## 1. Overview

For the Mazebot to escape the maze it needs to be able to follow the line and determine line intersection types. These two goals are fulfilled by the line sensor. It is located at the bottom of the robot and is placed only a couple of millimeters away from the floor / surface (see Figure 1) to avoid interference from other infrared sources which can be present in a competition environment (this is out of our control).

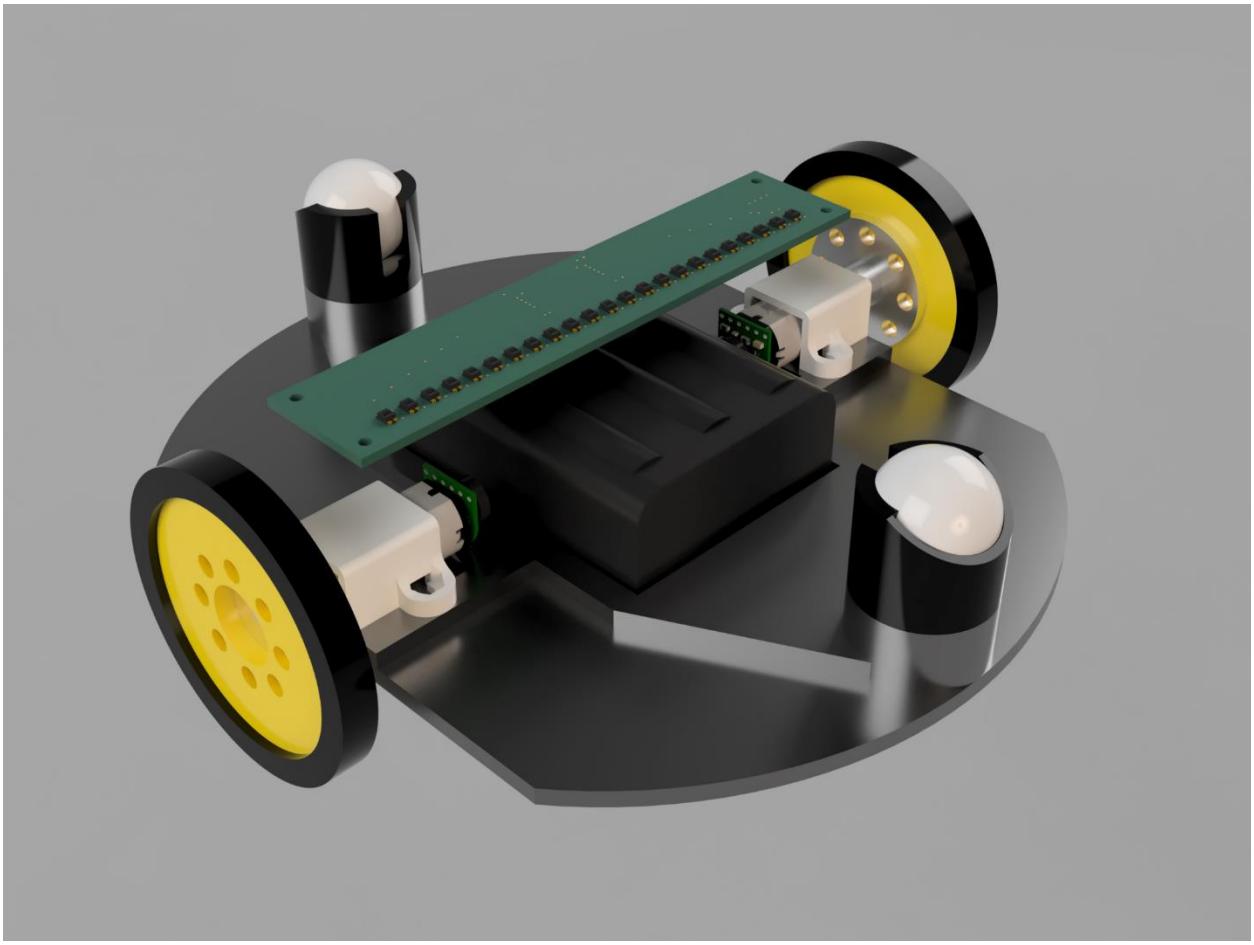


*Figure 1 Line sensor placement (bottom side of the robot). Sensor is in green.*

## 2. Requirements

Line sensor requirement:

- Sensor should power up and start sending data faster than robot can travel 1 millimeter:

$$time_{startup} \leq \frac{0.001}{v_{robot}}$$

- Sensor should be able to detect if it's stuck and reboot.
- Power provided to the sensor is from 6 AA batteries (~9 volts) or 3 18650 batteries (~11-13 volts).
- Frequency is hardcoded to 1.6 kHz.
- To send / receive data one of the following interfaces can be used: SPI, USART, $I^2C$. Interface is selected using hardware switch (requires reset after switch is changed). 2 interfaces are used in case client doesn't have enough hardware available.
- All connectors must have latches.
- Number of sensor units and their spacing can change from version to version. Client should change its code when swapping sensors which have different characteristics.
- Sensor should be designed with battery placement in mind. It shouldn't have any tall parts under batteries to allow for batteries to mounted close to the ground to keep center of gravity lower.
- Sensor board should have enough physical isolation to block any external infrared light which can interfere with sensor readings.

Sensor supports the following commands (each command is preceded with command prefix):

| **Command** | **Code (uint8_t)** | **Data** |
|---|---|---|
| Reset | 0x01 | N/A |
| Start calibration | 0x02 | N/A |
| Finish calibration | 0x03 | N/A |
| Use previous calibration data | 0x04 | uint16_t array of length NUMBER_OF_SENSORS which contains minimum value for each sensor unit |
| | | uint16_t array of length NUMBER_OF_SENSORS which contains maximum value for each sensor unit |
| Use hardcoded calibration data | 0x05 | N/A |
| Send latest sensor data | 0x06 | N/A |

Sensor command responses:

| **Command** | **Response data** | | |
|---|---|---|---|
| Reset | N/A | | |
| Start calibration | uint16_t prefix | | |
| | uint16_t status | | |
| | | 0x0000 | OK |
| | | * | Hardware/state failure |
| Finish calibration | uint16_t prefix | | |
| | uint16_t status | | |
| | | 0x0000 | OK |
| | | 0x0002 | Calibration failed (not enough difference between black and white on some / all sensor units) |
| | | * | Hardware/state failure |
| | uint16_t array of length NUMBER_OF_SENSORS which contains minimum value for each sensor unit | | |
| | uint16_t array of length NUMBER_OF_SENSORS which contains maximum value for each sensor unit | | |
| Use calibration data | uint16_t prefix | | |
| | uint16_t status | | |
| | | 0x0000 | OK |
| | | 0x0002 | Calibration failed (not enough difference between black and white on some / all sensor units) |
| | | * | Hardware/state failure |
| Use hardcoded calibration data | uint16_t prefix | | |
| | uint16_t status | | |
| | | 0x0000 | OK |
| | | * | Hardware/state failure |
| Send latest sensor data | uint16_t prefix | | |
| | uint16_t status | | |
| | | 0x0000 | OK |
| | | 0x0001 | Sensor data in the message was never transmitted before |
| | | 0x0002 | Not ready (no data was read) |
| | | 0x0004 | Sensor not calibrated (data will be in raw sensor format, which depends on sensor implementation) |
| | | * | Hardware/state failure |
| | uint16_t array of length NUMBER_OF_SENSORS in UQ1.15 format which contains value of the sensor unit | | |

To check current prefix value, see COMMAND_PREFIX and COMMAND_RESPONSE_PREFIX in line_sensor.h header file.

(*) Hardware / state failures:

| Error | Code | Description |
|---|---|---|
| Transmission / format error | `0x0008` | This error is reported if line sensor couldn't decode the message |
| Watchdog reset detected | `0x0010` | At some point in the past line sensor was reset by the watchdog |
| ADC DMA failure | `0x0020` | ADC DMA had a transfer error |
| Data buffer corrupted | `0x0040` | Data buffers got corrupted (software should do it's best to reset them to valid state) |

USART specific errors:

| Error | Code | Description |
|---|---|---|
| USART DMA failure | `0x0080` | USART DMA had a transfer error |
| USART noise error | `0x0100` | USART noise error was detected |
| USART framing error | `0x0200` | A framing error is detected when the stop bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.<br><br>For more see STM32L151xx reference manual |
| USART overrun error | `0x0400` | An overrun error occurs when a character is received when RXNE has not been reset. Data cannot be transferred from the shift register to the RDR register until the RXNE bit is cleared.<br><br>For more see STM32L151xx reference manual |

All line sensor receivers are located on robot's y-axis (for all sensors, x coordinate is equal to 0). For the case with 23 sensors, unit 11 has coordinates in robot frame of (0, 0), unit 1 has coordinates of (0, -0.044 m), unit 23 has coordinates of (0, 0.044 m). See Figure 2 for more information.

Line sensor with large number of units can be used to allow for more aggressive robot control system. Smaller spacing between units allows for higher resolution when estimating robot position (smaller uncertainty).

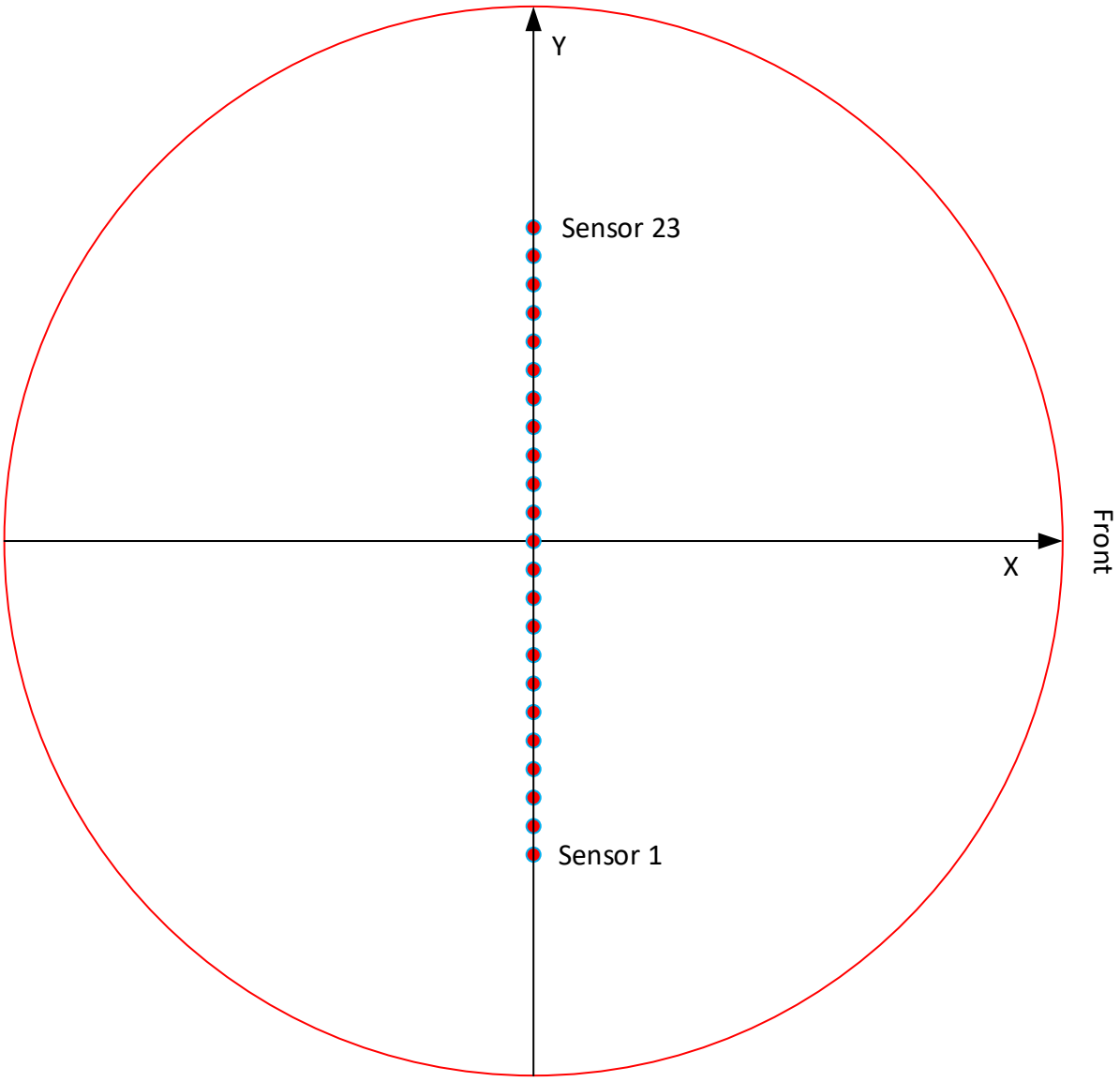



*Figure 2 Robot coordinate frame and sensors placement.*

When selecting number of sensors units, we need to keep the following in mind. When robot is moving along the line and meets an intersection, it must cross intersection completely before it can determine the type of an intersection (see Figure 3) and decide what to do next. If robot choses to continue moving straight ahead all it needs to do is to continue following the line. If robot choses to make a left / right turn or to execute a U-turn then things get more complicated. At this point we already crossed the intersection and we are still moving at top speed along the line. To execute the turn and not to lose track of the line robot has to make a sharp 90- or 180-degree turn (see Figure 4 for depiction of ideal world turn) which is hard as we have to overcome inertia of moving forward. Experiments shows that robot overshoots intersection by approximately 0.015-0.03 meters while moving at top speed (see Figure 5).

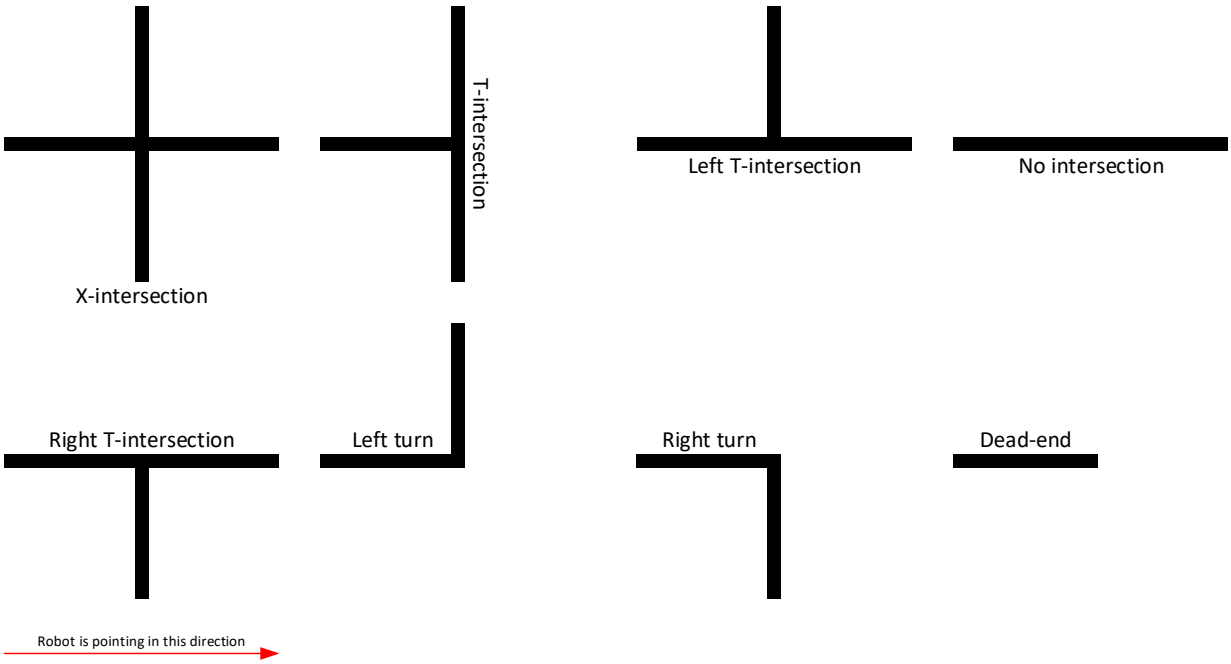



*Figure 3 Intersection types*

After executing the turn (which won't be exactly 90 or 180 degrees even though we use gyroscope) robot will be offset from the line in "y" direction by some amount, and this is where we can use extra sensors, as even if robot is 0.03 meters away from the line, we still have some leeway to bring it back. In general, the following factors contribute to the amount of overshoot:

- Sensors have lag of detecting change in reflected light
- ADC hardware will take some time to read changed sensor value
- MCU has to receive values from ADC
- Robot brain code running on MCU, runs at some frequency (1.6kHz in our case) and will take some time to process the information and issue commands to motors
- Motors will take some time to slow down (due to inertia, etc.)
- But the most important factor here is robot inertia. Robot will slide forward a bit (around 10-30mm) before it loses all the kinetic energy in that direction.

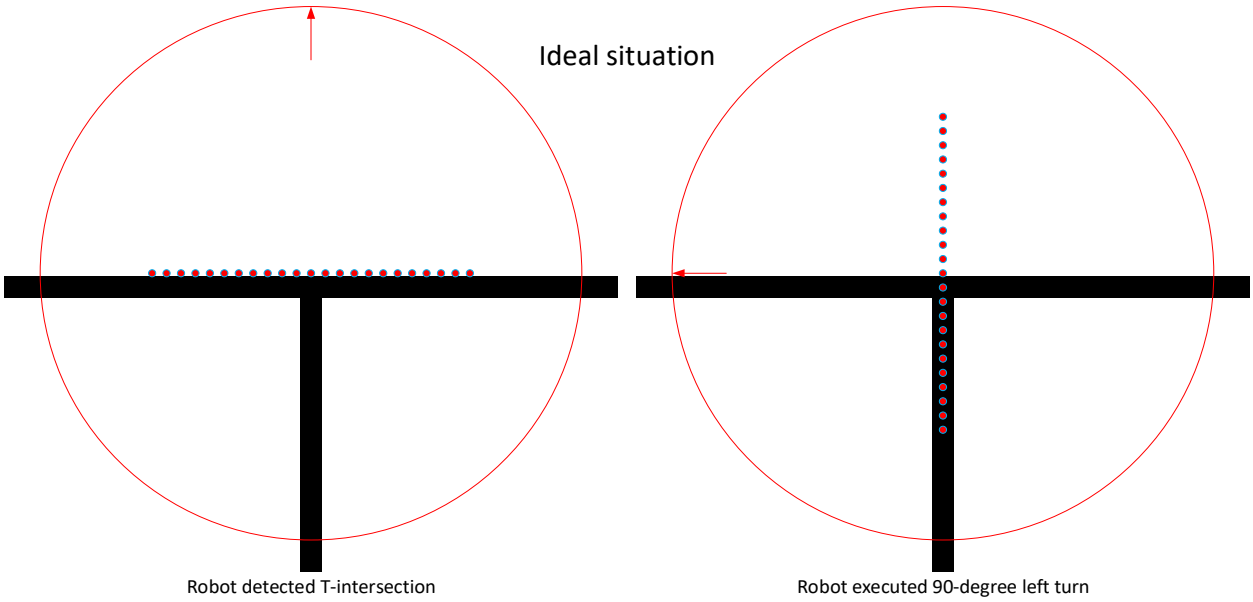



*Figure 4 Robot turn in an ideal world*

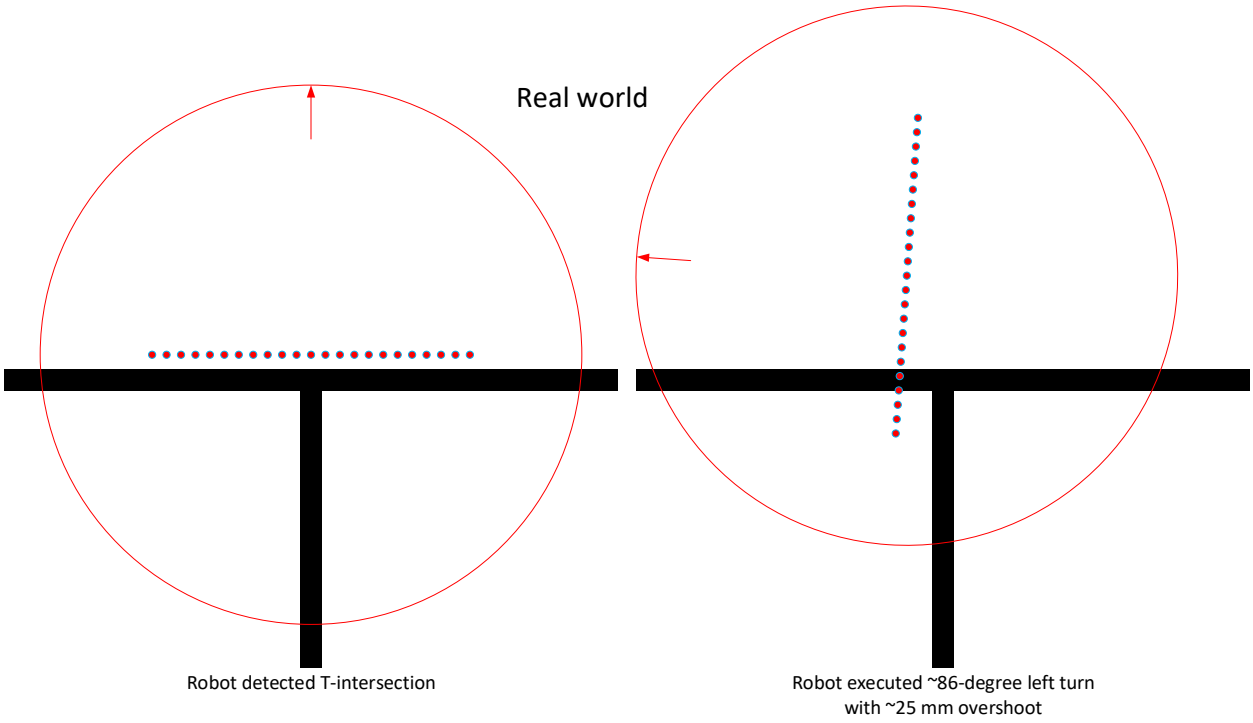



*Figure 5 Robot turn in real world*

## 3. Line sensor model

For now, we are going to talk about single line sensor unit let's say sensor 11 (one pair of infrared emitter and detector), and we are going to call it a sensor unit. Sensor unit can cross line in 2 directions "y" and "x". As always positive "x" axis points in the direction of robot movement, positive "y" axis points left. As robot moves along the line and only encounters perpendicular lines once it comes across the intersection, we are going to simplify out model and handle only "y" displacement of the sensor unit.

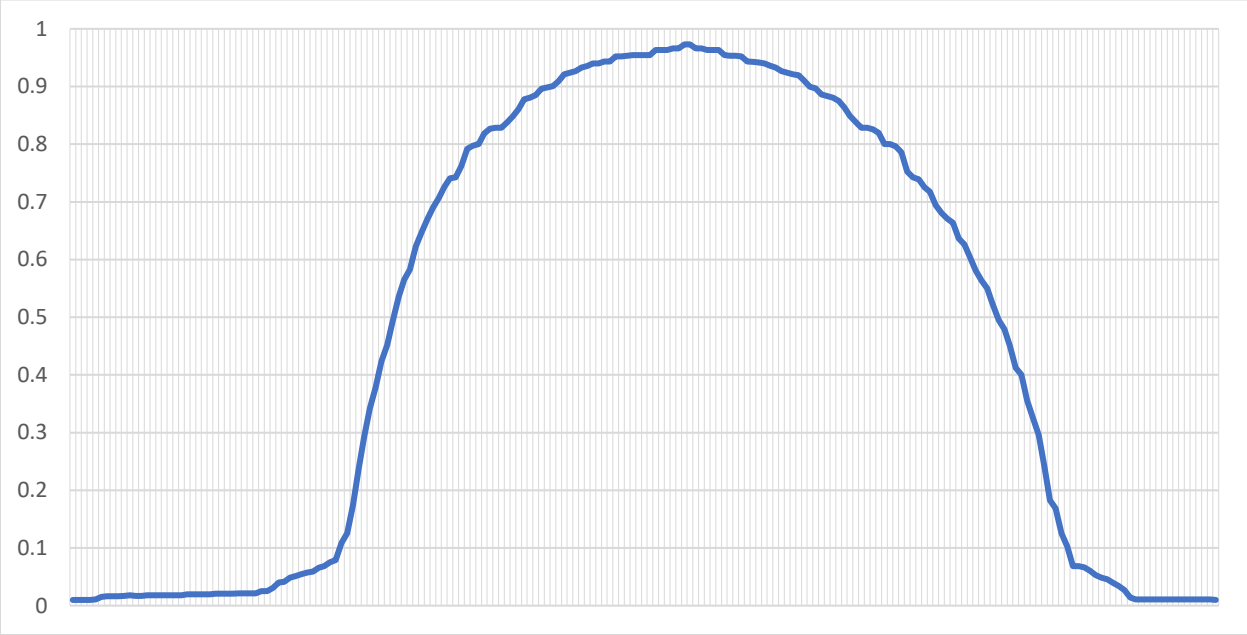



*Figure 6 Sensor output when crossing the line (non-canted sensor placement)*

Figure 6 shows sensor unit reading you are going to get if you position robot parallel to the line and then move robot across this line from left to right. As you can see reading looks approximately as a parabola, it is a bit skewed which is the result of emitter and detector not being on the same axis but being slightly displaced vertically (0.52 mm, see Figure 7). In "x" direction model looks similar but even more skewed as emitter and detector are displaced even more along x axis (1.4 mm).

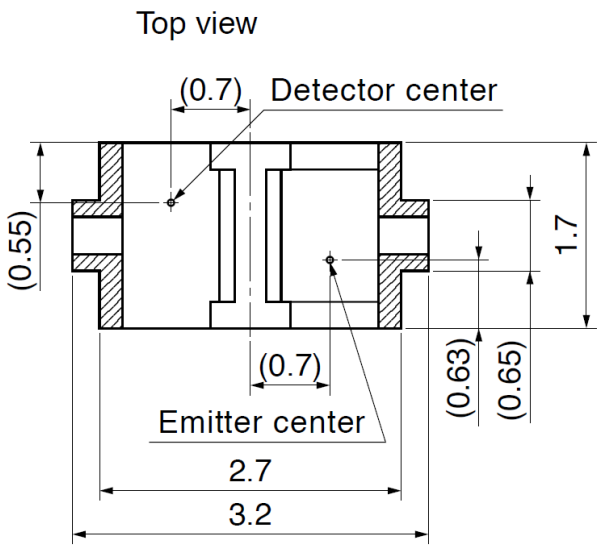



*Figure 7 Sensor unit top view (this is from Sharp GP2S60 specification).*

All sensors will be modeled using the following formula:

$sensor\_value(y) = a_n * y^2 + b_n * y + c_n$

Where $a_n$, $b_n$, $c_n$ are sensor unit specific coefficients. In this section we are going to assume that sensor_value is normalized to [0…1] and is filtered to remove some of the noise. Filter design and value clamping will be discussed later.

We will use Figure 8 in the upcoming discussion of the model. Sensor unit model has 2 regions. First one is blue region where we have values from 2 sensor units. In this region it is easy to pick a correct y value. If N-1 sensor unit value isn't zero then we use smaller value, if N+1 sensor unit value isn't zero we use larger value. We will handle cases where both neighbor sensor units have non-zero values later.

Second grey region is when we only have non-zero value from one sensor unit in this case there is no way to decide which y value to pick and we will simply use sensor unit center "y" position. Grey region can be reduced or even eliminated if we decrease distance between sensor units, but it will either reduce deviation or we will need more sensor units if we want to keep the same deviation value.

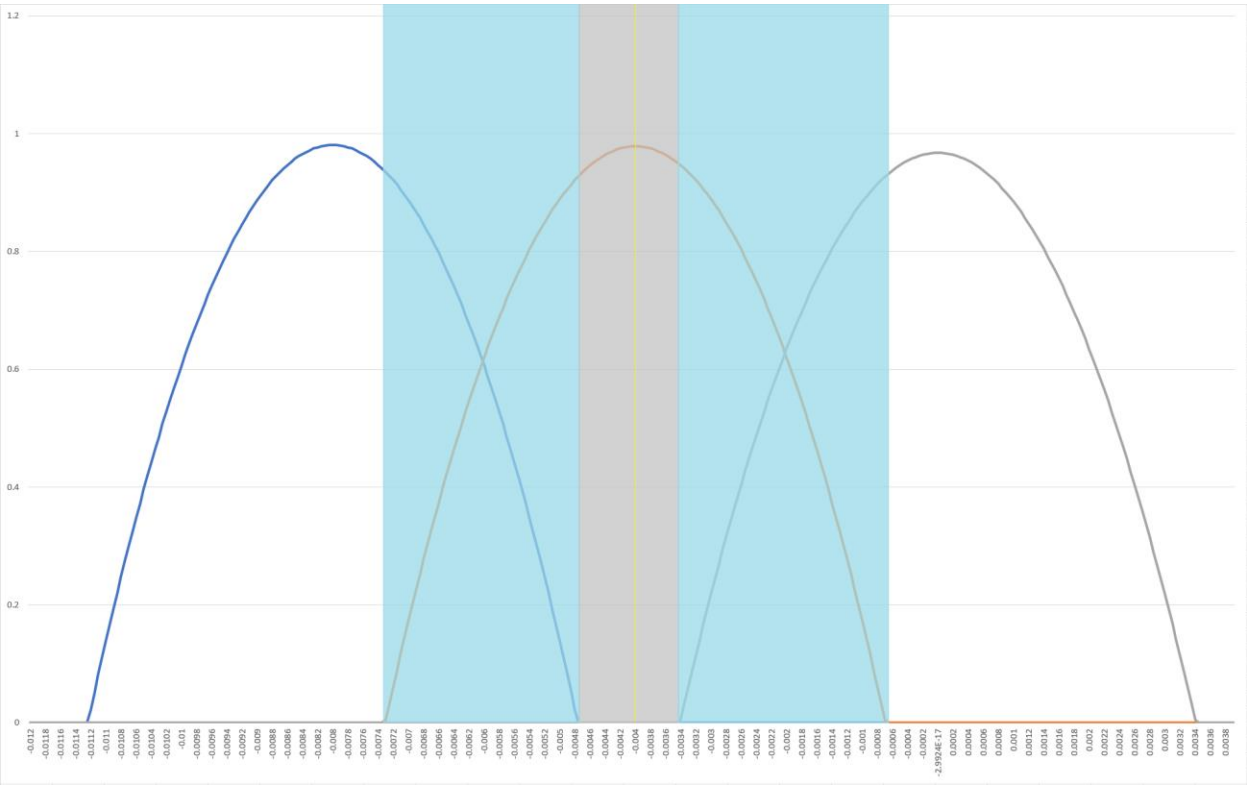


*Figure 8 Sensor unit model (with neighboring sensor units included)*

Now for the case where more than 2 sensors have non-zero values. In the following discussion group is one or more sensor units where all units in the group have non-zero values. There are 6 possible cases here (also see Figure 9).

- Case 1. All sensor units with non-zero values form a group in the middle
- Case 2. All sensor units with non-zero values form a group on the left
- Case 3. All sensor units with non-zero values form a group on the right
- Case 4. All sensor units with non-zero values form multiple groups
- Case 5. All sensor units have zero values (robot sensor is fully on white)

- Case 6. All sensor units have non-zero values (robot is either on the end of the maze circle or on the line perpendicular to the robot)

At the start of the run we know where robot is, and it is easy to figure out the “y” position of the group. For all future cases we need to figure out min and max y coordinate for each group. To do this we can use left and right sensor units in the group and use algorithm described above (for <= 2 non-zero sensor units). After this we can calculate probability of robot being in all those ranges given previous “y” position, yaw angle, odometry data, and gyroscope data. See Kalman filtering chapter for detailed design of the algorithm used for robot position and orientation estimation.

Figure 11 shows “y” position calculation (no Kalman filtering) using the model described above when robot is moving at a slight angle to the line and crosses it from left to right (see Figure 10). Horizontal sections are places where we couldn’t figure out which value of “y” to use as only 1 sensor has non-zero value.

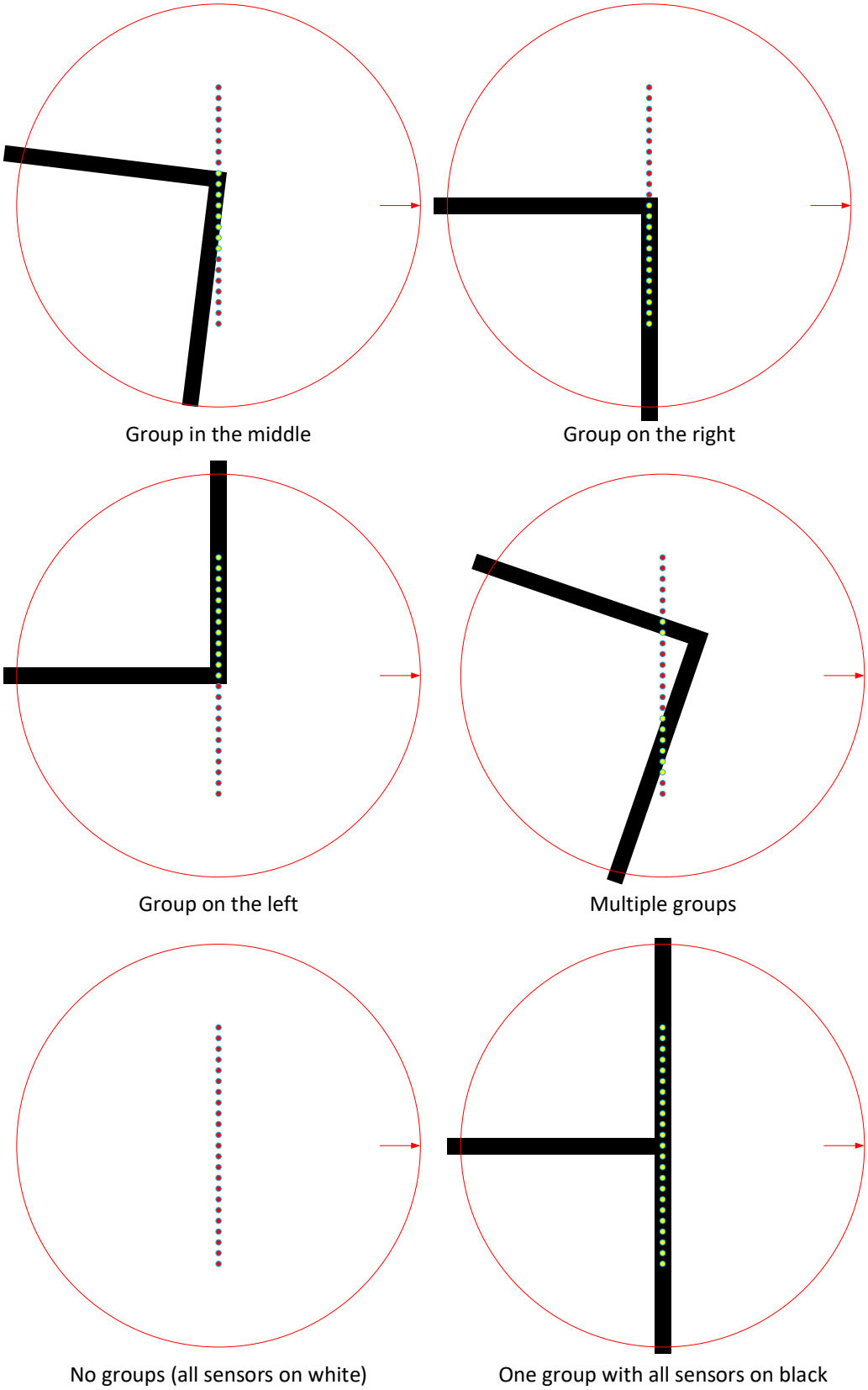



*Figure 9 Cases with more than two sensor units having non-zero values*

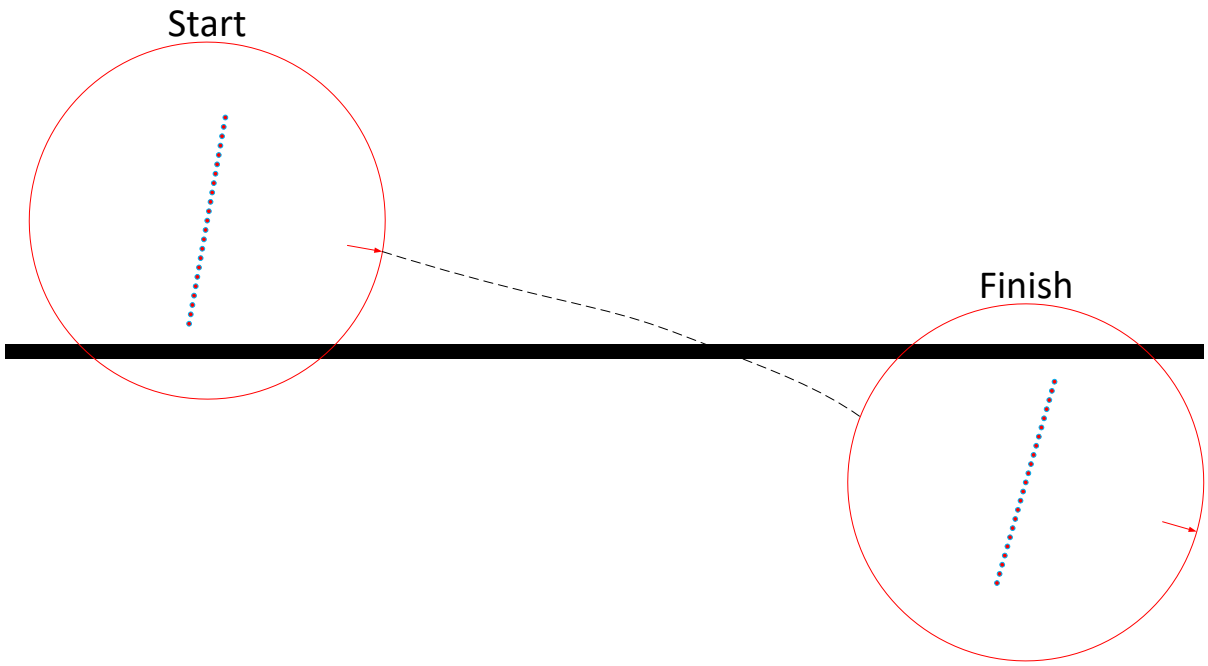



*Figure 10 Robot run for test "y" position estimation*

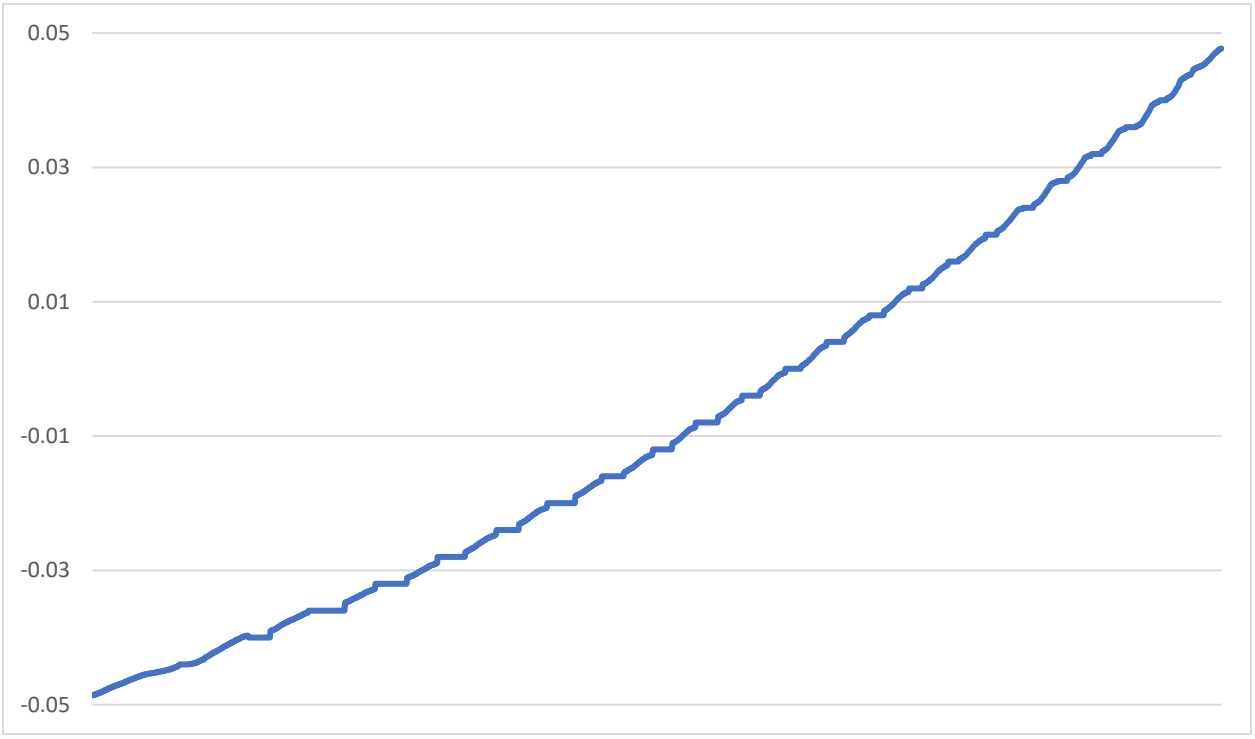



*Figure 11 Robot position estimation using sensor model*

## 4. Intersection type detection

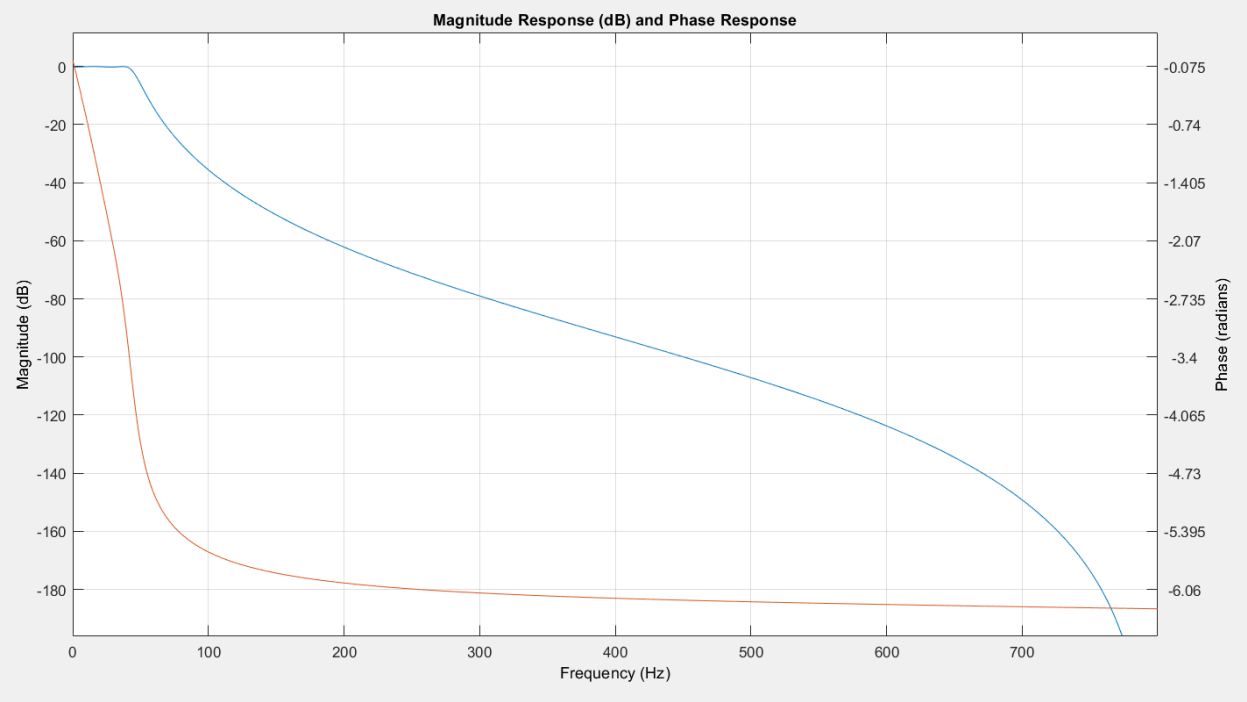
Intersection type detection is a higher-level function as robot can be moving at an angle to the feature we are trying to detect. It also helps if we know robot horizontal and angular speed. For more information see Kalman Filtering and Feature Detection chapters.

One possible solution is to have a 2D array of sensors or a camera. Problem with 2D array of sensors is that we need lots more sensors. Problem with the camera is that it needs to be away from the ground or have a very wide lens to reconstruct the intersection type, plus some even lighting which doesn't create glares and obstructs feature detection. In this case line sensor can be used for feature detection. In case having a 1D sensors ends up being too slow (reconstruction algorithm takes too much time) or hard to do (due to uncertainty it is hard to reconstruct the image) we might get back to evaluating use of 2D array of sensors or a camera. In case of adding intersection detection feature, sensor should still work with SPI/USART interfaces and might need to include additional data (detected feature and yaw angle).

## 5. ADC data filter design

For now, to filter out noise in ADC data, we are going to use IIR filter of 4th order, passband frequency is 40 Hz, passband ripple is 0.2. Later I need to move to FIR as they are easier to use if one needs zero phase at all / most frequencies.

Filter design can be found in mazebot\modelling\line-sensor\create_model.m.




As sensor data is going to start with values like 250 then after applying filter, some values at the beginning of the data range are going to be out of valid range. To prevent this, we are going to ignore samples at the beginning of the run or data set until the filter settling time. Alternatively, for the given data set we can duplicate 1st row enough times so filter stabilizes after the step input and then discard this data, this way we don't lose any of the data set information.

## 6. Model parameters calculation

Model parameter calculation is done using 2 separate pieces of software:

- Robot model data acquisition firmware which is used to collect data which will be processed later (mazebot\software\sensors-and-motors\main-modelling-line-sensor). Data is saved to SD card in modelling-line-sensor.dat file.
- MATLAB code which computes and visualizes robot model parameters (mazebot\modelling\line-sensor)

To get model parameters follow the following algorithm:

1. Place robot in parallel to the line. Press Start/Pause button and then move robot across the line left or right once. Then press Start/Pause button.
2. Now you can do either of the following two options:
    a. Wait until SD card isn't busy (light is off) and remove it. You are done with collecting the data. Go to step 3.
    b. You can move robot to a different line or rotate it and go to step 1.
3. Download data to your PC and run MATLAB code to get model parameters.

See code for full documentation. Here we are going to outline the high-level algorithm.

- Apply ADC data filter to the data.
- Now we need to find min and max black values. To split between white and black data, as a first step we mark every value as black if it is equal or larger than the split value. In our case:

$$splitValue = min + \frac{max - min}{2}$$

- Then we mark neighboring sensor units' values as black (first and last sensor unit are a special case here and we need to add some special code to handle them, see fix_column_filter MATLAB function). Using this procedure, we partition data into 2 disjoint data sets: black and white. Then to get minimum black value we take maximum of white data set for each sensor unit and to get maximum black value we get maximum of black data set.
- Next step is to normalize the data, so it is in range [0…1] using the following formula:

$$value_{normalized} = \frac{value_{raw} - min_{black}}{max_{black} - min_{black}}$$

- Now for each sensor unit we fit parabola for the given sensor unit data. If we had multiple data sets, we merge parabola parameters. We pick parabola with the largest sensor unit value (top of the parabola) and we average out min "y" and max "y" values. **One open question here is how to find good start / end of parabola (as we will have some near zero values on both ends).**

## 7. Calibration

During calibration we need to place robot somewhere in a maze, so that the robot is parallel with a line in a maze. Wait until ready to calibrate light is on. Press Start/Pause button. Calibration light will go on. Then move robot across the line left and right until calibration is done (robot front/back line should be in parallel with the line and you should make sure that all sensors are exposed to black and white at some point). Calibration is finished when calibration light goes off.

If calibration cannot be done due to min/max values not being far apart, error will be displayed 5 seconds after calibration has started, but you can still move robot around to finish calibration. If calibration is finished robot will stop displaying the error and will signal end of calibration. If calibration cannot be finished, you will need to diagnose the problem (calibration result is saved to SD card and can be used during investigation).

We find min and max values for black using the same algorithm as in chapter 6.

When robot is running, we set sensor unit value to 0 if unit's value is less than min black value and set unit's value to 1 if value is larger than max black.

## 8. Line sensor hardware overview

Line sensor unit output voltage depends on:

- If sensor unit is over all black, all white or over a thin black line. Reading over all black will have maximum value as almost no light is reflected to the sensor. Reading over all white will have the smallest value as almost all light is reflected to the sensor. Reading over thin black line will have value lower than reading over all black due to parasitic light reflected from neighboring sensors (which are over white). See Figure 13.
- Input voltage. Higher voltage improves all black and all white readings. Higher voltage improves reading over thin black line up to some voltage after which parasitic light might become so bright that output voltage might start decreasing. See Figure 14.
- Value of resistor R1 (resistor R2 depends on value of resistor R1). For schematics see Figure 12 for data see Figure 15. This is a current limiting resistor for photodiode. Larger value of R1 has the following pros/cons:
    - PRO limits amount of parasitic light which reflects to neighboring sensor units which reduces range between all white and thin black line
    - PRO reduces power consumption
    - CON if value of R1 is too large sensor unit might get too dim and reading over all white will be close or the same as over all black (see Figure 16)
    - CON sensed object must be close to the sensor. If value of R1 works for distance of 2 mm (see Figure 16) it might not work for distance of 4 mm (see Figure 17).

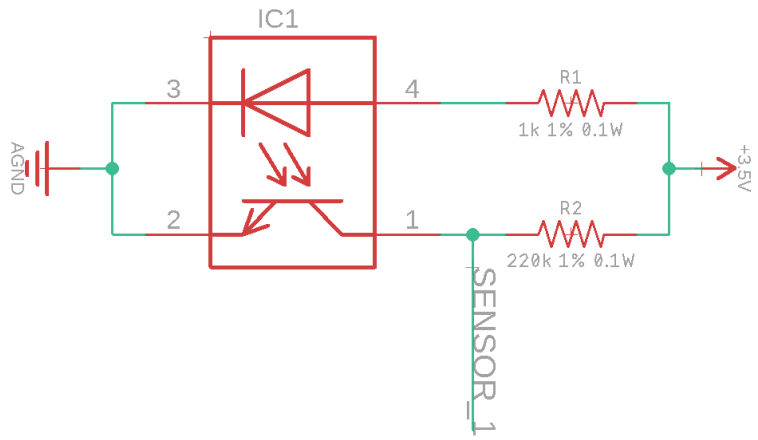



*Figure 12 Sensor unit schematic*

Line sensor schematics can be found at:

robot-corral\mazebot\hardware\line-sensor

Figure 12 shows schematic for the sensor unit. It is based on schematics from GP2S60 datasheet, http://www.pololu.com (lots of awesome boards which can be used in robot building and prototyping) and from the book Practical Electronics for Inventors by Paul Scherz and, Simon Monk.

For IC1 we use Sharp GP2S60 phototransistor output, compact reflective, photo interrupter.

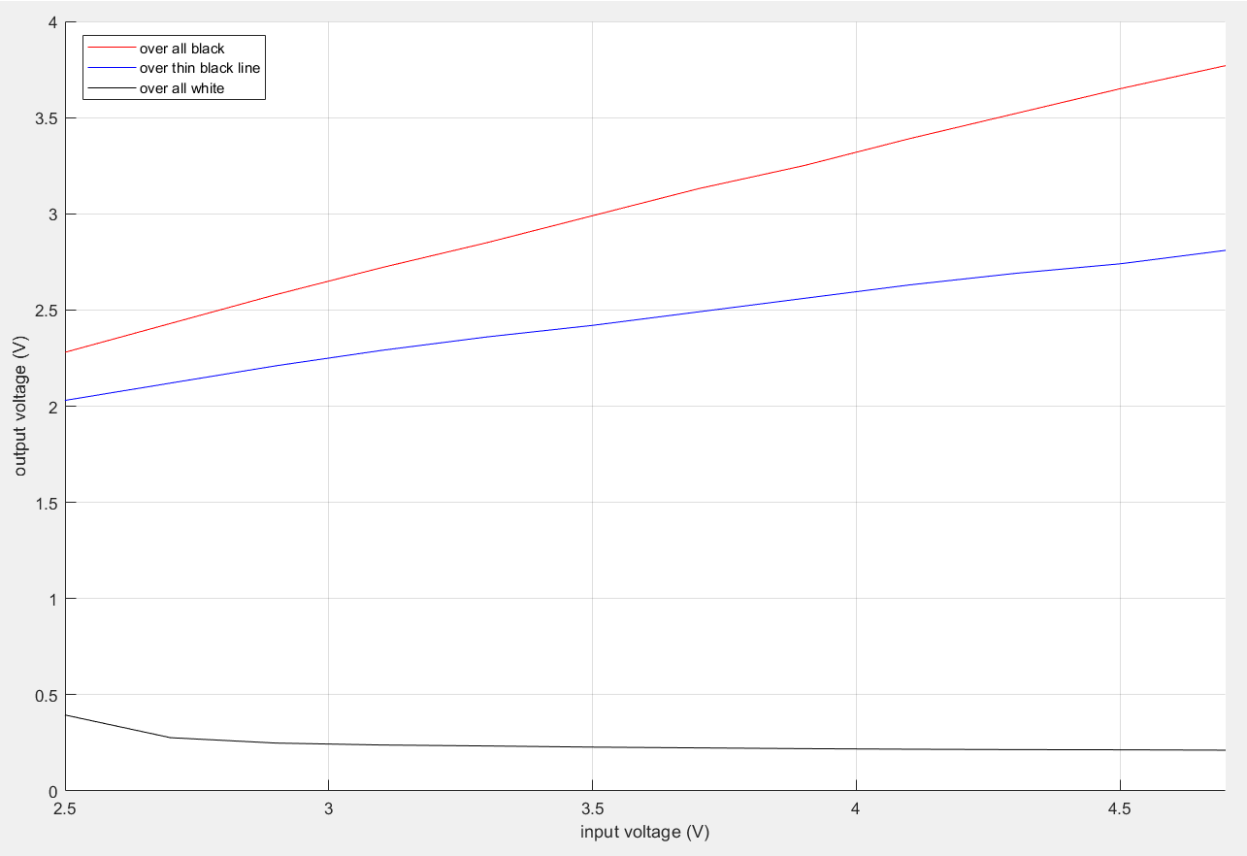



*Figure 13 Output voltage dependence on input voltage over different features. R1 = 1.5 kOhm, R2 = 220 kOhm, height = 2 mm.*

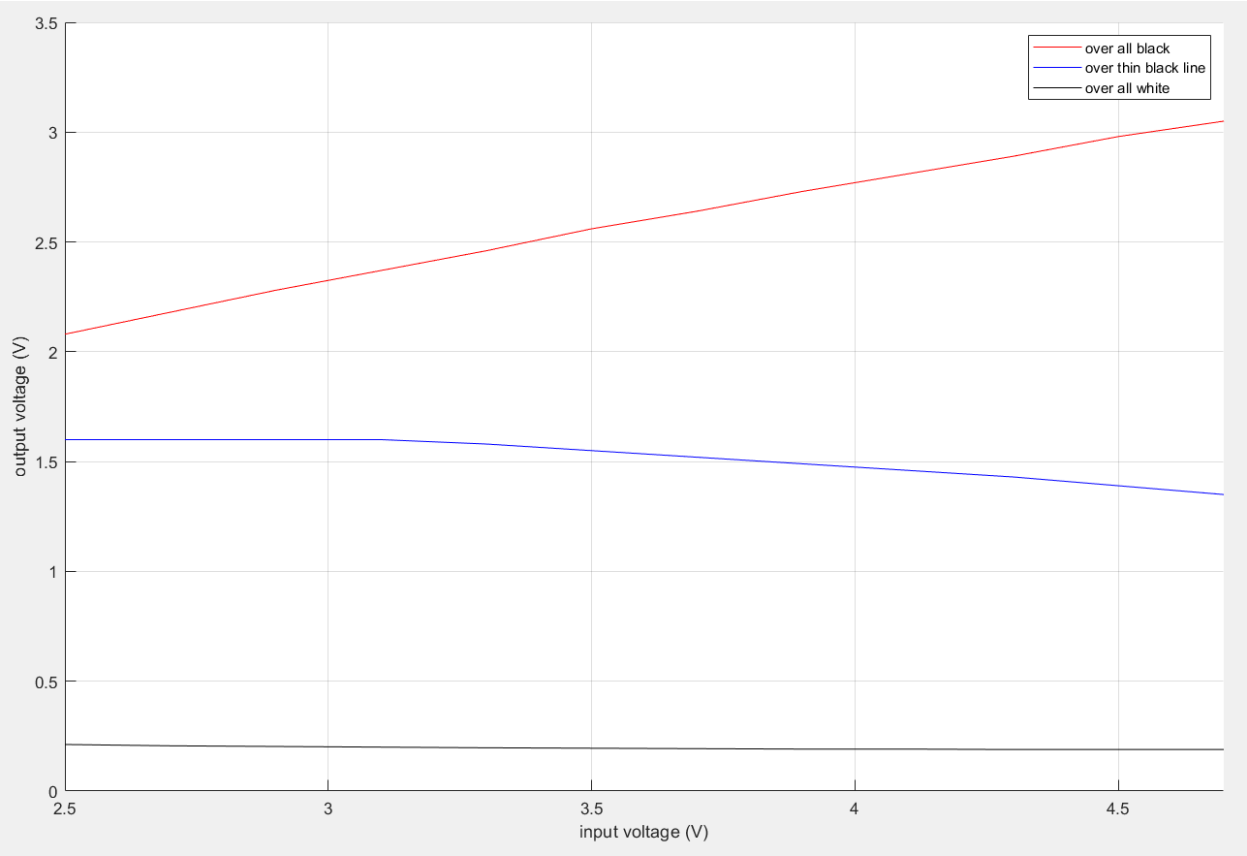



*Figure 14 Output voltage dependence on input voltage over different features. R1 = 1 kOhm, R2 = 220 kOhm, height = 2 mm.*

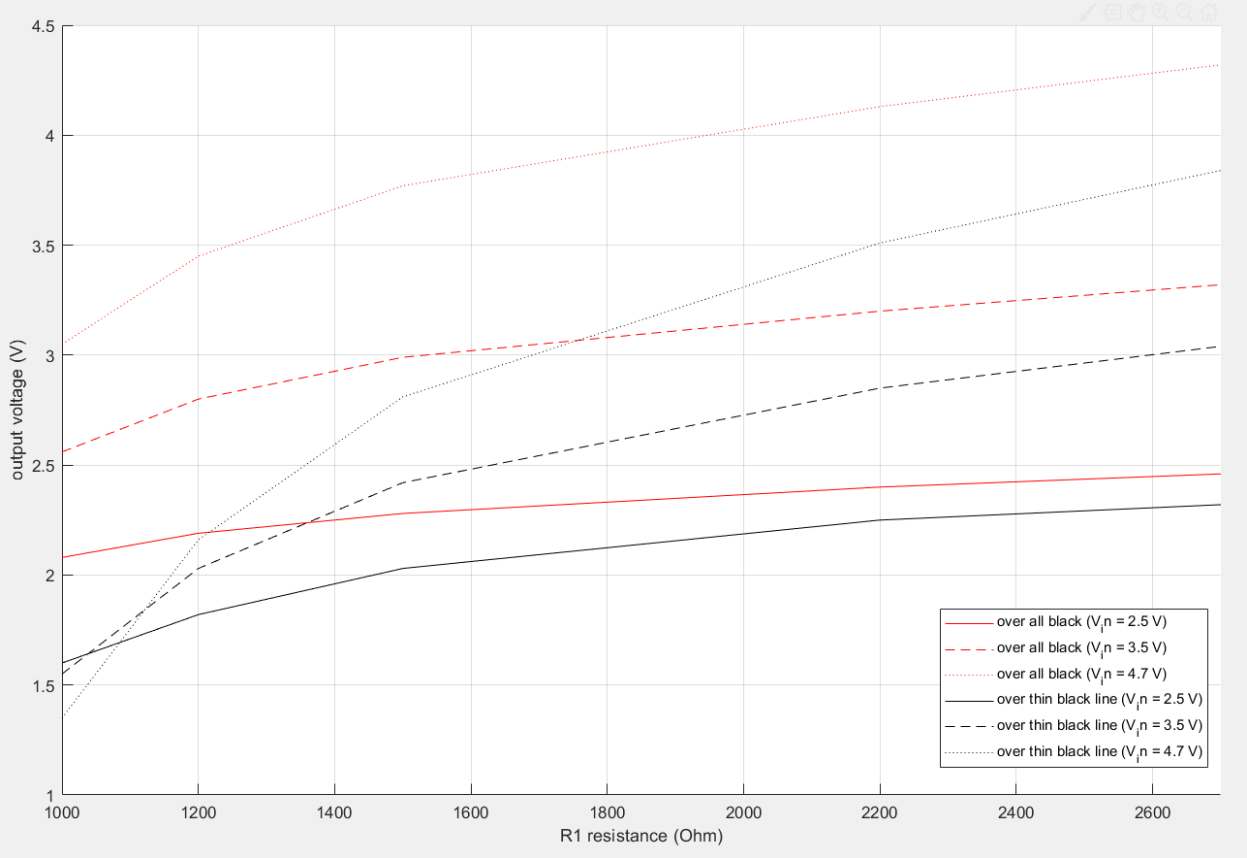



*Figure 15 Output voltage dependence on value of R1. R2 = 220 kOhm, height = 2 mm.*

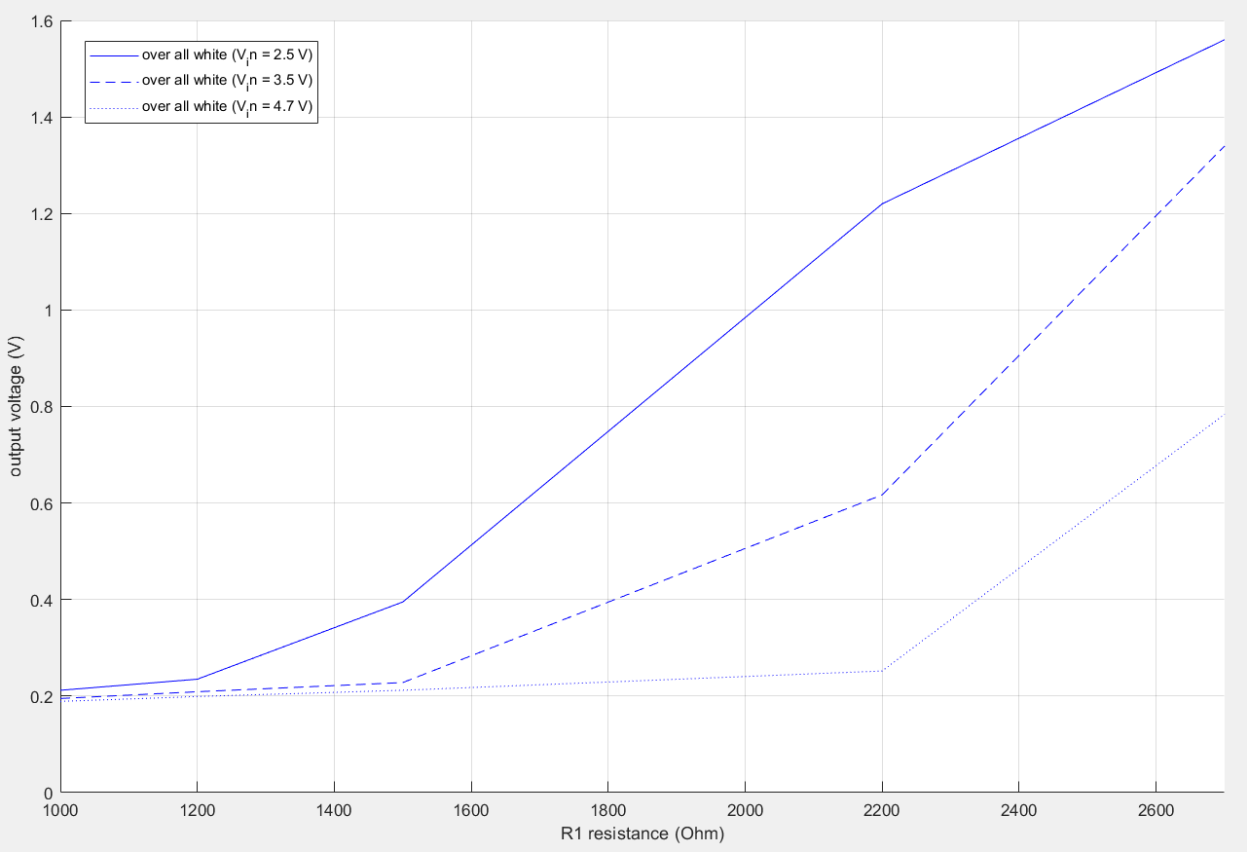



*Figure 16 Output voltage dependence on value of R1. R2 = 220 kOhm, height = 2 mm.*

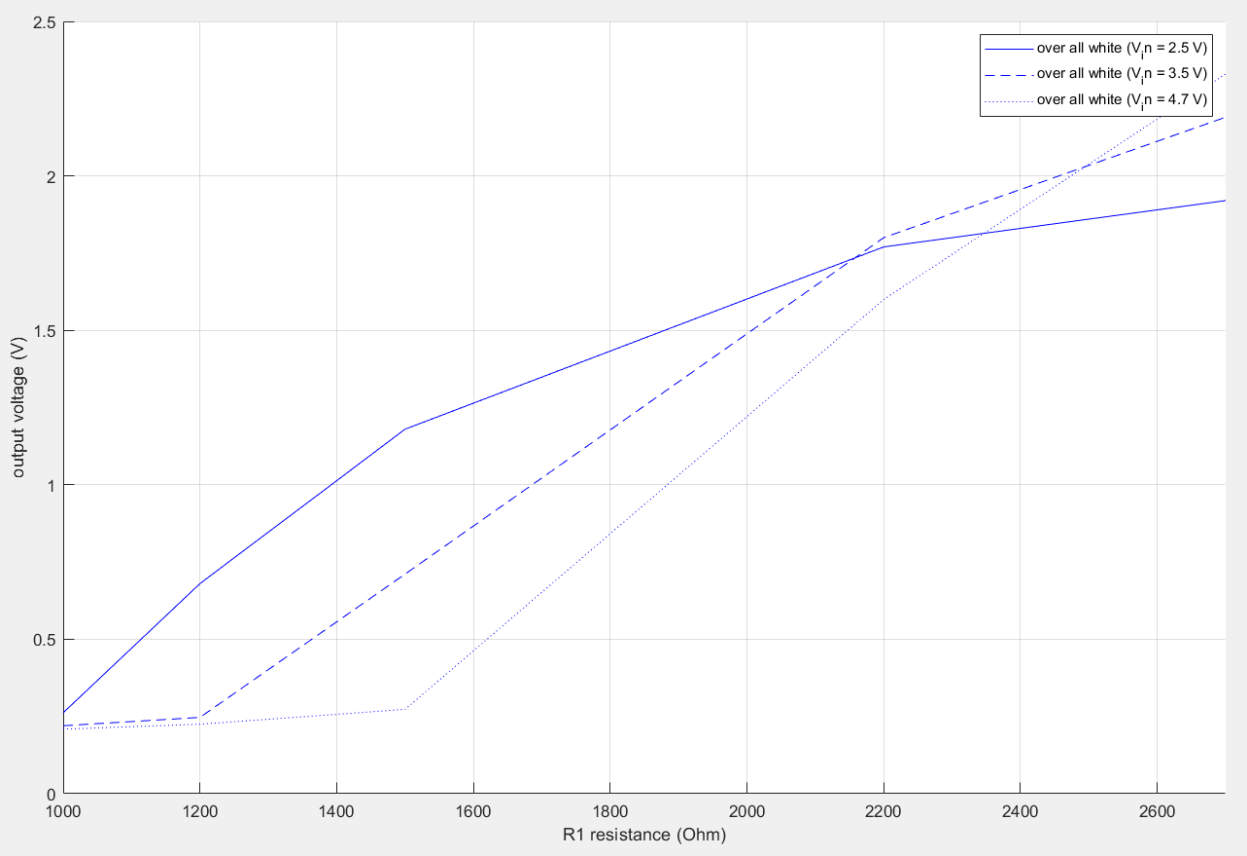



*Figure 17 Output voltage dependence on value of R1. R2 = 220 kOhm, height = 4 mm.*

## 9. Line sensor hardware design

For now, we use the following parameters for the sensor unit (still need to do an experiment on how changing resistance in multiple sensor units changes overall behavior):

Height = 2mm.

To pick design which has less noise overall, we are going to test the following configurations:

### (VAR1A) STM32L1 with one switching regulator for MCU and sensor units

Input and output voltage are 3.3V which is a constraint on design as we only have 1 voltage regulator and want to keep all MCUs at 3.3V logic level.

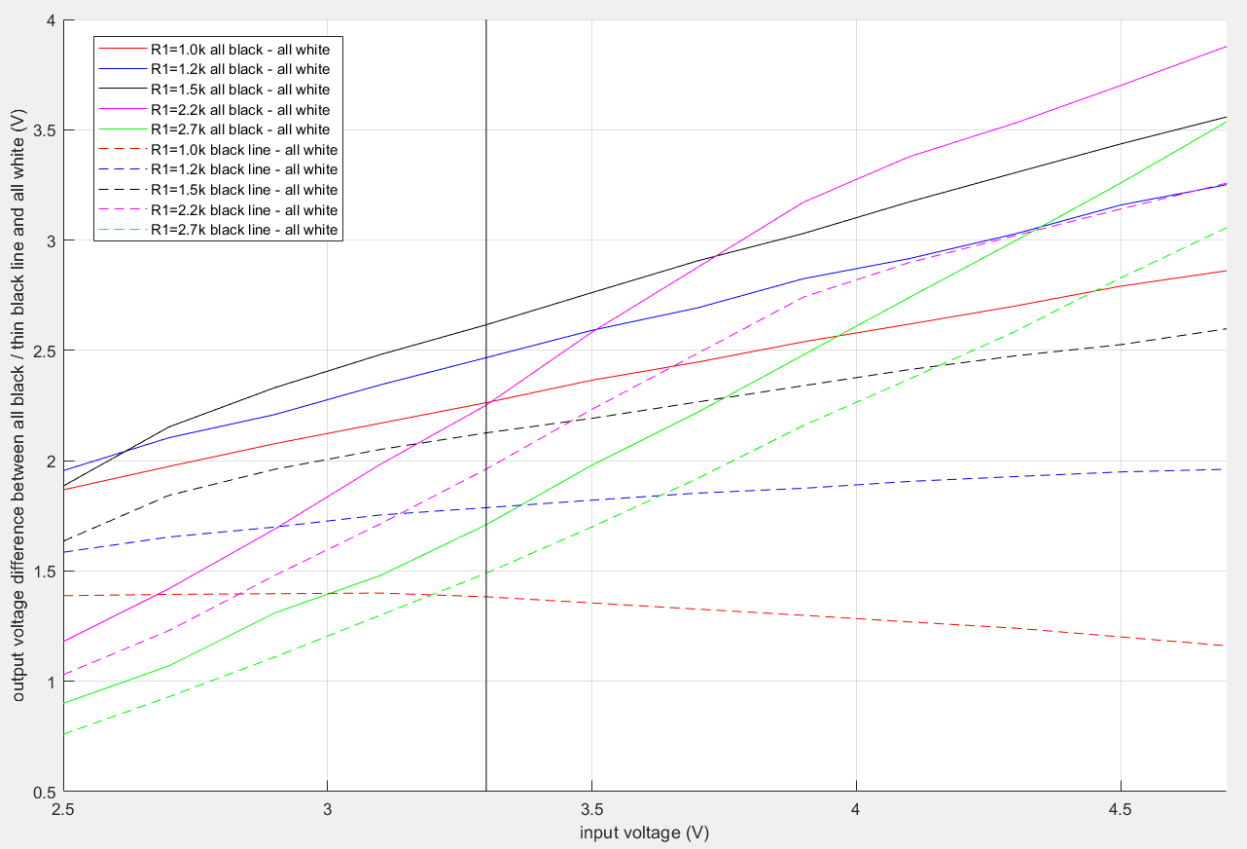



*Figure 18 VAR1A. Dependence of output voltage difference between all black / thin black line and all white on input voltage and value of resistor R1. R2 = 220 kOhm, height = 2 mm.*

Based on data in Figure 18 we pick R1 to be 1.5 kOhm. At input voltage of 3.3V it has highest resolution for both all black and thin black line over all other resistor values. Check that solid black line is over all other solid lines at 3.3V, as well as black dashed line is over all other dashed lines at the same input voltage. See Figure 18. We can play with values of R1 between 1.5 kOhm and 2.2 kOhm a bit more but to save time and to use standard resistor value we are going to stay with 1.5 kOhm.

See VAR1B and VAR2A design description for more information.

Max output voltage is ~2.8 volts. Maximum output voltage difference from all white is ~2.6 volts for all black and ~2.1 volts for think black line.

R1 = 1.5 kOhm. Tolerance can be +/- 1%.

R2 = 220 kOhm. Tolerance should be +/- 1% (this value depends on R1).

Tolerance of +/- 1% for R1 and R2 is needed to prevent drop in the range between all black and all white.

On Figure 19, Figure 20, Figure 21 you can see what ADC reads over different features. Reading over all white looks like gaussian, everything over black looks a bit random. Signal is quite clean with only less than +/- 1 % relative error. Apart from scaling analog signal to higher resolution by using higher input voltage there might be any point of adding linear regulators.

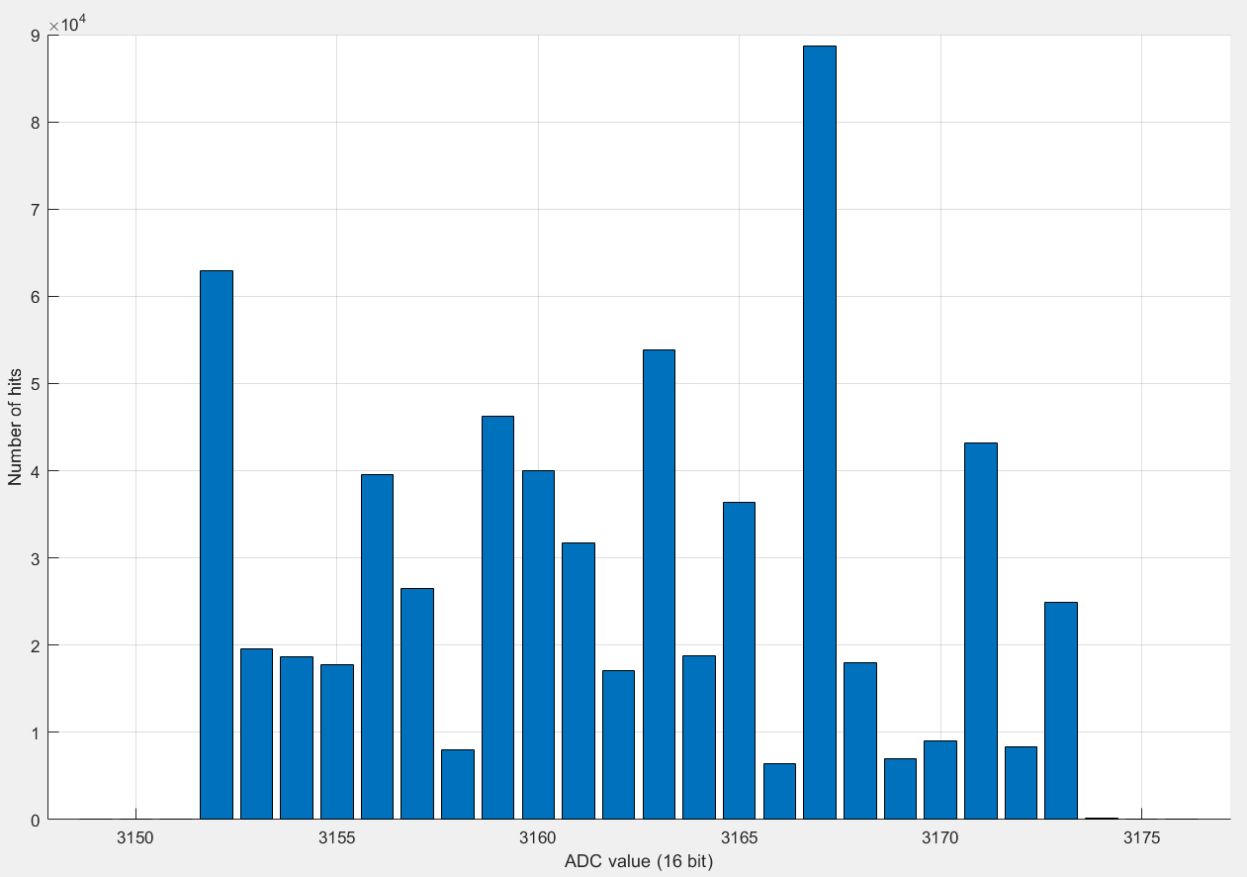



*Figure 19 VAR1A ADC read values for all black*

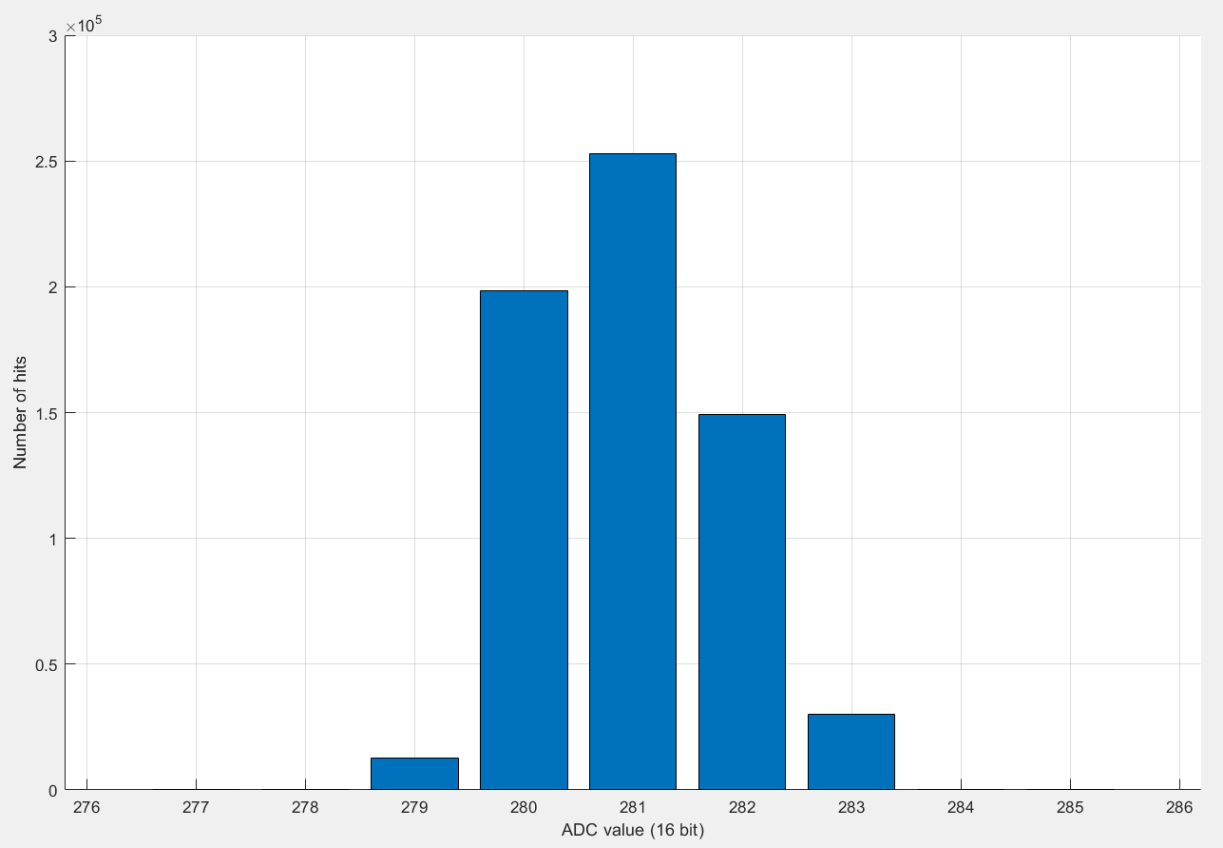



*Figure 20 VAR1A ADC read values for all white*

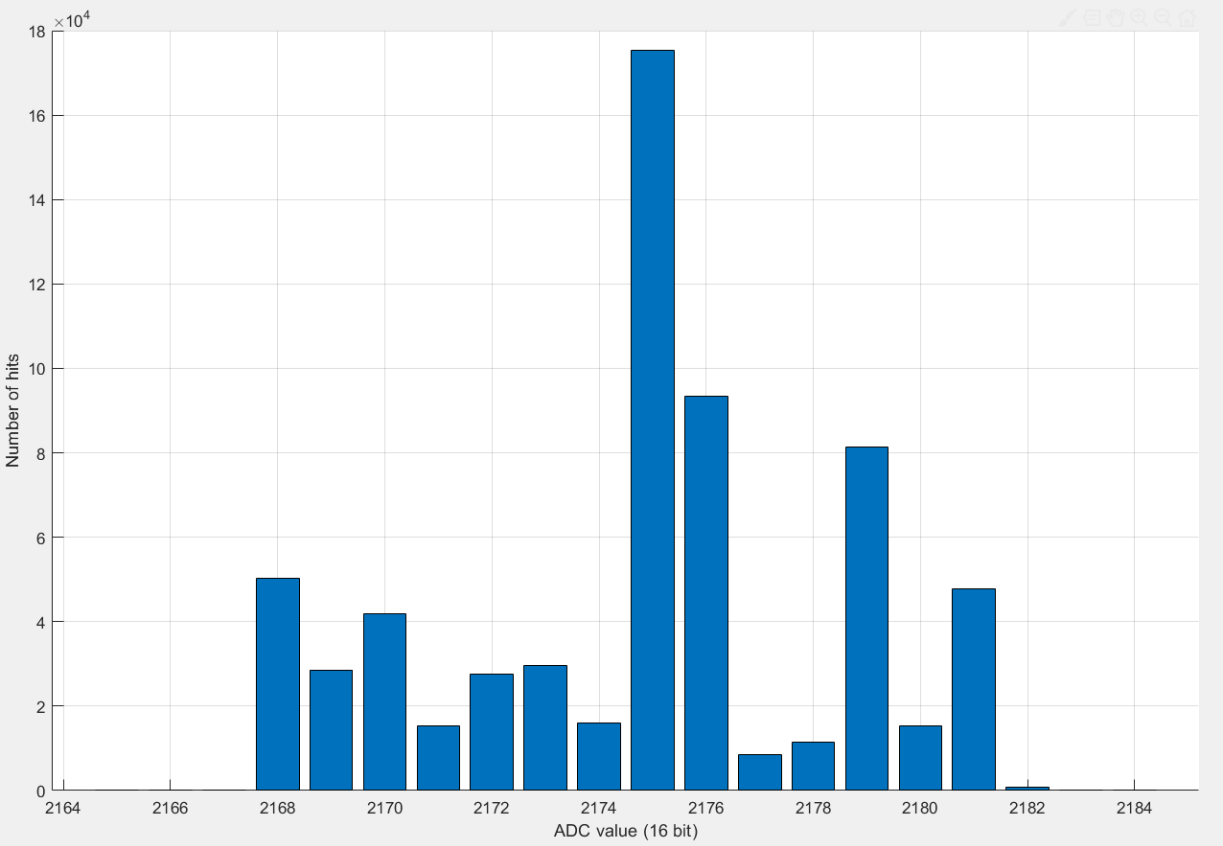



*Figure 21 VAR1A ADC read values for thin black line*

## VAR1B and VAR2A

- (VAR1B) STM32L1 with one switching regulator for MCU, stable voltage reference for MCU ADC, low pass filter for MCU ADC supply pin, and switching regulator / linear regulator pair for sensor units
- (VAR2A) STM32L4 with one switching regulator for MCU, stable voltage reference for MCU ADC, linear regulator for MCU ADC supply pin, linear regulator for the sensor units. In this case we will also need one switching regulator to provide voltage for both linear regulators

For VAR1B, VAR2A maximum output voltage is constrained to be <= 3.3 volts. This is because:

- VAR1B: VDDA must be the same voltage supply as VDD (3.3V) due to STM32L1 chip limitation
- VAR2A: VDDA must be less than or equal to VDD (3.3V)

In both variants VREF can be less than or equal to VDDA.

In VAR1B for high performance (16 MHz) VREF must be equal to VDDA. For medium performance (8 MHz) it must be >= 2.4 volts, for low performance (4 MHz) it must be >= 0.48 volts.

In VAR2A if VDDA is >= 2 volts then valid range for VREF is [2 volts, VDDA]. If VDDA is < 2 volts then VREF must be equal to VDDA. Minimum value for VDDA is 1.62 volts.

Based on data in Figure 22 and Figure 23 we can see that resolution is increasing as input voltage increases. To keep output voltage below 3.3V we have:

- R1 = 1.5 kOhm: maximum input voltage is around 3.9 volts
- R1 = 2.2 kOhm: maximum input voltage is around 3.5 volts
- R1 = 2.7 kOhm: maximum input voltage is around 3.4 volts
- We do not consider other values of R1 as they are inferior to the two cases above

For our design we want to keep output voltage above 2-2.5 volts as we are working in a noisy environment (presence of DC motors) and we want to keep noise level relatively low to the signal level.

We want to maximize difference between output voltage of a feature (all black or thin black line) and no feature (all white). This effectively increases resolution of data converted by ADC. It also decreases chance of noisy reading over no feature to be interpreted as a feature and vice versa.

We also want to avoid having large input voltage values as we are expecting ~4-14 volts from the battery.

It is nice to have smaller output voltage difference between all black and thin black line features. Sadly according to Figure 24 we couldn't do much about this goal without sacrificing higher importance goal above.

For both variants input voltage of 3.7 volts produces output voltage of 3.2 volts (with 0.1 volts safety as ADC input voltage must be less than VREF).

The best value of resistor R1 for input voltage of 3.7 is 1.5 kOhm. If we use 2.2 and 2.8 kOhm resistors then output voltage is higher than 3.3 volts, even though 2.2 kOhm resistor has better difference between thin black line and all white. 1.0 and 1.2 kOhm resistors have lower difference between all black / thin black line and all white. See Figure 22 and Figure 23.

Input voltage is 3.7 volts.

Output voltage is 3.3 volts.

R1 = 1.5 kOhm. Tolerance can be +/- 1%.

R2 = 220 kOhm. Tolerance should be +/- 1% (this value depends on R1).

Tolerance of +/- 1% for R1 and R2 is needed to prevent drop in the range between all black and all white.

To get all parts we set consumer voltages and currents and work our way back to battery voltage source. We need to keep voltage dropout in mind when selecting parts to keep battery minimum voltage as low as possible. See Figure 25 and Figure 26.

TODO: Add voltage regulator tolerances, safety margins and update voltages on all stages

TODO: Check cheaper 50mA 3.3 volts regulator for MCU or simply use 100 or 150 mA ones

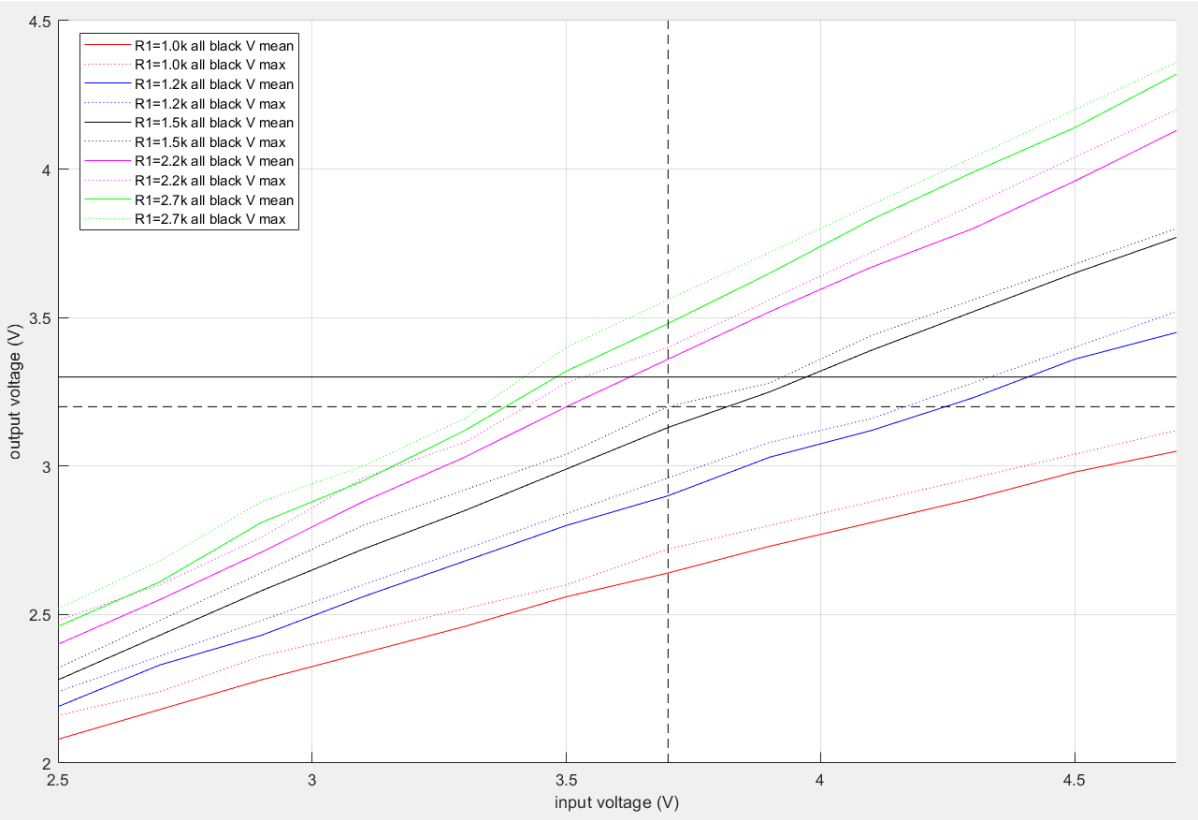



*Figure 22 VAR1B, VAR2A. Mean and maximum output voltage dependence on input voltage and value of resistor R1. R2 = 220 kOhm, height = 2 mm. Horizontal solid line is at output voltage of 3.3 V, horizontal dashed line is at output voltage of 3.2 V, vertical dashed line is at input voltage of 3.7 V.*

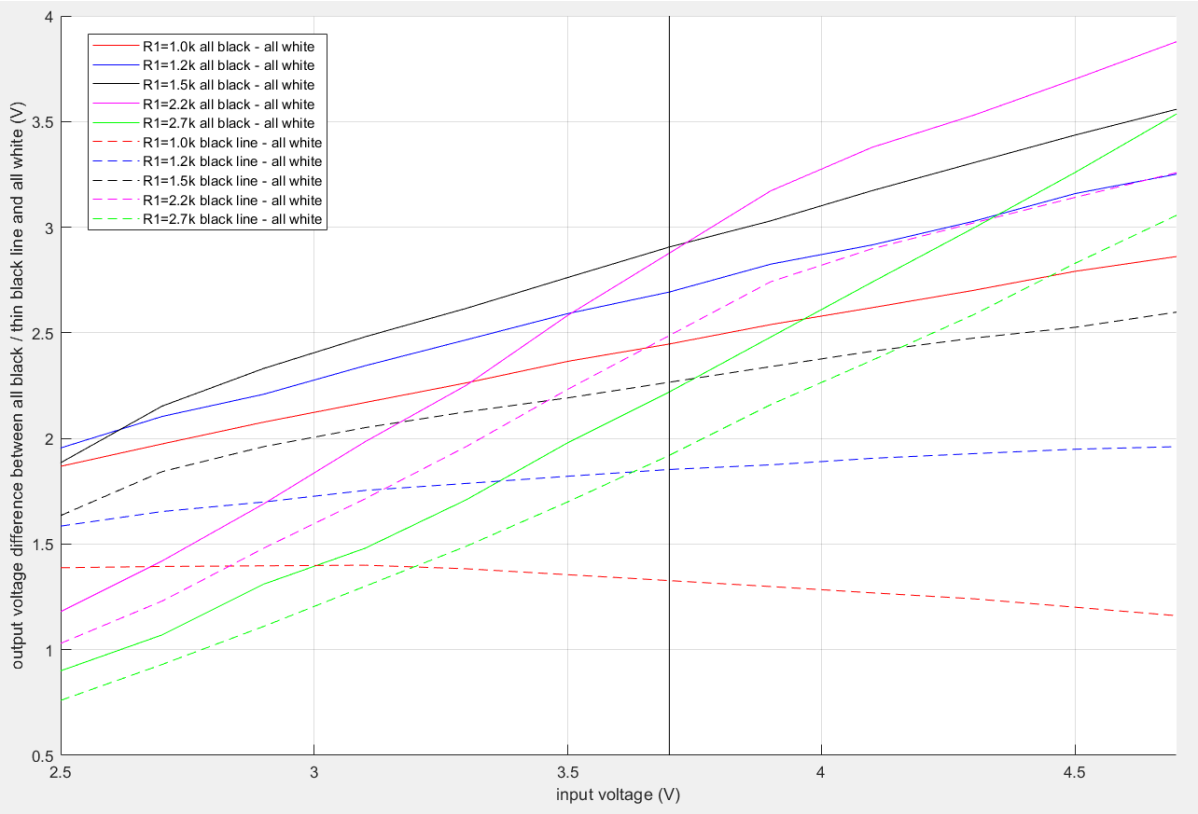



*Figure 23 VAR1B, VAR2A. Dependence of output voltage difference between all black / thin black line and all white on input voltage and value of resistor R1 R2 = 220 kOhm, height = 2 mm. Vertical line is at input voltage of 3.7 V.*

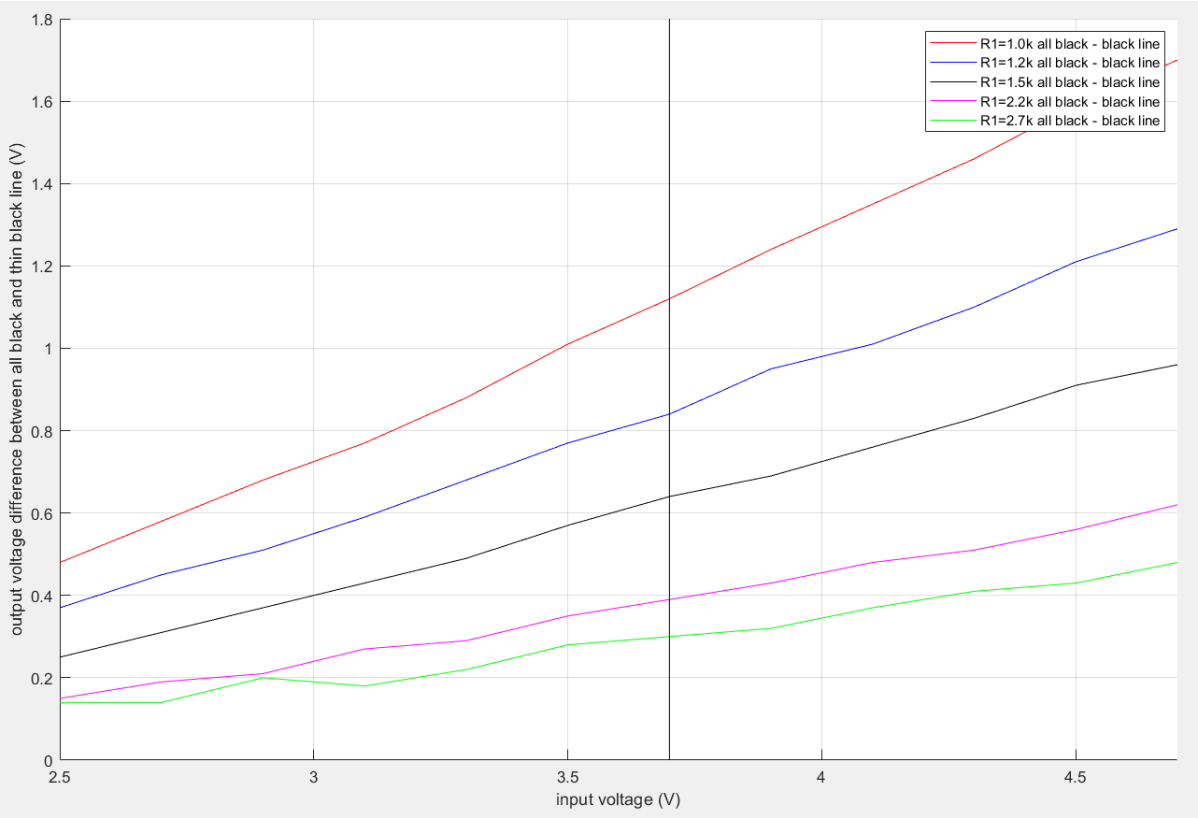



*Figure 24 VAR1B, VAR2A. Dependence of output voltage difference between all black and thin black line on input voltage and value of resistor R1. R2 = 220 kOhm, height = 2 mm. Vertical line is at input voltage of 3.7 V.*

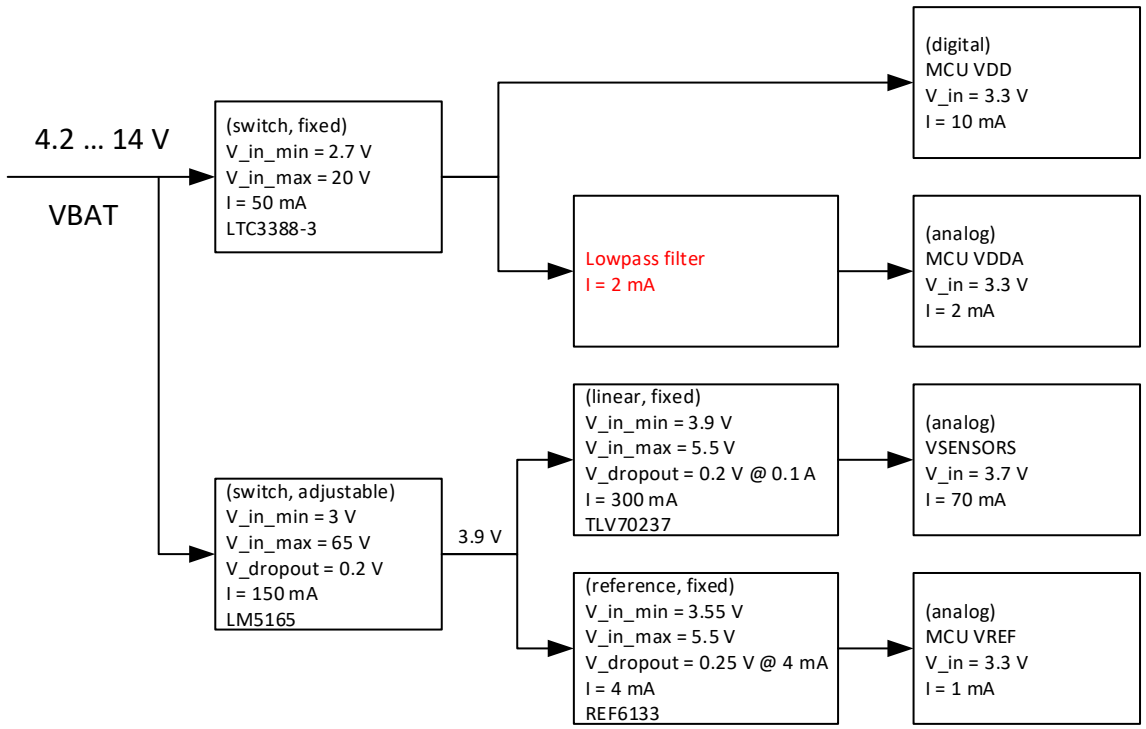



*Figure 25 Power supply scheme for VAR1B*

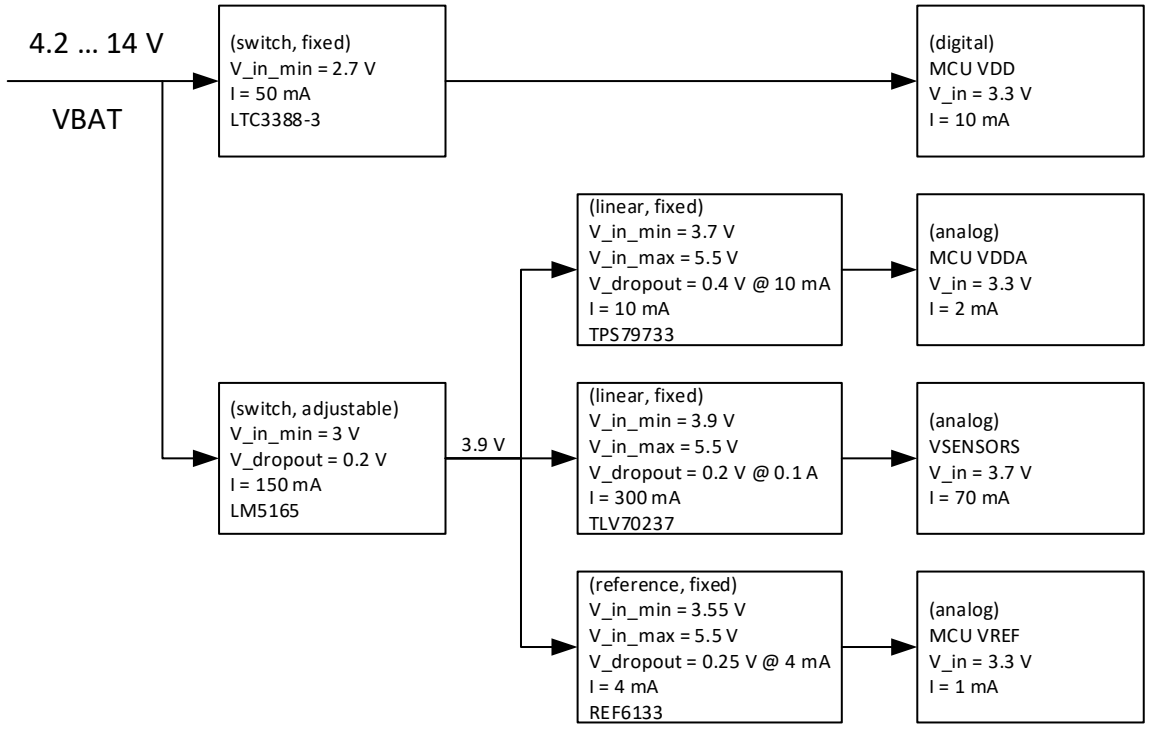



*Figure 26 Power supply scheme for VAR2A*

# 10. Firmware design

## [C] - Consumer

Consumer is the highest priority task. It is initiated by interrupts from USART/I2C/SPI DMA or from USART/I2C/SPI. No other interrupt has higher priority than this. This allows us to serve client requests as soon as possible and won't interfere with ADC much as ADC uses DMA. But all interrupts for [C] should be fast so as not to add too much delay for ADC handling.

Consumer doesn't need to ensure atomicity of any of it's accessed to buffer data.

Consumer always sends last read data [LR] if it is available. If it sends this record for the 1st time it will have new data flag, on subsequent sends of the same data this flag will be cleared.

If there is no last read data available, it should return NOT_READY error.

## [P] - Producer

Producer reads sensor values from ADC, normalizes them, and converts them to UQ1.15 format. It cannot overwrite last read data or data used by producer, apart from this any data buffer can be overwritten.

After read / conversion of new data is finished producer will set last read data pointer to this record.

Producer can get interrupted at any point so any access to shared data must be atomic.

We are going to use 3 buffers so one buffer can always be sent, one buffer can have latest read data, and last buffer can be receiving data from ADC. See Figure 27.

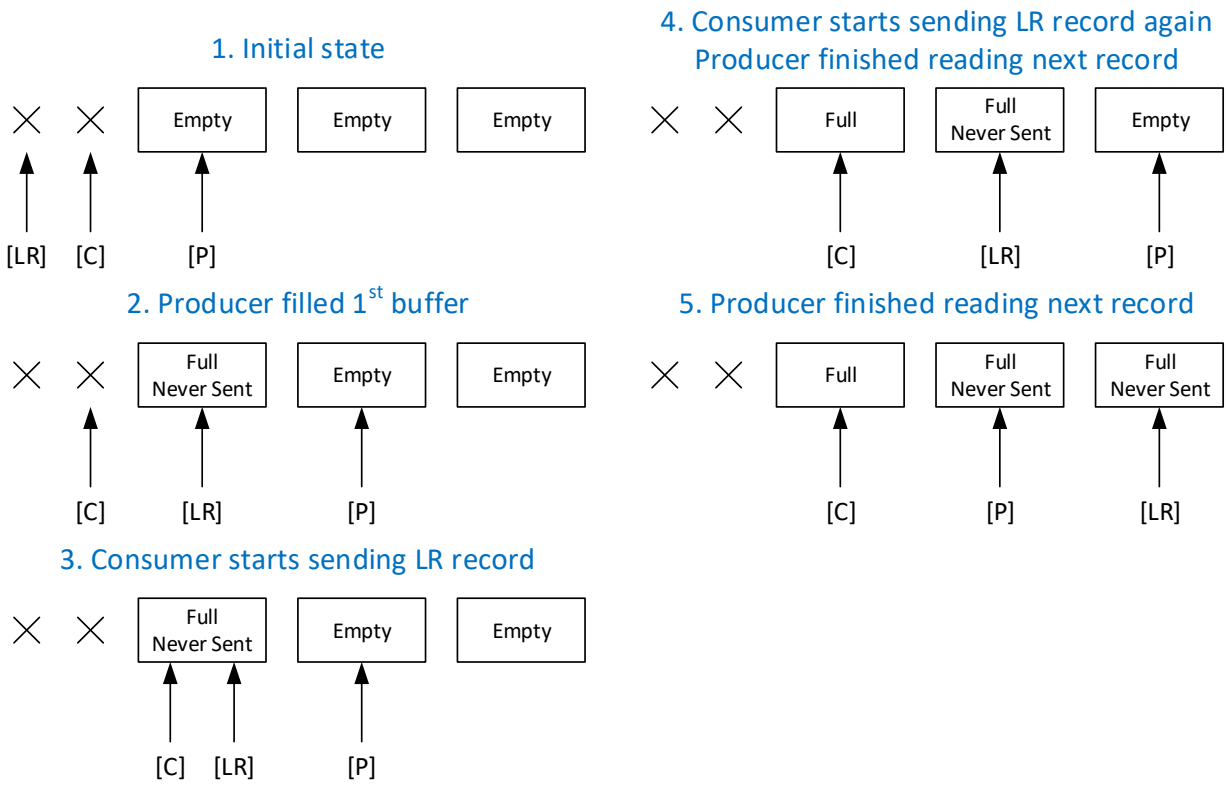



*Figure 27 Buffers usage by consumer [C], producer [P]. [LR] is a buffer which contains last read data.*

## Error handling

When any of the devices encounter an error, they set device specific global error flag to reflect this. Consumer when sending data can read those flags and notify client that we have some errors.

If device recovers from the error, it should reset device specific global error flag to reflect this.

See watchdog section for additional details.

## Watchdog

Watchdog will reset device if ADC failed to get data during last ((6 * 4) / 37000) = 0.648649 milliseconds. Client will now that device was reset as calibration data will be lost.

# 11. Communication

## USART

Client should send command, wait for the result, then send another command.

The only exception is reset command which can be sent at any point (though it must be a separate command packet).

USART speed is defined in line sensor API header.

To prevent invalid data from being sent we are going to use a 2-byte header and CRC code for the entire message at the end of it.