Deliverable 5: Final Report

Roderick Wu (1007829806), Yang Xu (1007932775)

Section 1. Introduction

This culminating project aims to combine the knowledge acquired from previous labs and put them together. Previous labs introduced robotic sensing, actuation, control, and localization. Using the humble Turtlebot, our task is to deliver mail on a closed loop circuit market with coloured patches representing offices. Given the topological map of the circuit, our Turtlebot is expected to "deliver mail" (stop and perform a choreographed manoeuvre) to three specific offices out of twelve total, which are numbered. However, the robot is only able to detect the colour of the office. The Turtlebot will need to implement Bayesian localization to estimate its true location and use the line-following control developed in a previous lab to stay on the circuit.

Section 2. Description of Robot Platform

The Turtlebot 3 Waffle Pi comes equipped with an on-board Raspberry Pi as the computer, an OpenCR board, a Raspberry Pi camera, a 360° LIDAR that measures range and bearing, an inertial motion unit (IMU), a compass, and gyroscopes. Of the sensors described above, we will be relying on the onboard camera to measure the colour directly underneath the robot for state estimation (ROB301, 2023).

The Turtlebot 3 Waffle Pi is a differentially steered robot with two wheels and each wheel is driven by a Dynamixel actuator. The motors can be operated in one of six possible modes, but for our project, we will only be controlling the robot by sending commands to set its linear and rotational velocities.

The software in the Turtlebot consists of firmware from the OpenCR board and 4 ROS packages. For this project, we will be writing our own ROS node and using some of the nodes provided by the project. The main node to be run, *final_project.py*, includes our calculations for localization and line following. It accomplishes this by subscribing to the topics "mean_img_rgb" — published by the node *camera_rgb* — and "line_idx" — published by *camera_mono*. This information is used for calculations, with the resulting desired behaviour of the Turtlebot being published onto the topic "cmd_vel".

Section 3. Overview of Solution Strategy

Our solution to this project was to program a ROS node that maintains an estimated state of the robot and output controls to the Turtlebot accordingly. The general structure of our solution can be stated as the following:

1. Receive colour and line readings from the Turtlebot and process them into usable measurements.
2. Update the estimated state of the robot based on the colour readings if our algorithm decides that the robot is at an office, not on a line.
3. Based on the estimated state, our ROS node may publish a variety of commands:
    a. If the robot is on a line/between offices: use a PID controller to calculate an angular velocity that corrects any deviation of the robot from the line based on the line measurements it receives.
    b. If the robot is on a coloured patch that is not the desired office: continue moving straight at the same orientation and velocity as just before entering the patch.
    c. If the robot is at the desired office: deliver the mail by turning 90° on the spot, pausing for a second, turning -90°, and continue moving straight as described in case (b) to find the next desired office.

The specific implementation details for each operation are described in the following section.

Section 4. Technical Details On Design Methodology

In this section, we will describe our specific approach to performing the required task.

First, our solution node processes the raw information it receives from the Turtlebot into usable measurements. The position of the line is provided directly from the “line_idx” topic and can be directly used by the PID controller (described later on). However, the “mean_img_rgb” topic only provides average RGB values read by the camera. We want to determine the exact colour the readings represent (either “line” or one of the colours specified in the given measurement model) to perform Bayesian Localization and send commands based on state (no state update if measurement is “line”, but needed for path following). To do so, we first convert the RGB readings to HSV (through experimentation, we found that HSV worked better for identifying colours in most cases). Then, to map the HSV readings to one of the office colours or “line”, we create a map of what HSV readings correspond to each colour (pre-determined by measuring each colour/line) and compare it to the current reading. Since the readings are noisy, we cannot match them to any particular colour exactly. Instead, we calculated a weighted-least squares error between the reading and every predetermined HSV value and took the colour (possibly “line”) with the least error. The weightings on the error terms were determined through testing, colours that were detected erroneously too often had their weightings increased so that the algorithm had to be more confident about the colour before predicting it. We also used two HSV values for “line” because we found that the camera reading for the lines changed greatly across the track, producing incorrect predictions.

Next, with an associated colour for each reading, we perform Bayesian Localization. Since we are given a topological map, we must convert our readings from a continuous string of measurements (continuous in the sense that we get more than one reading for each hallway segment and colour patch) to a discrete reading (only one measurement returned per section). In our implementation, our algorithm only returns a measurement to the Bayesian Localizer and performs a state update if: 1) there are more than 10 continuous measurements of the same colour in a row (tuned from experimentation and the colour must not be “line”) and 2) since all colour patches are separated by hallways/lines, the new measurement cannot be the same as the previous

(to prevent more than one measurement in the same section). Now, starting with a uniform state probability distribution initially, we can perform the a priori step and a posteriori steps following the algorithms presented in lecture with the state and measurement models provided in the project handout (with a slight tuning based on testing). Since our robot is only ever moving forward on the track, the action u_k is always just "1", corresponding to moving to the next node in the state model and topological map. With the updated state probability distribution, we take the state with the highest probability as our new estimated state.

Finally, with our estimated state and colour readings, we determine a control to send to the robot. In general, our robot always moves straight forward with a speed of 0.06 m/s (this figure was modified several times throughout the project, but generally we move slowly to obtain more camera measurements) and 0 rad/s angular speed, which of course changes based on the state. If our algorithm believes the camera is seeing a "line", then it parses the "line_idx" variable to find position and sends it to the PID controller to correct any deviation from the setpoint (we chose a desired setpoint of 320). Our PID control is calculated following the standard algorithm described in lecture with gains tuned from testing. For the derivative term, we approximate $\frac{de}{dt}$ by storing the last error and calculating the difference. For the integral term, we sum the last 300 measured errors (to prevent integral windup). We added an additional limit of 500 to the integral error because even when recording just the previous 300 measurements, there was still significant error accumulation impacting the line following ability in undesired ways (We believe there are more efficient ways to address integral windup, but this was functional so we did not change it). Next, in the case where our algorithm believes it is on a colour patch but not the desired office, we publish the default velocities (no PID corrections) to continue moving forward past the coloured patch because there is no line to follow. Finally, when the robot believes it has reached a desired office, it will "deliver the mail" by setting all linear velocity to 0, turn left at a rate of $\frac{\pi}{12}$ rad/s until it has turned 90°, pause for one second, and then turn right 90° at a rate of $-\frac{\pi}{12}$ rad/s before resetting to the velocities before the delivery and continue moving to the next desired office. We track the number of degrees the Turtlebot turns by tracking the time since it has started turning (using the Python time library) and multiplying by the angular velocity.

Section 5. Summary of Demonstration Performance

In our first attempt at the demonstration, our robot performed very poorly, stopping at an incorrect office and finding 0/3 of the desired goals. Despite having tested our solution the night previously to verify functionality, we had trouble tuning colour parameters the morning of the demonstration. Due to lighting changes in the morning versus night, we had to retune parameters such as the HSV values associated with each colour on the track and error weightings for estimating colour. In our first run, when the robot was entering or leaving blue and yellow colour patches, it often believed it saw significant amounts of yellow and orange respectively. This resulted in incorrect state updates as the robot moved around the track trying to find the desired office. For the erroneous yellow readings in particular, since there is only one place on the map where yellow follows blue, yellow readings coming off of blue patches not only produced incorrect state updates but caused the probability distribution to rapidly converge to the wrong state. To make matters

worse, these incorrect updates happened at the start of our run, producing significant changes to our probability distribution that were difficult to correct as the robot continued moving around the track. As a result of all this, the robot's state probability distribution was completely incorrect, resulting in it stopping at the incorrect first office. Our computer also crashed in the middle of our demonstration, so our robot stopped line following before the probability distribution could converge and potentially correct itself to find the two remaining offices.

After this first attempt, we implemented a few changes to our solution to resolve these errors. The first thing we did was recalibrate the HSV values for all our colours to ensure they matched the readings from the camera. Next, we re-tuned our error weightings for estimating the measured colour to try and produce more distinction in our readings. Primarily, we made it more difficult for the robot to predict yellow and orange in an attempt to stop the phantom readings being made when the robot entered and exited blue and yellow patches respectively. Finally, we added a "grace period" where we force the robot to read measurements and update its estimated state five times before allowing it to stop at the desired offices. This was intended to ensure that the probability distribution of our robot converged before it decided to stop, reducing the potential for incorrect deliveries and the impact of incorrect state updates at the beginning of the run.

In our second attempt, with the changes described above, our robot performed much better. It no longer suffered from incorrect state updates when entering and exiting blue and yellow patches and the state probability distribution converged before the robot decided to stop. This allowed our robot to deliver to all three correct offices. However, we did have to make three minor corrections during the run as the robot sometimes deviated too significantly from the track. Although we tested our path-following controls before our second demonstration to ensure there were no issues, the turning process for deliveries or maybe lighting changes produced inconsistency that threw our controller off.

Section 6. List of Potential Future Improvements

Despite having completed the required course successfully, several improvements could still be made to the algorithms used.

First, more research should be done into the use of HSV for colour detection. Specifically, for our method of colour matching. In the case of RGB, treating each colour as a vector and taking Euclidean distance for a weighted least squares error is a rather intuitive approach — simply match the existing reading to the closest colour. However, HSV is not quite the same. The HSV tuple, like RGB, gives three values: hue, saturation, and value (Cotnoir, n.d). Hue represents the colour itself, like that of a colour wheel. Saturation represents the purity of a colour — higher saturation appears to be more vibrant, and lower appears more dull. Value represents how light/dark the colour is. With this knowledge in mind, calculating errors does not appear to be as intuitive anymore. Since we are interested in matching colour, it feels as if hue should be weighted more than the others. More research and testing is needed to make an accurate assessment.

Another persisting issue that we faced was that although the Turtlebot was able to reliably follow the line as long as the line was within view, upon encountering an office — a coloured patch — the Turtlebot could no longer detect a line, so the line error would behave erratically. What we and most groups did was simply disable line following once the robot believed it was no longer on a line. Rather, move in a straight line without adjusting course. However, this presents an issue; if the

Turtlebot approaches the offices at an angle, it will stay at that angle and drive off course in the office, failing to catch the line at the exit of the office. One possible solution is to use the gyroscope to match the orientation of the Turtlebot to a predetermined expected orientation when the robot cannot see a line. Like colours, this requires calibration ahead of time.

Another method used by many groups for better line following and colour detection was to mount a light, generally just a phone flashlight. The idea was to provide a consistent amount of lighting so that the colours read by the camera would be more consistent, allowing for better control. We experimented with this method but found that there were no significant benefits to our control.

Section 7. Conclusion

This culminating project presents a comprehensive integration of the diverse concepts and skills acquired throughout the preceding labs. By leveraging knowledge in robotic sensing, actuation, control, and localization, the project challenges our understanding and application of these principles in a practical scenario. The utilization of the Turtlebot as a mail delivery agent on a closed loop circuit, with coloured blocks symbolizing offices, adds a layer of complexity. Through this undertaking, we not only consolidate our theoretical knowledge but also gain valuable hands-on experience in addressing real-world challenges in robotics.

Appendices.

References

ROB301, "LABORATORY I Meet Your TurtleBot 3 Waffle Pi", 2023.

Cotnoir, L. (n.d.). Hue, Value, Saturation. Artspeak. https://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/.

See Code Attached on Following Pages