# Control of Mobile Robotics
# CDA4621
# Spring 2020
# Lab 3
# Vision-based Navigation & Obstacle Avoidance
# Total: 100 points

### Due Date: 4-3-2020 by 8am

The assignment is organized according to the following sections: (A) Lab Requirements, (B) Lab Description, (C) Task Description, (D) Task Evaluation, and (E) Lab Submission.

## A. Lab Requirements

**Hardware:** "Robobulls-2018" robot.
**Software:** Python

## B. Lab Description

This lab will teach you about: (1) Basic Camera, (2) Simple Blob Detector, and (3) Camera Threading. This lab requires the use of the camera, distance sensors and motor control.

### B.1 Basic Camera

Read the supplementary material[1] on accessing the camera from Python. OpenCV is a popular library for real-time computer vision. It can be used for accessing the camera on the Raspberry Pi. The camera is initialized by creating a "VideoCapture" object. Frames can be retrieved at any time using the "read" function. These frames are in the "Mat" format, which allows the frame to be easily manipulated or displayed on screen with the "imshow" function.

### B.2 Simple Blob Detector

Read the supplementary material[2] on using the Simple Blob Detector feature in OpenCV. One of OpenCV's features is Simple Blob Detector, which allows for easy detection of blobs in an image, i.e. groups of connected pixels of the same color. This feature is initialized with a set of "Params" that specify the allowable size, color, and shape of blobs. Then a "Mat" object is input into the detector, which is a raw grayscale or color image. The detector returns a list of blobs in the form of "KeyPoint" objects. These "KeyPoint" objects contain the X and Y center position of each blob, along with its diameter. One major optimization that can be done for the blob detector is by masking out parts of the image that are not relevant for the detector. If a specific color needs to remain visible, this can be accomplished by first converting the "Mat" from RGB (red-green-blue) format to HSV (hue-saturation-value) format with the "cvtColor" function. Then

[1] https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/
https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

[2] https://www.learnopencv.com/blob-detection-using-opencv-python-c/
https://docs.opencv.org/3.4.1/da/d97/tutorial_threshold_inRange.html
https://docs.opencv.org/3.4.1/d2/d29/classcv_1_1KeyPoint.html

"inRange" function can be applied to black-out any pixels in the "Mat" that are not the correct color. This masked frame is then fed into the detector.

### B.3 Camera Threading

Read the supplementary material[3] on using threading to improve camera performance. Achieving low camera latency is important for proper object detection and tracking. The process of capturing a frame consumes a small amount of time, and when combined with a computationally intensive task like object detection leads to increased latency. Too much latency causes functionality like proportional control, to begin to malfunction, unless movement is slowed down significantly. You may need to implement camera threading if you are experiencing these issues.

## C. Task Description

The lab consists of 4 tasks leading to the Bug Algorithm:

- Task 1 - Goal Facing
- Task 2 – Motion To Goal
- Task 3 – Triangulation or Trilateration
- Task 4 – Bug Algorithm

### C.1 Task 1 – Goal Facing

Implement a program called "faceGoal.py". The program should make the robot rotate around its center until it faces one of the goals. Goals are large cylinders, 2 feet tall and 4 inches in radius, covered in non-reflective bright paper. Robots must be able to face any of the goals from a distance of up to 8 feet away. The robot must remain still once it faces the goal. If the goal is moved, the robot must reorient itself towards the goal. There will be 3 different colored cylinders setup inside the maze (Fig. 1 left) or outside (Fig. 1 right). The robot should be able to recognize both configurations from 4-8ft away. In either maze, at the beginning of the task, the robot will be specified a particular color goal that it needs to face. There are no obstacles on the floor between the robot and the goals for this task.

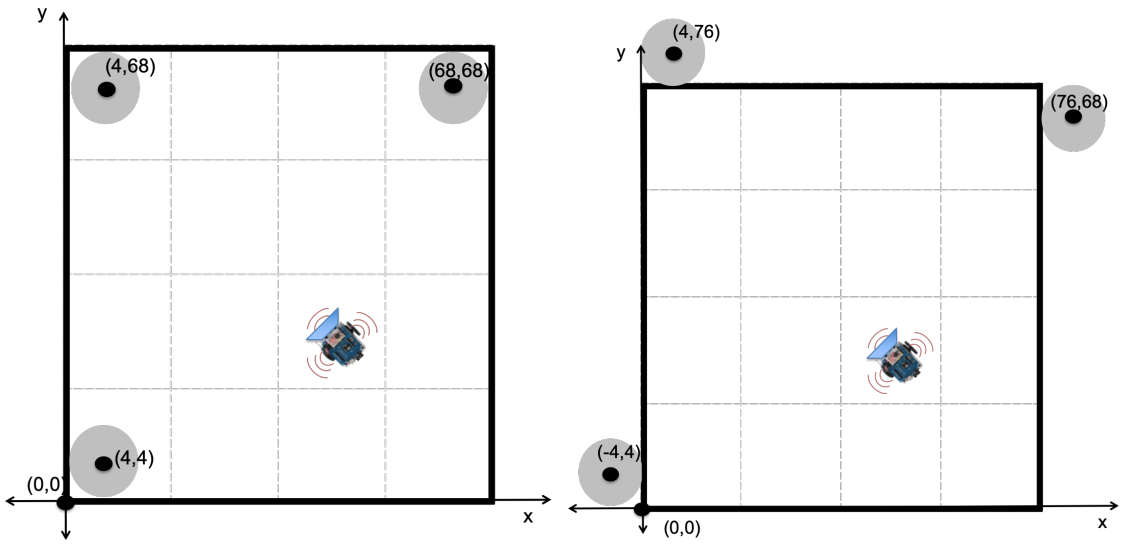


Figure 1. All coordinates are in inches. Square maze with 6ft (72 inches) sides. All cylinders have 4 inch radius. (left) Open maze with 3 internal colored cylinders. (right) Open maze with 3 external colored cylinders.

[3] https://www.pyimagesearch.com/2015/12/21/increasing-webcam-fps-with-python-and-opencv/

You will need to implement the Simple Blob Detector feature from the OpenCV library to locate the goal. This feature expects a "Mat" object such as the one returned from the camera feed. It is recommended to apply the "inRange" function to the "Mat" so that only the goal is visible to the detector. After running the detector, the software returns a list of blobs in the form of "KeyPoint" objects. You will need to determine which, if any, of the returned KeyPoints are actually the goal and not something else from the environment. You can adjust the "inRange" function along with the detector's "Params" to improve detection accuracy. Goal facing can be obtained by implementing proportional control. The input function can be the X position of the goal blob, and the output can be the angular velocity of the robot's wheels. Note that the goal may not be visible at all if the robot is facing a different direction. You will need to account for this by rotating the robot until the goal is in view. Also, the robot may be picked up and moved, so it should always be running the goal facing algorithm.

### C.2 Task 2 – Motion to Goal

Implement a program called "motionToGoal.py". This task is an extension of Task 1. The program should begin by facing the goal, and then continue by moving towards the goal. It must stop moving once the robot is 5 inches away from the goal. There are no other obstacles on the floor between the robot and the goal. Both the robot and the goal itself may be moved at any time during execution. The robot should react accordingly by facing the goal again and then moving towards it. If the robot moves towards the goal but begins to veer off to one side, it needs to realign itself and continue moving. In case the goal is moved further away, the robot needs to keep moving towards the goal and stop again at 5 inches from it. The robot should be able to recognize both configurations from 4-8ft away. A different color may be given to the robot in order to perform a new motion to goal. As part of this task you will need to include 2 plots, one for each type of maze configuration in Fig. 1, showing the distance of the robot to the goal relating blob size (y-axis) to measured distance (x-axis).

### C.32 Task 3 – Triangulation or Trilateration

Implement a program called "triangulation.py". You may use Triangulation or Trilateration[4] methods to determine robot location relative to the 3 goals (beacons) in the maze. Note that trilateration methods use relative distance to the goals, while triangulation methods require knowledge of distances and angles relative to the goals. As part of this task, the robot will be set at a random location in the field and by facing the different goals it needs to calculate its current location. You will need to compute distances to each colored cylinder and your program should compute the global location of the robot. The robot will need to rotate in order to face the 3 different goals in order to compute the their relative distances. The robot does not move towards any of the goals. The robot needs to print its global location for both maze configurations. The bottom left corner of the maze corresponds to (0,0) in the global reference frame. Your program should set in advance the (x,y) location of the goals.

### C.4 Task 4 – Bug Algorithm

Implement a program called "bugAlgorithm.py". The program should use any bug algorithm (preferably "Bug 0") to move the robot towards one of the goals while avoiding obstacles. The specific goal and maze configuration will be specified during the presentation. Up to three obstructing obstacles will be placed between the robot and the goal (Fig. 2 left and right). Each of the obstacles will be no closer than 12 inches from each other so the robot has enough space to move between them. The robot needs to perform face-goal and motion-to-goal, but if an obstacle is detected immediately in front of the robot (the distance on the front sensor is less than 10 cm),

[4] https://en.wikipedia.org/wiki/True_range_multilateration

it needs to switch to wall-following. Wall-following will continue until the robot has line-of-sight of the goal AND there is no obstacle immediately in front of the robot. Then it should switch back to motion-to-goal until the next obstacle is encountered. Failure to switch back to motion-to-goal will lead to a loss of points. Complete task should take less than 2 minutes.

While it is not a requirement, it is suggested that you implement this program using a *state machine*[5]. At least three states could be used: face-goal, motion-to-goal, and wall-following, along with any additional states as necessary. Whenever the robot detects a change in the environment, such as the goal becoming centered within the camera view or an obstacle being encountered, the state machine should switch to the appropriate state to handle this change.


Figure 2 (left) Maze with internal obstacles and goals. (right) Maze with internal obstacles and external goals.

## D. Task Evaluation

Task evaluation involves: (1) a task presentation of robot navigation with the TA, (2) the accompanying code to be uploaded to Canvas, and (3) task report to be uploaded to Canvas. All uploaded documents will be scanned through copy detection software.

### D.1 Task Presentation (80 points)

Task presentation needs to be scheduled with the TA. Close to the project due date, a timetable will be made available online for groups to select a schedule on a first come first serve basis. All team members must be present at their scheduled presentation time. On the presentation day, questions related to the project will be asked, and the robot's task performance will be evaluated. If it is seen that either or both presenters have clearly not understood a significant portion of the completed work, points will be deducted up to and including full points for the task presentation.

NOTE: During presentations, any robot calibration time is limited to 1 min.

It is important that all members of the team understand the work submitted.

The following table shows the rubric for tasks presentations:

- Task 1 – 20 points
  - Robot can face a single internal colored goal (5 points)
  - Robot can face a single external colored goal (5 points)

[5] https://en.wikipedia.org/wiki/State_diagram

    - Robot tracks the goal when the internal goal is moved (5 points)
    - Robot can recognize multiple internal goals and face either one (3 points)
    - Robot can recognize multiple external goals and face either one (2 points)
    - Robot takes longer than 20 seconds to face the goal (-4 points)
    - Robot is unable to stop turning after facing the goal (-4 points)
- Task 2 – 20 points
    - Robot can do motion-to-goal to single internal goal (5 points)
    - Robot can do motion-to-goal to single external goal (5 points)
    - Robot restarts motion-to-goal when the internal goal is moved (5 points)
    - Robot can do motion-to-goal move to any of the internal colored goals (3 points)
    - Robot can do motion-to-goal move to any of the external colored goals (2 points)
    - Robot takes longer than 20 seconds to face the goal (-4 points)
    - Robot is unable to stop turning after facing the goal (-4 points)
    - Robot hits the goal (-4 points)
- Task 3 – 20 points
    - Robot can measures and prints the 3 distances to each internal goal (5 points)
    - Robot can measures and prints the 3 distances to each external goal (5 points)
    - Robot can measures and prints a triangulation or trilateration measure to each internal goal (5 points)
    - Robot can measures and prints a triangulation or trilateration measure to each internal goal (5 points)
    - Robot provides correct relative goal distances within the correct grid cell (-1 point for each incorrect goal distance)
    - Robot provides correct triangulation or trilateration coordinates within the correct grid cell (-3 points)
- Task 4 – 20 points
    - Robot can do motion-to-goal and wall-following with 1 obstacle and internal goal (5 points)
    - Robot can do motion-to-goal and wall-following with 2 obstacles and internal goal (5 points)
    - Robot can do motion-to-goal and wall-following with 1 obstacle and one of the external goals (5 points)
    - Robot can do motion-to-goal and wall-following with 2 obstacles and one of the external goals (5 points)
    - Robot hits obstacle (-3 points each time)
    - Robot fails to switch back to motion-to-goal (-4 points each time)
    - Full task is not completed under 2 minutes (-5 points)

**D.2 Task Report (20 Points)**

The accompanying task report needs to be uploaded to Canvas as a PDF file together with ALL the files required to run robot navigation. Upload all files into a single "zip" file. The task report should include ALL of the following (points will be taken off if anything is missing):

1. List of all code files uploaded to Canvas. All requested code must be included as a list in the main report, adding a one-line description to each of the files being uploaded (1 point).
2. The two plots required in Task 2, showing distance measurements from the camera to the goal for the two maze configurations. Include brief explanation of the equations used. (5 points)
3. Explain the process of analyzing a video frame to recognize and extract a single goal position for both Fig 1 left and right. (5 points)

4. Explain the process of analyzing a video frame to recognize and extract the three goal positions for both Fig 1 left and right. (5 points)
5. Link to a video where the robot is shown performing all tasks. (2 points)
6. Conclusions where you analyze any issues you encountered when running the tasks and how these could be improved. Conclusions need to show an insight of what the group has learnt (if anything) during the project. Phrases such as "everything worked as expected" or "I enjoyed the project" will not count as conclusions. (2 points)

All plots must include title, axis names, units, sufficient tick marks and legend (if plotting more than one graph). Also, each data point should clearly be marked with a symbol. Plots of a variable vs time should always place time on the x-axis. Points will be taken off for not following the specified format.

## E. Lab Submission

Each group is required to submit its report through Canvas. Penalties will be applied for late submission (see syllabus). All documents need to be in PDF, while a single "zip" file must be uploaded to Canvas containing all requested files. Note: no diagrams or descriptions by hand will be accepted.

## F. Trilateration

In the case of unreliable angle calculations, robot localization (*x,y*) can be computed using trilateration methods based on the determination of absolute or relative measured distances to at least 3 known beacon locations[6], as shown in Fig 4:

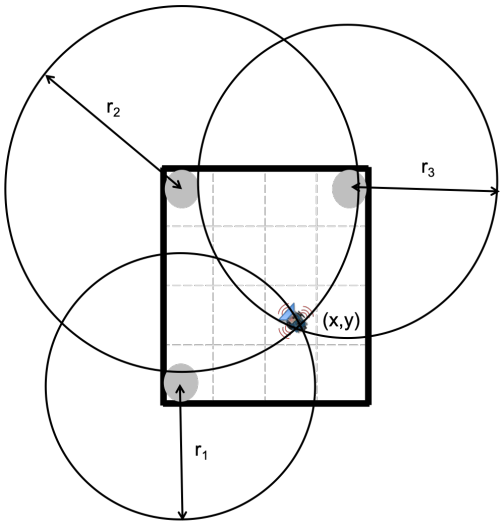


Fig 4. Intersection of 3 circles formed by the distances $r_1$, $r_2$, $r_3$, of the robot to the 3 beacons.

The circles equations obtained from the 3 measured distances are given by:

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$
$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

Expanding out these equations:

[6] http://www.telecom.ulg.ac.be/publi/publications/pierlot/Pierlot2011ANewThreeObject/index.html

$$x^2 - 2x_1x + x_1^2 + y^2 - 2y_1y + y_1^2 = r_1^2$$
$$x^2 - 2x_2x + x_2^2 + y^2 - 2y_2y + y_2^2 = r_2^2$$
$$x^2 - 2x_3x + x_3^2 + y^2 - 2y_3y + y_3^2 = r_3^2$$

Subtracting the second equation from the first generates the "Radical Axis[7]" or "Power Line" between the first two circles:

$$(-2x_1 + 2x_2)\,x + (-2y_1 + 2y_2)\,y = r_1^2 - r_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2$$

Subtracting the third equation from the second generates the "Radical Axis" or "Power Line" between the last two circles:

$$(-2x_2 + 2x_3)\,x + (-2y_2 + 2y_3)\,y = r_2^2 - r_3^2 - x_2^2 + x_3^2 - y_2^2 + y_3^2$$

These two power lines will intersect at $(x,y)$ to generate an approximate robot location (without orientation information) given by:

$$Ax + By = C$$
$$Dx + Ey = F$$

where:

$$A = (-2x_1 + 2x_2)$$
$$B = (-2y_1 + 2y_2)$$
$$C = r_1^2 - r_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2$$
$$D = (-2x_2 + 2x_3)$$
$$E = (-2y_2 + 2y_3)$$
$$F = r_2^2 - r_3^2 - x_2^2 + x_3^2 - y_2^2 + y_3^2$$

Coordinate $(x,y)$ is then given by:

$$x = \frac{(CE - FB)}{(EA - BD)},\ y = \frac{(CD - AF)}{(BD - AE)}$$

Note that there is an exception when EA=BD.

[7] https://en.wikipedia.org/wiki/Radical_axis