UNIVERSITY OF SOUTHERN DENMARK

ROBOT SYSTEMS ENGINEERING

START DATE: 22. FEBRUAR 2024

DUE DATE: 31. MAY 2024

GROUP NUMBER: 5

INSTRUCTOR: LEON BODENHAGEN

Control and Regulation of Robot Systems

Pan-tilt System

Submitted By:

ALAN RASHID, 07-04-1998, *alras22@student.sdu.dk*

ANAS BUTT HUSSAIN, 15-12-2003, *anhus22@student.sdu.dk*

JAKUB HUBERT RUDOWSKI, 01-02-2001, *jarud22@student.sdu.dk*

PATRICK ØRSTED POVEY ANDERSEN, 10-01-2000,
paand22@student.sdu.dk

ROBIN HANSEN, 08-11-1995, *rohan22@student.sdu.dk*

THOMAS KORSGAARD VILHOLM, 18-08-1997,
thvil22@student.sdu.dk

ANDERS BIRK KRISTOFFERSEN, 22-11-2001,
ankri22@student.sdu.dk

Contents

1 Introduction

The vision for the project was to create a working pan-tilt system designed to work as a manually controlled security camera. Two boards have been used in this project communicating via SPI: a master Tiva board and a slave Pynq Z2 board. The Master board sends signals to the slave board which then initializes and operates the motors of the pan-tilt system.

The Pynq Z2 board employs a double H-Bridge to supply voltage to the motors. These motors are equipped with encoders regulated by an algorithm. Feedback from the encoders is sent back to the master Tiva board allowing it to display the tilt and pan angles of the system. The closed-loop control system was not implemented in the project, yet it was described in section 6. The control system used in the pan-tilt system was an open-loop control. This report details each step of the process.

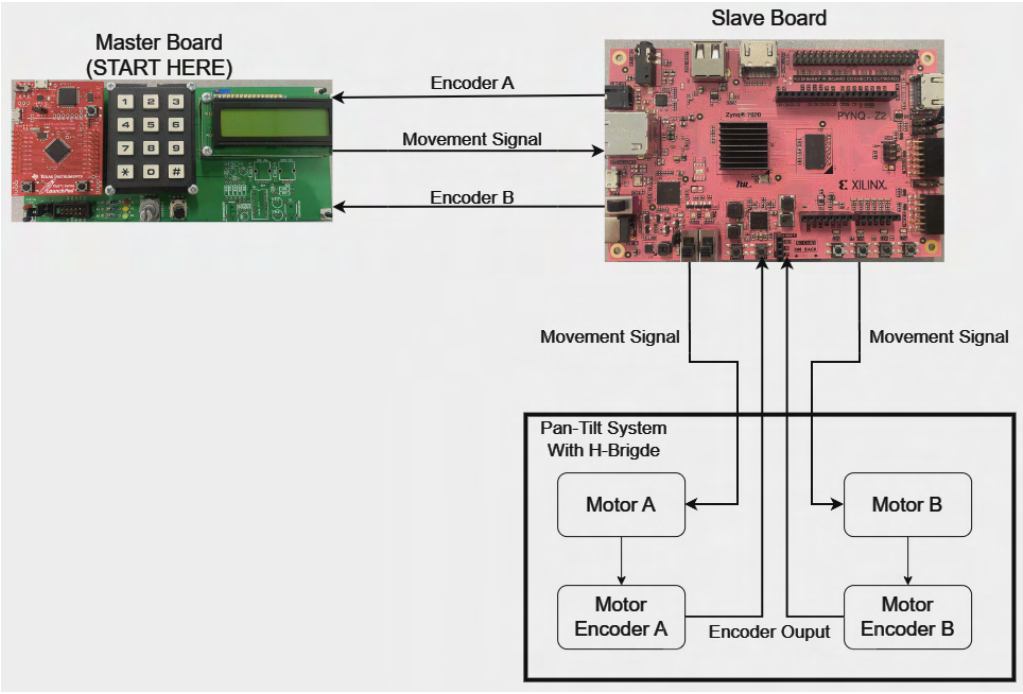


Figure 1: The diagram shows a simplistic view of how the whole system is supposed to work.

Below are the required tasks to complete the goal:

Task Number	Task	Task distribution
1	SPI protocol	Patrick, Anas, Robin
2	Motor movement	Jakub, Thomas
3	Control loop system	Anders
4	Experiments	Alan, Thomas, Jakub

Table 1: Task distribution table

2 Project Description

The goal for this project was to create a functioning pan-tilt system, by utilizing an SPI interface between master Tiva board and a slave Pynq Z2 board. The system's purpose was to serve as a manually controlled security camera.
The following key components were used:

1. Control loop system: Ensured that the system intercepted the input signals and determined which direction the pan-tilt system should move.
2. SPI protocol: Enabled effective communication between master and slave boards allowing data transmission in both directions.
3. Motor movement algorithm: Allowed the pan-tilt system to move in the desired direction and at the desired speed.

3 Tiva Microprocessor

This section is dedicated to the development of the software and hardware code for the TivaTM4C123GH6PM Microcontroller that was used in the project. Tiva board served as the Master in the SPI communication with the FPGA (see TivaTM4C123GH6PMMicrocontrol in Appendix, used for understanding the Tiva board).

3.1 RTOS

For this project the components used were the keypad and LCD on the extension board for the Tiva board. The keypad was used as input and LCD as output (see fig. 2)

Figure 2: The picture shows the Tiva board connected to the extension board and the needed hardware.

The initialization of the used hardware was a crucial step for using the needed components. The Tiva board was programmed using FreeRTOS (Real Time Operating System), where the code was divided into tasks, queues and semaphores. Task management and scheduling were possible due to the operating system's functionality in FreeRTOS.

These code components were initialized in the setupHardware() function seen in fig. 3

```c
static void setupHardware(void)
{
  // TODO: Put hardware configuration and initialisation in here
  // Warning: If you do not initialize the hardware clock, the timings will be inaccurate
  init_systick();
  init_gpio();

  xQueue_keypad = xQueueCreate(QUEUE_LEN, sizeof(INT8U));
  xSemaphore_keypad = xSemaphoreCreateMutex();

  xQueue_lcd = xQueueCreate(QUEUE_LEN, sizeof(INT8U));
  xSemaphore_lcd = xSemaphoreCreateMutex();

  xQueue_SPI = xQueueCreate(QUEUE_LEN, sizeof(INT16U));
  xSemaphore_SPI = xSemaphoreCreateMutex();
}

int main(void) {
    // Initialize hardware peripherals (e.g., GPIO) here
    setupHardware();

    xTaskCreate(SPI_task, "SPI", configMINIMAL_STACK_SIZE, NULL, LOW_PRIO, NULL);
    xTaskCreate(key_task, "keyboard", configMINIMAL_STACK_SIZE, NULL, LOW_PRIO, NULL);
    xTaskCreate(lcd_task, "lcd", configMINIMAL_STACK_SIZE, NULL, LOW_PRIO, NULL);
    // Start the FreeRTOS scheduler
    vTaskStartScheduler();
    return 0;
}
```

Figure 3: Show a part of the main.c code where the hardware is initialized and Queues, Semaphores and Tasks are created

The Task Diagram for the Tiva solution, (see fig. 4). The Tasks were distributed

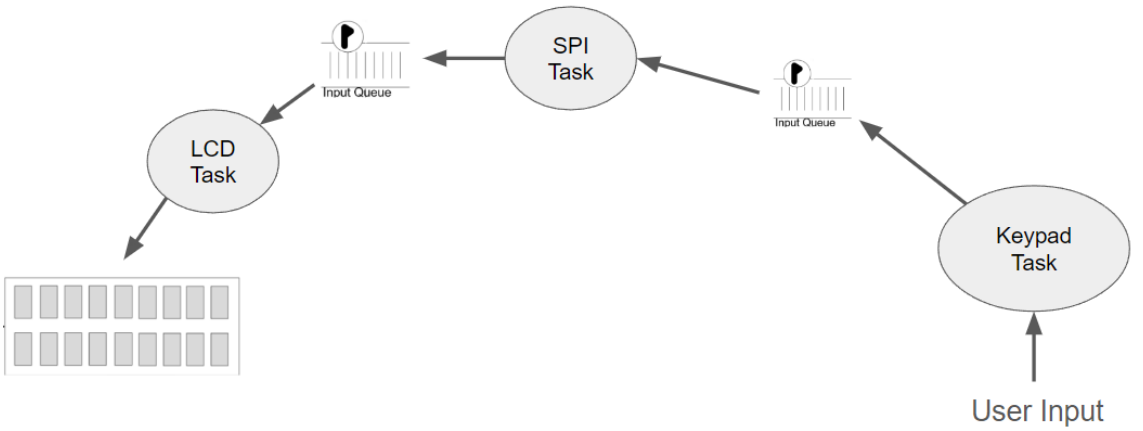


Figure 4: The figure shows a Task Diagram of the Tiva FreeRTOS solution.

3.2 SPI Communication

The SPI communication was implemented to enable a stable and reliable connection between the Tiva board (master) and the Pynq-Z2 board (slave). The Tiva board would

send data as commands for the Pynq-Z2 board and vice versa the data as pan and tilt angle in a 16 bit vector. The SPI communication was facilitated from four ports (Tiva board): SCLK, MOSI, MISO and CS_N/SS. Seen in fig. 5
Here is what each port does:

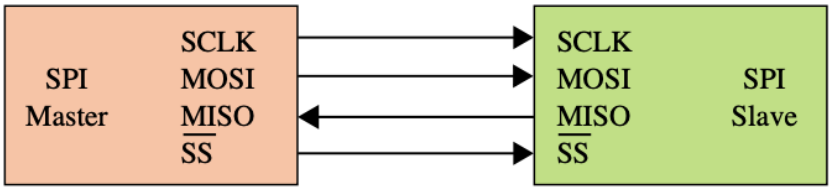


Figure 5: The diagram shows a simplistic view of the ports and how the SPI communication system works.

- **SCLK**: The Master clock, which acts as a mutual timer for both boards. The Master board uses its internal clock, while the slave uses the external clock from the Master.
- **MOSI**: Responsible for sending data serially from the Master to the Slave.
- **MISO**: Used by the Slave to send data back serially to the Master.
- **CS_N/SS**: The Slave select, which identifies the Slave device being communicated with.

3.3 SPI Design

To activate and control the individual motors in the pan-tilt system, the keypad on the extension board for the Tiva board was used, seen in fig. 6. The keypad controlled the pan-tilt movement by sending data through the MOSI line to the FPGA, where it was then processed.
Data as angle from the Pynq-Z2 was sent to the Tiva board, which was updated on the LCD display. The LCD on the extension board was utilized to display the output from the FPGA through the MISO line. The FPGA processed the data and showed the pan-tilt angles of the motor positions, as illustrated in fig. 6. This data allowed for measuring, experimenting with, and improving the system accuracy.

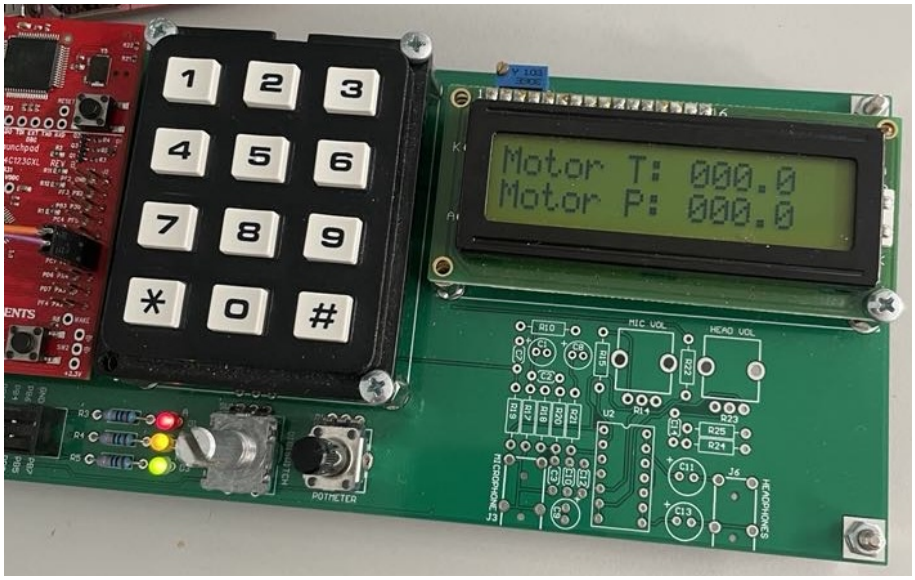


Figure 6: The user input keypad and the output data on the LCD.

To meet the SPI requirements the Freescale SPI Frame Format (Single Transfer) was chosen. Each time the Tiva board transmitted data through the MOSI line, data was simultaneously received on the MISO line. This process can be seen in fig. 7.

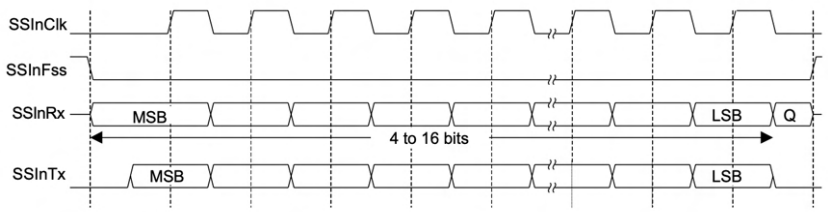


Figure 7: The diagram displays an instance of where a single transfer of 4-16 bits of data is being transmitted and received at the same time.

This way every time a key was pressed on the keyboard the slave select was set to low and the SPI clock would start transmitting commands to the slave. The angular positions were sent back to the master updating it with every key press. Once the data has been transmitted, the slave select is set to high and the clock was set to low.

3.4 SPI Implementation

To setup the SPI Master on the Tiva board, an alternate functionality called SSI had to be used. This SSI functionality is only present in specific pins on the Tiva board. The pins that have this functionality available are seen in fig. 8.

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
SSI0Clk	19	PA2 (2)	I/O	TTL	SSI module 0 clock
SSI0Fss	20	PA3 (2)	I/O	TTL	SSI module 0 frame signal
SSI0Rx	21	PA4 (2)	I	TTL	SSI module 0 receive
SSI0Tx	22	PA5 (2)	O	TTL	SSI module 0 transmit
SSI1Clk	30 61	PF2 (2) PD0 (2)	I/O	TTL	SSI module 1 clock.
SSI1Fss	31 62	PF3 (2) PD1 (2)	I/O	TTL	SSI module 1 frame signal.
SSI1Rx	28 63	PF0 (2) PD2 (2)	I	TTL	SSI module 1 receive.
SSI1Tx	29 64	PF1 (2) PD3 (2)	O	TTL	SSI module 1 transmit.
SSI2Clk	58	PB4 (2)	I/O	TTL	SSI module 2 clock.
SSI2Fss	57	PB5 (2)	I/O	TTL	SSI module 2 frame signal.
SSI2Rx	1	PB6 (2)	I	TTL	SSI module 2 receive.
SSI2Tx	4	PB7 (2)	O	TTL	SSI module 2 transmit.
SSI3Clk	61	PD0 (1)	I/O	TTL	SSI module 3 clock.
SSI3Fss	62	PD1 (1)	I/O	TTL	SSI module 3 frame signal.
SSI3Rx	63	PD2 (1)	I	TTL	SSI module 3 receive.
SSI3Tx	64	PD3 (1)	O	TTL	SSI module 3 transmit.

Figure 8: Overview of pins with the SSI-functionality

This required careful selection of pins for SPI communication, as they might already be used for other functions. With the keypad and the LCD on the extension board occupying many pins, it was essential to choose the SSI pins carefully to avoid interference with other processes.

Interface til Tiva TM4C123G evaluation Board

GPIO	*Connector*	*On Board*	*At EMP Board*
PA0	-	U0Rx	-
PA1	-	U0Tx	-
PA2	2.10	-	KEYB D
PA3	2.09	-	KEYB E
PA4	2.08	-	KEYB F
PA5	1.08	-	DIGI A
PA6	1.09	-	DIGI B
PA7	1.10	-	DIGI P2
PB0	1.03	-	PB0
PB1	1.04	-	PB1
PB2	2.02	-	PB2 / Analog output **
PB3	4.03	-	PB3
PB4	1.07	-	PB4
PB5	1.02	-	PB5 / Potmeter ***
PB6	2.07*	Connected to PD0	PB6
PB7	2.06*	Connected to PD1	PB7
PC0	-	TCK/SWCLK	-
PC1	-	TMS/SWDIO	-
PC2	-	TDI	-
PC3	-	TDO/SWO	-
PC4	4.04	-	LCD D4
PC5	4.05	-	LCD D5
PC6	4.06	-	LCD D6
PC7	4.07	-	LCD D7
PD0	3.03*	-	Always input
PD1	3.04*	-	Always input
PD2	3.05	-	LCD RS
PD3	3.06	-	LCD E
PD4	-	USB0D-	-
PD5	-	USB0D+	-
PD6	4.08	-	Status LED (Red)
PD7	4.09	NMI	-
PE0	2.03		KEYB G
PE1	3.07		KEYB H
PE2	3.08		KEYB J
PE3	3.09		KEYB K
PE4	1.05		Analog Output **
PE5	1.06	-	Analog input
PF0	2.04	SW2	
PF1	3.10	Red LED	Red LED
PF2	4.01	Blue LED	Yellow LED
PF3	4.02	Green LED	Green LED
PF4	4.10	SW1	
GND	2.01		
GND	3.02		
3.3V	1.01		
5.0V	3.01		
RESET	2.05		

Figure 9: Pin-out interface between Tiva board and extension board.

Looking at fig. 8 and fig. 9, only the pins PB4 to PB7 were available for use, as they were not occupied by other functions and had SSI functionality. The only minor detail was that the Potentiometer on the extension board had to be turned off using a jumper to free up pin PB4.

```
1   void SSI2_Init(void) {
2       int dummy;
3
4       SYSCTL_RCGCSSI_R |= (1<<2);
5       SYSCTL_RCGCGPIO_R |= (1<<1);
6
7       dummy = SYSCTL_RCGCSSI_R;
8       dummy = SYSCTL_RCGCGPIO_R;
9
10      GPIO_PORTB_AMSEL_R &= ~0xf0;
11
12      GPIO_PORTB_AFSEL_R |= 0b11110000;
13
14      GPIO_PORTB_PCTL_R |= (2<<16) | (2<<20) | (2<<24) | (2<<28);
15
16      GPIO_PORTB_DEN_R |= (1<<4) | (1<<5) | (1<<6) | (1<<7);
17
18      GPIO_PORTB_DIR_R |= 0b10110000;
19
20      SSI2_CR1_R = 0x00000000;
21
22      SSI2_CC_R = 0x00;
23
24      SSI2_CPSR_R = 2;
25
26      SSI2_CR0_R = 0x0000f;
27
28      SSI2_CR1_R |= 2;
29
30  }
31
32  void SSI2_Transfer(INT16U data) {
33
34          while((SSI2_SR_R & 2) == 0); // Bit 1 is the TNF (Transmit FIFO Not Full) flag
35          SSI2_DR_R = data;
36
37  }
```

Figure 10: Screenshot of how the SPI is initialized (SSI2_Init) on the microcontroller and the function (SSI2_Transfer) used to transmit data serially over the MOSI line.

By knowing what pins to use for SPI/SSI on the tiva board, the programming of SPI could begin. First, the SSI2 module had to be initialized on the chosen pins. This was done by setting up the SSI and GPIO clock for port , enabling alternate functions on the pins PB4-PB7. Furthermore disabling analog functionality, and enabling digital functionality. Finally, the port control registers were configured to assign PB4-PB7 for SSI2 functions, as shown in fig. 10. Specifically the code on line 26 in fig. 10 configures SSI2 for the Freescale SPI frame format with 16-bit data, as discussed in the section ”SPI Desgin”.

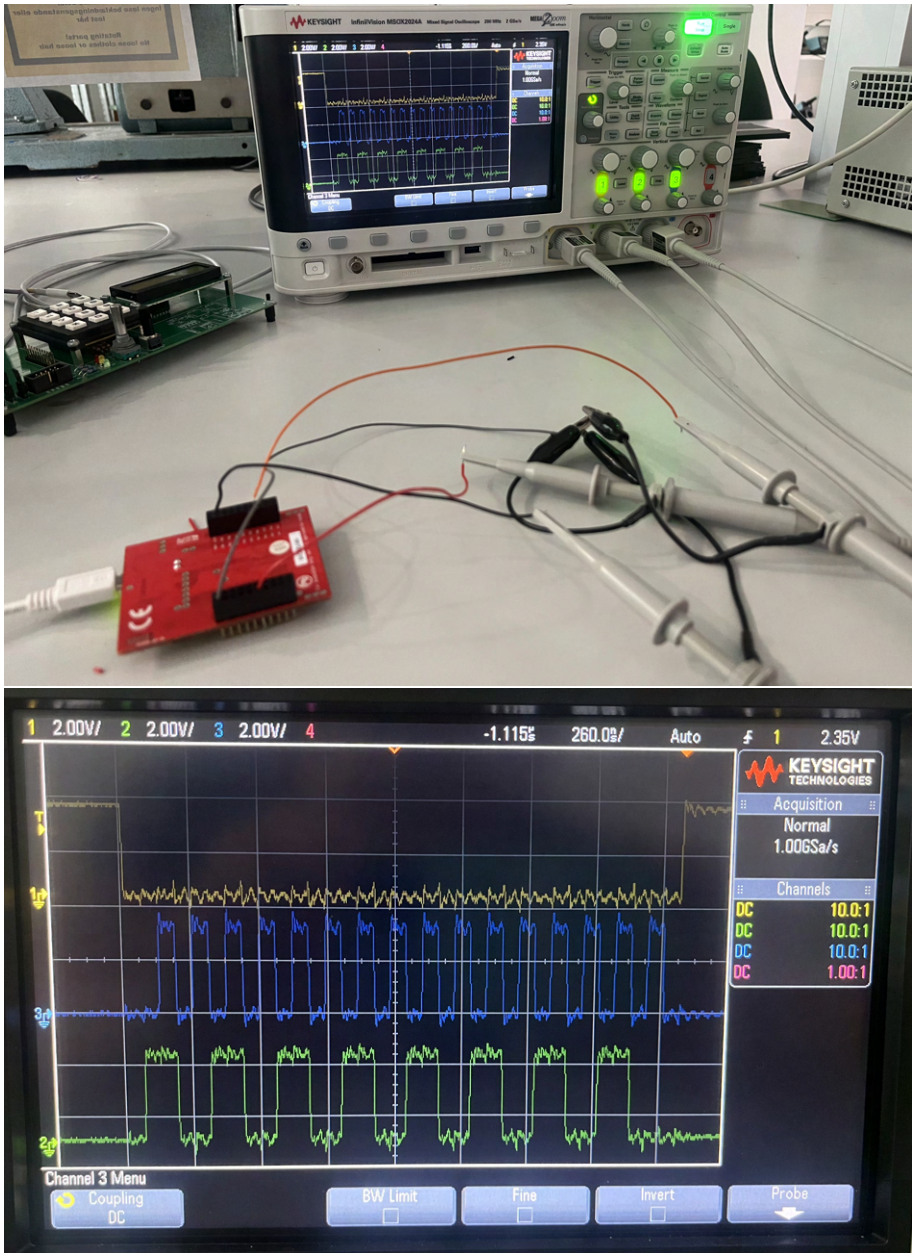


Figure 11: The pictures show how the SPI master outputs SCLK(blue) , MOSI(green) and SS(yellow) were measured with an oscilloscope to make sure that the SPI design worked.

To ensure the SPI design and the programmed microcontroller worked correctly, they were tested using an oscilloscope connected to the SPI pins, as shown in fig. 11. By looking at how the clock and the MOSI signal line up with one another, it was shown that they matched the chosen SPI design mentioned under ”SPI Design” and shown in fig. 7.

3.5 FPGA SPI

To setup the SPI communcation between the Tiva board and the FPGA, the dedicated SPI pins on the FPGA (see fig. 12) were connected with their matching pin functionality on the Tiva board using wires.

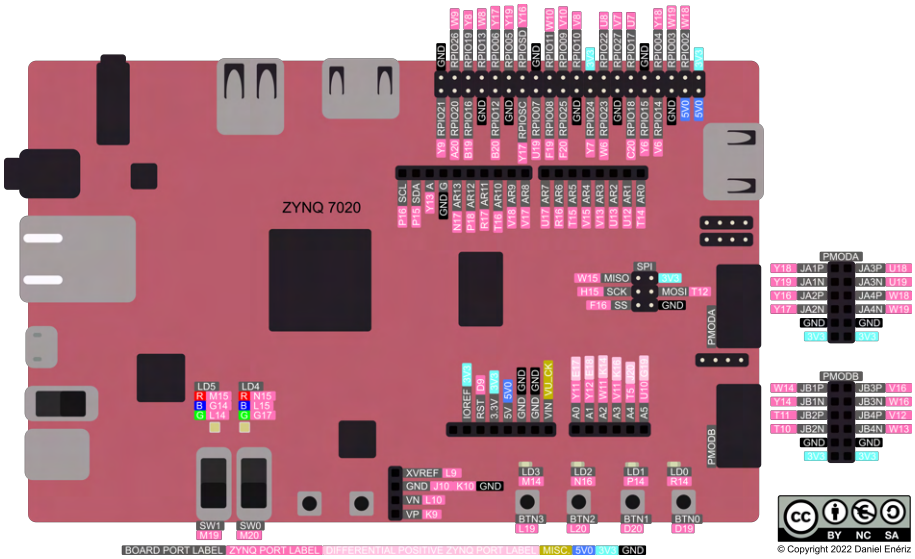


Figure 12: The figure shows PYNQ-Z2 Pinout and the used SPI pins are named[3].

To then configure the FPGA as the SPI slave, an interface was created using VHDL code with the SPI ports SCLK, CS_N (SS), MOSI and MISO shown in fig. 13.

```
8   entity SPI_SLAVE is
9       Generic (
10          WORD_SIZE : natural := 16 -- size of transfer word in bits, must be power of two
11      );
12      Port (
13          CLK      : in  std_logic;
14          -- SPI SLAVE INTERFACE
15          SCLK     : in  std_logic;
16          CS_N     : in  std_logic;
17          MOSI     : in  std_logic;
18          MISO     : out std_logic;
19
20          -- MOTOR CONTROL
21          hallsensor1A, hallsensor1B, hallsensor2A, hallsensor2B, sensor0, sensor1 : in STD_LOGIC;
22          IN1A, IN2A, IN1B, IN2B : out STD_LOGIC;
23          ENA, ENB : out STD_LOGIC
24
25
26      );
27  end entity;
```

Figure 13: The SPI slave interface.

To prevent meta stability the PynqZ2 CLK and the SCLK from the Tiva board were synchronised such that they would not interfere with each other (see fig. 14).

```
107         -- Synchronization reg to eliminate metastability.
108         sync_ffs_p : process (CLK)
109         begin
110             if (rising_edge(CLK)) then
111                 sclk_meta <= SCLK;
112                 cs_n_meta <= CS_N;
113                 mosi_meta <= MOSI;
114                 sclk_reg  <= sclk_meta;
115                 cs_n_reg  <= cs_n_meta;
116                 mosi_reg  <= mosi_meta;
117             end if;
118         end process;
```

Figure 14: The synchronization process.

To store the data transmitted by the master over the MOSI line, a process was created to execute on every rising edge of the CLK, similar to other processes. It detected when a rising edge on the SCLK occurred and the SPI slave was selected by CS_N being low. If these conditions were met, each bit from the MOSI line was shifted into the data_to_use register on every rising edge of SCLK. The code for this process is shown in fig. 15.

```
196    data_into_reg_p : process (CLK)
197    begin
198    if (rising_edge(CLK)) then
199     if (spi_clk_redge_en = '1' and cs_n_reg = '0') then
200      data_to_use <= data_to_use(WORD_SIZE-2 downto 0) & mosi_reg;
201     end if;
202    end if;
203    end process;
```

Figure 15: The process of storing the transmitted data from MOSI.

The remaining steps involved fully implementing the SPI slave protocol on the FPGA. These steps include clock edge detection, bit counting, data shift registers, MISO output, data validation, and ready flags. Detailed implementations of these processes are provided in the VHDL code found in the appendix.

4 Motor Movement

This section was dedicated to the motor movement and better understanding of the slave boards usage of motor movement.

4.1 Hardware

The motor movement was controlled by the slave Pynq Z2 board, which received input signals via SPI from the master Tiva board. The following points were considered: Which ports could be used for motor movement, which ports for receiving motor encoder signals, and how to achieve the desired motor speed.
The slave board ports PMODA and RASPBERRY PI GPIOs were used to connect the motors and sensors/encoders to the hardware side of the slave board. The following components are shown in figure 16, figure 18 and figure 19.

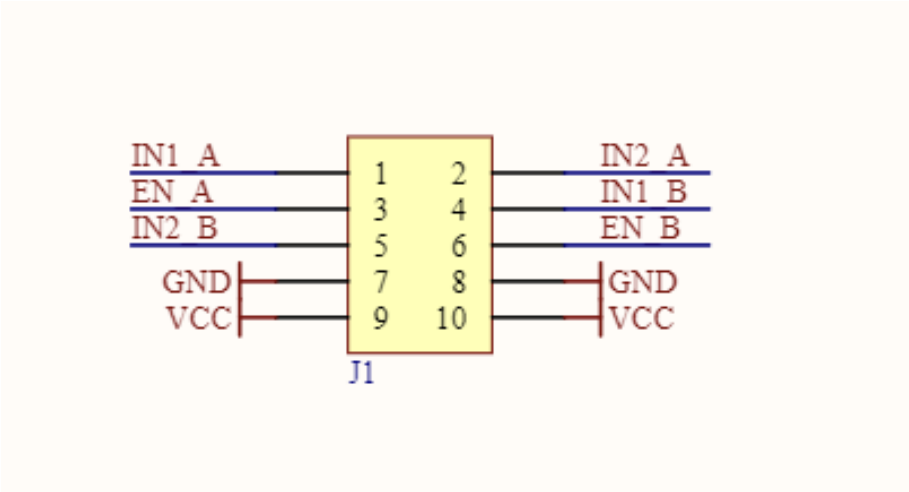


Figure 16: Motor inputs for control. Taken from Dual H-bro_V2.pdf in Appendix.

The IN1 A and IN2 A were responsible for controlling the direction and speed of motor A. So does IN1 B and IN2 B for motor B. As an example the IN1 A was set to 0 and IN2 A was set to the pwm_output for the motor to spin in one direction. When the IN1 A was set to pwm_output and IN2 A was set to 0, then the motor A would spin in the other direction. As seen in the fig. 17 below.

```
when x"0005" =>
    ENA <= '0';
    IN1A <= '0';
    IN2A <= '0';
    ENB <= '1';
    IN1B <= pwm_output;
    IN2B <= '0';
```

Figure 17: An example of code implementation for the motor movement.

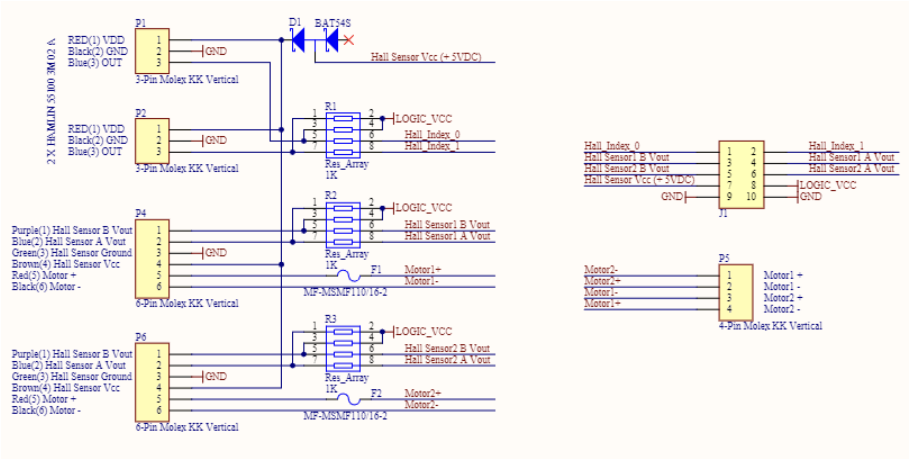


Figure 18: Motor outputs for the sensors, see J1 at the top right. Schematics of the EMP-30 protection. Taken from EMP30-protection.pdf in Apendix.

The EMP-30 protection circuit was solely used for the implemetation of the motor encoders and two magnetic sensors (see fig. 18).

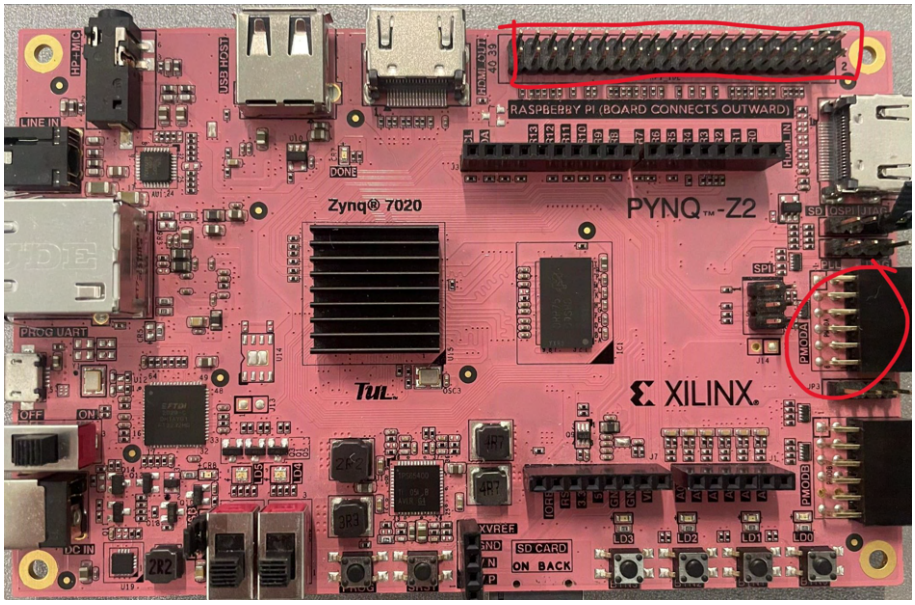


Figure 19: Picture of the ports used for motor control and intercepting the sensor signals.

The table below provides an overview of all the ports used for motor control and their respective functionalities.

Port	**Function**
IN1A, IN2A and ENA	Ports used for motor A pan movement. ENA is responsible for enabling/disabling the motor. The other two ports control the direction and speed of the motor.
IN2B, IN2B and ENB	Same functionality as above, but for motor B tilt movement.
Hall Index 0	Connection to the magnetic sensor tilt
Hall Index 1	Connection to the magnetic sensor pan
Hall Sensor 1A and Hall Sensor 2A	The encoder output for motor A
Hall Sensor 1B and Hall Sensor 2B	The encoder output for motor B

4.2 Code Implementation

The slave board was programmed in VHDL. The VHDL code was implemented to send and receive signals through the ports mentioned in section 4.1. The working process of the system functioned by creating a PWM signal for the motors to move.

Figure 20: The encoder signals shown are phased 90 degrees to show the direction of the motor movement (look only at signals 2 and 3).

The fig. 20 shows the encoder signals. As displayed they are identical for both motors. The signals were connected to the slave board and implemented in the code. The total_pulse value was either incremented or decremented based on the encoder signals. The value was incremented when a rising edge appeared in the signal 2 (see green line fig. 21) and signal 3 (see blue line fig. 21) when set to high or 1. It was decremented when signal 3 was set to low or 0. It functioned in that way, because of the systems requirements for getting the actual value of the position.

Then the pan and tilt values were changed from integers to a vector such that they could be sent through SPI (look section section 3). The tilt_vector was 9 bit and pan_vector was 7 bit which together formed a 16 bit vector.

```
    process(hallsensor2B, hallsensor1B)
begin
    if rising_edge(hallsensor2B) then
        if hallsensor2B = hallsensor2A then
            total_pulse_counter_tilt <= total_pulse_counter_tilt + 1;
        else
            total_pulse_counter_tilt <= total_pulse_counter_tilt - 1;
        end if;
        pulse_counter_tilt_vector <= std_logic_vector(to_unsigned(total_pulse_counter_tilt, pulse_counter_tilt_vector'length));
    end if;

    if rising_edge(hallsensor1B) then
        if hallsensor1B > hallsensor1A then
            total_pulse_counter_pan <= total_pulse_counter_pan + 1;
        else
            total_pulse_counter_pan <= total_pulse_counter_pan - 1;
        end if;
        pulse_counter_pan_vector  <= std_logic_vector(to_unsigned(total_pulse_counter_pan, pulse_counter_pan_vector'length));
    end if;

end process;
```

Figure 21: The algorithm to intercept the encoder signals.

The PWM output signal was generated using the internal clock, with a PWM period set to 1 MHz and a specific duty cycle for that period. Two PWM signals were created for each motor as the tilt motor needed more voltage to operate. The tilt motor was set to a 45% duty cycle, while the pan motor was set to a 20% duty cycle. The reason behind having set values for the PWM period and duty cycle was for the open-loop control system. That way the pulses from the motor encoders would be steady and the slave board would calculate its position based on the encoder feedback.

The PWM output was generated using the internal clock and a pwm_counter variable. The PWM output is set to 1 when pwm_counter is less than the duty cycle, and 0 when pwm_counter exceeds the duty cycle. (see fig. 22).

The FPGA PYNQ Z2 board used a double H-Bridge to operate the motors.

```
58      -- PWM variables and so on
59      constant PWM_PERIOD : integer := 1000000;
60      signal pwm_counter: integer range 0 to PWM_PERIOD - 1 := 0;
259 PWM : process(CLK)
260 begin
261      if rising_edge(CLK) then -- This is for the motor B tilt
262         pwm_counter <= pwm_counter + 1;
263         if pwm_counter < duty_cycle then
264             pwm_output <= '1';
265         else
266             pwm_output <= '0';
267         end if;
268         -- This one for motor A pan
269         if pwm_counter < duty_cycleA then
270             pwm_outputA <= '1';
271         else
272             pwm_outputA <= '0';
273         end if;
274     end if;
275 end process;
```

Figure 22: VHDL code implementation of creating the PWM output.

4.3 Control loop system

The control loop was responsible for enabling the motors and setting the desired PWM output for each, as seen in fig. 23. The input signal was generated by a button press sent through SPI. The controller in the VHDL code identifies which button was pressed and starts the corresponding moving process. The output signal would then be initialized to either motor tilt or motor pan.

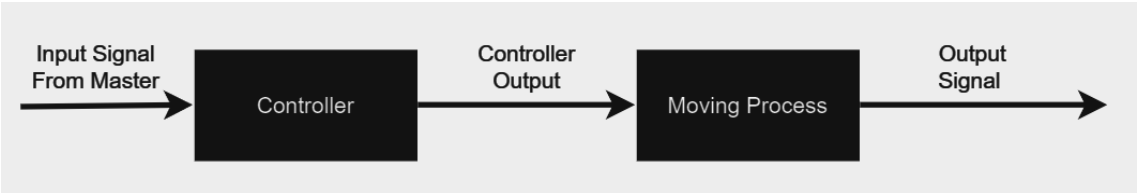


Figure 23: The diagram of the open-loop control.

5 Optimization

This section is dedicated to the potential improvements considered to enhance its efficiency and effectiveness.

5.1 Motor movement

An improvement to be made in the motor movement section was to make a speed regulator with a closed-loop system. The regulator would adjust the duty cycle based on the input signal and output feedback. More about that in section 6.

5.2 SPI communication

The MISO register which sent pan and tilt values from the slave to master board could be improved by continuously sending these values rather than only when a button on the master board is pressed. This would ensure that the master board always has the most up-to-date position data, improving the overall accuracy of the control system. The benefits include:

- **Real-Time Updates:** Providing the master board with continuous real-time updates of the motor positions.
- **Improved Accuracy:** Reducing latency in data transmission, resulting in more accurate control.
- **Enhanced Responsiveness:** Allowing the master board to react more quickly to changes in motor positions.

6 System modelling

This section was dedicated for the potential closed-loop control of the system.

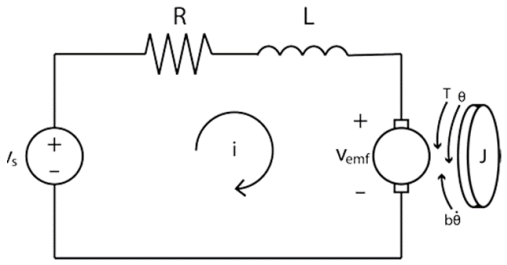
6.1 Transfer function

In order to make a PID controller, a mathematical model has to be made. These are used to to describe the DC-motor output with the input. This is used for the voltage,

current and positon. These can be found from the system fig. 24b and the parameters in fig. 24a.

R	Resistance
L	Inductance
E	Electromotive force
T	Motor friction force
θ	Motor Position
B	Friction in motor
$\dot{\theta}$	Angular velocity
J	Inertia
I	Current
V_S	Input voltage

(a) EMG30 motor parameters



(b) Electric motor model

Figure 24: EMG30 motor parameters and electric motor model

From this, it can be seen that ”back emf” (which is the electromotive force) can be described as:

$$V_{\text{emf}} = K_{\text{emf}} \cdot \dot{\theta} \tag{1}$$

where K_{emf} is the electromotive force constant. This means Kirchhoff's voltage law can be applied to derive the following formula:

$$L_i \cdot R_i = V_s - V_{\text{emf}} \tag{2}$$

If it is assumed that the magnetic field in the motor remains constant, then the driving force generated by the motor will be proportional to the current running through the motor. The driving force can therefore be described as:

$$T = K_t \cdot i \tag{3}$$

where K_t is described as the motor driving force constant and i is the current flowing through the motor. Newton's 2nd Law can now be applied:

$$J\ddot{\theta} + b\dot{\theta} = T \tag{4}$$

Here $K_t = K_{\text{emf}}$. These can therefore both be described as K, which is defined as the motor constant. Equations 2 and 4 will now be Laplace transformed from the time domain to the frequency domain to find the transfer functions for position, velocity,

and current:

$$I(s)(Ls+R) = V_s(s) - K\Theta(s)s \tag{5}$$

$$J\Theta(s)s^2 + b\Theta(s)s = KI(s) \tag{6}$$

To find the transfer function, $I(s)$ is isolated in equation 6, which then gives:

$$I(s) = \frac{s(Js+b)}{K}\Theta(s) \tag{7}$$

Now equation 7 is to be placed in equation 5:

$$\frac{\Theta(s)}{s(Js+b)/K(Ls+R)} = \frac{V_s(s) - K\Theta(s)s}{K} \tag{8}$$

This can be recalculated to the transfer function for position, regarding the voltage:

$$P_{\text{position}}(s) = \frac{\Theta(s)}{V_s(s)} = \frac{K}{s((Js+b)(Ls+R)+K)} \quad [\text{rad/V}] \tag{9}$$

If velocity is to be determined, you can differentiate the equation above, which would be the same as multiplying s on the right side of the equation:

$$P_{\text{velocity}}(s) = \frac{\Theta(s)}{V_s(s)} = \frac{K}{(((Js+b)(Ls+R)+K))} \quad [\text{rad/V}] \tag{10}$$

The current in relation to the voltage can be found with the same method; simply isolate $\Theta(s)$ in equation 6:

$$\Theta(s) = \frac{KI(s)}{(Js^2+bs)} \tag{11}$$

Equation 11 is now inserted into equation 5 and is solved for $\frac{I(s)}{V_s(s)}$. The transfer function for the current in relation to the voltage is then given by [4]:

$$P_{\text{voltage}}(s) = \frac{I(s)}{V_s(s)} = \frac{1}{(Ls+R) + \frac{K^2}{(Js^2+bs)}} \tag{12}$$

6.2 Modelling parameters

When creating a mathematical model of the pan tilt system, there are four extra parameters needed when modelling the engine. These can be seen in table 2.

Parameter	Value
Resistance	6.8144
Inductance	3.3475 mH
Engine friction	0.000931
Motor axis Inertia	0.0019 kg $\cdot$ m^2

Table 2: Model of the engine

Resistance and inductance are measured with an LCR-meter, while motor friction and axis inertia are found from the EMG30 article[1]. Another important parameter is the constant of the electromotive force and engine traction force. This constant can be calculated by measuring the engine speed without resistance and converting it to rad/s. This comes out to 22.619 rad/s. This radian value is then divided by the maximum voltage:

$$\frac{12\,\mathrm{V}}{22.619\,\mathrm{rad/s}} \approx 0.531$$

In a pan-tilt system, the inertia can vary depending on its position. To better understand the system's performance under different conditions, both the best-case and worst-case scenarios are calculated. This can be seen below in fig. 25.

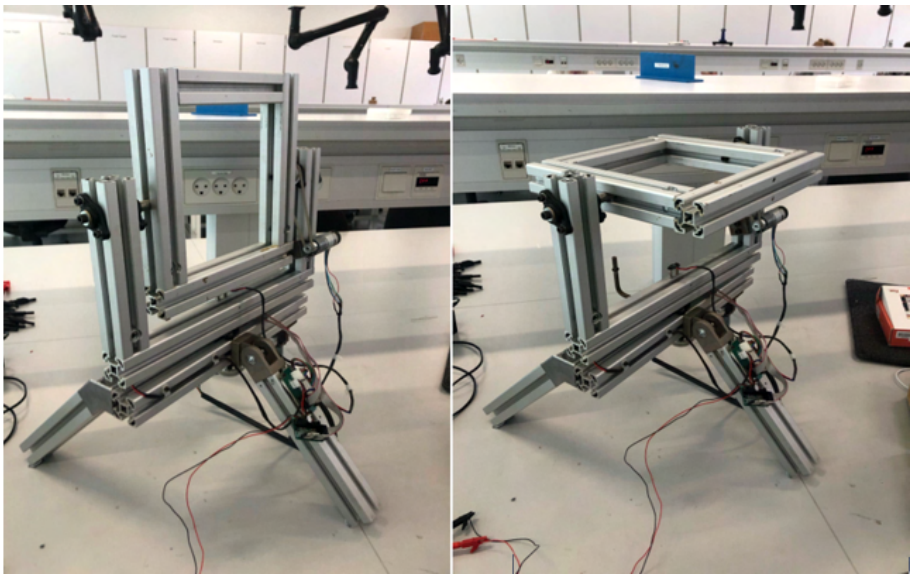


Figure 25: best case to the left, and worst case to the right.

The system inertia is calculated below, and the values are summarized in table 3.

Tilt	0.0263 kg·m^2
Pan best case	0.0889 kg·m^2
Pan worst case	0.0946 kg·m^2
Motor axis Inertia	0.0019 kg·m^2

Table 3

In our pan-tilt system, the moment of inertia varies with position, affecting the system's dynamics. A PID controller could adapt to these changes, ensuring precise control by continuously correcting errors. To calculate this the mass of the rod the volume is used and multiplied with the density of aluminum. The inertia of the rod on the tilt axis is divided into two separate. One inertia for the vertical and one for the horizontal. The horizontal and the vertical are both calculated as point masses. Inertia of the bottom and top rod are calculated with mass m_{d1} and the length d1.

Inertia for the point mass m_{d1} of the horizontal rods with the length d_1:

$$I_{\text{tilt}_1} = \left(\frac{1}{12} m_{d1} d_1^2\right) \cdot 2$$

Inertia for the point mass m_{d2} of the horizontal rods with the length d_2:

$$I_{\text{tilt}_2} = (m_{d2} r_1^2) \cdot 2$$

Total inertia for the tilt mass:

$$I_{\text{tilt}_{\text{total}}} = I_{\text{tilt}_1} + I_{\text{tilt}_2}$$

The inertia is first calculated for the bottom pan, then a best case and worst case for the tilt.

Inertia of bottom rod, with mass m_{d5} and length d_5:

$$I_{\text{base}_1} = \left(\frac{1}{12} m_{d5} d_5^2\right) \cdot 2$$

Inertia for the dot mass m_{d6} of the horizontal rods with the length d_6:

$$I_{\text{base}_2} = (m_{d6} r_4^2) \cdot 2$$

Total inertia for the base:

$$I_{\text{base}_{\text{total}}} = I_{\text{base}_1} + I_{\text{base}_2}$$

At the base there is a worst-case and best-case scenario. This is calculated in formulas (26) and (29).

Best-case for top and bottom masses m_{d3} with the rod with the length d_3:

$$I_{\text{pan}_{\text{best}_1}} = \left(\frac{1}{12} m_{d3} d_3^2 \right) \cdot 2$$

Best-case for the dot mass m_{d4} and the horizontal rod with the length d_4:

$$I_{\text{pan}_{\text{best}_2}} = (m_{d4} r_4^2) \cdot 2$$

The total inertia for best-case:

$$I_{\text{pan}_{\text{best}_{\text{total}}}} = I_{\text{base}_{\text{total}}} + I_{\text{pan}_{\text{best}_1}} + I_{\text{pan}_{\text{best}_2}}$$

Worst-case is calculated from 4 dot masses, with the masses m_{d1} and m_{d2} with the rod lengths d_1 and d_2:

$$I_{\text{worst}_1} = (m_{d1} r_4^2) \cdot 2$$

$$I_{\text{worst}_2} = (m_{d2} r_3^2) \cdot 2$$

Worst-case inertia:

$$I_{\text{worst}_{\text{total}}} = I_{\text{base}_{\text{total}}} + I_{\text{worst}_1} + I_{\text{worst}_2}$$

6.3 PID controller

The transfer function for the model, will allow for simulation in Matlab Simulink, which will make sure that the PID controller gets the correct Proportional, integral and derivative values. in this case, P = 10, I = 1 and D = 7. The PID controller can be seen in fig. 26.

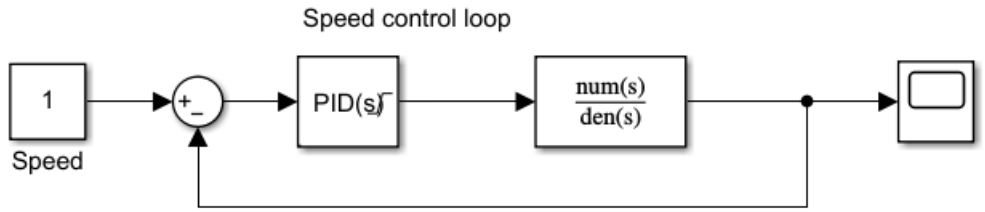
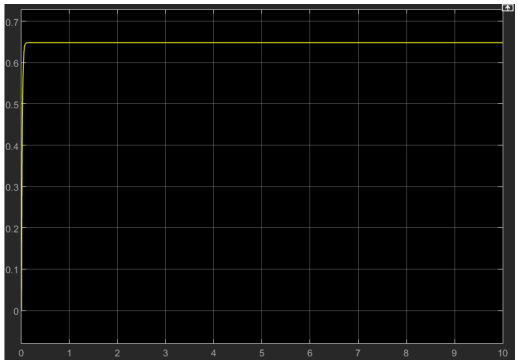
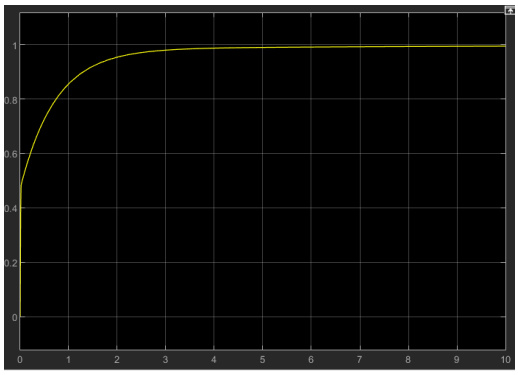


Figure 26: The PID controller

The reason for this configuration is to have a balance between rise time, settling time, while eliminating overshoot and steady state error. This can be seen in fig. 27, where in fig. 27a, the PID is set to P = 1, I = 0 and D = 0, and in fig. 27b. The PID controller er set to the values, 10, 1 and 7 [2].



(a) PID with the values 1, 0 and 0



(b) PID with the values 10, 1 and 7

Figure 27: PID controller configured and not

The figures above show the PID controller with the first configuration, and the final configuration. In the first there was a really fast rising time and settling time. However the values never hit 1 which was the preferred speed, therefore another PID controller was made, with the configurations as seen in fig. 27b, where there was a little slower rising time but it hits 1.

7 Experiments

This section focuses on verifying the system's performance.

7.1 Experiment: Angle Position Check

The purpose of this experiment was to verify if the physical angle matched the angle displayed on the Master LCD. The system was manually tilted to complete 8 full rotations, then stopped at the magnetic sensor to ensure the physical angle was 0 degrees. The main objective was to demonstrate that, under these conditions, the Master LCD would always display 0 degrees.

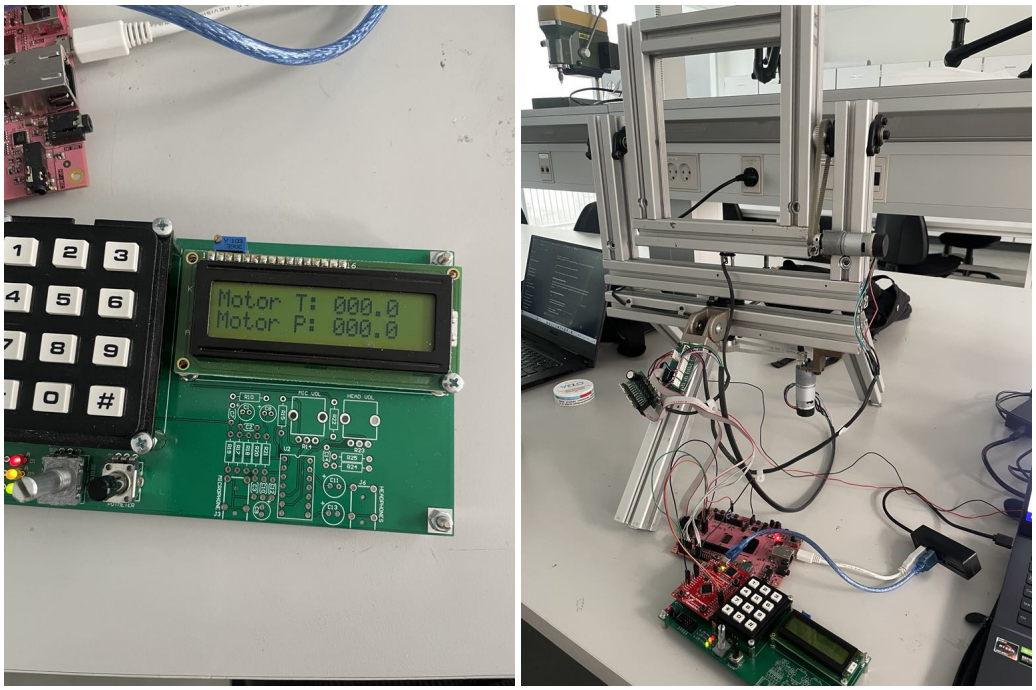


Figure 28: Set up of experiment 1.

The table beneath showcases the whole experiment.

Experiment number	Result
1	The LCD showed angle 0
2	The LCD showed angle 1.3
3	The LCD showed angle 1.3
4	The LCD showed angle 1.3
5	The LCD showed angle 0
6	The LCD showed angle 0
7	The LCD showed angle 1.3
8	The LCD showed angle 1.3
9	The LCD showed angle 1.3
10	The LCD showed angle 1.3

The actual value was calculated by subtracting the largest angle showed on the LCD with 360.

$$\frac{\text{Actual Value} - \text{Theoretical Value}}{\text{Theoretical Value}} \times 100 = \%\ \text{deviation}$$

$$\frac{358.7 - 360}{360} \times 100 = -0.36\%$$

The largest percentage deviation in this experiment was 0.36%.

7.2 Experiment: Voltage to Motor

This subsection was made to see if the motors received the desired voltage. The main reason for this experiment was to understand and verify the hardware part of the system.

$$V_{\text{max}} \times \text{Duty Cycle} = V_{\text{avg}}$$

The equation above, explains the relationship between Average Voltage, Maximum Voltage and Duty Cycle. The maximum voltage given to the motors was 12 V and the duty cycle of the tilt motor was 45 %. The calculation of the theoretical average voltage can be seen beneath.

$$12V \times 0.45 = 5.4V$$

The theoretical average voltage was calculated to 5.4V and the actual average voltage given to the motors was 4,96 V. This gives a percentage deviation of 8.15 %.

$$\frac{\text{Actual Value} - \text{Theoretical Value}}{\text{Theoretical Value}} \times 100 = \%\ \text{deviation}$$

$$\frac{4.96 - 5.4}{5.4} \times 100 = -8.15\%$$

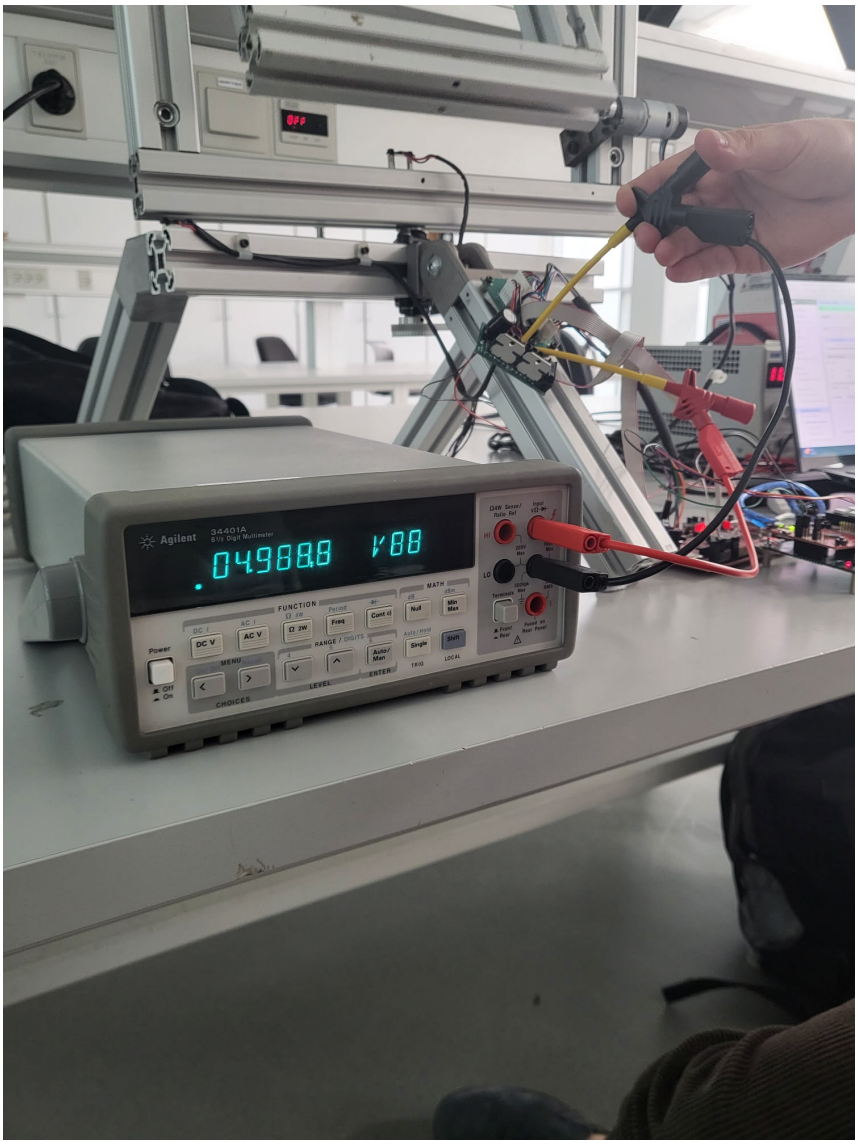


Figure 29: Set up of experiment 1.

8 Discussion

The PWM signals used, proved best with a duty cycle of 45 percent for the tilt motor and 20 percent for the pan motor. These values made the control reliable in both the tilt and pan direction. If a camera was added to the system these values might have to be tweaked a bit to account for the added weight of the camera and the overall change in moment of inertia. A control loop for the motors could be implemented using a gyroscope and a control algorithm, similar to how a Radio-controlled (RC) car uses a gyroscope to counter-steer. The gyroscope could continuously monitor the angular velocity of the pan-tilt system. If it detects unintended rotations in the pan or tilt direction, the control algorithm calculates the necessary gain to correct and adjusts the motor positions to stabilize the camera and maintain the desired orientation.

For instance, if the pan-tilt system is mounted on a moving vehicle or subject to vibrations, the gyroscope would detect these rotational movements. The control system could use the gyro data and adjust the motors to counteract the disturbances, keeping the camera steady and focused. This automatic stabilization enables high-quality video capture and precise monitoring, even in challenging moving conditions.

The Angle Position Check experiment had a deviation of 0.36%, which could be ascribed to not using a closed-loop system. The other reason for the deviation could be the misreading of the encoder signals by the slave board.

The results of the voltage experiment demonstrate that while the PWM signals provided reliable control, there was a minor deviation of 8.15% in the actual voltage received by the motors compared to the theoretical calculations. This deviation could be attributed to several factors, including potential losses in the circuitry due to the components used, inaccurate calibrations, variations in the power supply, or inaccuracies in the measurement process.

With the SPI, it was discussed whether or not the MISO should be continuously on instead of only on when the corresponding button was pressed. A couple of benefits were in mind like; always up-to-date positioning, improved accuracy and greater response time. These are just some anticipated benefits; the reality of these improvements is uncertain and further testing is required.

The LCD used in this project provided basic output functionalities, displaying the pan and tilt angles of system. However, the small screen size limited to 16 characters restricted the amount of information that could be displayed simultaneously. This made it challenging to provide detailed feedback or additional system status information to the user. With a better screen, such as a larger LCD or an OLED display, more information could be presented at once. A more sophisticated screen could support a menu-driven interface, allowing users to navigate through settings, calibrations, and system logs more efficiently, improving the overall user experience.

9 Conclusion

The aim for this project was to make a manually controlled pan-tilt system for security camera usage, which sends its input from the Master board to the Slave board through SPI. The SPI protocol was implemented on the Master board, which sends the input data through the MISO register. The Slave board sends the angle of the pan and tilt of the system through the MOSI register (see section 3). The data size was 16 bit, such that the first 9 bits were for the tilt and the last 7 bits were used for the pan of the system.

The motor movement was implemented on the slave board. The base for the movement was the input data sent through the MISO register. The generation of the pwm_ output for each of the motors was based on the pwm_period and duty cycle. The implementation of the motor movement was written in VHDL. The ports used for the motor movement were IN1A, IN2A, ENA, IN1B, IN2B and ENB which functionality is described in section 4.

The experiments were designed to verify the system's performance. In the position angle check experiment, which had a deviation of 0.36 %, the results showed that the system could accurately track its angle position. The voltage experiment had a deviation of 8.15%. These experiments are discussed in section 8.

An open-loop system was implemented as control, the input from the Master board would be recognized by an algorithm in the Slave board. The algorithm would then set the desired pwm_output to one of the motors and enable the motor which should be used. Although a PID controller was not used, a thorough description of how it could be integrated to enhance the stability of the system was provided. This theoretical framework lays the groundwork for future development and optimization of the control loop system.

10 Appendix

The attached zip file contains, a folder with the Master code and a folder with the Slave code for the boards.
And the following pdf:

- EMP30-protection.pdf
- TivaTM4C123GH6PMMicrocontrollerDatasheet.pdf
- PYNQ_Z2_User_Manual_v1.pdf
- Dual H-bro_V2.pdf

11 References

[1] Data sheet for emg30. ”https://www.robot-electronics.co.uk/htm/emg30.htm”. Accessed: 12-05-2024.

[2] Matlab simulink. https://ctms.engin.umich.edu/CTMS/index.php?aux=Home. Accessed: 17-05-2024.

[3] eneriz daniel. Pynq z2 pinout. https://discuss.pynq.io/t/pynq-z2-pinout/4256, May 2022. Accessed: 03-05-2024.

[4] Katsuhiko Ogata. *Modern Control Engineering*. 5th edition.