Lab 3: Motion Planning

CDA 4621
Fall 2024
Lab 3: Motion to Goal and Bug 0
Due Date: 11-11-2024 by 11:59pm

Objective

This lab will teach you how to plan motion for a robot to reach a goal while avoiding obstacles. This lab will teach you about how to utilize a camera with object detection and the bug zero algorithm to navigate through an obstacle-rich environment to reach a goal location.

Requirements

Programming: Python
Robot: Webots 2023b (FAIRIS)

RGBD Camera with Object Detection

In FAIRIS_Lite, the robot is equipped with an RGBD camera, which can be accessed via the `rgb_camera` attribute of the `MyRobot` class. This sensor allows for both image capture and object recognition, providing detailed information about objects detected within the robot's field of view.

The function `robot.rgb_camera.getRecognitionObjects()` returns a list of recognized objects, each containing attributes such as position, size, and color. This is crucial for detecting landmarks, identifying obstacles, and enhancing navigation capabilities.

Key Functions and Their Usage:

- `robot.rgb_camera.getRecognitionObjects()`: This function returns a list of recognized objects. Each recognized object is a Webots `RecognitionObject` with various attributes such as:
  - `landmark.getId()`: Returns a unique identifier for the detected object.
  - `landmark.getPosition()`: Returns the 3D position of the object relative to the camera as an array [X, Y, Z].
  - `landmark.getSize()`: Returns the relative size of the object as an array [Y, Z].
  - `landmark.getPositionOnImage()`: Returns the 2D position of the object on the image as an array [X, Y], where (0,0) represents the top-left corner.
  - `landmark.getSizeOnImage()`: Returns the 2D size of the object on the image as an array [X, Y], representing the object's bounding box.
  - `landmark.getColors()`: Returns the RGB color of the object as an array [R, G, B].

Example Usage:

```
rec_objects = robot.rgb_camera.getRecognitionObjects()

# if camera has detected an object
if len(rec_objects) > 0:
    # extract detected object
    landmark = rec_objects[0]
    # object ID
    object_id = landmark.getId()
    # object position relative to the camera [X, Y, Z]
    object_position = landmark.getPosition()
    # object relative size [Y, Z]
    object_size = landmark.getSize()
```

RGBD Camera (cont.)

```
# object position on image [X, Y]
object_position_on_image = landmark.getPositionOnImage()
# object size on image [X, Y]
object_size_on_image = landmark.getSizeOnImage()
# object color [R, G, B]
object_color = landmark.getColors()
```

This code checks if any objects are detected by the RGBD camera. If an object is detected, it extracts the first object and prints details such as the object's ID, position, size, and color.

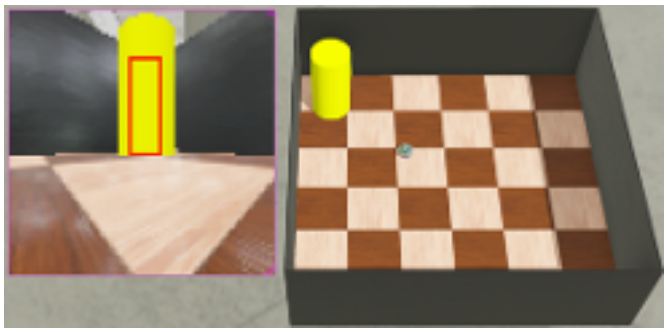

FIGURE 1. RGBD Camera View in Webots with Object Detection

For more information on the available functions and object attributes, please refer to the Webots Documentation and the FAIRIS_Lite Documentation.

Motion to Goal

One of the core objectives in mobile robotics is navigating towards a goal, referred to as "motion to goal." In this assignment, your robot will navigate toward a goal represented by a yellow cylinder. The task involves rotating the robot to face the goal, moving toward it, and stopping at a specific distance in front of it.

Steps for Motion to Goal:

(1) **Detect the Goal:** The robot must first identify the goal object (the yellow cylinder) using its RGBD camera. Use the object recognition function `robot.rgb_camera.getRecognitionObjects()` to detect the goal.

(2) **Center the Goal in the Camera Frame:** Once the goal is detected, the robot should rotate in place to center the object within the camera's frame. This is done using the goal's position on the image, combined with a PID controller to rotate the robot smoothly.

(3) **Move Towards the Goal:** After the goal is centered in the camera frame, the robot should move forward while ensuring it stops a set distance (e.g., 0.5 meters) in front of the goal.

Helpful Hints:

- The position of the object in the image can be obtained from `landmark.getPositionOnImage()`. The robot should rotate until the x-coordinate of the object is centered in the image.
- A simple PID controller can be used to adjust the robot's rotational speed based on the error between the object's x-coordinate and the center of the image.
- Once the object is centered, move forward and stop the robot when the object is a set distance away using a forward motion PID controller (similar to the one you created in Lab 2).

Motion to Goal (cont.)

- Think about how you will handle when the robot can not see the goal.

Bug Zero Algorithm

The Bug Zero algorithm is a simple, reactive path-planning algorithm used by robots to navigate around obstacles while moving toward a goal. The robot moves directly towards the goal unless it encounters an obstacle, in which case it follows the edge of the obstacle until it can resume a direct path to the goal.

Key Concepts:

- **Straight Line to the Goal:** The robot starts by moving in a straight line towards the goal. This is the primary behavior when no obstacles are detected.
- **Obstacle Avoidance:** If the robot detects an obstacle in its path (using sensors such as the LIDAR or RGBD camera), it must navigate around the obstacle. In the Bug Zero approach, the robot follows the boundary of the obstacle until it can safely move directly toward the goal again.
- **Returning to the Goal Path:** As the robot follows the obstacle's edge, it continuously checks whether it can resume moving directly toward the goal. Once the path is clear, the robot returns to the direct line toward the goal.

Steps to Implement Bug Zero:

(1) **Move Towards the Goal:** Initially, the robot moves straight toward the goal (e.g., a yellow cylinder). It uses the goal's position detected by its sensors to guide its movement.
(2) **Detect Obstacles:** If an obstacle is detected (e.g., using LIDAR or the camera), the robot should switch to obstacle avoidance mode, where it moves around the object.
(3) **Follow the Obstacle's Boundary:** The robot should trace the perimeter of the obstacle while continuing to monitor whether it can resume its direct path to the goal. You can apply the wall-following behavior developed in Lab 2 to assist in this process.
(4) **Return to the Direct Path:** Once the robot finds an open path toward the goal, it should stop following the obstacle and return to the goal path, continuing until it either reaches the goal or encounters another obstacle.

Helpful Hints:

- Use LIDAR or the RGBD camera to detect obstacles in the robot's path. The LIDAR can give you precise distance measurements, while the camera can help recognize obstacles and landmarks.
- When following the obstacle's edge, make sure the robot stays close enough to avoid losing track of the obstacle but far enough to not collide with it.
- **Reusing Wall-Following Behavior:** You can reuse the wall-following algorithm you developed in Lab 2 to help the robot trace the obstacle's boundary efficiently. This will save you time and help with the implementation of Bug Zero.
- Continuously check whether the path to the goal is clear while tracing the obstacle. Once a clear path is found, the robot should stop tracing the obstacle and resume direct motion toward the goal.

The Bug Zero algorithm provides a robust yet simple solution for navigating in environments with obstacles. Although it does not guarantee the shortest path, it ensures the robot can avoid obstacles and eventually reach the goal.

Task 1: Motion to Goal

The goal of this task is for the robot to reach a yellow cylinder and stop when it is 0.5 meters away from it. There are no obstacles between the robot and the goal, as illustrated in Figure 2. The robot should be tested by starting from different locations and orientations.

Task Details:

Task 1: Motion to Goal (cont.)

- **Initial Condition:** The robot may begin facing away from the goal, meaning it will need to turn until the goal is detected.
- **Goal Detection:** The robot should use its RGBD camera to recognize the yellow cylinder (goal) and determine both the distance and orientation to it.
- **PID Controller:** You may implement a PID controller to help guide the robot smoothly toward the goal. The input to the controller could be the distance and orientation of the yellow cylinder relative to the robot, while the output would be the linear velocities of the left and right wheels.
- **Navigation and Stopping:** Once the goal is detected, the robot should move toward it and stop once it is exactly 0.5 meters away. The camera can be used to identify the goal's position and orientation, and the range sensors (e.g., LIDAR) can be used to compute the distance to the goal.

Testing and Timing:

- Test your solution in the *maze5.xml* world file.
- During the task, the robot should turn, locate the goal, move toward it, and stop at the correct distance.
- Test the robot from multiple starting positions and orientations to ensure robustness.
- For the video recordings, be sure to capture the robot performing motion to goal from different locations and orientations. This can be done by pausing the simulation and clicking and dragging the robot or the yellow cylinder.

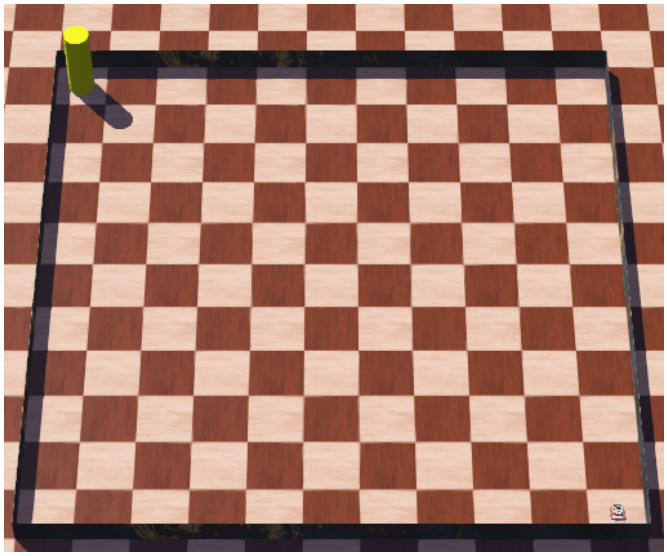

FIGURE 2. Yellow Cylinder Goal for Task 1: Motion to Goal (*maze5.xml*)

Task 2: Bug Zero Algorithm

In this task, the robot should implement the Bug Zero algorithm to navigate toward the goal while avoiding obstacles, as shown in Figure 3. The goal "G" is represented by a yellow cylinder, similar to the one used in Task 1 (Figure 2). The robot starts at position "S" and must reach the goal while avoiding any obstacles in its path.

Task Details:

- **Initial Condition:** The robot begins at position "S" and must move toward the goal "G."
- **Bug Zero Algorithm:** The robot should implement the Bug Zero algorithm to handle obstacles. If an obstacle is detected, the robot should navigate around it, staying no further than 0.5 meters away from the obstacle.

Task 2: Bug Zero Algorithm (cont.)

- **Obstacle Avoidance:** The robot should be capable of making either left or right turns when tracing the obstacle's boundary, depending on the environment.
- **Sensors:** Use the RGBD camera and/or range sensors (e.g., LIDAR) to compute the distance and orientation to the goal as well as the obstacles.
- **Completion Time:** The task should be completed within 3 minutes, meaning the robot must reach the goal within that time while successfully navigating around obstacles.

Testing and Timing:

- Test your solution in the *maze6.xml* world file.
- Test the robot from position "S" in multiple trials to ensure it can reach the goal consistently without collisions.
- The robot should efficiently avoid obstacles, staying within 0.5 meters of the obstacle boundary, and resume its path to the goal once the obstacle is cleared.
- Ensure the robot can perform both left and right turns around obstacles.

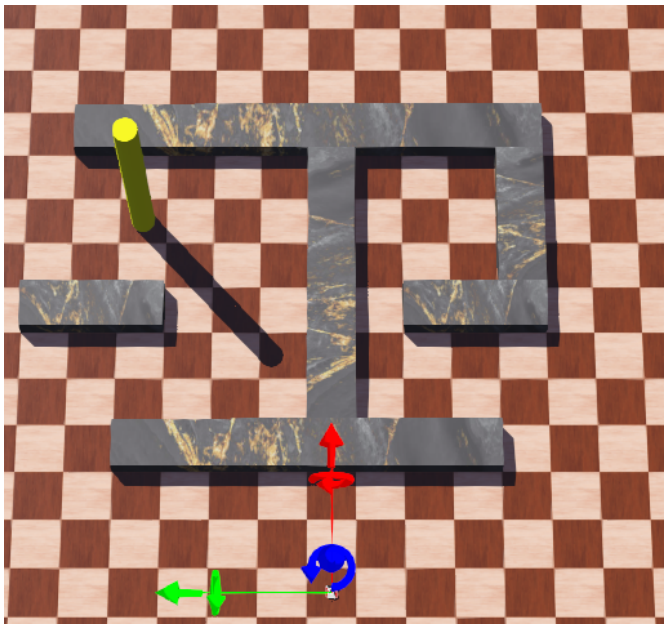

FIGURE 3. World setup for Task 2: Bug Zero Algorithm (*maze6.xml*)

Code Evaluation Task 1 (30%)

Robot reaches the cylinder from any starting position and orientation (15 points)

- **Full Marks (15-13 points):** The robot reliably reaches the yellow cylinder from any starting position and orientation. The robot smoothly navigates toward the cylinder without deviating significantly.
- **Partial Marks (12-8 points):** The robot reaches the cylinder but may have slight deviations or take longer from some starting positions or orientations.
- **Minimal Marks (7-4 points):** The robot struggles to reach the cylinder from some starting positions or orientations, exhibiting significant deviations or slow performance.
- **No Marks (3-0 points):** The robot fails to reach the cylinder from most starting positions or orientations.

Robot finds the cylinder when the cylinder is not seen in the camera (10 points)

Task 1: Motion to Goal (cont.)

- **Full Marks (10-9 points):** The robot successfully turns and locates the yellow cylinder when it starts with the cylinder out of view. The robot quickly identifies the goal and begins navigating toward it.
- **Partial Marks (8-6 points):** The robot finds the cylinder, but the process is slower or less efficient, possibly taking too long to turn or locate the cylinder.
- **Minimal Marks (5-3 points):** The robot occasionally fails to find the cylinder, or it takes significant time to locate it when out of view.
- **No Marks (2-0 points):** The robot fails to locate the cylinder when it starts with the goal out of view.

Robot stops 0.5 meters or less from the cylinder (5 points)

- **Full Marks (5 points):** The robot stops exactly 0.5 meters from the yellow cylinder with high accuracy and minimal oscillation.
- **Partial Marks (4-3 points):** The robot stops close to 0.5 meters but may overshoot slightly or stop just before reaching the desired distance.
- **Minimal Marks (2-1 points):** The robot stops significantly outside the 0.5-meter threshold, either too early or too late.
- **No Marks (0 points):** The robot fails to stop within 0.5 meters of the cylinder.

Robot hits the cylinder (-5 points)

Deduction: If the robot hits the cylinder during the task, 5 points will be deducted from the overall task score.

Code Evaluation Task 2 (60%)

Robot completes correct Bug Zero algorithm and stops within 0.5 meters of the cylinder (20 points)

- **Full Marks (20-18 points):** The robot successfully completes the Bug Zero algorithm, reaching the goal and stopping exactly 0.5 meters from the yellow cylinder.
- **Partial Marks (17-13 points):** The robot completes the Bug Zero algorithm but may stop slightly outside the 0.5-meter threshold.
- **Minimal Marks (12-7 points):** The robot completes the Bug Zero algorithm but often stops too far from the cylinder or misses key elements of the algorithm.
- **No Marks (6-0 points):** The robot fails to complete the Bug Zero algorithm or does not stop near the cylinder.

Robot moves correctly around walls (15 points)

- **Full Marks (15-13 points):** The robot smoothly navigates around walls, maintaining the required distance of 0.5 meters and demonstrating consistent obstacle avoidance.
- **Partial Marks (12-8 points):** The robot navigates around walls but may struggle to maintain the 0.5-meter distance, occasionally coming too close or too far.
- **Minimal Marks (7-4 points):** The robot struggles to navigate around walls, often coming too close or deviating significantly from the desired path.
- **No Marks (3-0 points):** The robot fails to navigate around walls correctly.

Robot switches correctly from motion to goal to wall following (10 points)

- **Full Marks (10-9 points):** The robot seamlessly switches from moving towards the goal to following the wall when an obstacle is detected.
- **Partial Marks (8-6 points):** The robot switches from motion to goal to wall following, but the transition may be delayed or inconsistent.
- **Minimal Marks (5-3 points):** The robot struggles with the transition, occasionally failing to recognize obstacles and switch to wall following.

Task 2: Bug Zero Algorithm (cont.)

- **No Marks (2-0 points):** The robot does not switch to wall following when encountering an obstacle.

Robot switches correctly from wall following to motion to goal (10 points)

- **Full Marks (10-9 points):** The robot correctly transitions from following the wall back to moving towards the goal when the path becomes clear.
- **Partial Marks (8-6 points):** The robot switches back to motion to goal, but the transition may be slow or inefficient.
- **Minimal Marks (5-3 points):** The robot occasionally fails to switch back to motion to goal, continuing to follow the wall unnecessarily.
- **No Marks (2-0 points):** The robot does not switch back to motion to goal after following the wall.

Robot performs correct left and right turn algorithm (5 points)

- **Full Marks (5 points):** The robot performs both left and right turns correctly without switching between them inappropriately.
- **Partial Marks (4-3 points):** The robot performs the turn algorithm but may occasionally exhibit slight errors or switch between left and right turns unnecessarily.
- **Minimal Marks (2-1 points):** The robot struggles with performing correct turns, often switching incorrectly between left and right turns.
- **No Marks (0 points):** The robot does not correctly perform left and right turns as expected.

Robot hits the cylinder (-5 points)

Deduction: If the robot hits the cylinder during the task, 5 points will be deducted from the overall task score.

Robot travels a hardcoded path defined in advance (-10 points)

Deduction: If the robot travels along a hardcoded path rather than implementing the Bug Zero algorithm dynamically, 10 points will be deducted from the overall task score.

Report & Video Evaluation (10%)

A project submitted without a report, or without including the corresponding task video, will not be graded and will receive a "0" for the complete lab.

Report Sections: The report should include the following (points will be deducted if anything is missing):

- **Mathematical Computations:** Show how you calculated the speeds of the left and right servos given the input parameters for each task.
- **Conclusions:** Analyze any issues encountered when running the tasks and discuss how these could be improved. Discuss your results, including any navigation errors. Conclusions need to reflect an insight into what was learned during the project. Statements like "everything worked as expected" or "I enjoyed the project" will not count as valid conclusions.
- **Video:** Upload the video to Canvas showing the robot executing the task (details below).
- **Authorship Statement:** Include and sign the following statement at the end of the report:
  "I developed all the code and written report with the following exceptions:
  [Student name: signature, date]
  [Code sections (specify lines): references and additional comments]
  If any external tools were used to generate code, you need to include the prompt and output."

Task 1 Video: Record a video of the robot executing the "Motion to Goal" task. The video should demonstrate:

- The robot starting from various positions and orientations, successfully detecting and moving towards the yellow cylinder.
- The robot finding the yellow cylinder when it starts outside of the camera's initial field of view and turning to locate it.
- The robot stopping exactly 0.5 meters from the yellow cylinder, with minimal or no overshoot.
- Any issues such as the robot hitting the cylinder or stopping too early should be highlighted. If applicable, mention how these issues were resolved or could be improved.

Task 2 Video: Record a video of the robot executing the "Bug Zero" algorithm. The video should demonstrate:

- The robot starting from position "S," moving towards the goal (yellow cylinder), and correctly switching from motion-to-goal to wall following upon encountering obstacles.
- The robot following the boundary of obstacles, ensuring it stays within 0.5 meters of the wall, and switching back to motion-to-goal when the path is clear.
- The robot successfully performing both left and right turns without switching between them incorrectly.
- The robot reaching the goal (yellow cylinder) and stopping within 0.5 meters of it.
- Any issues such as the robot hitting the cylinder or traveling along a hardcoded path should be explained. If applicable, mention how these issues were resolved or could be improved.

Lab Submission

For Lab 3 there are two separate Canvas Assignments: Lab 3 Code Submission and Lab 3 Report Submission. Please follow the Submission policy outlined below. **Note:** Failure to adhere to the submission policy will result in a reduction of 20 points.

Code Submission Policy

Students will need to upload a Python file (i.e. Webots controller) for each Task. The Python file should follow this naming convention *USF_ID_LabX_TaskY.py*, where *X* is the lab number and *Y* is the task number. Example *rockybull5_Lab1_Task1.py*. Additionally, if a student has created any functions that are not present in the controller but rather the *MyRobot.py*, the student will also need to upload that python file as well using the following naming convention *USF_ID_MyRobot.py*.

If custom functions are used in a student's submission and they are not in the controller or the MyRobot Python files, an automatic Zero will be assigned for the code submission.

Each file should be uploaded to the same submission. **DO NOT** upload a zip file, Canvas allows multiple uploads per submission. Failure to meet these submission policies will result in 20 points off.

Report Submission Policy

Students will need to upload their videos to their USF OneDrive or USF Sharepoint and generate a shareable link. Please ensure that you make it viewable to everyone or at the very least the TA and the Professor (use their emails to add them to the access list).

Students will need to upload a PDF file of their lab report. Be sure that the lab report contains the link to the videos and any Metrics requested by the Lab. These Metrics will be used to determine which student receives the extra credit. Use the naming convention *USF_ID_LabX_Report.pdf*. Only PDFs will be accepted.

If the student used ChatGPT to generate code they will also need to provide the conversation used to generate the code. This can either consist of screenshots of the conversation or a shareable link that the system provides. This should be a separate document and not part of the report.

Each file should be uploaded to the same submission. **DO NOT** upload a zip file, Canvas allows multiple uploads per submission. Failure to meet these submission policies will result in 20 points off.

TA may ask you additional questions to gauge your understanding via Canvas Message or MS Teams. Failure to reply may result in point deduction.