Sensor Reliability Analysis for Obstacle Avoidance

Allison Aprile
ISYE 6420
April 21, 2024

Introduction

Mobile robots can be equipped with a variety of sensors to interact with their environment and accomplish their objectives. For example, an odometry sensor is useful for localization, while a camera is useful for downstream perception tasks, such as objection detection. Many sensors can be applicable for navigation tasks, including range (e.g. infrared, ultrasonic), visual (e.g. camera), or position (e.g. GPS) -based devices. [1] Sensor selection is nontrivial; a number of factors must be considered, for instance environment conditions (e.g. light, terrain), additional sensor feedback, cost, noise, and more. While the sensors themselves may not incur much cost, calibrating and testing the sensors for reliability can be expensive, especially for more high-stakes tasks such as obstacle avoidance. Therefore, simulations facilitate the development and testing of almost all robotics applications, and are particularly helpful for sensor selection.

In this project, we perform an analysis on the reliability of range sensors for obstacle avoidance. Specifically, we observe the time-to-collision (measured in steps) of autonomous robots equipped with an infrared versus ultrasonic sensor. Both sensors return the distances from the transceiver to the nearest object, although can have difference performance subject to conditions. Infrared (IR) sensors emit infrared light and measures the amplitude of this light reflected from an object, but therefore can be limited in environments with infrared interference, such as sunlight or incandescent lighting. Ultrasonic (US) sensors use the same methodology but with sound waves, which is notably more accurate but slower than IR [2] [3].

The experiment will involve robots navigating a closed map at-will (i.e. without a map or plan) using a naive obstacle avoidance algorithm. The robots will have randomized starting poses and operation speeds. First, we will model the time-to-collision based only the ‘sensor’ variate. Then, we will expand the model to include all measured variates (i.e. sensor, speed, initial pose), with the goal of further observing how the environment affects the reliability of the sensor. We will recommend the more reliable sensor based on the results of this experiment.

Data

The data for this experiment was procured through a series of simulations created with the Python Jyro Simulator [4]. The simulation process included creating a map, robot, and defining simulation conditions.

Map

The maps were defined using 25x25 binary arrays, which were then translated to box Cartesian coordinates for simulation use. Map 4 was ultimately used in the simulation because it allowed

for better observation of collisions; while collisions were occurring in Maps 1-3, early simulations showed most robots operating within a small, clear radius of the map and never colliding. This caused an issue with excessive right-censored values, particularly in the superior ultrasonic-equipped robots. Maps 1-3 have an obstacle area to map area ratio of 0.2432, 0.3072, and 0.3712, respectively, while Map 4 has this ratio equal to 0.4096.

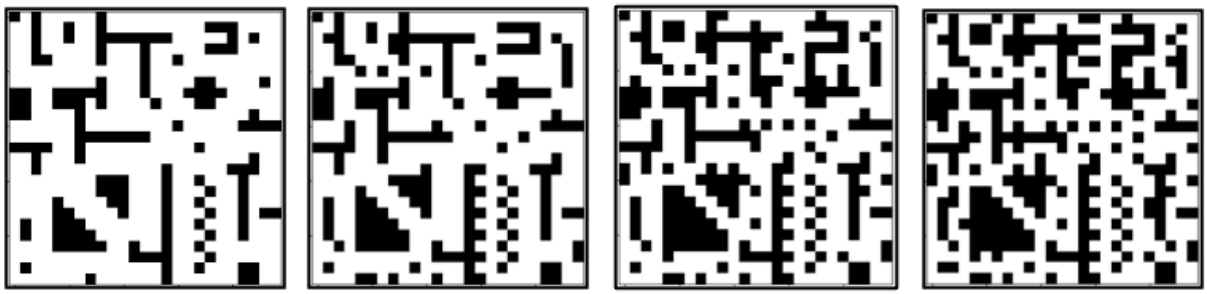

Figure 1. Map iterations: from left to right, Map 1, Map 2, Map 3, and the reported Map 4.

Robot

The simulated robot is based on the Pioneer 3-DX, which is a two-wheel, two-motor differential drive robot [5]. For the simulation, we equipped it with the standard Pioneer eight front and eight back sonars (although only the front eight were activated) and the two Myro infrared sensors.

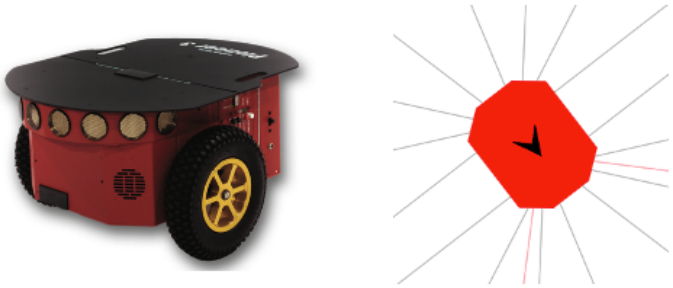

Figure 2. Left: Pioneer 3-DX [5]. Right: Jyro simulated Pioneer 3-DX with 16 ultrasonic (black) and two infrared (red) sensors.

For this experiment, the simulated robots are equipped with both sensors (but only one activated on instantiation) with a noise value of 0.1. They are programmed with the same naïve obstacle avoidance algorithm, denoted the ‘brain’:

```
get sensor data

if min(sensor distance) < 0.5:
      rotate clockwise at a random speed between 0.0 and 1.0
      if turn steps > 500:
            change rotation direction
else:
      move straight at robot speed
```

Algorithm 1. Naïve obstacle avoidance.

In summary, the 'brain' will check the minimum distance reading (in meters) from the onboard sensor. If it is below the collision threshold of 0.5 meters, it will rotate at a random speed. Otherwise, it will move straight at the assigned robot velocity. To avoid the behavior of staying in the vicinity of the map, the robot is designed to change its rotation direction after 500 turning steps.

The robots have a pre-installed collision check system, denoted by the 'stall' variable. This indicates if the robot is stuck (i.e. unable to make any further movements) or has collided with an obstacle.

Simulations

15 trials were simulated for each sensor type, resulting in a total of 30 data points. For each trial, a robot was initialized with a random pose at any non-obstacle location within the map, having random heading between 0 and 2π. One of the two onboard sensors were activated and the robot was randomly assigned an operating speed from 3.0, 4.0, 5.0, 6.0, 7.0, and 8.0. These values were decided from trial and error, specifically with the attempt to avoid excessive right-censored data points; essentially, lower operating speeds (1.0 and 2.0) would be 'too safe', with the robot never colliding in the observed timeframe), while higher operating speeds (9.0+) would result in instant collisions.

Each trial ran for 1000 steps, which can be translated to the operating time of the user's choice. For simplicity, we left our observations in the step unit. We set the end-of-study time at 300 steps (a choice divisible by 60 for easier minute/second, etc., conversion), but continued observing subjects for an additional 700 steps. If a trial resulted in an instant collision (i.e. survival equal to 0 steps), it was disregarded.

For testing, we ran an additional two simulations for 1000 steps, with both robots having equivalent starting poses and speeds but different sensors. As with the aforementioned simulations, the end-of-study time was set to 300 steps.

Each trial's robot sensor, speed, starting pose, time-to-collision (in steps), as well as its stalled status and censoring indicator, were recorded and compiled into a dataset.

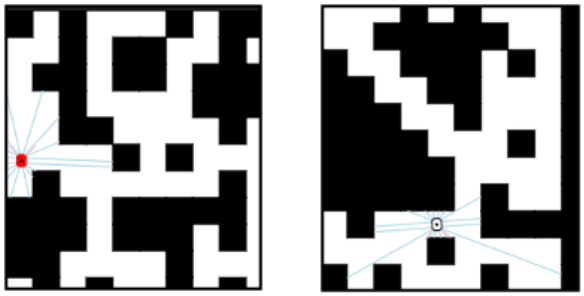

Figure 3. Left: Simulated robot without stall. Right: Simulated robot with stall (collision).

Models

To perform reliability analysis of the ultrasonic versus IR sensors, we model time-to-event (here, time-to-collision in steps) of the robots. We use PyMC to model this time-to-collision given a) only the 'sensor' variate; and b) all the measured variates (i.e. sensor, speed, and initial pose), with the goal of further observing how the environment affects the reliability of the sensor.

We use the Weinbull Distribution to model this time-to-collision, T, parameterized in PyMC as:

$$T|\alpha, \mathrm{B}, x_i, \beta \sim Weinbull(\alpha, \mathrm{B})$$
$$\alpha = \gamma > 0, \ \mathrm{B} = \lambda^{-1/\alpha} > 0$$
$$(F(t|\alpha, \mathrm{B}) = 1 - e^{-(t|\mathrm{B})^{\alpha}}, for\ t > 0)$$

$$g(\lambda) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$$
$$\beta \sim N(0, \sigma_0^2)$$

[6]. We set a slightly non-informative Exponential prior with $\lambda = 1$ on α, as this is required to be positive for the Weinbull distribution. For the intercept(s) and slope (β), we select more non-informative Normal priors, centered at 0 and with a small precision ($\tau = 0.01$). We use the natural logarithm link function $g(\cdot)$, with its inverse being the exponential function, to convert the linear combination in our data to a proportion value between 0 and 1.

The time-to-collision variable (steps) is right-censored for three samples, therefore having Missing-at-Random (MAR) (also called CAR in the survival analysis context) ignorable missingness [7]. This is handled by PyMC with the censoring indicator included in the data.

Results

All model traces indicate an $\hat{r} < 1.01$, which allows us to be fairly confident that we have good sampling coverage.

Before running the simulations, we hypothesized that the IR-equipped robots would be at disadvantage; they are outnumbered by the ultrasonic sensors, which provides a wider range. Without modelling, the simulation data clearly shows a higher time-to-collision for the ultrasonic-equipped robots. This is confirmed by the modeling outcomes.

Model 1 (the 'sensor' only model) results in median survival times (in steps) of 25.947 and 276.952, and 95% HPD credible sets of [8.512, 49.679] and [72.220, 575.808], for the IR versus sonar-equipped robots, respectively. This pattern is replicated in the Model 2 results, which are broken down by sample due to the multivariate inputs:

	mean	sd	hdi_2.5%	hdi_97.5%
ir_median[0]	23.659	29.064	1.278	67.596
ir_median[1]	34.399	30.489	1.995	86.968
ir_median[2]	49.599	47.801	3.385	127.402
ir_median[3]	38.318	22.452	8.386	81.240
ir_median[4]	24.781	15.725	3.328	53.209
ir_median[5]	60.018	66.128	2.614	163.779
ir_median[6]	50.055	43.798	5.099	123.077
ir_median[7]	33.543	46.470	0.803	99.759
ir_median[8]	28.530	30.758	2.084	75.562
ir_median[9]	51.256	36.866	6.219	116.523
ir_median[10]	49.639	45.565	5.001	128.475
ir_median[11]	31.449	21.877	4.426	70.464
ir_median[12]	64.316	55.376	6.974	154.002
ir_median[13]	23.989	18.740	2.540	57.907
ir_median[14]	33.815	30.487	2.173	86.557
sonar_median[0]	1058.161	1707.712	35.178	3263.365
sonar_median[1]	297.354	415.771	19.445	873.409
sonar_median[2]	309.356	274.690	42.383	754.000
sonar_median[3]	214.675	251.141	10.520	590.426
sonar_median[4]	178.737	231.571	9.860	525.840
sonar_median[5]	230.094	210.696	22.937	581.821
sonar_median[6]	571.389	683.485	39.941	1556.753
sonar_median[7]	326.835	419.076	17.179	956.102
sonar_median[8]	238.149	423.069	17.104	700.399
sonar_median[9]	344.511	457.114	30.945	907.149
sonar_median[10]	1286.422	2201.718	78.613	3846.822
sonar_median[11]	152.482	159.537	16.422	413.691
sonar_median[12]	188.955	152.946	24.394	461.726
sonar_median[13]	379.599	466.468	25.786	1019.013
sonar_median[14]	680.229	1140.673	15.767	2049.053

Figure 4. Model 2 (multivariate) trace.

Further, to better observe the effect of the additional variates on these statistics, we sampled the value for the posterior median of the testing samples' time-to-collision. For Model 1, this is equivalent to the aforementioned (25.947 and 276.952, and 95% HPD credible sets of [8.512, 49.679] and [72.220, 575.808], for the IR versus sonar-equipped robots, respectively). However, for Model 2, we observe higher medians and wider credible sets for both sensors: for the IR robot, median of 58.945 and 95% HPD credible set of [6.809, 146.957], while for the sonar robot, median of 552.450 and 95% HPD credible set of [41.185, 1503.705].

Clearly, the initialization of the robot's pose within the map, as well as the speed, affects the measured reliability of the sensor. By analyzing the more detailed Model 2 results, it is evident, for example, that the higher speeds and/or initialization in more-confined areas of the map typically result in lower survival time. However, it is still evident that the ultrasonic sensor is superior, at least in the simulated testing environment.

Therefore, we recommend the ultrasonic sensor.

Conclusion

Sensor selection for robotics is nontrivial; with so many options available for any task, it can be time and cost-intensive to select the right option. Simulating the different sensors is one way to evaluate their performance before taking the risk in production. In this paper, we conducted an experiment to analyze and compare the reliability of two distance sensors, namely ultrasonic and infrared, in the task of obstacle avoidance. Through simulations with varying calibrations, we observed a higher time-to-collision (measured in steps) for the robots equipped with the ultrasonic sensors, resulting in our recommendation for such in this particular task.

Future Work

Given the time and simulation constraints, this reliability analysis is not as exhaustive as robotics testing standards would prefer. In future iterations, we would find it beneficial to add additional sensors and use a less-naïve obstacle avoidance algorithm, as well as to run more trials for longer at slower operating speeds. It would also be useful to find a better way to model and incorporate sensor noise. From a statistical standpoint, it would also be interesting to further analyze the effects of each variate on the time-to-collision; for example, some sort of visualization with the map to show areas where collision is likely, which could be later added to the navigation map to enhance the safety measures where needed.

References

[1] Abiola, A. (2023, 16 January). *Types of Sensors in Robotics*. Wevolver. https://www.wevolver.com/article/sensors-in-robotics.

[2] RoboSavvy. (2018). *IR Sensor vs. Ultrasonic Sensor: What is the difference?* RoboSavvy. https://robosavvy.co.uk/ir-sensor-vs-ultrasonic-sensor-what-is-the-difference#:~:text=The%20biggest%20difference%20between%20IR,not%20an%20object%20is%20present.

[3] Robot Platform. (2024). *Types of Robot Sensors.* Robot Platform. https://www.robotplatform.com/knowledge/sensors/types_of_robot_sensors.html.

[4] Blank, D. (2017). *Jyro Simulator*. Jyro. https://jyro.readthedocs.io/en/latest/.

[5] Adept Technology. (2011). *Pioneer 3-DX*. Adept Technology. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf.

[6] Reding, Aaron. (2023). *1. Basic Distributions*. Aaron Reding GitHub. https://areding.github.io/6420-pymc/unit4/Unit4-basicdist.html

[7] Atkinson A, Kenward MG, Clayton T, Carpenter JR. Reference-based sensitivity analysis for time-to-event data. *Pharmaceutical Statistics*. 2019;18:645–658. https://doi.org/10.1002/pst.1954.

Accompanying Files

- Project_Analysis_Aprile.ipynb: Reliability (time-to-collision) analysis of distance sensors using PyMC
- Project_Simulation_Aprile.ipynb: Dataset creation for sensor reliability (time-to-collision) analysis
- maps:
    - create_map.ipynb: Demo for using the Map helper class to create and visualize maps for the Jyro simulation
    - map.py: helper class for creating, visualizing, and displaying maps for simulation
    - saved_maps: Numpy files of Maps 1- 4