**USC AUV Intro Project**

This project is designed to help you get experience with developing in the ROS environment and introduce you to the majority of things you are going to find in USC's AUV code base. This is designed to encourage you to use Google and in particular the ROS Tutorials, so don't be discouraged if on the first pass it looks confusing.

The task:

Create a package in ROS that functions as a PID controller (a common technique used for stabilization).

Lets begin!

0. Before beginning this project, you should be familiar with the python programming language, as well as have a basic understanding of the structure of ROS.

**Setting up the package**

1. First, the package for the node needs to be created! Go ahead and create this in the AUV ros environment. Understand, you do not need to write the nodes right now, so just create the package! Call this package “auv\_ros\_pid” without the quotes.

2. After this is done, instantiate a git repository in the package. This is so we can keep track of this package and others will be able to access it from the GitHub as well as collaborate on it (safely). Go ahead and make your first commit.

3. At this point, your package should be up and recognized by the ROS ecosystem (try typing “rosrun auv\_ros\_p” and hitting tab a couple times...do you see your package show up?). It should also be a git repository. Go ahead and create the file “README.md” inside of the package. This file should have the name of your package as well as when it was created, what its for, and how it works/how to run it. Understandably, we don't have all those things yet, so just give the name and when it was created. Additionally, as nodes grow, you can create a “documentation” directory and put any and all (and you should have a LOT) documentation in there.

4. Now, create a directory called “scripts” in your package. This houses all of the code for the package, including nodes as well as supporting code for nodes. Go ahead and create a node called “pid”. At this point, we'll look at what exactly we want our package to look like:

Specs:

Package name: auv\_ros\_pid

Node(s) name: pid

Roslaunch: auv\_ros\_pid.launch

Parameters: /auv/pid/proportional

/auv/pid/integral

/auv/pid/derivative

Services provided: ChangePidCoefficients

Subscriber(s): /auv/pid/current\_position

Publisher(s): /auv/pid/output\_velocity

**Writing the nodes**

1. At this point, we can begin writing some of the code for the node! As seen above, there are a couple of topics that we want to interact with. Specifically, we want to listen to a channel that gives our current position, and we want to publish to a channel that tells which direction we should move and how fast (“/auv/pid/current\_position” and “/auv/pid/output\_velocity”, respectively). Likewise, we want a provide a service called “ChangePidCoefficients” that lets another node change the coefficients of the PID on the fly. Finally, we want to write a launch file called auv\_ros\_pid.launch that automatically grabs three parameters from the parameter server on startup that will provide our default coefficients for the PID. Yes, we could easily hard code this in, but that's not the point here!

2. In the scripts directory, create a file called “pid.py”. This will hold all of our code. First things first, let's get the file set up. Put the appropriate headers, and import the modules you know you'll need (obviously, we can go back later and put more if we need). Then, lets just put a simple print statement in, to know that it's getting run correctly.

3.Try and run it and see if you're statement shows up. If so, we're ready to move on!

4. Now, let's set up the subscriber and publisher so we can communicate with the ROS system. For now, just instantiate them, and then put the node in an infinite loop (there's a ROS function specifically designed for this, you should use it here). Start the node up and then run “rostopic list” in another terminal. Do you see the topics we just subscribed to there? If so, we're set to go!

5. Every time a message is published to a topic, ROS runs a callback function. Lets write our callback function for “/auv/pid/current\_position” so when we get an update on our position, we can make the appropriate actions. For now, just write the function and have it a print the message that we received, so we know it's working as expected! Once that's done, save your file, start the node, and in another terminal, run “rosrun intro\_exercise exercise\_1.py”, which will continously publish messages on “/auv/pid/current\_position”. As before, if you see the messages being printed by your node, continue onwards.