Quadcopter
Project documentation

Dominik Michalczyk, Kacper Filipek

January 2024

1. Project description

The goal of the project is to create a working model of a quadcopter UAV, which can be controlled with a Xbox controller. The project utilises the MPU6050 inertial measurement unit, which combines MEMS accelerometer and gyroscope into a single IC. The combination of sensor readings from the IMU combined with an adequate mathematical model and a regulating system allows to make a quadcopter with a stable control system.

2. Problem analysis

For the quadcopter to be stable, it needs to continuously measure the sensor data and estimate its attitute angles and correct to the set position and drive the motors in a way that reduces deviations from the user-set attitude.

Reading the sensor data is done by reading from specific registers of MPU6050. Adresses and descriptions of all registers are written in detail in the datasheet of the IC. A library for using IC must be written.

In order to control the drone, it needs to have a wireless communication circuit aboard. The nRF24 module allows for digital radio communication on 2.4GHz band.

3. Project implementation

The MCU used in the project implementation is STM32F103C6T6, an ARM Cortex-M3 microcontroller, containing 32Kbytes of ROM. F1303 has built-in hardware support for I^2C and SPI busses used for data transmission between on-board peripherals.

The MPU6050 has been used as a sensor. It is an inertial measurement unit (IMU), a MEMS integrated circuit combining in itself an accelerometer and a gyroscope. For easy and intuitive reading of the sensor data at the code level, a library for initialising and handling the MPU has been written. It uses the built-in I^2C bus which is used to exchange data between the IMU and MCU. The transmission is made in 400kHz mode. In order to save MCU execution time, a DMA controller has been used, allowing for sending data from RAM to the peripheral.

Similarly, a library for nRF24L01 radio module has been written. nRF is being used for digital radio transmission between the UAV and a PC. It is connected to the MCU by the SPI bus and also uses the DMA mechanism. Another nRF connected to the PC is used to communicate with the quadcopter. The transceivers are used in order to read telemetry data and for piloting the UAV using an Xbox controller connected to the PC.

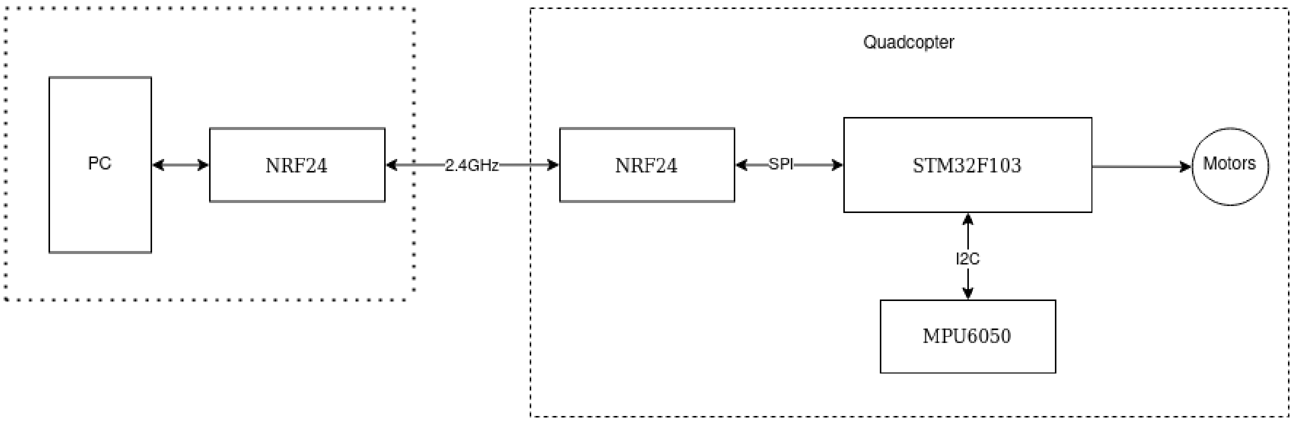
To keep the UAV stable it is essential to keep it at a set (stable) angle during movement. For instance, when we want the drone to fly up vertically, the roll and pitch angle have to be set to 0 degrees. The IMU supplies the data about displacement and angular velocity and a suitable mathematical model processes the data in order to estimate the angles.

Finally the integration of data from both sensors with the help of filtration algorithms, such as complementary filter or Kalman filter allows us to gain precise information about the angles and for effective stabilisation of the quadcopter during flight. In our project we used the Kalman filter. It is an aimed at estimating the actual state of the system based on the observer measurements with Gaussian noise. The algorithm works in two main steps: estimation of the state of the system based on previous state and correcting it during the next measurement. Kalman filter uses the mathematical model and data about Gaussian noise in order to minimalise the estimates and measurement unceirtanties.

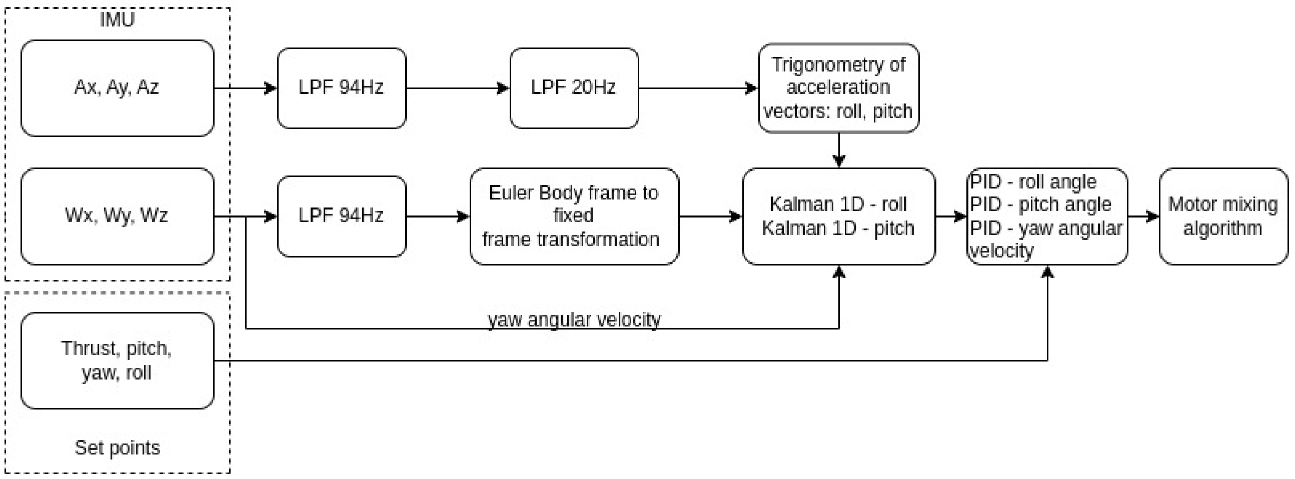
The angle estimates and yaw angular velocity is fed into the regulation system consisitng of three PID regulators for every quantity. The PID regulator works by minimising the difference between the actual state of the system and a desirable reference state. It contains three components: proportional (P), which reacts proportionally to current deviateion, integral (I), which accumulates the deviation as time passes and differential (D), which takes into account the rate of change of the difference. The output of the PIDs is a duty cycle fed into a signal mixer, which sets the duty cicle for each of the motors.

4. Block diagrams

A simplified block diagram depicting the wiring of the circuit and its connection to PC



The code for reading the sensor data and motor regulation is made according to the block diagram below.



5. Code description

5.1. nrf24l01 library

- `HAL_StatusTypeDef NRF24L01_Init(NRF24L01_STRUCT *nrf24l01, NRF24L01_CONFIG *nrf24l01_cfg);`
  - Initializes the 2.4Ghz module and sets the default setting

- `HAL_StatusTypeDef NRF24L01_Enable_ACKN_Payload(NRF24L01 *nrf24l01);`
  - Enables acknowledge payloads support. Thanks to it, you can immediately send a message to the radio that previously transmitted the data.

- `HAL_StatusTypeDef NRF24L01_Open_Reading_Pipe(NRF24L01_STRUCT *nrf24l01, uint8_t pipeAddr, uint64_t rxAddr, uint8_t payloadSize);`
  - Opens a pipe through which messages from the transmitter will be received.

- `HAL_StatusTypeDef NRF24L01_Get_Info(NRF24L01_STRUCT *nrf24l01);`
  - Reads all nrf24l01 status registers, used for debugging.

- `HAL_StatusTypeDef NRF24L01_Read_PayloadDMA(NRF24L01_STRUCT *nrf24l01, uint8_t len);`
  - Reads the payload sent by the transmitter using SPI in combination with the DMA (direct memory access controller).

- `void NRF24L01_Read_PayloadDMA_Complete(NRF24L01_STRUCT *nrf24l01, uint8_t *data, uint8_t len);`
  - After receiving data from the radio using DMA, it saves it to the given buffer.

- `HAL_StatusTypeDef NRF24L01_Write_ACKN_Payload(NRF24L01_STRUCT *nrf24l01, void *data, uint8_t len);`
  - Saves data that will be sent after receiving the next message from the transmitter.

- `HAL_StatusTypeDef NRF24L01_Start_Listening(NRF24L01_STRUCT *nrf24l01);`
  - Starts listening for incoming packets and enables interrupts from nrf24l01.

- `void NRF24L01_Stop_Listening(NRF24L01_STRUCT *nrf24l01)` - Stops listening for incoming packets and disables interrupts from nrf24l01.

5.2. rc library

- `void RC_Receive_Message(uint8_t message[8], RC_t *rc)`
  - Turns the received message into a control structure that is used to control the quadcopter

5.3. angle_estimation library

- `void Estimate_Angles_Init(float dt, float alpha, float tau)`
  - Initialize Angle estimator: dt - sampling time, alpha - coefficient for complementary filter, tau - coefficient for accelerometer lowpass filters, $\tau = \frac{1}{2\pi f}$.

- `void Estimate_Angles(float angles[2], float angular_velocities[3], float acc_buf[3], float gyro_buff[3])`
  - Calculates angles using trigonometry of acceleration vectors and transforms body frame

to fixed frame using Euler rotation matrix. Finally, it combines these two readings using Kalman Filters or Complementary filters and thus estimates the angles at which the drone is located.

5.4. stabilizer library

- `void Stabilizer_init()`
  - Sets all PIDs and filters needed to stabilize the quadcopter.

- `void Stabilize(float angles[2], float angular_velocities[3], int8_t control_inputs[4])`
  - Stabilizes the drone's movement using PID controllers.

5.5. motors library

- `void Motors_SetPWR(uint8_t thrust, int8_t yaw, int8_t pitch, int8_t roll)`
  - Using the Motor Mixing Algorithm sets the pwm of all four motors.

- `void Motors_Switch(uint8_t power_on)`
  - Turns the motors on and off.

5.6. MPU6050 library

- `MPU6050_STRUCT` - a structure tpye containing the I^2C address of MPU6050 and addresses of buffers for the data being read.

- `MPU6050_config` - structure type for MPU configuration

- `MPU6050_config MPU_get_default_cfg(void)` - returns the default configuration for MPU

- `HAL_StatusTypeDef MPU_init(MPU6050_STRUCT *mpu, MPU6050_config* cfg)` - initialises the MPU according to the config passed as argument.

- `HAL_StatusTypeDef MPU_measure_gyro_offset(MPU6050_STRUCT* mpu, uint16_t samples`
  - measures the constant offset of the gyroscope based on set sample number.

- `HAL_StatusTypeDef MPU_measure_acc_offset(MPU6050_STRUCT* mpu, uint16_t samples)`
  - measures the constant offset of the accelerometer.

- `HAL_StatusTypeDef MPU_clear_int(MPU6050_STRUCT *mpu)` - clears the interrup flag from MPU's interrupt status register.

- `HAL_StatusTypeDef MPU_read_acc_gyro_DMA(MPU6050_STRUCT *mpu)` - reads accelerometer and gyroscope data using DMA to a memory location passed using the config struct.

- `HAL_StatusTypeDef MPU_read_acc_gyro_DMA_complete(MPU6050_STRUCT *mpu)` - calculates the sensor values from raw bytes after the readout has been completed.

Note that the functions for measuring the offsets write the result to a static variable defined in the library source file. The functions for reading the measurements of the IMU substract the offsets automatically.