Team 02 EECS 467 Report

Christopher Nguyen, Kun Huang, Zhihao Ruan

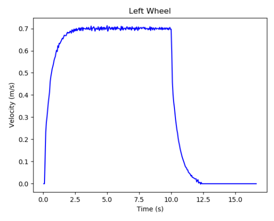
***Abstract*—In this report, we discuss about the four different control algorithms we implemented for driving the Mbot. We first implemented a closed-loop velocity to control the angular speed of the wheels and we tuned the PID for three different configurations: over-damped, under-damped and critically-damped. Next, we implemented a open-loop controller which drives the Mbot for a set amount of time and didn't use feedback from the odometry to minimize any errors. We improved on this by implementing a closed-loop controller to minimize the error between Mbot's odometry and the desire pose. Finally, we implemented another closed-loop controller but we calculate the odometry by using a lidar instead of encoders. This provided more accurate updates to the Mbot's odometry and thus a more accurate driving in the real world.**

I. Results

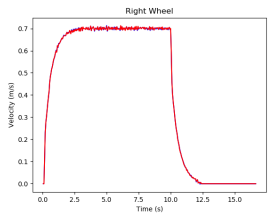
A. Closed-Loop Velocity Control

We first tuned a closed-loop velocity controller for each of the wheels. These controllers will be used in the later tasks to set the velocities determined from other controllers. The PIDs were the same for both controllers and the set point was set to 7m/s. We tuned the PIDs to achieve three different configurations: overdamped, underdamped and critically damped.

$$Overdamped : k_p = 0.4, k_i = 2, k_d = 0$$

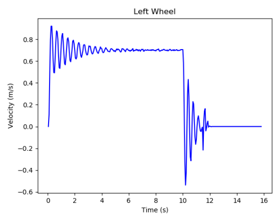


Left

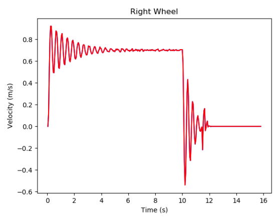


Right

$$Underdamped : k_p = 0.4, k_i = 20, k_d = 0$$

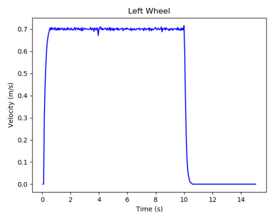


Left

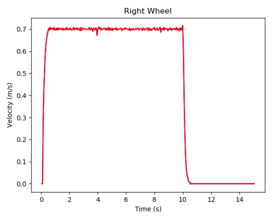


Right

$$Critically - damped : k_p = 0.4, k_i = 6, k_d = 0$$



Left



Right

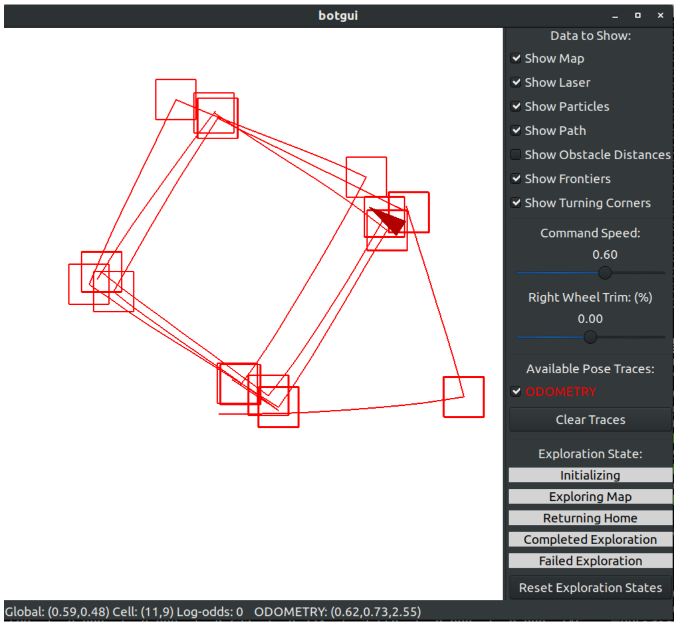
B. Open-Loop Control

This is a simple control algorithm which doesn't consider the robot's position and just applies a constant velocity for a predetermined amount of time. We implemented this control by utilizing the velocity controller from the previous task to apply a constant velocity to the wheels. The algorithm to drive the Mbot in a square is described below. The time to apply the velocity was calculated with $t = \frac{distance}{velocity}$, and then later tweaked with experience.

Algorithm 1 Open Loop Control

```
for $i = 0$ to 3 do
    for $j = 0$ to 4 do
        mb_motor_msg.ang_v = 0.0
        mb_motor_msg.trans_v = 0.2
        rc_nanosleep(1e9)
        mb_motor_msg.ang_v = -(PI * 0.25)
        mb_motor_msg.trans_v = 0.0
        rc_nanosleep(1e9)
    end for
end for
```

This algorithm was not very accurate in driving. The Mbot would always drift off course and could not correct itself, resulting in the graph shown in figure 1.



```
Global: (0.59,0.48) Cell: (11,9) Log-odds: 0   ODOMETRY: (0.62,0.73,2.55)
```

Fig. 1: Driving in a square with a open-loop control algorithm

C. Closed Loop Control with Odometry

We improve our driving by next implementing a closed loop control with odometry. We computed the odometry by first computing the forward and angular motion. These are computed as the following:

$$ds = \frac{enc_{left} + enc_{right}}{2} \times \frac{2\pi R}{48 \times 34.014}$$

$$d\theta = \frac{enc_{right} - enc_{left}}{B} \times \frac{2\pi R}{48 \times 34.014}$$

where $R = 0.04$ and $B = 0.21$. Once we get the change in motion, we use them to offset Mbot's odometry as such:

$$odo_x = odo_x + ds \times \cos(odo_{theta})$$

$$odo_y = odo_y + ds \times \sin(odo_{theta})$$

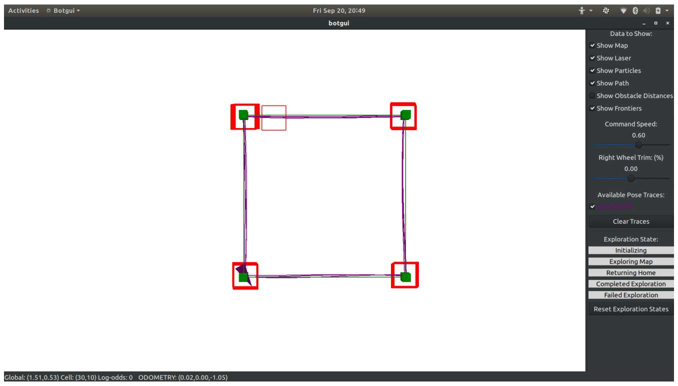
$$odo_{theta} = odo_{theta} + d\theta$$

We use the motion changes to calculate the instantaneous translational and angular velocities at that timestamp.

$$\nu_t = ds/DT$$

$$\omega_t = d\theta/DT$$

where DT is the inverse of the sample rate which we read encoder counts. With the computed odometry, we use two seperate closed-loop controller. One controller outputs the translational velocities and the error it tries to minimize is the euclidean distance between Mbot's current odometry and the setpoint position. The second controllers determines the angular velocities and minimizes the difference between Mbot's theta and the setpoint theta. We use the velocity controller from Task III to set both the translational and angular velocities.
These two controllers significantly made Mbot's driving a lot more accurate. As shown in figure 2, the graph shows a near perfect square and display the corrections that the Mbot makes whenever it drifts off course, unlike the driving captured in figure 1.



```
Global: (1.51,0.53) Cell: (30,10) Log-odds: 0   ODOMETRY: (0.02,0.00,-1.05)
```

Fig. 2: Captured Mbot's odometry when driving in a square with closed-loop controllers

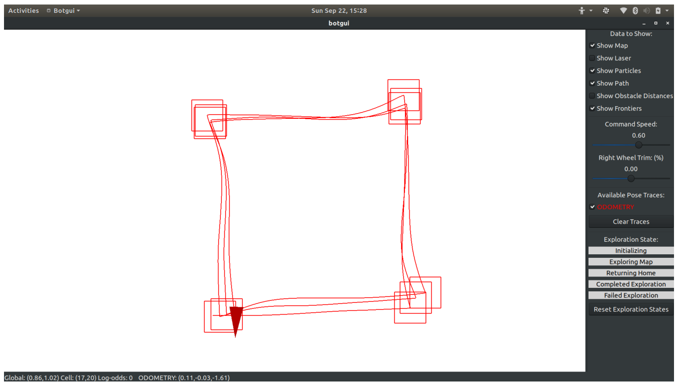
D. Closed Loop Control with Laser Scans

Algorithm 2 Closed-loop Laser-based Controller

```
Input: Distance to Walls $d$, angle between robot and right
    wall $\theta_{rightwall}$ (counter clock-wise)
Output: Command of forward velocity and angular velocity
  Initialisation:
  State = DRIVE
  DESIRED_DIST $= 0.5$
  LOOP Process:
  if State == TURN then
    $v = 0$
    if $d_{backwall} <$ DESIRED_DIST $+ 0.1$
    && $d_{frontwall} < 2 -$ DESIRED_DIST $- 0.1$ then
      $\omega = 0$
      State = DRIVE
    else
      $\omega =$ const_turn
    end if
  else if State == DRIVE then
    if $d_{frontwall} <$ DESIRED_DIST $+ 0.1$ then
      $\omega = 0$
      $v = 0$
      State = TURN
    else
      $v = \min(1.2 \times d_{frontwall} + 0.05, 0.1)$
      $\omega = -K_1 \times \theta_{rightwall} - K_2/v \times (d_{rightwall} -$
      DESIRED_DIST)
    end if
  end if
  return $v$ and $\omega$
```

Algorithm 3 Distance to Walls (Right Wall)

```
Input: Lidar data
Output: Distance to right wall $d$, angle between robot and
    right wall $\theta_{rightwall}$ (counter clock-wise)
  Initialisation:
  $d = \inf$
  $\theta_d$
  USABLE_DATA $= 0.2$
  LOOP Process:
  for $ray$ in $num\_of\_rays$ do
    if $ray.theta \in (75°, 105°)$ then
      if $ray.range >$ USABLE_DATA then
        if $ray.range < d$ then
          $d = ray.range$
          $\theta_d = ray.theta$
        end if
      end if
    end if
  end for
  return $d$ and $\theta_d$
```

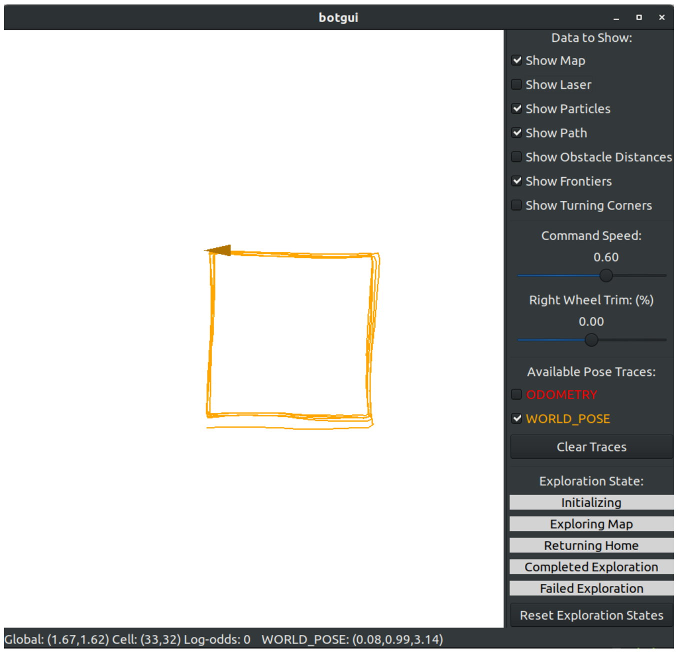


Global: (0.86,1.02) Cell: (17,20) Log-odds: 0   ODOMETRY: (0.11,-0.03,-1.61)

Fig. 3: Closed Loop Control with Laser Scans, visualized using odometry.

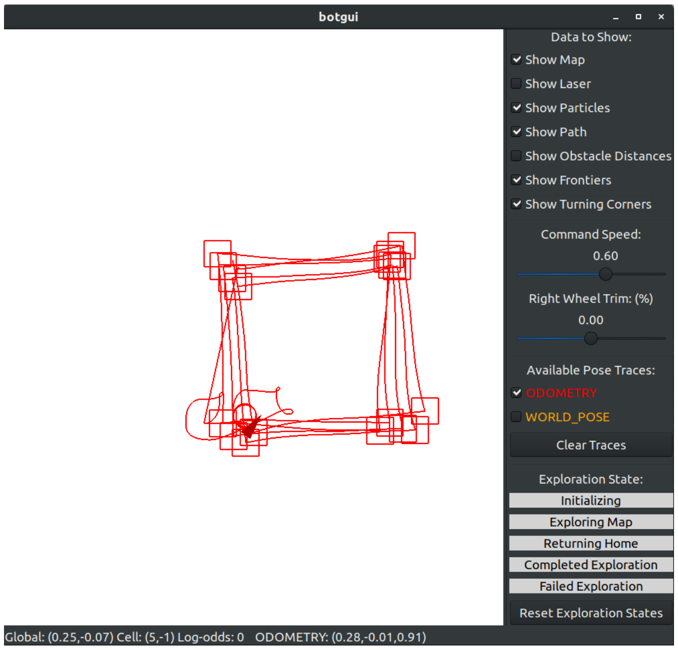
As you can see in figure 3, the odometry drifts gradually after running loops over loops.

We also visualize the world frame pose using distance to the right wall and distance to the front wall. The comparison between visualization using odometry and visualization using world frame pose of the same experiment is shown below.



Global: (1.67,1.62) Cell: (33,32) Log-odds: 0   WORLD_POSE: (0.08,0.99,3.14)

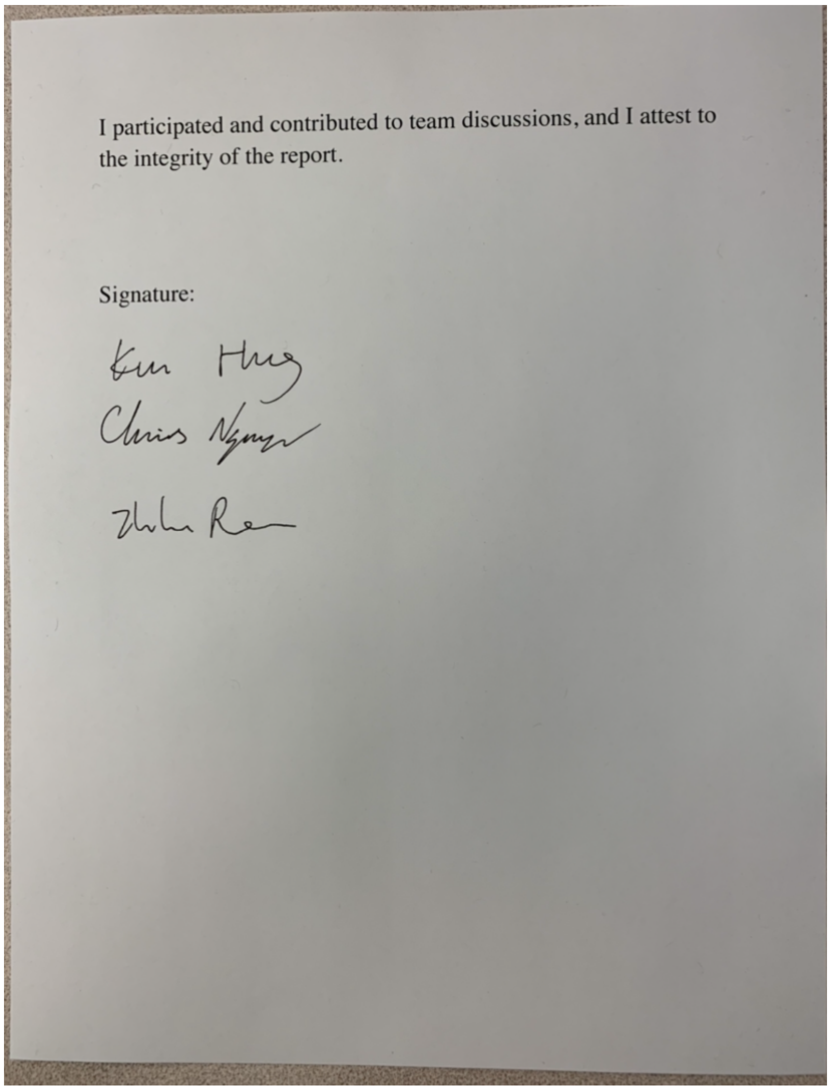
Fig. 4: Closed Loop Control with Laser Scans, visualized using world frame pose.



Global: (0.25,-0.07) Cell: (5,-1) Log-odds: 0   ODOMETRY: (0.28,-0.01,0.91)

Fig. 5: Closed Loop Control with Laser Scans, visualized using odometry.

As the robot runs, the odometry gradually drifts. But the world frame pose stays stable since we are using world frame pose to correct the trajectory of the robot.



I participated and contributed to team discussions, and I attest to the integrity of the report.

Signature:

[illegible]

[illegible]

[illegible]