Mapping robot documentation

Jedrzej Herbik

March 13, 2021
v1.0.0

Contents

List of Figures

Chapter 1

Project introduction.

Main purpose of project is platform which knows position of itself and can generate map of room. Platform should know where is in this map.
Project is created with idea to further develop into vacuum-cleaner or grass cutter.

Chapter 2

Hardware and main design assumptions.

2.1 Description and assumptions of platform.

Hardware design required microprocessor which will manage all peripherals as engines and ultrasonic distance meters based on interrupts functions, processor based on linux system as raspberry Pi for advanced algorithms.
Microprocessor as stm32 for peripherals, guarantee fast manage of measurements without big delays.
All measurements will be shared by SPI bus.
Same bus will be used to manage engines by sending coordinates for PIDs.
Microprocessor will be configured by data from EEprom memory which will be set by Linux based processor.
Linux based processor is master of SPI transmission.
Between MPUs is UART bus to log informations about microprocessor state.

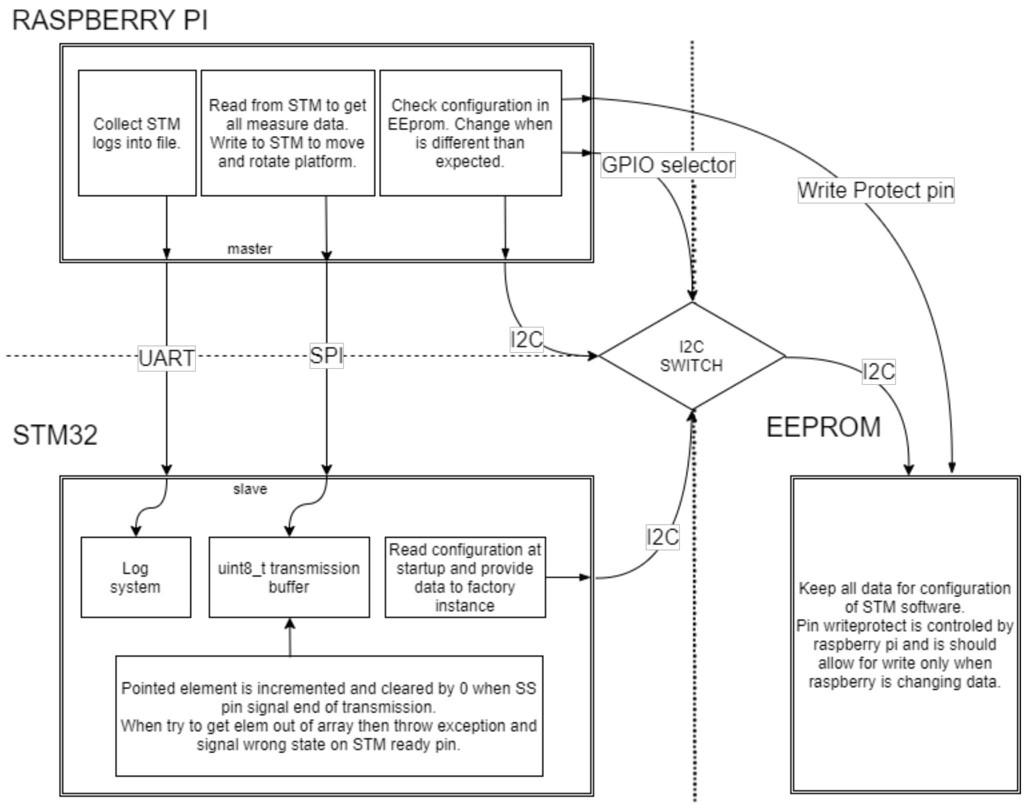
RASPBERRY PI
Collect STM logs into file.
Read from STM to get all measure data. Write to STM to move and rotate platform.
Check configuration in EEprom. Change when is different than expected.
master
GPIO selector
Write Protect pin
UART
SPI
I2C
I2C SWITCH
I2C
EEPROM
STM32
slave
Log system
uint8_t transmission buffer
Read configuration at startup and provide data to factory instance
I2C
Pointed element is incremented and cleared by 0 when SS pin signal end of transmission. When try to get elem out of array then throw exception and signal wrong state on STM ready pin.
Keep all data for configuration of STM software. Pin writeprotect is controled by raspberry pi and is should allow for write only when raspberry is changing data.

Figure 2.1: Main connection between main modules.

2.2 Hardware solutions for room mapping.

To create map of surrounding area, platform will use distance meters.
In first board version are used 3 ultrasonic distance meters similar to attached photo.

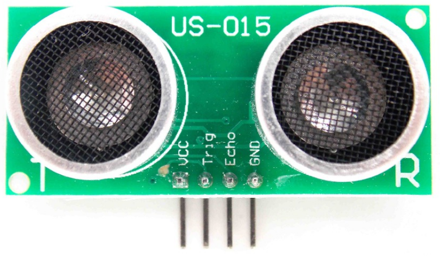

Figure 2.2: US-015 ultrasonic distance meter.
Used distance meter has to have 4 pin in order:
VCC, TRIG, ECHO, GND from left pin to right pin.
First board version uses 3 distance meters for measure:
left side (-90°), front side (0°) and right side (90°).
Projects also sets up to use one distance meter rotated by servomechanism.
Those data will be shared by SPI to raspberry where data will filtered be processed.

2.3 First PCB board explanation as example.

Main features on board:

- Raspberry Pi 3B+ socet
- stm32f411ret with external oscilator 8MHz
- L298P - engines driver
- MPU6050 - accelerometer and gyrospkope
- AT24C - EEprom memory
- SN74LVC1G3157 to select I2C bus
- LM7805 and AP1117 - 5V and 3.3V supplies
- Socet for UART adapter
- socets for 3 ultrasonic sensors
- socet to connect stm programator
- switch to sellect UART connection

- 2 couples for engine connectors
- Connector for 2 LiPo batteries
- switch for power supply

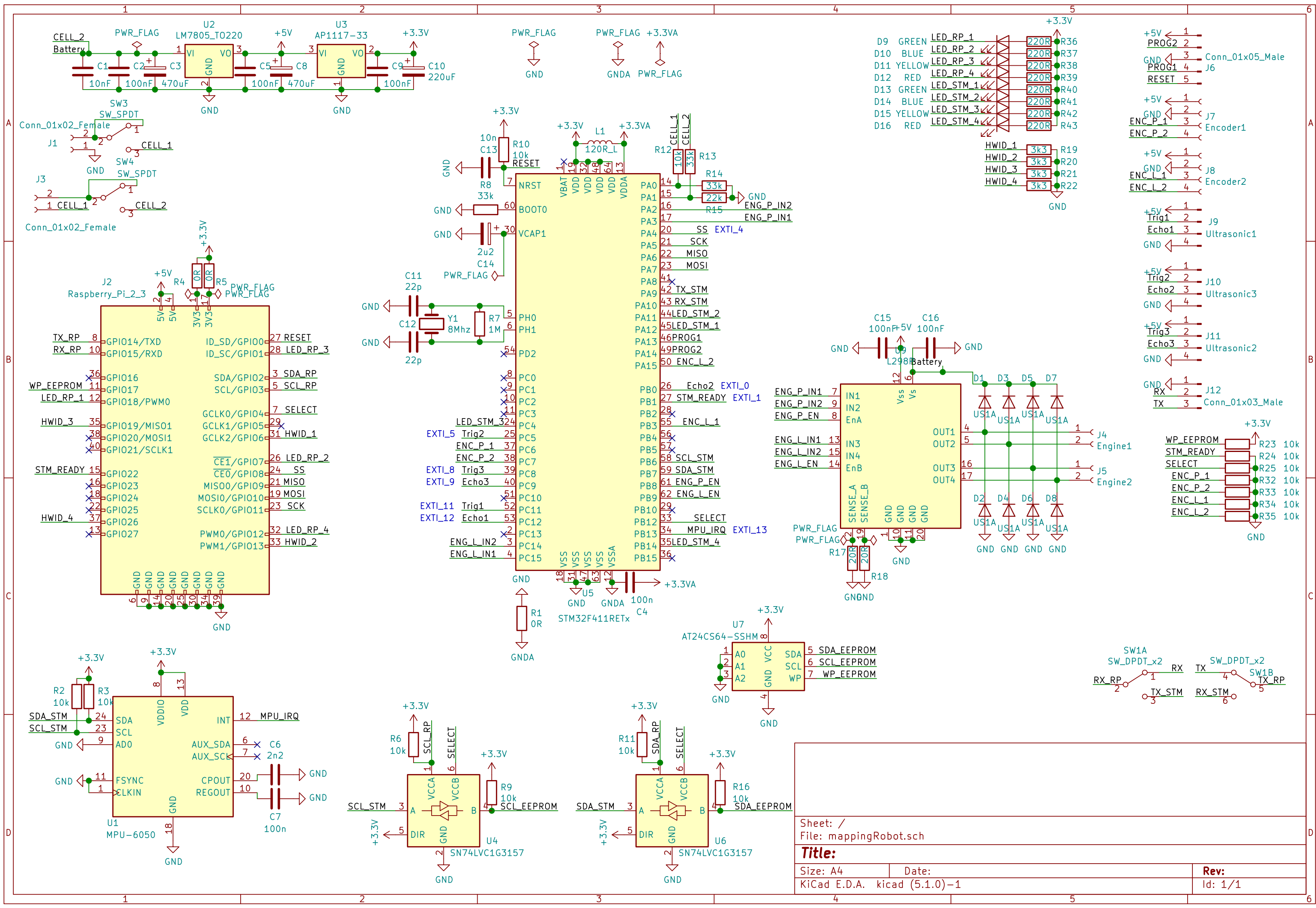

Figure 2.3: Electronic schematic of first version of mapping platform.

Chapter 3

STM code.

3.1 Main assumptions.

Stm is used to manage HW where time is very important. Main purposes are distance measurements, engines PIDs and encoders monitoring. Because of important role of time, every time dependent issue has to be solved in driver methods triggered by interrupt functions. It's recommended to use static allocation and dynamic allocation during startup/initalise procedure. It's recommended to not use dynamic allocation during runtime of stm.

3.2 Architecture.

In project will be used drivers from common driver library. It's recommended to define platform drivers also in Drivers namespace defined in this library. All necessary drivers from this namespace (common and platform specific) could be given into main logic part by structure with pointers to them. All drivers have to be initialised in area close to factory. Factory is structure to create all platform drivers based on data in eeprom memory.

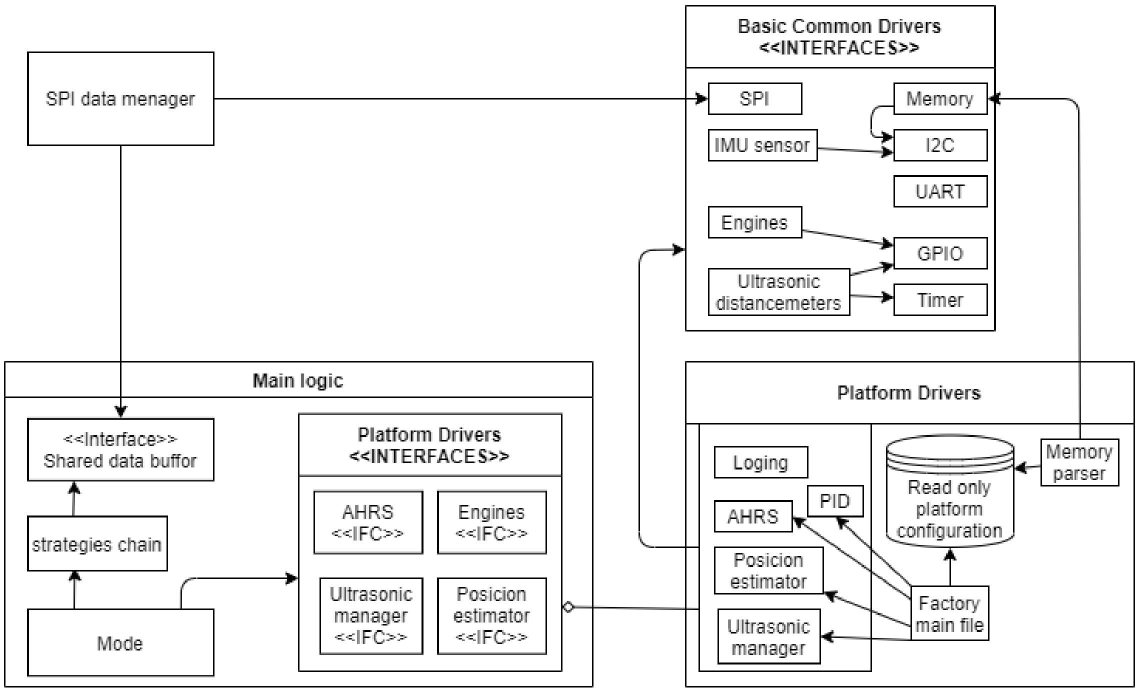

Figure 3.1: General schema of stm system.

Interfaces of common drivers are defined by common drivers repository.

3.3 Communication buses.

All buses should use DMA functionality. If stm doesn't has enough DMA streams to cover all buses, then most prioritized are buses UART for logging and SPI for communication with Raspberry Pi.

Chapter 4

Algorithms.

This chapter describes all mathematical and logic solutions for problems occured in projects.

4.1 Engines encoders.

In projects it's important to know current position.
To get this information it has to be calculated from engine encoders.
To get reliable estimation of position it's necessary to assump lack of wheels slips.
Second assumption, engines control is set in same loop as encoders calculation to position (that's mean with the same frequency).
Lack of slips allows to simplify mathematical model to pair of wheels rotating with constant speed equal or different for both wheel.
Equal speed of wheels means that pair move forward or backward at straight line.
Different speed of wheels means they move on curve with constant radius until moment of next calculation (next engine control value set).
Drawing below presents pair of wheels in (0, 0) position rotated by β angle.

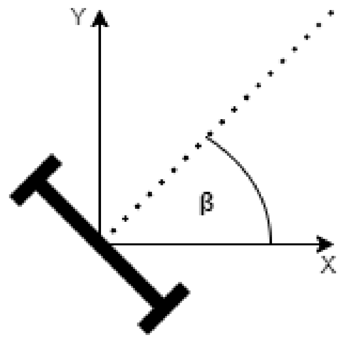

Figure 4.1: Wheels in 0,0 position rotated by β angle.

Situation in picture below presents pair of wheels in point x', y' βangle rotated,
which moved into new position x, y and rotation incresed by ψ angle.
Distance between point (x', y') and point (x, y) is ΔS.
Every wheel moves on circle with common center and by same angle ψ.
Left wheel move by S_L and right wheel move by S_R. Radius of outer wheel equal to $R + \frac{L}{2}$.
Radius of inner wheel equal to $R - \frac{L}{2}$.

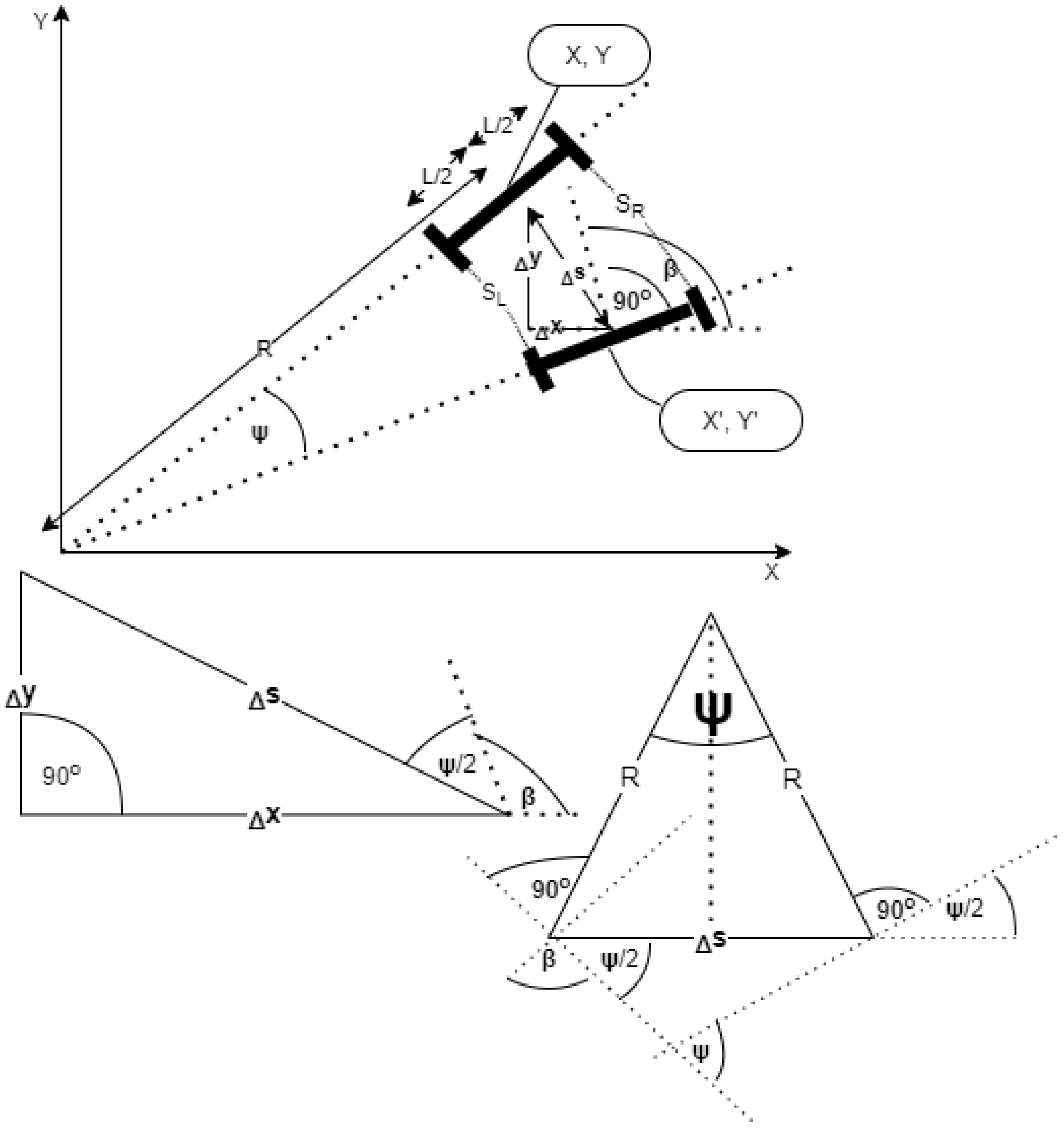

Figure 4.2: Engines encoders equations drawings.

Triangle created by both radius R and movement is isosceles triangle. That allows to calculate movement from radius and angle ψ.
ΔS, Δx, Δy create rectangular triangle which allows to calculate all of those 3 values based on (β+ ψ/2) angle.

If S_L is equal to S_R then equations .1 and .2 have to be used.

In other case equations from .3 to .7.

$$\begin{cases}\Delta x = S_R * cos(\beta) \\ \Delta y = S_R * sin(\beta)\end{cases} \tag{4.1}$$

$$\begin{cases} x_{n+1} \\ y_{n+1} \\ angle_{n+1} \end{cases} = \begin{cases} x_n + \Delta x \\ y_n + \Delta y \\ angle_n \end{cases} \tag{4.2}$$

$$\begin{cases} S_R = 2 * \pi * (R + \frac{L}{2}) * \frac{\psi}{2*\pi} \\ S_L = 2 * \pi * (R - \frac{L}{2}) * \frac{\psi}{2*\pi} \end{cases} \Longrightarrow \begin{cases} S_R = R * \psi + \frac{L}{2} * \psi \\ S_L = R * \psi - \frac{L}{2} * \psi \end{cases} \quad /+ \Longrightarrow$$

$$\Longrightarrow S_R + S_L = 2 * R * \psi \Longrightarrow R = \frac{S_R + S_L}{2 * \psi} \tag{4.3}$$

$$\begin{cases} R = \frac{S_R + S_L}{2*\psi} \\ S_R = R * \psi + \frac{L}{2} * \psi \end{cases} \Longrightarrow S_R = \frac{S_R + S_L}{2 * \psi} * \psi + \frac{L}{2} * \psi \Longrightarrow \frac{L}{2} * \psi = S_R - \frac{S_R + S_L}{2} \Longrightarrow$$

$$\Longrightarrow L * \psi = 2 * S_R - (S_R + S_L) \Longrightarrow \psi = \frac{S_R - S_L}{L} \tag{4.4}$$

$$\frac{\Delta S}{2 * R} = \sin \frac{\psi}{2} \Longrightarrow \Delta S = 2 * R * \sin \frac{\psi}{2} \tag{4.5}$$

$$\begin{cases}\Delta x = \Delta S * cos(\beta + \frac{\psi}{2}) \\ \Delta y = \Delta S * sin(\beta + \frac{\psi}{2})\end{cases} \tag{4.6}$$

$$\begin{cases} x_{n+1} \\ y_{n+1} \\ angle_{n+1}(\beta) \end{cases} = \begin{cases} x_n + \Delta x \\ y_n + \Delta y \\ angle_n + \psi \end{cases} \tag{4.7}$$

Chapter 5

Raspberry Pi and stm SPI interfaces.

All data transmitted and received via SPI interface has to be converted to 8-bits type and after SPI transaction reconverted to origin type. Also project required to send/receive different kind of variables depends on working mode (Normal mode, PID tuning mode, etc.). Because of mode, interface needs mechanism to choose what packages of variable will be parsed. It's necessary to have solution flexibly changing of parsing package of bytes to different variables. In code in Raspberry and STM it's solved by chain of strategies. One strategy can convert PID set value or battery percent state to group of bytes and group of bytes back to specific variable. Those base strategies has to be defined in STM side and Raspberry side. Then on both devices could by define same chain of strategies executed every time after SPI transmission.

5.1 Strategies.

In this section will be listed all strategies supported and required for STM and Raspberry. Every strategy should be derived class of strategy interface. This interface should contain 4 virtual methods:

- to parse incoming transaction data
- to parse outcoming transaction data
- returning amount of bytes which are required to parse
- introducing own name

Data to parse should be given to first strategy. Every strategy after successful processing should pass the data decreased by those already used, to next strategy in relation chain. If processing was done with failure, strategy cannot execute next strategy. Instead, it should signal an error.

5.1.1 Position strategy.

This strategy parsing information about:

- actual position(X, Y, rotation) - current position of robot.
- expected position(X, Y, rotation) - position which robot wants to reach.
- information does expected position should be applied / used.

Expected position and apply information is sent by Raspberry and received by STM. Actual position is sent by STM and received by Raspberry. Strategy needs 10 bytes to parse those information for SPI transaction. Before parsing, parameters X and Y for actual and expected positions are multiplied by 1000 and converted to 32-bit unsigned integer. Expected rotation should be in range $< 0^\circ, 360^\circ)$ and it's converted to 15-bit data, where 0° map to 0 and max angle $(\approx 359,99^\circ)$ map to 0x7FFF value.

Raspberry - write
STM - read

	MSB			bites				LSB
	8	7	6	5	4	3	2	1
1.	expected X [32 - 25]							
2.	expected X [24 - 17]							
3.	expected X [16 - 9]							
4.	expected X [8 - 1]							
5.	expected Y [32 - 25]							
6.	expected Y [24 - 17]							
7.	expected Y [16 - 9]							
8.	expected Y [8 - 1]							
9.	AP	expected rotation [15 - 9]						
10.	expected rotation [8 - 1]							

AP - if set to 1, position will be applied

Raspberry - read
STM - write

	MSB			bites				LSB
	8	7	6	5	4	3	2	1
1.	actual X [32 - 25]							
2.	actual X [24 - 17]							
3.	actual X [16 - 9]							
4.	actual X [8 - 1]							
5.	actual Y [32 - 25]							
6.	actual Y [24 - 17]							
7.	actual Y [16 - 9]							
8.	actual Y [8 - 1]							
9.	NU	actual rotation [15 - 9]						
10.	actual rotation [8 - 1]							

NU - not used

5.1.2 Double strategy.

This strategy parsing information of double variable. Conversion should be made using below union.
Union:

- table of 8 elements — type: 8-bit
- variable — type: double.

Strategy needs 8 bytes to parse those information for SPI transaction. Double value is sent and received in both STM and Raspberry. That means STM and Raspberry parse implementation is identical.

Raspberry - read/write
STM - read/write

	MSB			bites			LSB	
	8	7	6	5	4	3	2	1
1.	element of table 1/8							
2.	element of table 2/8							
3.	element of table 3/8							
4.	element of table 4/8							
5.	element of table 5/8							
6.	element of table 6/8							
7.	element of table 7/8							
8.	element of table 8/8							

5.1.3 Battery strategy.

This strategy parsing information of battery cells value.
Strategy contain below informations:

- voltage of cell - measured voltage of cell
- voltage to tune cell - voltage witch is used as reference connected to cell during tuning procedure
- index of cell - specify about which battery cell are shared informations

Strategy needs 2 bytes to parse those information for SPI transaction. Minimum measured voltage is 0.0 volts and maximum is 40.955 volts. Information about measured voltage is mapped as 0x0 for 0.0 volts and 0x1FFF for maximum volts. Tune voltage is in same range and map to raw byte data same as measured voltage. If value of tune voltage is equal 0.0 volts then tuning procedure shouldn't be executed. Index of cell is value from value 0 which means 1'st cell, to 7 which means 8'th cell.

Raspberry - write
STM - read

	MSB			bites			LSB	
	8	7	6	5	4	3	2	1
1.	cell index			tune voltage [13 - 9]				
2.	tune voltage [8 - 1]							

Raspberry - read
STM - write

	MSB			bites			LSB	
	8	7	6	5	4	3	2	1
1.	cell index			measured voltage [13 - 9]				
2.	measured voltage [8 - 1]							

5.1.4 Distance measure strategy.

This strategy parsing information of measurement from distance sensor.
Strategy contain only informations about distance.
Strategy needs 2 bytes to parse those information for SPI transaction.
Measured distance is in range $< 0, 2500)$ mm.
Information about measured distance is mapped as 0x0 for 0.0mm and 0xFFFF for maximum distance.
Usefull range is from 0x0 to 0xFFFE beacause value 0xFFFF means measure out of range.
In case of measure out of range distance to object should be interpreted as to far to measure.
In this strategy STM sends data to Raspberry.
Data from Raspberry could be dummy data and will not be used.

Raspberry - read
STM - write

	MSB			bites				LSB
	8	7	6	5	4	3	2	1
1.	distance measure [16 - 9]							
2.	distance measure [8 - 1]							

5.1.5 All data used strategy.

This strategy is not parsing any information and do not need any byte of data.
This strategy has to be last strategy in strategies chain.
Strategy will check are used all data from transaction. If all data are used then strategy finish processing with success and do not call next strategy. If not all data are used then strategy finish processing with failure.

5.1.6 Transmission key strategy.

This strategy parsing information of transmission key. Strategy contain one 8-bit key. Strategy needs 1 byte to parse information for SPI transaction. Strategy sends own key and receive other key. Both keys have to be known for strategy. They should be provided in configuration. If incoming key isn't identical as known key then transaction should stop processing with failure and do not call next strategies. This strategy should be first strategy in strategies chain to provide break of transmission when data are broken.

Raspberry - write
STM - read

	MSB			bites				LSB
	8	7	6	5	4	3	2	1
1.	Raspberry key [8 - 1]							

Raspberry - read
STM - write

	MSB			bites				LSB
	8	7	6	5	4	3	2	1
1.	STM key [8 - 1]							

5.2 Strategies chain.

Structure for SPI data parsing is defined per mode. Every mode has it own strategies chain which contain every strategy necessary to share data important for mode. Mode should be defined in EEprom configuration. Based on mode STM should create chain during startup. Raspberry should create chain in driver factory based on mode. Factory should create all HW drivers with strategies chain clearing old drivers and old strategy chain. It allows to change mode on Raspberry without exit program. Every factory creation process should reset STM to create strategy chain same as in Raspberry.

Below image shows example of strategies chain execution.

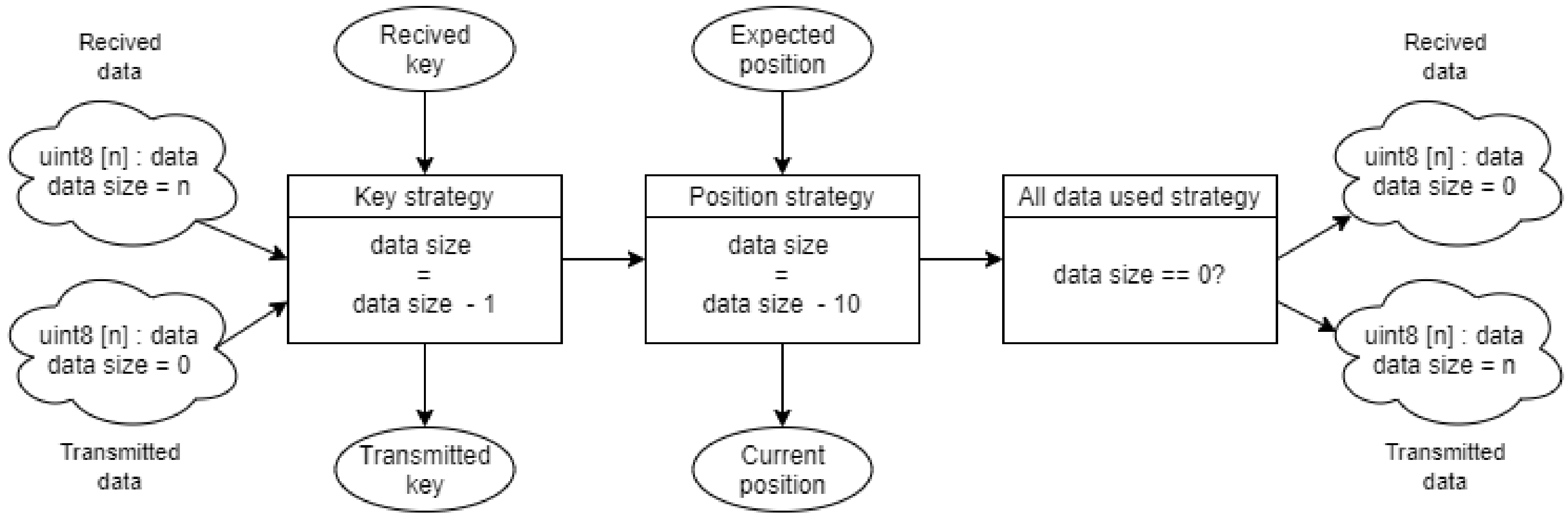

Figure 5.1: Strategies chain example.

In below subsections will be described every supported mode. Every mode has unique number. This number can be value from 0 to 255 (0xFF). This ID is used by Raspberry and STM to set environment.

5.2.1 Normal mode.

This mode ID is 0. Main idea of this mode is to use both PID for rotation and moving. In this mode robot is able to keep expected position, collect and send data about current position, battery cells and distance meters.

Strategy chain contain bellow strategies.

	Strategy	Data sent from STM to Raspberry	Data sent from Raspberry to STM
1.	Transmission key	Raspberry and STM key	Raspberry and Raspberry key
2.	Position	current angle, X, Y positions	expected angle, X, Y positions and apply flag
3.	Battery	Voltage of every cell	Tuning reference voltage for every cell
4.	Distance measure	Measurement from distance meter	none data
5.	Distance masure	Measurement from distance meter	none data
6.	Distance masure	Measurement from distance meter	none data
6.	All data used	none data	none data

5.2.2 TuneRotatePID mode.

This mode ID is 1. Main idea of this mode is to not use PID and collect data of rotation when constant value is set in engines. In this mode robot is able to collect and send data about rotation measure and battery cells.
Important!
Working frequency of PID (100Hz) and SPI transaction frequency (40Hz). That makes measured data sent by SPI bus not valuable to tuning PID. Value measured on STM with frequency 100Hz should be contained in STNM logs. This data is useful in PID tuneing.
Strategy chain contain bellow strategies.

	Strategy	Data sent from STM to Raspberry	Data sent from Raspberry to STM
1.	Transmission key	Raspberry and STM key	Raspberry and Raspberry key
2.	Double	measure of rotation	% value to set on engines
3.	Battery	Voltage of every cell	Tuning reference voltage for every cell
6.	All data used	none data	none data

5.2.3 TuneMovePID mode.

This mode ID is 2. Main idea of this mode is to use rotation PID and collect data of forward/backward movement when constant value is set in engines. In this mode robot is able to keep constant angle and collect, send data about movement measure and battery cells.
Important!
Working frequency of PID (100Hz) and SPI transaction frequency (40Hz). That makes measured data sent by SPI bus not valuable to tuning PID. Value measured on STM with frequency 100Hz should be contained in STNM logs. This data is useful in PID tuning.
Strategy chain contain bellow strategies.

	Strategy	Data sent from STM to Raspberry	Data sent from Raspberry to STM
1.	Transmission key	Raspberry and STM key	Raspberry and Raspberry key
2.	Double	measure of movement	% value to set on engines
3.	Battery	Voltage of every cell	Tuning reference voltage for every cell
6.	All data used	none data	none data

Chapter 6

Raspberry Pi and stm I2C interfaces.

Every information which STM needs to work should be provided in EEprom memory. Those could be informations like: transmission keys, mode, PID params, amount of distance measurements, battery ADC parameters, amount of battery cells, etc. STM is allowed to read everything in memory but is not allowed to modify anything. Raspberry by GPIO pin has possibility to switch acces to I2C bus of EEprom memory. In this solution STM and Raspberry could be declared as master of I2C bus. HW is designed to block writing in EEprom memory when raspberry is not connected to PCB board. Raspberry should take care to always set writing protection of EEprom memory when I2C bus switch connect STM to memory. This solution guarantee STM will read exactly same data from EEprom as those set by Raspberry. STM have no possibility to check witch CPU is chosen in I2C switch, that means STM should be restarted after using EEprom by Raspberry. To keep flexible solution for providing configuration for STM based on configuration known for Raspberry, two thinks are required:

- Have one algorythm of parsing data to 8-bits EEprom format on STM and Raspberry. It has to be done by using unions which contain tabke of 8 elements 8-bit type and one variable of required type to/from convert.
- Keep straight to register map defined in below subsections.

EEprom memory has 256 pages of registers numbered from 0 to 0xFF.
Every page has 256 8-bit registers numbered from 0 to 0xFF. Full address of one 8-bit cell is 4 digit hexadecimal value from 0x0000 to 0xFFFF.

6.0.1 EEprom memory page 0x00.

address	data type	data
0x00 ⋮ 0x09	uin8 table table size: 10	EEprom key - Used for validation. It's list of 10 constant elements used to check are they correct read from EEprom. 0x15, 0x26, 0x37, 0x48, 0x59, 0x6a, 0x7b, 0x8c, 0x9d, 0xae
0x0A	uin8	SPI key compared by STM
0x0B	uin8	SPI key sent by STM
0x0C	uin8	Strategy chain mode. More information in section 5.2
0x0D ⋮ 0xFF		Not used

6.0.2 EEprom memory page 0x01.

address	data type	data
0x00 ⋮ 0x07	double	P of forward/backward PID Proportional factor of PID regulator controlling displacement movement.
0x08 ⋮ 0x0F	double	I of forward/backward PID Integral factor of PID regulator controlling displacement movement.
0x10 ⋮ 0x17	double	D of forward/backward PID Derivative factor of PID regulator controlling displacement movement.
0x18 ⋮ 0x1F	double	Anti wind-up of forward/backward PID Anti wind-up factor of PID regulator controlling displacement movement.
0x20 . . . 0x27	double	P of rotation PID
0x28 . . . 0x2F	double	I of rotation PID
0x30 . . . 0x37	double	D of rotation PID
0x38 . . . 0x3F	double	Anti wind-up of rotation PID
0x40 ⋮ 0xFF		Not used

6.0.3 EEprom memory page 0x02.

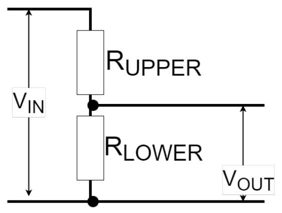

Figure 6.1: Voltage divider. Schemat of upper and lower resistors.

address	data type	data
0x03	uint8	Number of used battery cells.
0x04 ⋮ 0x0B	double	Max ADC voltage Maximum voltage which could be measured by ADC converter.
0x0C ⋮ 0x0F	uint32	Battery Adc max word Unsigned value which return ADC converter when measure max ADC voltage.
0x10 ⋮ 0x13	uint32	Upper resistor of cell 1 Resistance in ohm of upper resistor of voltage divider connected to battery cell and ADC input.
0x14 ⋮ 0x17	uint32	Lower resistor of cell 1 Resistance in ohm of lower resistor of voltage divider connected to battery cell and ADC input.
0x18 . . . 0x1B	uint32	Upper resistor of cell 2
0x1C . . . 0x1F	uint32	Lower resistor of cell 2
0x20 . . . 0x23	uint32	Upper resistor of cell 3
0x24 . . . 0x27	uint32	Lower resistor of cell 3
0x28 . . . 0x2B	uint32	Upper resistor of cell 4
0x2C . . . 0x2F	uint32	Lower resistor of cell 4
0x30 . . . 0x33	uint32	Upper resistor of cell 5
0x34 . . . 0x37	uint32	Lower resistor of cell 5
0x38 . . . 0x3B	uint32	Upper resistor of cell 6
0x3C . . . 0x3F	uint32	Lower resistor of cell 6
0x40 . . . 0x43	uint32	Upper resistor of cell 7
0x44 . . . 0x47	uint32	Lower resistor of cell 7
0x48 . . . 0x4B	uint32	Upper resistor of cell 8
0x4C . . . 0x4F	uint32	Lower resistor of cell 8
0x50 ⋮ 0xFF		Not used

Chapter 7

List of document versions.

Date	Version	Comment of changes
13th March 2021	v1.0.0	First version of documentation. Defined STM side code and interface between STM and Raspberry. Not clarified Raspberry side code. Only HW code is described. Logic part will be clarified in next major version when architecture of this part will be designed.