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).

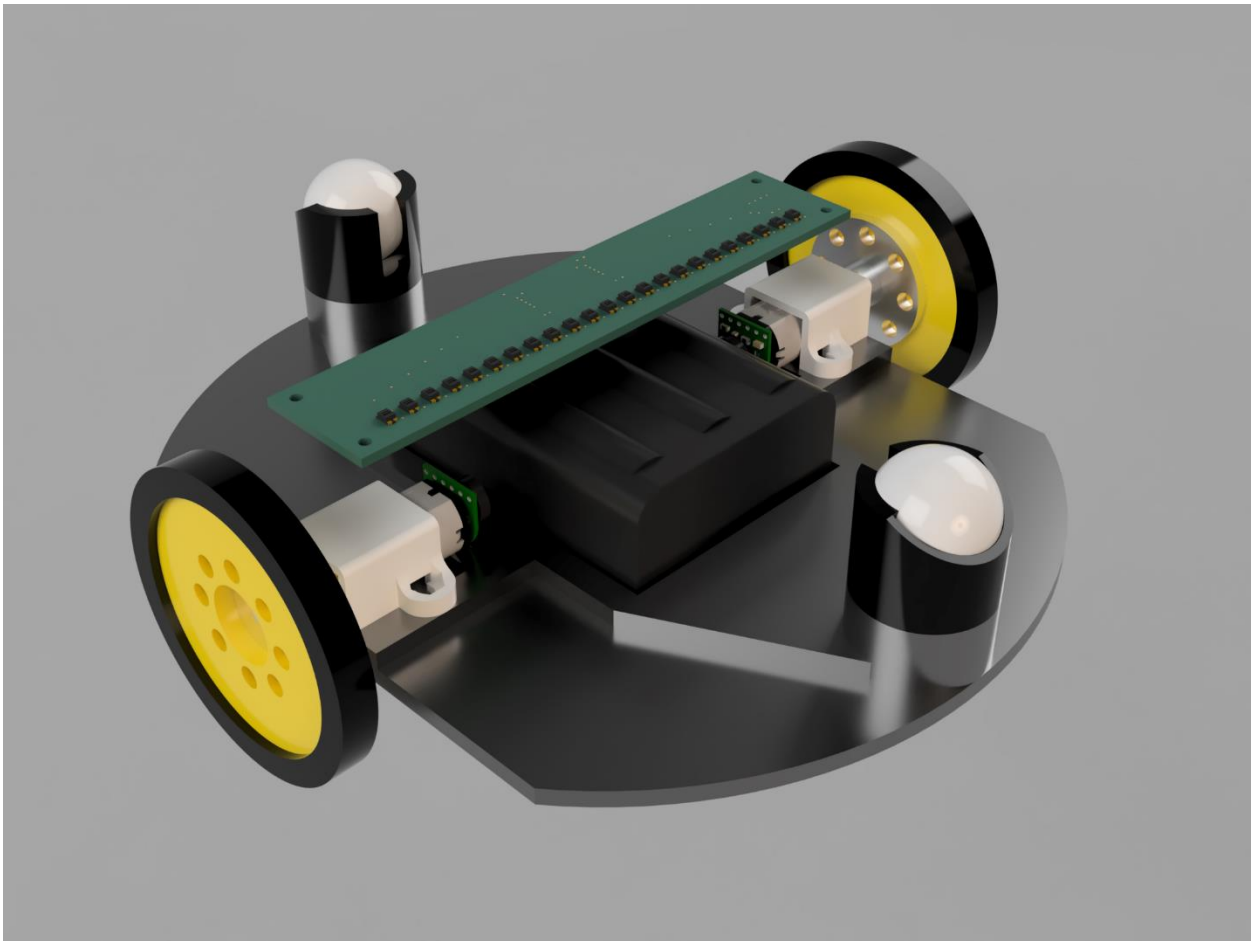


Figure 1 Line sensor placement (bottom side of the robot). Sensor is in green.

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:

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
Send latest sensor data	0x05	N/A

Sensor command responses:

Command	**Response data**		
Reset	N/A		
Start calibration	uint8_t command code		
	uint8_t status		
		0x00	OK
		0x01	Critical failure (unrecoverable failure, try reboot)
Finish calibration	uint8_t command code		
	uint8_t status		
		0x00	OK
		0x01	Critical failure (unrecoverable failure, try reboot)
		0x02	Calibration failed (not enough difference between black and white on some / all sensor units)
	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	uint8_t command code		
	uint8_t status		
		0x00	OK
		0x01	Critical failure (unrecoverable failure, try reboot)
		0x02	Calibration failed (not enough difference between black and white on some / all sensor units)
Send latest sensor data	uint8_t command code		
	uint8_t status		
		0x00	OK
		0x01	Critical failure (unrecoverable failure, try reboot)
		0x03	Sensor not calibrated (data will be in raw sensor format, which depends on sensor implementation)
		0x80	Flag, if set means data in the response in new (was never transmitted)
	uint16_t array of length NUMBER_OF_SENSORS in UQ1.15 format which contains value of the sensor unit		

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).

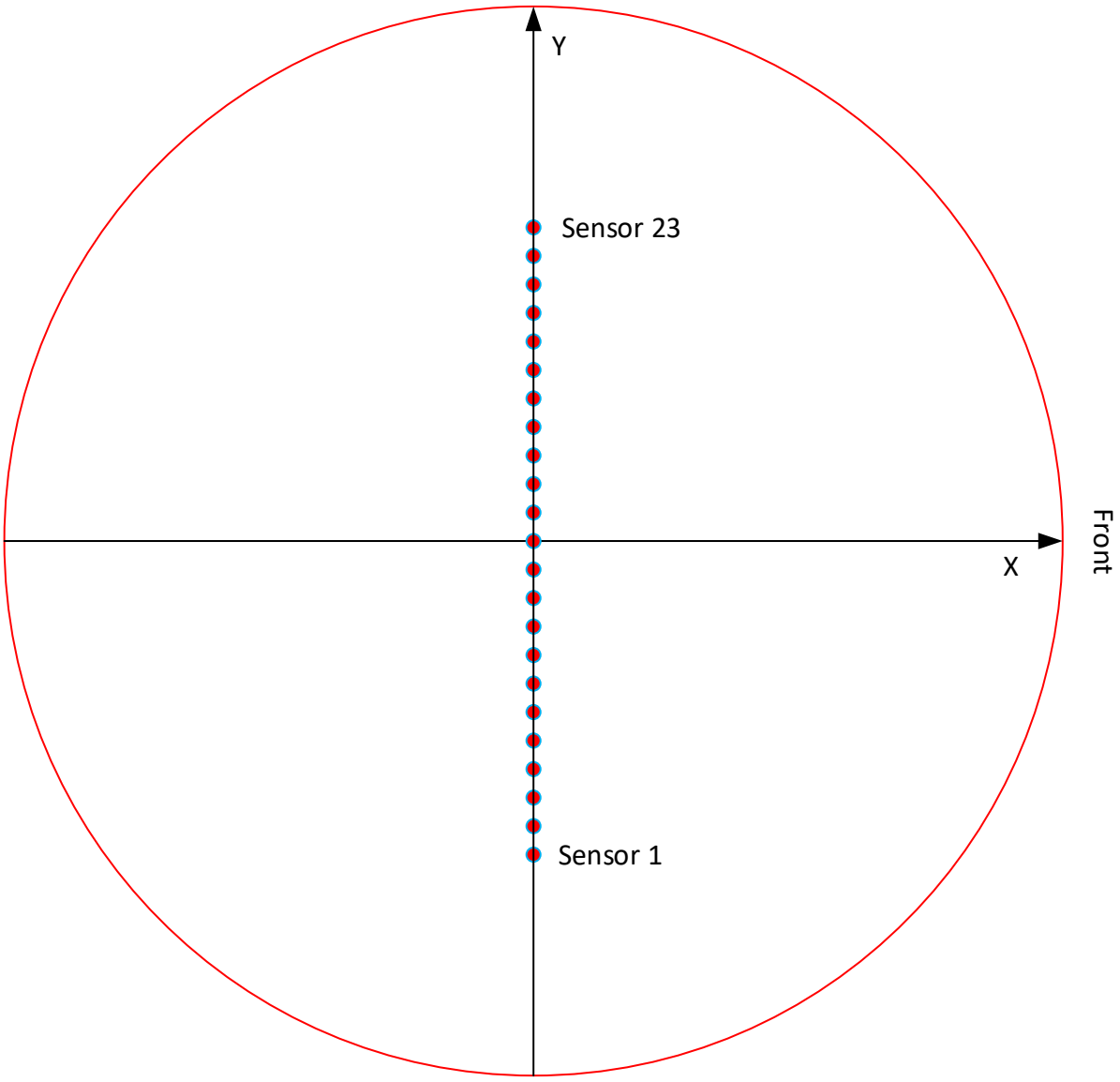


Figure 2 Robot coordinate frame and sensors placement.

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).

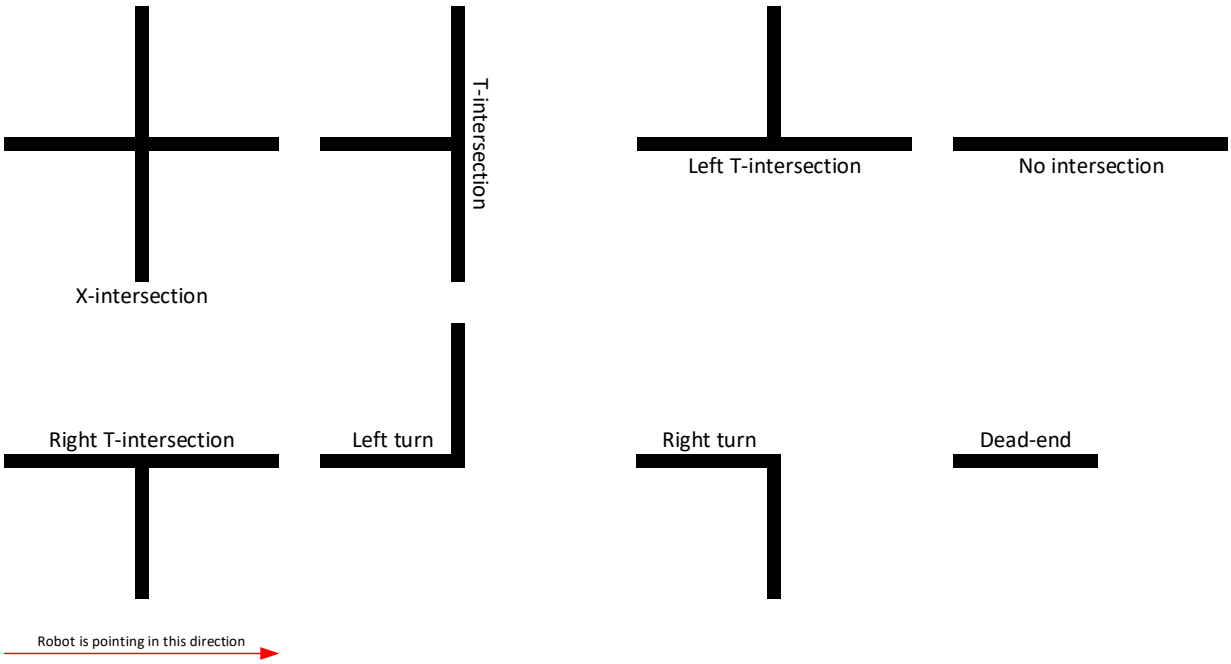


Figure 3 Intersection types

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.

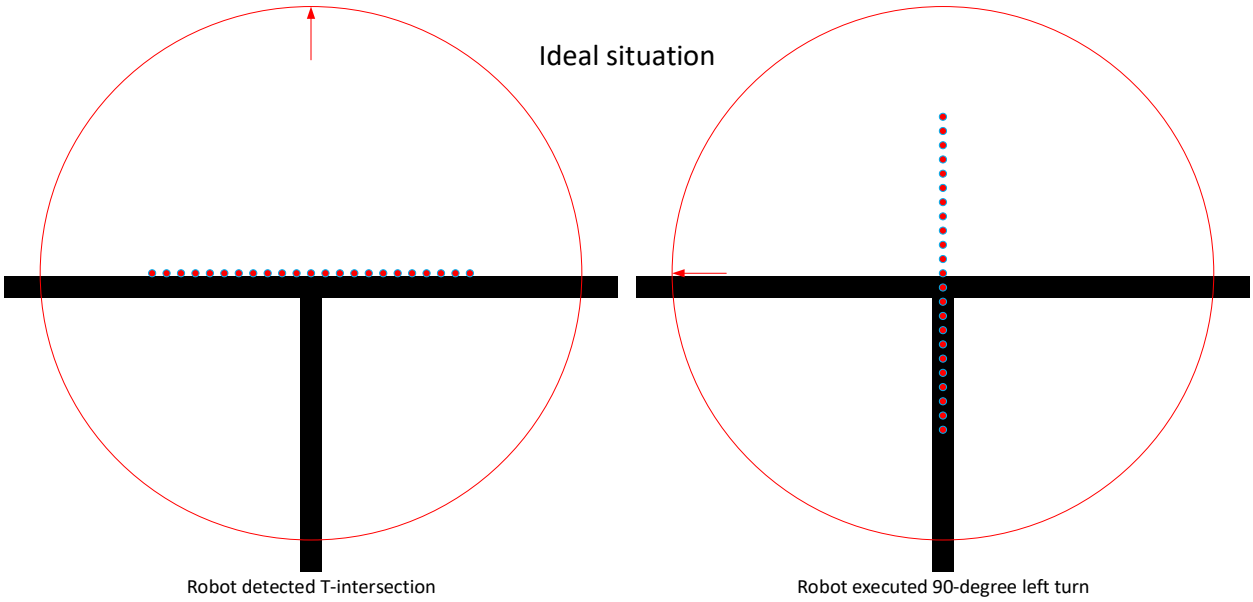


Figure 4 Robot turn in an ideal world

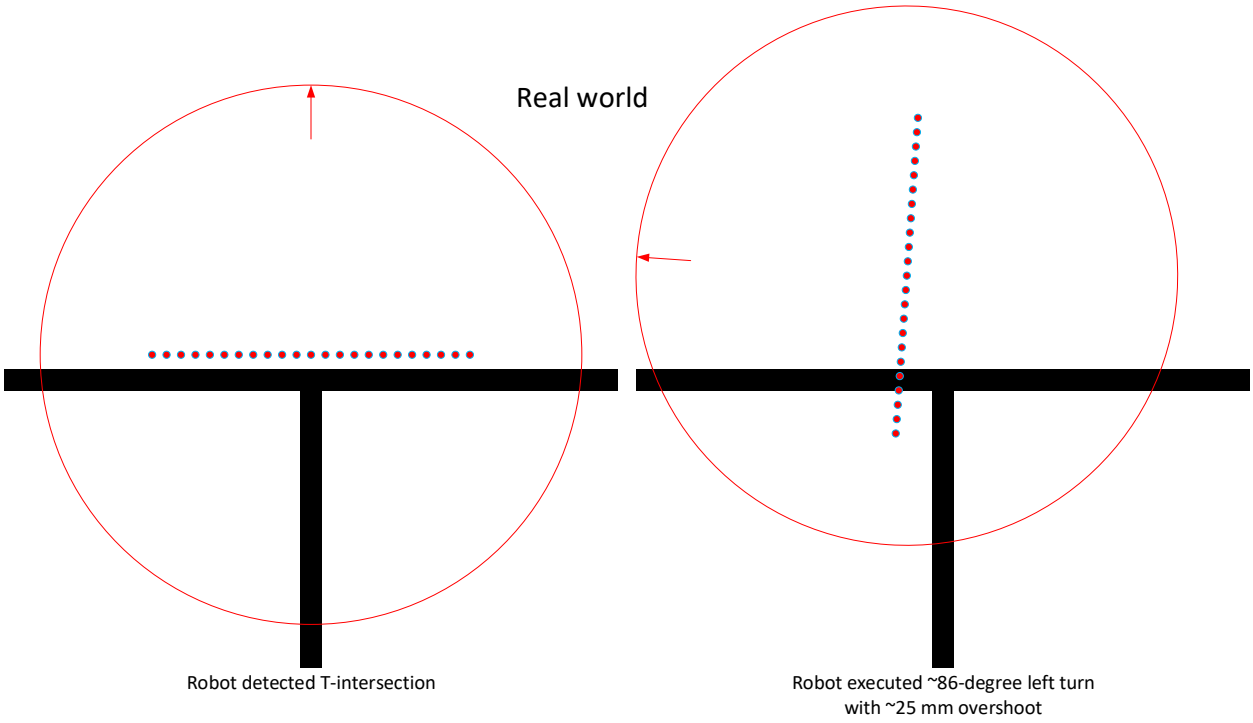


Figure 5 Robot turn in real world

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.

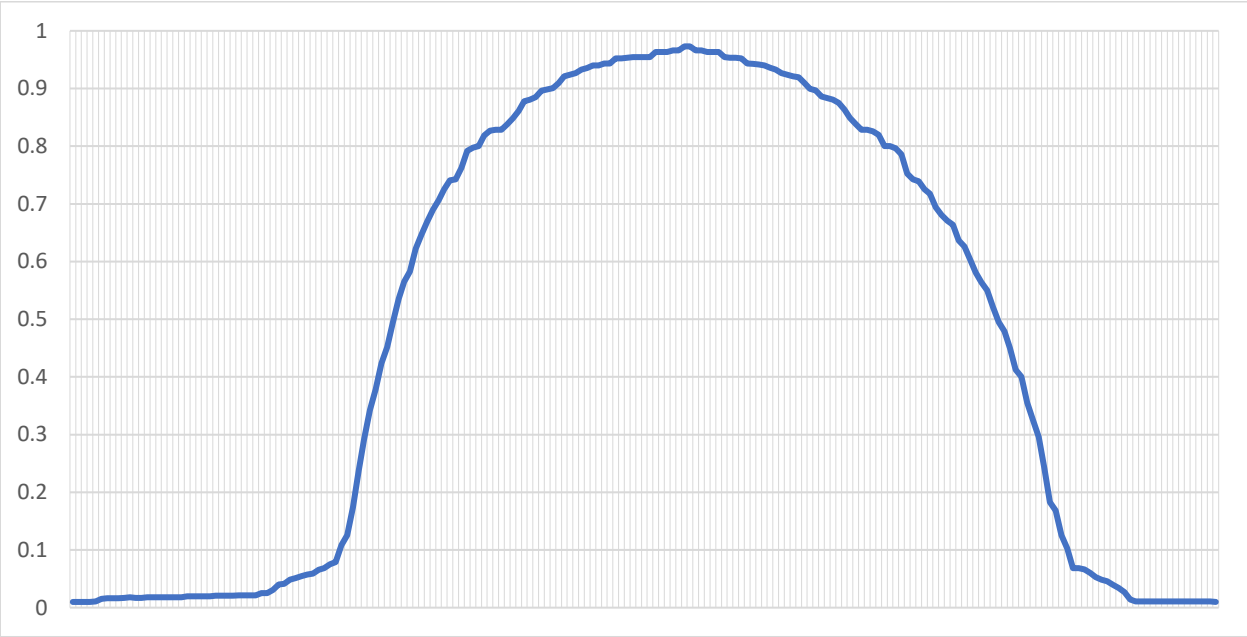


Figure 6 Sensor output when crossing the line (non-canted sensor placement)

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).

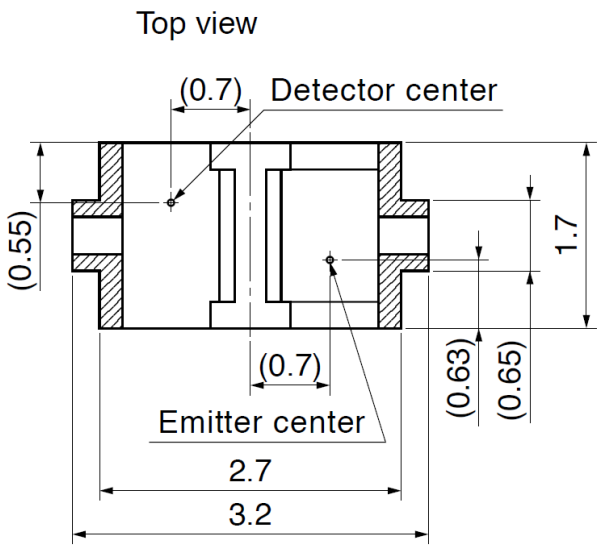


Figure 7 Sensor unit top view (this is from Sharp GP2S60 specification).

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.

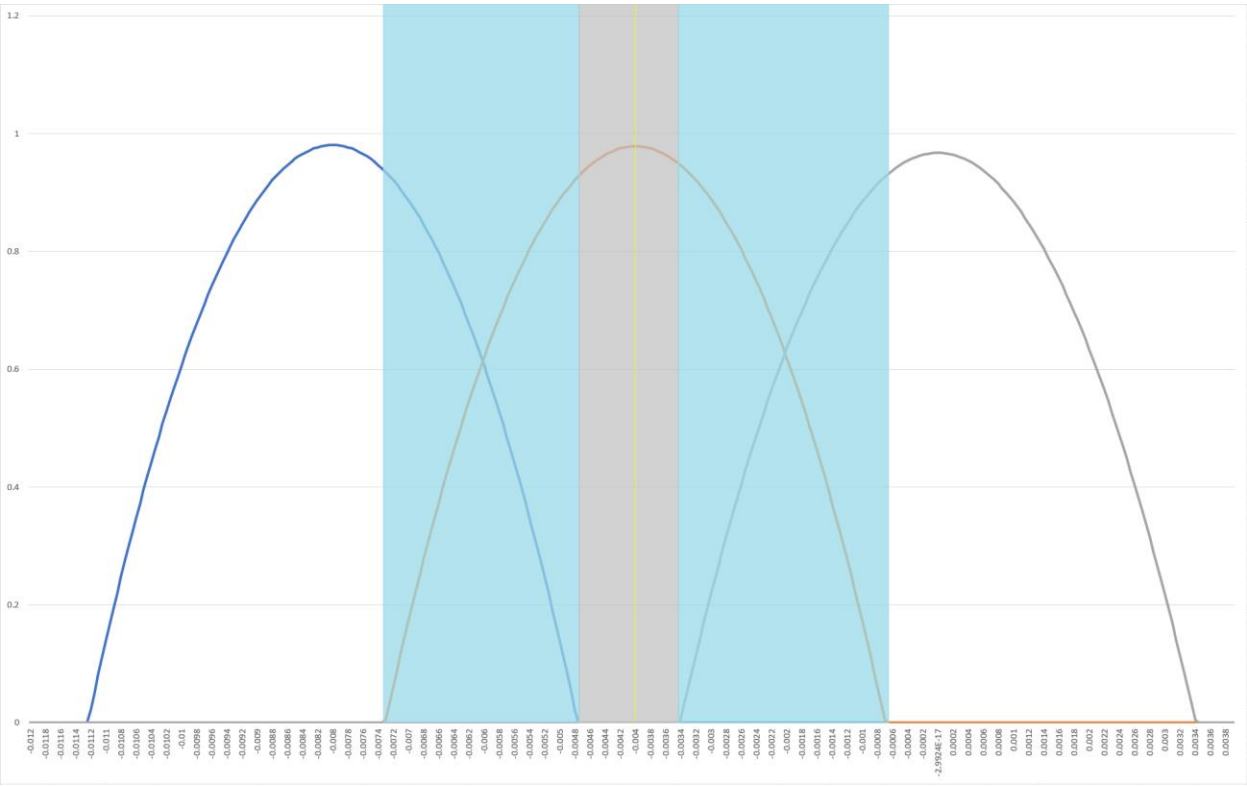


Figure 8 Sensor unit model (with neighboring sensor units included)

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.

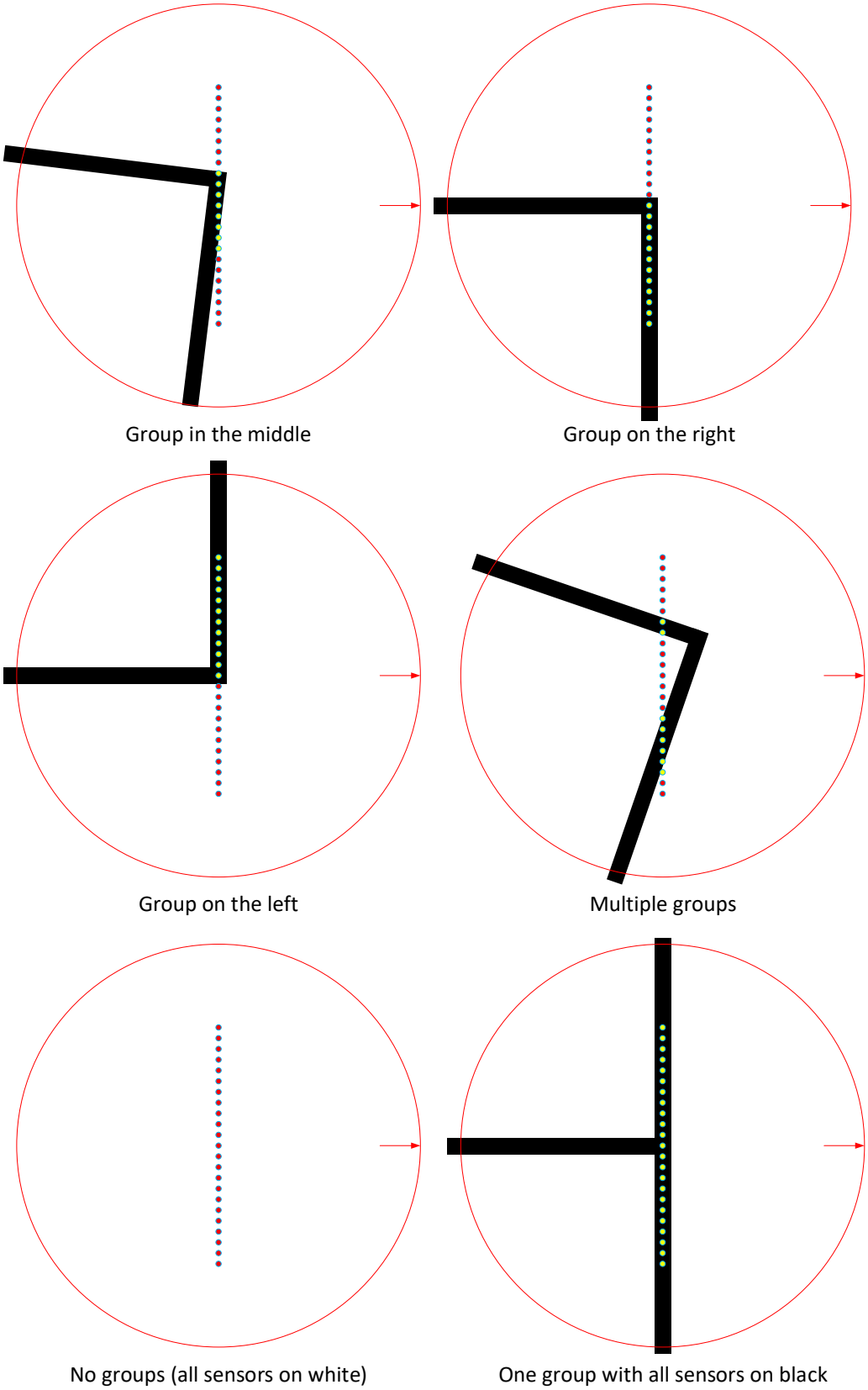


Figure 9 Cases with more than two sensor units having non-zero values

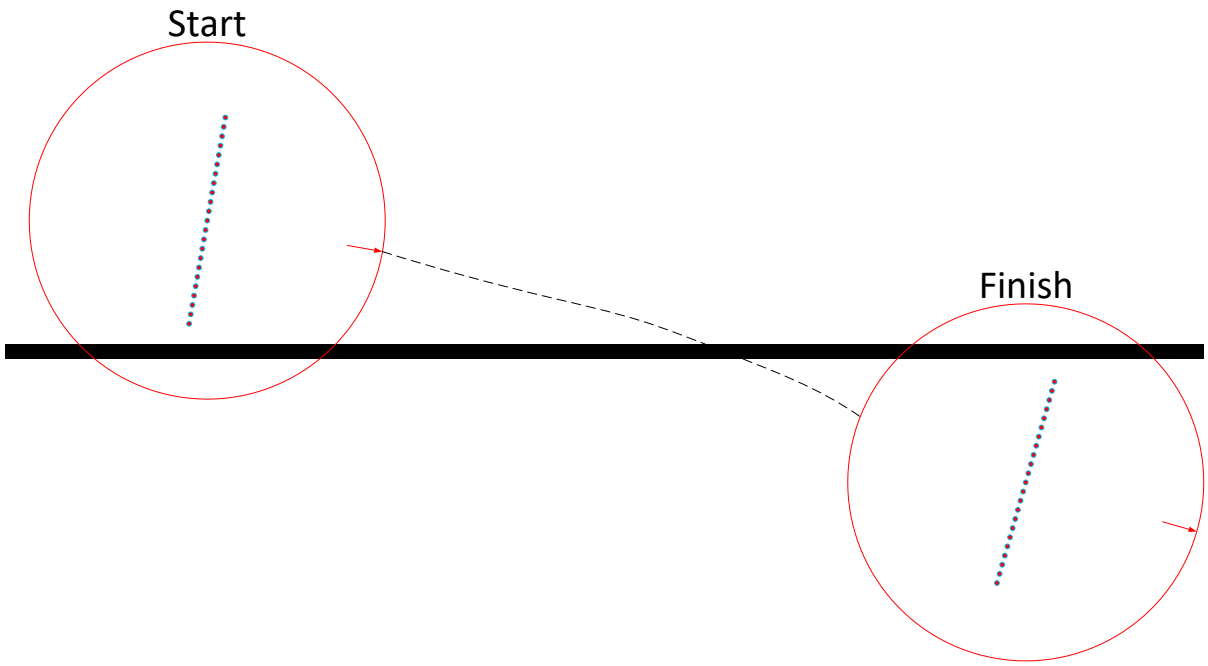


Figure 10 Robot run for test "y" position estimation

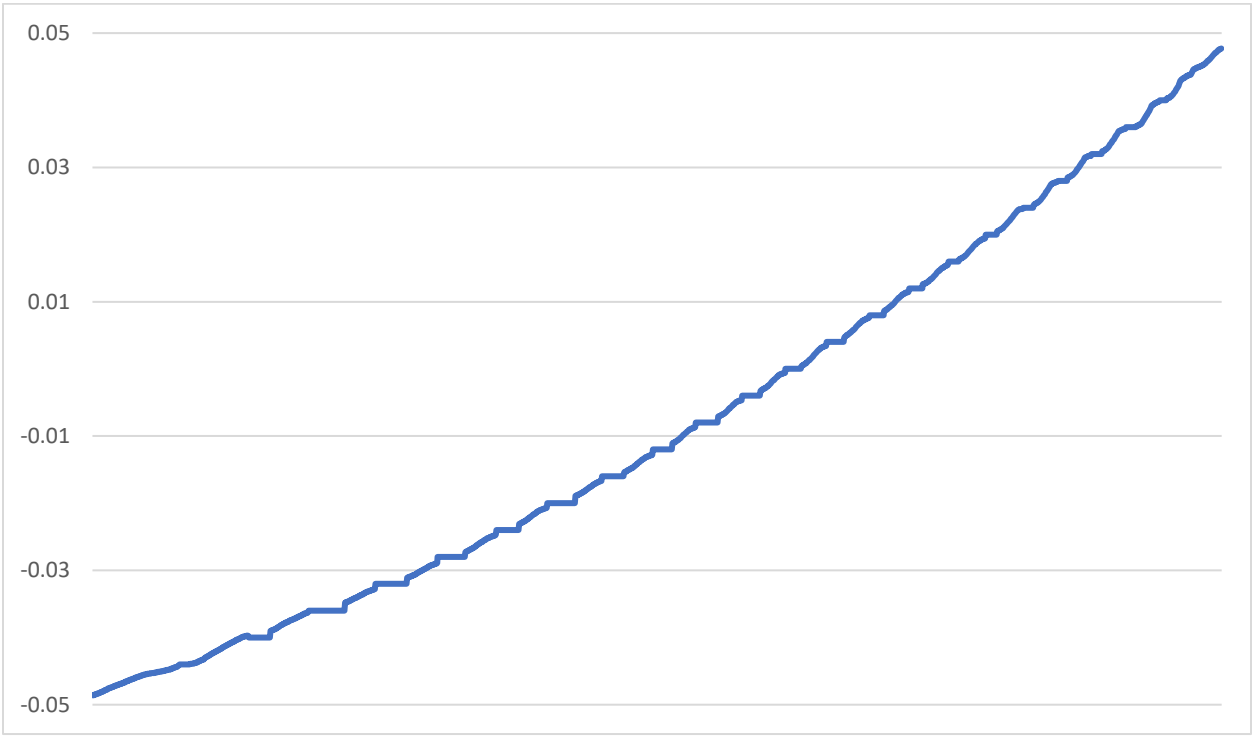


Figure 11 Robot position estimation using sensor model

4. Intersection type detection

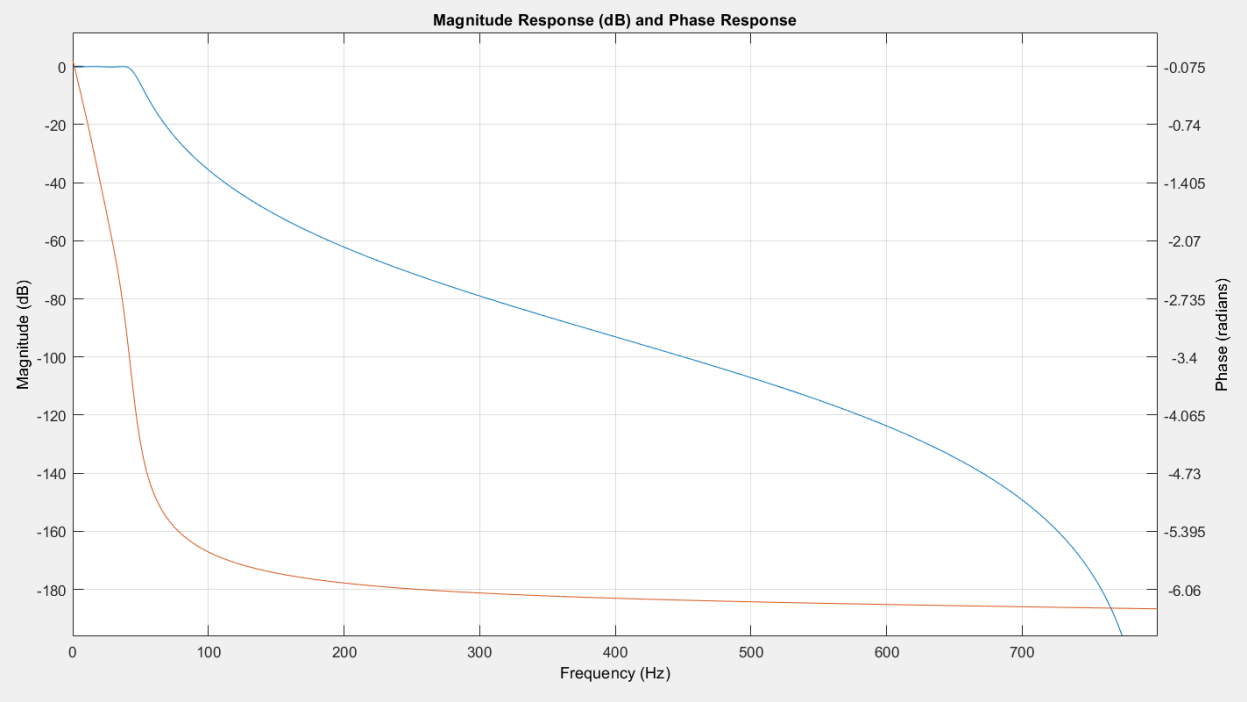
Intersection type detection is a higher-level function as robot can be moving at an angle to the feature we are trying to detect. It also helps if we know robot horizontal and angular speed. For more information see Kalman Filtering and Feature Detection chapters.

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 output voltage and current consumption

Line sensor unit output voltage depends on:

- If sensor unit is over all black, all white or over a thin black line. Black thin line is surrounded by white and neighboring sensor units light will be reflected to our sensor unit and will lower its value (see Figure 15)
- Input voltage. Higher voltage improves all black (see Figure 15) and all white (see Figure 13) readings but makes thin black line readings a bit worse if input voltage gets too large (see Figure 15). The biggest drawback of higher voltage increase is that it also increases power consumption and minimum voltage on the batteries.
- Ratio of values of resistors R1 / R2 (see Figure 12). If ratio is too low or too large there is very little difference between reading of all white and all black (Figure 14). Ratio should be chosen somewhere in the middle to maximize distance between all white and all black readings.
- Value of resistor R1. 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 sensed object must be close to the sensor

For ADC we want maximum value of output voltage to be < 3.3 V (this will allow us to have max ADC reading speed). To pick the best value we can use data from Figure 15 which has maximum values for each resistance and input voltage. Another important consideration is to pick max range between all white and feature readings to increase resolution (see Figure 16).

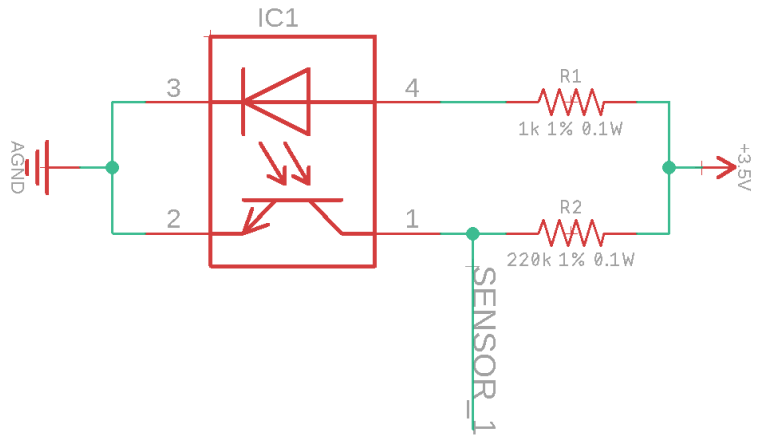


Figure 12 Sensor unit schematic

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.

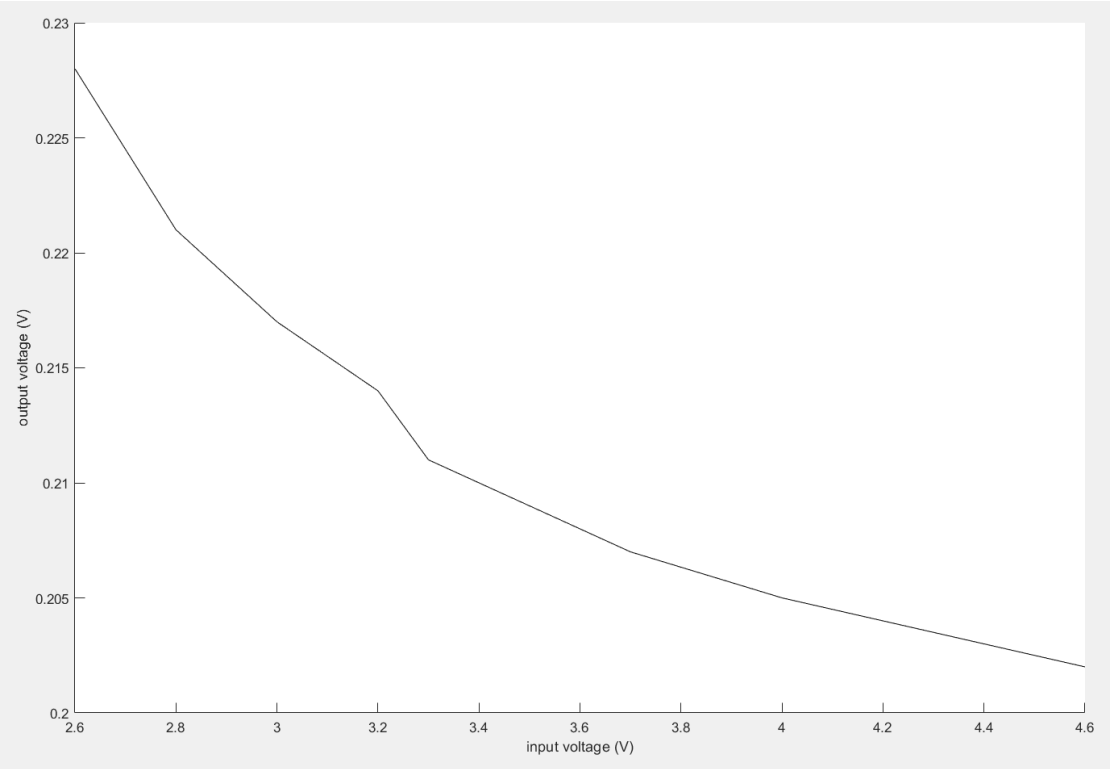


Figure 13 Relationship between sensor output voltage and input voltage when sensor is over all white (R1=1 kOhm, R2=220 kOhm). This is for single sensor unit.

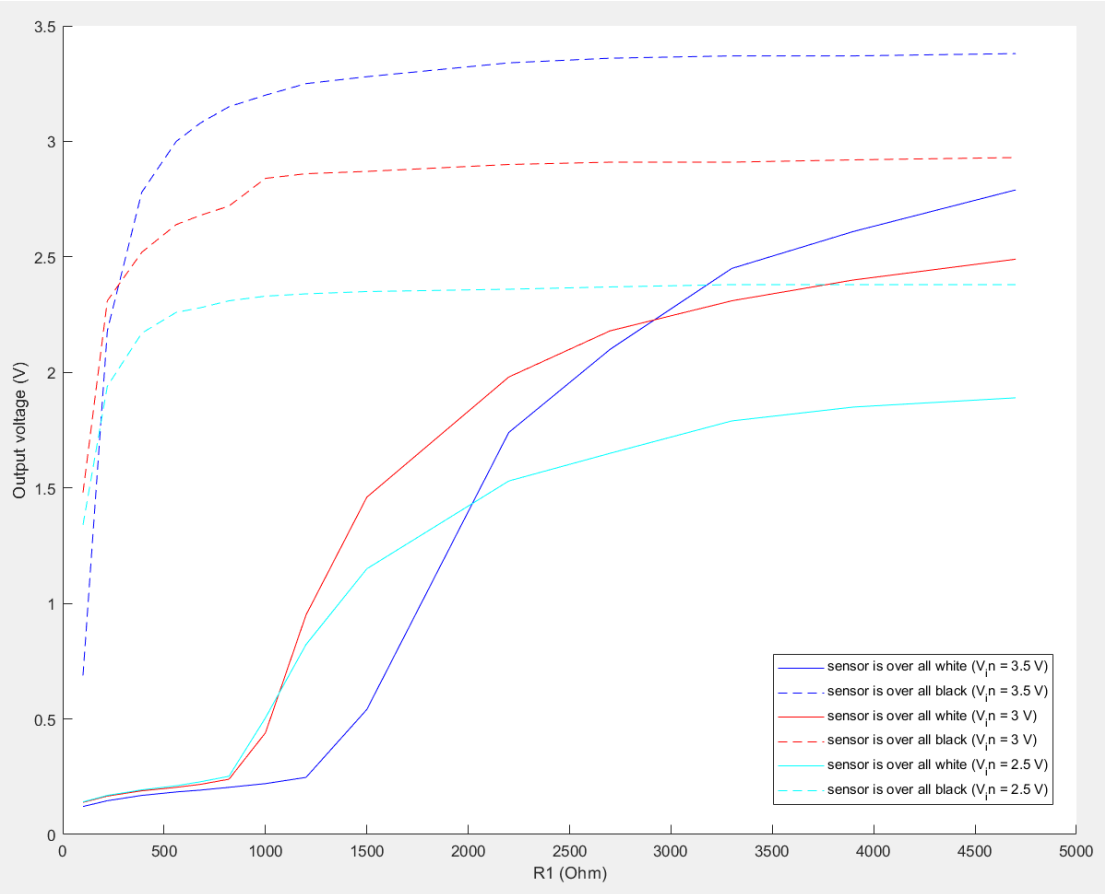


Figure 14 Output voltage dependence on R1 (R2 = 220 kOhm, and lighting conditions are fixed). This is based on single sensor unit readings; multiple units will have smaller all black value.

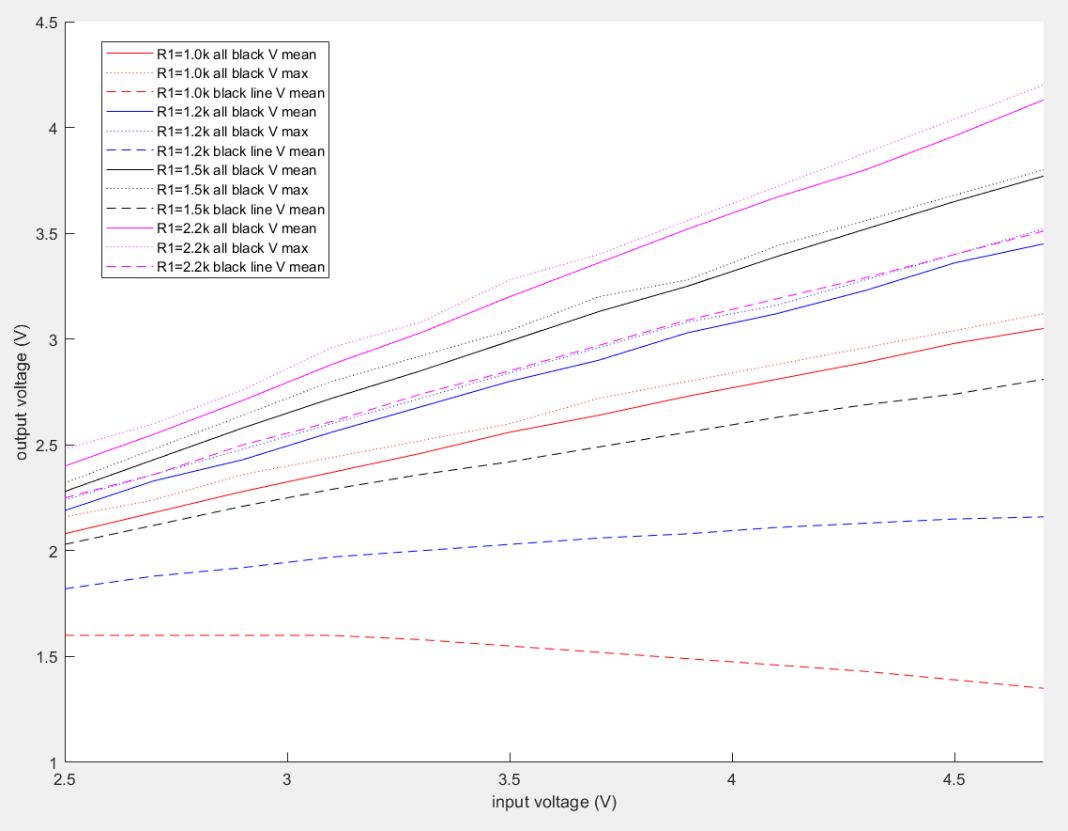


Figure 15 Dependence of output voltage on input voltage, R1 value, and type of surface (2mm above surface, multiple sensor units)

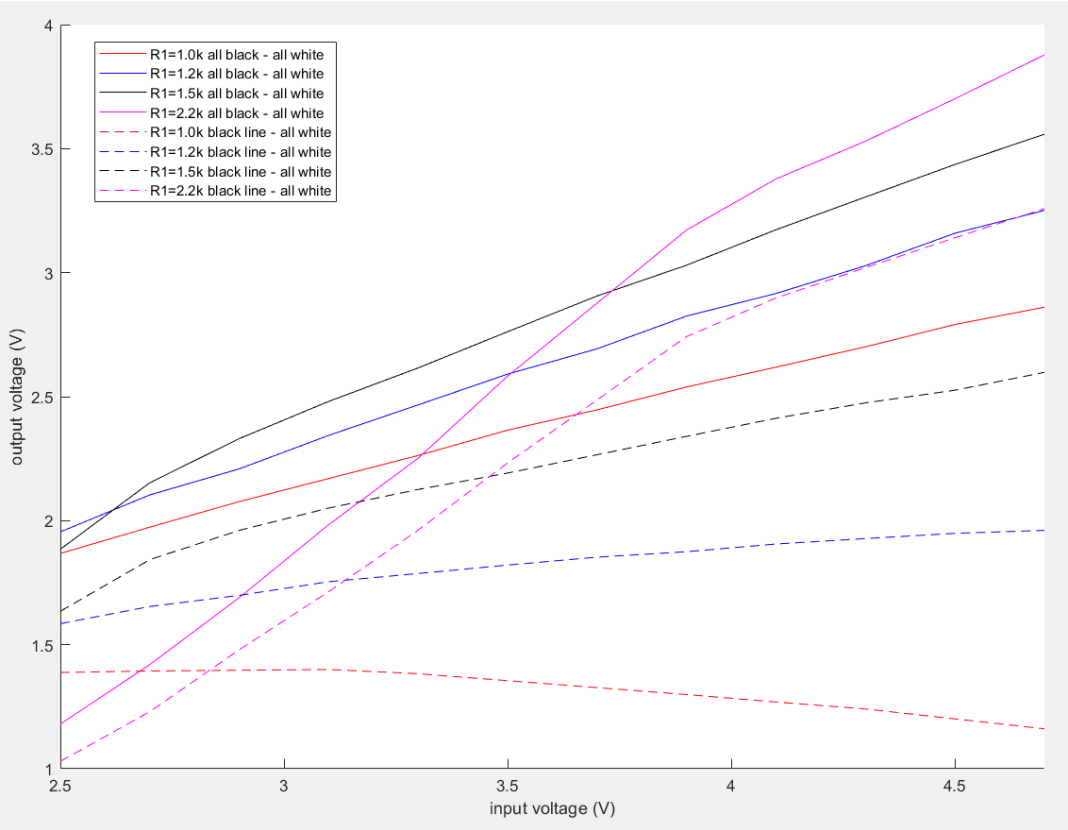


Figure 16 Difference of sensor reading over a feature with sensor reading over all white for different values of R1 (2mm above surface, multiple sensor units)

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

R1 = 1.5 kOhm. Tolerance can be +/- 1% (based on input voltage).

R2 = 220 kOhm. Tolerance should be +/- 1%.

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

Input voltage is 3.3V as we only have 1 voltage regulator and want to keep all MCU at 3.3V logic level.

- (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

STM32L1 requires that VDDA and VDD be from the same voltage supply, STM32L4 can have different voltage supplies for those pins which can help to reduce the noise.