CSCE 439/839 LAB 2

The ROS Researchers

Team members: Rachael Wagner, Bethany Krull, David Ryan

Introduction

In this lab, we integrated two types of external sensing to the Balboa robot. First, the reflectance sensors on the base of the robot were enabled, allowing us to determine the light/darkness of the area under the robot. Next, we mounted an IR range sensor onto the robot. This allowed the distance to various obstacles to be determined. For both sensors, we incorporated the raw sensor values into the ROS system for controlling the robot. This allowed us to implement the mapping algorithm and reactive/object following controller discussed in this report.

Part of utilizing both the reflectance and IR sensor was fully characterizing the sensors and understanding their meaning and limitations. Both the sensors have distance boundaries where the sensor feedback becomes unreliable and essentially meaningless. So, before any development was done on our end we took care to push the limits of each sensor to understand how to properly operate them within their optimal operating range.

Accessing the sensor data within our various algorithms throughout this lab has been rather simple after setting up our system to read in and interpret the raw data. We made modifications to the **balboa_core** to include both the reflectance and IR sensors in the message that runs from the robot to our nodes (**balboaLL.msg**). Taking advantage of the existing architecture made this modification both easy to implement and straightforward to use later on in our code.

Reflectance Sensor Mapping

For this part of the lab, we incorporated line sensors used to measure reflectance into the Balboa robot code. These sensors were previously installed on the device, but uninitialized in the firmware. In order to do this, we had to update the Arduino code that is loaded on the Balboa robot. We used the Line Sensor class (https://pololu.github.io/balboa-32u4-arduino-library/), part of the Balboa32U4 library to do this. Some of the code needed was the **lineSensors.setCenterAligned()** command to initialize the library, and **lineSensors.read** command to read the sensor values. Next, we modified the serial data packet that the robot sends to ROS during use, also part of the Arduino code, in the **sendDataToROS()** function. We included the line sensor values as part of the data packet. Finally, we updated the ROS code that receives the data packet from the robot to include the new sensor values. This involved updating the **balboaLL.msg** message type, and the **balboa_serial.py** node.

We spent a bit of time characterizing the sensor to determine what sensor reflectance values corresponded to the grayscale. We created the following table showing the approximate range of values that result in various shades. Note that we rarely saw values less than 200 being returned, even on bright white substrates.

Reflectance Sensor Value Range	**Shade**
200 - 500	White
500 - 900	Light Gray
900 - 1300	Gray
1300 - 1600	Dark Gray
>1600	Black

We also were able to determine that the robot performed best on identifying a line at least greater than the width of the middle three sensors (about 1 inch). If the line was smaller than that, inconsistent shade readings would result from position oscillations caused by robot attempting to stay balanced.

The next step after integrating the reflectance sensors into the system was to create a mapping algorithm to detect markings on the substrate the robot is driving over. In order to do this, we created two ROS nodes. The first is a PID node that takes in a target value and commands the robot to reach that location. We changed the PID node from Lab 1 to not take in arrow key commands, but encoder clicks. The second is a mapping node that instructs the robot to drive in a grid pattern, storing reflectance data as it goes. It commands the robot to drive in a straight line, with 10 evenly-spaced data collection points (the mapping resolution) before completing two 90 degree turns to turn around. We complete four rows while collecting reflectance data and storing them in a struct variable. Finally, the mapping node prints out an ASCII chart depicting the reflectance data the robot collected of the floor markings.

The PID node performs fairly well, it takes in encoder targets and adds them to its current position to create a new target position. This is done at different unit scales for both its angular and linear position. Because we do not ensure that the robot has achieved a low error with each new target in the mapping node, the entire mapping system has an open loop structure. This did initially result in us sending new target positions to the PID loop too quickly for the robot to respond, so we slowed down the loop rate of the mapping node to allow the robot to hit each new setpoint. As for the mapping node, it proved to be more difficult to develop and perform less consistently. Sensor drift was a notable problem, as our new commanded positions would not result in a meaningful target position change for the robot when the sensor value magnitude had drifted larger and larger. Driving in a consistent grid pattern was also difficult, as the robot would sometimes not complete perfect 90 degree turns. Finally, storing and printing the ASCII characters that represent reflectance in a custom struct type also proved to be somewhat problematic in the timeframe provided.

IR Range Finder

An additional task required by the lab was to add an IR range finding sensor to the Balboa robot. Once again, we had to update the Arduino code loaded onto the Balboa microcontroller. This involved reading the sensor value (**analogRead**) using an analog input pin. Again, we modified the serial data packet that the robot sends to ROS, to include the IR range value. Finally, we updated the ROS code that receives the data packet to include the IR range values. This involved updating the **balboaLL.msg** message type, and the **balboa_serial.py** node. We then verified that the sensor was working; using the Serial Plotter from the Arduino IDE, we could observe that the IR sensor data would update corresponding to the distance of a notebook placed in front of the robot.

We decided to install the IR range finder on one of the bumpers of the Balboa Robot. We ziptied it to the center of the front of the robot (facing the positive Z axis), so it could sense obstacles in the positive Z direction. The final configuration is shown below.

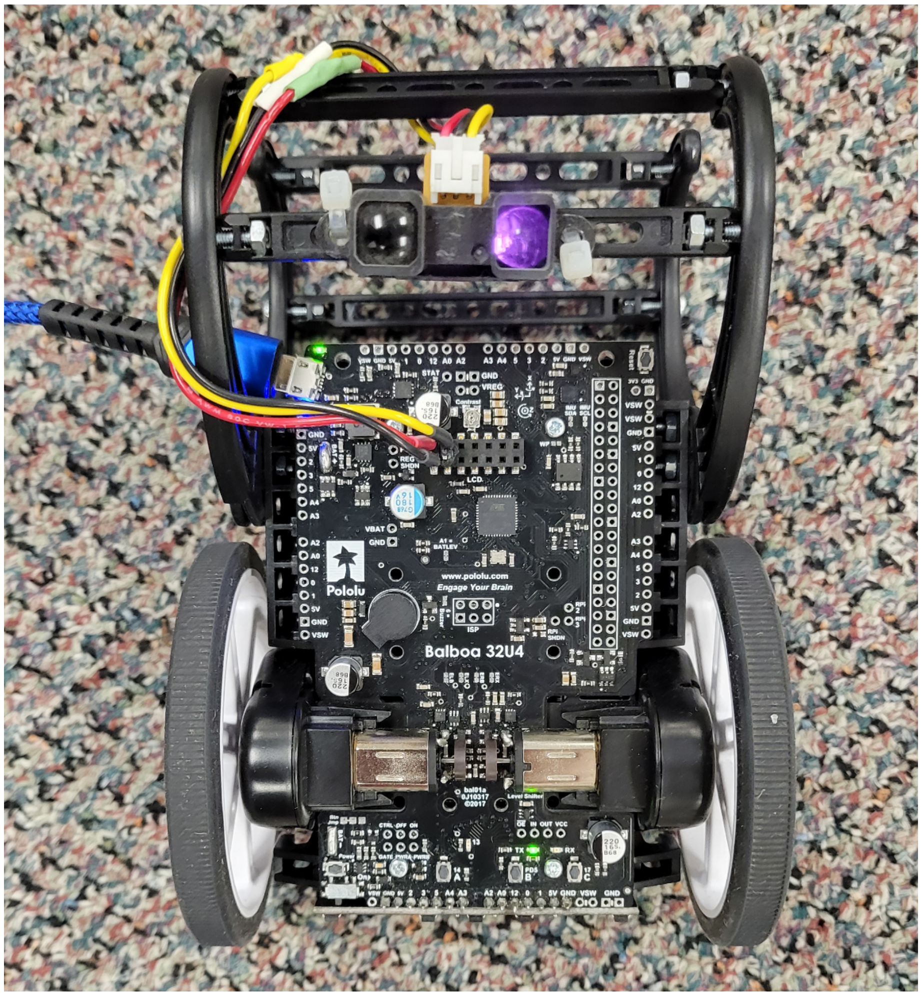
Figure 1. The final IR sensor mounting location on the Balboa robot

After installing the sensor on the robot we focused on characterizing the voltage to range conversion. We were able to print out the analog sensor values using the Serial Monitor of the Arduino IDE while changing the range of an object in front of the sensor. We recorded sensor values at 1 - 6 foot distances, incrementing in units of 1 foot (see plot below).

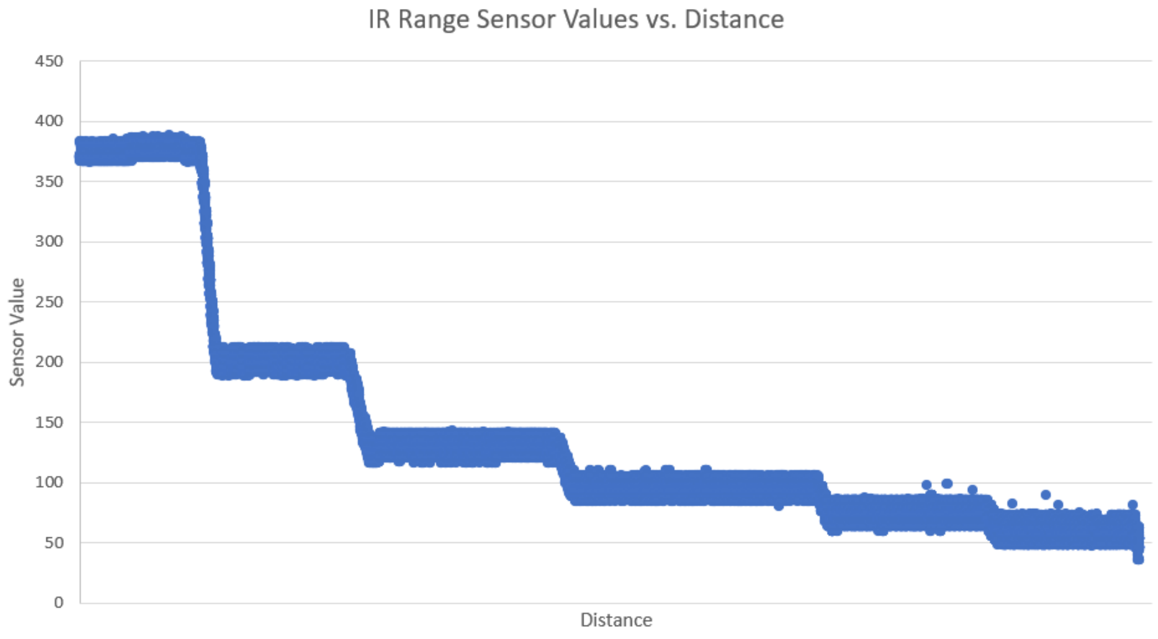

Figure 2. IR Sensor values for ranges between 1 and 6 feet

Using the raw sensor values, a 10-bit variable between 0-5V, we can determine the approximate voltage for each distance as shown in the table below. The results are comparable to what is presented in the sensor's datasheet, with a peak at relatively close ranges that decreases asymptotically to the baseline sensor value.

Sensor Range	Approximate Sensor Value	Voltage
1 ft	375	1.83 V
2 ft	200	0.98 V
3 ft	135	0.66 V
4 ft	100	0.49 V
5ft	75	0.37 V
6ft	65	0.32 V

In order to characterize the sensor distance, we experimented by observing the sensor results for objects of various sizes at various distances. On a short distance scale (e.g. a foot or less), small or skinny objects, like an eraser for example, can be sensed with notable error. At longer distances (e.g. greater than 3 feet), medium sized objects like notecards result in regular range values. However, larger objects, such as notebooks, consistently provide accurate sensor readings at all ranges. Therefore we believe that the beam width must fall around <2in in size at close ranges, and grow larger to around >4-6in at further distances.

Reactive Control

This portion of the assignment was fairly simple (relative to the rest of the assignment that is). The first step that we took was making a copy of our or Lab 1 PID node and edited it lightly. We changed what topic the node was listening to from a **Twist** vector to a regular **Int32**. This allowed us to publish the raw sensor values from the IR distance sensor. We set a permanent target IR value (the distance we wanted the robot to maintain from an obstacle) and used continuously updating IR values to calculate the error. This error was then used in a normal PID equation. We had to slightly retune our PID controller to work smoothly with the new algorithm and the added weight of the IR sensor (it tended to pull one side of the robot down and make it hard to balance).

Our reactive control is very accurate and the PID controller eliminates a lot of error or jerkiness when trying to maintain the target IR reading. The robot is especially good at moving backwards, especially compared to how it moves forwards. Since the IR sensor is attached to the front side of the robot, its vision changes when the robot tilts forward, to try to move quickly to reach a far away goal. This causes the IR reading to be less accurate and can lead to failure if the IR dips low enough that it reads the distance to the floor. These misreads cause the robot to think it is suddenly way too close to the obstacle and jerk backwards. We mitigated this issue in two ways. We move the obstacle slower when guiding the robot, to prevent the robot from tipping too much. Additionally, we modified our PID values to keep the robot from reacting so strongly to target distances that are far away. This keeps the robot from speeding to the target and causing tilting that would affect the balance state.

Note again that we plotted the various distances that fall within the readable and reliable range of the IR sensor in the above IR Range Finder section. We did this by setting the robot in a fixed position and moving an obstacle in front of the IR sensor at fixed distances. We noted that the distance the readings became unusable were consistent with the datasheet.

Conclusions

The code for this lab is in the zip file included with the lab submission. In response to feedback for the previous lab, we were sure to comment our code more clearly. For this lab, we developed in both cpp and Python, whereas for the previous lab we had mainly used Python. This allowed us to feel comfortable in developing using both languages in ROS. We found that using cpp allowed us to have more control over our data types which allowed us to do more complicated computations with more confidence.

Originally, we had planned to create a specific reflectance sensor message type that published the data from the IR sensors. We eventually decided to reduce complexity and build this functionality into the mapping node. Over the course of editing the CMake file to accomplish this, somehow the balboaLL message type became dependent on including the custom messages in our package folder structure, otherwise it would not compile. Although the message type is redundant, we still included the msg folder in our custom package as we saw this as a minor bug not affecting functionality and focused our efforts elsewhere.

The system rqt_graph for the Reflectance Sensor Mapping portion of the lab is shown below. It can be observed that the balboaLL robot data is used both for the PID node **pidMap** and the mapping algorithm node **mapDrive**. The mapping algorithm sends new target locations to the PID node to trace a grid over the desired area. Then, the reflectance sensor data is read and printed corresponding to each grid location.

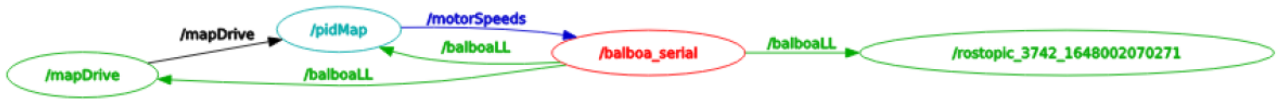

Figure 3. Reflectance Sensor Mapping system diagram

The system rqt_graph for the IR Reactive Control portion of the lab is shown below. Note that the logic used to run the PID loop is entirely in the **/ir_reactive_control** node. Then, the PID loop publishes new motor speeds that update the balboa node.

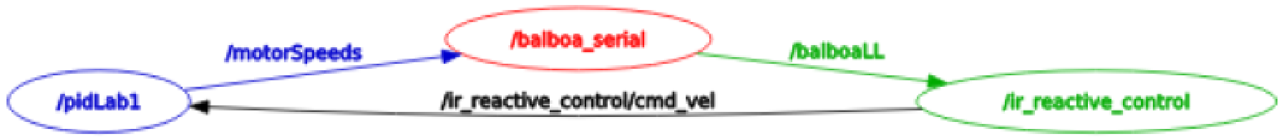

Figure 4. IR Reactive Control system diagram

A common difficulty that we had throughout this assignment was message and variable types. It has been a challenge wrapping our head around how messages were not basic types and so therefore could not be operated on like normal values. Using the '**.data**' attribute we can access the actual value of the message. However a custom message may have different types inside that will operate as primitive types.

We also struggled with getting the robot to drive in a controlled grid, to build up the functionality of the mapping path. This was a very frustrating portion of the assignment because on the surface it would seem that this is a simple problem using the previously built PID node. We initially tried to create a closed loop control in the mapping node, but it was not responsive enough. Over time, the sensor drift was enough to prevent the robot from hitting the new setpoints created in the mapping node. Therefore we sent open loop commands to the PID node in order to achieve the desired grid results. Again, because we struggled with developing the mapping grid, we had trouble implementing the printing portion of the algorithm in the timeframe provided.

No section of the lab handout was unclear. However, there are several compound questions (questions that are asking for multiple things at once) that could potentially be clarified by stating each point in its own statement.

As a team we divided the work completely evenly. Each of us spent time coding and filling out the report. Mostly we would all work together on one task at a time with one person typing while everyone else helps brainstorm and refine the logic. Occasionally we would all be working on different things at once but this was mainly to add to the report than to work on coding tasks.

Sections:	**Hours:**	**Members:**
Checkoff 1	20	All
Checkoff 2	15	All
Report Finalization	5	All
Total:	40	All