CSCE 496/896: Robotics
Lab 2: PID, Range Finders, and Tangent Bug

Instructor: Carrick Detweiler
carrick _at_ cse.unl.edu
University of Nebraska-Lincoln
Spring 2011

Started: September 16, 2011
Lab 2 Due Date: October 7, 2011

1 Overview

In this lab you will implement a PID controller to control the angle of your hovercraft. You will also attach range sensors to your hovercraft, characterize them, and then implement a variety of algorithms that utilize the range finders. Make sure to read through the entire lab before starting to implement these controllers. You should also map out the final architecture of your ROS nodes with your group before you begin implementing these.

Remember and review the safety instructions from Lab 1.

2 PID (30 pts.)

In the previous lab you implemented control of the hovercraft using the remote control. In order to rotate, you just mapped the joystick rotation command into a raw rotational thruster command. You had no way to specify a particular angle to go to (you had to manually control it). In addition, when you were translating, your hovercraft may have rotated if the translational thrusters also had a rotational component of force.

In this section of the lab lab, you will implement a Proportional-Integral-Derivative (PID) controller to control *angular position*. 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 will also cover this topic in class.

I suggest you implement a new ROS node for PID control of position. You could call it `angularPositionPID` or something similar. This should send messages to your `thrusterMapping` node (or whatever equivalent node you have). As an input, it should probably take the same message type as your `thrusterMapping` node, but instead of having the angular value mean "thrust" it will mean "rotational position." The advantage of this structure and using the same message type (note they will all have different message/topic names) is that you can easily reconfigure your joystick node or the algorithms you implement to control using angular position or raw thrust.

Here is a brief summary of PID control. 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 gyro) of a_i, then you can control the rotational thrust, r, of your hovercraft 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 thrust and when it gets closer the thrust 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:

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

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 $(e_{i-1} - e_i)$ 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:

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

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 hovercraft 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 2.1 on setting parameters). 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 the hovercraft, 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.

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

There are a number of other practical issues when controlling rotational systems by degrees. Fortunately, the `Hovercraft` ROS node already accumulates the angle continuously (it does not reset to zero when passing 360). This eliminates numerous problems you could with your controller if it had to deal with this discontinuity. However, another problem you need to address is deadbands. If you leave your PID controller on all the time, then the hovercraft will maintain its angle. However, very small offsets may cause current to go to the thrusters without them actually moving. This is a waste of energy. Thus, it is good if you have deadbands so that the thrusters will never just be on a very small amount (and not turning). Or jumping on for very tiny angular errors that may be very difficult to correct for.

Question: Did you end up needing deadbands? If so, what values did you use?

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. Also, by using this approach, the hovercraft will also maintain the target angle when it is sitting still or while it is translating.

Question: If you also use your rotational thrusters for translation there will be a conflict between the PID controller and your translational commands. It is possible to account and correct for this, how did you do so?

Question: A final issue you need to address is making sure that the PID controller never exceeds the maximum rotational rate the gyro can handle. How did you address this problem?

Your hovercraft joystick node should use the PID control node. To do this, you should reconfigure your joystick node to send messages to the PID node. However, since the PID node takes a target angle as input, you will need to change your joystick node so that it outputs a target angle, instead of an angular rate. There are a couple of ways you can do this. The first approach is to integrate the yaw control input in your hovercraft joystick node to produce an absolute target angle. The other approach is to create a new

[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.

node that takes the output of the hovercraft joystick node and integrates the angular rate command into an absolute angle command before passing the target angle onto the PID controller.

Regardless of how you structure the system, you should integrate the joystick yaw commands at a fixed rate (and not just everytime you get a joy message as these are irregular). To do this, you will need to use a ROS timer. You can find details at `http://www.ros.org/wiki/roscpp/Overview/Timers`. Here are the basic commands you need to use to create a timer:

```
//Define the timer as a global member of your class
ros::Timer timer;
//Create the timer to run at 10Hz, assuming your node is named joyControl
timer = nt.createTimer(ros::Duration(0.1), &JoyControl::updateSetpoints, this );
//The callback function called at 10Hz
void JoyControl::updateSetpoints(const ros::TimerEvent& e){
   //Integrate your readings and publish messages
}
```

In essence, this will make your joystick control the speed that the hovercraft rotates. You should scale the frequency of the update and potentially add a scaling factor to the integration so that moving the stick all the way in one direction will correspond to the maximum rotational rate you can achieve.

Question: If you manually prevent the hovercraft from rotating, the integration will continue. When you let go, it will spin rapidly to "catch up," even if you have already let go of the joystick. This isn't very nice behavior. Correct this behavior so that it will always "respect" the joystick command (e.g. if the joystick is centered it should not be rotating and if it is half way it should be spinning at half speed even if you manually stop it and then release it). How did you correct this[2]?

In addition to having the joystick control the rotational rate, make one of the buttons on your joystick rotate the hovercraft by 90 degrees. This will be useful as a method to test the performance and stability of your PID controller.

Question: Plot outputs of your current angle, target, and rotation thrust levels for different PID values when changing the target angle. Do this both for changing the target angle with the joystick and also with the 90 degree rotations with the joystick button. Make sure to include an output from your final PID controller. Also include plots from a purely P controller when it has a value that causes oscillation and when it does not cause oscillation. Include any other plots that you find useful.

2.1 ROS Parameters

You have seen how parameters can be set in ROS launch files. 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[3]. 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);
```

[2]One approach is to do PID control on velocity instead of position, but for this lab, keep your PID controller in terms of angle, not velocity

[3]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.

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:

```
  <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?

2.2 Arbitrators and Triangles

Up until now, you have only controlled your hovercraft using the joystick controller. Now we will add the ability to control the hovercraft autonomously. Our first autonomous controller will be a dead-reckoning "triangle" driving controller. Before we do this, however, you will need to create a new ROS node that is an arbitrator for the commands being sent to your PID controller. This node should subscribe to two different message types. The first should be an "arbitrated control" message and the second should be a "state" message. The arbitrated control command should be a new message type defined as:

```
Header header
Control control
string stateName
```

where `Control` is the name of the message type your PID controller accepts and the `stateName` is a string that describes the state that this command corresponds to (and of course, it is always good to include a Header, although you can omit it if your Control message has a header).

The other message this node subscribes to is a state message, which should have a string that sets the current `stateName` that should be passed onto the PID controller.

Question: Describe the structure of your arbitrator and how it works and interacts with other nodes.

Now implement a new node that drives the hovercraft in a triangle. You can do this by using a series of 120 degree turns followed by a period of constant forward thrust. Make the arbitrator use this node when one of the buttons on your controller is pressed. Make this node really simple. Do not subscribe to the joy message. Instead, this node should always be sending out triangular trajectories (it may need to subscribe to the gyro angle to know what the target rotational angle should be). The arbitrator will handle deciding which node to pay attention to.

Question: Describe the operation and structure of your triangle trajectory node. Also comment on the performance and how well it actually drives a triangle. Include any useful plots (e.g. the gyro angle as it travels and the thruster outputs).

Question: It is easy now to use `rostopic echo` to see what state is set and also the input to the PID controller. However, you may also want to examine inputs to the arbitrator that are not currently selected. If you merely echo all inputs to the arbitrator it will be extremely confusing. However, `rostopic echo` allows you to filter commands. How can you filter to only see messages generated by your triangle node?

3 IR Rangers, Reactive Control, and Mapping (25 pts.)

In section you will augment your hovercraft with two IR range finders. You will characterize the sensors and then develop some basic reactive controllers and also a basic mapping system.

3.1 IR Range Finder

Each group has two IR range finders. These are made by Sharp and the model number is `GP2Y0A02YK0F`. It is a 5V sensor that outputs an analog voltage. It can sense ranges roughly between 15cm and 150cm. You can find a datasheet and more information on the sensor here: `http://www.sparkfun.com/products/8958`. The first thing you need to do is create and solder a connector to connect the sensor to your hoverboard. These should plug into ports `5V_A2D_0` and `5V_A2D_1` on the hoverboard. If you are uncertain how these should be connected, please ask the instructor. *Note: Your hoverboard will need a firmware update before you will be able to read these ports in ROS.*

You can quickly see if the sensor is operating properly by looking at the ADC reading messages that the `HoverboardLL` node produces. Verify that you are obtaining reasonable readings by echoing the ADC messages.

Next, mount the range finders to your hovercraft. Where you locate them is up to you (and you may decide to change it in the future). Make sure to read through the rest of the lab and decide on reasonable locations to accomplish the tasks in this lab. Note that if they are too close together they may interfere. Also, be aware that if multiple hovercrafts are operating in the same area you may get some interference from the other robots.

Question: Where did you end up mounting your IR range finders? Why did you put them in these locations? Include a picture of the final configuration.

The next step is to convert these raw ADC readings into ranges. You can do this by either creating a new ROS node that subscribes to the ADC message and publishes two different range messages or you can modify the `Hovercraft` node to publish new range messages. The datasheet has a graph indicating the conversion between range and voltage. However, this may not be completely accurate for your sensor and you will also need this information in your ROS node to convert between voltage and distance.

Question: Do an experiment to characterize the voltage-to-range conversion of your sensor. Report on your results and include a plot showing the response.

Question: Do some experiments to characterize the smallest objects that the IR range finder can find at various distances and report on the results.

Question: Experimentally determine the beam width of the IR sensor and report on how you determined this and the results.

3.2 Reactive Control

Your next task is to develop a reactive controller that tries to maintain a fixed distance from an obstacle in front of your hovercraft. To do this, create a new ROS node that subscribes to the range messages and publishes control commands to the arbitrator node. In your joystick node, make it so you can activate the reactive control by pressing one of the buttons (you may need to remove the LED button commands from the previous lab to have enough buttons to run all of the controls in this lab).

There are a number of approaches you can use to try to maintain a fixed distance from an object. You can use thresholds and fixed thrusts, a PID controller, or other methods. You should test your controller by moving something (e.g. a book) closer and further away from the hovercraft to see that it tracks it. Since your PID controller is still running, the hovercraft should maintain the same angle while it is doing this, although it may drift in translation.

Question: How did you implement distance control? How well does it work? Include a plot of the distances obtained from the range finders as well as directional control outputs as you move an object around in front of the hovercraft.

Since you have two range finders, you can also implement a behavior where the hovercraft will turn to keep an object in front of it in addition to maintaining a fixed distance. While not required for this lab, you may be inspired to implement this behavior. If you do, report on it as well.

4 Mapping and Tangent Bug (20 pts.)

4.1 Mapping

Your next task is to use the hovercraft and range finders to create a rough "map" of the environment. If your hovercraft tends to translate significantly while rotating, now would be a good time to calibrate it so that the translational thrusters mostly compensate for any rotation-induced translation.

As in the previous sections, create a new ROS node that publishes to your arbitrator and make a button on your joystick control the arbitrator. One difference from the previous nodes is that this node probably needs some knowledge about when it should start the mapping. You can do this by subscribing to the state topic, joy topic (not recommended), or having your joystick control node publish a message to this node.

In order to map, you should rotate slowly and collect and store range finder information. After you have completed the rotation you will have a set of ranges to everything around your robot. We will use these in Section 4.2 to implement some slightly more complex behaviors. For this section, you should implement the following behavior. After mapping your local environment, your robot should move towards the closest point it finds (or perhaps cluster of points if you find that there is noise in the sensor readings). Once it starts moving towards the closest point, you should switch to the reactive control mode you developed in Section 3.2. If it loses sight of the object it should perform the search again. With this behavior, your robot should be able to follow you around (slowly) as long as you are the closest object to it.

Question: Describe how you implemented this simple mapping function.

Question: What speed does your robot rotate at? Why did you choose this speed?

Question: How did you implement the changing of states? Hint: you may want to implement another node that manages the states, although this is not necessary.

4.2 Tangent Bug

Now that you are able to do basic mapping, you should implement a variant of the tangent bug algorithm[4]. In class, we discussed the `Bug1` and `Bug2` algorithms for navigating an unknown environment. These algorithms assumed that the bugs/robots could only detect an object if they bumped into it. Our hovercraft are far more advanced in that they have IR range finders that can measure distances to nearby objects. Tangent bug looks ahead for obstacles and if there is one in the path to the goal, it instead moves to the furthest detected point along the obstacle boundary.

Unfortunately, the hovercrafts do not have good location information, so we will implement a variant of tangent bug. The idea is that you should give your hovercraft a target angle for it to travel along. If it detects an obstacle along this path, it should scan (potentially using the node from Section 4.1 or a variant) to find the closest angle to the target angle that is not obstructed. You may want to ignore anything further than 50cm away for this implementation since the room is relatively small compared to range that can be detected by the range finders.

Question: Describe your implementation and develop some small obstacle courses to test your implementation. How well does it work? Characterize your implementation and system experimentally. *While there is only one question in this part of the lab, I expect that your report will have a significant amount of discussion on this section. I'm leaving much of the implementation decisions up to you, so I am also leaving it up to you to decide interesting and appropriate experiments to characterize your robot and algorithm.*

As part of your lab presentation when you hand in your lab we will setup a small obstacle course to test all of the robots on!

[4] You can read the original article: I. Kamon, E. Rimon, and E. Rivlin, "TangentBug: A Range-Sensor-Based Navigation Algorithm," The International Journal of Robotics Research, vol. 17, no. 9, pp. 934–953, 1998. here: `http://ijr.sagepub.com/content/17/9/934.abstract`

5 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 email before the start of class on the due date. A pdf formatted document is preferred.

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 **10 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 10 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 `rxplot` 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?

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