Winter 2026
Robot Mapping Workshop

https://existentialrobotics.org/RobotProvingGrounds/

CONTEXTUAL ROBOTICS INSTITUTE

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Agenda

5:05 - 5:20: Introduction to ERL

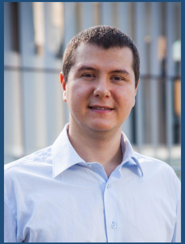
5:20 - 6:35: Mapping Algorithm Presentation and Activities

6:35 - 6:50: Pybullet Simulation (optional)

6:50 - 7:00: Live Robot Demo and Food

Who are we?

Faculty Mentors



Nikolay Atanasov

Shatha Jawad

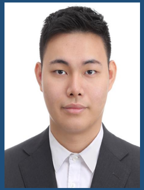
PHD Student Mentors

Nikola Raičević

Mani Amani

Yinzhuang Yi

Undergraduate and K12 Students



Hyungjun Doh

Afraz Hameed

Ryan Teck

Stephen Huang

Allie Dinh

Lek Man

Steven Hsiao

Tarun Jaikumar

Past Contributors

- Eric Zheng
- Ali Hussain
- Muhammad Fadli Alim Arsani
- Shuyan Tan
- Andrew Nemeth
- Darren Ng
- Zeyu (Jeffrey) Chen
- Shuyan Tan
- TanMuhammad Arsani
- Risab Sankar
- Chengkai Yao
- Aniket Bhosale
- Minghan (Travis) Wu
- Chakshan Kothakota
- Fayyad Hassan
- Kibum Kim
- Ali Hussain
- Yaobang Deng
- Weifan Ou
- Trung Tran
- Aditi Krishnakumar
- Anthony Hirales Ahuatzin
- Peter Stratton
- Shreyas Arora
- Hannah Hui
- Farnia Nafarifard
- Ryan Goh
- Aaron Yu
- Rohan Bosworth
- Ke Ou

What We Do

- Teach basics of various robotics algorithms
- Ex: Mapping, Localization, Planning, and Control

https://existentialrobotics.org/RobotProvingGrounds/

What is Mapping?

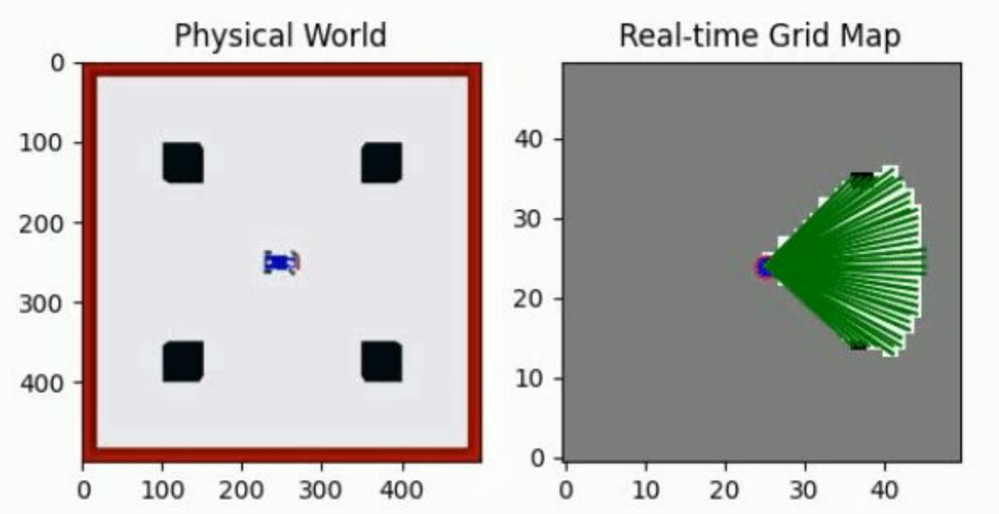
Mapping Problem

Roomba

Waymo

- For full autonomy, robots need to know what is in their environment
- This allows for many possibilities (exploring dangerous environments, providing services, etc.)

OGM Mapping



- Occupancy Grid Mapping (OGM)
- Uses sensors to create a map of the world
- Map tells if a space is filled, empty, or unknown

Light Detection and Ranging (LiDAR)

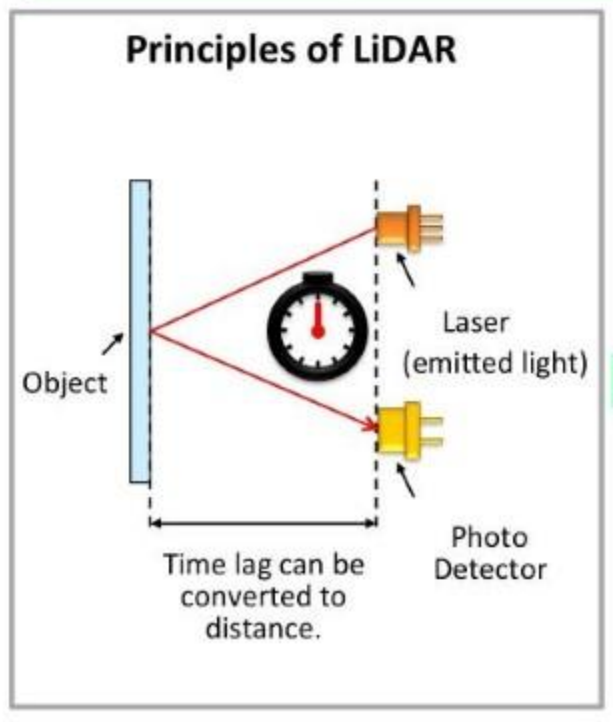
- Emits lasers and measures return time off reflections.

- LiDAR does have a max distance

- LiDAR spins while emitting lasers at high speeds "see" surroundings

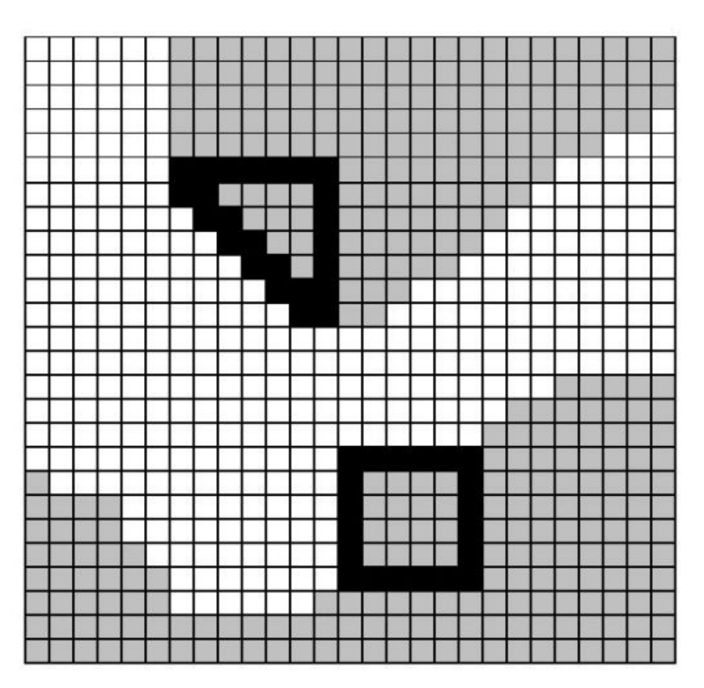
- Measurements:

  Angle & Distance



Map Updates

- Stores map in a matrix
- Uses sensor data to mark filled cells
- Space between robot and sensor hit is marked as empty
- Draws empty spaces with line drawing algorithm



Line Drawing (Bresenham)

Line Drawing

Basic Line Drawing (Python)

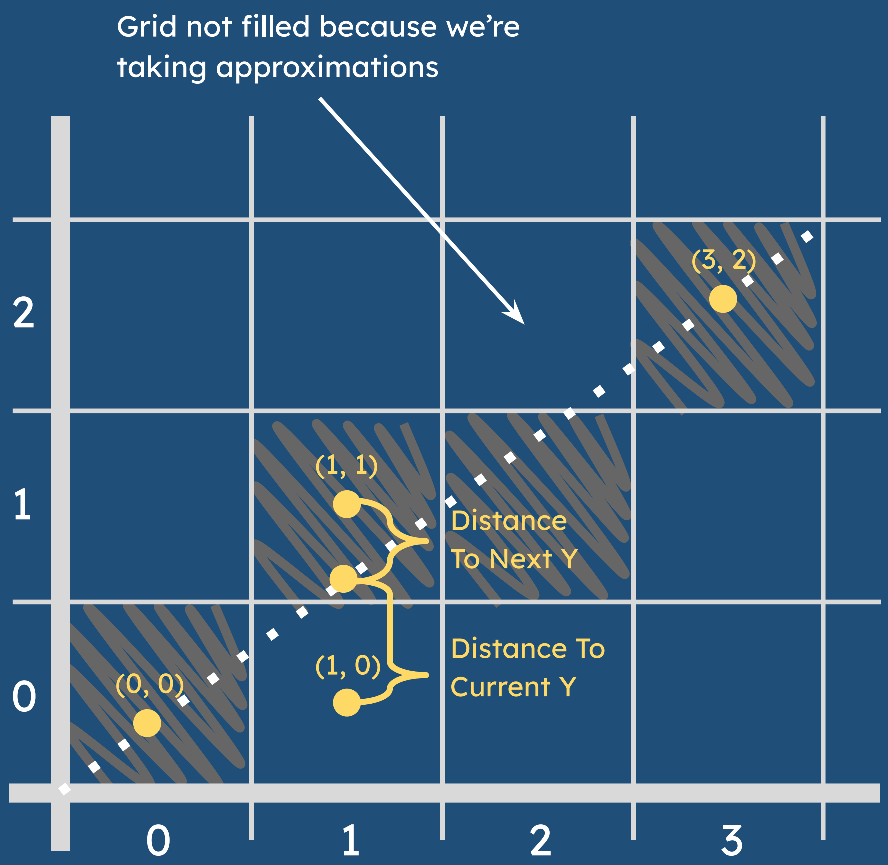
```
def draw_line(x0, y0, x1, y1):
  m = (y1 - y0) / (x1 - x0)
  dx = abs(x1-x0)

  for i in range (dx):
      y = round(m * i + y0)
      print(dx + i, y)
```

Based off: **<u>“y = mx + b”</u>**

Finds Slope

Uses “y=mx + b” to find “y”
Round decides which whole integer “y” should be

Line Drawing



- Using decimals can slow down code
- Variation of Bresenham Algorithm Solves This!
- When To Increase Y:
  - When the real y-value is closer to the next integer
- We'll keep track of this choice with a **decision variable** called **"D"**

Line Drawing

y_0 = starting y-value
y = current y-value
i = x-distance moved

$f(i+1) = m(i+1) + y_0$
↼ function to get y-coord based on x-distance moved

1 D = Distance To Current - Distance To Next

2 $= [\ f(i + 1) - y\ ] - [\ y + 1 - f(i + 1)\ ]$

3 $= [\ m(i + 1) + y_0 - y\ ] - [\ y + 1 - (m(i + 1) + y_0)\ ]$

4 $= 2m(i + 1) + 2y_0 - 2y - 1$

5 $= 2(\Delta y/\Delta x) * (i + 1) + 2y_0 - 2y - 1$

Interpreting D's results

When D is positive

- We're going to be closer to the next y-value and thus increment

When D is zero

- We're going exactly to the next y-value and thus should increment

When D is negative

- We're going to be closer to the current y-value and thus should stay

Line Drawing

1 $$D = 2\left(\Delta y/\Delta x\right)^{*}(i+1) + 2y_0 - 2y - 1$$

2 $$(\Delta x)\,D = 2(\Delta y)(i+1) + 2(\Delta x)\,y_0 - 2(\Delta x)\,y - (\Delta x)$$

3 $$D = 2(\Delta y)(i+1) + 2(\Delta x)\,y_0 - 2(\Delta x)\,y - (\Delta x)$$

4 $$D = 2(\Delta y)\,i + 2(\Delta y) + 2(\Delta x)\,y_0 - 2(\Delta x)\,y - (\Delta x)$$

Since we only care about sign of D, we can forgo the Δx on the left side

Line Drawing

Pseudocode Recap!

- Intake x0, x1, y0, and y1
    - Calculate dx & dy
    - Set y to y0
    - Loop through x-axis
        - Mark current coords
        - Calculate Decision Var
        - Check if D is non-negative
            - If yes, increment y

Line Drawing

What if we calculate D earlier?

- Intake x0, x1, y0, and y1
  - Calculate dx & dy
  - Set y to y0
  - Calculate Decision Var
  - Loop through x-axis
    - Mark current coords
    - Check if D is non-negative
      - If yes, inc y
    - Update Decision Var

Line Drawing

What if we calculate D earlier?

- Intake x0, x1, y0, and y1
  - Calculate dx & dy
  - Set y to y0
  - Calculate Decision Var
  - Loop through x-axis
    - Mark current coords
    - Check if D is non-negative
      - If yes, inc y
    - Update Decision Var

$$D = 2(\Delta y)i + 2(\Delta y) + 2(\Delta x)y_0 - 2(\Delta x)y - \Delta x$$
$$= 2(\Delta y)(0) + 2(\Delta y) + 2(\Delta x)y_0 - 2(\Delta x)y_0 - \Delta x$$
$$= 2(\Delta y) - \Delta x$$

Line Drawing

What if we calculate D earlier?

- Intake x0, x1, y0, and y1
  - Calculate dx & dy
  - Set y to y0
  - Calculate Decision Var
  - Loop through x-axis
    - Mark current coords
    - Check if D is non-negative
      - If yes, inc y
    - Update Decision Var

$$D = 2(\Delta y)\,i + 2(\Delta y) + 2(\Delta x)\,y_0 - 2(\Delta x)\,y - \Delta x$$

SO…

If we don't increment y

$D \mathrel{+}= 2(\Delta y)$

If we do increment y

$D \mathrel{+}= 2(\Delta y) - 2(\Delta x)$

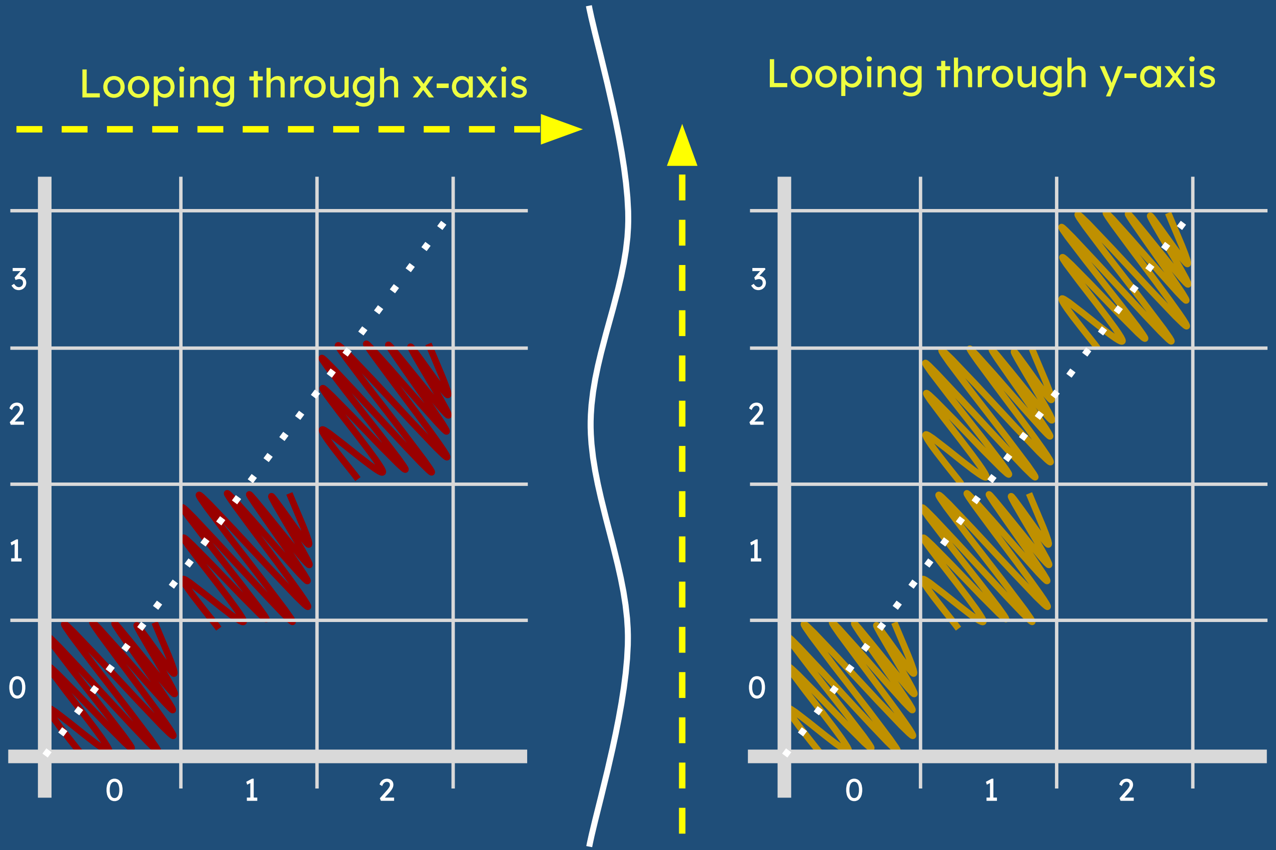
Line Drawing

Potential Problems

1. Slope being negative
2. Direction of Line
3. Slope greater than 1

- Keep track of whether you're increasing or decreasing the x and y axis

Line Drawing



Dominant Variable

If dy is greater than dx:

- Swap all references of x and y

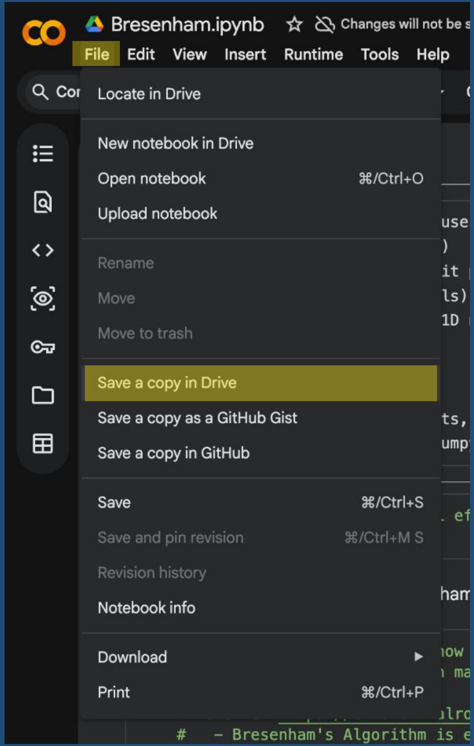
Bresenham Activity

- Go to website
- Navigate to “Workshop” tab
- Scroll down to “Notebooks (During Workshop)”
- Open “**Bresenham**” notebook
- Code! (Instructions in notebook)

https://existentialrobotics.org/RobotProvingGrounds/

Bresenham Activity

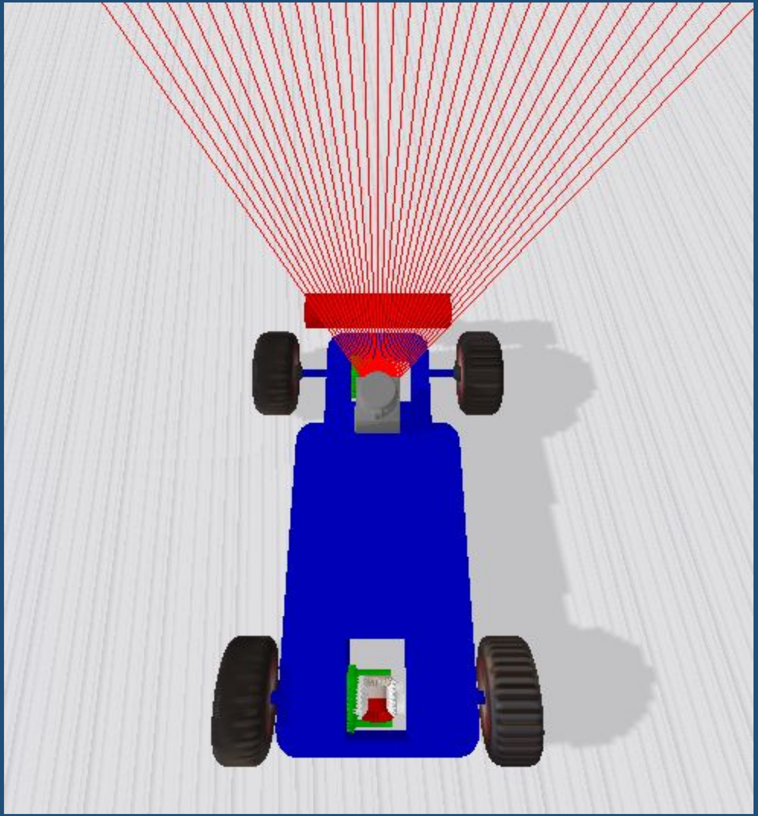
File > Save a copy in Drive



https://existentialrobotics.org/RobotProvingGrounds/

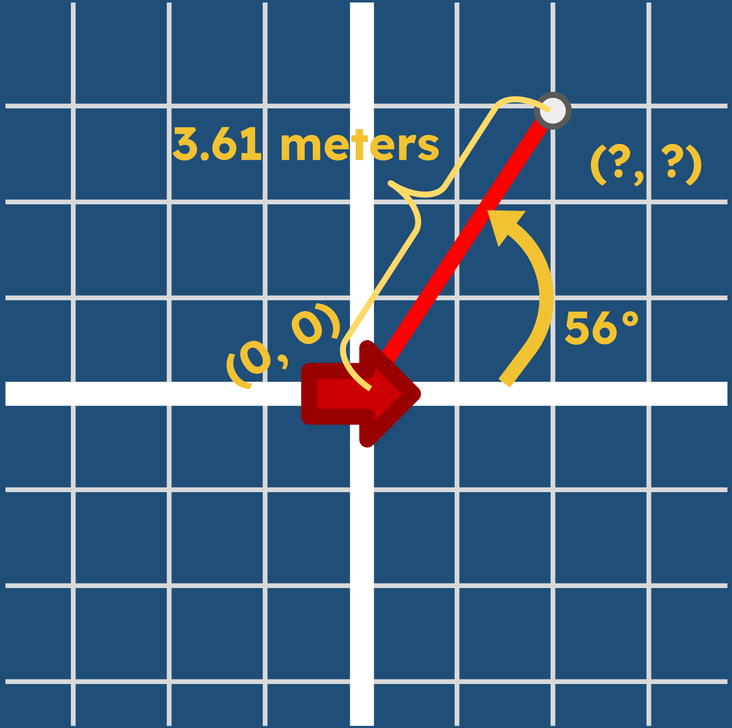
LiDAR

Angle Separation



- Known Specifications of LiDAR
  - Start Angle
  - End Angle
  - Number of rays
  - Max Ray Range (length)
- Angle Separation: Angle between adjacent rays
  - Angle Separation = FOV / # of Rays
- Can determine angle of each ray from LiDAR's perspective

LiDAR Endpoints



*Assume LiDAR faces 0°

What We Know

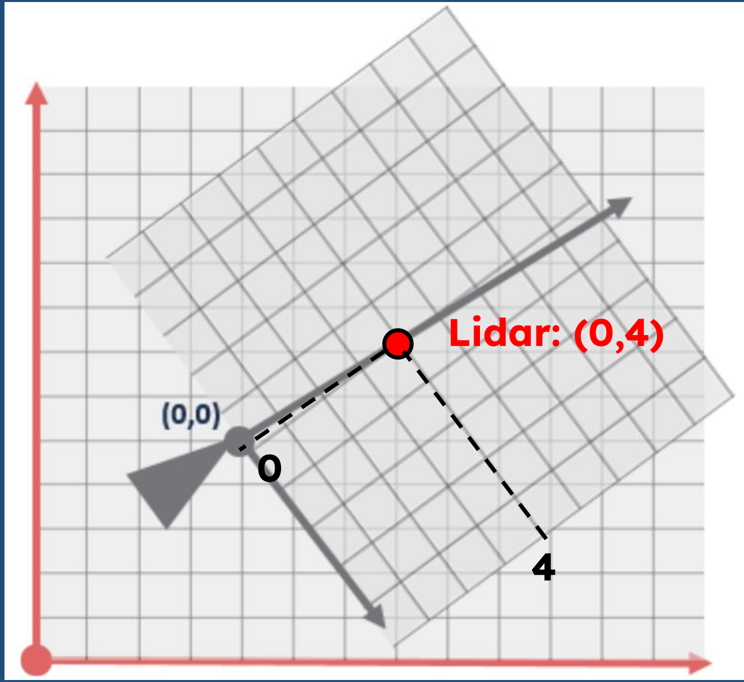
- Ray Angle (θ_L): 56° counterclockwise
- Ray Distance (d): 3.61 meters

- How to map onto grid?
  - Need (x, y)
- Use Trigonometry:

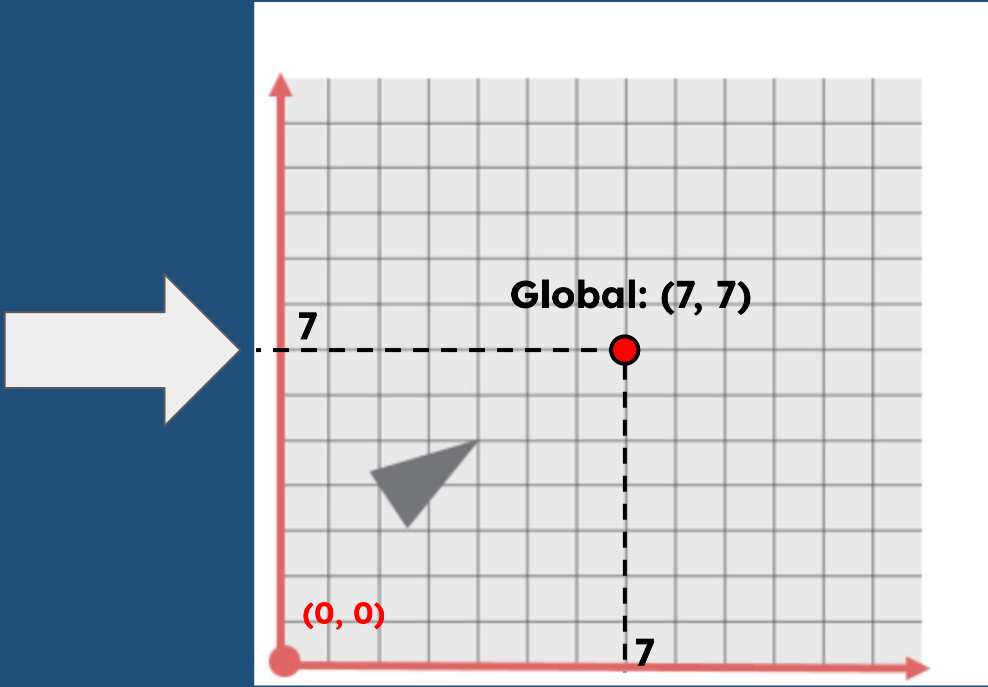
  $x = d * \cos(\theta_L) = 3.61\ m * \cos(56°) = 2\ m$

  $y = d * \sin(\theta_L) = 3.61\ m * \sin(56°) = 3\ m$

Frame Transformations



Robot Frame



Global Frame

Frame Transformations

Finding the position of a point in frame A, knowing:

- The point in frame B
- The relative position of B to A

$$\begin{bmatrix} x_A \\ y_A \end{bmatrix} = \begin{bmatrix} x_{B/A} \\ y_{B/A} \end{bmatrix} + \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x_B \\ y_B \end{bmatrix}$$

$$x_A = x_{B/A} + [x_B\cos(\phi) - y_B\sin(\phi)]$$

$$y_A = y_{B/A} + [x_B\sin(\phi) + y_B\cos(\phi)]$$

x_A & y_A = Point in Frame A

x_B & y_B = Point in Frame B

$x_{B/A}$ & $y_{B/A}$ = Position of Frame B in Frame A

ϕ = Angle of Frame B in Frame A

LiDAR Frame Transformations

$$\begin{bmatrix} x_{\text{world}} \\ y_{\text{world}} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} d\cos(\theta) \\ d\sin(\theta) \end{bmatrix}$$

$$x_{\text{world}} = x_0 + d * \cos(\phi + \theta)$$

$$y_{\text{world}} = y_0 + d * \sin(\phi + \theta)$$

x_{world} & y_{world} = Global Frame

x_0 & y_0 = LiDAR Position

ϕ = LiDAR Angle

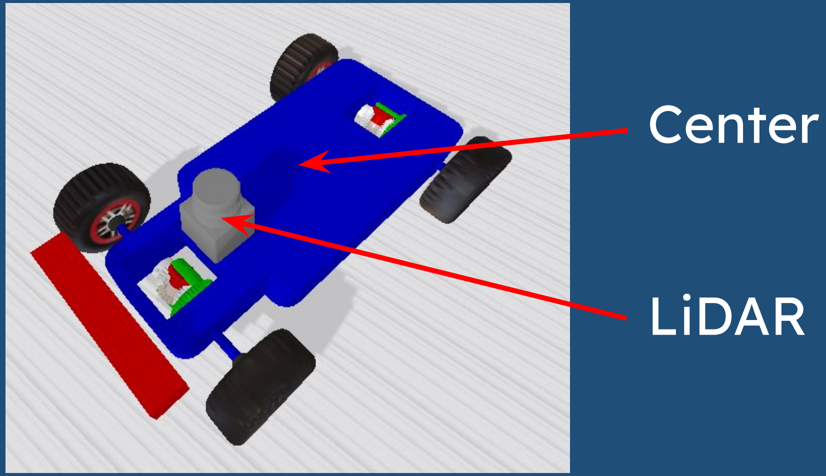
d = Ray Hit Distance

θ = Ray Angle

LiDAR Frame Transformations

What if......

Robot



Frame Transformations

1) LiDAR Frame ⇒ Robot Frame ⇒ Global Frame
    1. Point in LiDAR Frame ⇒ Robot Frame
        - Obtain point in Robot Frame
    2. Point in Robot Frame ⇒ Global Frame
        - Obtain point in Global Frame

2) LiDAR Frame ⇒ Robot Frame (only 1 transformation needed)

Occupancy Grid Mapping (OGM)

Binary VS Probabilistic

Binary OGM

Uses latest sensor data only to identify new hits

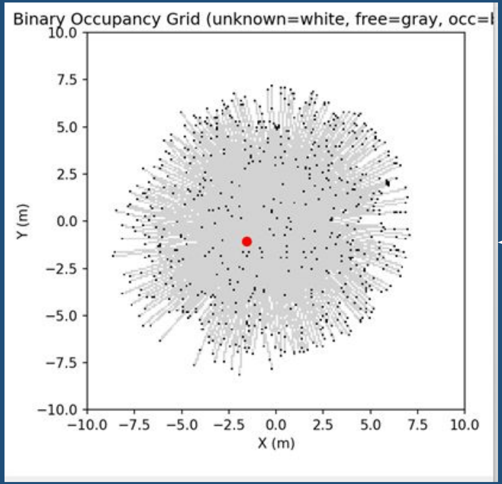
Probabilistic OGM

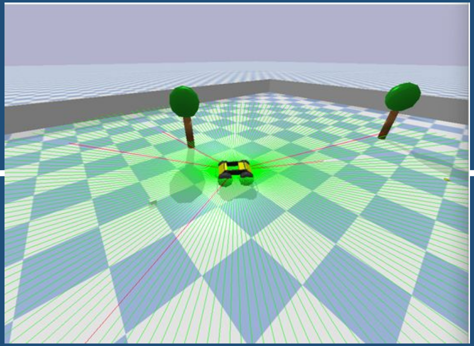
Combines latest and past sensor data to build map

Can Help Partially with Unpredictability (ex: Faulty Readings, Moving Objects/People)

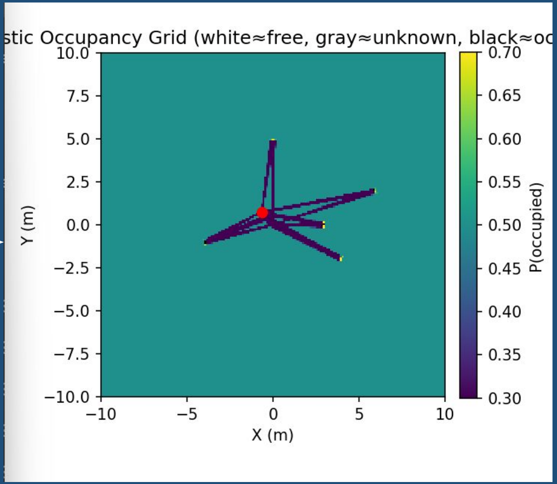
Binary vs Probabilistic Occupancy Grid Mapping (with Sensor Noise)

Binary OGM





Probabilistic OGM



* Noise produces permanent false obstacles *

* Noise is averaged out through evidence accumulation *

Probabilistic

Bayes' Theorem

- Probability of event A occurring given that event B has occurred

Log-Odds formulation is more numerically manageable

Probability of sensing a hit given that previous data says it's a hit

Previous Probability

$$P_{occ} = \frac{P(z_t \mid occ)}{P_{prev}\ P(z_t)}$$

New Probability

Probability of sensor at time t correctly sensing a hit

Odds and Log-Odds

Odds

- Odds is the ratio between our confidence in success over confidence in failure.

$$\text{odds} = \lambda / (1 - \lambda)$$

Confidence in success

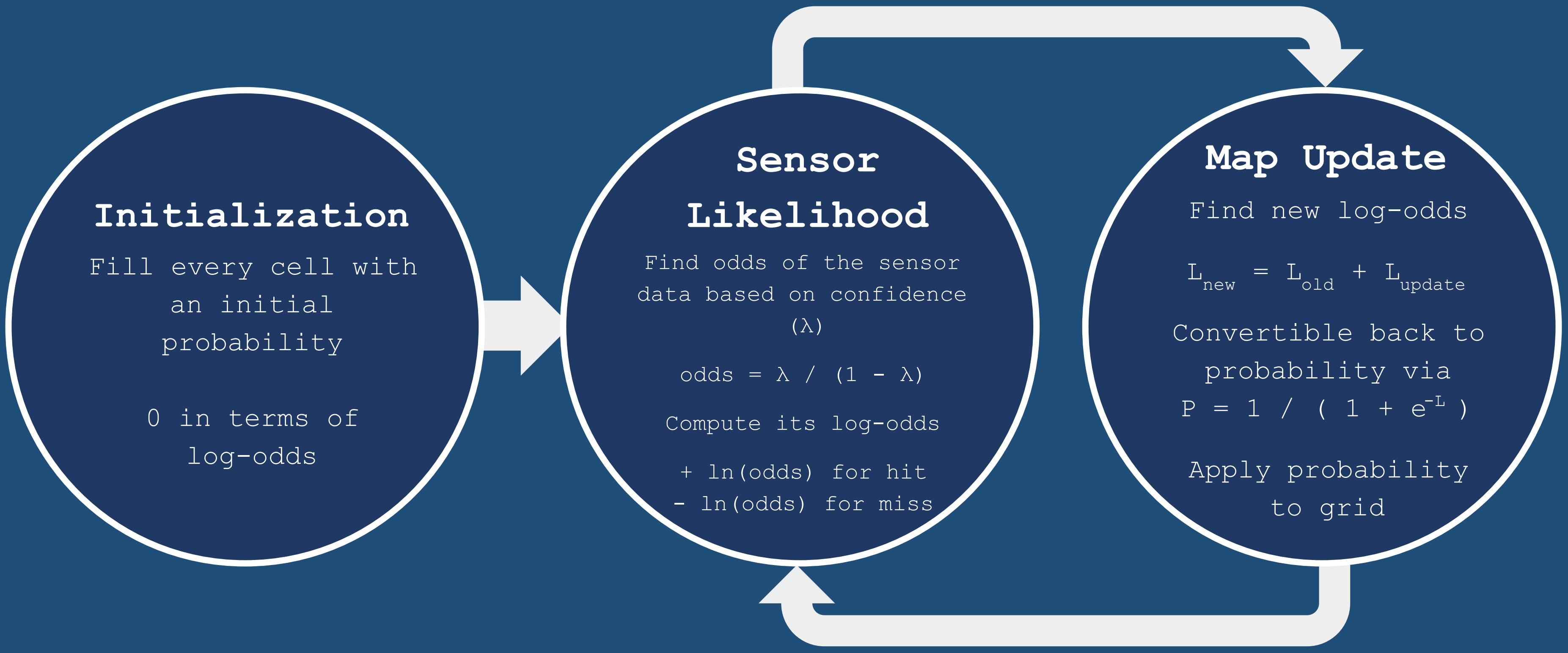
Confidence in failure

$$\ln(\text{odds}) = \ln(\lambda / (1 - \lambda))$$

$$\text{odds}_{t+1} = \prod^{t}_{i=1}(\text{odds}_i) * \text{update}$$

$$\ln(\text{odds}_{t+1}) = \Sigma^{t}_{i=1}\ln(\text{odds}_i) + \ln(\text{update})$$

OGM Algorithm



LiDAR and OGM Activity

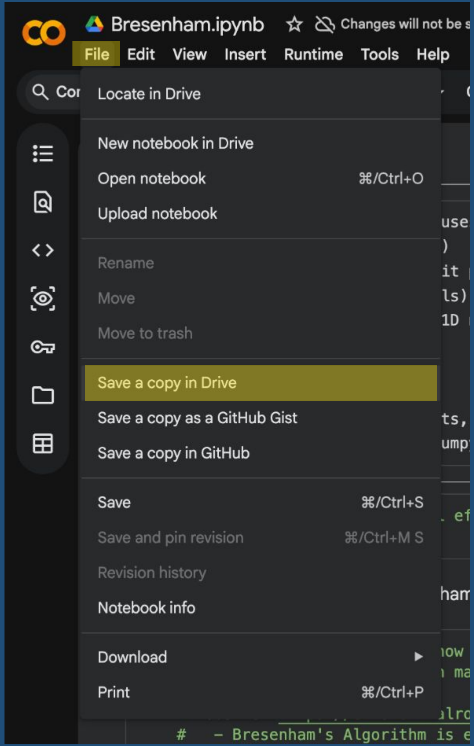
- Go to website
- Navigate to “Workshop” tab
- Scroll down to “Notebooks (During Workshop)”
- Complete **“LiDAR and Frame Transformations”** & **“Occupancy Grid Mapping (OGM)” notebooks**

Note: Follow the instructions in the text cells carefully

https://existentialrobotics.org/RobotProvingGrounds/

LiDAR and OGM Activity

File > Save a copy in Drive

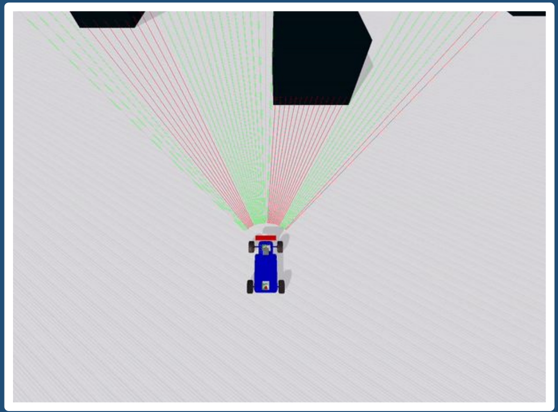


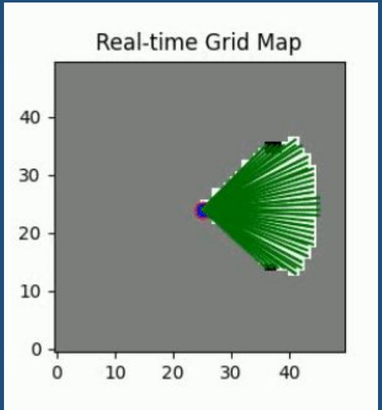
https://existentialrobotics.org/RobotProvingGrounds/

Simulation

Simulations

- Real life has unpredictable factors
- Simulations allow for reliable code testing
- Ex: Pybullet, IsaacGym, Gazebo, ManiSkill, Mujoco





PyBullet Activity

1) On your desktop, navigate to the “Workshop” page of the website.
2) Download the zip folder corresponding to your OS. Then, unzip the folder.
3) Follow the instructions on the README. Total installation time is around 10 minutes.
4) Paste these functions (bresenham_line, computer_lidar_hitpoints, prob_to_logodds) you created in the collab into functions.py
5) Run the python file pybullet_map.py
6) Use the arrow keys to move and observe a separate window with your OGM map

https://existentialrobotics.org/RobotProvingGrounds/

PyBullet Activity

https://existentialrobotics.org/RobotProvingGrounds/

Post-Attendance Survey

https://forms.gle/SMgxSiuvW9dSz8YPA

Website/Google Collabs

https://existentialrobotics.org/RobotProvingGrounds/

Q&A

Any Questions?