Control of Mobile Robotics

CDA4621

Spring 2019

Lab 3

Bug Algorithms

Total: 100 points

Due Date: 3-28-19 by 8am

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

A. Lab Requirements

The lab requires use of the "Robobulls-2018" robot hardware provided at no charge for the duration of the semester. Required software can be downloaded free of charge from the web. All labs are to be done by teams of one or two students depending on robot availability. Note that no diagrams or descriptions by hand will be accepted. 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.

A.1 Hardware Requirements

The "Robobulls-2018" is the main robot hardware used for the course.

A.2 Software Requirements

Python

B. Camera Descriptions

This lab introduces you to the use of the camera and requires you to implement a bug algorithm to navigate around obstacles and reach the goal position. This section will provide you with necessary information to use the camera and it will introduce you to some usual issues and pitfalls encountered. Understanding this section is a prerequisite for the rest of the lab.

B.1 Basic Camera Usage

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 and then frames can be retrieved at any time using the "read" function. These frames are in the "Mat" format, which allows the frame to easily be manipulated or displayed on screen with the "imshow" function.

B.2 Simple Blob Detector Usage

Read the supplementary material[2] on using the Simple Blob Detector feature in OpenCV.

---

[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/

One of OpenCV's features is Simple Blob Detector, which allows for easy detection of blobs (groups of connected pixels that are the same color) in an image. This feature is initialized with a set of "Params" that specify the allowable size, color, and shape of blobs. Then a "Mat" object is fed into the detector, which is a raw grayscale or color image. The detector then 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, then 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 the "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, this 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

C.1 Task 1 – Goal Facing

Implement a program called "faceGoal.py". The program should rotate the robot until it is facing the goal. Then the robot must remain still until the robot is moved or the goal itself is moved.

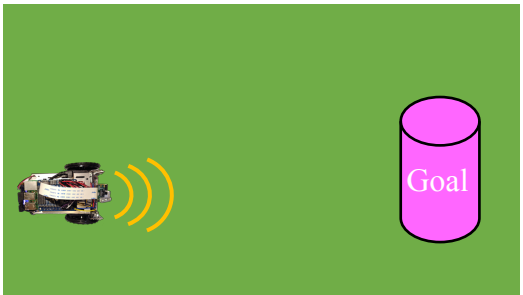

Fig. 1: Visualization of robot and goal

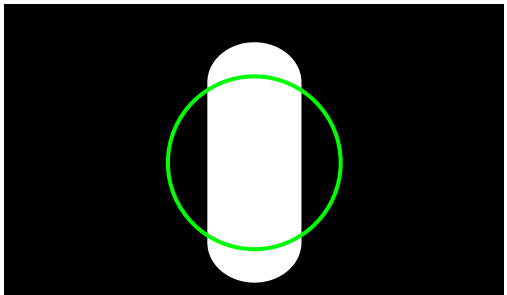

Fig. 2: Masked camera image with keypoint

The goal itself is a large cylinder that is 2 feet tall and 8 inches in diameter. It is covered in non-reflective bright pink paper in order to be easy to detect in any environment. Robots must be able to locate this goal from a distance of up to 8 feet away (Fig. 1). Additionally, the robot may be placed in such a way that it is facing away from the goal, so it would not see the goal from the

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

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

camera view until it turns around. There are no other obstacles on the floor between the robot and the goal.

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 itself is visible to the detector (Fig. 2). 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 in-place rotation of the robot's wheels. Be aware that the goal may not be visible at all if the robot is facing in the opposite direction. You will need to account for this by rotating the robot until the goal is in view, and then using proportional control to directly face the goal. Also, be aware that 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". 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. Once the robot reaches the goal and stops, it needs to continue checking its distance to the goal in the case that the robot or the goal itself is suddenly moved somewhere else.

C.3 Task 3 – Bug Algorithm

Implement a program called "bugAlgorithm.py". The program should use any bug algorithm (preferably "Bug 0") to move the robot towards the goal while navigating around any obstacles.

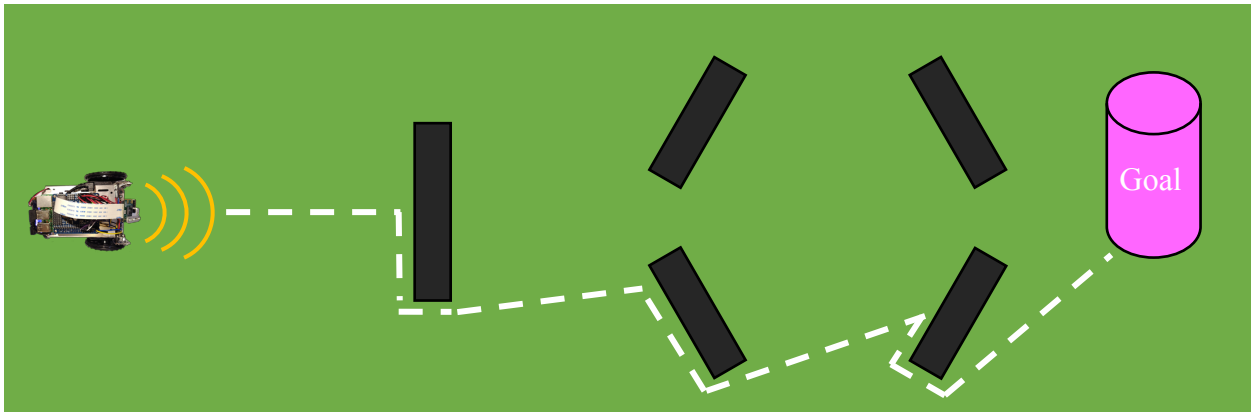

Figure 3: Visualization of bug algorithm

This task is an extension of task 2. Up to three obstructing obstacles will be placed between the robot and the goal (Fig. 3). 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 still needs to perform goal-

facing 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), 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.

While it is not a requirement, it is suggested that you implement this program using a *state machine*[4]. At least three states could be used: goal-facing, motion-to-goal, and wall-follow, 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 would switch to the appropriate state to handle this change.

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 (70 points)

The task presentation needs to be scheduled with the TA. Close to the project due date, a time table will be made available online from which groups will be able 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. To do so, the submitted code will be downloaded at that time from Canvas and uploaded to the robot.

The following table shows the rubric for tasks presentations:

- Task 1 – 15 points
  - Robot can face the goal when positioned approx. 4ft away (5 points)
  - Robot can face the goal when positioned approx. 8ft away (5 points)
  - Robot tracks the goal when the goal is moved (5 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 – 15 points
  - Robot can do motion-to-goal with no obstacles, approx. 4ft away (5 points)
  - Robot can do motion-to-goal with no obstacles, approx. 8ft away (5 points)
  - Robot restarts motion-to-goal when the goal is moved (5 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 – 40 points
  - Robot can do motion-to-goal and wall-following with 1 obstacle (10 points)
  - Robot can do motion-to-goal and wall-following with 2 obstacles (15 points)
  - Robot can do motion-to-goal and wall-following with 3 obstacles (15 points)
  - Robot hits obstacle (-3 points each time)
  - Robot moves slower than 1 in/s (-10 points)
  - Robot fails to switch back to motion-to-goal (-4 points each time)

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

To get full credit on each item, the robot must perform the specified action successfully on the first trial. If unable to do so: (a) 75% credit will be given if done on second trial, (b) 50% if done on the third trial, and (c) 25% if done on the fourth trial. No more than 4 trials will be allowed per rubric item.

D.2 Task Report (30 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. List of problems found while performing task 1 and/or task 2 (at least 3 in total). A brief description should be given for each problem and how it was dealt with. Otherwise state with why it could not be solved. (6 points)
3. A diagram with brief explanation of the bug algorithm. Do NOT include the code as an explanation. (6 points)
4. List of problems found while performing the bug algorithm (at least 3). A brief description should be given for each problem and how it was dealt with. Otherwise state why it could not be solved. (6 points)
5. Explain the process of analyzing a video frame to extract the goal position. (6 points)
6. Link to a video where the robot is shown performing all the tasks. (2 points)
7. Conclusions where you analyze any issues you encountered when running the tasks and how they could be improved. Conclusions need to show an insight of what the group has learned (if anything) during the project. Phrases such as "everything worked as expected" or "I enjoyed the project" will not count as conclusions. (3 points)