**CSCE 439/839: Robotics: Algorithms and Applications, Spring 2020**
**Lab 1**

Started: Monday, Feb 3
Lab 1 Checkpoint 1 (Up through Sec. 2.5): Monday, Feb 10
Lab 1 Checkpoint 2 (Up through Sec. 3.1): Monday, Feb 17
Lab 1 Due Date: Monday, Feb 24

# 1 Overview

In this lab you will integrate your robot with ROS, develop a PID controller, and perform experiments to characterize your robot. This lab has checkpoints. In lab on the date of the checkpoint you will be responsible for showing the instructor your progress. You do not have to have the section written up yet (although that might not be a bad idea), but you need to be able to demonstrate all the tasks in the section.

Before starting you should read through the whole lab. Some parts can be done in parallel, while some sections rely on the completion of previous sections. You should discuss your plan of attack for the lab in your group and decide how you will work together and divide the work. Everyone, however, is responsible for knowing about all sections of the lab. In addition to completing the lab report, on the due date, you will demonstrate what you accomplished to the instructor.

Also, the points for this lab do not add up to 100, but each lab is worth the same amount as any other lab even if the points are not equal.

# 2 ROS and Interfacing with Balboa (30pts)

In this part, we will explore the ROS interface to the Balboa and write code to control the robot using your computer keyboard (like you can control the turtle).

## 2.1 Version Control

It is highly recommended that you use a version control repository for your code (e.g. git, subversion, mercurial, etc.). While this is the recommended option, it is not required for this lab. You can also use a usb stick, space on CSE or UNL shared drives, or other services like Box or Dropbox to backup your code. If your drive crashes it is **not** an excuse to turn in the lab or assignment late. Nor does the "my partner has the code" excuse work. No extensions for this will be granted.

## 2.2 Basics

Create a workspace for your code and a lab1 node. You should then be able change directories to the sample code directory by typing in the terminal:

```
roscd lab1
```

or alternatively you can manually `cd` to the directory. The command `roscd` is useful, however, because you can use it to go to the source directory for any ROS module. Explore ros a little bit and find out where `roscpp` is.

***Question:*** What is the absolute path of the `roscpp` module? What does `roscpp` do?

You should download the `balboa_core` code from the course website. This package contains the ROS code and Arduino code to allow you to control your Balboa robot from ROS. Put this in your workspace (remember to `catkin_make`), program your robot with the Arduino code, and then explore the operation of the system.

***Question:*** What are the names of the nodes, the topics, message types, etc that allow you to control the robot? How can you start the node(s) needed to control the robot?

Once you get this configured properly you should be able to start receiving messages from the robot after starting this node. Remember to be careful when turning on your robot that it is on a safe surface and won't run off the side of your desk if it starts up unexpectedly.

### 2.3 rostopic and rosmsg

The command `rostopic` lets you examine messages that have been advertised and are being published. Use the command `rostopic list` to show all active message topics. Use `rostopic echo` to display the output the message published from the robot.

***Question:*** What command did you use to do this? And what information do you get from the output? How fast is this published and how did you determine this?

The command `rostopic` can also be used to publish messages. Use the command `rostopic pub --help` to determine how to publish messages, you can also refer to the ROS online tutorials.

***Question:*** Publish a command to make the robot drive forward (slowly at first!). Describe how you figured out how to do this, you can use the command `rosmsg` to determine parameters for messages.

Make sure you are ready to pickup the robot and turn it over to turn off the motors if it goes out of control.

### 2.4 rqt_plot

The command `rqt_plot` is a command you can use to quickly visualize and plot values in published message topics. You can plot the tilt data, for instance, by doing:

```
rqt_plot /balboaLL/angleY
```

If you separate fields by commas they will be plotted on one graph, if you use spaces instead, they will be plotted in different graphs.

***Question:*** What are the min and max values for the tilt? Explore the other angles and describe which axis each value represents (a picture of the robot helps).

***Question:*** Gyros are prone to drift over time. Characterize the drift of the gyroscope on all 3 axes.

### 2.5 Keyboard Control

*Note that up to and including this section must be completed for Checkpoint 1.*

Write a new node that allows you to drive your robot around using the `turtle_teleop_key` node.

***Question:*** Describe how you got this to work and characterize the driving performance.

***Question:*** When you release the keys the robot should stop. Make sure you describe how you did this.

***Question:*** Make sure to include an `rqt_plot` showing how everything connects together.

## 3 PID (30 pts.)

In the previous section you implemented control using the keyboard. You had no way to specify a particular angle or distance to go to (you had to manually control it). By angle, I mean the rotational angle around the long axis of the robot when it is standing up. You can control the angle by giving differential wheel speed commands (e.g. -5 and +5). In this section of the lab, you will implement a Proportional-Integral-Derivative (PID) controller to control the angle and travel distance. Below I describe a general PID controller, you can also read more about PID control on the wikipedia page: `http://en.wikipedia.org/wiki/PID_controller`. We also covered this topic in class.

You should create a new ROS node for PID control of rotation angle. You could call it `angularPID` or something similar. Similarly, you should create a `distancePID` that controls how far the robot travels

(forwards or backwards). You should make these nodes such that you can insert it between your keyboard control node and the node that sends commands to the robot. That way you will be able to easily configure your system to run either the keyboard doing direct control or with the PID just by modifying the launch file. Look into the `remap` keyword for the launch file in order to do this quickly and easily.

***Question:*** Describe what `remap` does and how you used it.

**For groups with majority of 839 students only (this paragraph):** you should also implement a PID controller to control the angular velocity of the robot. That way you will be able to tell it to spin at, for instance, 30° per second. This should sit between the angular position controller and the `balboa_serial` node. Start by implementing this node that controls the velocity before implementing the position PID. You should perform experiments and answer questions for this PID controller as well.

Here is a brief summary of PID control as we discussed in class. You should make sure to read this whole section before starting as some of the work for the later questions will be helpful in answering the earlier ones. If you have a target angle $a_t$ and a current angle (as read by your IMU) of $a_i$, then you can control the rotation, $r$, of your robot according to:

$$r = P(a_t - a_i) \tag{1}$$

$P$ is the proportional scaling term of the controller (and can be positive or negative). With a $P$ controller, if you are far off from your target angle, it will have high rotational rates and when it gets closer the rate will decrease. This can work well, but if $P$ is too low it will never reach the target and if it is too high, it will overshoot and oscillate.

By introducing a derivative, $D$, term oscillations can be damped:

$$r = P(a_t - a_i) + D(e_i - e_{i-1}) \tag{2}$$

where $e_i$ is the error from this time step, $a_t - a_i$, and $e_{i-1}$ is the angle error, $a_{t-1} - a_{i-1}$, from the previous time step and $D$ is the derivative scaling term of the controller. The value $(e_i - e_{i-1})$ is basically a numeric derivative of the error and it tends to slow how fast the output term $r$ will change (damps oscillations).

A purely PD controller often works well and I recommend that you first try a PD controller before trying to introduce an integral, $I$, term. The problem is that a PD controller will not always achieve the set point [1]. Adding an integral term can ensure that you reach your target, but at the cost of potentially adding oscillations and having to deal with issues such as windup. Adding the $I$ component will yield:

$$r = P(a_t - a_i) + I(integral_i) + D(e_i - e_{i-1}) \tag{3}$$

where, $integral_i = integral_{i-1} + (a_t - a_i)$, and $integral_0 = 0$. The windup problem is that if there is an offset (e.g. you hold your robot in one position and don't let it rotate for a while), then the integral term will continue to grow unbounded. So typically, you need to bound the integral term by a maximum value to mitigate this problem.

In essence, a PID controller is not too difficult, however, there are a number of practical issues that you must address when using one. The first, is tuning the PID values (see Section 3.1 on setting parameters, which you may want to complete before this section). Wikipedia has a number of suggestions as to how to tune the controller. My preferred method is to first try a PD controller (almost always works better than a purely P controller). The idea is that you first set $D = I = 0$ and then tune P until it oscillates slightly, then add $D$ until the oscillations are damped. On your robot, the I term probably isn't needed, but if you find that there is an offset from the final position, then you can add a little bit of $I$ to make sure it achieves the target.

For these questions, make sure to answer for both the angle control and also the position control (839: answer for the velocity controller as well).

***Question:*** Did you need to use an $I$ term? What were your final values for the PID parameters? What are the units of the parameters?

[1] Think of a PD controller as pulling a car up a hill with a spring, no matter how stiff the spring, if you put one end where you want the car to end up, it will never get there.

There are a number of other practical issues when controlling rotational systems by degrees. For instance, when doing more than a single rotation, there will be a discontinuity in the angle measurement.

***Question:*** How did you deal with the discontinuity between the angle measurements if you do multiple full rotations?

Another issue with a PID controller is timing. The gain parameters will depend on the frequency at which you update the commands. In your PID controller, you will need to subscribe to and receive both the angle target message as well as the current gyro position message. You should base your main PID control loop on the rate at which you receive the current gyro position. The reason to do this is because the rate that you receive this message is relatively constant, whereas you will likely receive angle target messages at an irregular rate.

***Question:*** Describe how you reconfigured your keyboard control node (or even better would be to put a new node after the keyboard control node so you do not have to modify the keyboard node) so that it can be driven "naturally," while using the PID systems. By natural, I mean when you release keys it should stop fairly quickly. Hint: this requires reading the current angle/position and updating the targets passed to the PI node when it is released.

***Question:*** Show sending a target distance of one foot, followed by -1 foot (in other words, go forward then backwards) and demonstrate the robot going that distance. Have some measurement device (ruler, tapemeasure, etc.) in the background. **Include a short video of your robot doing this that also shows the commands you are sending (e.g. video your computer screen and robot).**

***Question:*** Show sending a target angles of 90 degrees, followed by 180 degrees, followed by 360 degrees. **Include a short video of your robot doing this that also shows the commands you are sending (e.g. video your computer screen and robot).**

***Question:*** Your PID nodes (distance and rotation) should be able to take target commands and go there. Demonstrate, evaluate, and discuss how well this works when telling the robot to draw "CS" or some other favorite letters if you are getting tired of drawing CS with your robot. How does this compare to how it worked before you used PID nodes? **Include a short video of your robot doing this that also shows the commands you are sending (e.g. video your computer screen and robot).**

***Question:*** **For majority 839 groups only:** Modify the keyboard control code so that you can also press some other key to have the robot drive a set distance or rotate a particular angle.

***Question:*** **For majority 839 groups only:** Evaluate how well your robot works when operating on a hill versus on a flat surface. You can use cardboard or something similar to create a hill.

## 3.1 ROS Parameters

When you are developing your PID controller, it is convenient to be able to set the PID values from the launch file so that you do not need to recompile your code each time you want to change the parameter[2]. Look at the website `http://www.ros.org/wiki/roscpp/Overview/Parameter%20Server` to see how to read/set parameters.

In general, to read a parameters in C++ you should do:

```
P = 1.0;
nh.param("rCtrl/P",P,P);
I = 0.0;
nh.param("rCtrl/I",I,I);
D = 0.0;
nh.param("rCtrl/D",D,D);
```

This will set default values (in case none are specified in the launch file), but use the value from the parameter server if any are present. The launch file should then have:

[2]In addition to being able to set PID values in the launch file, you may want to create a message that allows you to adjust the PID values without having to relaunch the node. But even if you do this, you should also be able to set default parameters in the launch file.

```
<param name="rCtrl/P" type="double" value="1.0"/>
<param name="rCtrl/I" type="double" value="0.0"/>
<param name="rCtrl/D" type="double" value="2.0"/>
```

to set the parameters.

***Question:*** What command-line command can you use to get the values of these parameters? What about setting them? What happens if you set them on the command line, do the values the program is using change? What about if you change them using the command line and then relaunch your launch file? If you relaunch your node using `rosrun`? What does this tell you about the order/priority of the various ways you can set parameters?

# 4 To Hand In

You should designate one person from your group as the point person for this lab (each person needs to do this at least once over the semester). This person is responsible for organizing and handing in the report, but everyone must contribute to writing the text. You should list all group members and indicate who was the point person on this lab. Your lab should be submitted by handin, `http://cse.unl.edu/~cse439/handin/`, before the start of class on the due date. Include a pdf of your report and your source code in handin.

Your lab report should have an introduction and conclusion and address the various questions (highlighted as ***Question:*** ) throughout the lab in detail. It should be well written and have a logical flow. Including pictures, charts, and graphs may be useful in explaining the results. There is no set page limit, but you should make sure to answer questions in detail and explain how you arrived at your decisions. You are also welcome to add additional insights and material to the lab beyond answering the required questions. The clarity, organization, grammar, and completeness of the report is worth **15 points** of your lab report grade.

In addition to your lab report, you will demonstrate your system and what you accomplished up to this point to the instructor at the beginning of lab on the due date. This is worth **15 points** of your overall lab grade. You do not need to prepare a formal presentation, however, you should plan to discuss and demonstrate what you learned and accomplished in all sections of the lab. This presentation should take around 5 minutes.

***Question:*** Please include your code with the lab report. Note that you will receive deductions if your code is not reasonably well commented. You should comment the code as you write it, do not leave writing comments until the end.

***Question:*** Include an `rqt_plot` of your final system and comment on your overall system architecture.

***Question:*** For everyone in your group how many hours did each person spend on this part and the lab in total? Did you divide the work, if so how? Work on everything together? Give details on the hours each group member worked (e.g. keep a list of how many hours per day each person worked and on what).

***Question:*** Please discuss and highlight any areas of this lab that you found unclear or difficult.