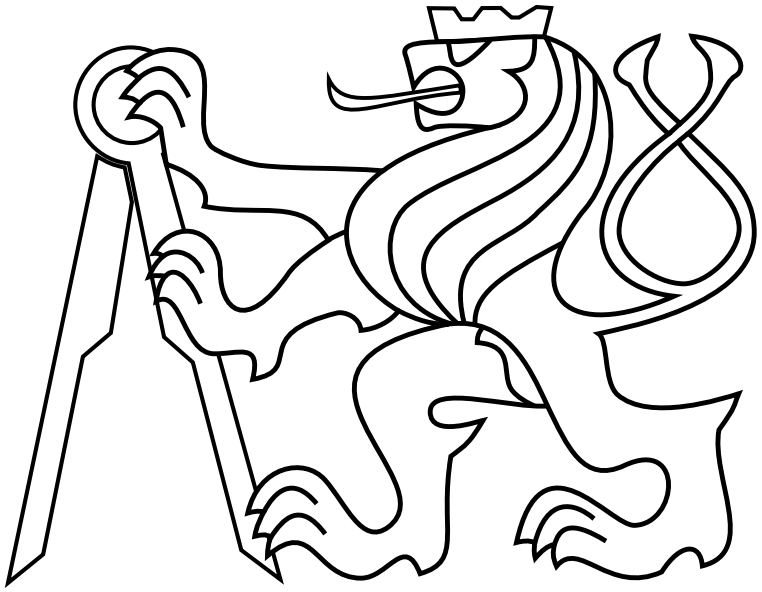
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Martin Klučka

User interface and failure detection for support of multi-MAV experiments

Department of Cybernetics

Thesis supervisor: **Dr. Martin Saska**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne..... .....

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Martin K l u č k a

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: User Interface and Failure Detection for Support of Multi-MAV Experiments

Guidelines:

The aim of the thesis is to design and implement a user interface that facilitates experiments with a swarm of simultaneously flying Micro Aerial Vehicles (MAVs). The interface should enable to commit plans of movements for particular MAVs, to save and subsequently analyse telemetric data, and to detect and visualize failures of MAV subsystems.
Student designs and implements:
1. tool for logging of telemetric data onboard of MAV using OpenLog module,
2. user interface that enables to send commands into MAVs, to visualize telemetric data, and to highlight detected failures,
3. mechanism for saving telemetric data, which are selected by operator, and its subsequent analysis,and
4. algorithm for automatic detection of failures based on analysis of telemetric data.

Bibliography/Sources:

[1] Heredia, G. - Ollero, A. - Bejar, M. - Mahtani, R.: Sensor and actuator fault detection in small autonomous helicopters, Mechatronics, Volume 18, Issue 2, March 2008.
[2] Saska, M. - Chudoba, J. - Přeučil, L. - Thomas, J. - Loianno, G. - et al.: Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In Proceedings of 2014 International Conference on Unmanned Aircraft Systems (ICUAS).
[3] Cork, L. - Walker, R.: Sensor Fault Detection for UAVs using a Nonlinear Dynamic Model and the IMM-UKF Algorithm, Information, Decision and Control - IDC '07, 2007.
[4] Sanchez, S. - Perhinschi, M. - Moncayo, H. - et al.: In-Flight Actuator Failure Detection and Identification for a Reduced Size UAV Using the Artificial Immune System Approach, AIAA Guidance, Navigation, and Control Conference, 2009.

Bachelor Project Supervisor: Ing. Martin Saska, Dr. rer. nat.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 23, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Martin K l u č k a

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Uživatelské rozhraní a automatická detekce poruch pro podporu experimentů s více helikoptérami

Pokyny pro vypracování:

Cílem práce je navrhnout a implementovat uživatelské rozhraní, které usnadní experimentální práci se skupinou současně letících helikoptér. Systém umožní jednoduché povelování skupiny i jednotlivých jedinců, ukládání a následnou analýzu telemetrických dat a rychlou detekci a vizualizaci poruch některého ze subsystémů letounů.
Student navrhne a implementuje:
1. nástroj pro rychlé logování telemetrických dat přímo na palubě letounu s využitím OpenLog modulu,
2. uživatelské rozhraní umožňující povelovat jednotlivé helikoptéry, zobrazovat online telemetrická data a vizualizovat detekované poruchy,
3. mechanismus pro ukládání vybraných telemetrických dat a jejich následnou analýzu a
4. algoritmus pro automatickou detekci poruch na základě analýzy telemetrických dat.

Seznam odborné literatury:

[1] Heredia, G. - Ollero, A. - Bejar, M. - Mahtani, R.: Sensor and actuator fault detection in small autonomous helicopters, Mechatronics, Volume 18, Issue 2, March 2008.
[2] Saska, M. - Chudoba, J. - Přeučil, L. - Thomas, J. - Loianno, G. - et al.: Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In Proceedings of 2014 International Conference on Unmanned Aircraft Systems (ICUAS).
[3] Cork, L. - Walker, R.: Sensor Fault Detection for UAVs using a Nonlinear Dynamic Model and the IMM-UKF Algorithm, Information, Decision and Control - IDC '07, 2007.
[4] Sanchez, S. - Perhinschi, M. - Moncayo, H. - et al.: In-Flight Actuator Failure Detection and Identification for a Reduced Size UAV Using the Artificial Immune System Approach, AIAA Guidance, Navigation, and Control Conference, 2009.

Vedoucí bakalářské práce: Ing. Martin Saska, Dr. rer. nat.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 23. 1. 2015

Acknowledgements

First of all, I would like to thank the supervisor of this thesis, Dr. Martin Saska for his guidance and help throughout this project. Furthermore I thank my family for their support during my studies. I would also like to thank Tomáš Báča for valuable comments and advices.

Abstract

The aim of the thesis is to design and implement a graphical user interface (GUI). The GUI facilitates experiments with a swarm of simultaneously flying Micro Aerial Vehicles (MAVs). The GUI is able to send commands to one or more MAVs at same time. The GUI can log data from MAVs to the file. The main goal of this thesis is a failure detection of controllers. The GUI is able to detect and visualize failures of the MAV controllers. The failure detection was simulated and experiments was designed to verify functionality of detecting errors.

Keywords: micro aerial vehicles, graphical user interface, failure detection

Abstrakt

Cílem práce je navrhnout a implementovat grafické uživatelské rozhraní, které usnadní experimentální práci se skupinou současně letících helikoptér. Systém umožní jednoduché povelování skupiny i jednotlivých jedinců a ukládání telemetrických dat do souboru. Hlavním cílem práce je detekce chyb, které budou způsobené regulátory helikoptér. Grafické uživatelské rozhraní bude umět odhalit chyby a následně je zobrazit. Detekce chyb byla odsimulována a naslědně byly provedeny experimenty pro ověření funkčnosti detekce chyb.

Klíčová slova: malé bezpilotní letouny, grafické uživatelské prostředí, detekce chyb

Contents

List of Figures

1 Introduction

An unmanned aerial vehicle (UAV), also known as a drone, is a flying vehicle without a human pilot aboard. UAVs are becaming popular as a hobby, but also have large usage in the industry. For example, firefighters can explore the area to find people with a thermal camera in inaccessible areas [16, 15]. Futhermore UAV can be used in the military industry mainly as an scout or a stock supplier [5].

A micro aerial vehicle (MAV) is a class of UAV that has a limited size. MAVs can be used like swarm due to their smaller size. It is inspired by swarm of insects. Swarm of MAVs can explore area faster and safer.

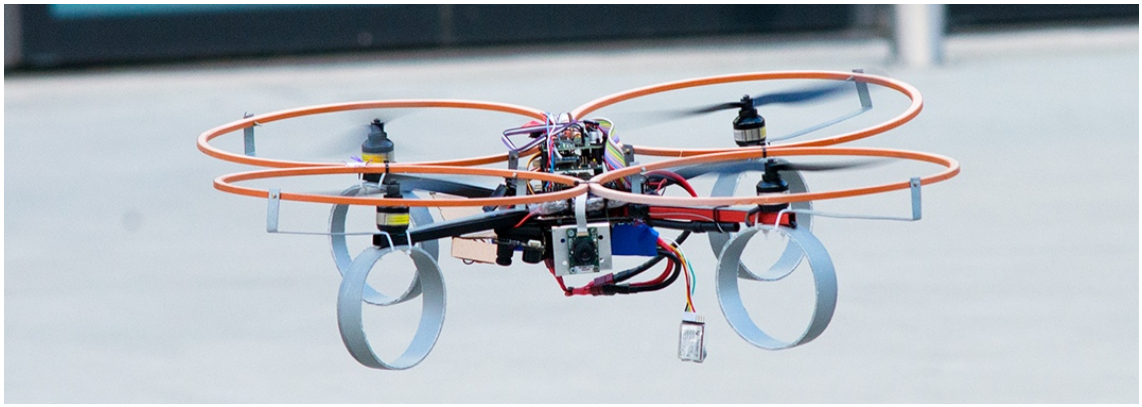


Figure 1: MAV demonstration

The main goal of this thesis is a design and implementation of a graphical user interface (GUI) for support experiments with swarms. GUI will allow sending commands to one or a whole group of MAVs, showing states or drawing telemetry data in one, two or four graphs. The next goal of this thesis is logging the telemetry data onboard of MAV using an OpenLog. The OpenLog fulfills the function of as a black box system. It is a device that can be viewed in terms of its inputs and outputs.

The final goal of this thesis is an automatic failure detection based on analysis of telemetric data. Failure detection will be implemented by modeled system of MAV. The modeled system of MAV will run parallely with the real MAV. The implemented failure detection will compare the differences between the real MAV and the modeled system and then analyze the data for error detection.

2 Related work

There is a lot of research about automatic failure detection of the UAV [4, 14, 19, 10, 11, 12]. It is very important for safe deployment of the systems. Damage can be prevented by checking the sensors. There is a reseach of fault detection using the model and the IMM-UKF algorithm [4]. Fault detection and identification algoritms may rely on knowlege of underlying system dynamics while some eschew this modeling in favor of data-driven anomaly detection. It considers model-based residual generation and data-driven anomaly detection for a small, low-cost unmanned aerial vehicle.

The research presented in [18] deals with negative selection algorithm (NSA). It solves a simple failure detection of an actuator of a helicopter swashplate. A simple model for the actuator failure is developed and coupled with a helicopter flight dynamics model, and some trajectories for the algorithm training and simulated validations. The performance of the algorithm is slightly improved by observing the covariance states of the flight dynamic, instead of the variables themselves.

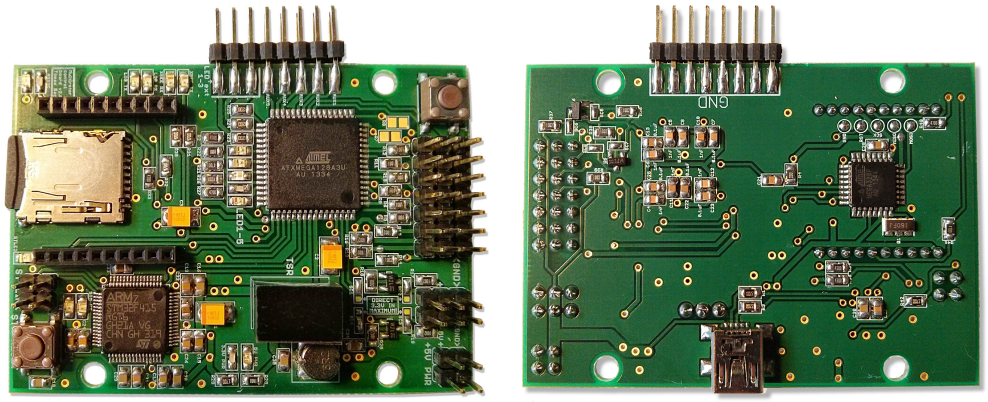
Swarm of MAVs is now popular because it is possible to reduce the size of UAV and also MAV swarm could be better for a deployment. In the research [20] is a section about complex system for swarm control of Micro Aerial Vehicles stabilized by an onboard visual relative localization. The main purpose is to verify the possibility of self-stabilization of multi MAV groups without an external global positioning system. Such an approach enables the deployment of MAV swarms outside laboratory conditions and it may be considered an enabling technique for fleet utilization of MAVs in real-world scenarios.

3 Hardware

This thesis is based on an MAV made by Multi-robot Systems Group (MRS Group) from Czech Technical University in Prague (CTU).

3.1 Control board

The main unit in the MAV is Control board v.2 (see figure 2) made by Tomáš Báča [2]. The control board contains xMega, ARM, socket for XBee, OpenLog and micro SD card socket. External modules and sensors can be attached to the control board via UART or I2c.



(a) Top of the control board (b) Bottom of the control board

Figure 2: Control board v.2

3.1.1 ATxMega

ATxMega 128a3u is an 8-bit AVR microcontroller. ATxMega is mainly used for a communication, controller or trajectory calculations. The microcontroller featuring 128KB self-programming flash memory, 8KB SRAM, 2048-Byte EEPROM, 4-channel DMA controller and 8-channel event system [1].

3.1.2 ARM

STM32F415RGT6 is a 32-bit ARM device used for calculating MPC regulator and Kalman filter. This microcontroller only calculates and sends out data further to microcontroller ATxMega [28].

3.1.3 Open Log

The Sparkfun OpenLog (referred as OpenLog) is an open source data logger, which communicates via a serial port (see figure 3). The OpenLog communicates with the microcontroller ATxMega. The OpenLog is integrated into the Control board.

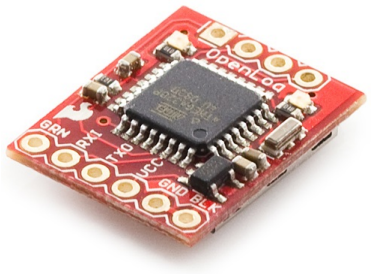


Figure 3: SparkFun OpenLog [27]

3.2 Modules

Modules are connected to the control board via UART or I2c. There are two modules connected at this moment, PX4Flow and a Camera module.

3.2.1 PX4Flow

The MAV is equiped with Pixhawk Px4Flow Smart Camera module with Maxbotix HRLV-EZ4 (see figure 4). Module is used for measure altitude from 0.3m to 4m. Furthermore, it is used for measuring vertical velocity regardless of the altitude. The module includes 3-axis gyroscope.

Figure 4: PX4Flow module

3.2.2 Camera module

The camera module contains the *Caspa*™ camera and the *Overvo*® computer made by Gumstix [13]. The MAV is equiped with the camera module (see figure 5). The camera module is used for relative visual location of the another MAV by equipped circular pattern. Circular pattern is shown in figure 6.

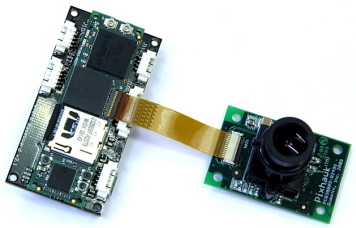


Figure 5: Gumstix camera module

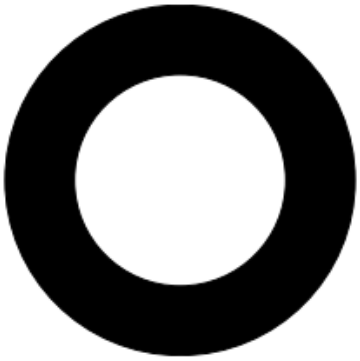


Figure 6: Example of circular pattern

3.3 Controllers

The experimental platform uses two types of controllers, PID and MPC. The PID controller is used to control the altitude using relative ground distance. The ground distance is got by an ultrasonic range finder. The MPC controller is used for position control. PID controller is implemented on the xMega and MPC is implemented on the ARM.

3.4 XBee

XBee Pro S2B (see figure 7) is a ZigBee module made by Digi. The main purpose of the XBee is to procure communication between MAVs and a ground station [6].

Figure 7: XBee Pro 2SB

3.4.1 Communication via XBee

This thesis was parallely developed with thesis of my colleague [9] whose goal is to create communication via XBee. The most important commands for this thesis obtain data through API, such as telemetric data, status of MAV etc.

Commands for telemetry are "getTelemetry (unsigned char kopter, unsigned char type)", where "kopter" is an unsigned char used to select the address of MAV and "type" is an unsigned char belonging to specific telemetry. Return type of command is float. The name of each varable begins with "TELEMETRIES" so the full name is for example "TELEMETRIES.GROUND_DISTANCE_ESTIMATED" (see table 1 and table 2).

Type	Description
ALTITUDE_ESTIMATED	Estimated distance from ground
ALTITUDE	Distance from ground
ELEVATOR_SPEED	Speed of evelator
AILERON_SPEED	Speed of aileron
ELEVATOR_SPEED_ESTIMATED	Estimated speed of evelator
AILERON_SPEED_ESTIMATED	Estimated speed of aileron
ELEVATOR_POSITION	Position of elevator
AILERON_POSITION	Position of aileron
ALTITUDE_CONTROLLER_OUTPUT	Output of altitude controller

Table 1: API commands for telemetry

Commands for states have the following form "getStatus (unsigned char kopter, unsigned char type)", where "kopter" is an unsigned char used to select the address of MAV and "type" is an unsigned char belonging to specific status. Return type of command is unsigned char. The name of each varable begins with "COMMANDS" so the example of a full name is "COMMANDS.CONTROLLERS" (see table 3).

Type	Description
ALTITUDE_SPEED	Speed of altitude
AILERON_CONTROLLER_OUTPUT	Output of aileron controller
ELEVATOR_CONTROLLER_OUTPUT	Output of elevator controller
ALTITUDE_SETPOINT	Setpoint of altituder
ELEVATOR_POS_SETPOINT	Setpoint of elevator position
AILERON_POS_SETPOINT	Setpoint of aileron position
ELEVATOR_ACC	Acceleration of elevator
AILERON_ACC	Acceleration of aileron
VALID_GUMSTIX	If MAV see the blob
OUTPUT_THROTTLE	Output of throttle
OUTPUT_ELEVATOR	Output of elevator
OUTPUT_AILERON	Output of aileron
OUTPUT_RUDDER	Output of rudder
BLOB_ELEVATOR	Elevator distance from blob
BLOB_AILERON	Aileron distance from blob
BLOB_ALTITUDE	Vertical distance from blob
PITCH_ANGLE	Angle of Pitch
ROLL_ANGLE	Angle of roll
ELEVATOR_ACC_ERROR	Acceleration error of elevator
ELEVATOR_ACC_INPUT	Acceleration input of elevator
AILERON_ACC_ERROR	Acceleration error of aileron
AILERON_ACC_INPUT	Acceleration input of aileron

Table 2: API commands for telemetry

The MAVs can be controlled by commands from API. There are different commands for turning the controller on or off, landing etc. Command are "setController (unsigned char kopter,unsigned char option)", where "kopter" is an unsigned char used to select the address of MAV and "option" is an unsigned char of specific command (see table 4). Commands for landing are "land (unsigned char kopter,unsigned char on)", where "kopter" is an unsigned char used to select the address of MAV and "on" is an unsigned char of specific command for turning landing on (ONOFF.ON) or off (ONOFF.OFF).

Command "setSetpoint (unsigned char kopter,unsigned char type,unsigned char incType,float value)" is for adjusting setpoints. There are four variables. The first one is an unsigned char "kopter", which serves to select the address of MAV, unsigned char "type" serves to select a specific setpoint, another variable is an unsigned char "incType", which serves to select a relative or absolute distance and the last variable is float "value", which serves to adjusting the setpoint value. The form of variable "incType" is either "POSITIONS.RELATIV" or "POSITIONS.ABSOLUT" (see table 5).

The last two commands turn on or off gumstix or following preset trajectory points.

Type	Description
CONTROLLERS	States of controller which one is on
LANDING	MAV states of landing
GUMSTIX	On/Off states of gumstix
TRAJECTORY_FOLLOW	On/Off states of trajectory follow

Table 3: API commands for states

Option	Description
CONTROLLERS.OFF	Turn off both controllers
CONTROLLERS.VELOCITY	Velocity controller is turned on
CONTROLLERS.POSITION	Position controller is turned on
CONTROLLERS.BOTH	Both controllers are turned on

Table 4: API commands for controllers

Type	Description
SETPOINTS.THROTTLE_SP	Setpoint of throttle
SETPOINTS.ELEVATOR_POSITION	Setpoint of elevator position
SETPOINTS.AILERON_POSITION	Setpoint of aileron position
SETPOINTS.ELEVATOR_VELOCITY	Setpoint of elevator velocity
SETPOINTS.AILERON_VELOCITY	Setpoint of aileron velocity

Table 5: API commands for the setpoints

Gumstix shows the visibility of the circular pattern on the another MAV. Those commands are “trajectoryFollow (unsigned char kopter,unsigned char on)” and “setGumstix (unsigned char kopter,unsigned char on)”. An unsigned char “kopter” used to select the address of MAV and “on” is an unsigned char of specific command for turning landing on (ONOFF.ON) or off (ONOFF.OFF). Trajectory following has one additional command used for settings trajectory points, which it is “trajectoryAddPoint (unsigned char kopter,unsigned char index,float time,float elevatorPos,float aileronPos,float throttlePos)”, where “index” serves as a pointer to array of trajectory points. This array has 10 points. Variables “elevatorPos”, “aileronPos” and “throttlePos” serve to set all possible setpoints.

4 Logging

Data from MAVs may be logged in two ways. The first way is onboard logging via SparkFun OpenLog and the second way is logging using the GUI. Onboard logging data is written into an SD card using the OpenLog. Data can be used for the determination of errors or for later analysis.

4.1 Onboard logging via SparkFun OpenLog

The Sparkfun OpenLog [27] is an open source data logger, which operates through a serial port. The OpenLog communicates with the microcontroller ATxMega. The OpenLog currently supports FAT16 and FAT32 formatting and the size of microSD card up to 64 GB. Basic baud rate is 9200 bps but it can be increased up to 115200 bps (see figure 3).

4.1.1 Commands

Upon powering up into the default 'NewLog' mode the OpenLog will write to output three characters "12 < ". Characters from input will be recorded in the LOG#### .TXT file, where #### is the file serial number. Upon pressing Ctrl+z three consecutive times (the different ASCII character can be set in the config.txt file), the OpenLog will exit the record mode and enter the command mode. Following commands can be entered [26].

4.1.2 Basic commands

• **new file.txt** - Create a new file in the current directory. The supported file name length is up to eight characters plus extension length is up to three characters (e.g. LOG12345.TXT).

• **append file.txt** - Append the text to the end of file.txt. Serial data is then read from UART in a stream.

• **write file.txt 12** - Write the text to file.txt starting from OFFSET, where OFFSET is an input parametr of this command. Exiting this state is performed by sending an empty line.

• **rm file.txt** - Delete file.txt from the current directory.

• **rm -rf myDirectory** - Remove myDirectory, all sub-directories and all files within these directories.

• **size file.txt** - Display the size of file.txt in bytes.

4.1.3 System settings

System configuration is stored on SD card in config.txt.

• **echo on/off** -Echo can be allowed by this command. The OpenLog does not repeat text in the command line, when echo is turned off.

• **verbose on/off** - This command turns error reporting on and off. The OpenLog responds only "!", if verbose is turned off .

• **baud** - Brings up a system menu to enter a baud rate. Basic baud rate is 9200 bps. The OpenLog supports any baud rate between 300 bps and 1 Mbps.

• **boot up mode** - The OpenLog has 3 basic modes. The first mode is "New file logging", which creates a new file (LOG####.txt) each time the OpenLog powers up and starts recording data to the file. The second mode is "Append File Logging". This mode creates a file called "SEQLOG.txt" and appends any recieved data to the file. The last mode is "Command prompt" that uses the command line for creating files or write to them.

• **escape character** - Stop recording is posible by the characters, which are contained in config.txt.

• **number of escape characters** - This command sets the number of characters that have to be pressed.

4.1.4 Implementation

There are two implementation options. Logging and reading data or only logging data to SD card. The second implementation is used for faster data logging.

The first implementation

Config for the OpenLog was adjusted as requied. Echo has been turned off since it is not needed in the project. Baud rate was set to 115200 bps to be able to log the data. Boot up mode has been set to "Command prompt" so that it can be read from microSD card. Escape character has been set to "$$$".

Separated thread was created for logging so that logging time is constant and independent on the rest of the code (logging speed set to 30logs per second). The OpenLog starts recording data immedietly after turning Control Board on. Default file name is DATA-

LOG.TXT. File name can be changed through communication with XBee.

The second implementation

The difference between the implementations is mainly in the settings of Boot up mode. Boot up mode has been set to "New File Logging". This implementation enables only logging data to the file, which is generated automaticaly to the SD card.

4.2 Telemetry logging using the GUI

Telemetry can be logged by GUI. Logging telemetry data has the option to change the filename or switch it on or off (see figure 8). Logging can be switched by using the checkbox. The first record on the line is always a time from ground station. Telemetry data are logged in legible form due to the possibility of its re-ploting and analyzing. Each *MAV window* has its own log file and telemetry data are logged independently of the other *MAV windows*. More about implementation of logging telemetry is in the subsection 6.8.

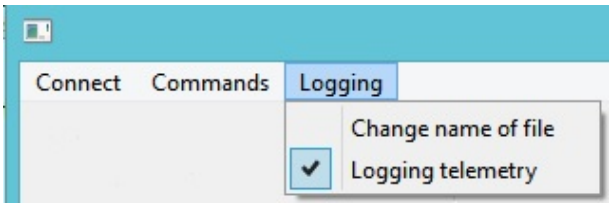


Figure 8: Logging options in GUI

5 Failure detection

The main goal of the failure detection is to detect errors caused by measurement or controllers. Errors are checked when regulators are enabled. Failure detection was solved using a modeled system of MAV, which runs independently on the GUI. The modeled system of MAV first initializes all the necessary variables for futher calculations. Failure detection waits until the regulator is turned on. Identification of the MAV model was not a part of this thesis. The simulations (see subsection 5.2 and 5.5) were created by using the C++ code with GUI, which was created in this thesis.

5.1 Model of MAV

Discrete formulation of the dynamical system is used in this thesis. State space formulating is written in a discrete form with a constant sampling rate $1/\Delta$t.

5.1.1 Altitude model

Altitude model was identified in [8]. The following form describes a discrete time-invariant system with a main matrix $\mathbf{A}$ and input matrix $\mathbf{B}$

$$q_{[t+1]} = \mathbf{A}\mathbf{q}_{[t]} + \mathbf{B}\mathbf{u}_{[t]}. \tag{1}$$

There is a state vector $\mathbf{q}_z = (z, \dot{z}, \ddot{z}, \ddot{z_u})^T$, where z represents altitude position, $\dot{z}$ represents altitude speed, $\ddot{z}$ represents altitude acceleration and $\ddot{z}_u$ represents altittude acceleration with gravitation acceleration. An input is $\mathbf{u}_z = (U_D, 1)^T$, where U_D represents collective thrust of a propeller. Altitude dynamics model includes Earth's gravitation pull so there is a need for an additional input to be added to the system. The second input value is always equal to 1 due to gravitation pull. Altitude LTI system for the state vector $\mathbf{q}_z$ and the input $\mathbf{u}_z$ is defined as

$$\mathbf{A}_z = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & \Delta t & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & P_1 \end{bmatrix}, \mathbf{B}_z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & -g \\ P_2 & 0 \end{bmatrix}, \tag{2}$$

where $g \approx 9.8ms^{-2}$ is a gravity acceleration, Δt is the sampling period and P_1 and P_2 are the parameters of the first order system $U_D \rightarrow \ddot{z}$.

Parameters for the altitude system are used as

$$g = 9.8, \Delta t = 0.0114, P_1 = 0.9658, P_2 = 0.0023. \tag{3}$$

Matrix $\mathbf{A}_z$ and Matrix $\mathbf{B}_z$ are used as

$$\mathbf{A}_z = \begin{bmatrix} 1 & 0.0114 & 0 & 0 \\ 0 & 1 & 0.0114 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.9658 \end{bmatrix}, \mathbf{B}_z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & -9.8 \\ 0.0023 & 0 \end{bmatrix}. \tag{4}$$

These matrices are used for state calculations of the altitude modeled system.

5.1.2 Attitude model

Attitude model was identified in [2]. The following form describes a discrete time-invariant system with a main matrix $\mathbf{A}$ and input matrix $\mathbf{B}$.

$$q_{[t+1]} = \mathbf{A}\mathbf{q}_{[t]} + \mathbf{B}\mathbf{u}_{[t]}. \tag{5}$$

There are state vectors $\mathbf{q}_x = (x, \dot{x}, \ddot{x}, \ddot{x_u}, \ddot{x_d})^T$ and $\mathbf{q}_y = (y, \dot{y}, \ddot{y}, \ddot{y_u}, \ddot{y_d})^T$, where x, y represent attitude position, $\dot{x}, \dot{y}$ represent attitude speed, $\ddot{x}, \ddot{y}$ represents attitude acceleration, $\ddot{x}_u, \ddot{y}_u$ represents attitude contribution in acceleration from the control input and $\ddot{x}_d, \ddot{y}_d$ represent attitude disturbance in acceleration. An input is $\mathbf{u}_x = \psi_D$ and $\mathbf{u}_y = \theta_D$, where ψ_D represent desired roll angle of the MAV and θ_D represent desired pitch angle of the MAV. Attitude LTI system for the state vectors $\mathbf{q}_x, \mathbf{q}_y$ and the inputs $\mathbf{u}_x, \mathbf{u}_y$ is defined as

$$\mathbf{A}_{x,y} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & P_3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{B}_{x,y} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ P_4 \\ 0 \end{bmatrix}, \tag{6}$$

where Δt is the sampling period and P_3 and P_4 are parameters of the first order transfer from a desired angle to the actual angle of attitude.

Parameters for the attitude system are used as

$$\Delta t = 0.0114, P_3 = 0.9796, P_4 = 5.13 \times 10^{-5}. \tag{7}$$

Matrix $\mathbf{A}_{x,y}$ and Matrix $\mathbf{B}_{x,y}$ are used as

$$\mathbf{A}_{x,y} = \begin{bmatrix} 1 & 0.0114t & 0 & 0 & 0 \\ 0 & 1 & 0.0114 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0.9796 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{B}_{x,y} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 5.13 \times 10^{-5} \\ 0 \end{bmatrix}. \tag{8}$$

These matrices are used for state calculations of the attitude modeled system.

5.2 Simulation of a real MAV

A *Simulation of real MAV* is performed under C++ code with GUI. A model of the real MAV (see section 5.1) is used for the simulation. The modeled system of MAV is complemented by controllers, which are implemented on the real MAV. The altitude modeled system contains PID controller and the attitude modeled system cointains MPC. The modeled system is simulated in an ideal world conditions and therefore it is necessary to distort the output signal (see in section 5.2.1).

5.2.1 Signal noise

The MAV sensors emit a noise, which needs to be simulated. The assumption is that the noise is drawn from a normal distribution $\mathcal{N}(\mu, \sigma)$. Deviation of sensors was calculated by

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[x_i - p_i]}, \tag{9}$$

where n is the number of data values, the $\vec{x}$ is the vector of position values and the $\vec{p}$ is the vector of polynomial values fitted on the $\vec{x}$. Normal distribution for altitude position is $\mathcal{N}(0, 0.0153^2)$ and for attitude speed is $\mathcal{N}(0, 0.08)$. Values from the Normal distribution are added to true values of altitudie position and attitude speed from the modeled system simulation.

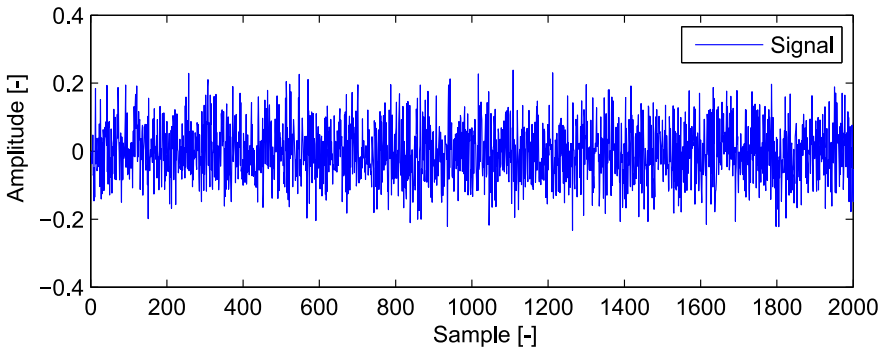


Figure 9: Example of a noise for Normal distribution N(0,0.08)

5.2.2 Examples of MAV simulation

The setpoint of the altitude modeled system was set to *1* meter from the initial value. Initial value *0* is the position where the system was located at startup. Noise signal is not seen because it is very small (see fig. 10).

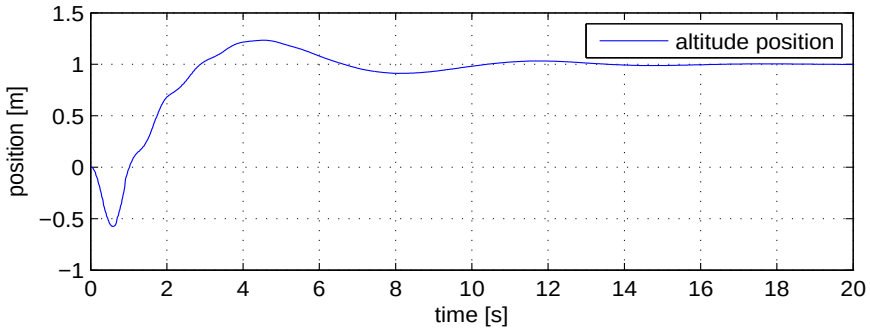
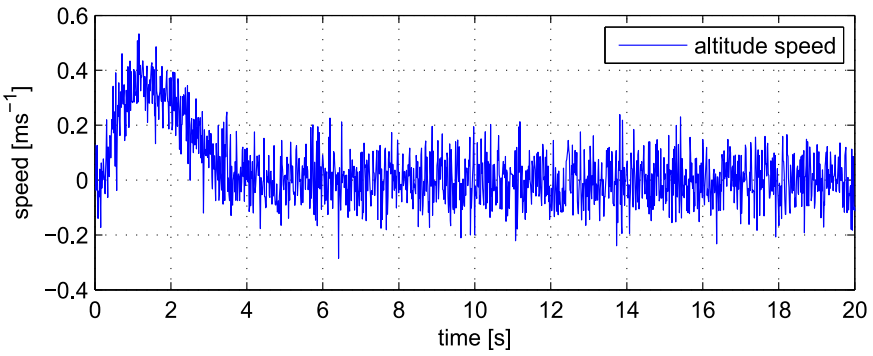
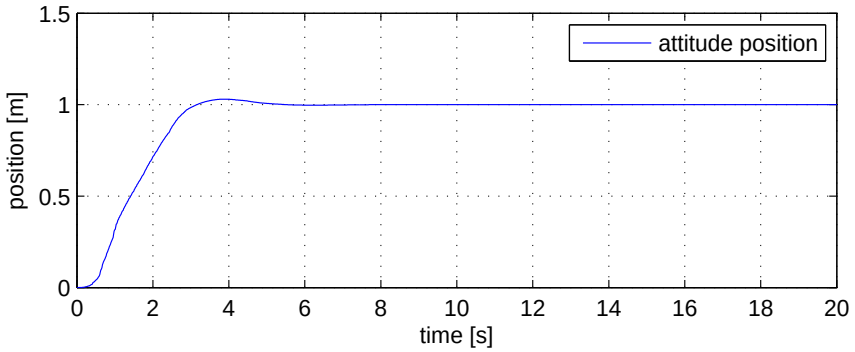


Figure 10: Example of the altitude modeled system

Setpoint of the attitude modeled system was set to 1 meter from the initial value. Noise signal is visible on the attitude speed (see fig. 11a).



(a) Attitude speed



(b) Attitude position

Figure 11: Example of the attitude modeled system

5.3 Failure detection system

An MAV model for *Failure detection system* is the same as for the simulations in section 5.2. The modeled system used for the failure detection does not contain any controllers. The modeled system is used to estimate the behavior of a real MAV. Therefore, the modeled system receives all states from a real MAV via XBee communication.

5.3.1 Implementation of Failure detection

Failure detection is implemented as a thread in C++ code with GUI. The *Failure detection thread* begins with inicialization variables and matrices, which are used for calculating states of modeled system. Each real MAV is running his own *Failure detection thread* due to the speed of detection. The *Failure detection thread* is checking the states of the controllers after initialization of all variables. The *Failure detection systems* starts when the MAV controllers are turned on. The PID controller and the MPC are checked separately because altitude and attitude *Failure detection systems* are implemented separately.

When controllers are turned on, the *Failure detection systems* receives initial states for system from a real MAV and the *Failure detection systems* starts to predict behavior of a real MAV. The *Failure detection systems* obtains only control action and states of the *Failure detection systems* calculated by altitude and attitude systems. The *Failure detection systems* receives all states from a real MAV every two seconds in order to level states. When regulators are turned off, the *Failure detection thread* is turned off too. The *Failure detection systems* will wait until the controller is turned on. For the graphical display of errors of MAV see subsection 6.9.

5.4 Data analysis from failure detection modeled system

Data is analyzed in each cycle of the *failure detection thread*. Data is stored in an array. The array has a size of three hundred entries. The array moves dynamically and always contains the last three hundred entries. The array contains the absolute value of the difference between the position of real MAV and the *Failure detection systems*. Sum of all data, peak to peak and median from the array were chosen for data analysis of the array (for result see figure 12 and figure 13).

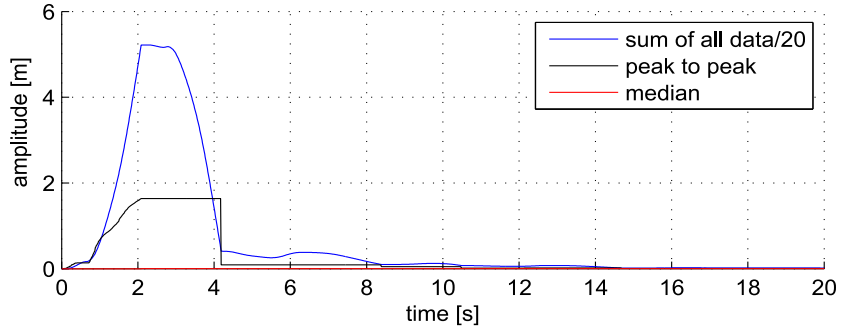


Figure 12: Example of errors in simulation of altitude system with artificial error

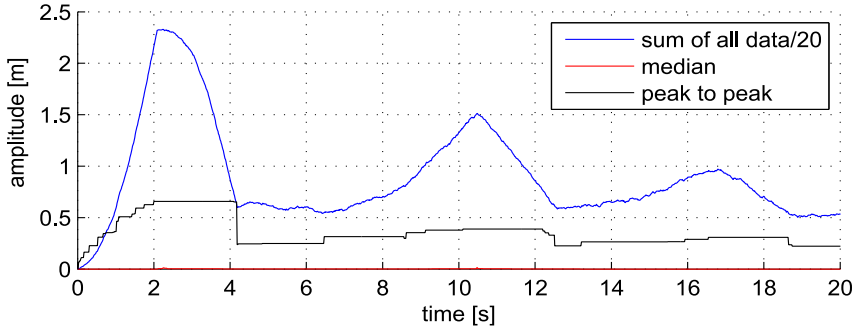


Figure 13: Example of errors in simulation of attitude system with artificial error

Artificial error was included in figure 12 and figure 13 in first four second. From figures is possible to establish a threshold, which was experimentally determined to 70% of the maximum value. Median seen in the figures is not suitable for this analysis of errors. The sum of all the data reveals errors but it is not very reliable as seen on figure 13. Peak to peak was chosen as the best option for this case. Peak to peak was able to detect the error in this simulation reliably.

Figure 14 and figure 15 were created without artificial error. It is possible to see that values of the peak to peak and the sum of all data are smaller. The failure detection did not found the error with the selected threshold.

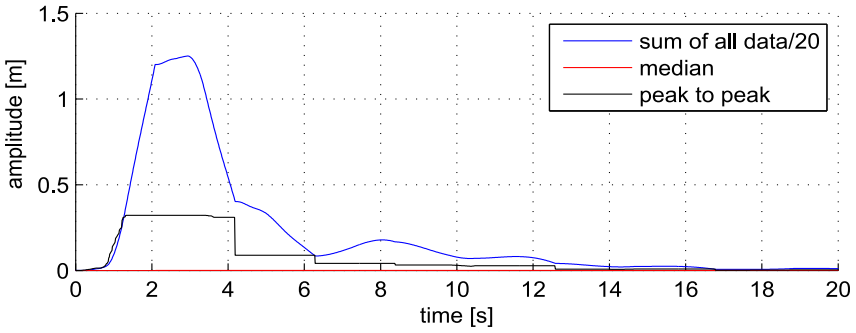


Figure 14: Example of errors in simulation of altitude system

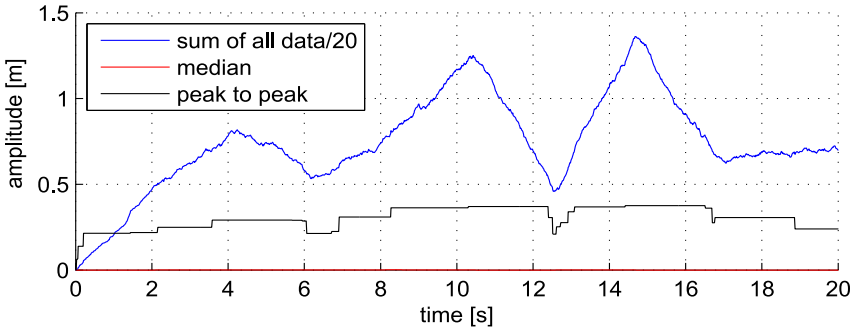


Figure 15: Example of errors in simulation of attitude system

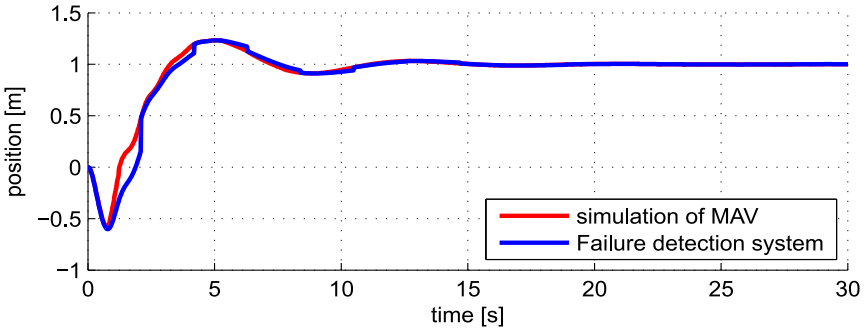
5.5 Simulations

Simulations were performed under C++ code with GUI. Simulation system of MAV and the *Failure detection systems* ran on their own threads. All data from systems was logged during simulations using the C++ code with GUI. Data was retrospectively analyzed and results were recorded. Simulations were performed with previous systems.

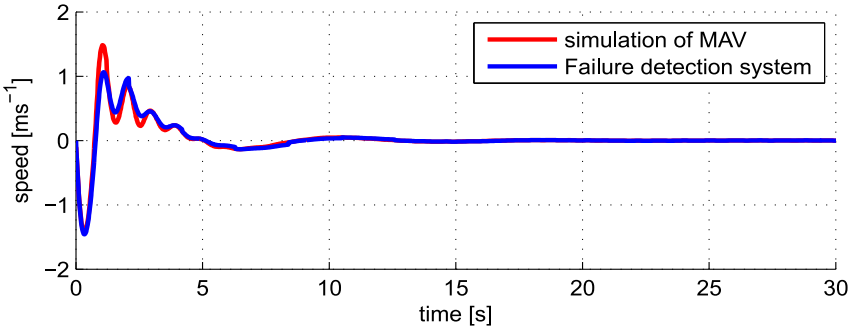
5.5.1 Systems of MAV simulation and failure detection

The first experiment based on the control action was designed to predict a behavior of MAV simulation through *Failure detection systems* without regulators.

In the figures (figure 16 and figure 17) is an example of prediction of the *Failure detection systems* behavior in MAV simulation. In figure 17b is an example of inaccuracy in comparison with the simulation model of real MAV. Behavior prediction of the MAV simulation from the *Failure detection systems* is shown in the figures 16 and 17. Inaccuracy in the comparison states of MAV simulation and the *Failure detection system* is shown in figure 17b.

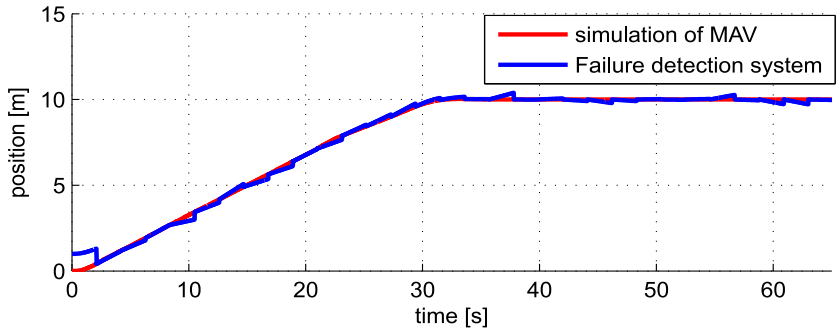


(a) Altitude position

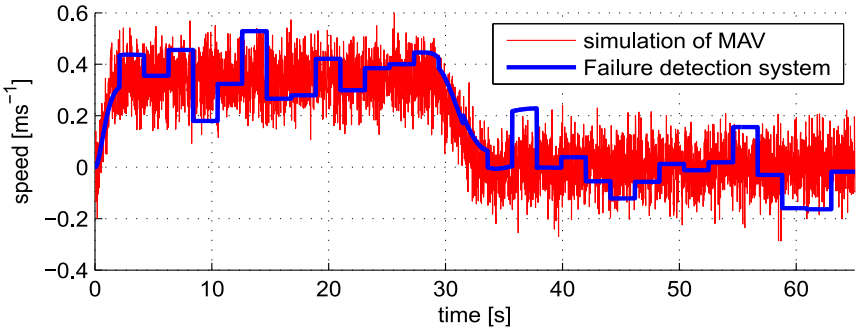


(b) Altitude speed

Figure 16: Altitude system



(a) Attitude position



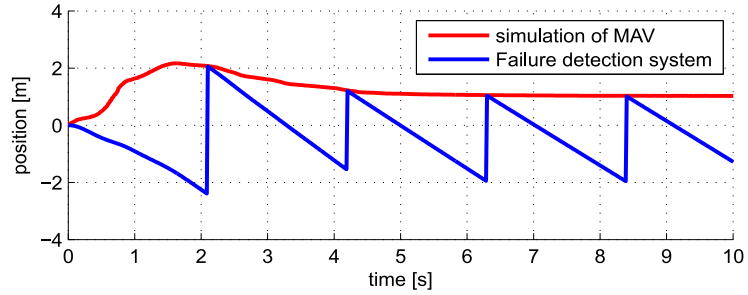
(b) Attitude speed

Figure 17: Attitude system

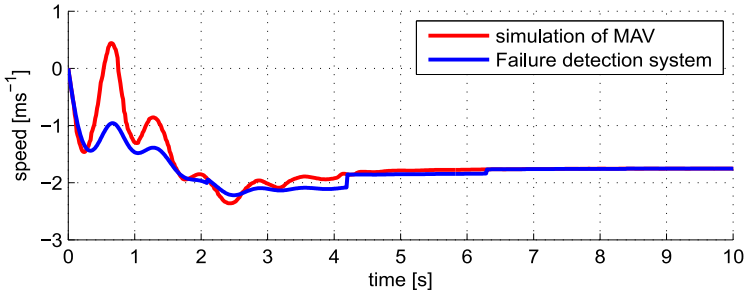
5.5.2 Artificially created error

The second simulation was designed to verify the functionality of the Failure detection. Artificial error was created by adding 0.02 m to the measured positions of both systems. Value of error was experimentally chosen to cause explicit errors to both systems.

The differences between positions of the *Failure detection systems* and MAV simulation is seen in the figure 18 and the figure 19. The *Failure detection systems* therefore behaves as desired in this thesis.

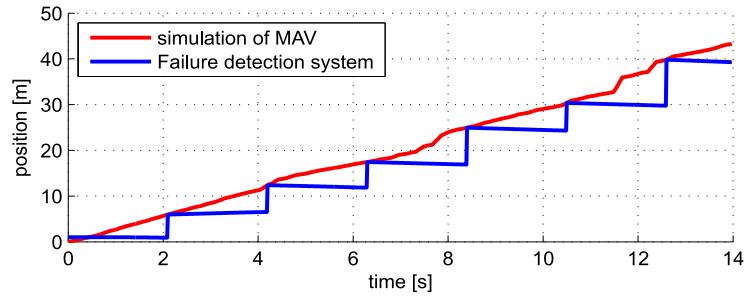


(a) Altitude position

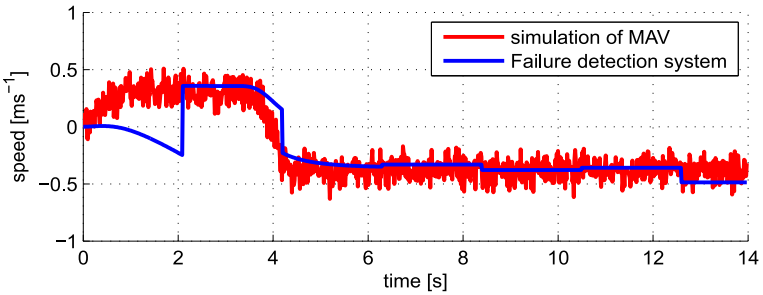


(b) Altitude speed

Figure 18: Altitude system with error



(a) Attitude position



(b) Attitude speed

Figure 19: Attitude system with error

6 Graphical user interface

This section explains graphical user interface (GUI), the GUI elements and functionality. The GUI is developed under the QT framework. The main goal of the GUI is facilitates experiments with a swarm of simultaneously flying MAVs. The GUI is used to control the MAVs and display their telemetry data in a realtime. The Realtime data is obtained by communication via XBee. The interface could send commands for a particular MAVs or entire group of MAVs. The interface saves and subsequently analyses telemetric data and detect failures of MAV subsystems. Detected errors are shown in the interface.

6.1 QT framework

QT is a cross-platform application and UI framework for developers using C++ or QML. QT and the supporting tools are developed as an open source project. QT can be used under open source license (GPL v3 and LGPL v2.1). QT Creator, which supports the cross-platform QT IDE, is used for the development in QT [3]. Real-time elements were developed in QT Framework 4.8 and therefore it is necessary to have version 4.8 or newer.

6.2 MAVs GUI window

The main aim of the *MAVs GUI window* (see figure 20) is to start communication via XBee that runs in own thread and chooses how many graphs in the next window are required.

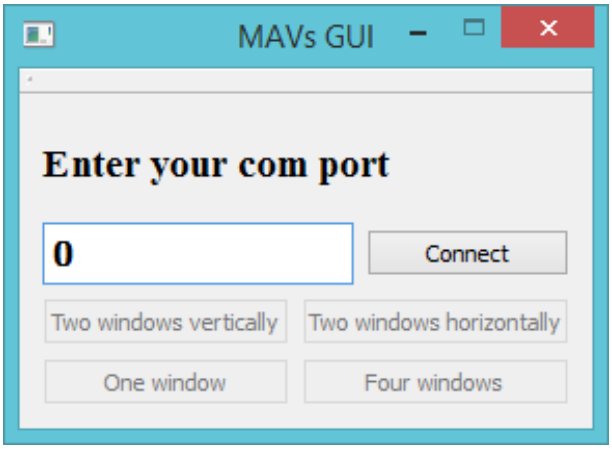


Figure 20: MAVs GUI window

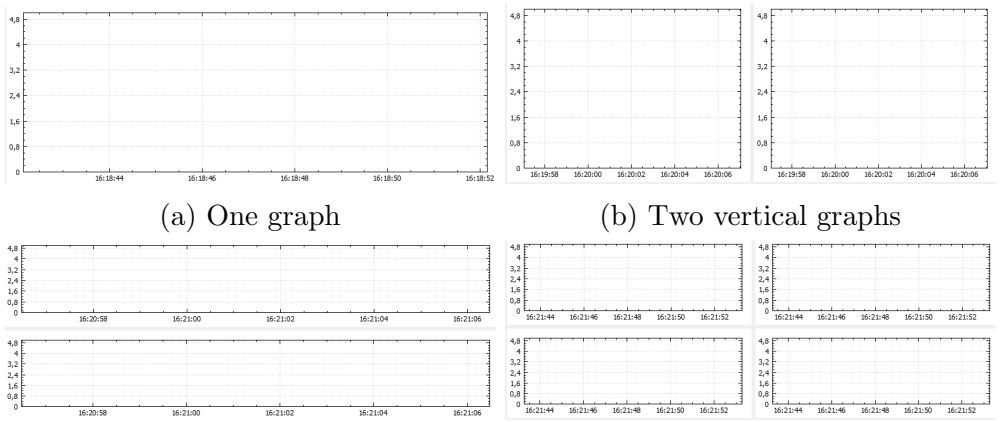
6.2.1 Communication thread

The communication thread is used to receive packets from XBee. Before running the thread, it is necessary to write the COM port number, where XBee is connected and click Connect button. Each ground station can connect XBee to another COM port. After entering the COM port GUI starts a new communication thread, which communicates with API. Communication thread only receives the data for API, which handles the data for future use.

6.2.2 The possibility of setting a number of graphs

After the startup of the communication thread the buttons for setting a number of graphs become visible. The *MAVs GUI window* may now open new separated windows with the specific *MAV window*.

The *MAVs GUI window* allows to choose the number of graphs in the *MAV window*. The other settings are not dependent on the number of graphs. Four options can be selected from the *MAVs GUI window*: one graph is outstretched over the entire area of graphs, two graphs are arranged either horizontally or vertically and four graphs are arranged in a square (see figure 21).



(a) One graph (b) Two vertical graphs

(c) Two horizontal graphs (d) Four graphs

Figure 21: Showing location of graphs

6.3 QCustomPlot

QCustomPlot is a library written in C++ programming language [17]. It is a *QWidget*, which is used for plotting and data visualization. This library focuses at plotting graphs and charts. The QCustomPlot offers high performance for real-time visualization applications. Source code and Software are licensed under the GNU General Public License.

6.4 Graphs

The library QCustomPlot is used for plotting graphs. The QCustomPlot is used for co-operation with QT framework in QT Creator. The QCustomPlot can be linked to *QWidget*. *QWidget* is used for working with the graphical elements. Real-time graphs are supported by the QCustomPlot, which is needed in *MAVs GUI window*.

Telemetry data is taken from the API and plotted to the graphs. Graphs are refreshed with the highest priority rate dependent on a workload of the ground station. Telemetry data is selected in a scroll area where are the *QCheckBoxes* (see subsection 6.4.1). Telemetry data is plotted only if a specific *QCheckBox* is checked. A legend was added for better orientation in graphs.

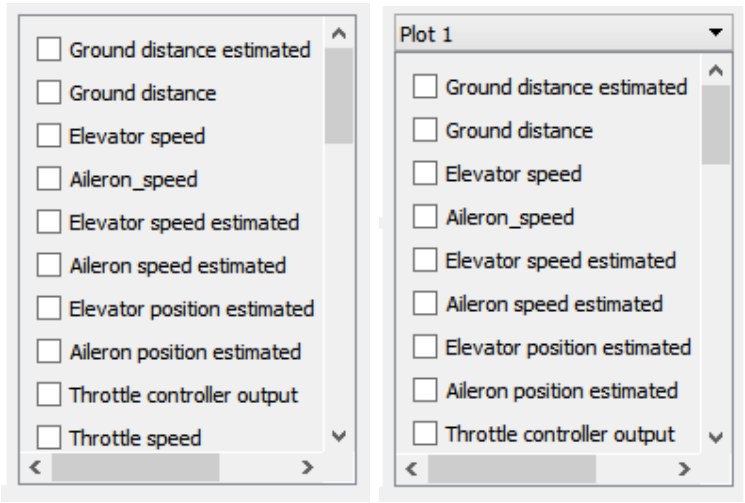
Range of Y-axis depends on the maximal and minimal values of the telemetry data that is curently plotted. The X-axis contains enrollment of time taken by the ground station.

6.4.1 QCheckBoxes for Graphs

A *QCheckBoxes* are placed in a scroll area to save space in the interface (see figure 22). Each *QCheckBox* allows toggling a single telemetry data for plotting. A *QComboBox* is there for ability to switch between the graphs. Each graph in *QComboBox* has a own set of *QCheckBoxes*. The *QComboBox* is not visible if *MAV window* contains only one graph. The *QComboBox* contains a number of graph if *MAV window* has two or four graphs.

6.4.2 Graph legend

The legend is placed in a scroll area to save space in the interface. The legend was added there for a better orientation in the graph. It contains a label with color and name for each signal that can be plotted in any graph (see figure 23).



(a) MAV window with one graph

(b) MAV window with two or more graphs

Figure 22: Scroll area with QCheckBoxes

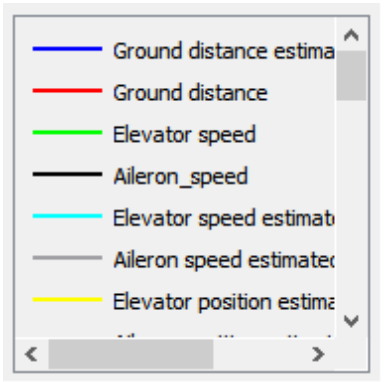


Figure 23: Scroll area with legend

6.5 Connecting GUI with MAV

The MAV is connected to *MAV window* over a *QMenu* (see figure 24) every time the *MAV window* is opened. The address of the XBee is loaded by pressing a button, which is named after the XBee. Following actions are performed after pressing any button. At first, GUI loads the address of the XBee and sends commands that enable acquistition the specific telemetry data for the failure detection. Finally, the GUI starts the failure detection thread (more about failure detection thread is in subsection 6.9) and sets the address of the XBee for the failure detection thread.

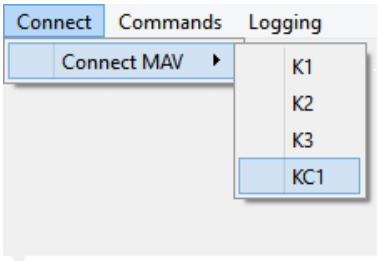


Figure 24: Connect QMenu

6.6 Commands for control the MAV

Commands are included in *QMenu* (see figure 25), where commands can be selected and then sent to the MAV. The first command is used to land the MAV. Additional commands are used to enable or disable controllers or to set points for altitude, aileron or elevator. Trajectory command is there only to turn on or off following preset trajectory points. Gumstix command enables or disables acquistition of telemetry data that shows whether circular pattern on the another MAV is registered. Group commanding is the *QCheckBox* which determines if commands are sent to the specific MAV or to all MAVs.

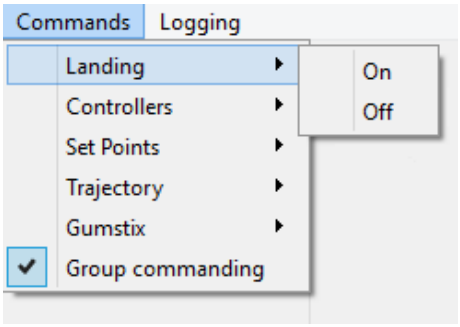


Figure 25: Commands QMenu

6.7 States of MAV

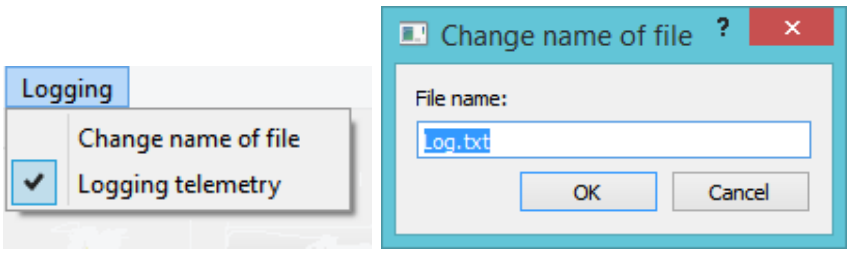
The GUI can observe four states from MAV (see figure 26). Controller status shows which controller is turned on. Landing status indicates whether the MAV recieved a command to land. Gumstix status determines if camera module sees a circular pattern or not. Trajectory follow status determines if following preset trajectory points is turned on or off. States are refreshed with the highest priority rate dependent on a workload of the ground station.

Figure 26: States in QGroupBox

6.8 Logging telemetry

Logging telemetry starts every time the *MAV window* is opened. Filename depends on the date and the time of startup of the *MAV window*. The *MAV window* gets the time from the ground station during initialization of all necessary variables. The GUI is logging telemetry data recieved via API every time it draws another point to the graph. Logging telemetry is running on the same thread as graphs. The first record on the line is always a time from ground station, then follows telemetry data according the graph legend.

Logging telemetry is switched on during initialization. For turning telemetry data on and off is used *QCheckBox* in *LogMenu* (see figure 27a). Logging telemetry can be turned on and off during runtime. The file name of log can be also changed in logging menu in following dialog window (see figure 27b). Telemetry will start logging to the new file if filename was confirmed.



(a) Logging QMenu (b) QDialog of change file name

Figure 27: Logging figures

6.9 Failure detection

Failure detection is designed to find and alert errors. Failure detection is implemented as a thread, which is started after XBee address is selected. The *Failure detection thread* begins with inicialization of variables and matrices for model. Variables and matrices are used for calculating the states of modeled system. Matrices and work with them is implemented by Eigen library [7]. Each *MAV window* has its own *Failure detection thread*.

Errors are written in array where *1* represents an error and *0* represents no error. The error array is checked together with the MAV states. Each position of the array represents one controller. Each array position is checked separately in order to determine the origin of the error. More about failure detection is in section 5.

6.9.1 Representation of failure detection

Error area contains the two *QLabels* and the *QButton* (see figure 28). Error area is contained in the *MAV window*. Text of the first *QLabel* is allways "Error status:". The errors are shown in the second *QLabel*. Text of the second *QLabel* is showing error text. When an error is detected, the GUI will change color of the second *QLabel* to red and the text of the second QLabel is changed to "ERROR" (see figure 28b). When no errors are detected, the GUI will change color of the second *QLabel* to green and the text of the second *QLabel* is changed to "No errors" (see figure 28a).

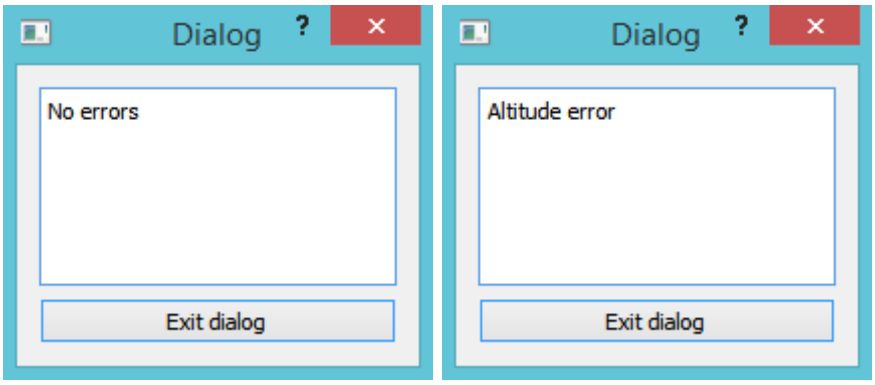
(a) No errors detected

(b) Errors detected

Figure 28: Area with error labels and button

The *QButton* opens a *QDialog*. The *QDialog* contains a *QTextArea* and the *QButton* (see figure 29). The *QTextArea* contains text dependent on error array. The *QButton* is used to end the *QDialog*. When the array does not contain any errors, *QDialog* will write to the *QTextArea* "No errors" (figure 29a). In case of error ocurance, name of failed controller

is written in the *QTextArea* (figure 29b).



(a) Error dialog window when error is not detected

(b) Error dialog window when error is detected

Figure 29: Error dialogs

7 Experiments

Experiments were performed to determine functionality of application and to detect possible errors.

7.1 Experiment with two MAVs

Experiment with two MAVs was designed to verify the functionality of the GUI. The GUI was able to draw graphs for both MAVs at the same time. The GUI was also able to show states of MAVs. The GUI was used as a support for the experiment with the blob detection. Display status of gumstix (the first MAV with the camera module sees a circular pattern on the second MAV) was required from the GUI.

Figure 30: Two MAVs in flight

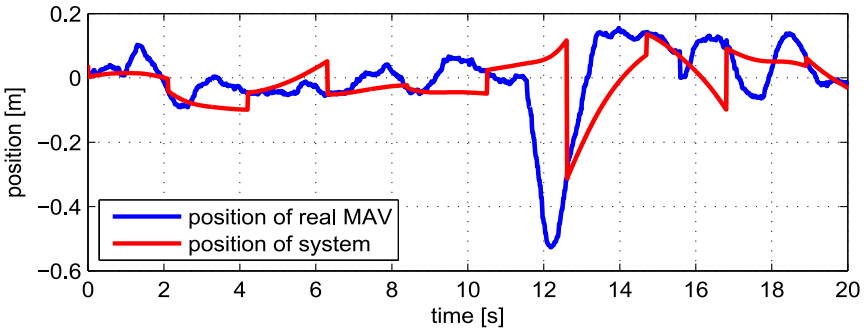
7.2 Failure detection on a real MAV

This experiment was designed to verify the functionality of the Failure detection on a real MAV.

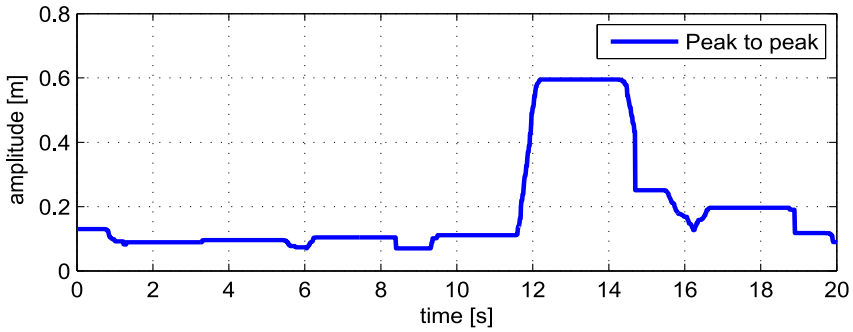
7.2.1 Attitude system

Attitude *Failure detection system* behaved satisfactorily while following the position of real MAV (see figure 31). Behavior of the altitude *Failure detection system* and the real MAV is presented in figure 31a. Altitude *Failure detection system* behaved satisfactorily

while following the attitude position of real MAV. There is an error in figure 31 starting at 11 seconds. Threshold was set experimentally by measured data. The error signal is shown in figure 31b.



(a) Position of the attitude system



(b) Error signal of attitude system

Figure 31: Attitude system

7.2.2 Altitude system

Altitude of *Failure detection system* was less precise. Altitude of *Failure detection system* with gravitational force (see section 5.1.1) was not identical with real MAV altitude. Control action from MAV is changed based on gravitational pull, loss of battery power, value of RC receiver etc. Futhermore, Control action depends on external disturbance (e.g. wind). Control action from MAV was small to maintain the correct position of *Failure detection system* (see figure 32).

The gravitational pull was subsequently removed from altitude *Failure detection system*. States, which were included into the altitude *Failure detection system* had to be modified too. The gravitational acceleration was removed from the states. Control action from MAV has also been modified. Integral component was subtracted from the control action from MAV. Integral component compensates control action according to the situation such as gravitational pull, loss of battery power etc.

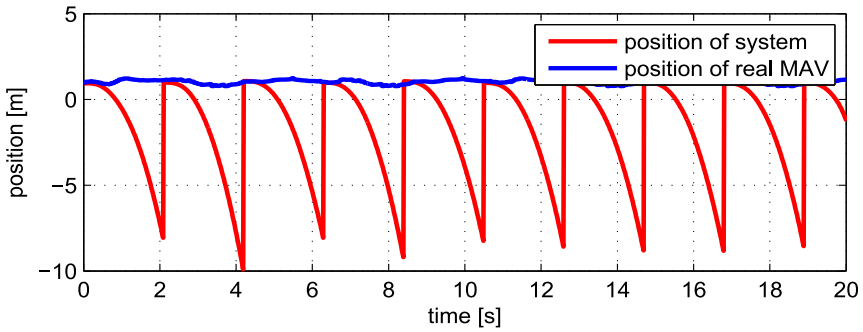


Figure 32: Altitude system with gravitional force

New altitude *Failure detection system* is defined as

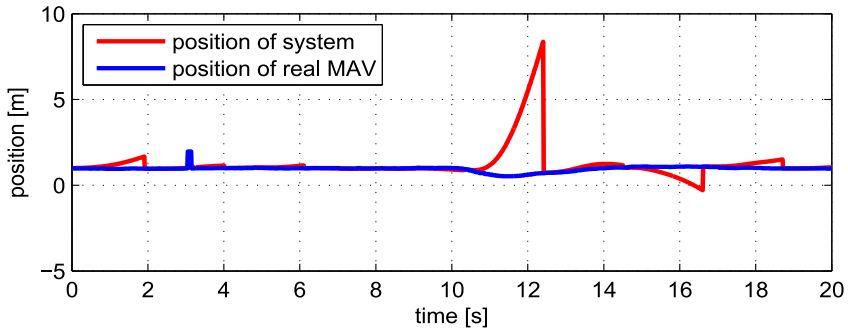
$$\mathbf{q}_z = (z, \dot{z}, \ddot{z})^T, \mathbf{u}_z = (U_D)^T, \mathbf{A}_z = \begin{bmatrix} 1 & 0.0114 & 0 \\ 0 & 1 & 0.0114 \\ 0 & 0 & 0.9658 \end{bmatrix}, \mathbf{B}_z = \begin{bmatrix} 0 \\ 0 \\ 0.0023 \end{bmatrix}, \tag{10}$$

where z represents altitude position, $\dot{z}$ represents altitude speed and $\ddot{z}$ represents altitude acceleration. Input U_D represents collective thrust of a propeller.

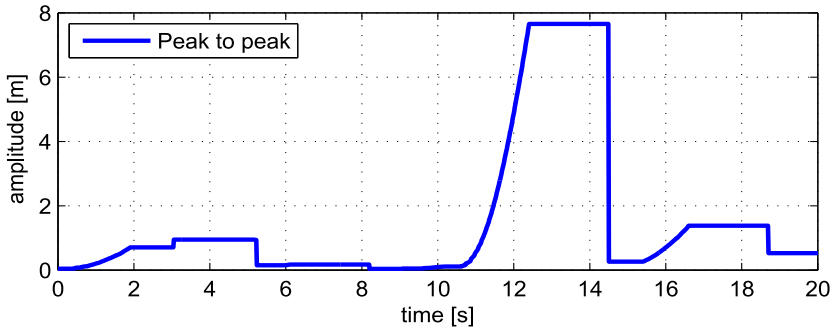
Altitude of *Failure detection system* was modified and the result is presented in the figure 33. Behavior of the system and the real MAV is shown in the figure 33a. Altitude *Failure detection system* behaved satisfactorily while following the position of the real MAV. In the figure 33 is presented caused error that starts at 10 second. Threshold was set experimentally by measured data. The error signal can be seen in the figure 33b.

7.3 Summary

Both systems of the *Failure detection system* are able to detect errors caused by an unexpected behavior of MAV. Altitude *Failure detection system* with gravitional pull does not behave in the real-world conditions according to MAV. Altitude *Failure detection system* was modified by removing the gravitional pull and changing control action from the MAV. New altitude *Failure detection system* is able to predict a behavior of a MAV reliably after the adjustments.



(a) Altitude system without gravitional force



(b) Error signal of altitude system

Figure 33: Altitude system

8 Conclusion

This thesis was aimed to design, develop and test GUI and failure detection which was made for support experiments with a swarm of simultaneously flying MAVs. I have designed and implemented GUI which supports experiments with the real MAVs. Also I have created failure detection which works with the real MAV. Failure detection is able to detect errors caused by controllers in realtime.

The GUI was designed and implemented by using the QT framework and QT Creator. The GUI can open more windows at the same time. Each window supports one the MAV. The GUI can display realtime telemetry and states of the MAV. The GUI can also send commands to control the MAV. The GUI contains a thread for the *Failure detection*.

Experiments with simulated model of the MAV demonstrate the functionality of the *Failure detection*. Simulation of the *Failure detection system* is slightly delayed but it detects errors reliably.

The first real experiment with two MAVs represents the functionality of the work with more MAVs at the same time. The other experiment with MAV is targeted at failure detection in the real world conditions. Experiment showed that the *Failure detection system* responds to the MAV. Errors in the real world conditions are detected reliably after adjusting the threshold experimentally by measured data.

During the work on the thesis I have learnt that working with a real hardware requires a lot of time. I have also learnt that simulations and experiments are very important in any new development because they can reveal some hidden faults and illustrate the functionality of the work. I have also learnt that theoretical model is different from a real word dynamic system.

The GUI which was created in this thesis is a useful tool for experiments with MAVs. The GUI can draw graphs from telemetry data, which is received from the MAV. The GUI can also show states of MAVs or sending commands to one MAV. The failure detection is detecting errors caused by controllers on the MAV. The mentioned features of my work could be useful in following applications. Firstly, created GUI can be used for performed experiments with MAV formations. The realtime failure detection features are very helpful [25, 23, 22]. Secondly, the GUI was implemented so that it supports any number of MAVs. This can be useful for experiments with MAV swarms [24, 21]. Futhermore GUI can show telemetry data for each individual member of the swarm.

9 Bibliography

[1] Atmel. Atxmega 128a3u - http://www.atmel.com/devices/atxmega128a3u.aspx, October 2014.

[2] Tomáš Báča. Model predictive control of micro aerial vehicle using onboard microcontroller. Master's thesis, Czech Technical University.

[3] The QT company. https://www.qt.io/qt-framework/, 2015.

[4] L. Cork and R. Walker. *Sensor Fault Detection for UAVs using a Nonlinear Dynamic Model and the IMM-UKF Algorithm.* Information, Decision and Control - IDC, 2007.

[5] Breaking defense. http://breakingdefense.com/2015/03/teaching-drones-how-to-see-fire-scout-kestrel/, March 2015.

[6] Digi. http://www.digi.com/, 2015.

[7] Eigen. http://eigen.tuxfamily.org/, January 2015.

[8] Václav Endrych. Control and stabilization of an unmanned helicopter following a dynamic trajectory. Master's thesis, Czech Technical University.

[9] Jirka Fiedler. Synchronized control of group of helicopters using direct communication. Bachelor's thesis, Czech Technical University.

[10] Gerardo R. Flores-Colunga, H. Aguilar-Sierra, R. Lozano, and S. Salazar. Faulr estimation and control for a quad-rotor mav using a polynomial observer. part i: Fault detection. In *First Iberian Robotics Conference: Advances in Robotics*, 2013.

[11] Michael Frangenberg, Johannes Stephan, and Walter Fichter. Fast actuator fault detection and reconfiguration for multicopters. In *AIAA Guidance, Navigation, and Control Conference*, 2013.

[12] P. Freeman, R. Pandita, N. Srivastava, and G. J. Balas. *Model-Based and Data-Driven Fault Detection Performance for a Small UAV.* Mechatronics, IEEE/ASME Transactions, Volume 18, Issue 4, May 2013.

[13] Gumstix. https://www.gumstix.com/, 2015.

[14] G. Heredia, A. Ollero, M. Bejar, and R. Mahtani. *Sensor and actuator fault detection in small autonomous helicopters.* Mechatronics, Volume 18, Issue 2, March 2008.

[15] Vice Media LLC. http://motherboard.vice.com/blog/turns-out-drones-make-great-firefighters, 2015.

[16] project of CentMesh. https://sites.google.com/a/ncsu.edu/firefighting-drone-challenge/, September 2014.

[17] Emanuel Eichhammer QCustomPlot. http://www.qcustomplot.com/, April 2015.

[18] Thomas Rakotomamonjy and Thomas Rakotomamonjy. A negative selection algorithm applied to helicopter actuator fault detection. In *Control, Decision and Information Technologies (CoDIT)*, 2014.

[19] S. Sanchez, M. Perhinschi, and H. Moncayo. In-flight actuator failure detection and identification for a reduced size uav using the artificial immune system approach. In *AIAA Guidance, Navigation, and Control Conference*, 2009.

[20] M. Saska, J. Chudoba, L. Přeučil, J. Thomas, and G. Loianno. Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance. In *International Conference on Unmanned Aircraft Systems*, 2014.

[21] M. Saska, J. Chudoba, L. Preucil, J. Thomas, G. Loianno, A. Tresnak, V. Vonasek, and V. Kumar. Autonomous Deployment of Swarms of Micro-Aerial Vehicles in Cooperative Surveillance. In *Proceedings of 2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, volume 1, pages 584–595, Danvers, 2014. IEEE Computer society.

[22] M. Saska, Z. Kasl, and L. Preucil. Motion Planning and Control of Formations of Micro Aerial Vehicles. In *Proceedings of The 19th World Congress of the International Federation of Automatic Control*, pages 1228–1233, Pretoria, 2014. IFAC.

[23] M. Saska, T. Krajnik, V. Vonasek, Z. Kasl, V. Spurny, and L. Preucil. Fault-Tolerant Formation Driving Mechanism Designed for Heterogeneous MAVs-UGVs Groups. *Journal of Intelligent and Robotic Systems*, 73(1-4):603–622, January 2014.

[24] M. Saska, J. Vakula, and L. Preucil. Swarms of Micro Aerial Vehicles Stabilized Under a Visual Relative Localization. In *ICRA2014: Proceedings of 2014 IEEE International Conference on Robotics and Automation*, pages 3570–3575, Piscataway, 2014. IEEE.

[25] M. Saska, V. Vonasek, T. Krajnik, and L. Preucil. Coordination and Navigation of Heterogeneous MAV–UGV Formations Localized by a ‘hawk-eye’-like Approach Under a Model Predictive Control Scheme. *International Journal of Robotics Research*, 33(10):1393–1412, September 2014.

[26] SparkFun. https://github.com/sparkfun/openlog/wiki/command-set, August 2014.

[27] SparkFun. Openlog - https://www.sparkfun.com/products/9530, 2015.

[28] STMicroelectronics.

Appendix A CD Content

In Table 6 are listed names of all root directories on CD.

Directory name	**Description**
thesis	Bachelor's thesis in pdf format.
thesis_sources	latex source codes
GUI	C++ source codes
Photos	Photos of experiments

Table 6: CD Content

Appendix B List of abbreviations

In Table 7 are listed abbreviations used in this thesis.

Abbreviation	Meaning
UAV	unmanned aerial vehicle
MAV	micro aerial vehicle
GUI	graphical user interface
API	application programming interface
UART	universal asynchronous receiver-transmitter
MPC	model predictive controller
LTI	linear time invariant

Table 7: Lists of abbreviations

Appendix C The MAV window

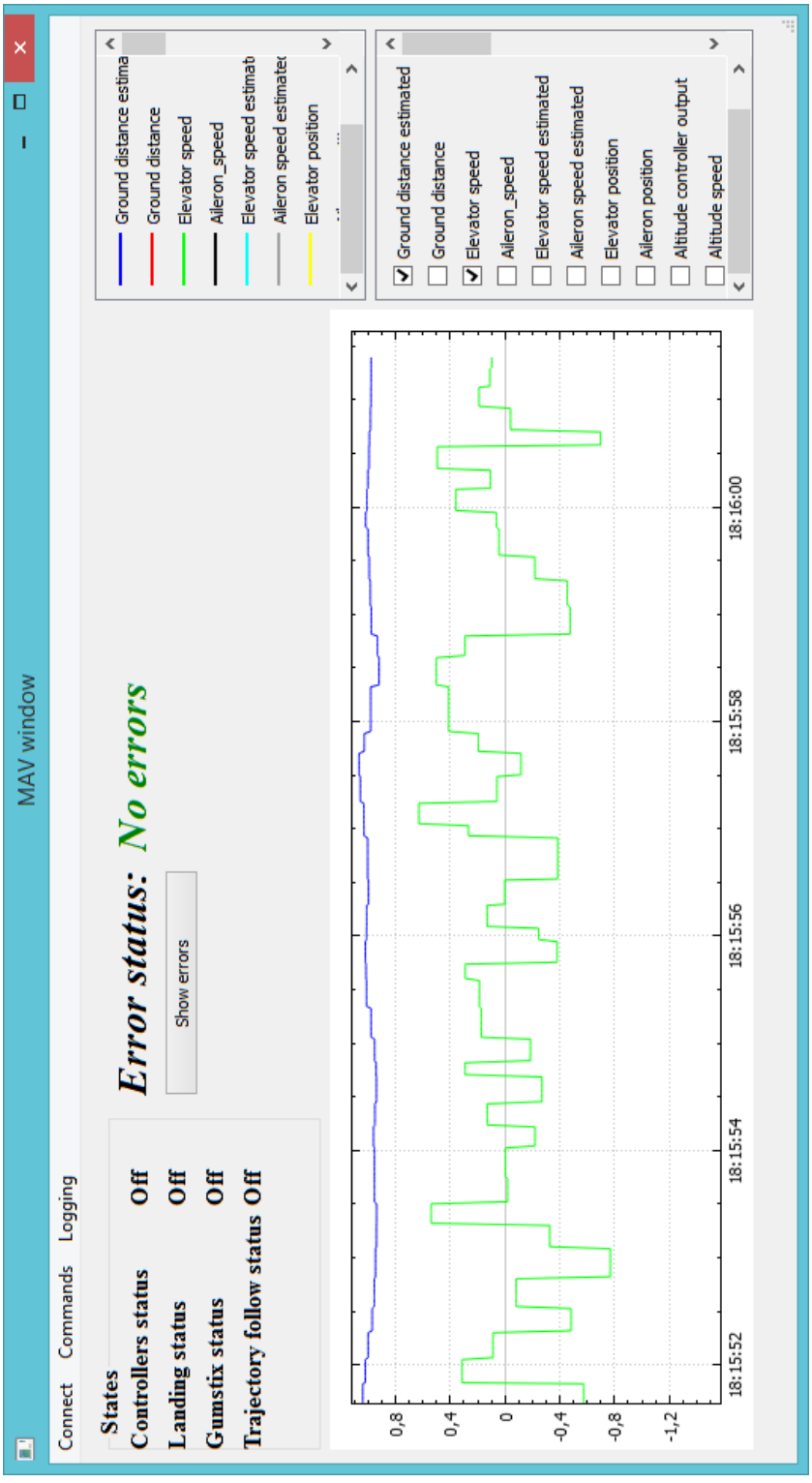


Figure 34: The MAV window

Appendix D The Failure detection

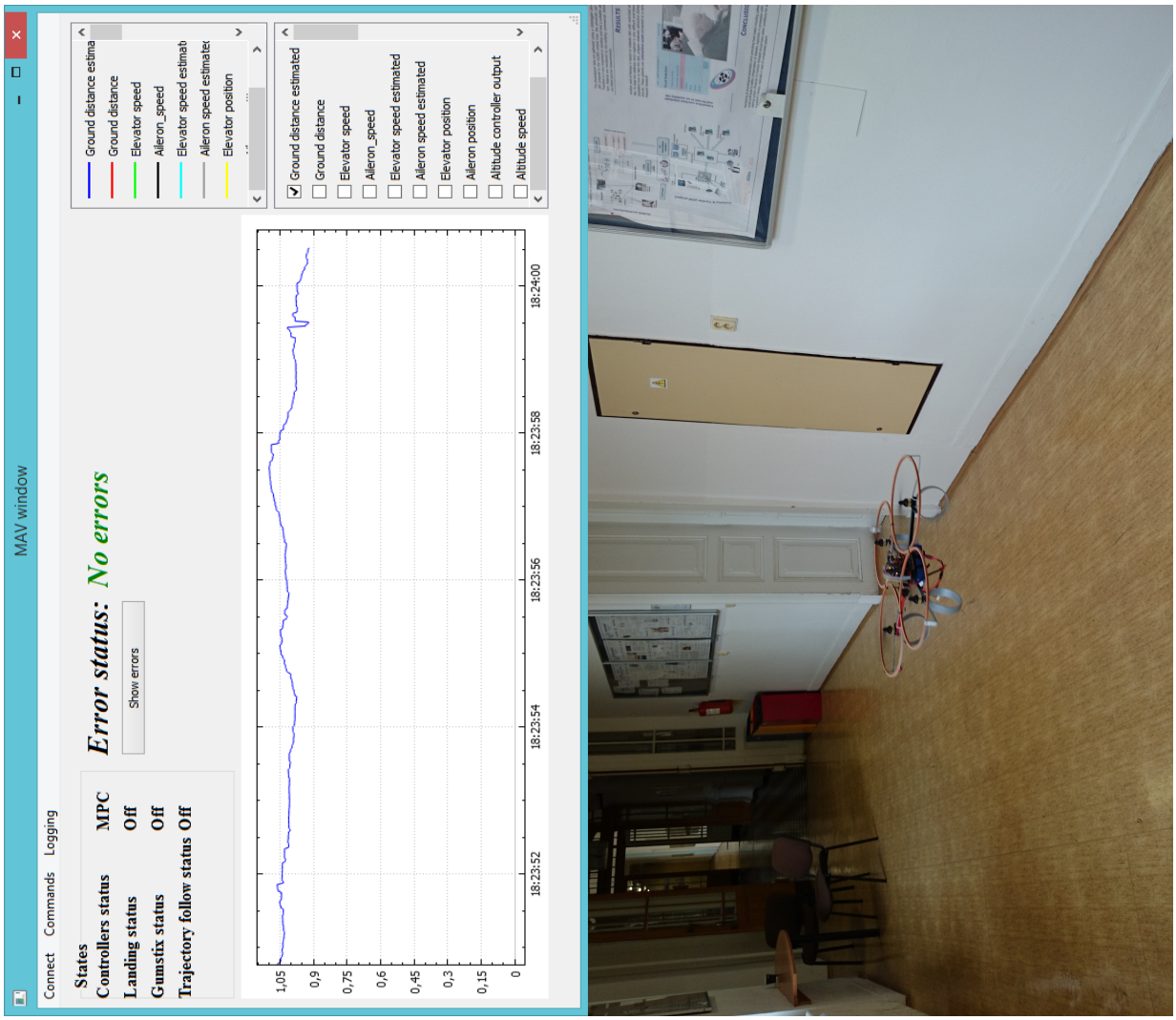


Figure 35: Illustration of Failure detection

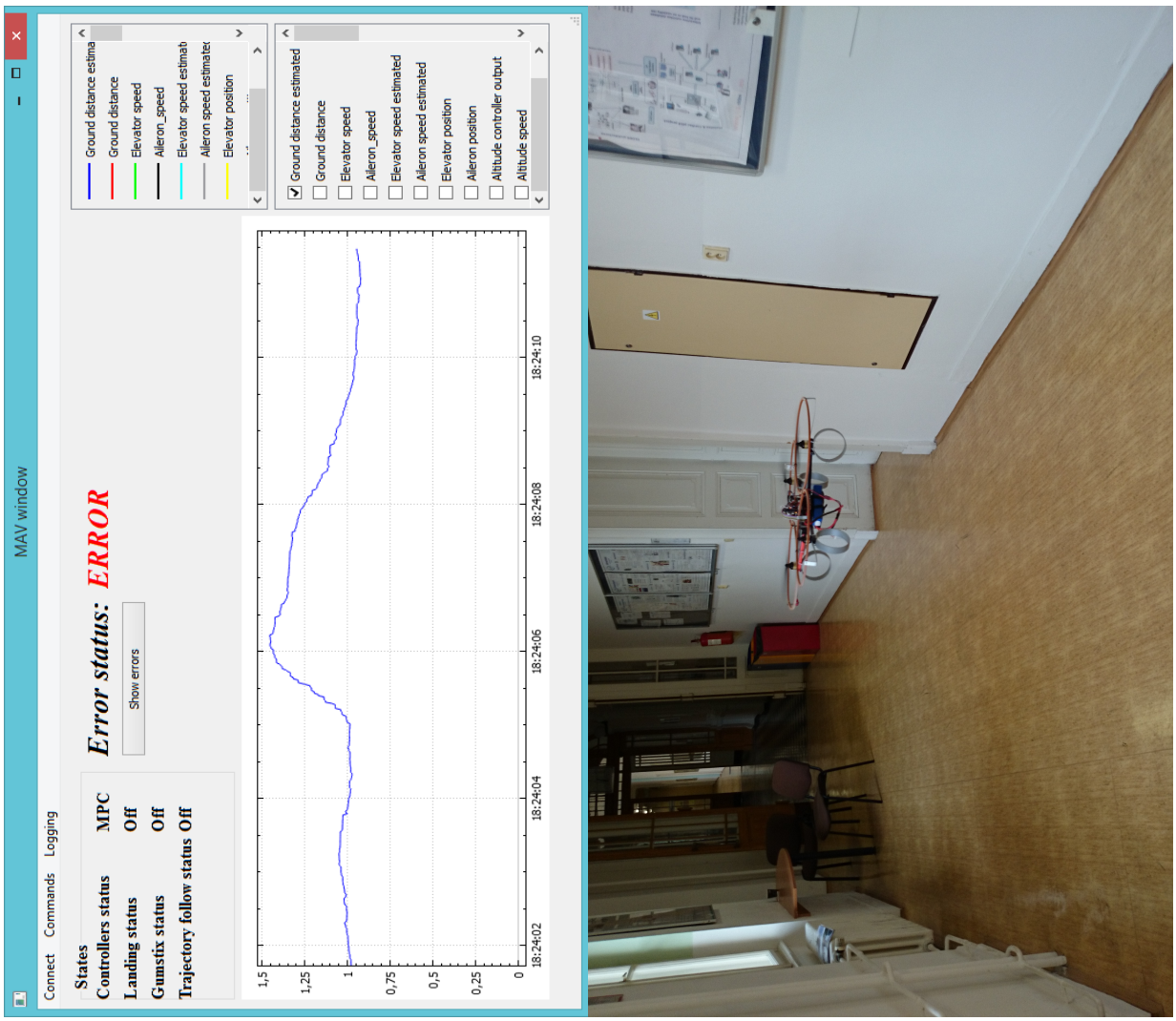


Figure 36: Illustration of Failure detection

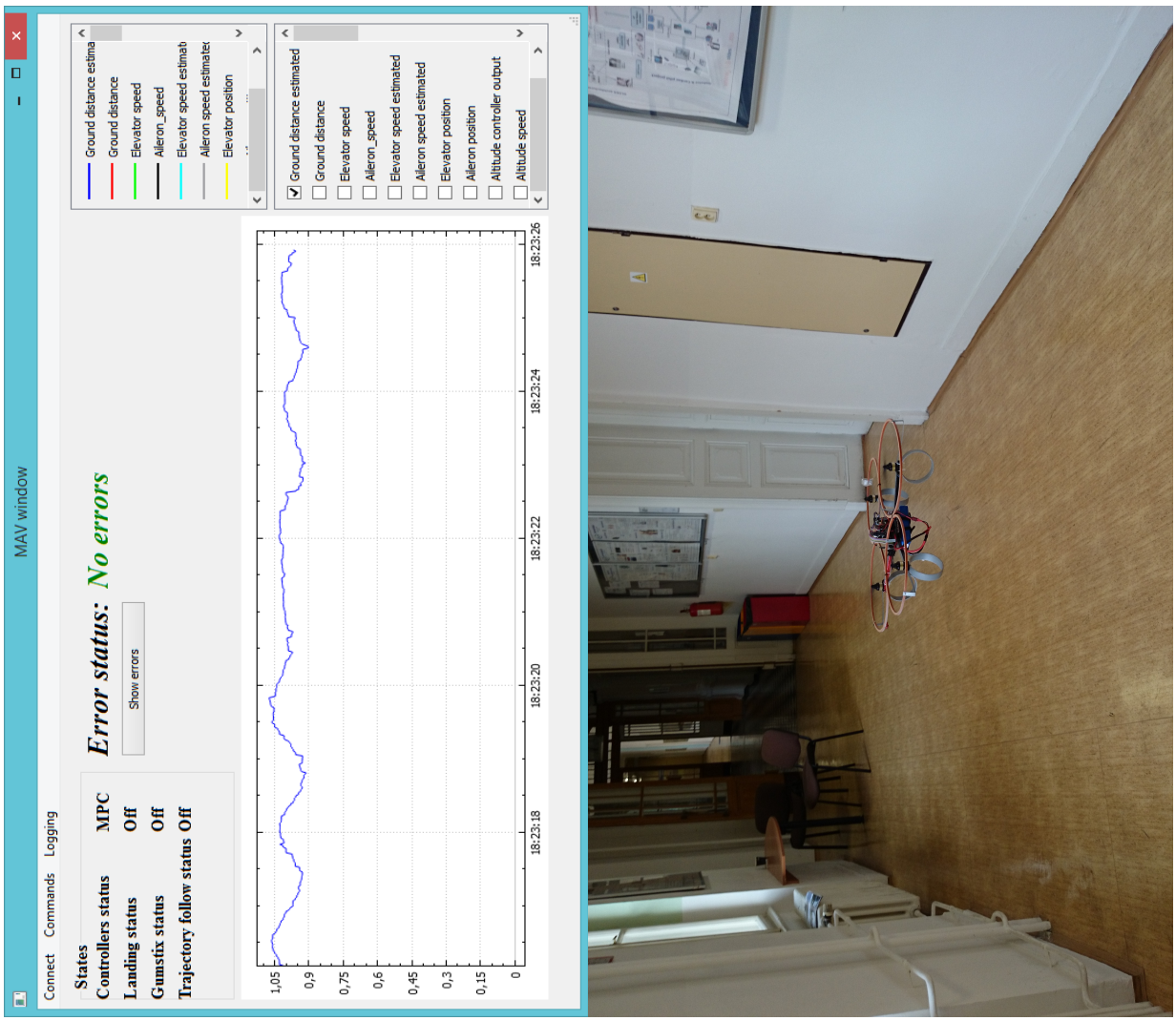


Figure 37: Illustration of Failure detection