication between the MCU, the sensors, and actuators be established, and a suitable feedback controller will be created for the motor. Tests will be performed in order to ensure that the chosen components perform satisfactorily before progressing to step two.
Step two: In step two, the feet and actuation of the waist will be implemented. A complete system model and a control algorithm will be created. Step two is completed when the robot is able to land on its feet when dropped reliably.
Step three: In step three, the focus is to make the cat robust without changing the robot's dynamics. Parts designed or implemented for ease of assembly/disassembly should be evaluated, and changed where needed. Evaluations regarding the appearance of the robot should also be done to enable skins for the robot. Step three is completed when the performance of step two is maintained while confident in the robustness of the cat's mechanical parameters.

3.2 Component Selection

In this section the components required to complete step one is identified, and suitable components are chosen. In order to choose one component, assumptions need to be made about the rest. This process might require several iterations in order to result in a suitable combination of components.

3.2.1 Required Components

In order to create a robot cat, a few components are required. As the name of the concept suggests, two cylinders are required. The outer cylinder will be referred to as the outer pipe, and the inner cylinder is called the inner tube. Suitable materials and dimensions have to be identified.

As the two cylinders have to rotate relative to each other, actuators are required. Speed and torque requirements will have to be identified in order to select a suitable motor.

The angle between the two cylinders has to be measured. Suitable sensors and resolution of sensors have to be identified.

The robot cat has to detect the fall in order to activate the righting reflex. The angle of the robot, not only the relative angle between the two cylinders are of interest. For this, an Inertial Measurement Unit (IMU) is required. The IMU contains both an accelerometer able to detect the drop in gravitational force induced by the fall and a gyroscope able to measure the angular velocity of the robot.

In order to process sensor data, and produce an actuation signal for the motor, a Microcomputer Unit (MCU) is required. In the problem description, it is stated that a computing platform is needed, multibody dynamics and closed-loop simulation, and tests using the Motion Lab shall be performed. Matlab/Simulink has solutions to implement all of these tasks. 3D models can be created in SolidWorks and imported into Simulink simscape multibody using a simple plug-in. This creates a 3D model of the system, which can be used to perform simulations and test control algorithms. Position feedback from the Motion Lab can be read in Simulink in real-time. It is, for these reasons, desirable to choose an embedded platform that is also compatible with Matlab.

A power supply is also required. The power supply must be able to supply the voltage and current required by the components.

3.2.2 Motor

In this section, the motor requirements will be identified and evaluated before one is chosen. A suitable motor fulfills the requirements, is low-weight, low-cost, and compact.

There are usually two requirements the motors needs to fulfill: the speed and the torque requirements. To be able to correctly calculate these requirements the mechanical structure of the robot-cat must be known, which it is not. The requirements for the motors is therefore estimated by making four assumptions. The first assumption is that the torque requirement for each motor can be estimated by considering one half of the robot-cat: the front or the rear, and keeping the inner part in a fixed position. The second assumption is that the outer assembly consists of one outer tube, two bearings, two circular plates for connecting the outer tube to the bearings, and two shock absorbers to act as legs. The legs are not a part of the first stage of component selection, but should be considered when sizing the motor. The third assumption is regarding the material parameters of the outer assembly required to calculate the inertia. The fourth assumption is about the velocity profile for the movement.

Motor requirements

$$\sum T = 0 = I_o \alpha_o - T_{motor} \tag{3.2}$$

$$\Rightarrow T_{motor} = I_o \alpha_o \tag{3.3}$$

Equation 3.2 is the torque equation between the outer tube, and the motor where I_o is the inertia of the outer tube, α_o is the angular acceleration of the outer tube and T_{motor} is the torque supplied by the motor. The inertia is constant, and the angular acceleration is determined by time to perform the desired motion and the velocity profile. The torque requirement can, therefore, be estimated by obtaining the highest required acceleration and the inertia of the parts directly connected to the outer tube.

The parts connected to the outer tube are two bearings, two circular plates for connecting the outer tube to the bearings and two shock absorbers to act as legs. The weight of the two shock absorbers pulls the center of gravity of the outer assembly out of the axis of revolution, to prevent these two counter inertias are added to the calculation.

$$I_{pipe} = \frac{m}{2}\left(r_1^2 + r_2^2\right) \tag{3.4}$$

Where I_{cyl} $[kg \cdot m^2]$ is the inertia of a cylinder, m $[kg]$ is the mass, r_1 $[m]$ is the inner radius and r_2 $[m]$ is the outer radius.

$$I_{rod} = \frac{ml^2}{12} \tag{3.5}$$

where I_{rod} $[kg \cdot m^2]$ is the inertia of the rod, m $[kg]$ is the mass and l $[m]$ is the length of the rod.

$$I = I_{cg} + md^2 \tag{3.6}$$

where I $[kg \cdot m^2]$ is the total inertia, I_{cg} $[kg \cdot m^2]$ is the inertia about the center of gravity, m $[kg]$ is the mass and d $[m]$ is the distance away from the axis.

$$\sum I_{total} = I_{outertube} + 2I_{plate} + 2I_{bearing} + 4I_{shockabsorber} \tag{3.7}$$

Where I_{total} $[kg \cdot m^2]$ is the total inertia of the outer assembly.

Table 3.7: Assumed dimensions and material properties

Part	**Eq**	**ID**	**OD**	**L**	**t**	**D**	ρ	**m**	**I**
Outer Tube	3.4	0.120	0.105	0.24	-	-	1240	-	0.0025
Plate	3.4	0.0105	0.048	-	0.005	-	1240	-	7e-05
Bearing	3.4	0.048	0.025	-	-	-	-	0.1	4e-5
Shock Absorber	3.5, 3.6	-	-	0.1	-	0.12	-	0.05	8e-4
Total	3.7								0.0058

Table 3.7 show the calculated inertia for each part, and the total inertia of the outer assembly.

As discussed in section 3.1, rotating the cylinder 360° aligns the legs with the spine bend. For the speed requirement, the outer assembly should, therefore, rotate one rotation in as little time as possible.

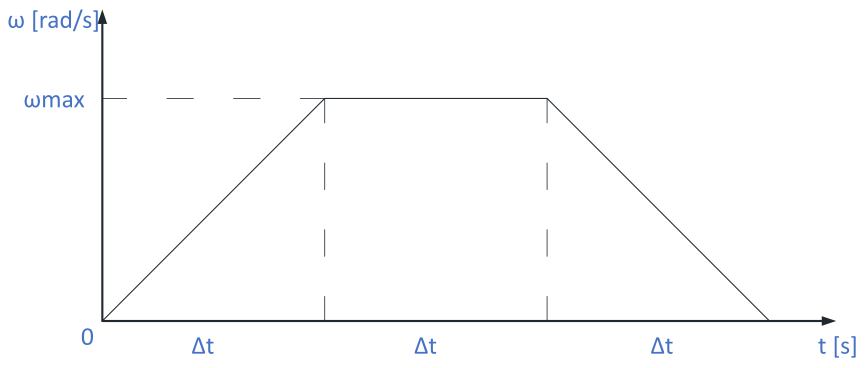


Figure 3.10: Trapezoidal velocity profile

A trapezoidal velocity profile, as seen in figure 3.10, is used to obtain the motor's acceleration and speed requirement[7]. The change in angle is the area under the curve. The timesteps for acceleration is equal to that of the deaccleration and constant speed. This produces the following equations for maximal speed and acceleration.

$$\omega_{max} = \frac{\Delta\Theta}{2\Delta t} \tag{3.8}$$

where ω_{max} is the speed requirement, $\Delta\Theta$ is the change in angle, Δt is the time duration.

$$\alpha_{max} = \frac{\omega_{max}}{\Delta t} \tag{3.9}$$

where α_{max} is the maximal angular acceleration. By keeping $\Delta\Theta$ constantly equal to 2π, the max speed and angular accelation is a function of the desired time.

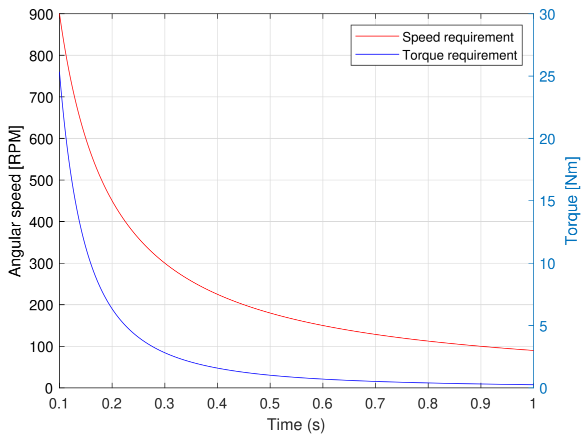


Figure 3.11: Motor requirements as a function of time to complete one rotation

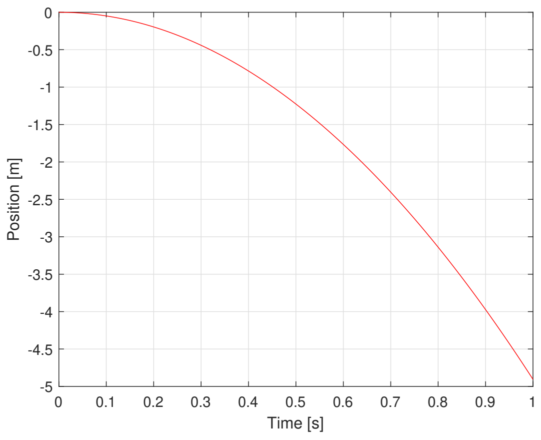


Figure 3.12: Position of a object in free fall

Figure 3.11 show the required motor speed and torque as a function of time to complete one rotation based on the assumed inertia and velocity profile. As time decreases, the motor requirements increase. Figure 3.12 show the position of an object in free fall under the impact of gravity initially at rest, as a function of time. The velocity of the falling object quickly increases, and the object drops rapidly.

There is, therefore, a trade-off between the time to complete one rotation and the motor requirements. By choosing a smaller timeframe, the length of the drop decreased, but the motor requirements increased. Higher requirements require larger, heavier, and more expensive motors and the assumptions made regarding inertia might need to be reevaluated. Choosing a larger time frame results in lower motor requirements, but a larger drop. This is not desirable as an actual cat can turn in the fraction of a second, and the drop causes a higher impact force on the components.

A drop time of 0,5 $[s]$ is therefore considered to be a reasonable time-frame. The drop is then 1,3 $[m]$, the torque requirement is 1.0136 $[Nm]$ and the speed requirement is 180 $[RPM]$.

The torque requirement of the motor is relatively high compared to the speed requirement. Brushless permanent magnet motors have a high torque/mass ratio, which is desirable for this project [16]. These motors generally offer high speed but low torque, which can be compensated for by adding a high-reduction gearbox. Retail electronic stores offer a wide selection of DC motor; however, none is found that meets the requirements. Electric drives manufacturers offer a larger selection and the option to customize an electric drive combination consisting of a dc motor, gearbox and control unit.

Maxon Brushless DC motor

When sizing a motor, the load characteristics for the specific motor must be considered. Applying a torque over the nominal value for an extended period causes the motor to overheat. However, the cat-robot requires bursts of torque over a very short time duration and is therefore not limited by the thermal effects of operation over the continuous operation limit. In the selection process, torque up to the torque where the motor stalls is considered. The maximal angular velocity is calculated based on the power of the motor at a given torque. The calculations include efficiencies for the motor, gearbox, and controller.

$$T_{max} = Tn\eta_m\eta_g \tag{3.10}$$

where T_{max} $[Nm]$ is the output torque from the gearbox, T $[Nm]$ output torque of the motor, n $[-]$ is the gear ratio, η_m $[-]$ is the motor efficiency and η_g $[-]$ is the gear efficiency.

$$\omega = \frac{P}{T} \tag{3.11}$$

where P $[W]$ is the power of the motor, T $[Nm]$ is the torque of the motor and ω $[\frac{rad}{s}]$ is the output speed.

$$\omega_{out} = \frac{\omega_{motor}}{n}\frac{60}{2\pi} \tag{3.12}$$

where ω_{out} $[RPM]$ is the output speed, ω_{motor} $[RPM]$ is the motor speed and the constants are convertion from $\frac{rad}{s}$ to RPM.

$$I_{req} = \frac{T_{max,m}}{k_t\eta_c} \tag{3.13}$$

Where I_{req} $[A]$ is the current requirement the for the controller, $T_{max,m}$ $[Nm]$ is the motor torque, $k_t[\frac{Nm}{A}]$ is the torque constant of the motor and η_c $[-]$ is the efficiency of the controller.

Equation 3.10 is used to calculate the output torque from the gearbox, equation 3.11 is used to calculate the highest speed the motor can output at the calculated torque, equation 3.12 is used to calculate the output speed from the gearbox, and equation 3.13 is used to calculate the current requirement of the controller.

The combination of EC-i 45W brushless dc motor, planetary gearhead GP 32 C and ESCON Module 50/5 controller was identified as a suitable option.

Table 3.8: Motor parameters EC-i 45, brushless

Parameter	**Value**	**Unit**
Part number	539480	[-]
Power	45	[W]
Nominal voltage	12	[V]
Stall torque	0,731	[Nm]
Max efficiency	86	[%]
Weight	0,150	[kg]
Diameter	0.03	[m]
Cost	2573	NOK

Table 3.8 shows the motor parameters for a brushless dc motor: EC-i 30

Table 3.9: Gearbox parameters Planetary Gearhead GP 32 C

Parameter	**Value**	**Unit**
Part number	166933	[-]
Reduction	14:1	[-]
Max. intermittent torque	3,75	[Nm]
Max. intermittent speed	8000	[RPM]
Max efficiency	75	[%]
Weight	0,160	[kg]
Diameter	0,032	[m]
Cost	2195	NOK

Table 3.9 show the parameters of interest for a planetary gearhead: GP 32 C gearbox.

Table 3.10: Controller parameters ESCON Module 50/5

Parameter	**Value**	**Unit**
Part number	438725	[-]
Max. output current	15	[A]
Max efficiency	98	[%]
Weight	0,009	[kg]
Cost	1952	NOK

Table 3.10 show the parameters of interest for a speed controller: ESCON Module 50/5.

Table 3.11: Electric drive specifications

Parameter	Value	Unit
Required output speed motor	3285	[RPM]
Max. intermittent speed gearbox	8000	[RPM]
Required output torque	1,164	[Nm]
Max. intermittent torque gearbox	3,75	[Nm]
Required current	9,55	[A]
Power motor	45	[W]
Max power controller	250	
Diameter	0,032	[m]
Weight	0,322	[kg]
Cost	6720	NOK

The cost of the products are converted to NOK from Euro assuming a currency exchange rate of 10,9 NOK to one euro, and include the customs fee of 25%, transportation costs are not included.

Table 3.11 show the parameters of interest for the electric drive combination. The maximal input torque and speed of the gearbox is lower than the required output from the motor. The maximal current of the controller is lower than the current required by the motor. The power rating of the motor is lower than the specified maximum for the controller. And as can be seen, the suggested combination of components satisfies both the speed and the torque requirements, including losses caused by efficiency according to equations 3.10-3.13.

Servo motor

As discussed in section 2.2, a hobby servo motor is a geared DC motor with a feedback loop. The hobby servo motors are usually restricted to a rotation of 180° by the feedback system and a physical barrier placed in the gearing. Substituting the potentiometer with two fixed resistors, and removing the physical blocking on the gear will convert the servo motor into a continuous rotation motor without feedback. This is because the feedback system tries to correct for an assumed angle that does not change due to the resistors. The larger the input, the larger the proportional gain between the angle and the position assumed by the servo controller. The PWM input, therefore, becomes an open-loop speed signal. What remains is a DC motor, controller and gearing, the loop can be closed again by implementing a position sensor.

There are drawbacks of choosing a hobby servo motor. The modification process is not without risk, one wrong move, and the servo motor becomes useless. Also, the suppliers generally only provide information about the maximal torque and the speed of the servo motor in $\frac{s}{60^\circ}$. This is not much to go on, but with system identification tools, as discussed in section 2.4.2 and proper motor calibration, it should be possible to create a sufficiently accurate model motor.

Table 3.12: Futaba HPS-700

Parameter	Value	Unit
Torque @7.4V	4.3	[Nm]
Speed @7.4V	143	[RPM]
Weight	0,076	[kg]
Cost	1980	NOK

Table 3.12 show the provided information about the servo motor Futaba HPS-700. As can be seen, the motor has a significantly larger output torque than required, but the maximal speed is below the requirement.

$$T_{ratio} = \frac{T_m}{T_{req}} = \frac{4,4[Nm]}{1,0136[Nm]} = 4,34[-] \tag{3.14}$$

$$\omega_{ratio} = \frac{\omega_m[\frac{rad}{s}]}{\omega_{req}[\frac{rad}{s}]} = \frac{143}{180} = 0,7937[-] \tag{3.15}$$

As shown in equation 3.14, the torque is over four times higher, and the motor speed is about 80% of the requirement. By creating a gear between the motor and outer assembly with a gear ratio greater than 1:0,7937, both the torque and the speed requirements are met. This gearing can be 3D-printed and placed within the outer assembly. The motor is, for this reason, considered to meet the requirements.

Selection

Selecting a motor is done by the same selection process as for the concepts in section 3.1. The motors are given a score from one to ten, the scores are weighted and summed, and a final score is calculated.
Cost: is scored by the price of the electric drive. A score of one indicates a high cost, and ten indicates a low. The cost is weighted at 0.6.
Weigth: is scored by the weight of the component. A score of one indicates a high weight, and ten indicates a low. The weight is weighted at 0.4.

Table 3.13: Motor scores

Selection criterion	Futatba	Maxon
Cost	6	2
Weigth	8	6
Total	6,8	3,6

As can be seen from table 3.13, the Futaba servo motor scores significantly higher than the Maxon motor based on the selection criteria. Futaba HPS-700 is for this reason, chosen as the motor for the robot-cat.

3.2.3 MCU

In this subsection, an MCU is selected. The MCU should be compatible with Matlab/Simulink and must be able to handle the real-time constraints.

The robot-cat should be able to complete the desired motions in 0,5 seconds. For smooth control of the actuators, the feedback signal from the sensors needs to be processed quickly and at the right time. It is, therefore, essential that the MCU has real-time properties.

Pixhawk 4

The pixhawk 4 is chosen as the MCU for this project. The pixhawk 4 is designed as drone flight controllers and is based on the Pixhawk-project FMUv5 and runs on NuttX Real-Time Operating System (RTOS). It has a STM32F765 main processor and a designated STM32F100 IO processor. The pixhawk has 16 PWM output ports, five serial ports, three I2C ports, two external SPI buses, two analog inputs, and up to two CANBuses. The controller comes with a power module that regulates the voltage to the controller while being able to output higher voltages and currents to actuators. Built into the controllers are two high-performance, low-noise inertial measurement units.

Matlab offers a support package called "Embedded Coder Support Package for PC4 Autopilots". The support package comes with Simulink blocks for reading sensor data, generating PWM signals, log data to SD-card, communication, and reading and writing data to and from uORB topics used for internal communication in the pixhawk. The package generates C++ code from a Simulink model, and uploads it to the controller. This is a simple way to create custom code, running with real-time constraints on the controller. The controller can run on external mode on Simulink. In external mode, the uploaded code runs on the pixhawk, and the data is sent to the host computer, where the desired parameters can be displayed and tuned.

The proposed design requires three PWM output ports for the motors and input ports for position sensors. The pixhawk offers a wide selection of interfacing possibilities and PWM output ports, which should be more than enough to interface with the actuators and sensors required for the robot-cat. The pixhawk also has two built-in IMUs, which eliminates the need for an external gyroscope.

3.2.4 Rotary encoders

In this section, different rotary encoders will be evaluated, and a suitable sensor will be chosen. A suitable encoder has low noise, high resolution, is compact and low cost.

As discussed in section 2.1, encoders are divided into two main categories: incremental and absolute encoders. The controller must be fully aware of the orientation of the relative position between the outer and inner parts of the cat. Absolute encoders can provide this information on start-up, while incremental encoders require three channels and homeing with the index pulse before the position is known. Both are, therefore, suitable.

As discussed in section 2.1, the physical resolution is the smallest difference in angle or angular speed the encoder can measure. It is, therefore, directly linked to the smoothness of the feedback signal. To ensure a smooth control signal to the actuators and overall accuracy, it is, therefore, important that the resolution of the encoder is sufficiently high.

The velocity resolution of an encoder is calculated using equation 2.4. It is desirable to get at least 100 readings in the duration of the fall of 0,5 $[s]$. The differentiation time is, therefore, 0,005 $[s]$.

In figure 3.13, the physical velocity resolution is plotted as a function of the counts per revolution (CPR) for encoders. A scale of the percentage of maximal speed has been added to the plot. This is used as a reference for relative significance for this particular application. As can be seen, the

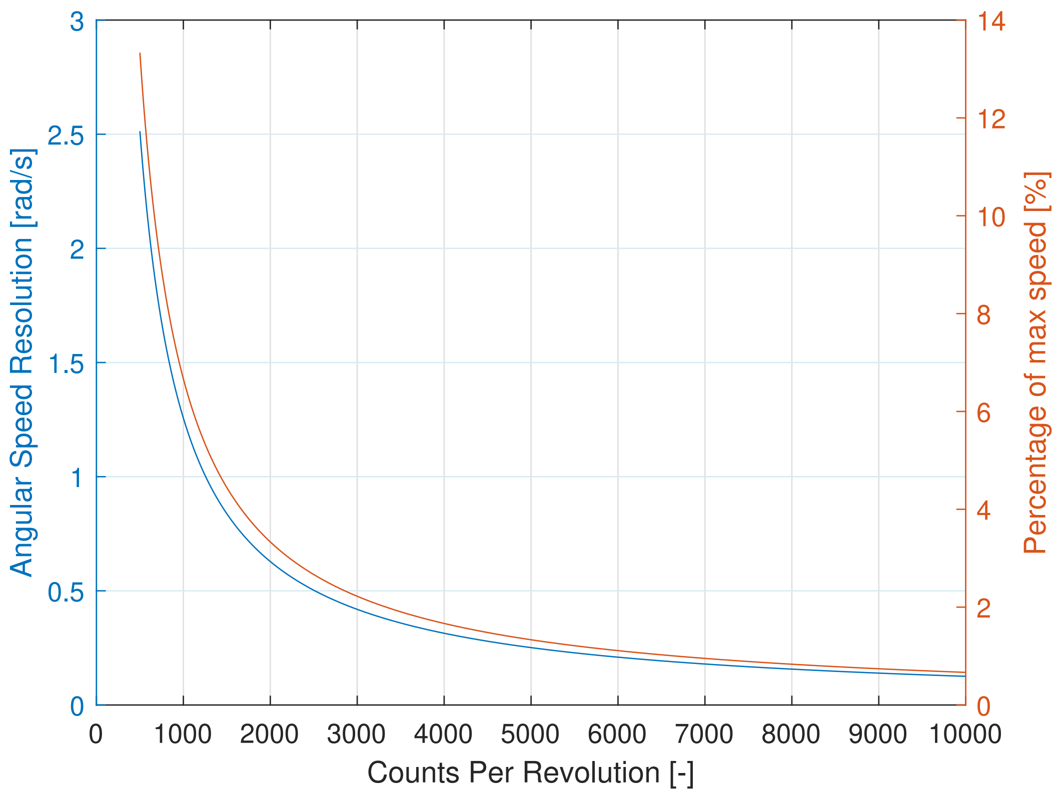


Figure 3.13: Velocity resolution evaluated for a time interval of 5 $[ms]$

velocity resolution drastically increases in the lower CPR region but stabilizes as the CPR increases. Between 500 and 4000 CPR, the resolution changes 11,7 [%] of the maximal speed, while between 4000 and 10000 CPR, the resolution only changes 1,0 [%]. A resolution in the region of 4000 CPR is therefore desirable.

Small, low-cost encoders with sufficient resolution are hard to find. No suitable option is found for absolute encoders. Broadcom HEDM-5540 is a three-channel incremental optical rotary encoder with 1000 CPR. In quadrature mode, this is quadrupled, thus satisfying the resolution requirement. The operating voltage is 4,5-5,5 $[V]$, and the encoder supports rotational speeds of up to 30000 $[RPM]$. The cost of the encoder is 637,5 [NOK]. Other incremental rotary encoders with the same resolution was significantly more expensive, larger, and heavier. The HEDM-5540 is therefore chosen.

The encoder outputs a quadrature signal, which is not supported by the pixhawk. With a rotational speed of 180 $[RPM]$ and 4000 CPR, the encoder outputs 12000 pulses per second. This increases the workload of the MCU. It is, for these reasons, beneficial and necessary to use a quadrature encoder buffer along with incremental encoders. SuperDroid Robots Dual LS7366R Quadrature Encoder Buffer is therefore acquired. The quadrature counter has two channels, which counts the pulses from the encoders and stores them in a 32-bit buffer. The quadrature counter has an operating voltage of 3-5 $[V]$ and communicates over SPI, which is compatible with both the pixhawk and the encoders. The price of the quadrature counter is 562 [NOK], with a dollar to NOK exchange rate of 9,83 and a customs fee of 25[%].

3.2.5 Mechanical Components and Power Supply

In this subsection, the remaining components are selected. The remaining components are the outer pipe, inner tube, bearings, and the power supply. Stress calculations for the pipe and tube are not done because the drop of 1,3 [m] is considered to be small compared to the dimensions of the components. Sizing bearings are usually done by calculating the desired lifespan based on load and expected use. Because of the nature of the load and the intended use, this is not considered to be a reasonable approach for this application. The selection of mechanical components is, therefore, primarily based on assumptions about what reasonable parameters are for each component and availability from local vendors.

Outer pipe

It is desirable than the outer pipe has high strength properties, is lightweight and low cost. Metallic and aluminum tubes are not considered because of their weight, it is, therefore, desirable to use a plastic pipe.

High-density polyethylene (HDPE) is generally used for pressure pipe systems for drinking water, gas distribution, industrial applications, et cetera. The pipes have high impact strength, high tensile strength, excellent machinability, high abrasion-resistance, and low weight compared to other pipes used for the same purposes [19]. It is for these reasons expected to be sufficiently strong and used as the outer pipe for the robot-cat.

These pipes are not sold in retail, so a pipe vendor is contacted. The vendor, Hallinplast AS, provided a pipe of choice, free of charge.

The pipe has an inner diameter of 0,110 [m] and an outer diameter of 0,125 [m].

Table 3.14: Material properties PE100 pipe [1]

Parameter	Value	Units
Density	960	[kg/m^3]
Tensile strength at yield	25	[MPa]
Tensile strain at yield	>600	[%]
Tensile modulus	1100	[MPa]

Inner tube

The inner tube should be lightweight and sufficiently strong, with an inner diameter significantly lower than the outer pipe. As it has a significantly lower diameter, it is more sensitive to stress than the outer pipe. Plastic tubes are, for this reason, considered too weak for this application, and aluminum tubes are used as the inner tube for the cat-robot.

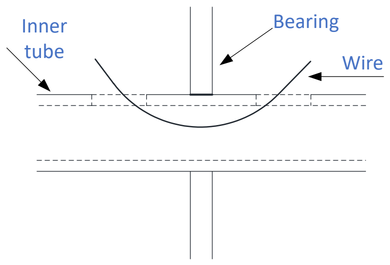


Figure 3.14: Passage for wire though rotating parts

Utilizing a tube, rather than a solid rod offers a passage for wires through the rotating wall connecting the outer pipe to the inner tube, as shown in figure 3.14.

An aluminium tube with inner diameter 0,016 $[m]$ and outer diameter 0,020 $[m]$ is identified as a suitable option. The cost of the pipe is 163 NOK for a length of one meter, which should be sufficient.

Bearings and encoder shaft

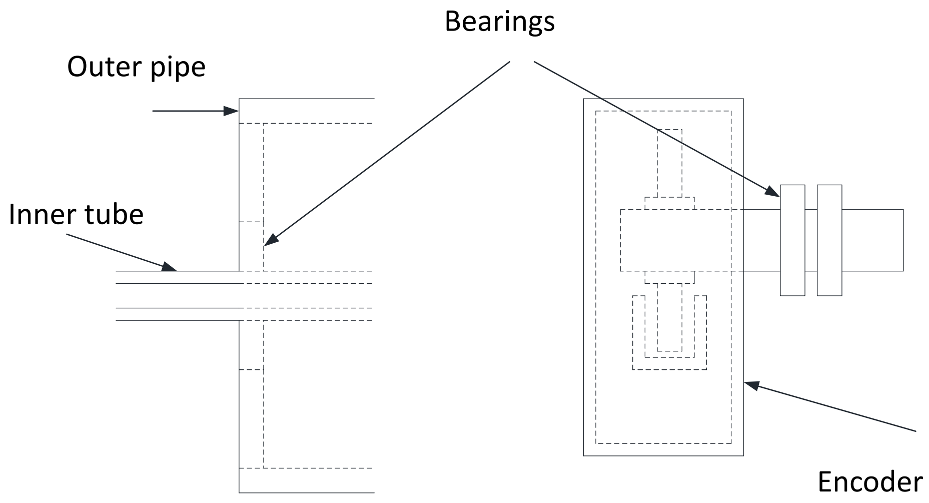


Figure 3.15: Required bearings

Bearings are used to reduce friction between parts rotating relative to each other. For the robot-cat two sets of bearing required, one set for connecting the outer assembly to the inner tube and one set for connection to the shaft used for the encoder as illustrated in figure 3.15.

Usually, calculations would need to be done in order to press-fit the bearings to assembly. However, as proto-typing might require multiple iterations, the components need to be easily assembled and disassembled. The bearings are, therefore, dimensioned with the same inner diameter as the outer diameter of the part they are attached. Based on production tolerances, the bearings will either barely or barely not fit. If they do not fit, it is assumed that minor processing is required.

To prevent the bearings from sliding on the shaft or tube, small dents are made, as illustrated in figure 3.16. This creates a softer connection than the interference fit, and should be sufficient to keep the bearings in place for the prototyping of the robot-cat.

For the connection between the inner tube and outer pipe, four Biltema 6004 2RS bearings are

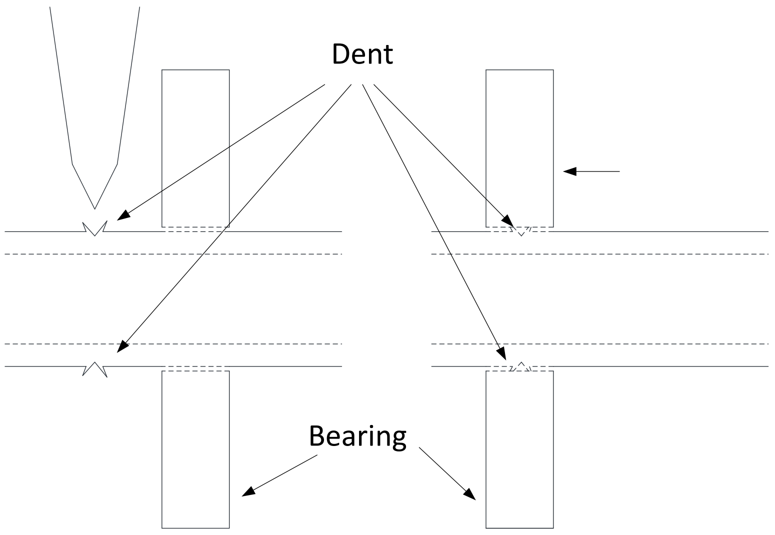


Figure 3.16: Fitting bearings

used. The bearings have an outer diameter of 0,042[m], inner diameter of 0,020 [m] and a width of 0,012 [m].

For the support of the encoder shaft, four SER-1397 Ball bearing 8x12x3,5 SS are used. It is decided to use two bearings per encoder as the encoder shaft is not supported, as shown in figure 3.15. The additional bearing is intended to reduce play in the shaft connection.

Power supply

The power supply must be able to provide sufficient voltage and current to drive the two servo motors and be within the supply voltage range of the power module connected to the pixhawk.

The pixhawk power module accepts voltages in the range 7-50 [V], and the servo motors have an operating range between 6-8,4 [V]. The supply voltage should, therefore, be in the range 7-8,4 [V].

Futaba does not specify any current requirements for the power supply. However, the internet retail store VDWModels.nl states that for two motors, the supply current should be able to output 40-45 [A] with a capacity of 2 [Ah], and 80-100[A] and 4 [Ah] for five motors. It is desirable to open for the possibility of adding a motor at a later stage for actuating the spine joint. The supply current should, therefore, be in the range 50-80 [A] and 2-4 [Ah]. VDMModels also recommend a Lithium Polymer (LiPo) battery as power supply.

An Eton LiPo 2S 35C battery has a capacity of 2,2 [Ah], a supply voltage of 7,4 [V]. The battery weighs 0,116 [kg], has sufficient capacity, is within the required voltage level, and can output a continuous current of 77 [A]. This is sufficient to power three motors, the pixhawk, and the encoders, and is therefore acquired. The cost of the battery is 206 NOK.

3.3 Implementation and Design

This section will cover how the selected components are implemented in the design of the robot cat and the necessary steps required to achieve communication between the devices.

3.3.1 Interfacing

The Pixhawk is the center of center of the electrical components. It processes the data from the sensors and produces an actuation signal for the servo motors. This subsection will cover the steps required for the Pixhawk to receive encoder data, and actuate the servo motors.

Pixhawk

Although the Pixhawk 4 has a connection for SPI communication, the "Embedded Coder Support Package for PX4 Autopilots" used to generate and upload code to the Pixhawk does not have a driver for it. The Pixhawk does, therefore, not have a way of directly communicating the encoder buffer. The only external communication form supported by the support package is serial communication over UART. It is for this reason necessary to implement a device able to communicate with both the encoder buffer and the Pixhawk. Arduino NANO is chosen for this purpose. The Arduino Nano has a wide variety of interfacing possibilities, including SPI and UART, it weighs 0,007 [kg], with dimensions of 0,018 by 0,045 [m]. Pixhawk 4 Mini offers the same features as the Pixhawk 4, except it has fewer interfacing possibilities. As the "Embedded Coder Support Package for PX4 Autopilots" does not support the additional interfacing possibilities of the Pixhawk 4 it is decided to use the Pixhawk 4 Mini.

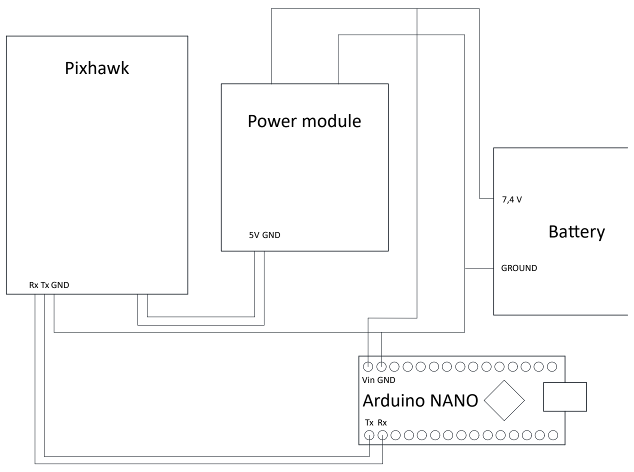


Figure 3.17: Connection between the Pixhawk 4 mini and the Arduino

As shown in figure 3.17, the Pixhawk is connected to both a power module and the Arduino. The power module is required, as the battery voltage of 7,4 [V] is above the input range of the Pixhawk of 4,75-5,5[V]. The power module converts voltages from 7-42 [V] into outputs between 5,1-5,3 [V]. The connection to the Arduino is configured for UART communication. The Tx and Rx pin from the dev/ttyS1 port on the Pixhawk is connected to the Rx and the Tx pin on the Arduino. A ground connection is also added.

Quadrature counter

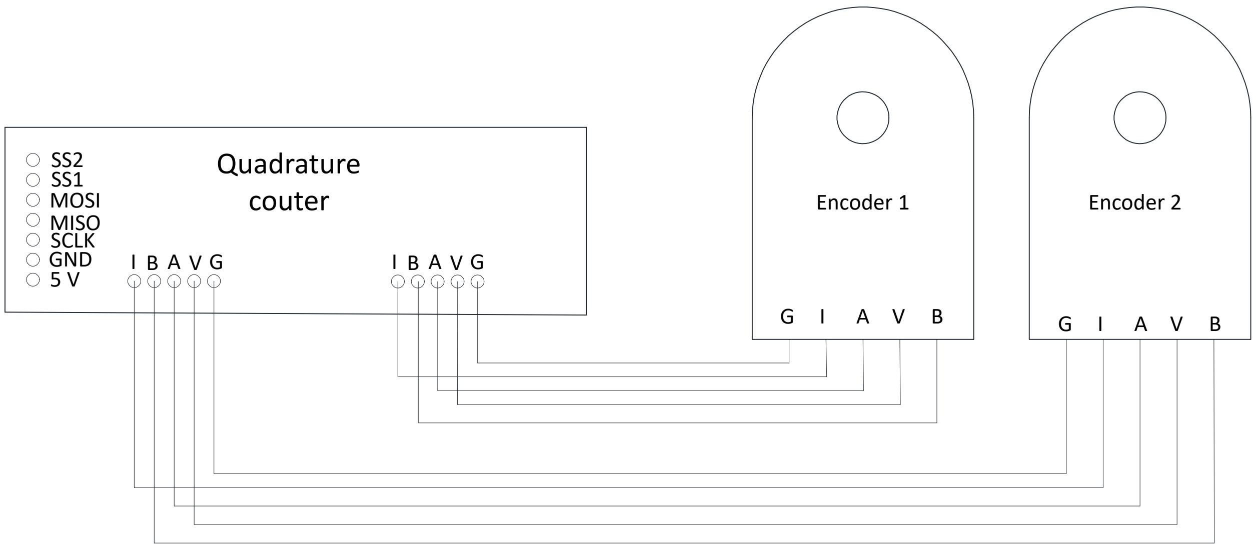


Figure 3.18: Connection between encoders and quadrature counter

The encoder pulses are read directly by the quadrature counter, the connection between them can be seen in figure 3.18. The board is equipped with two LS7366R 32-bit quadrature counters with SPI interface, and can, therefore, keep track of two encoders at the same time. The encoder A, B, and I channels are directly connected to the counter, where the count can be stored as the free count, upper and lower limits can be set, or be configured to a reset at the desired value. The quadrature counter offers a wide selection of user configuration options through different control registered.

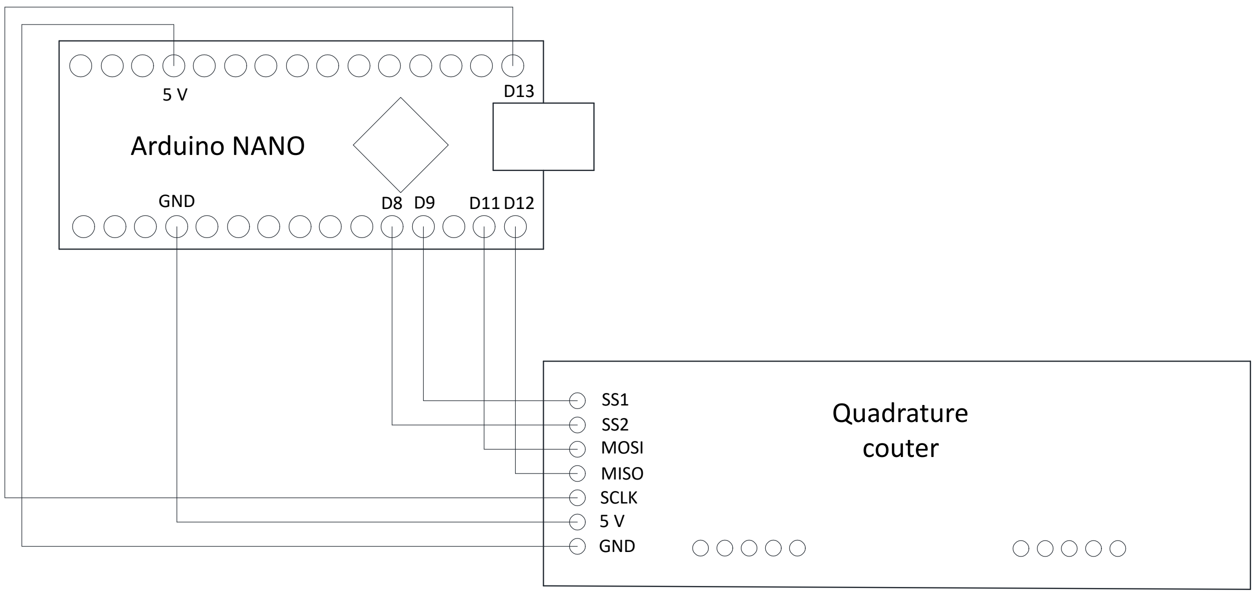


Figure 3.19: The connection between the quadrature counter and the Arduino

Communication with the quadrature counter is, as mentioned, done over SPI. The connection between the Arduino and the quadrature counter can be seen in figure 3.19. The Arduino is the master unit, and the quadrature counters, two LS3766R chips, are the slaves.

When communication is initiated with the quadrature counter, the first byte that is transferred is always the instruction byte. The two first bits in the instruction byte tells the counter whether the master wishes to read from, write to or clear a register, and the next three bits tells it which register. As this is all the instructions the counter needs, the three last bits do not provide any information and can be set arbitrarily. The available registers on the LS7366R chip are STR, DTR, OTR, MDR0, and MDR1. STR is the status register, which stores information about the direction, index latch, power loss, et cetera. The MDR0 and MDR1 registers are used to configure the counter modes, index pulse configuration, quadrature mode, and the number of bytes used to store the counter value. OTR is a register used to load the counter value into when the count is requested. This allows the count register to keep counting encoder pulses while the count is sent over SPI from the OTR. The DTR register can be used to specify counter limits or a modulo numer to reset the counter.

The datasheet for the LS7366R does not specify if a high or low encoder value activates the index pulse. Testing revealed that it is activated by a low value, while the encoder uses a high value for indexing. As the incremental encoders only count pulses and not absolute angle, having an index pulse in a fixed position which can be used to calibrate the position is a necessity. It is, therefore, desirable to use the index pulse to reset the counter. However, with the mismatch in polarity between the encoder and the buffer, this results in the counter being reset 999 out of 1000 counts per rotation, which is not a viable option. Additional testing revealed that the index latch in the status register is latched by high value. Thus, by clearing the status register upon startup, keeping the index pulse active until the index latch latches, then deactivating it allows calibration. This is not an ideal solution. The index pulse can, for this reason, not be active while counting. However, the count before the index pulse latches is not interesting; after the calibration is done, it has served its purpose.

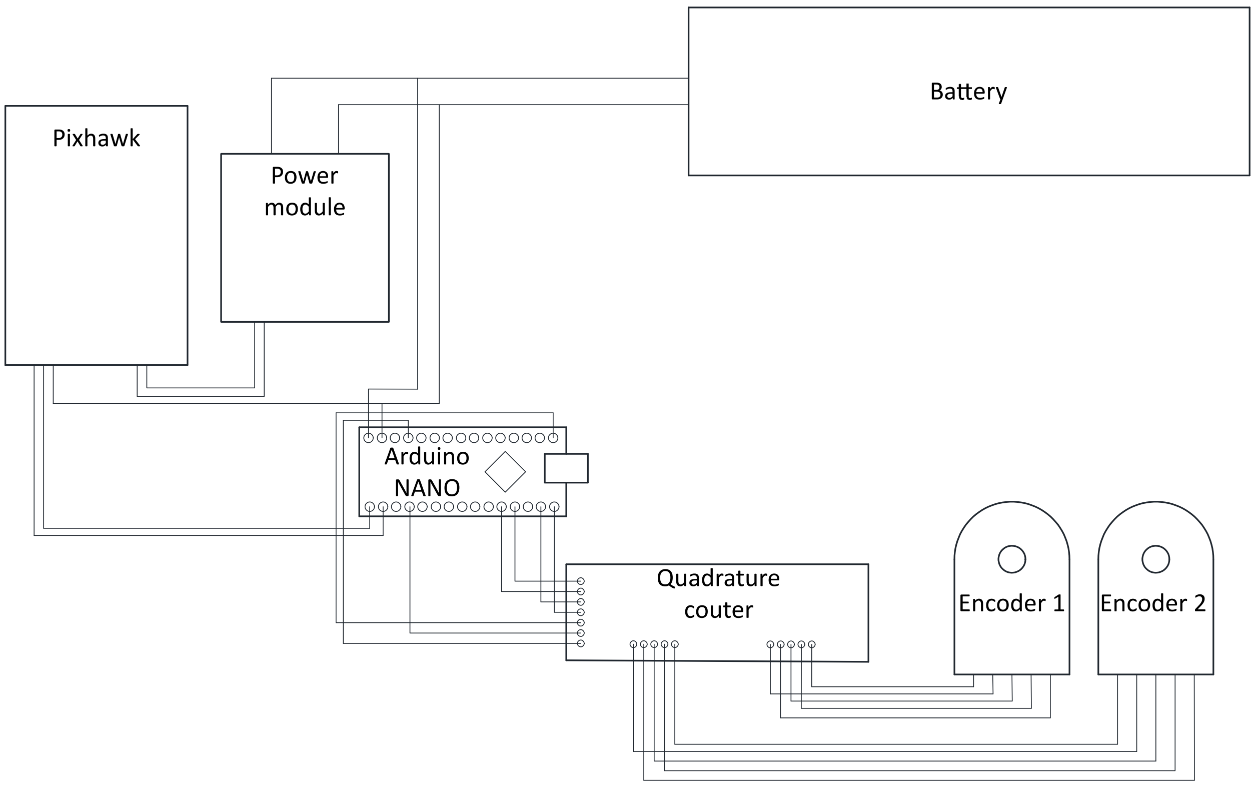


Figure 3.20: Connection between the encoders and the pixhawk 4 mini

3.3.2 Serial Communication

Adding the Arduino is not elegant but a necessary solution in order to get the encoder data to the Pixhawk without spending an unreasonable amount of time creating an SPI driver. However, it does require some extra steps. As the quadrature counter has a preset communication protocol, reading it is very straight forward. By using the instruction byte, both the quadrature counter and the user knows what actions are requested, and what information is transferred. The UART, on the other hand, has no such preset options. As mentioned in section 2.3.2, the UART only sends and receives data, what data is sent, and what data is received must, therefore, be known to both the devices. In this subsection, the communication necessary for the Pixhawk to request and receive encoder data is described.

The pixhawk first needs to send the Arduino an information request over UART. The Arduino then recognizes the request and initiates an SPI conversation with both LS7366R chips on the quadrature counter. The data from the counter is then sent back to the pixhawk over UART.

As long as the index pulse is activated on the quadrature buffer, the counter is continuously reset. The index pulse can, therefore, only be used for calibration purposes. This means that two modes are required, one for calibration, and one for measuring the position.

Pixhawk

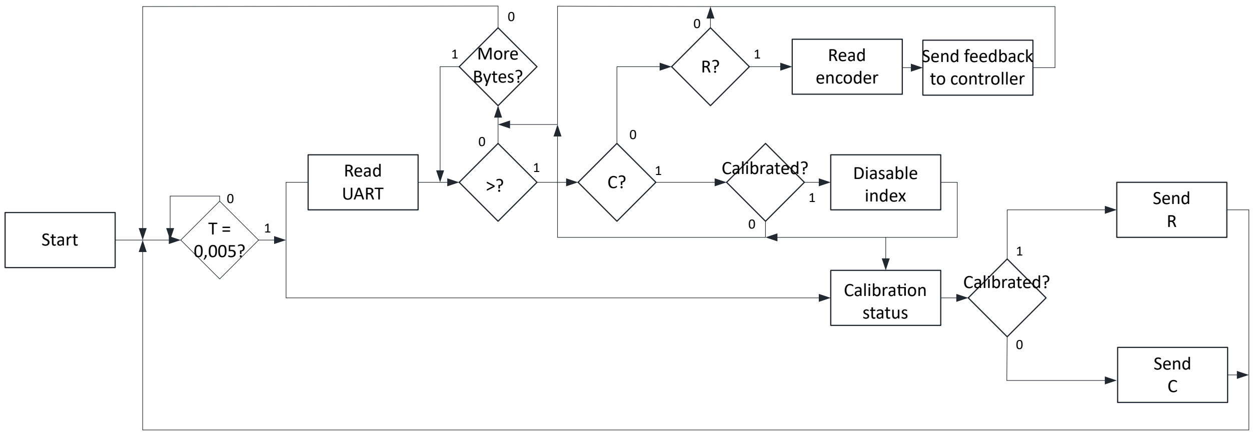


Figure 3.21: Flowchart for UART communication for the Pixhawk

The pixhawk initiates all communication. In figure 3.21, a flowchart of the communication protocol for the pixhawk is illustrated. As seen, the Pixhawk sends an information request to the Arduino and reads the UART receive buffer, every 0,005 $[s]$. There are two messages sent from the pixhawk to the Arduino, the integer number representing the ASCII character 'R' is sent to request encoder data, and 'C' is sent to request calibration status. When starting up, the Pixhawk sends 'C' messages over the UART. When the Pixhawk receives information that both the encoders are calibrated, a calibration latch is set, and the Pixhawk switches to sending 'R' requests.

When reading the UART, the pixhawk searches through the receive register until it finds the number representing the ASCII character '>', indicating the message is intended for the Pixhawk. The following character indicates if it is a calibration status or an encoder value, 'C' is used to indicate calibration status, and 'R' indicates encoder data. If the message is a calibration status, the next two bytes contain the calibration status for encoder one and two. If the message is encoder data, the next four bytes contain the encoder count for encoder one, and the four after that, the count for encoder two. To make the four-byte encoder value usable, they are written into a 32-bit integer by using bit shift operators. If the UART message does not contain the specified characters, the Pixhawk waits until the next iteration.

Sending data to the serial port of the Arduino is a useful tool to troubleshoot when writing code, however, the data is also sent over the UART to the Pixhawk. By adding the characters at the start of the transmissions, the Pixhawk filters out the unwanted data. It also knows what information is received and how to process it into useful values.

Arduino

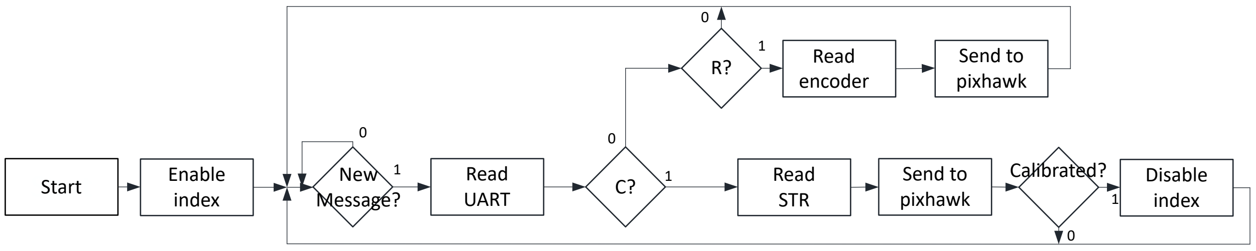


Figure 3.22: Flowchart for serial communication for the Arduino

As the Pixhawk controls the timing of the communication, the Arduino only responds to incoming requests. The Arduino, therefore, continuously checks for incoming UART messages. There are two valid requests, 'C' for calibration status, and 'R' for encoder data. As illustrated in figure 3.22, when starting up the Arduino enables the index pulse on the quadrature buffer, it also clears the counter register and the status register. If the instruction, 'C', is received, the Arduino reads the status register for each encoder from the quadrature counter via SPI. The STR consists of one byte, where each bit indicates a particular status. The AND operator is used to check if the index latch bit is active. This returns a 2^n value, where n is the placement of the index bit in the byte, or zero. Zero indicates that latching has not occurred, and 2^n indicates that the index is latched. The characters '>' and 'C', followed by one or a zero indicating latched or not latched for each encoder, is then sent to the pixhawk over the UART. When the index latch is activated, the index pulse for the specific encoder is disabled, such that the counter starts counting the pulses even if the other encoder is not yet calibrated.

If an 'R' is received, the counter value is read from the quadrature counter over SPI. The quadrature counter stores the value as a 32-bit integer. The SPI uses 8-bit shift registers, and the encoder count is, therefore, sent as four separate bytes. As the Arduino has no interest in the actual value of the encoder, '>' and 'R' followed by the four bytes per encoder, are sent directly to the pixhawk over UART.

3.3.3 Design and Implementation

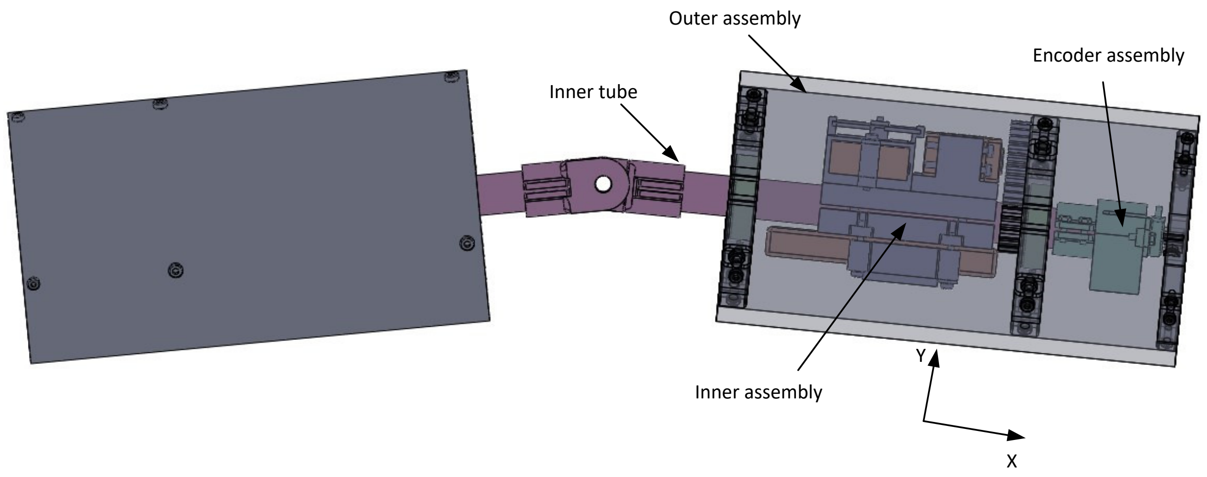


Figure 3.23: The design for the robot cat

In this subsection the design and implementation of parts of the robot-cat is discussed. In figure 3.23, the design for the robot-cat can be seen. The design is built up of two parts, symetric about the revolute joint joining them in the middle. Each half has one inner tube, one outer assembly, one inner assembly and one encoder assembly.

3.3.4 Encoder Assembly

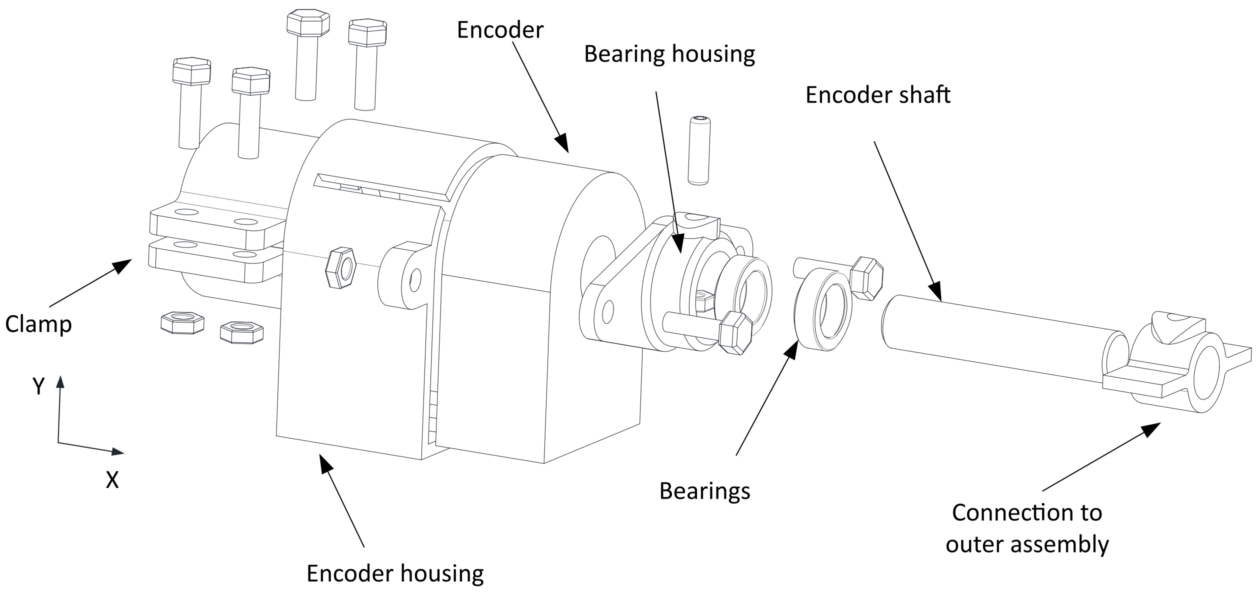


Figure 3.24: The encoder assembly

The encoder assembly is designed to allow the encoder to measure the angle between the inner tube and outer assembly of the robotic-cat. The encoder assembly consists of a clamping mechanism used to attach the assembly to the tube, an encoder, an encoder housing, an encoder shaft, a centering device, and two bearings, as seen in figure 3.24.

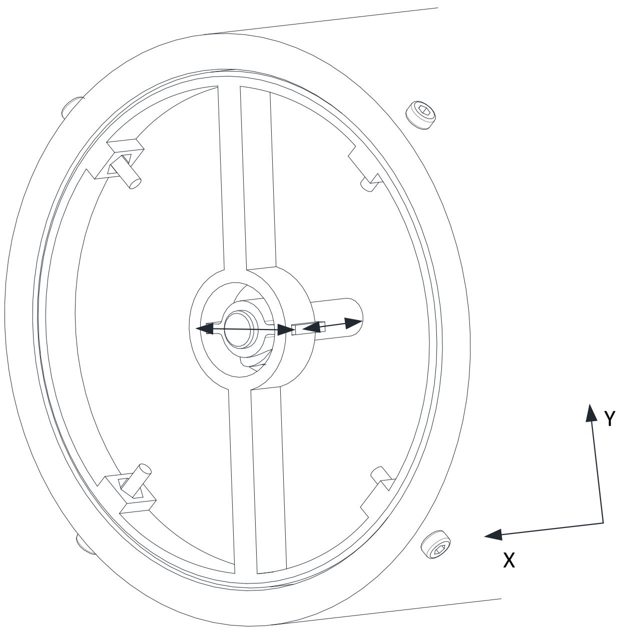


Figure 3.25: Allowed movement

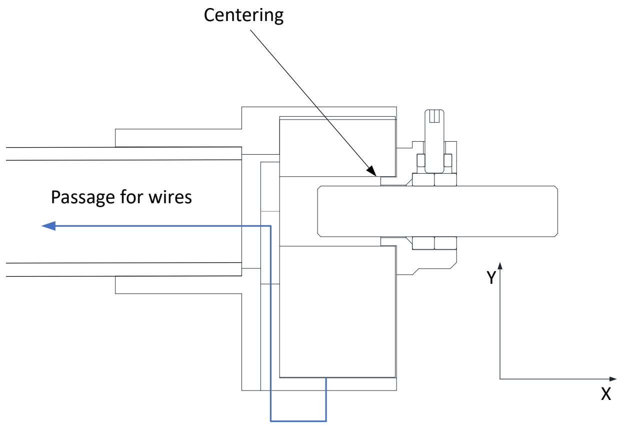


Figure 3.26: The centering of encoder shaft and passage for wires

The wall where the encoder shaft enters the encoder has three holes meant for attaching the encoder directly on a shaft that is expected to be supported. This attachment option is not feasible for this purpose. An encoder housing is therefore designed to keep the encoder in a fixed position. The encoder housing has a clamping mechanism to attach to the inner tube. The wire output is on the bottom of the encoder, to allow passage of wires, the housing is designed with space for wires between the encoder and the back wall. This allows the wires to pass into the inner tube, as can be seen in figure 3.26.

The code wheel of the encoder needs to be centered in the encoder to ensure proper performance. The encoder does not have an option for centering, so a centering device is designed. The centering device is connected to the encoder housing on both sides of the encoder. As the shaft entrance on the encoder has a larger diameter than the shaft, the centering is designed to fill the difference, as shown in figure 3.26. The centering device is also designed with two roller bearings. The bearings are intended to reduce friction and to support the weight of the encoder shaft.

Most of the parts used to fit the components together are 3D printed, some inaccuracies are, therefore, expected. These inaccuracies cause unwanted misalignment between the encoder shaft and the connection to the outer assembly, increasing the stress on the shaft support. To allow movement between the shaft and the outer assembly, the encoder shaft is connected to the outer assembly by a plate. The plate fits into a track in the outer assembly. As this is not a fixed connection, some movement can occur between the two parts, as shown in figure 3.25.

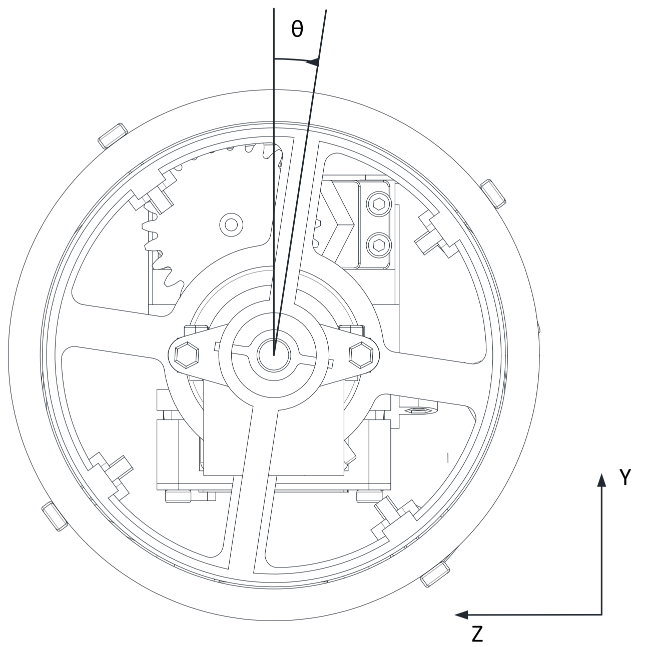


Figure 3.27: The angle measured by the encoders

As the encoder assembly is fixed to the inner tube and the outer assembly can rotate freely. The encoder measures the relative angle between the inner tube and the outer assembly, as shown in figure 3.27.

Inner Assembly

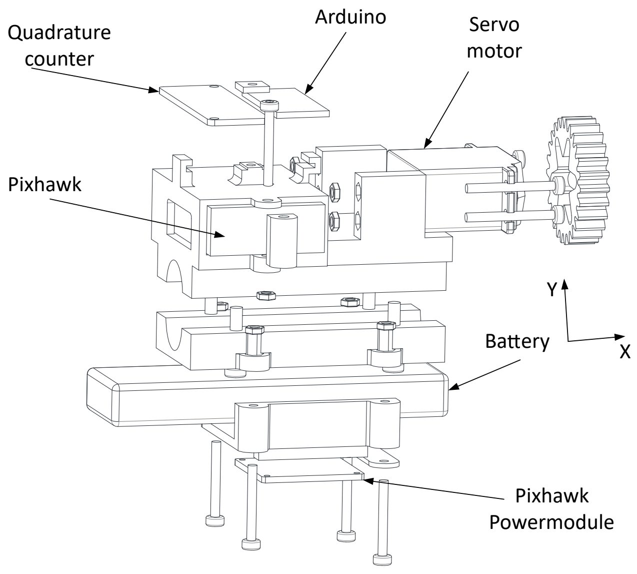


Figure 3.28: The robot

The inner assembly is designed to fit the servo motor, Pixhawk, power module, Arduino, quadrature counter, and the battery to the inner tube, as shown in figure 3.28. The motor and Pixhawk is placed on the opposite side of the inner tube of the battery to obtain better weight distribution.

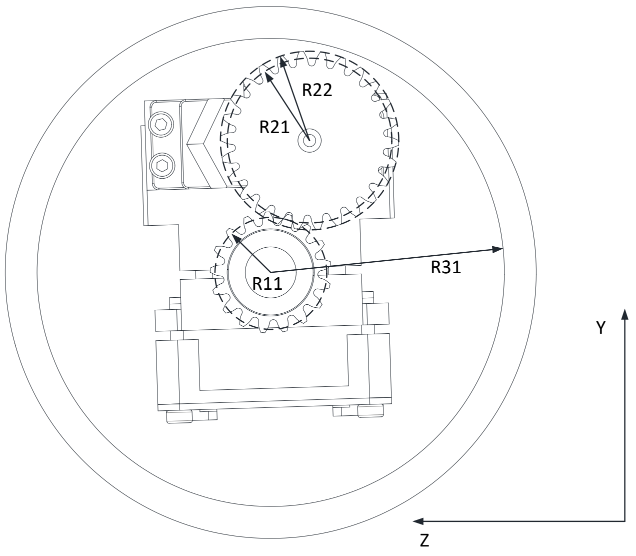


Figure 3.29: The dimensions for gear design

Figure 3.29 shows the gearing between the motor and the outer assembly. As the servo motor torque is significantly larger than the estimated requirement, and the speed of the motor is below, the gearing aims to increase the speed and low the torque. This is done by designing the cogwheel of the motor larger than that of the outer assembly.

$$R31 > R11 + R21 + R22 \tag{3.16}$$

Where R31 $[m]$ is the inner diameter of the outer pipe, R11 $[m]$ is the pitch radius of the cogwheel connected to the outer assembly, R21 $[m]$ is the pitch radius, and R22 $[m]$ is the radius to the tip of the teeth of the cogwheel connected to the motor. The gearing is constrained by the inequality described in equation 3.16.

$$n = \frac{Z_m}{Z_o} = \frac{26}{17} = 1,53 \tag{3.17}$$

Where n is the gear ratio, Z_m is the number of teeth of the motor, and Z_o is the number of teeth of the cogwheel connected to the outer assembly [14]. As the motor torque is far above the estimated requirement, the aim is to create a large gear ratio.

Assuming reasonable gear parameters, the gear dimensions are calculated using otvinta.com's involute gear calculator, using the parameters: Z = 17 and 26, α = 20, X = 0, m = 1.5 [13]. This gave the results R11 = 0,01275$[m]$, R21 = 0,0195 $[m]$, R22 = 0,021 $[m]$. R31 is given by the inner radius of the outer tube, thus R31 = 0,055 $[m]$. The right hand side of equation 3.17 is thus 0,05325 $[m]$, satisfying the inequality.

The gearing reduces the torque from the motor acting on the outer assembly but increases the speed by a factor of the gear ratio.

$$T_o = \frac{T_m}{n} = \frac{4,4[Nm]}{1,53[-]} = 2,9[Nm] \tag{3.18}$$

$$\omega_o = \omega_m n = 143[\frac{rad}{s}] \cdot 1,53[-] = 218,79 \tag{3.19}$$

Where T_o $[Nm]$ torque acting on the outer assembly from the motor, T_m $[Nm]$ is the motor torque, ω_o $[RPM]$ angular speed of the outer assembly, ω_m $[RPM]$ is the angular speed of the motor. Note that both the output torque and the output speed now satisfy the estimated speed and torque requirement of 1,0136 $[Nm]$ and 180 $[RPM]$.

Outer Assembly and Inner Tube

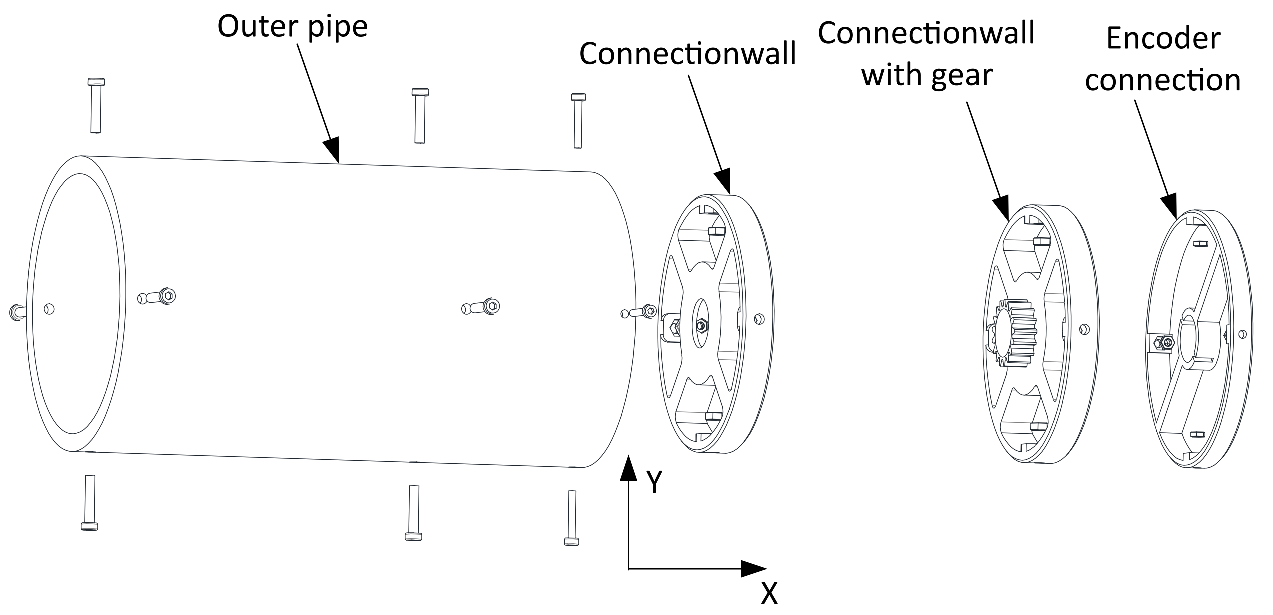


Figure 3.30: The outer assembly

The outer assembly is designed to allow rotation between the inner tube and the outer pipe. The assembly is connected to the inner pipe by two connection walls with bearings, as shown in figure 3.30. One of the connection walls have a gear, which is used to transmit torque from the servo motor to the outer assembly. The thrid connection wall is designed with a connection to the encoder shaft, allowing the encoder to measure the angle between the inner tube and the outer pipe.

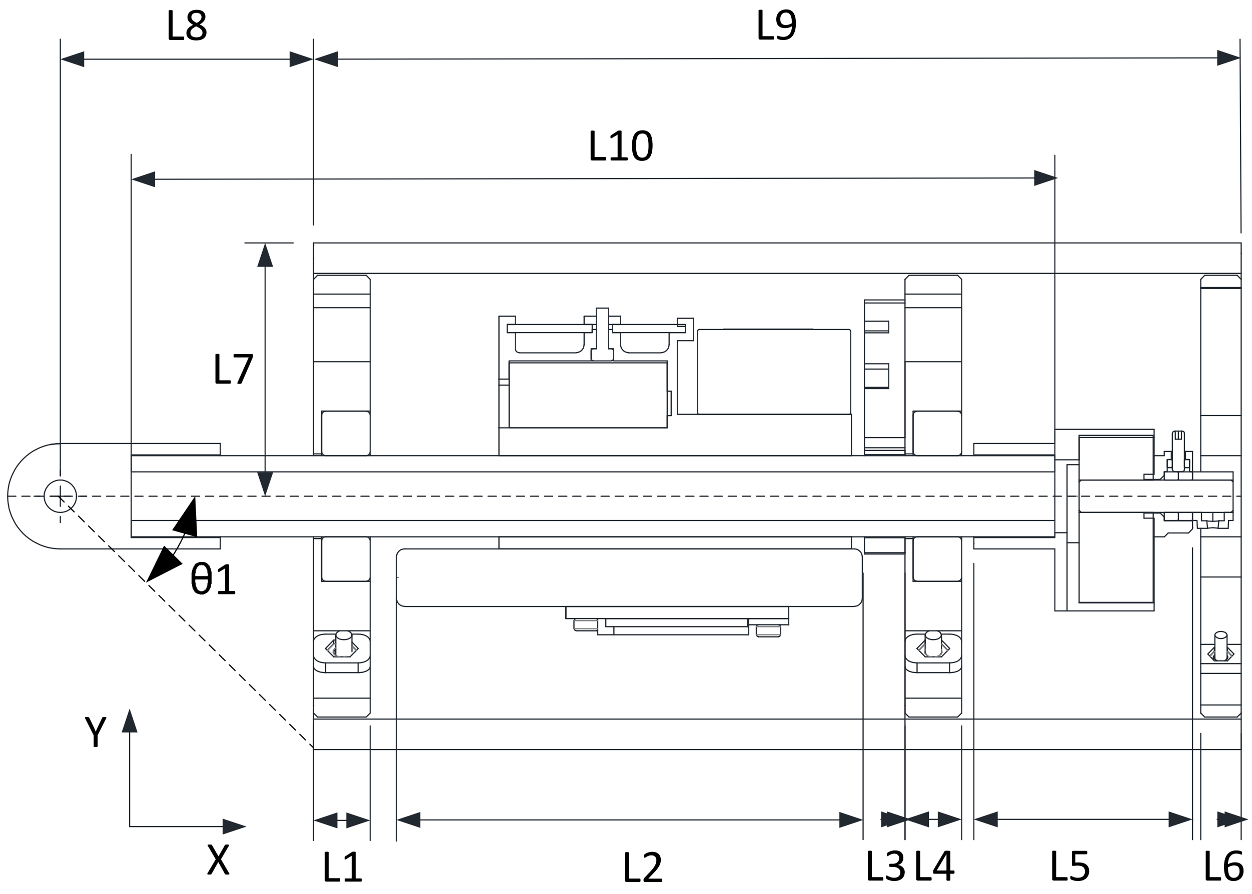


Figure 3.31: The dimensions dictating the length of the outer assembly

Table 3.15: Dimensions

Length	Notation	Value	Units
Thickness connection wall	L1	0,0140	$[m]$
Length of battery	L2	0,1130	$[m]$
Thickness gear	L3	0,0100	$[m]$
Thickness connection wall	L4	0,0140	$[m]$
Length of encoder assembly	L5	0,0540	$[m]$
Thickness connection wall to encoder	L6	0,0100	$[m]$
Diameter outer tube	L7	0,0625	$[m]$
Length from center joint to outer assembly	L8	0,0625	$[m]$
Length of outer tube	L9	0,2290	$[m]$
Length of inner tube	L10	0,2280	$[m]$
Maximum bend angle	$\theta 1$	45,0	$[°]$

Figure 3.31 show the dimensions used to determine the length of the outer pipe and the inner pipe, the values can be read in table 3.15. L1-L6 pluss a small gap is used to determine L10, which is the length of the outer tube. As can be seen, the length is largely determined by the length of thickness of the connection walls, the length of the battery and the length of the encoder assembly. The inner pipe is determined by the length from the start of the encoder assembly and the bending angle of 45°.

3.4 Servo motor

In this section, the servo motor is modified to allow continuous rotation. The input range of the PWM signals for the motor also has to be calibrated to obtain better controllability.

3.4.1 Modification

In this subsection, the steps required to get continuous rotation out of a servo motor is explained. As mentioned, the hobby servo motors, such as Futaba HPS-700, are usually constrained to operate in a specific range of angles. This is due to a physical restriction placed on the gears of the servo motor and upper and lower limits placed in the control system. The feedback system is tuned to respond to position input references with position feedback from a potentiometer.

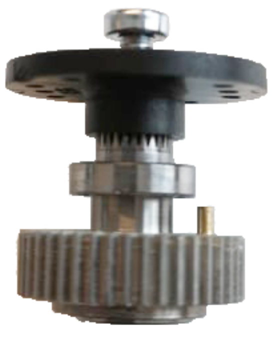


Figure 3.32: Gear with restriction rod

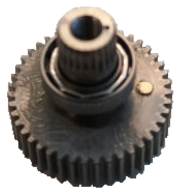


Figure 3.33: Gear without restriction rod

The first step is to disassemble the servo motor. In figure 3.32 the physical blocking can be seen. This is a rod placed on the gear, rotating the gear too far in any direction causes the rod to hit the other gear, thus blocking the movement. The rod is therefore removed by filing it away, as shown in figure 3.33.

The servo motor uses the potentiometer as feedback. As mentioned in section 2.1.1, the potentiometer measures the displacement by sliding a slider arm over a resistive material. The difference in voltage corresponds to the displacement of the arm. Exchanging the potentiometer with a fixed resistance will, therefore, disable the feedback from the system. The control system will sense a voltage reference that does not change. The error of the controller and thus, the proportional gain will, therefore, increase with the input reference. The larger the input reference, the more the controller tries to correct for an error that does not change. The controller, therefore, becomes a speed controller with no feedback.

Figure 3.34: Potentiometer

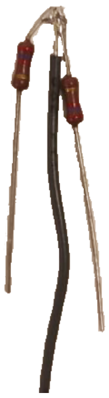


Figure 3.35: Fixes resistance

The potentiometer is for this reason exchanged for two resistors series, and an output wire, as shown in figure 3.35. The total resistance over the potentiometer is 5,2 [$k\Omega$], while the only available resistors were 2.7 [$k\Omega$]. Connected in series, that becomes a total resistance of 5.4 [$k\Omega$]. As the servo motor comes in a very compact housing, making a connection of multiple resistors is not an option. This might affect the zero reference of the input value, but in lack of a better option, it should suffice.

When the servo motor is reassembled, continuous rotation is enabled.

3.4.2 Motor calibration

The input to the servo motor is a PWM signal. The servo motor measures the duration of the on-time of the PWM signal and uses it as an input reference. The range of valid PWM inputs is generally between 1-2 [ms]. Within the input region, there is a deadband, a region where the input does not produce an output, and an upper and lower speed limit. To effectively control the motor, these parameters must be identified. In this subsection, the upper and lower speed input limit and the dead band is identified, and a function is created mapping an input between [-1,1] to the output range between the upper and lower speed limit while avoiding the deadband. This is done by performing a series of tests. In the first test, the deadband is identified and guess of the maximal positive and negative input. In the second test, a ramping function designed to induce positive and negative saturation sent to the input of the servo. A mapping function is then created. In the third test, the mapping function is tested by ramping to the positive and negative limits of the mapping function.

In figure 3.36 the test setup is shown. For convenience, only one half of the cat robot is assembled. The encoder and the servo motor are the only electrical components attached to the robotic cat. The battery, Pixhawk, Arduino, and quadrature counter are all external. The inner tube is extended and is held in place while the tests are performed. This is done to enable easier handling of the robot cat, and easier access to the Pixhawk and Arduino.

The first test reveals that an PWM on time of 1513 [μs] does not produce an output response from the motor, and the motor seems to stop increasing speed after 1513 $\pm$ 50 [μs]. This was observed

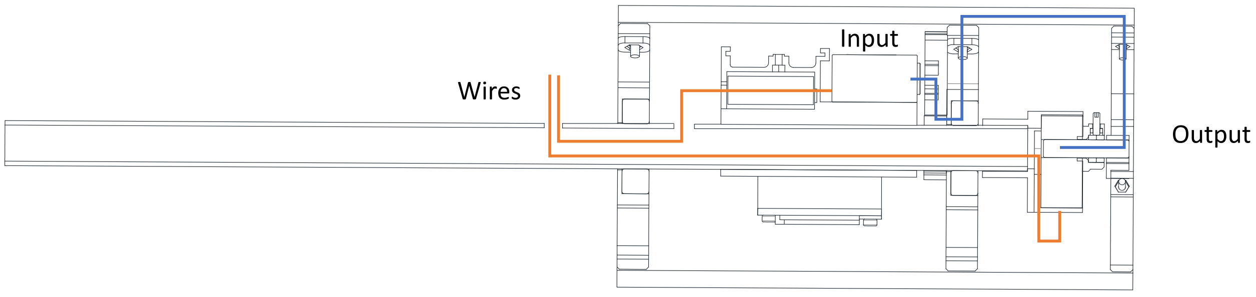


Figure 3.36: Test setup

by manually changing the input. This information is used to set the limits for the input ramp function used for the second test. The test is meant to drive the motor to saturation. The rate of change for the ramp is set to change the input ± 100 [μs] over 10 [s]. The sampling time for the encoder is $5ms$; this provides 20 readings per input point.

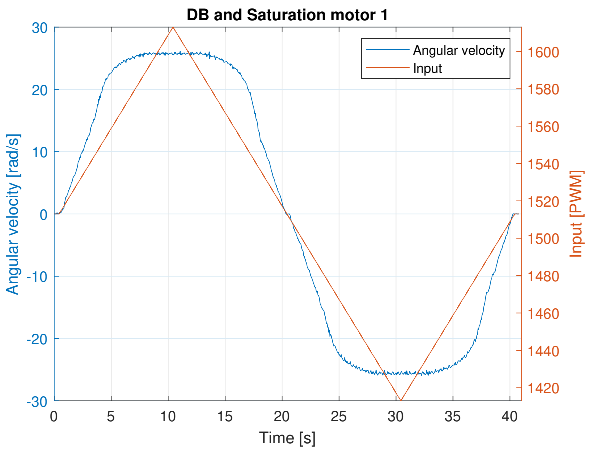


Figure 3.37: Uncalibrated input/output response of motor one

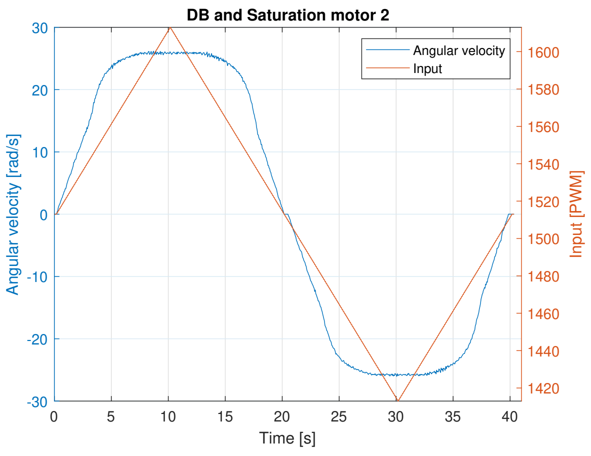


Figure 3.38: Uncalibrated input/output response of motor two

The results from the second test can be seen in figure 3.37 and 3.38. As can be seen, saturation occurs at 1585 and 1440 for motor one, and 1590 and 1438 for motor two. The figure also shows deadband is between 1514 and 1511 for motor one and 1514 and 1509 for motor two. A function is created to map the input between [-1,1] corresponding to the positive and negative regions for the input signal. The input is now changed from a PWM reference to a value between [-1,1] corresponding to the desired motor velocity.

The next test is performed to test the performance of the mapping function. The test is performed in the same manner. The input is ramping with a rate of change of ± 1 over 10 seconds.

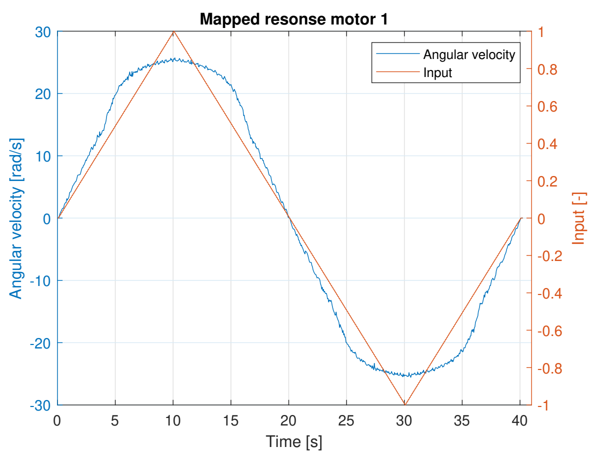


Figure 3.39: Calibrated input/output response of motor one

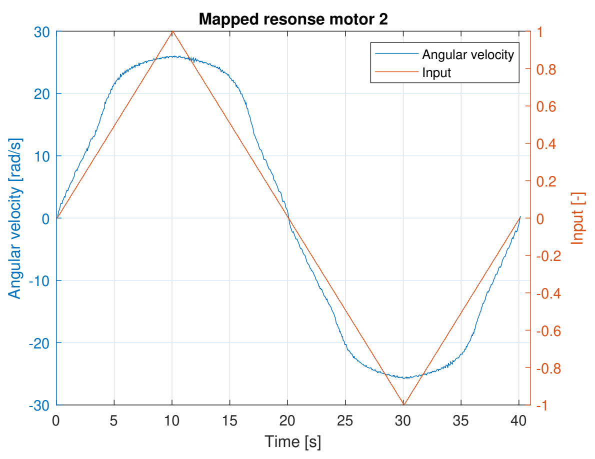


Figure 3.40: Calibrated input/output response of motor one

As can be seen from figure 3.39 and 3.40, the output velocity does not saturate and smoothly passes through zero, thus avoiding both saturation and the deadband. However, the output speed is far from linear relative to the input when approaching the max speed. This is not desirable as system identification requires a linear relationship between input and output.

To solve this, the least square approximation discussed in section 2.4.3 is applied. If the output from a linear input can be described as a polynomial, the same polynomial can be used to create a linear output. Two polynomial of the 10th degree, per motor, is approximated, one valid for the input range [0,1] and one for the input range [-1,0].

$$\mathbf{X} = \begin{bmatrix} \left(\frac{\omega_i}{\omega_{max}}\right)^{n} & \left(\frac{\omega_i}{\omega_{max}}\right)^{n-1} & \left(\frac{\omega_i}{\omega_{max}}\right)^{n-2} & \dots & \left(\frac{\omega_i}{\omega_{max}}\right)^{n-n} \end{bmatrix} \tag{3.20}$$

For linearizing output response, the output vector, **Y**, contains the input PWM signal, and the input matrix, **X**, contains the measured output speed divided with the maximal speed. Dividing the measured output speed on the maximum measured speed ensures that the signal is valid from [0,1]. This inverts the measured speed and used it as an input reference. The resulting function, therefore, creates a nonlinear output, which is canceled by the nonlinearities of the motor, thus creating an linear output to linear input.

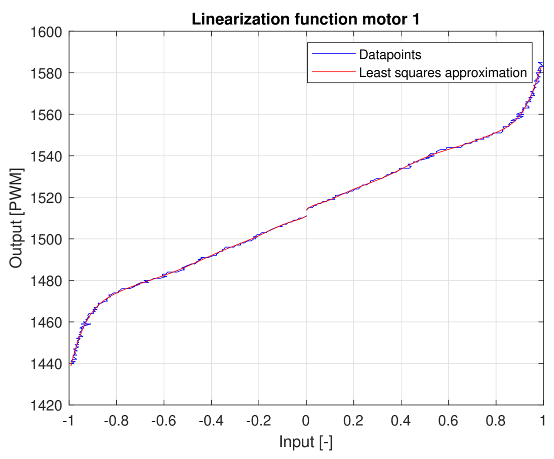


Figure 3.41: Least square approximation and datapoints for motor one

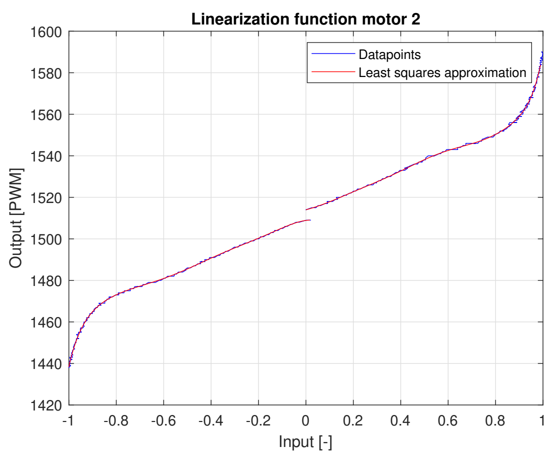


Figure 3.42: Least square approximation and datapoints for motor two

Figures 3.41 and 3.42 show the resulting mapping function. As see, the approximation fits the data points very well. Note that the output PWM signal is the input to the servo motor.

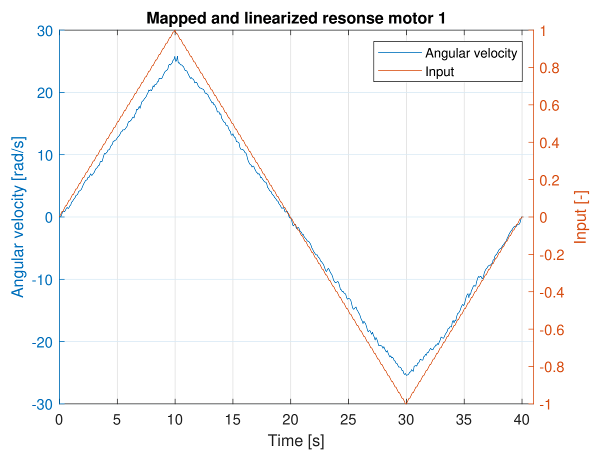


Figure 3.43: Linearized input/output response of motor one

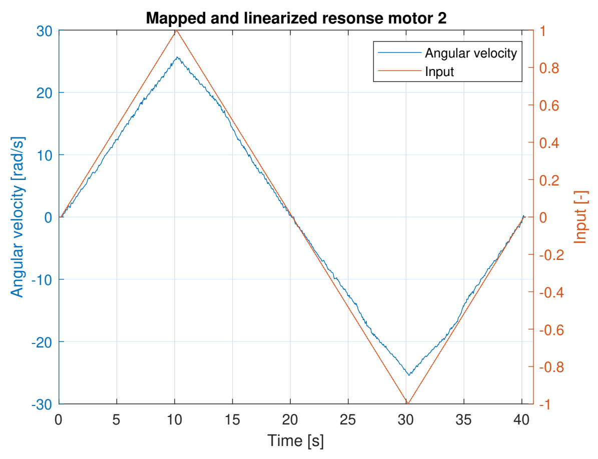


Figure 3.44: Linearized input/output response of motor two

Figures 3.43 and 3.44 show the input to the mapping function, and the measured output speed. As seen, the motor still avoids the deadband and saturation, but now also responds fairly linearly to the input signal. The motor is now calibrated and ready for use.

3.5 Control theory

In this section, system identification is performed on the actuation between the motor and the outer assembly. A transfer function is obtained from experimental data through blackbox frequency analysis, and a state-space controller is implemented.

3.5.1 Frequency response

When creating a controller for the motor, a transfer function is a useful tool, as it provides information about the system's response to an input. The transfer function can usually be estimated by evaluating the differential equations governing the system. However, the servo motor already has a control system and is only accessible through a PWM input. No information is provided by the manufacturer about the motor parameters, or the structure of the control system, making it near impossible to estimate a suitable transfer function from first principles. A blackbox frequency analysis is, therefore, performed. No prior information about the system is required, only that the control variable is measurable. As discussed in section 2.4.2, the frequency analysis is done by sending a sinusoidal input to the system and measuring the output. For LTI systems, the response will also be sinusoidal with a different amplitude, with a phase delay. This is done for a range of frequencies.

For low frequencies, the output should be able to follow the input signal, but as the frequency increases, the motor does not have time to react to the rapid change and starts lagging. At which frequencies this occurs depends on the system. The test is performed for fifteen frequencies in the range from 0,2 to 5π $[Hz]$. The frequencies are evenly spaced in the logarithmic scaled bode plot, as shown in figure 3.45.

The tests are performed with the same setup as the mapping of the servo motor. The log file contains a time stamped input and output signal, used for further processing.

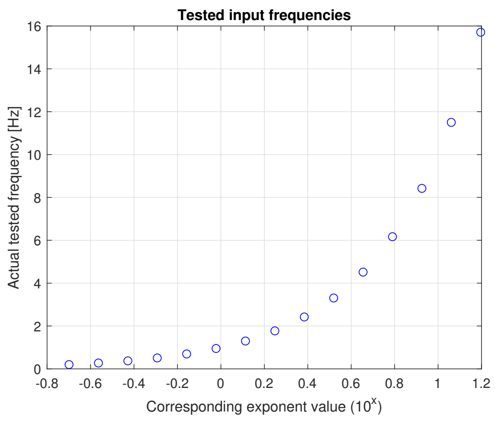


Figure 3.45: Visiualisation of tested input frequencies

Every tested frequency corresponds to a single point on the Bode plot. The bode plot requires the magnitude between the input and output amplitude, and the phase shift. This information has to be obtained from the logged files. By inspecting the logged file, it is observed that the output position consists of three parts: bias from the initial position, a ramp from non-ideal motor calibration, and the desired sinusoidal output response.

$$C_1 \cdot \cos(\omega t - \phi) = C_2 \cdot \cos(\omega t) + C_3 \cdot \sin(\omega t) \tag{3.21}$$

$$C_1 = \sqrt{{C_2}^2 + {C_3}^2} \tag{3.22}$$

$$\phi = \arctan\left(\frac{C_3}{C_2}\right) \tag{3.23}$$

Where ω is the frequency $[\frac{rad}{s}]$, t $[s]$ is the time, ϕ $[rad]$ is the phase and C_{1-3} are arbitrary constants.

In order to determine the phase and not only the amplitude of the signal, some additional processing is required. Equations 3.21 to 3.23, shows that both the amplitude and the phase can be determined by evaluating a combination of a sine and a cosine function. This is used to determine the least-squares input matrix, and thus the coefficients for the **r** vector.

$$\mathbf{X} = \begin{bmatrix} \cos(\omega_t \cdot t_i) & \sin(\omega_t \cdot t_i) & {t_i}^1 & {t_i}^0 \end{bmatrix} \tag{3.24}$$

Where **X** is the input matrix for the least squares approximation, ω_t is the test frequency, t_i is the vector containing all logged time stamps.

By applying the least square approximation, where the **X** vector is described in equation 3.24, and the **Y** vector containing the measured output, the data points are turned into an analytical representation of the measured signal. The **r** vector then contains the approximation of the amplitudes, a bias and a ramp.

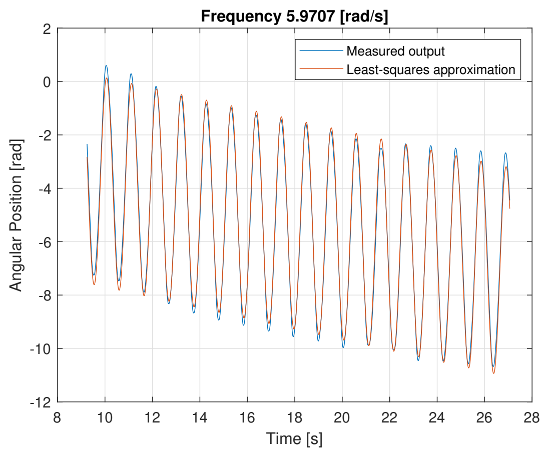


Figure 3.46: Results of least square approximation from a frequency test

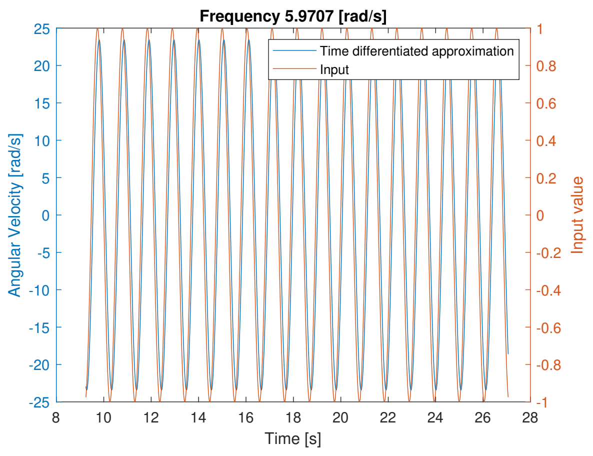


Figure 3.47: Time derivated approximated signal and input

In figure 3.46, the measured data and the corresponding least-squares approximation can be seen. The approximation fits the data points very well. By removing the bias and the ramp, only the sine and the cosine remain. By manipulating it using equations 3.21 to 3.23 both the amplitude and the phase shift of the output is obtained. As this is a position output, the velocity is obtained by differentiating the cosine with respect to time. The output velocity and the input signal can be seen in figure 3.47

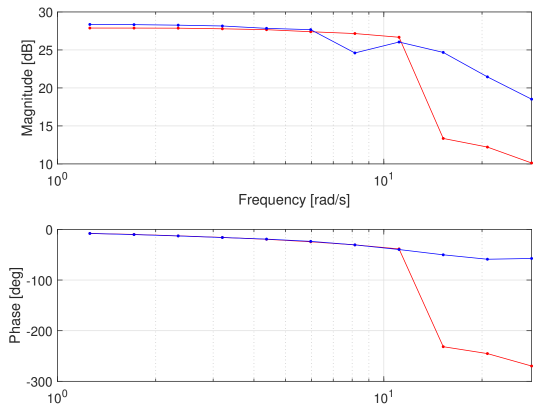


Figure 3.48: Bode plot from the data points

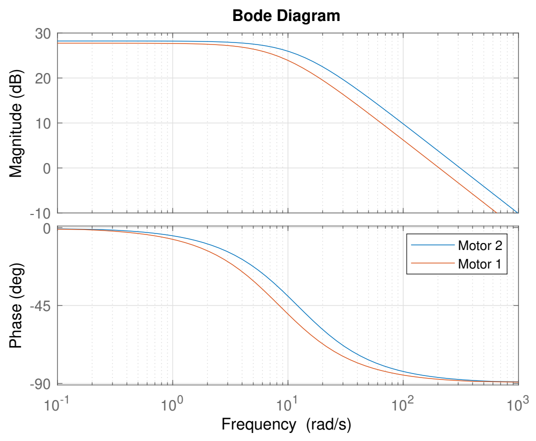


Figure 3.49: Bode plot of the estimated transfer function

In figure 3.48 the bodeplot of the measured frequencies can be seen for both motors. As can be seen, the response is very similar until the test at 15,2 $[\frac{rad}{s}]$, where motor one starts to lose performance, but motor two still performs well. For this reason, the frequency test were not done for all the planed frequncies, the last test was at 28,3691 $[\frac{rad}{s}]$.

Transfer functions for the two motors were estimated using the "System Identification" toolbox in Matlab and the "tfest" function. The function uses data from the frequency response test, and the desired number of zeroes and poles to estimate the best fit transfer function. The best estimations were obtained by estimating the motors as first-order transfer functions with no zeroes, and the bode plot of the obtained transfer functions can be seen in figure 3.49.

Transfer function for motor 1: $\frac{205}{s+8{,}405}$
Transfer function for motor 2: $\frac{311{,}4}{s+12{,}07}$

3.5.2 State-space control System

As mentioned in section 2.4, the state-space offers a very general solution to the control problem. Based on previous experience, the state-space controller provides good results. It is chosen as the control strategy for the robotic cat. In this subsection, the theory from section 2.4 is applied to create a controller for the rotation between the inner and outer parts of the robot cat.

3.5.3 Model

The state-space model and the transfer function are different forms for the same information, the state-space model could, therefore, be obtained from the Matlab function "tf2ss". The function takes the numerator and denominator and returns the **A**, **B**, **C** matrices, and a D term which is zero. Using the transfer function obtained from the frequency analysis resulted in the following state-space models.

$$
\begin{array}{llll}
Motor1: & \mathbf{A}_1 = [-8.4049] & \mathbf{B}_1 = [1] & \mathbf{C}_1 = [205.0256] \\
Motor2: & \mathbf{A}_2 = [-12.0730] & \mathbf{B}_2 = [1] & \mathbf{C}_2 = [311.3979]
\end{array}
$$

As this is a first-order transfer functions, the **A**, **B**, and **C** are 1x1 matrices, the D term is zero. Such things as canonical forms are therefore not applicable.

3.5.4 Control structure and design

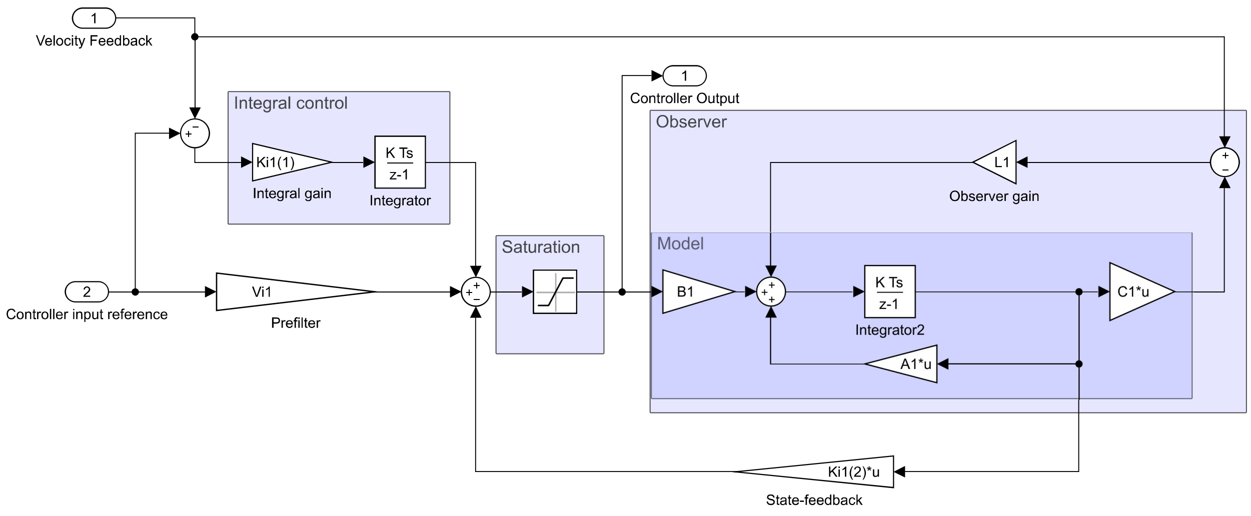


Figure 3.50: Control structure

The model uses state-feedback with a gain vector, **K**, to change the poles of the system. A prefilter gain is added to relate the reference input to the desired system output. Integral control is added to account for model inaccuracies, remove the steady-state error, and handle disturbances. An observer is also included. As the linearization for the motors inputs are only valid from [-1,1], a saturation block is added to the input of the controller. The control structure can be seen in figure 3.50.

The controller is tuned by placing the poles, this is done by the *place* function in Matlab. The function has the **A**, **B** matrix and the desired poles as input, and outputs the resulting gain vector, **K**, which is responsible for the feedback gains. As the controller uses an integrator, which increases the number of states for the system, the system matrix and input vector require some modification.

$$
\mathbf{A}_i = \begin{bmatrix} 0 & \mathbf{C} \\ 0 & \mathbf{A} \end{bmatrix} \qquad \mathbf{B}_i = \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix}
$$

where $\mathbf{A}_i$ and $\mathbf{B}_i$ is the system matrix, and the input vector used for pole placement. The corresponding **K** vector therefore contains two value which needs to be set to tune the controller.

$$Motor1: \quad \mathbf{P}_{i1} = \begin{bmatrix}-7 & -15\end{bmatrix} \quad \mathbf{K}_{i1} = \begin{bmatrix}0.5121 & 13.5951\end{bmatrix} \quad \mathbf{V}_1 = 0.0732$$
$$Motor2: \quad \mathbf{P}_{i2} = \begin{bmatrix}-8 & -19\end{bmatrix} \quad \mathbf{K}_{i2} = \begin{bmatrix}0.4881 & 14.9270\end{bmatrix} \quad \mathbf{V}_2 = 0.0610$$

Where **P** is the poles, **K** is the corresponding gain, and **V** is the prefilter gain. The first index is the integrator pole and gain, and the second is for the state-feedback.

3.5.5 Observer design

The observer requires faster dynamics than the observed system in order to converge to the measured signal quickly. Choosing the observer poles is, however, a trade-off between faster dynamics and noise amplification [6]. The poles are usually set to two to six times the poles of the system[6]. The poles for the robotic cat is set to three times the poles of the system. As the observer uses the original system model, it only has one pole, resulting in the following pole placement and feedback gain.

$$Motor1: \quad \mathbf{P}_{obs1} = \begin{bmatrix}-25.2148\end{bmatrix} \quad \mathbf{L}_1 = \begin{bmatrix}0.0820\end{bmatrix}$$
$$Motor2: \quad \mathbf{P}_{obs2} = \begin{bmatrix}-36.2190\end{bmatrix} \quad \mathbf{L}_2 = \begin{bmatrix}0.0775\end{bmatrix}$$

3.5.6 Performance test

To test the performance of the controller and check if the motor can deliver the torque and speed required to fulfill the requirements it is selected for; a test is performed. The test relies on the work done with the motor calibration and linearization, system identification, and the chosen controller.

Test spesification

The test is performed with the same setup as the calibration and frequency tests, half the cat-robot is assembled at the time, and all internal components except the motor and encoder are external to the robot.

A trapezoidal velocity profile is created, with equally large acceleration, steady-state, and deacceleration time. The total time of the is set to 0,4 $[s]$, with maximal angular speed of 7,5π, which corresponds to a change of position of 2π.

Implementation

As mention in section 3.3.1, the controller runs on the Pixhawk, and the code running on the Pixhawk is generated from a Simulink model using the *Embedded Coder Support Package for PX4 Autopilots* support package. The controller for the motor is therefore created connecting block diagrams in Simulink, as shown in figure 3.50.

Encoder data is read every 0,005 $[s]$ with the procedure described in section 3.3.1. The encoder count is converted to from counter value to radians by a factor of $\frac{2\pi}{4000}$. To obtain the angular

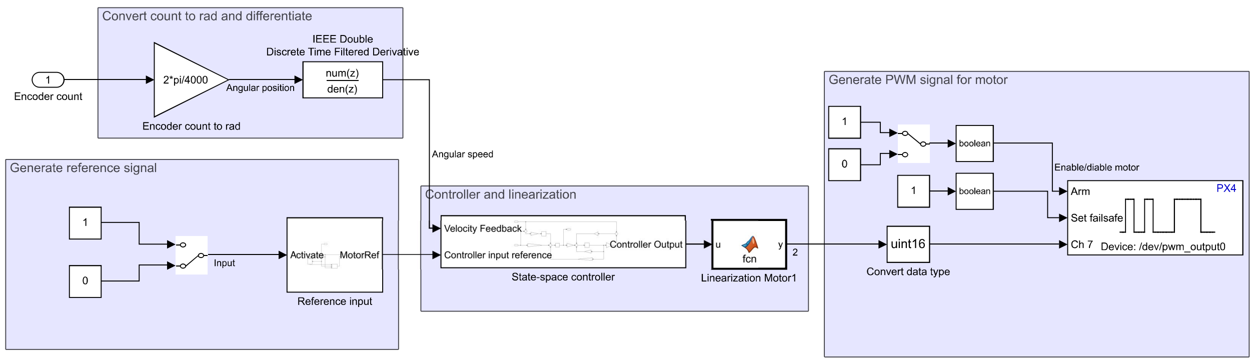


Figure 3.51: Simulink test model

velocity, an IEEE double discrete filtered derivative is used [8].

$$\omega = C_e \left(\frac{2\pi}{4000} \right) \left(\frac{z-1}{z + T_s/T - s} \right) \tag{3.25}$$

Where ω $[\frac{ras}{s}]$ is the angular velocity between the inner and outer assembly, C is the encoder count $[-]$, T $[-]$ is the time constant, T$_s$ $[s]$ is the sample time

Chapter 4

Results

4.1 Concept selection

Based on a literature study, five concepts are suggested for building a robot cat and weighed selection criteria are specified. The concepts are scored, and the sum of the weighted score is used to choose a concept. Ultimately a concept based on the rotating cylinder description is selected.

4.2 Components and design

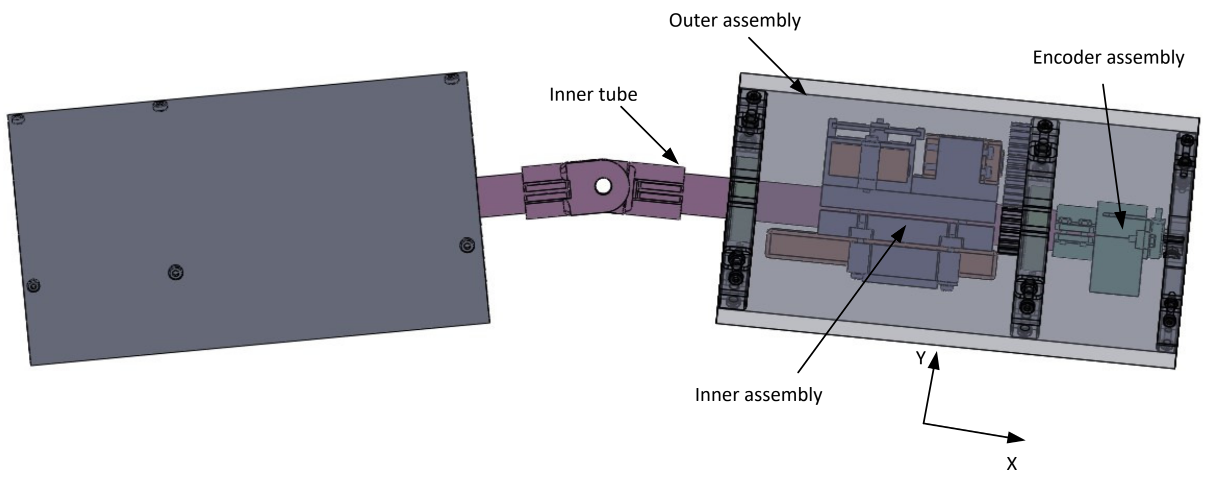
Based on the chosen concept, the components required to make a physical model is identified. Based on assumptions about the structure of the robotic cat and component requirements, components are chosen. The Matlab script used for component selection can be seen in appendix A.0.1.

Table 4.2: Selected components

Part	**Name**	**Number**
Motor	Futaba HPS-700	2
Encoder	Broadcom HEDM 5540	2
Quadrature counter	SuperDroid Robots Dual LS7366R	1
MCU	Pixhawk 4 Mini	1
IMU	ICM-20689 (Pixhawk)	1
IMU	BMI055 (Pixhawk)	1
Aluminium pipe	20x14	2
PE100 tube	125×110	2
Bearing	Biltema 6004 2RS	4
Bearing	SER-1397 Ball bearing 8x12x3,5 SS	4
Battery	Etop 2s 2200 mAh 35c	2
Total	3.7	

4.3 Design and implementation

A design is proposed. The design is built on the chosen concept and implements the chosen components. The design consists of a front, and a rear, symmetrical about the joint connecting them, except only one part has the control units.



The two parts consist of an inner tube, an encoder assembly, an outer assembly, and an inner assembly.

4.4 Interface

Communication between the devices is established. A communication protocol is created for communication between the Pixhawk, the Arduino, and the quadrature counter. The Pixhawk transmit data requests, and the arduino responds. Code is created for both the Pixhawk and the Arduino, allowing them to recognize different messages and process them accordingly. There are currently two recognizable messages, one for calibrating the encoder, and one for counter value. The code is designed to implement more requests easily. Stable communication is achieved, and data is transmitted every 0,005 $[s]$. The code runing on the Arduino can be seen in appendix A.0.3 and the Matlab script used to decode UART messages can be seen in appendix A.0.2.

4.5 System identification and Control system

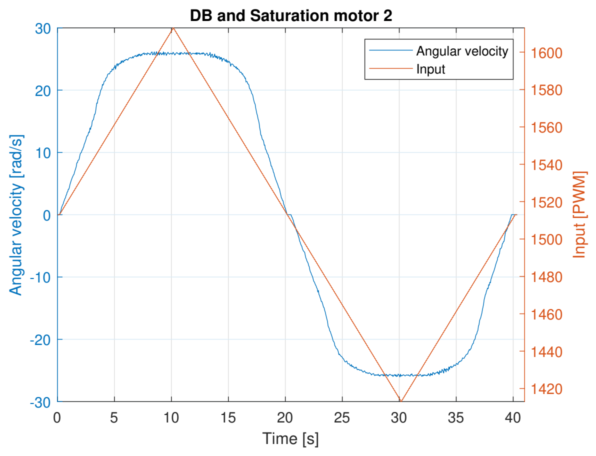


Figure 4.1: Input/output response before calibration

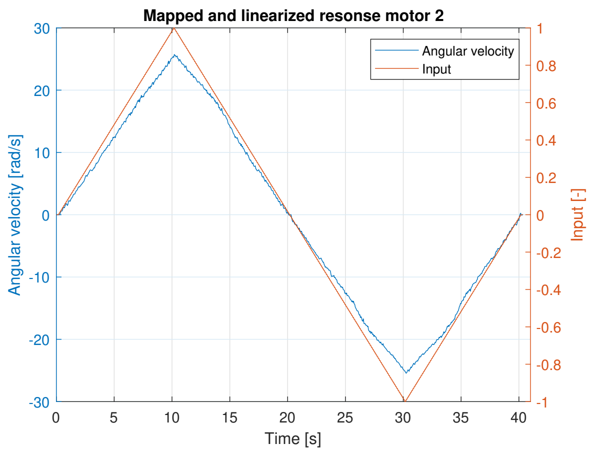


Figure 4.2: Input/output response after calibration and linearization

Modifications are done to the servo motors in order to enable the continuous drive. Motor calibration is performed. The upper and lower input PWM limits, as well as the deadband of the motor,

Table 4.3: Coefficients for linearization polynomials

	Motor one		**Motor two**	
Power	**Positive**	**Negative**	**Positive**	**Negative**
10	28835.9	-6805.3	-168842.3	64560.9
9	-105929.2	-18195.6	840447.9	340381.0
8	149230.9	-8819.9	-1777192.8	761974.4
7	-93830.0	18998.9	2080831.9	945313.6
6	16025.6	27737.6	-1473171.8	709901.9
5	10493.4	13327.2	646567.3	330404.6
4	-5651.7	1385.5	-173598.1	93596.7
3	843.9	-876.3	27128.7	15134.2
2	24.9	-298.7	-2220.8	1201.2
1	32.4	15.5	122.9	78.0
0	1514.0	1508.9	1513.7	1511.0

is identified. The least-squares approximation is applied to create a linear relationship between the input PWM signal and the output angular velocity of the outer assembly. The coefficients of the approximated polynomials is shown in table 4.3, and the Matlab script used to create the linearization function can be seen in appendix A.0.4.

Table 4.4: Maximal measured output speed [RPM]

	Positive	**negative**
Motor one	246,4	242,2
Motor two	245,7	243,0

The highest measured output speeds for motor one and two are listed in table 4.4. As can be seen, this exceeds output speed expected based on the information provided by the motor manufacturer, and the applied gear ratio between the motor and the outer assembly.

Due to the lack of available parameters required for creating a model of the motor response, black box frequency analysis is performed. Frequency response data is gathered, and the least-squares approximation is applied to create an analytical representation of the measured data. Further manipulation of the data resulted in the estimation of a first-order transfer function relating the motor input to the angular speed of the outer assembly. The Matlab script used during the frequency tests can be seen in appendix A.0.5

A control structure is created. It is based on a state-space model and consists of state-feedback, integral control, and an observer. The tuning of the controller is not performed. The control system is created using Simulink and uploaded to the Pixhawk. The Matlab script used to generate the transfer function and calculate the parameters for the controller can be seen in appendix A.0.6

4.6 Test Results

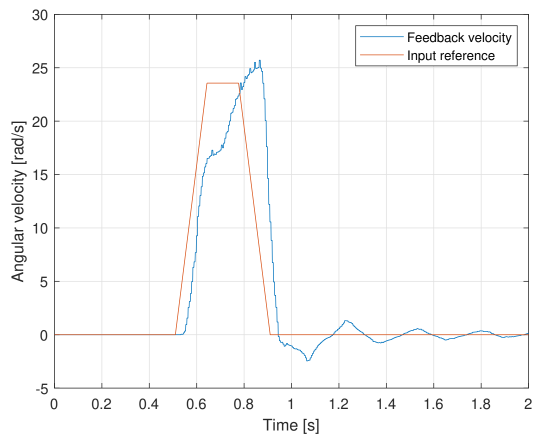


Figure 4.3: Test results motor one

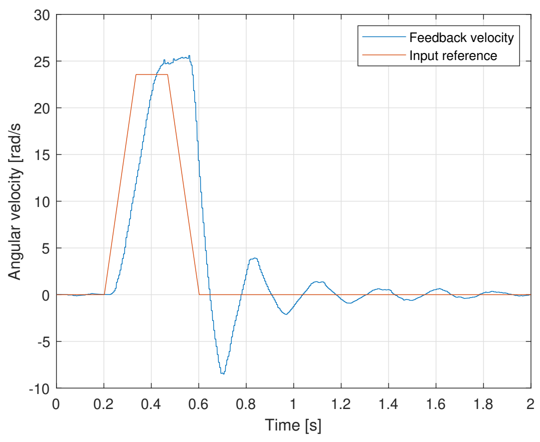


Figure 4.4: Test results motor two

Figure 4.3 and 4.4 show the control input and the measured output. As can be seen, the motors are not able to catch up to the input reference, and both motors display overshoot and stabilization issues. Motor one also displays some strange behavior during the ramp-up, seem to converge towards the input reference, up to that point. It also responds rapidly to change in input value, as can be seen in the response as the control input changes to zero. Motor two seems to diverge as the input reference increase, but converge it decrease. It does, however, respond poorly to the change in input value as input reference changes to zero, and displays larger overshoots than motor one.

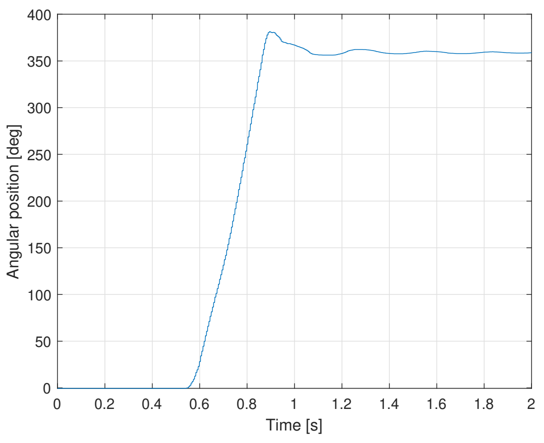


Figure 4.5: Change in position for motor one

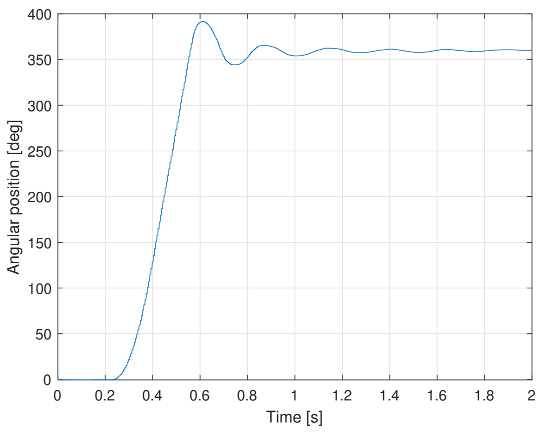


Figure 4.6: Change in position for motor two

Figure 4.5 and 4.6 show corresponding position, which is the desired output. As expected from the velocity data, both motors overshoot the target of 360°, and display an undesired settling time.

In figure 4.7, the estimated velocity of the IEEE filter and a discrete derivation block are both fed the encoder position. As can be seen, the filter creates a smoother signal. However, high-speed dynamics disappear. The filter also introduces a possibly significant delay.

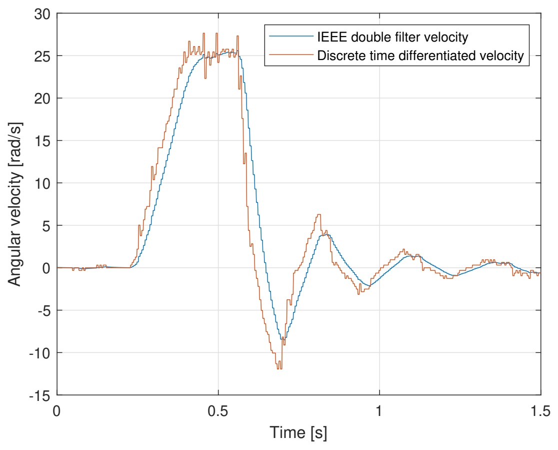


Figure 4.7: Performance of IEEE filter

Chapter 5

Discussions

Finding components for the robot cat was not an easy task. High performance, low weight, small size, and low cost generally do not coexist. Assumptions about the performance requirements had to be made. Selecting a motor that required modification, with little knowledge about what performance could be expected, was the last resort solution. However, with the compact size, low weight, low cost relative to the alternative, and a high potential for torque, the motor showed great potential and was considered worth testing. The motors does, however, perform well. They exceeded the expected velocity, and based on the performed test, the motors produce sufficient torque for the purpose they are selected for.

The process of getting the feedback signal from the encoder to the Pixhawk required a great deal more work than what was expected during component selection. As the Matlab support package supports serial communication, and the Pixhawk has an SPI port, it was assumed that it could be used, which is not the case. Creating a driver for the SPI port is possible. However, it requires a good knowledge of the NuttX operating system and the hardware configuration of the Pixhawk. It would, therefore, be very time-consuming. Finding the solution of implementing an Arduino, and creating a communication protocol for UART, therefore, required a good deal of extra work. Configuring the index pulse of the encoder was also more challenging than expected. As none of the quadrature counters configurations meant for the index pulse could be used, one had to be created. This added extra complexity to the UART communication as the devices have to differentiate between different signals. Neither the implementation of the Arduino nor the configuration of the index pulse is elegant solutions. That being said, the suggested communication protocol works very well. Once the encoder is in the right position, the mode switched. As can be seen from the time plots, stable encoder readings are achieved. This is also due to the design of the encoder assembly. Poor centration would be noticable in the encoder readings.

The SPI communication between the Arduino and the quadrature buffer is based on sample code provided by the manufacturer. The code is however modified substantially to fit the purposes of of the robotic cat [17].

The battery is dimensioned to be able to drive three Futaba HPS-700 motors. As the battery is a relatively has substantial mass, and running motor currents side by side with the encoder signal is undesirable, an additional battery was added to the other side. The size of the battery can, therefore, be reduced. As the battery limits the length of each side, the length of the robot cat could be reduced by choosing a smaller battery.

As the tests had to be conducted with limited options, the inner tube was handheld during the tests. This impacts the results as it is impossible to keep the inner tube in a completely fixed position, and the encoder measures the relative angle. This could impact the results regarding the calibration of the motor, especially during zero crossings. It most likely resulted in the strange behavior in the acceleration phase of motor one during the controller test. Investigating the control input showed that it was not the reason.

The motor calibration, frequency tests, and controller was first developed for motor one, and the same tests were later conducted on motor two. A drop in phase was expected, and motor one was tested until it occurred, and a couple of frequencies more. Motor two was tested for the same frequencies as motor one. It does, however, not show the same drop in phase lag and magnitude as motor one. The frequency test should, therefore, have been extended to include more frequencies. As the least-squares approximation approximates the position, it is not affected by the delay caused by time differentiation, visible in the performed test. The position is, however, delayed up to 0,005 $[s]$, based on the time to transmit the request to the Arduino and for the Arduino to read the SPI data, which was not but should have been included in the phase angle calculations.

The transfer functions were estimated to be first-order transfer functions due to a lack of coherence with the estimated second-order transfer functions. For both motors, the phase drops about 10 degrees between each measured frequency, this might be due to non-ideal test conditions mentioned, and the delayed sampling increasingly impacts the phase as the frequency increases. Motor one drops $180°$ between 11.1 and 15.2 $[rad]$, characteristic of a second-order system. Better test conditions, more test frequencies in the region of the phase drop, and including the sampling delay should provide more accurate transfer function estimations, if required.

The suggested control structure was created for a previous project and provided excellent results. As the first-order transfer function has one state, which is measured, the observer does not really provide any functionality to the controller. However, by increasing the transfer function to a second-order system, accounting for the angular acceleration, the observer would reconstruct the state from the measured velocity. This would prevent the extra delay and noise amplification associated with further filtering or differentiation.

The performance of the controller is, however, none conclusive. The measured results do not provide the required performance. The controller is not tuned. The poles were placed a little faster than the original poles for the transfer function. As there was no time for further testing at the end of the project, this is the presented results. Both motors have stability problems. This could be the result of plenty of problems such as too high controller gains, none ideal linearization, poor test conditions, the delayed feedback signal, and the list goes on.

The controller and the results for the frequency test require that the system is linear. None ideal linearization and deadband compensation, saturation, viscous friction from the bearings are examples of nonlinearities which could degrade the performance of the controller. The linear relationship assumed when identifying the system and creating a controller will most likely not hold up when dropping the robot cat, and the controller will require further tuning. It is however desirable to test that the desired performance can be achieved on a linear model, as its not only the controller that is tested.

Many factors required significantly more time than expected. Due to unforeseen circumstances, the school closed during the semester. A lot of the tools required for the project were therefore unavailable, and improvisations had to be made. Simple practical tasks quickly become time-consuming when the right tools are not available.

The robotic cat is not yet completed. Based on the steps mentioned at the end of the concept selection section, the project is at the end of phase one. When the motors are tuned, the robot is ready for the drop test. Further development involves creating a mathematical system description and a control system for the robot cat, implementing legs, and the joint for actuating the waist. Proper groundwork is, however, laid for further development.

Chapter 6

Conclusions

A literature study was performed, resulting in five concepts. Ultimately a concept built on the rotating cylinder description was chosen. The required components for building a robot cat was identified, and suitable components are chosen, based on assumptions made about the performance requirements. The components are implemented into the design of the robot, except for the legs and an actuated joint to combine the front and the rear of the robot cat. A communication protocol using both SPI and UART between the sensors and the control unit is established. Motor calibration and linearization using the least-squares approximation is performed in order to obtain a linear relationship between the motor input and the sensed output. Blackbox frequency analysis is performed to generate a transfer function for the actuation of the angle between the two rotating cylinders. A state-space velocity controller is suggested using state feedback, integral control, a prefilter, and an observer, however due to limited time left on the project, the controller is not tuned. Although the robot cat is not finished, a good foundation is laid for further development.

Appendix A

Datasheet A

A.0.1 Component Selection

```
close all;
clear;
clc;


%% Motor selection
% Intertia pipe
rho_pe = 1240; %Density kg / m^3
d_pe = 12*10^-2; %Outer diameter m
l_pe = 24*10^-2; %Length m
t_pe = (12.5-11.0)/2*10^-2; %Wall thickness m
r1_pe = d_pe/2; %Outer radius m
r2_pe = (d_pe-2*t_pe)/2; %Inner radius
a_pe = pi * (r1_pe^2 - r2_pe^2); %Surface area m^2
v_pe = a_pe * l_pe; %Volume m^3
m_pe = v_pe * rho_pe; %Mass kg
iz_pe = m_pe * (r1_pe^2 + r2_pe^2) / 2; %Inertia about Z-axis kg * m^2

% Inertia Shock absorbers
m_abs = 0.1; %Mass kg
l_abs = 0.12; %Length m
ilocz_abs = m_abs * l_abs^2 / 12; %Inertia about local Z-axis kg * m^2
imc_abs = m_abs * (r1_pe + l_abs / 2)^2; %Move contribution of intertia about ...
    axis of rotation kg * m^2
itot_abs = ilocz_abs + imc_abs; %Total inertia about cat Z-axis kg * m^2

% Inertia bearing
d1_b = 48 * 10^-3; %Outer diameter bearing m
d2_b = 25 * 10^-3; %Inner diameter bearing m
r1_b = d1_b / 2; %Outer radius bearing m
r2_b = d2_b / 2; %Inner radius bearing m
m_b = 0.1;
i_b = m_b / 2 * (r1_b^2 + r2_b^2);


% Inertia of front/back wall
```

```
r1_w = r2_pe; % Outer radius wall m
r2_w = r1_b; % Inner radius wall m
t_w = 5*10^-3; % wall thickness m
a_w = pi * ( r1_w^2 - r2_w^2 ); % Surface area of wall m^2
v_w = a_w * t_w; % Volume of wall m^3
m_w = v_w * rho_pe; % Mass of wall, under the assumption of printed plastic has ...
    the same density as pe
i_w = m_w * ( r1_w^2 + r2_w^2 ) / 2; % Inertia of wall

% Mass of shockabsorber counterweight
i_cw = itot_abs; %Inertia kg * m^2


rads2rpm = 60/(2*pi);
deg2rad = pi/180;
% Velocity profile
dPhi = 2*pi;
dT = 0.5;
wMax = dPhi/dT*3/2;
aMax = wMax / (dT / 3);

%Total inertia
i_tot = iz_pe + 2 * itot_abs + 2 * i_cw + 2 * i_w + 2 * i_b;
m_tot = m_pe + 2 * m_abs  + 2 * m_w + 2 * m_b;
Treq = i_tot * aMax;


%% Torque vs speed

dt = 0.1:0.001:1;

for i=1:length(dt)
    w1Max(i) = dPhi/dt(i)*3/2;
    a1Max(i) = w1Max(i) / (dt(i) / 3);
    Tr(i) = i_tot*a1Max(i);
    Pr(i) = w1Max(i) * Tr(i);

    v0 = 0;
    x0 = 0;
    zdd(i) = 9.81;
    zd(i) = zdd(i) * dt(i) + v0;
    z(i) = zdd(i) * dt(i)^2/2 + v0 * dt(i) + x0;

end

%% Encoder

    t = 5/1000;
    rot = 2*pi;
    maxspeed = 180*2*pi/60; %rpm 2 deg/s
    start = 500;
    stop = 10000;

for i = 1 : stop-start + 1
    k = i + start-1; %PPR
    thetares(i) = 2*pi / k; %Position resolution
    omegares(i) = thetares(i)/t; %Velocity resolution
    percent(i) = omegares(i) / maxspeed * 100; %Resolution as percentage of max ...
        speed
    res(i) = k; %PPR
end

upto4k = percent(1)-percent(3501);
from4kto10k = percent(3501) - percent(stop-start+1);

figure(1)
yyaxis left
plot(res,omegares);
ylabel('Angular Speed Resolution [rad/s]')
hold on
yyaxis right
ylabel('Percentage of max speed [%]')
xlabel('Counts Per Revolution [-]')
plot(res,percent);
grid;

figure(2)
plot(res,thetares);
ylabel('Angular Speed Resolution [rad/s]')

grid;


figure(2)
plot(dt,w1Max*rads2rpm,'-r')
grid
xlabel('Time (s)')
ylabel('Angular speed [RPM]')
hold on
yyaxis left
%plot(dt,z);
yyaxis right
plot(dt,a1Max*i_tot,'-b')
ylabel('Torque [Nm]')
legend('Speed requirement','Torque requirement')
```

A.0.2 Pixhawk Read UART messages

```matlab
function [Enc1, Enc2, CalibStatusEnc1, CalibStatusEnc2, ERROR, NoMessage]  = ...
    ProcessSerialData(data)

%Initialize the variables
data = data';
StartMarker = uint8(62);
StartNumber = 1;
NoMatch = false;
LenEncData = 4;
CheckForStart = uint8(0);
Enc1 = int32(0);
Enc2 = int32(0);
CalibStatusEnc1 = uint8(0);
CalibStatusEnc2 = uint8(0);
ERROR = 0;
NoMessage = 0;

CalibrationStatus = 67; %Ascii for C, indicates message contains calibration ...
    data
EncoderData = 82; %Ascii for R. indicates message contains encoder data


while(CheckForStart ~= StartMarker)
    CheckForStart = data(StartNumber);
    StartNumber = StartNumber + 1;

    if(StartNumber > length(data))
        NoMatch = true;
        break
    end
end


if(NoMatch == false)

    MessageFormat = data(StartNumber)

    if(MessageFormat == EncoderData)
        for i = 0: LenEncData
            Enc1 = bitshift(int32(data(StartNumber + 1 + i)),8*(LenEncData-i-1))...
                 + Enc1;
            Enc2 = bitshift(int32(data(5 + StartNumber + i)),8*(LenEncData-i-1))...
                 + Enc2;
        end
    elseif(MessageFormat == CalibrationStatus)
        CalibStatusEnc1 = uint8(data(StartNumber + 1));
        CalibStatusEnc2 = uint8(data(StartNumber + 2));

    % Insert more message formats here
    else
        ERROR = 1;
    end
else
```

```
    NoMessage = 1;
end

return
```

A.0.3 Communication Arduino

```
#include <SPI.h>

const uint8_t disableSlaveMode = 10; //Keeps the arduino from accedentally ...
    entering slavemode
const uint8_t ss1 = 2; //Slave select 1
const uint8_t ss2 = 3; //Slave select 2
long enk1;
long enk2;

long t1;
long t2;

const byte waitForIndex = 0x23;
const byte indexFound = 0x03;

volatile bool messageReady = false;
volatile bool readEncoder = false;
byte instruction;

bool enc1Calibrated = false;
bool enc2Calibrated = false;
bool calibrated = false;
byte encCalibStatus[3] = {0, 0};

long counterEnc1 = 0; //Encoder one counter
long counterEnc2 = 0; //Encoder two counter


void indexSetting(uint8_t ss, byte configuration) {

  digitalWrite(ss,LOW);        // Begin SPI conversation

  SPI.transfer(0x88);                         // Write to MDR0
  SPI.transfer(configuration);                        //

  digitalWrite(ss,HIGH);       // Terminate SPI conversation

}



void clearEncoderCount() {

  // Set encoder1's data register to 0
  digitalWrite(ss1,LOW);      // Begin SPI conversation
  // Write to DTR
  SPI.transfer(0x98);
  // Load data
  SPI.transfer(0x00);  // Highest order byte
  SPI.transfer(0x00);
  SPI.transfer(0x00);
  SPI.transfer(0x00);  // lowest order byte
  digitalWrite(ss1,HIGH);     // Terminate SPI conversation

  delayMicroseconds(100);  // provides some breathing room between SPI ...
      conversations

  // Set encoder1's current data register to center
  digitalWrite(ss1,LOW);      // Begin SPI conversation
  SPI.transfer(0xE0);    //0
  digitalWrite(ss1,HIGH);     // Terminate SPI conversation

  // Set encoder2's data register to 0
  digitalWrite(ss2,LOW);      // Begin SPI conversation
  // Write to DTR
  SPI.transfer(0x98);
  // Load data
  SPI.transfer(0x00);  // Highest order byte
  SPI.transfer(0x00);
  SPI.transfer(0x00);
  SPI.transfer(0x00);  // lowest order byte
  digitalWrite(ss2,HIGH);     // Terminate SPI conversation

  delayMicroseconds(100);  // provides some breathing room between SPI ...
      conversations

  // Set encoder2's current data register to center
  digitalWrite(ss2,LOW);      // Begin SPI conversation
  SPI.transfer(0xE0);    //0
  digitalWrite(ss2,HIGH);     // Terminate SPI conversation

  delayMicroseconds(100);  // provides some breathing room between SPI ...
      conversations

  // Clear STR
  digitalWrite(ss1,LOW);
  SPI.transfer(0x30);
  digitalWrite(ss1,HIGH);

  delayMicroseconds(100);  // provides some breathing room between SPI ...
      conversations

  digitalWrite(ss2,LOW);
  SPI.transfer(0x30);
  digitalWrite(ss2,HIGH);
}


void setup() {
  Serial.begin(230400);

  pinMode(disableSlaveMode, OUTPUT); // Prevent the arduino from entering slave ...
      mode
  pinMode(ss1, OUTPUT);
  pinMode(ss2, OUTPUT);  // Set slave selects as outputs

  digitalWrite(ss1,HIGH);
  digitalWrite(ss2,HIGH);

  SPI.begin();

  indexSetting(ss1, waitForIndex);
  indexSetting(ss2, waitForIndex);

  Serial.println("Encoders Initialized...");

  clearEncoderCount();
  Serial.println("Encoders Cleared...");


}

void loop() {
  instruction = receiveInstructions();
  executeCommand(instruction);
  }

byte receiveInstructions(){
  byte rd = 0; //byte to read into
  bool instructionReady = false;

  while(Serial.available() && instructionReady == false){

    rd = Serial.read();
    instructionReady = true;

  }

  return rd;

}

void executeCommand(byte instruction){
  switch (instruction) {
    case 67:
      calibrateEncoders();
      //Serial.println("entering case 1");
      break;
    case 82:
      sendEncoderData();
      //Serial.println("entering case 2");
      break;
    default:
      //Do nothing

      break;
  }
}

void sendEncoderData(){
  byte encoderArray1[4];
  byte encoderArray2[4];

  digitalWrite(ss1,LOW);      // Begin SPI conversation
  SPI.transfer(0x67);                        // Instruction to read counter
  encoderArray1[0] = SPI.transfer(0x00);           // Read highest order byte
  encoderArray1[1] = SPI.transfer(0x00);
  encoderArray1[2] = SPI.transfer(0x00);
  encoderArray1[3] = SPI.transfer(0x00);           // Read lowest order byte
  digitalWrite(ss1,HIGH);     // Terminate SPI conversation


  digitalWrite(ss2,LOW);      // Begin SPI conversation
  SPI.transfer(0x67);                         // Instruction to read counter
  encoderArray2[0] = SPI.transfer(0x00);           // Read highest order byte
  encoderArray2[1] = SPI.transfer(0x00);
  encoderArray2[2] = SPI.transfer(0x00);
  encoderArray2[3] = SPI.transfer(0x00);           // Read lowest order byte
  digitalWrite(ss2,HIGH);     // Terminate SPI conversation

  Serial.write('>'); // Start byte
  Serial.write('R'); // Indicates encoder data is sendt
  Serial.write(encoderArray1, sizeof(encoderArray1));
  Serial.write(encoderArray2, sizeof(encoderArray2));
}

void calibrateEncoders(){
    byte str1;
    byte str2;
    byte idx1 = 0;
    byte idx2 = 0;

    if(calibrated == false){
      if(enc1Calibrated == false){
        digitalWrite(ss1,LOW);      // Begin SPI conversation
        SPI.transfer(0x70);                        // Request STR register
        str1 = SPI.transfer(0x00);            // Read STR register
        digitalWrite(ss1,HIGH);     // Terminate SPI conversation

        idx1 = str1 & 16; // returns 16 if index is latched

        if(idx1 == 16){
          enc1Calibrated = true;
          encCalibStatus[0] = 1;
          indexSetting(ss1, indexFound);

        }
      }

      if(enc2Calibrated == false){

        digitalWrite(ss2,LOW);      // Begin SPI conversation
        SPI.transfer(0x70);                        // Request STR register
        str2 = SPI.transfer(0x00);            // Read STR register
        digitalWrite(ss2,HIGH);     // Terminate SPI conversation

        idx2 = str2 & 16; // returns 16 if index is latched
        //idx2 = 16; // Disable calibration of encoder two for testing

        if(idx2 == 16){
          enc2Calibrated = true;
          encCalibStatus[1] = 1;
          indexSetting(ss2, indexFound);
        }
      }
    }
    Serial.write('>'); // Start byte
    Serial.write('C'); // Indicates calibration status is sent
    Serial.write(encCalibStatus, sizeof(encCalibStatus));

}
```

A.0.4 Linearize motor

```matlab
close all;
clear;
clc;

%% Convert files an array
numberOfFiles = 10;
dataPointsPerFile = 1000;

tempData = zeros(1,dataPointsPerFile * (numberOfFiles+1) );
tempTime = zeros(1,dataPointsPerFile * (numberOfFiles+1) );
tempData1 = zeros(1,dataPointsPerFile * (numberOfFiles+1) );

for i = 0: numberOfFiles
    loadName = "DataArchiving2_" + num2str(i);
    load(loadName);

    indexStart = 1+dataPointsPerFile*i;
    indexEnd = dataPointsPerFile*(i+1)-(1000-length(ScopeData3.time(:)'));

    tempTime(1,indexStart:indexEnd) = ScopeData3.time(:)';
    tempData(1,indexStart:indexEnd) = squeeze(ScopeData3.signals(1).values);
    tempData1(1,indexStart:indexEnd) = squeeze(ScopeData3.signals(2).values);
end

%remove the empty end of the array
newEnd = i*dataPointsPerFile + length(ScopeData3.time(:)');

angle = tempData(1,1:newEnd);
input = tempData1(1,1:newEnd);
time = tempTime(1,1:newEnd);

%% Time derivation
sample = 10;

tempVelData = zeros(1,ceil(length(time)/sample));
tempVelData1 = zeros(1,ceil(length(time)/sample));
tempVelTime = zeros(1,ceil(length(time)/sample));


for i = 1:ceil(length(time))/sample-1
   tempVelData(i) = (angle(sample*i+sample)-angle(sample*i))/(time(sample*i+...
       sample)-time(sample*i));
   tempVelTime(i) = time(sample*i);
   tempVelData1(i) = input(sample*i);
end


%% Mapping

velRisePos = tempVelData(181:382);
timeRisePos = tempVelTime(181:382);
inputRisePos = tempVelData1(181:382)./100;

```

```
velRiseNeg = tempVelData(581:781);
timeRiseNeg = tempVelTime(581:781);
inputRiseNeg = tempVelData1(581:781)./100;


dbMid = 1513;
satUp = 72;
satDown = 73;
dbPos = 1;
dbNeg = 2;

minInput = dbMid+dbPos;
maxInput = dbMid+satUp;


%% Least square (+ side)

xPos = (velRisePos/16345);
YPos = (ceil(dbPos + (satUp - dbPos) * inputRisePos + dbMid))';

m = 10;
for i=0:m
    XPos(:, m+1-i) = xPos.^i;
end


rPos = (XPos'*XPos)\XPos'*YPos;

zPos = flip(min(xPos):0.01:max(xPos));
overfitPos = polyval(rPos',zPos);
okPos = polyval(rPos',zPos);

%% Least square (- side)


xNeg = (velRiseNeg/16345);
% x = ceil(dbPos + (satUp - dbPos) * inputRise + dbMid);
YNeg = (floor(-dbNeg + (satDown - dbNeg) * inputRiseNeg + dbMid))';

n = 10;
for i=0:n
    XNeg(:, n+1-i) = xNeg.^i;
end


rNeg = (XNeg'*XNeg)\XNeg'*YNeg;

zNeg = flip(min(xNeg):0.01:max(xNeg));
overfitNeg = polyval(rNeg',zNeg);
okNeg = polyval(rNeg',zNeg);


%% Plotting


```

```
%Plot least square approximation:

plot(xPos,YPos,'-b')
hold on
plot(zPos, okPos,'-r')
% legend('Data points','Approximation')
% title('Least square approximation')
% xlabel('X')
% ylabel('Y')


hold on
plot(xNeg,YNeg,'-b')
hold on
plot(zNeg, okNeg,'-r')
% legend('Data points','Approximation')
% title('Least square approximation')
xlabel('Input [-]')
ylabel('Output [PWM]')
legend('Datapoints','Least squares approximation')
title('Linearization function motor 1')
grid

% plot(inputValue,output(2:end));
%
% logTime = tempVelTime(181:986)-tempVelTime(181);
% logVel = tempVelData(181:986)/4000*2*pi;
% logInput = tempVelData1(181:986)/100;
%
% yyaxis left
% plot(logTime,logVel);
% title('Mapped resonse motor 1')
% xlim([0 41])
% ylabel('Angular velocity [rad/s]')
% yyaxis right
% plot(logTime,logInput);
% ylim([-1 1])
% ylabel('Input [-]')
% legend('Angular velocity','Input')
% xlabel('Time [s]')
% grid;
```

A.0.5 Process frequency measurements

```
close all;
clear;
clc;


% Equidistantly placement in bode plot
w_log10 = linspace(log10(0.2),log10(5*pi),15);

%Test frequency
for i=1:size(w_log10')
    w_test(i) = 10^(w_log10(i));
end

% Paramaters to specify:
%---------------------------------
w = w_test(6) * 2 * pi; %Tested frequency
numberOfFiles = 3; %Number of logged files
%----------------------------------

dataPointsPerFile = 1000;

tempData = zeros(1,dataPointsPerFile * (numberOfFiles+1) );
tempTime = zeros(1,dataPointsPerFile * (numberOfFiles+1) );

for i = 0: numberOfFiles
    loadName = "FrequencyLog_" + num2str(i);
    load(loadName);

    indexStart = 1+dataPointsPerFile*i;
    indexEnd = dataPointsPerFile*(i+1)-(1000-length(ScopeData1.time(:)'));

    tempTime(1,indexStart:indexEnd) = ScopeData1.time(:)';
    tempData(1,indexStart:indexEnd) = squeeze(ScopeData1.signals(1).values);
    tempData1(1,indexStart:indexEnd) = squeeze(ScopeData1.signals(2).values);

end

%remove the empty end of the array
newEnd = i*dataPointsPerFile + length(ScopeData1.time(:)');

output = tempData(1,400:newEnd);
input = tempData1(1,400:newEnd);
time = tempTime(1,400:newEnd);

y = output';


%Specify X vector
for k=1:length(time)
    x(k,:) = [cos(w*time(k)), sin(w*time(k)), time(k)^1, time(k)^0];
end

%Calculate coefficients
```

```
r = (x'*x)\x'*y;

%Calculate phase and amplitude
A = sqrt(r(1)^2+r(2)^2);
phi = atan2(r(2),r(1));

%Simulate results
for l=1:length(time)
    z(l) = r(1) * cos(w*time(l)) + r(2) * sin(w*time(l)) + r(3)*time(l)^1 + r(4)...
        *time(l)^0;
    z1(l) = A * cos(w*time(l) - phi) + r(3) * time(l) + r(4);
    z1d(l) = - w * A * sin(w*time(l) - phi);
end

%Plot
figure(1)
yyaxis left
% plot(time,output,time,z1)
plot(time,z1d/4000*2*pi)
xlabel('Time [s]')
ylabel('Angular Velocity [rad/s]')
yyaxis right
plot(time,input)
ylabel('Input value')
grid;
legend('Time differentiated approximation','Input')
grid;
title('Frequency 5.9707 [rad/s]')
figure(2)
plot(time,output/4000*2*pi,time,z1/4000*2*pi)
xlabel('Time [s]')
ylabel('Angular Position [rad]')
legend('Measured output','Least-squares approximation')
grid;
title('Frequency 5.9707 [rad/s]')
```

A.0.6 Transfer function estimation and state-spacecontroller

```
close all;
clear;
clc;


%% Estimating transfer functions

%Gathered POSITION data from frequency tests
A = 1/4000*2*pi*[1.2551e+04, 9.1861e+03, 6.7166e+03, 4.8708e+03, 3.5197e+03, 2...
    .4984e+03, 1.7795e+03, 1.2319e+03, 194.6511, 125.0606, 71.9402];
phi = [-3.0029,- 2.9679, -2.9189, -2.8647, -2.8003, -2.7139, -2.6078, -2.4680, 0...
    .9007, 1.1381, 1.5669];
A2 = 1/4000*2*pi*[1.3258e+04, 9.6763e+03, 7.0299e+03, 5.0841e+03, 3.5914e+03, 2...
    .5782e+03, 1.3262e+03, 1.1459e+03, 716.9085, 362.3451, 189.1750];
phi2 = [-3.0012, -2.9628, -2.9159, -2.8614, -2.8064, -2.7291, -2.6075, -2.4471, ...
    -2.2652, -2.1130, -2.1396];


% Test frequencies, equidistantly placement in bode plot
w_log10 = linspace(log10(0.2),log10(5*pi),15);
%Test frequencies
for i=1:size(w_log10')
    w_test(i) = 10^(w_log10(i));
end

% Calculate phase
% phi_in-phi_out
phi = (0-(phi+pi))/pi*180;
phi2 = (0-(phi2+pi))/pi*180;

% Calculate magnitude in dB for tested frequencies
for i = 1 : length(A)
   M(i) = 20*log10((2 * pi * w_test(i) * A(i)));
   M2(i) = 20*log10((2 * pi * w_test(i) * A2(i)));
end

%Eliminate untested frequencies
W = w_test(1:length(M)) * 2 * pi; % w_log10(1:length(A)) ;
P = phi(1:end);
P2 = phi2(1:end);

%Convert data into usable form
frdata1 = frd(10.^(M/20).*exp(1i.*P*pi/180), W);
frdata2 = frd(10.^(M2/20).*exp(1i.*P2*pi/180), W);

%Estimate transfer function
tfVel = tfest(frdata1, 1, 0);
tfVel2 = tfest(frdata2, 1, 0);

%% State-space controllers
[A1, B1, C1, D1] = tf2ss(tfVel.Numerator, tfVel.Denominator);
[A2, B2, C2, D2] = tf2ss(tfVel2.Numerator, tfVel2.Denominator);

```

```
m1ss = ss(A1, B1, C1, D1);
m2ss = ss(A2, B2, C2, D2);

Observable1 = rank(obsv(m1ss));
Observable2 = rank(obsv(m2ss));
Controlable1 = rank(ctrb(m1ss));
Controlable2 = rank(ctrb(m2ss));


pole1 = -15;                              pole2 = -19;
fastP1 = min(pole(m1ss));                 fastP2 = min(pole(m2ss));
ObsP1 = 3*fastP1;                         ObsP2 = 3*fastP2;
L1 = place(A1', C1', ObsP1)';             L2 = place(A2', C2', ObsP2)';
obs1 = ss(A1-L1*C1, [B1, L1], C1, D1);   obs2 = ss(A2-L2*C2, [B2, L2], C2, D2);
Ai1 = [0, C1; 0, A1]; Bi1 = [0; B1];     Ai2 = [0, C2; 0, A2]; Bi2 = [0; B2];
polei1 = [-7, pole1];                     polei2 = [-8, pole2];
Ki1 = place(Ai1,Bi1, polei1);             Ki2 = place(Ai2,Bi2, polei2);
Vi1 = (C1*(B1*Ki1(2)-A1)^-1*B1)^-1;      Vi2 = (C2*(B2*Ki2(2)-A2)^-1*B2)^-1;



%% Plotting:
figure('Name', 'Bode of cyl2')
subplot(2,1,1)
semilogx(W, M,'-r', W, M,'r.', W, M2,'b',W,M2, 'b.');
xlabel('Frequency [rad/s]')
ylabel('Magnitude [dB]')
grid
subplot(2,1,2)
semilogx(W, P,'-r', W, P,'r.', W, P2,'b',W,P2, 'b.');
ylabel('Phase [deg]');
grid
%
% % Transfer function for cylinder 1
figure(2)
bode(tfVel2,tfVel)
legend('Motor 2','Motor 1')
grid;
% %
% figure(3)
% plot(w_log10,w_test,'bo')
% xlabel('Corresponding exponent value (10^x)')
% ylabel('Actual tested frequency [Hz]')
% title('Tested input frequencies')
% grid;
```

Bibliography

[1] Borealis AG. *Product datasheet: BorSafe HE3490-LS-H.* Mar. 2020. URL: https://www.borealisgroup.com/storage/Datasheets/borsafe/he3490-ls-h/HE3490-LS-H-PDS-REG_WORLD-EN-V2-PDS-WORLD-37207-10053009.pdf.

[2] J. R. Benton. "How a Falling Cat Turns over." In: *Science* 35.890 (1912), p. 104–105. ISSN: 00368075, 10959203. URL: http://www.jstor.org/stable/1637423.

[3] J. T. Bingham et al. "Orienting in mid-air through configuration changes to achieve a rolling landing for reducing impact after a fall." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2014, pp. 3610–3617.

[4] D. M. Calcutt, Frederick J. Cowan, and G. Hassan Parchizadeh. *8051 Microcontroller : An Applications Based Introduction.* Newnes, 2004. ISBN: 9780750657594. URL: http://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=104785&site=ehost-live.

[5] George Ellis. *Control System Design Guide.* 4 ed. Butterworth-Heinemann, 2012. ISBN: 978-0-12-385920-4.

[6] Abbas Emami-Naeini Gene F.Franklin J.David Powell. *Feedback Control of Dynamic Systems.* 7 ed. Pearson, 2015. ISBN: 978-1-292-06890-9.

[7] Hong-Jun Heo, Yungdeug Son, and Jang-Mok Kim. "A Trapezoidal Velocity Profile Generator for Position Control Using a Feedback Strategy." In: *Energies* 12 (Mar. 2019). DOI: 10.3390/en12071222.

[8] "IEEE Recommended Practice for Excitation System Models for Power System Stability Studies." In: *IEEE Std 421.5-2016 (Revision of IEEE Std 421.5-2005)* (2016), pp. 1–207.

[9] Jean J. Labrosse. *Embedded systems building blocks.* R&D Books, 2000. ISBN: 0879306041.

[10] Normal S. Nise. *Control Systems Engineering.* 6 ed. Wiley, 2011. ISBN: 978-0-470-64612-0.

[11] Godfrey C. Onwubolu. *Mechatronics : Principles and Applications.* Butterworth-Heinemann, 2005. ISBN: 9780750663793. URL: http://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=195086&site=ehost-live.

[12] Adam Osborne. *An introduction to microcomputers.* Osborne McGraw-Hill, 1987. ISBN: 0879306041.

[13] Otvinta. *Involute Gear Calculator.* Apr. 2020. URL: http://www.otvinta.com/gear.html.

[14] Kurt M.. Marshek Robert C.. Juvinall. *Machine component design.* Wiley, 2012. ISBN: 9781118092262.

[15] Gunter Roppenecker. "Zustandsregelung linearer Systeme - Eine Neubetrachtung (State Feedback Control of Linear Systems - a Renewed Approach)." In: *Automatisierungstechnik* 57 (Jan. 2009), pp. 491–498. DOI: 10.1524/auto.2009.0796.

[16] Clarence W. de Silva. *Mechatronics: A Foundation Course*. CRC Press, 2010. ISBN: 978-1-4200-8211-1.

[17] SuperDroidRobots. *Encoder-Buffer-Breakout*. May 2020. URL: https://github.com/SuperDroidRobots/Encoder-Buffer-Breakout/blob/master/DualEncoderBreakout/DualEncoderBreakout.ino.

[18] M.P.Schere T.R.Kane. "A DYNAMICAL EXPLANATION OF THE FALLING CAT PHENOMENON." In: *Int J Solids Structures* 5.10 (1969), pp. 663–670. DOI: 10.1016/0020-7683(69)90086-9.

[19] Cornelia Vasile. *Practical Guide to Polyethylene*. S.l. : Smithers Rapra, 2005. ISBN: 9781305076556.

[20] Cliff Wootton. *Samsung ARTIK Reference*. Berkeley, CA, 2016. ISBN: 1-4842-2322-5.