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).

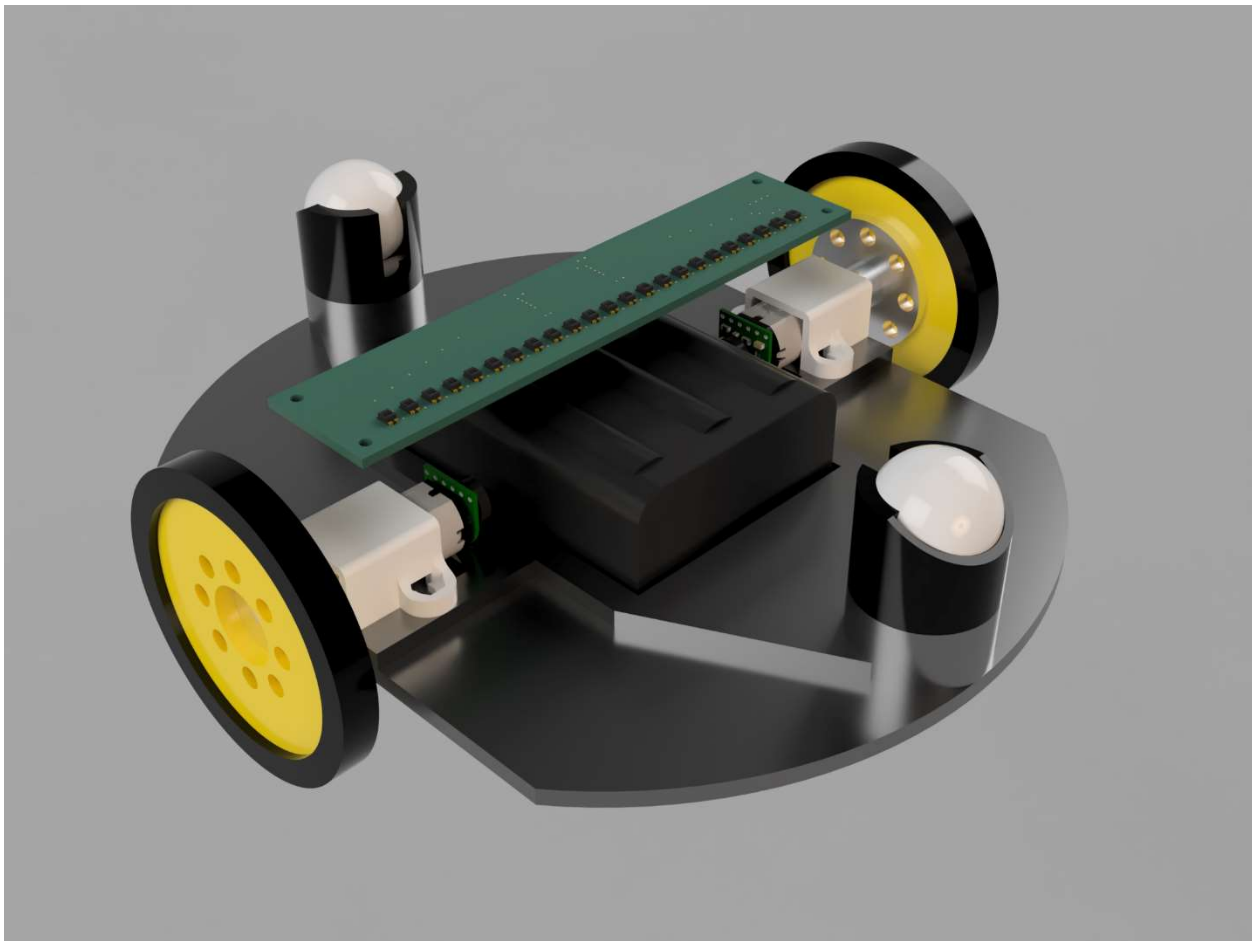


Figure 1 Line sensor placement (bottom side of the robot). Sensor is in green.

2. High level design

Line sensor requirement:

- Power provided to the sensor is from 6 AA batteries (~9 volts).
- Frequency is hardcoded to 1.6 kHz.
- To send / receive data SPI and USART interfaces are used. 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 is responsible for filtering data it collects before sending it to the client.
- 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:

- (receive) Reset
- (receive) Start calibration
- (receive) Finish calibration
- (receive) Calibration data (from previous calibration in the same environment, can be used after reset)
- (send) Send calibration data to the client
- (send) Send data to the client

Calibration data sent to the client has the following format:

- Error code (8 bits). Zero means line sensor is calibrated.
- Array of size 2xN containing values for 1st minimum N values and then maximum N values for all N sensor units. Each entry is 16-bit unsinged integer.

Data sent to the client has the following format:

- Error code (8 bits). Zero means line sensor is operational.
- Array of size N containing values for N sensor units. Each entry is 16 bits in fixed point Q1.15 format. Values are normalized to [0…1] range.

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).

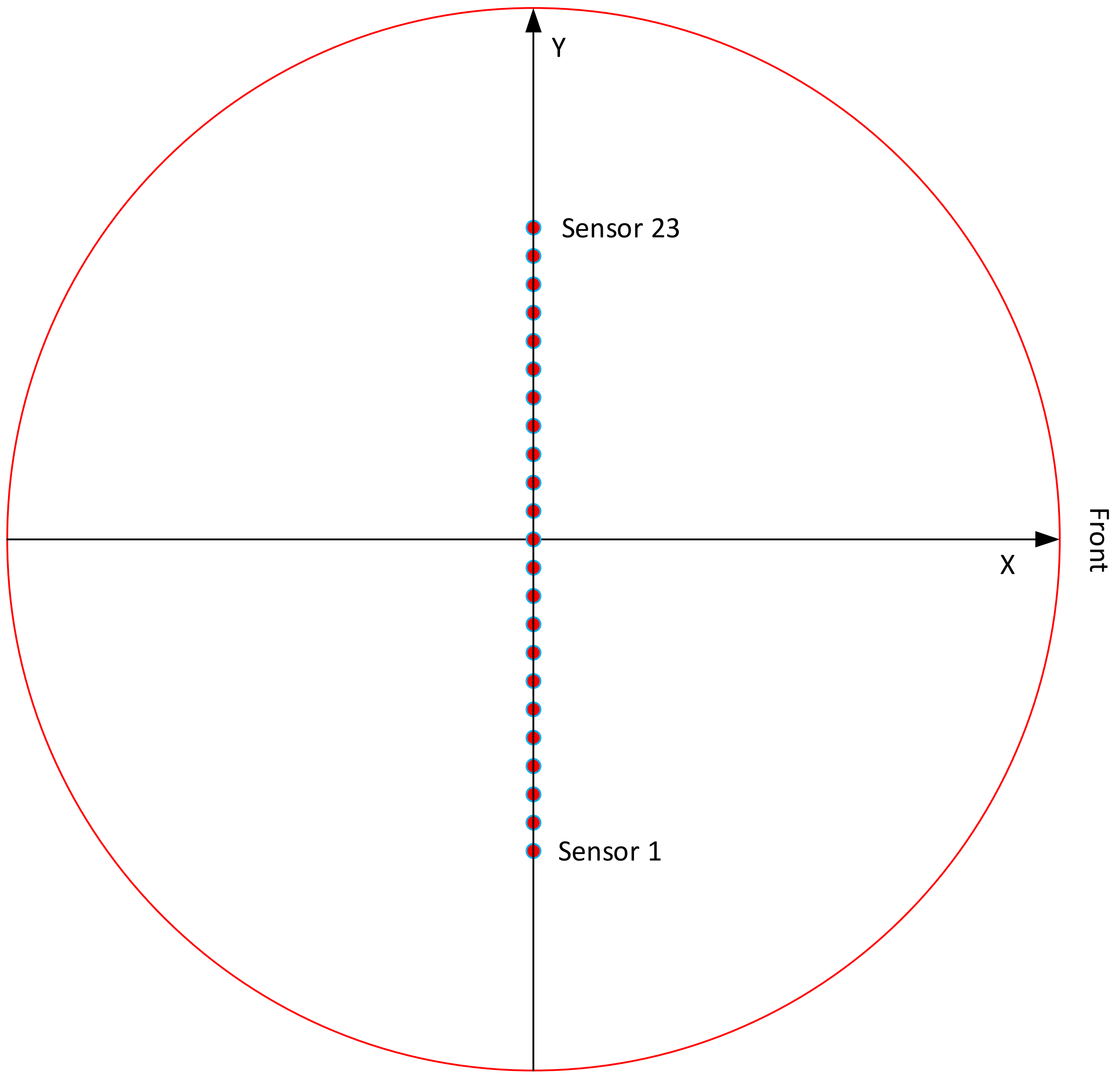


Figure 2 Robot coordinate frame and sensors placement.

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).

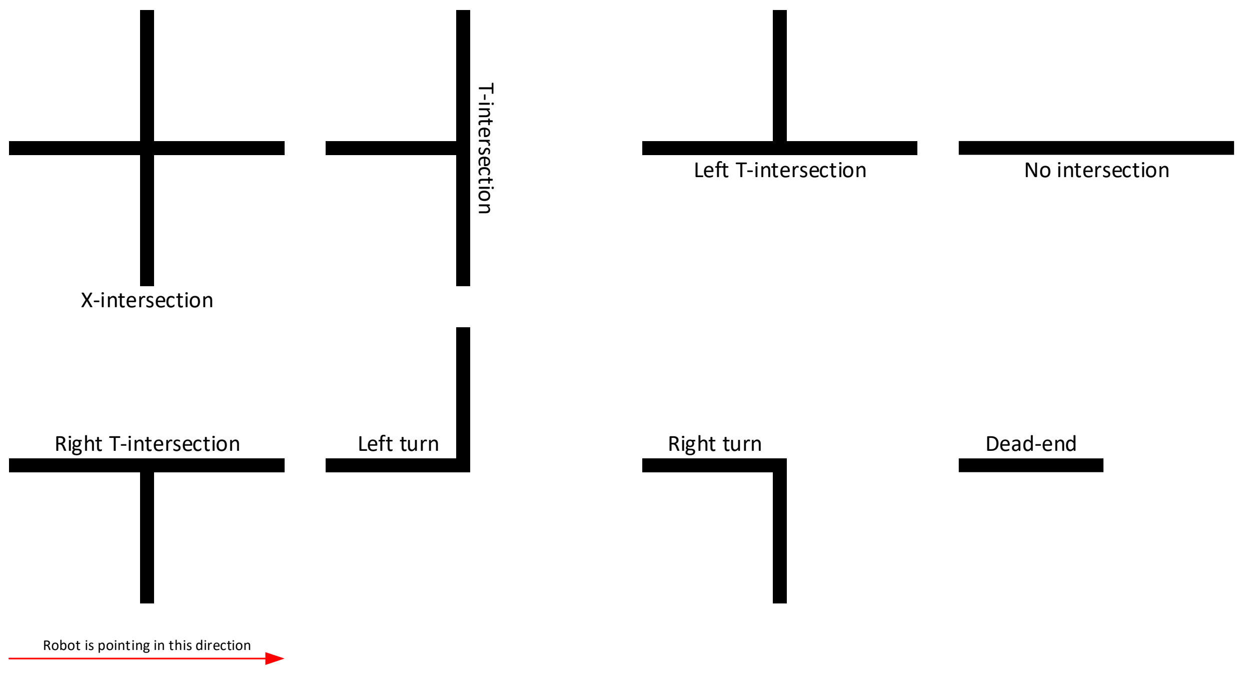


Figure 3 Intersection types

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.

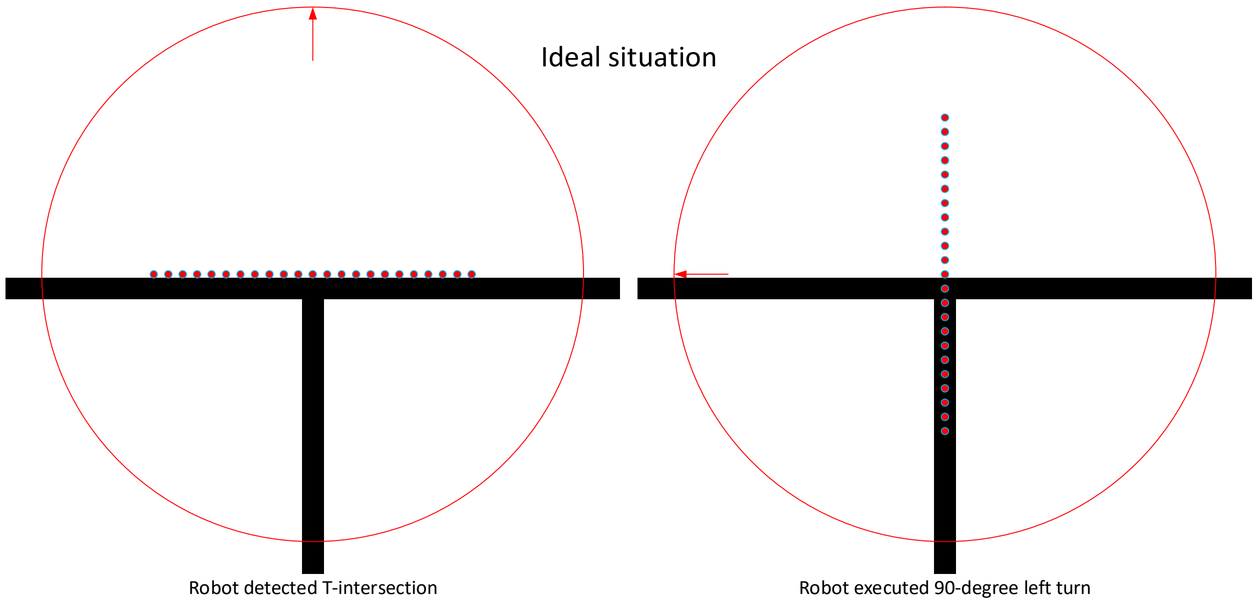

Ideal situation
Robot detected T-intersection
Robot executed 90-degree left turn

Figure 4 Robot turn in an ideal world

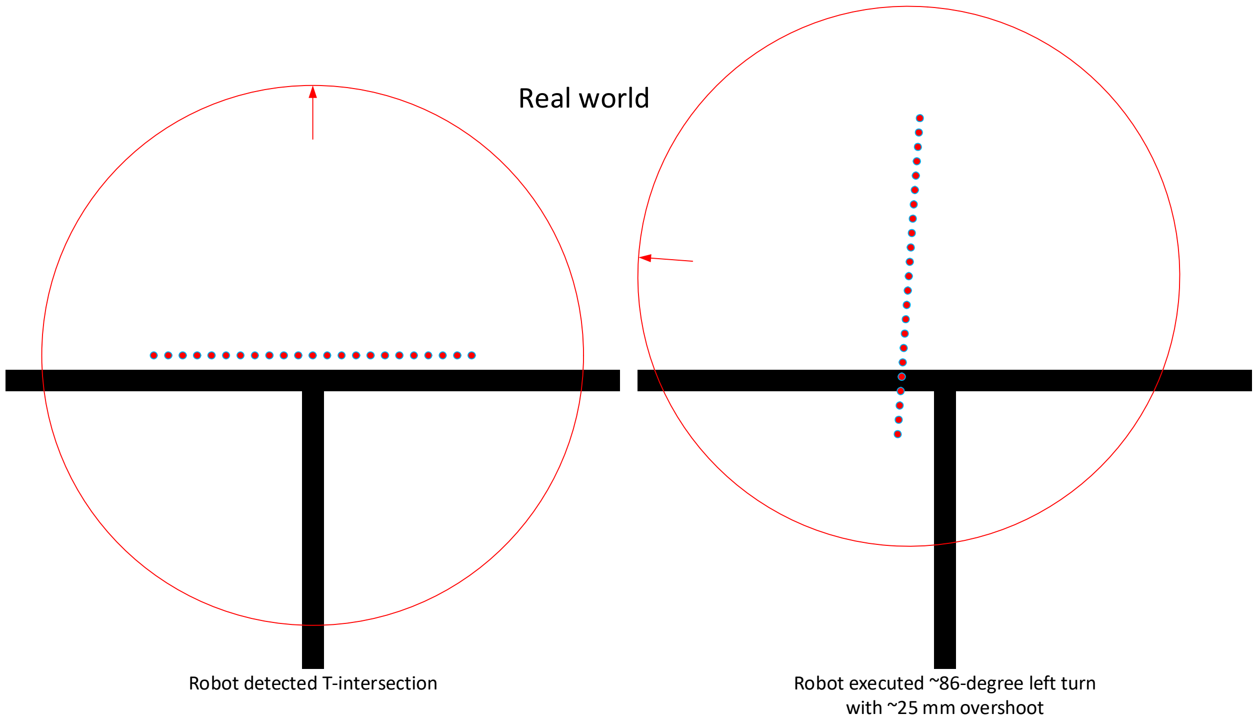

Real world
Robot detected T-intersection
Robot executed ~86-degree left turn
with ~25 mm overshoot

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.

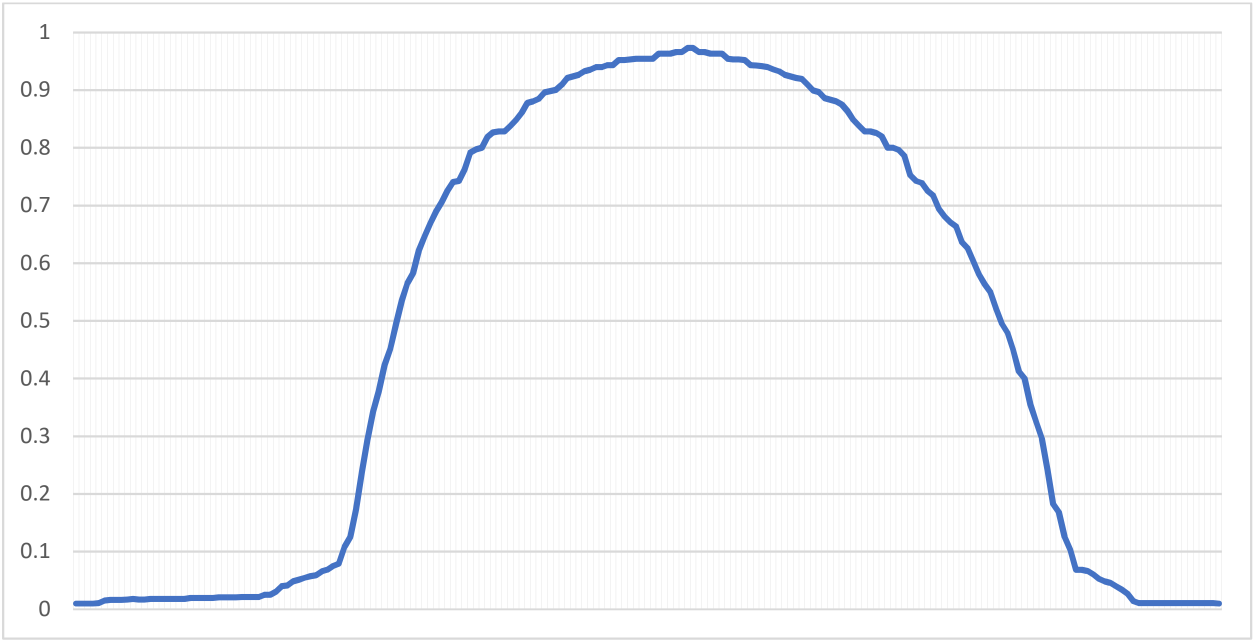


Figure 6 Sensor output when crossing the line (non-canted sensor placement)

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).

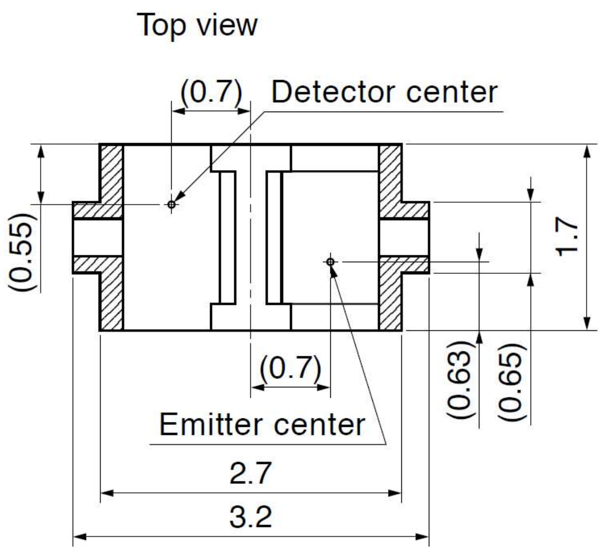


Figure 7 Sensor unit top view (this is from Sharp GP2S60 specification).

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.

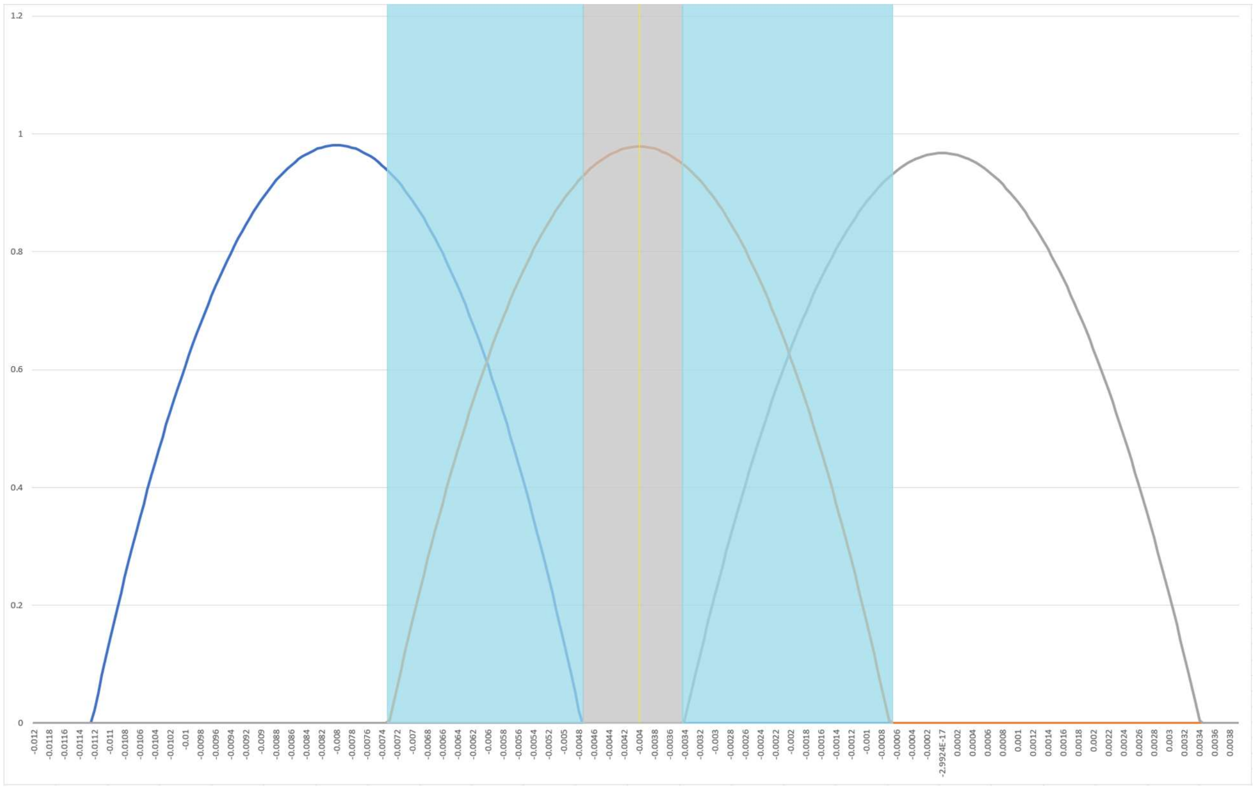


Figure 8 Sensor unit model (with neighboring sensor units included)

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.

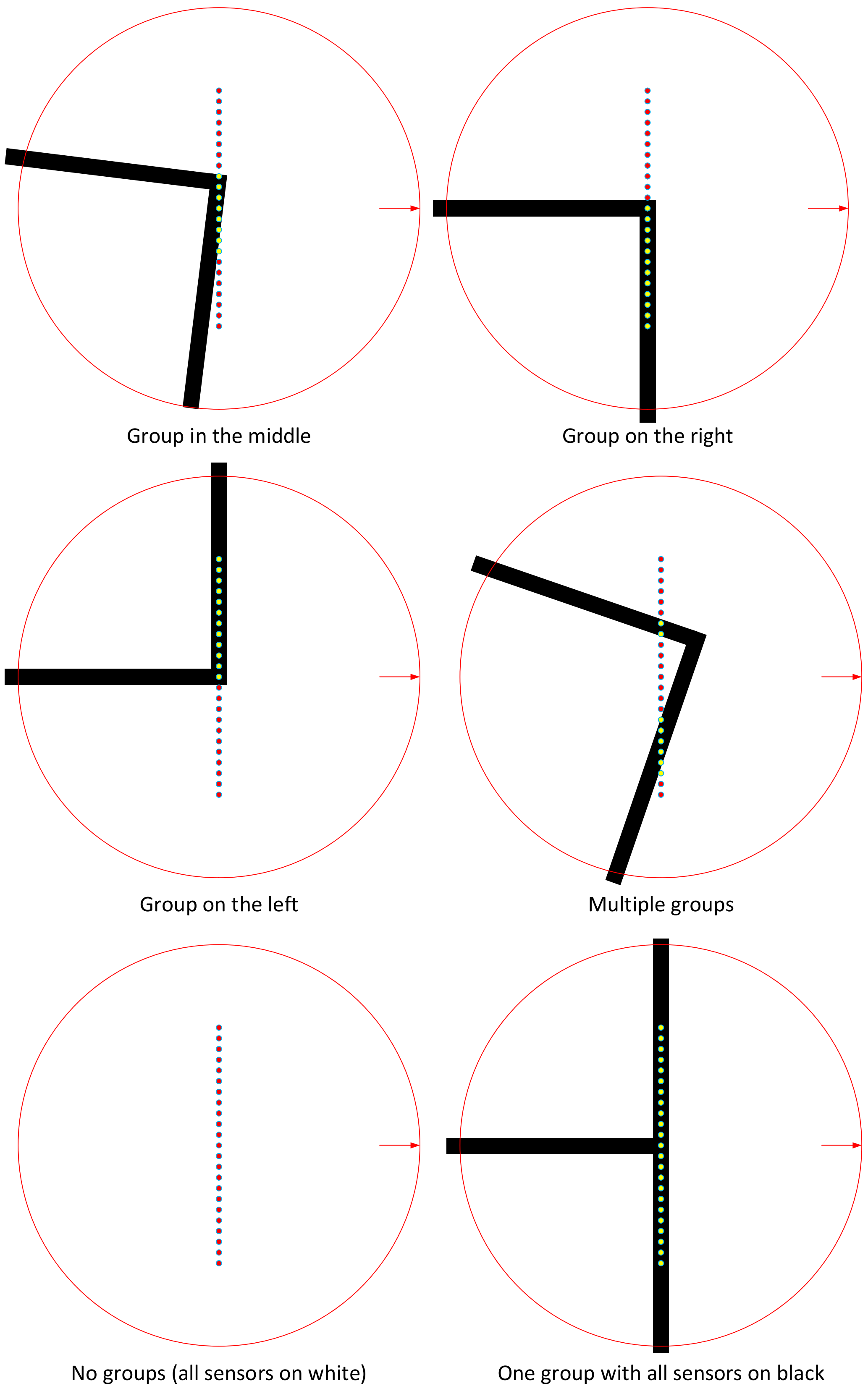


Figure 9 Cases with more than two sensor units having non-zero values

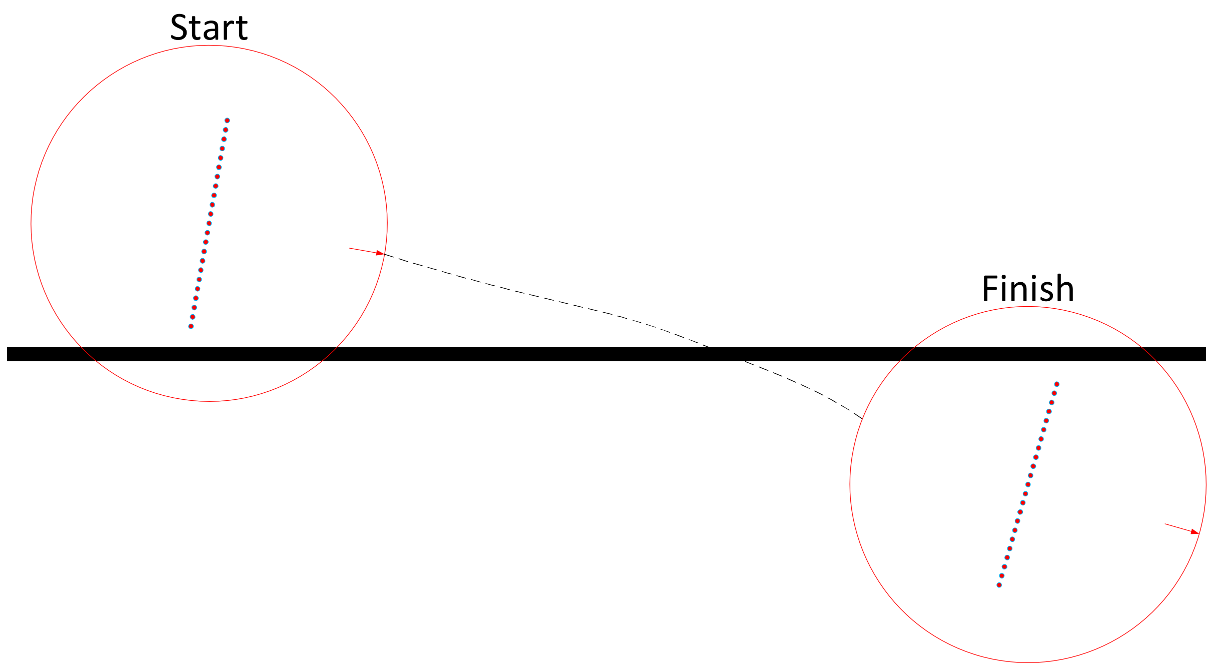


Figure 10 Robot run for test "y" position estimation

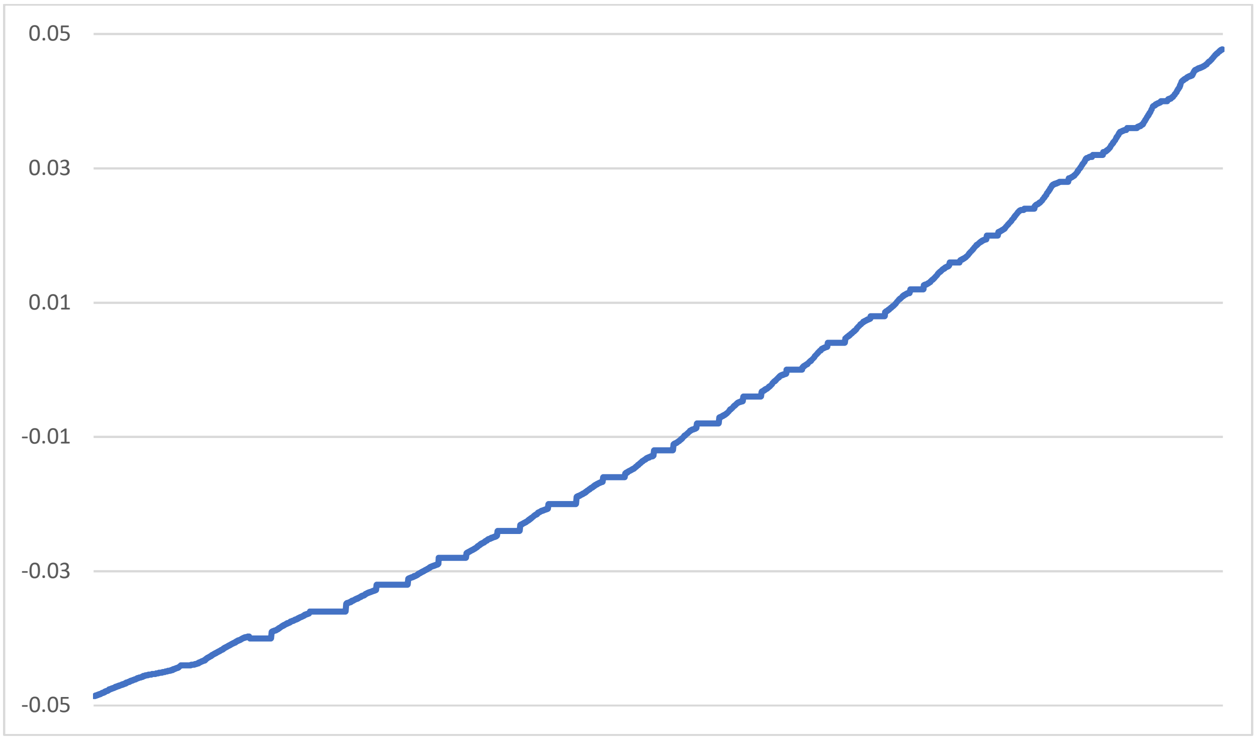


Figure 11 Robot position estimation using sensor model

4. Intersection type detection

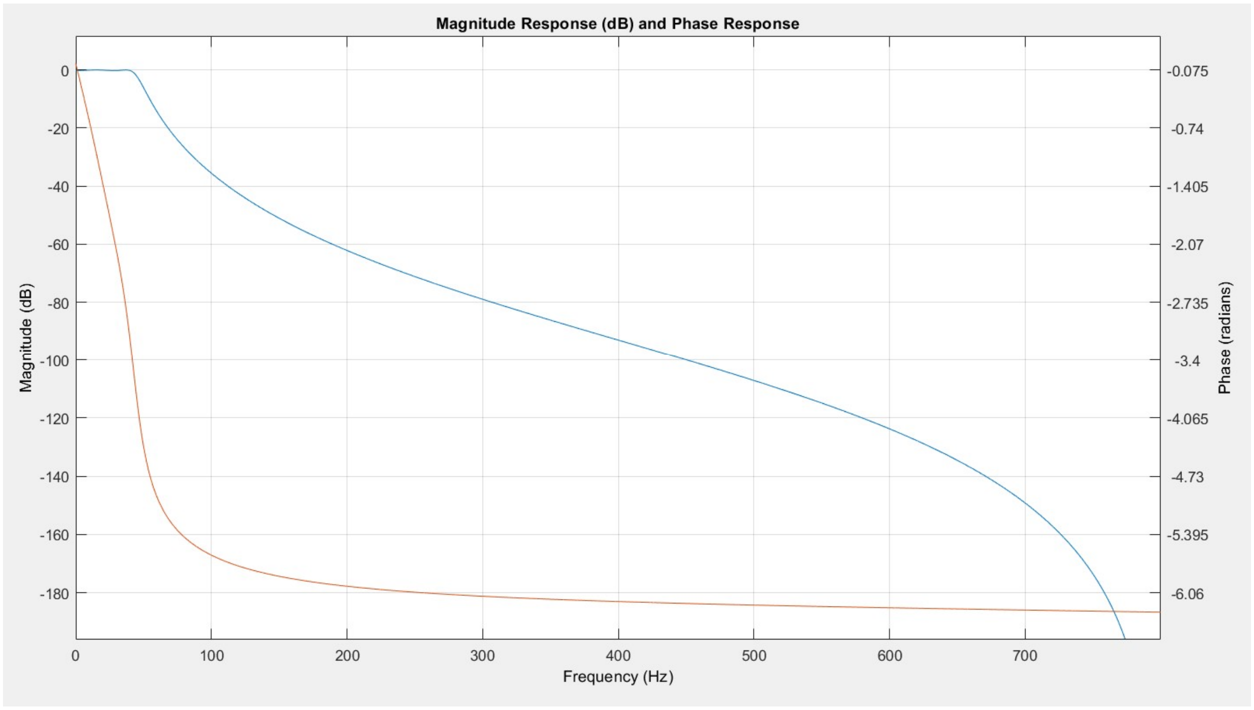
Intersection type detection is a higher-level function as robot can be moving at an angle to the feature we are trying to detect. It also helps if we know robot horizontal and angular speed. For more information see Kalman Filtering and Feature Detection chapters.

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. Hardware & 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 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.

Resistor values were chosen so black is close to 3.3V as well as to reduce power consumption. As sensor is very close to the sensed object, we don't need a lot of infrared light from the emitter, in fact having too much light causes whiteout and difference in voltage between black and white gets smaller. This can be observed on first and last sensor units, both are only lit from one side (they only have one neighbor) and difference in voltage between black and white for those sensor units is higher compared to any other units.

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

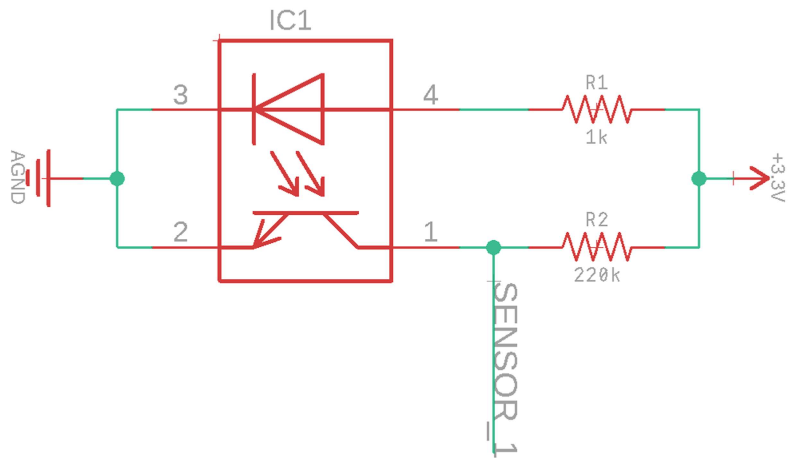


Figure 12 Sensor unit schematic