Introduction

Continuing your training with the CyBot platform, this week you will complete a simple mission. You will develop an embedded application that can autonomously identify the smallest width tall object in a test field, and allows you to manually navigate to that object using only sensor data. To help guide you through this mission, we have decomposed it into several parts.

Some high-level ground rules:

1. Allowed sensors
   a) `cyBOT_Scan`
   b) The following Open Interface sensors:
      i. `distance`
      ii. `angle`
      iii. `bumpLeft, bumpRight`
      iv. `cliffFrontLeft, cliffFrontRight`
2. Objects that can be encountered
   a. Short objects: detected by the `bump` sensors
   b. Holes: detected by the `cliff` sensors
   c. Tall objects: detected by `cyBOT_Scan`, and that the CyBot should never collide with, or come within 2 cm of.

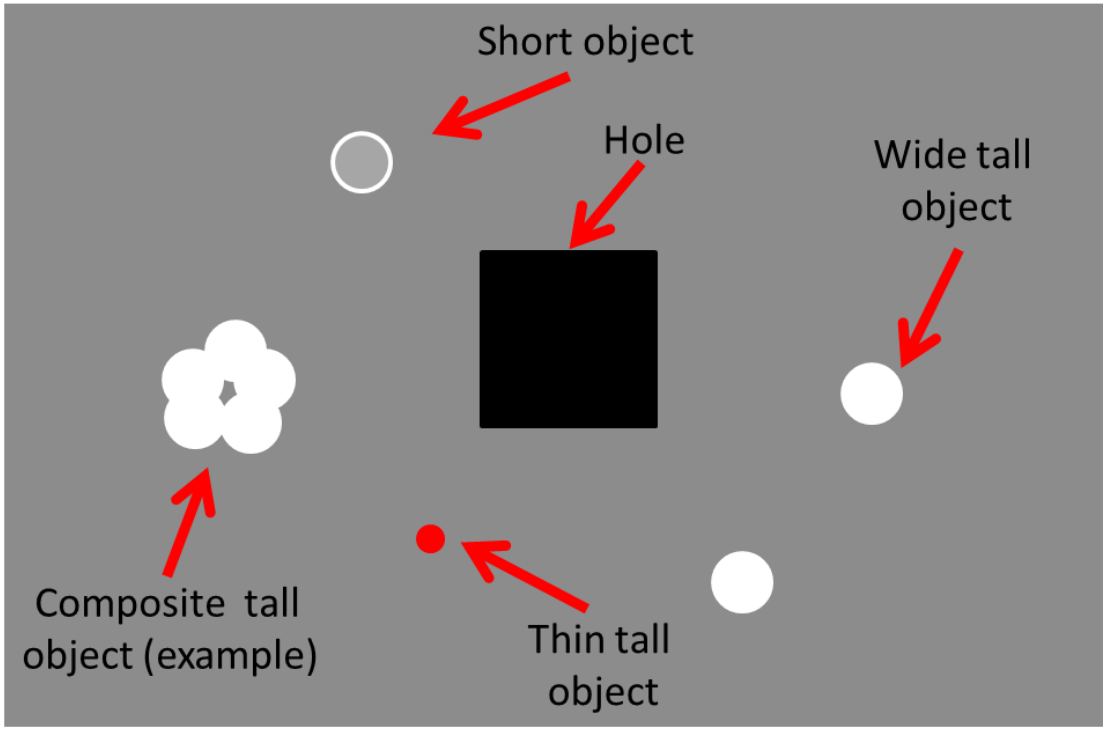


Figure 1: Example test field

Part 1. Analyzing sensor data (Checkpoint 1)

In this part of lab, you will examine a text logfile (**mock-cybot-sensor-scan.txt,** found on the Resources tab of the course website) containing some mock sensor data. Each row of the logfile contains an angle at which the Cybot makes a distance measurement, and the distance it measured. It collects data scanning from 0 to 180 degrees in 4-degree increments.

1. Analyze text logfile: Open the text logfile, and based on the values in the file indicate the following:
i) How many objects are being observed?

ii) How far away is each object? (take distance at the center of the object)

iii) What is the angular size of each object? (i.e., the starting angle of the object – the ending angle of the object)

2. Analyze the logfile graphically: Use the python script provided (**CyBot-Plot-Sensor-Scan-Values-From-File.py,** found on the Resources tab of the course website) to plot the text logfile, and to graphically verify your answers from **1**.

3. Debugging algorithms/functions that process sensor data:
i) Discuss with your lab partner the advantages and disadvantages of **manually** inspecting a text logfile of sensor data to aid in debugging an algorithm/function that processes this sensor data. **Identify at least one advantage, and one disadvantage.**

ii) Now have a similar discussion with respect to the advantages and disadvantages of inspecting an **automatically generated graphical view** of a sensor data text logfile to aid in debugging an algorithm/function that processes this sensor data. **Identify at least one advantage, and one disadvantage.**

Checkpoint: Demonstrate to your Mentor that you can analyze the sensor data logfile: 1) by manually inspecting logfile text, and then 2) by inspection an auto-generated graphical view of the logfile data. Also explain to them an advantage of manually inspecting the logfile of sensor data to aid in debugging an Algorithm/Function that processes that sensor data, and an advantage of analyzing such a logfile graphically to aid in debugging an Algorithm/Function that processes that sensor data.

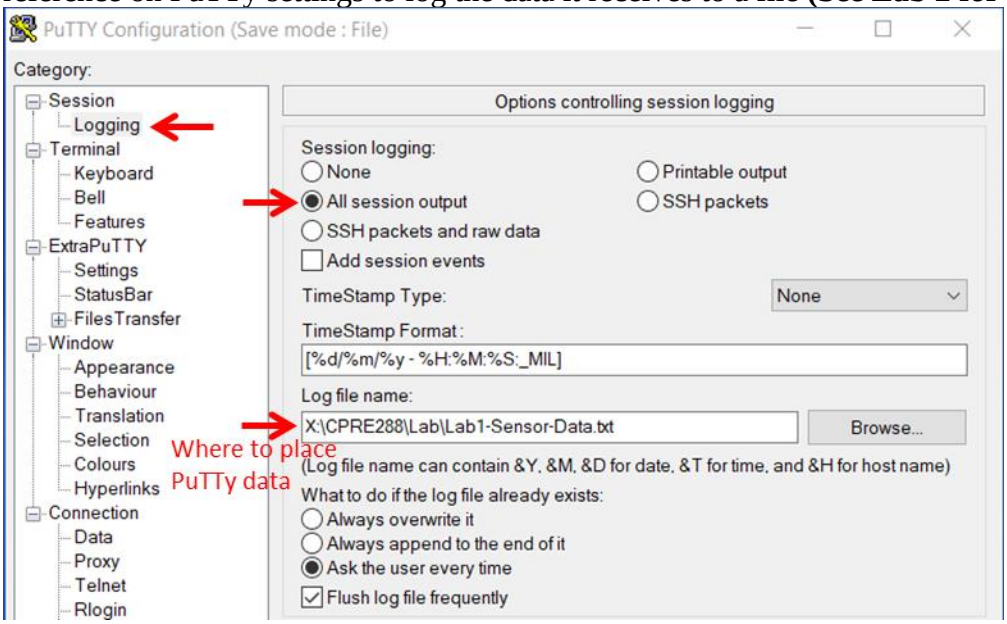
Part 2: Collect, format, and display sensor data (Checkpoint 1)

The test field will be populated with 2 to 3 "tall" objects, one of which will have the smallest width. The first step in identifying the object with the smallest width is to collect data using the Cybot sensors.

Have your embedded application preform the following autonomously:

Use the `cyBOT_Scan` function to perform a 180 degree scan (**in 2-degree increments**), and display the angles and collected distance information to PuTTy when an "m" is sent from PuTTy to the Cybot (see display format below). **IMPORTANT:** for distance information use `sound_dist` only, **do not use** `IR_raw_val`.

Additionally: have PuTTy send this collected and formatted data to a file (**to be used later**). Here is a quick reference on PuTTy settings to log the data it receives to a file **(See Lab 2 for setting up PuTTy in general)**:



Tip: Review the function `sprintf()`. This function is extremely useful for formatting sensor data before transmitting it from the Cybot to your Desktop PC. See the C-string lecture slides on `sprintf()` and/or https://www.javatpoint.com/sprintf-in-c for additional details (and usage examples).

Excerpt of an example for how to display sensor information to PuTTy:

```
Degrees        Distance (cm)
0               120
2               40
4               40
...             ...
178             150
180             150
```

Checkpoint: Demonstrate to your Mentor that your embedded application can collect, format, display to PuTTy, and log to a file the sensor data as specified.

Part 3: Calibrate servo, compare collected sensor data with mock sensor data, and manually cleanup sensor data (Checkpoint 1)

In this part of lab, you will i) calibrate the sensor servo, ii) investigate the differences between the mock sensor data from Part 1, and the actual sensor data you collect in this part, and iii) manually clean up sensor data.

Calibrate servo:

When you request a Cybot servo to scan from 0 to 180 degrees, you will notice the direction it points is significantly different from your request. Each Cybot servo needs to be individually calibrated, thus the initial settings used by `cyBOT_Scan` only roughly point the servo in the direction you command. Use the Servo Calibration Reference Sheet to make a simple program that will allow you calibrate your servo to accurately point in the direction commanded.

Compare with mock data (sources of noise in sensor data):

A. Collect sensor data: Collect data for at least 2 test-field setups (i.e., different numbers & placements of tall objects).
1. Place your Cybot on the floor in a reasonably open area
2. Place 2-3 "tall objects" around the Cybot. Make sure object centers are separated by at least 45º, within the 0 – 180-degree scan range. **For one** test-field setup, place objects equidistant from the Cybot.
3. Command your Cybot to scan from 0 to 180 degrees (**at 2-degree increments**), and send the collected information to PuTTy, and store to a logfile using the formatting of Part 2.

B. Examine the collected sensor data:
1. Logfile: When inspecting your sensor data logfiles, what are challenges when interpreting the number of objects and their locations compared to performing this analysis with Part 1's mock sensor data logfile?
2. Plot: Use the script form **Part 1** to graphically examine a plot of your sensor data logfiles. What challenges occur when graphically interpreting how many objects are being detected and their locations compared to performing this analysis with a plot of the mock sensor data logfile from Part 1?

C. Sources of noise, and ideas for "cleaning up" sensor data:
1. What are potential sources of noise? For example, people walking into your scan, interference from other CyBots scanning, etc.
2. How might you deal with each noise source? In other words, what actions could a program take to clean up the collected sensor data to "remove"/"reduce" the noise. **Spend at least 5 minutes discussing approaches with your lab partner.**

Manually cleanup sensor data:
1. Make a copy of each of your sensor data logfiles, rename them, and open each in a text editor
2. Manually edit the renamed files to "clean up" the data to make them less "noisy".
3. Use the **Part 1** Python script to plot your "cleaned up" data to visually verify your modifications
4. Compare these plots with plots of the non-modified collected sensor data logfiles.
5. Now that you have experience manually cleaning up sensor data, **revisit with your lab partner (for at least 10 minutes)** protentional approaches for having a program autonomously clean up the sensor data.

Checkpoint: Demonstrate to your Mentor that: **i)** you can calibrate the sensor servo, **ii)** your embedded application can collect, format, display, and log to a file the sensor data as specified in part 2, **iii)** you can manually analyze the sensor data to determine how many objects are in the test field and where, and **iv)** you can manually clean-up the sensor data, and have ideas for how a program could clean up the data autonomously.

Part 4: Detect smallest width object using "cleaned" sensor data (Checkpoint 1)

In this part of lab, you will use your manually "cleaned up" sensor data from Part 3 to develop an embedded application that can autonomously analysis this data to: **i)** detect each object in the test-field, **ii)** compute the width of each object, and **iii)** identify the smallest width object.

Part A: Prototyping your sensor data analysis

For **Part B**, and **Part C**, first prototype and test your approach on a Desktop or Laptop computer (using C or any language you prefer) before deploying your sensor data analysis code to the Cybot (**Part D**). This will allow you to distinguish between issues in your approach, versus issues with how you implement your approach, versus specific Cybot platform related issues. Also, this allows conducting most of your embedded application implementation, debugging, and testing without needing to have a Cybot or even be in Lab.

Note: Options for C prototyping are: i)installing a C development environment on your own machine, ii) using a Linux machine in the TLA, or iii) remotely logging into a Department Linux machine that has a gcc C compiler (see: https://etg.ece.iastate.edu/vdi/ Click on "How to connect")

Part B: Have your embedded application perform the following autonomously

- **Assumption:** Objects are places about equidistance from the CyBot. This assumption is removed for **Part C.**
- **Assumption:** Objects are located in front of the CyBot so that a 180-degree scan sees all objects

1) Based on your manually "cleaned up" sensor data, have your embedded application determine the following for each object detected and display the following (**to PuTTy during Part D**):

a) Number each object detected
b) Distance to object
c) Angle at which each object is detected
d) The **radial** width of each object (i.e., Degrees within the scan an object appears. For example, if the scan starts seeing an object at angle 30 degrees and stops seeing it at 35 degrees, its radial width is 5 degrees)

Excerpt of an example display of sensor information (**to PuTTy during Part D**):

```
Degrees        Distance (cm)
0               120
2               40
4               40
...
178             150
180             150

Object#   Angle          Distance  Width
1         85             60        10
2         120            35        5
3         150            70        7
```

2) Identify the smallest **radial** width object. **In Part D,** the Cybot should point the servo this object. Show your mentor that the displayed data matches the test-field setup, and that your embedded application's analysis of the data matches your manual analysis of the data with respect to finding the object with the smallest **radial** width.

3) Explain to your mentor a drawback of using **radial** width to identify an object with the smallest actual (linear) width, when objects are not placed equidistant from the CyBot.

Part C: Repeat Part B to find the object with the smallest actual (linear) width

Hint: Assume both edges of an object are the same distance from the CyBot. Thus, after scanning, for each object you will know i) its angular (radial) width, and ii) the distance to each edge. In other words, you know the length of the congruent sides of an isosceles triangle, and the angle between them. Use geometry/trig to find the length of the third side. This will be considered the actual (linear) width of an object.

Note: For this part, each object will be placed at a different distance from the Cybot.

Part D: Port and integrate your sensor data analysis prototype onto the Cybot
Once you have verified that Parts A-C work as expected running in a Desktop/Laptop environment, modify your embedded application as needed to deploy on to the Cybot. Debug issues that come up. For example, issues related to the size of your data structures, or introducing bugs when integrating with Cybot specific functionality (e.g., LCD display, PuTTy communication, servo interface, etc).

Checkpoint: Demonstrate to your mentor that your embedded application autonomously points to the object with the smallest width based on your manually cleaned up sensor data.

Part 5: Detect smallest width object using "noisy" sensor data (Checkpoint 2) (Bonus: 2pts if completed by Checkpoint 1 deadline)

In this part of lab, you will **a)** autonomously cleanup your sensor data, **b)** use the autonomously "cleaned up" data from **Part A** to develop an embedded application that can autonomously analysis this data to: **i)** detect each object in the test-field, **ii)** compute the width of each object, and **iii)** identify the smallest width object. The test-field will be populated with 2 to 3 tall "objects" one of which will have a smaller width than the others.

Part A: Autonomously "clean up" sensor data

Now that you have experience working with manually cleaned up sensor data, in this part of lab develop code to automatically "clean up" your sensor data. As in Part 4, first prototype and test your code on a Desktop or Laptop computer, before deploying it on to the Cybot.

Part B: Repeat Part 4 using autonomously cleaned-up sensor data

Use the cleaned-up sensor data from Part 5A, and repeat Part 4A–4C. You will likely notice that your autonomously cleaned-up sensor data is not as "clean" as when you manually cleaned-up the sensor data. Thus, you will need to adjust your sensor data analysis code developed in Part 4 to account for these differences, and/or improve your code for autonomously cleaning up sensor data.

Part C: Integrate your sensor data collection, autonomous sensor data cleanup, and object detection into your Cybot embedded application

Integrate your autonomous sensor "clean-up" functionality into your Part 4D code. Test and debug as needed to account for issues that may occur related to integrating with Cybot-specific functionality. **Your first step** for integration should be hard-coding your noisy data into an array. Once that works on the Cybot, refine your code as needed to handle sensor data being collected at runtime from a test-field.

Debugging Tip: Since it is not uncommon for unexpected situations to occur while testing in the test-field, it is recommended that you have a debugging mode for your embedded application where the Cybot returns to PuTTy: i) the unprocessed raw noisy sensor data, ii) the autonomously cleaned-up sensor data, and iii) analysis of the data. Below is a simple example format for this debugging mode. Depending on the approach you used for collecting, cleaning, and analyzing sensor data, you may want to return additional information beyond this. **Having this type of information is extremely helpful in locating the root-cause of bugs in your system.**

```
**Raw noisy sensor data**
Degrees        Distance (cm)
0               40
2               120
4               40
...
176             150
178             90
180             150

** Cleaned data **
Degrees        Distance (cm)
0               40
2               40
4               40
...
176             150
178             150
180             150

Object#    Angle           Distance   Width
1          85              60         10
2          120             35         5
3          150             70         7
```

Checkpoint: Demonstrate to your mentor that your embedded application points to the object with the smallest width.

Part 6: Drive to the smallest width object using only sensor data (Checkpoint 2) (Bonus: 1pt if completed by Checkpoint 1 deadline, and Part 5 has also been completed by Checkpoint 1 deadline)

The test field will be populated with 2 - 3 tall "objects", one of which will have a smaller width than the others.

Part A: Manually drive your CyBot to the smallest width object using only sensor data collected as you drive the CyBot. You are not allowed to watch the CyBot (TAs will put up a wall to help block your view). To guide your driving, manually inspect the sensor logfile after each scan, and graphically inspect a plot of this data using a Python script.

Note: If your Part 5 works well, then your embedded application should automatically process the sensor data after each scan, and inform you of each object's location, and their widths. However, you may still want to manually inspect the sensor data to double check your embedded application's analysis.

- **Assumption:** Objects are located in front of the CyBot so that a 180 degree scan sees all objects

Manually drive to within 5 cm of the object with the smallest width

Part B (Bonus 1pt): Update Part A to deal with objects being placed at any angle around the CyBot

Checkpoint: Demonstrate to your Mentor that you can manually drive your CyBot to within 5 cm of the smallest width object only using sensor data.