# RovoMapper Applications

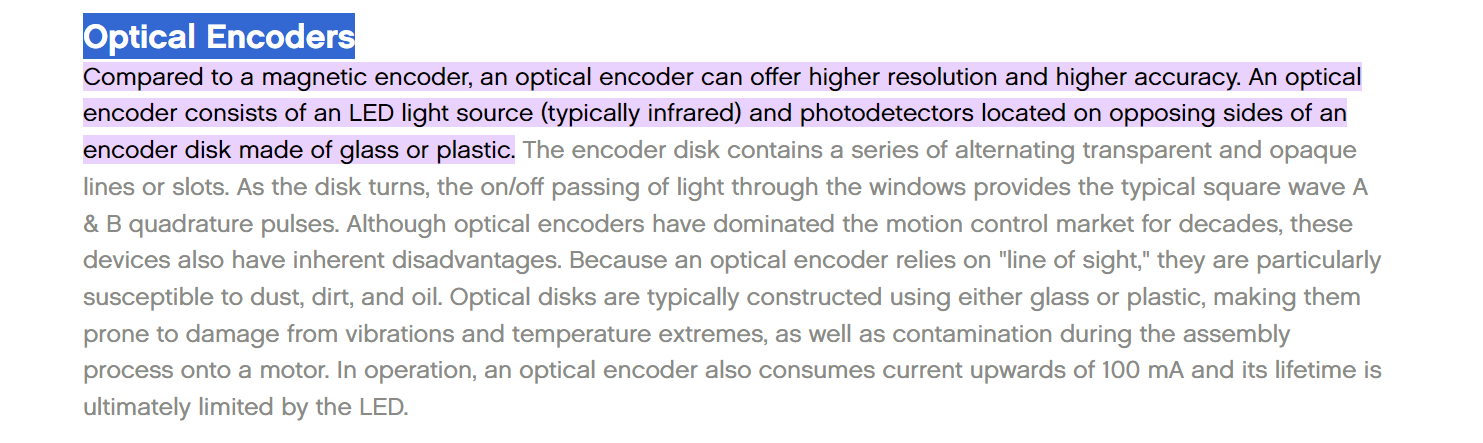
* <https://docs.google.com/document/d/1jYQgElaPiUtGEZ-9D9PM2ZRiJGUls_K5C8QNxR820j0/edit?usp=sharing>

# SLAM Algorithms

* Iterative approach better for this problem

# Rotary Optical Encoders

Functionality of the sensor matters most so price isn’t a huge constraint. Our priority is maximizing the accuracy of the measurements and minimizing drift while incorporating old code with minimal changes.



* **Resolution**: the smallest angular change that the encoder can detect and report as a distinct position change. It quantifies the level of detail in the position measurement that the encoder can provide. Resolution is either CPR or PPR
* Sensor placement? Shaft, hub, surface mounted
* How big is the wheel - necessary to determine how high resolution the encoder should be
* Measurements needed
  + Shaft size
  + Wheel thickness
  + Wheel diameter

Rotary optical encoders comparison:

<https://docs.google.com/spreadsheets/d/1QJ1Ak1p0TCMKAS-D0Aqq2mVkwf2FhZ19v-UrRdtOyXw/edit#gid=0>

# Aggregation and Transformation

I will start with how the data will travel. Below is a key of layers

Initialize Layer

Aggregation Layer

Transformation Layer

Visualization Layer

Mixed

Data Storage

Call diagram:

<https://www.figma.com/file/Vp7uYtZlDxoxrRqdHUt4Oq/API-Call-Diagram?type=whiteboard&node-id=0-1&t=nLNQJMas037TVHsx-0>

1. Intialize.py
   * Objective: Main entrance for the application. Starts up all the processes required to make this work and initializes the from config.json.
   * Methods
     + shutdown()
       - Shutdown all the processes currently running.
     + interrupt\_handler\_main()
       - Looks like it was used for some debugging.
     + Main()
       - Brings in the config file and initializes all the variables needed.
       - Seems to be a lot of multi-threading going on. A queue was set up.
       - Tests are followed.
       - Instantiate the transformation layer processes
       - Checks if the data logger is on.
       - Starts the threads for transformation layer and sleeps them for (1)
       - Some more error handling towards the end
2. Sensor to raw part
   * CSV\_to\_RAW.py
     + Objective: Mimics the data aggregation layer by publishing run data from csv file
     + Methods:
       - \_\_init\_\_
         * Collects args and sets them to 0.0
       - run()
         * Make sigint object
         * Open csv and collect all data from csv and put in data
         * Sends data to the transformation layer?
       - interrupt\_handler()
         * Handles signals and kills thread
   * JSON\_to\_RAW.py
     + Objective: Mimics the data aggregation layer by publishing run data from a text file where each line is a JSON string
     + Method:
       - \_\_init\_\_
       - run()
         * Make sigint object
         * Opens JSON file
         * Stream data into data variable
         * Sends to transformation layer
       - interrupt\_handler()
         * Handles signals and kills thread
   * Sensor\_to\_raw\_msg\_handler.py
     + Objective: Initializing the two MetaWear sensors and the Data stream begins.
     + Methods
       - \_\_init\_\_()
         * Takes in args and initializes the incoming data.
         * Has a class inheritance structure
         * Information on what exactly each argument is.
       - run()
         * Set up signal interrupt and message handler
         * Creates sensor objects, connects sensors, and configures them.
         * Starts a timer
         * Tells parent that process are set up and polling is done
         * Start event loop?
       - on\_message()
         * Handles when a reset is sent from the visualization layer
       - interrupt\_handler()
         * Interrupts process and triggers process shutdown
       - shutdown()
         * Resets sensors and waits for all threads to be done and kills threads.
     + Sensor.py
       - Objective: holds state information and callback functions required to initialize the sensor, start a data stream, and collect, pair, and publish the message handler.
       - Methods
         * \_\_intit\_\_()

Same from before^^^

* + - * + data\_handler()

Callback function that is executed for each packet of data

Starts timer

Multithreading occurs with semaphore lock, faster thread first.

Parses data and adds it store

Sends data to msg\_handler

* + - * + setup()

Setup acc

Setup gyro

Get acc signal

Get gyro signal

Fuse signals

Call backs? Recursion?

* + - * + Start()

Initializes data stream

Start acc sampling

Start gyro sampling

* + - * + Shutdown

Disconnects sensors and unsubscribes from data signals and kills threads?

1. From raw to linear part
   * Raw\_to\_linear\_msg\_handler.py
     + Objective: Process that handles subscribing to the raw data passing to raw\_to\_linear.
     + Methods
       - \_\_init\_\_
       - run()
         * Setup signal interrupt
         * Create transformer and msg handler
         * Start event loop
       - interrupt\_handler()
       - on\_message()
         * Creates the message payload for raw\_to\_linear
     + Raw\_to\_linear.py
       - Objective: Class for transforming raw sensor data into linear distance traveled by each wheel
       - Methods:
         * \_\_init\_\_

Creates instance for class with default values

* + - * + Transform()

Used to reset process when reset is hit in UI

* + - * + calculate\_linear\_distance()

Transforms data from DPS as rotational velocity to linear distance traveled(unit less)

Makes the calculations and returns linear distance

* + - * + dps\_filter()

Filter for raw gyroscope reading

Assumption that this is from the drifting Dr. Tamer mentioned.

* + - * + calculate\_distance\_over\_interval()

Calculate distance from both wheels Updates payload accordingly

Calls calculate\_linear\_distance 2x

Updates local data

Update the message payload data sent back to message handler

* + Linear\_to\_location\_msg\_handler.py
    - Objective: Background process that handles subscribing to the topic that stores linear data, passing the data to linear\_to\_location.py
    - Methods
      * \_\_init\_\_
      * Run()
        + Turns on sigint
        + Create transformer and msg handler
        + Start event loop
      * Interrupt\_handler()
        + Catches signal
        + Kills process
      * on\_message()
        + Sets up the data payload for the sending
    - Linear\_to\_location.py
    - Objective: Transforming linear wheel distance data to x,y location
    - Methods
      * \_\_init\_\_
        + Set variables to 0
      * rotate()
        + Helper for the turn method. Rotates one point (point) around another point (pivot) by the specified angle (angle)
        + Sees what the angle is ?
      * turn()
        + Transforms linear data of two wheels into x,y coordinates
        + Checks how the turn went with mathematics based of the +- direction and pivot
        + Determines angle of the turn in radians
        + Adjusts rotation based on current heading?
        + Add new coordinates to previous coordinates
        + Updates heading
      * track()
        + If reset is passed in data then it clears coordinates
        + Else instantiates current coordinates

1. Visualization and storage
   * Visualization Layer
     + Visualization-PyQt is the main function for graph and GUI
     + Just python framework
   * Location\_to\_log.py
     + Test\_output.log
       - Worked as a temporary memory. Unable to find actual log.