Paths in a labyrinth - Junction classification

Yannick Lang
yannick-stephan.lang@stud.uni-bamberg.de

SME-PHY-B
Physical Computing
Otto-Friedrich-Universität Bamberg
Summer Term 2022

1 Introduction

The following paper describes an arduino based system capable of distinguishing types of junctions via ultrasound.

The problem has been touched upon in [1]. For this project, the focus lies on classification of the environment into corridor, X-Junction or T-Junction. Classification is performed after capturing distance measurements during a 180 deg turn of the arduino platform from left to right.

Quick note about Terminology: Unlike [1], angles will be used from 0 deg for the left path, up to 180 deg for the right path. 90 deg therefore corresponds to the path ahead.

2 Setup

The following section describes the physical components for the project, starting with the actual hardware and also covering the environment.

2.1 Hardware

From a hardware perspective, two main platforms are used: an arduino micro controller which captures the output from the sensors and a laptop, which processes the data.

Arduino This consists of an arduino Nano v.3, a MPU6050 used for its gyroscope sensor and an ultrasonic sensor (HCSR04). The wiring can be seen in Figure 1.

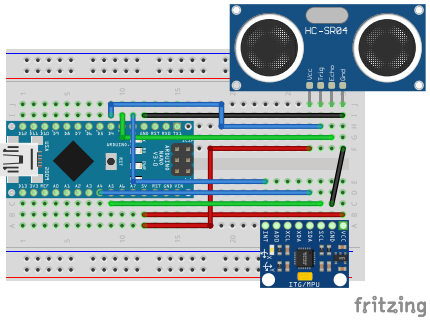


Figure 1: Wiring of the Sensors. Both sensors are mounted upright, with the ultrasonic sensor aimed at the top of the image. Note: this may cause the order of pins to be reversed, wiring in the schematic is according to the labels, not the order.

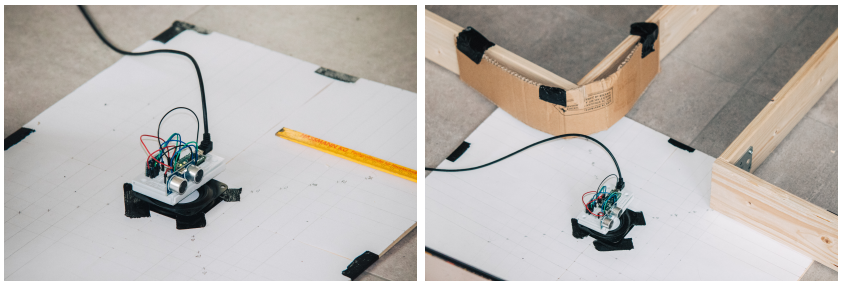


Figure 2: Images of the mounted board

Laptop The arduino is connected via mini USB cable to a laptop, which provides the micro controller with power and receives and evaluates sensor data from it. To enable a smooth rotation, a angled USB cable may be used, but this is not necessary. While the receiver does not need to be a Laptop, for instance a Desktop computer or even a raspberry pi could be used instead, I will refer to this component as a Laptop in the following sections.

2.2 Environment

For this project, the breadboard, on which the arduino and both sensors are connected, is mounted on a swivel plate. This allows the for easy and smooth rotation. The board should be mounted in a way that puts the ultrasound sensor close to its center of rotation.

To create the types of junctions that have to be classified, I created two L shaped pieces, consisting of two planks of wood each, joined in a 90 deg angle. Those two are sufficient to create every relevant type of environment, as can be seen in Figure 3.

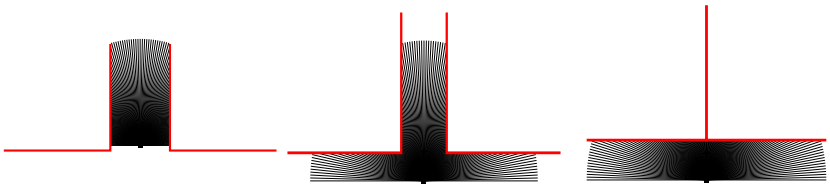


Figure 3: visualization of wooden planks and area the sensors scan. Left corresponds to corridor, middle to X Junction and right to T Junction

3 Software

The software for this project consists of two main parts: The code for the micro controller, written in C, and the code for the laptop, written in Typescript. This section details the methods used for this project.

3.1 Arduino

As stated previously, the code that controls the arduino is written in C. Its purpose is to configure the sensors, monitor them for new data and relay that to the Laptop when available. Communication with the MPU6050 sensor happens via the Inter-Integrated Circuit (I^2C) protocol. Communication with the laptop uses a serial connection via USB at a BAUD rate of 250000.

The main control logic for the micro controller is quite simple. Upon startup, all ports and sensors are configured and the clock is initialized. Then, a infinite loop is started, which relays sensor data to the laptop if a new value is available. Availability of new sensor data is determined via interrupts.

For the MPU6050, a interrupt signals that new data is available. In the interrupt handling routine, a variable is the set to one. In the main loop, this variable is checked, and if it is one, the x value of the gyroscope is read from the mPU via I^2C. After sending the data to the laptop via the serial interface, the variable is then reset to 0.

For the Ultrasonic sensor, interrupts are used to determine logic level changes on the pin that is connected to the echo signal of the sensor. Upon activating the sensor, the interrupt is first fired when the echo signal goes high. Once that happens, the current value of the 16 Bit counter, which is enabled and set to the 8 times prescaler, is saved. The next time the interrupt is triggered will be when the echo signal goes back down. When this happens, the timer value is saved again and another variable is set to one. Analogous to reading the MPU, this variable is used inside the main loop to determine the availability of sensor data. If this variable is set, the difference of times, i.e. the duration the echo pin has been high, can be sent to the laptop.

To simplify evaluation at the laptop side, values are sent with a prefix indicating the type of sensor, followed by a colon, a space, the value to be transmitted and a line break sequence.

While the readings from the ultrasonic sensor are dependant on temperature, corrections for this are not in scope for this project, as we are not interested in exact distances but rather in the ratio between distances at various angles.

3.2 Laptop

The reasons I choose Typescript for the Laptop-side software are that this is the language I have the most experience with and that, when used in the context of a webapp, it facilitates easy visualization of data. This is especially helpful for debugging, but comes at the price of performance, which I assume is lower than what can be achieved with other languages such as C.

Proxy Because a webapp cannot directly access the serial port, I wrote a proxy script, which reads from the serial port and provides received data via the WebSocket protocol.

Detecting a Rotation Detection of a 180 deg rotation from left to right is implemented via a state machine. The states can be seen in Figure 4. The relevant sensor values for this detection is the X value of the gyroscope on the MPU6050. Because the values are not 0 even when the board is completely still, I decided to use the first 50 received values for calibration. This means that the board should be stable for the first about 2 seconds (see chapter 4.1) after connection with the laptop is established. Those values are used to calculate mean and standard deviation.

If a received value is within x times the standard deviation of the mean, the platform is considered to not be turning. If the value is above or below the threshold, the state machine goes to the $Over/Under$ states respectively. While in the $Over/Under$ states, a counter is maintained, which is increased for each measurement outside the threshold and decreased if the platform is stable again. If the counter reaches 0, the state returns to the standard, i.e. $Steady$. If the counter reaches a threshold, the state changes to $Over\_Steady$ or $Under\_Steady$ depending on previous state. While these additional states are not currently utilized, they allow to further customize the detection behavior.

While in the $Over$ or $Over\_Steady$ states, if value significantly under mean is detected, state is instantly reverted back to $Steady$ and the current rotation ends. Same thing happens for values significantly over mean while in $Under$ or $Under\_Steady$ state

Start of rotation is therefore detected when the state changes away from $Steady$ and end of rotation upon returning to $Steady$. For detecting the length of rotation, the sensor values are summed up while state is not $Steady$. After a rotation ends, this value can be compared against thresholds to determine what direction this turn was (corresponds to the sign of the sum) and how big the turn was (corresponds to the absolute value of the sum). Given the known frequency and sensitivity of the sensor, this sum can also easily be turned into an estimate of distance of rotation, see Equation 1. For this project, range is

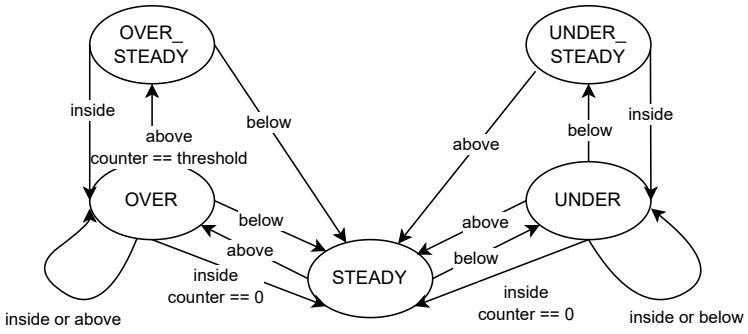


Figure 4: States of the turn classification

250, since the sensor is configured to use the $\pm 250\,\mathrm{deg}$ s range. Frequency was determined to be 31.5, see 4.1.

$$(sum * range)/(2^{15} * frequency) \tag{1}$$

Capturing Ultrasound Data Every reading of the ultrasonic reading that comes in while the device is in rotation is saved together with the current angle of rotation, i.e. the sum of gyroscope readings. Additionally, a buffer is kept to include the last n (5) values once the start of a turn is recognized.

Normalizing Data Due in part to the variable update rate, which is discussed in Section 4.1, the captured readings need to be normalized. This is done by using the sum of gyroscope readings starting from the beginning of rotation as x-coordinate for the current distance reading. The total sum of gyroscope readings for a rotation then corresponds to $180\,\mathrm{deg}$ and other values can be scaled accordingly. While the gyroscope readings give a speed of rotation rather than a distance, a Sum should still work given the very steady frequency of updates from the MPU (see chapter 4.1). Due to this normalization, a steady speed rotation is not required for detection. The scan is however more reliable when performed slower, because more points are measured and combined. For distance of rotation would be speed of rotation multiplied by time, if we have constant time a simple sum should suffice. Distances are also scaled to a range between 0 and 1 for the shortest and longest measured distance respectively.

Evaluating Data Once the end of a turn is registered by the state machine, we have all the data necessary to evaluate the junction at hand. Classification

happens by comparing the List of x, y objects, where x is the normalized angle and y the normalized distance, to known profiles of T-Junctions, X-Junctions and Corridors.

For this, the data is converted to a discrete array. For every integer x from 0 to 180, all data points are collected whose angle starts with x. Then, the average of their distance measurement is calculated. This leaves us with a list of up to 180 numbers, as not every angle may have been recorded. We now calculate the correlation between this series and a series of reference value for each of the junction types, where missing data in either series is removed from both.

This leaves us with correlation values for each junction type and we can return a list of junction types sorted by how likely this type is to correspond with our observation. This means that the first value in the returned list is the best fit and therefore our evaluation result.

For the reference data, I first wanted to use theoretically computed data, as can be seen in Figure 5, however this did not work well. Therefore I decided to use empiric data created by adding measurements from several scans together and normalizing them. The resulting data can be seen in Figure 6.

A different approach I tried but decided to discard in favor of the approach described above used KMeans clustering. One of the benefits was the vastly reduced amount of data that needed to be saved. The idea was to generate 5 clusters from the scanned data via KMeans. The center of these Clusters could then be compared to the center of the clusters for the reference data. Thus for each of the three reference classes only the center of the five clusters would need to be saved, instead of the 180 data points. However, this did not work reliably. Maybe this approach could be used in the future, if clusters are calculated multiple times and distances are then combined.

Regardless of classification method is used, upon classifying the Junction the webapp adds a new chart to the page that includes the classification result and plots the normalized data.

Extensibility The code can easily be modified to include different handlers for the sensor data. The best place for this is the switch statement in the main.ts file, where the prefix is used to differentiate between sensor types. For example, the code includes a function that can be used to live-plot values as soon as they arrive.

The junction classification could also easily be modified to not only return the sorted list of junction types, but also the correlation values, which could be used as confidence values.

4 Analysis

Detection of turns Detection of turns works very well. The thresholds for detecting a turn from left to right, i.e. a 180 deg clockwise turn, have been set to -170 deg and -210 deg. When a the end of a turn is detected, and the sum

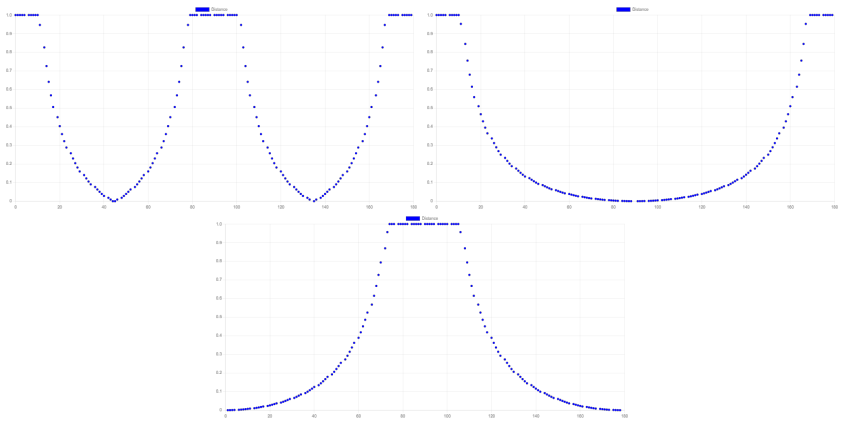


Figure 5: Expected Scan data for X-Junction, T-Junction and Corridor

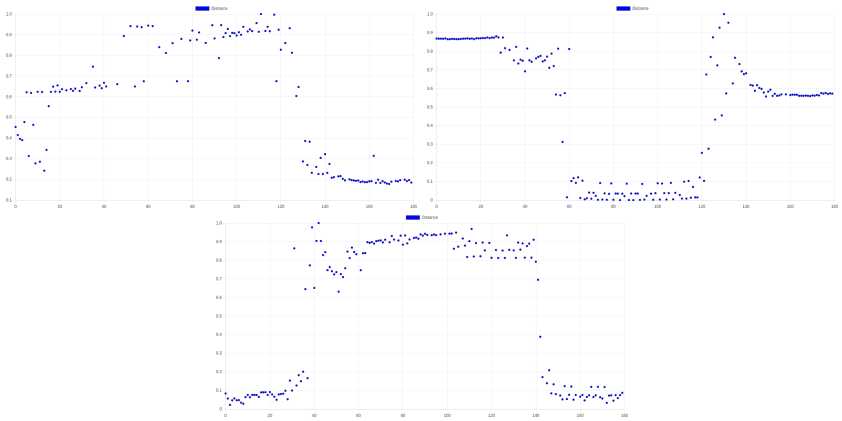


Figure 6: Empiric data for X-Junction, T-Junction and Corridor. Mixture of several scans, used as reference in classification

	Measured Distance	Gyroscope	Ultrasound
10	10.33490	31.54639	602.1531
20	20.25844	31.59596	441.9388
30	30.32993	31.59184	347.5556
50	50.38063	31.59794	252.7551
75	75.66509	31.60204	187.5859
100	99.08760	31.60204	147.4949
150	150.85837	31.60204	103.9596
200	203.61689	31.60825	79.0102

Table 1: Expected and measured distance in cm from sensor and update frequency of sensors, average over 100 seconds

of the turn translates to a value between those bounds, the turn is deemed to have been valid, i.e. a turn from left to right. The target of 180 deg is not at the center of the interval, because experiments showed it to work more reliable that way. Reliability in this context refers to the detection rate for turns that feel like they should have been valid. Because the direction of the turn is regarded as well, measurements can be easily taken and the platform can be rotated back to its initial position without triggering a new measurement (if the platform is turned counter-clockwise in order to go back to the initial position).

Detection of Junctions Overall, detection works pretty well in the scenarios I tested. If misclassifications happened, it was mostly with the X Junction. This is probably due to the higher variance a X Junction has compared with the other two types.

To help with debugging classification, the webapp also includes a section that can generate expected scan data and run the junction detection on this data. This is the same utility that was used to generate the theoretic data before, see 5. The classification with the generated data as input pretty reliably, apart from the corridor scenario, which is sometimes misclassified as X Junction when the corridor width is too small.

4.1 Benchmarks

Frequency of Updates To determine the frequency of updates, values received per second where averaged over 100 seconds, although the data from the first and last second were ignored due to the potential of partial data. The rate of updates from the gyroscope is very steady, at about 31.6 Updates per second. The frequency of new values from the ultrasonic sensor is not as steady, as it depends on the distance being measured. This is because longer distances take longer to measure, and the arduino immediately starts a new measurement upon completion. Figure 7 shows a plot of the measured values, also presented in Table 1.

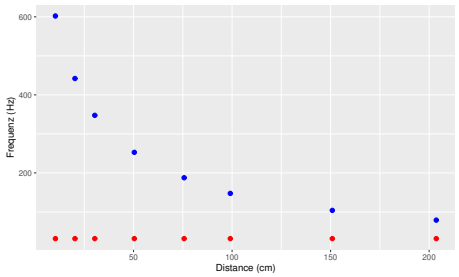


Figure 7: The data from 1 visualized

4.2 Known Problems

The following sections discusses some of the problems I faced, and solutions or mitigations I employed.

Incorrect Ultrasonic Measurements during I^2Ccommunication During an earlier stage of the project, Ultrasonic measurement received immediately after Gyroscope reading were off by a lot. While this does not seem to be an issue anymore, I am neither sure what caused this nor why it works now. Potential mitigation strategies for this would have been laptop sided, were values received right after gyroscope data could either be dropped or substituted by a suitable average.

Unknown path width Classification could be much simpler if path width was known. Three distance measurements would suffice for classification, one at 0, one at 90 and one at 180 Degrees. A X Junction would have all three measurements larger than path width, T Junction would have first and third measurement larger, second measurement equal to path width (technically more like half path width) and a corridor would just have the second measurement greater than path width. My first idea was to use the distance measurements at 45 deg and 135 deg, where the corners would be expected for a X Junction, to estimate this variable. This did not turn out to be a workable approach for several reasons. First, the ultrasonic sensor fails to get reliable readings for the distance if the object is at too much of an angle. This could be mitigated by the assumption that corners are not sharp, but rounded, as may be the case with pipes. This improves corner detection greatly. A workaround for rounding corners can be seen in 2. Secondly, this all turned out to not matter, as it would only be useful for X-Junctions, due to the other two types not having the corners necessary.

With the current method of classification, based on similarity to past data, this is not needed at all as the data is normalized.

Graphing the environment from the scan data Probably as a result of the difficulty to get measurements at an angle, the data cannot be easily graphed to show the scanned environment. Considering our data consists of angles and distances, it can be understood as polar coordinates. Therefore, converting it to cartesian coordinates and plotting those should give a pretty good image of the junction. This works very well in theory, but not as well with values actually measured by the sensors, see Figure 8. This is not of much importance to the overall project, but would have been helpful for visualization.

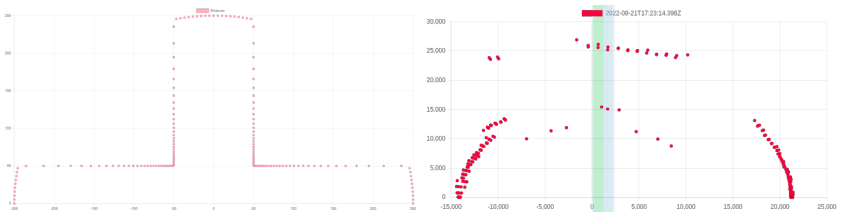


Figure 8: Left: simulated data, converted to cartesian coordinates and graphed. Right: actual data, converted to cartesian coordinates and graphed. Notice the missing sections, that are mostly obstacles at an angle.

Turns are not detected correctly When a turn ends, the calculated angle is logged to the console. If this is off, the most likely reason for this that the platform was not stable during calibration. To fix this, simply reload the app and make sure the platform does not turn for one or two seconds, until the calibration is done.

5 Summary

This paper describes the process of building a platform for distinguishing x junctions, t junctions and corridors via a gyroscope and a ultrasonic distance sensor. Turns of the platform are reliably detected via a state machine. Upon detection, data that has been recorded during the rotation is used to compare the scan of the environment against previously recorded data, thus allowing a classification.

The source code for this project is supplied with this paper and can additionally be found at `https://github.com/layaxx/s6_PhysicalComputing`. This includes a readme file with instructions on how to use the code.

References

[1] Frank Kirchner and Joachim Hertzberg. “A Prototype Study of an Autonomous Robot Platform for Sewerage System Maintenance”. In: *Autonomous Robots* 4 (May 1997). DOI: 10.1023/A:1008896121662.

Appendix

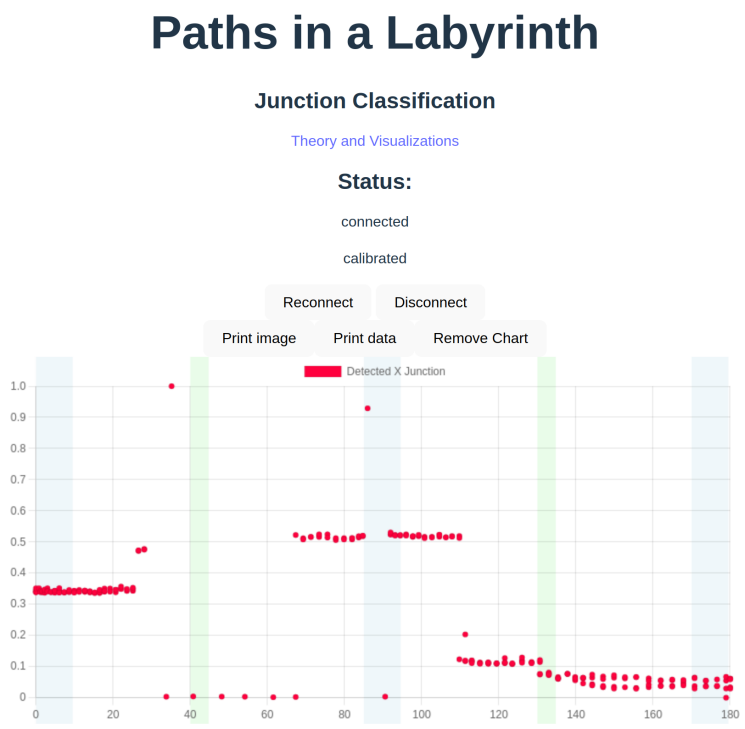


Figure 9: Screenshot 1: Successfully detected a X Junction

Visualization

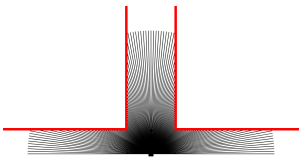
Back to main App

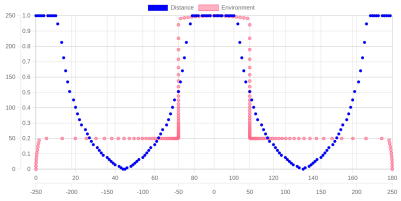
X Offset 50

Y Offset 50

Line Length 250

Path width: 100



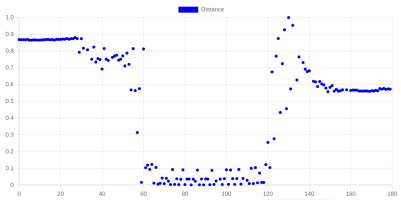


Classification: X Junction

Print to console

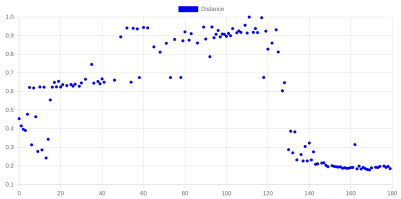
Combined reference data

T Junction



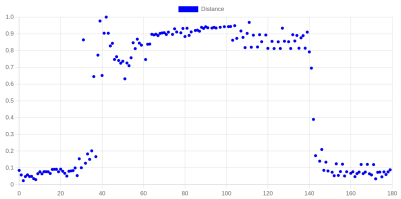
Print to console

X Junction



Print to console

Corridor



Print to console