Autonomous Underwater Vehicle
Sponsor: NAVSEA
Mentor: Mike Kapus
Alex Koszarek, Matthew Godmere, Feross Salameh, Jason Trost

Overview
Robosub is an Underwater Autonomous Vehicle that is being designed and built by Washington State University and University of Idaho students. The submarine uses sensors, thrusters, cameras, and software to complete specific underwater tasks at a competition held in the summer. Our team’s goal is to develop the AI and Vision systems for the submarine and complete several tasks at competition this year.

Objectives
● Develop a fast computer vision system capable of using multiple cameras and algorithms to identify competition tasks
● Create a high level AI system capable of abstracting away mission details, allowing rapid implementation of missions
● Integrate new systems with existing architecture and be able to perform simple pool tasks

Background
The underlying mindset when developing this year’s software was creating a strong framework that could be built upon in future years and would not need to be modified heavily or replaced. The software from previous years seemed inflexible, non-extensible, and overall incomplete. In particular the previous AI and vision implementations were simply static objects that resembled more of a library than a runtime architecture. It is this issue that led us to developing our new vision and AI systems.

Vision
The computer vision system was designed around a multithreaded environment that could be controlled through a central module. Processes can be spawned through broker commands and utilize a shared memory segment for frambeuffers obtained through each of the cameras. To meet strict performance requirements, vision processes also utilize custom built image filtering trees that minimize copying of data, and allow runtime optimizations of algorithms through OpenCL.

Buoy Image Filtering Example
Source Image
Color Correction / Smoothing
Thresholding / Centroid Moments
This is the original image from an underwater camera. The image is heightened to make the water appear clearer, and the colors of the objects are more prominent. This is the final step before thresholding. The thresholding process is used to separate the objects in the image. The final result is a binary image where the objects are highlighted.

Communicator
The communicator is an object used by the modules to trade messages with the broker. Both the broker and communicator use ZMQ for their communication using sockets to send strings over ports. The broker receives all messages from modules then sends the correct message to the recipient module. All of these messages can also be serialized into C++ objects using RapidJSON.

GUI
Interactive computer vision tool made for rapid vision algorithm development

Artificial Intelligence
The AI system is designed to give the robosub the ability to make intelligent high level decisions based on vision and sensor data.
● Written in Python
● Utilizes a tree structured mission manager to complete required tasks in a hierarchical manner
● Breaks down a competition task into smaller subtasks that involve simple AI objectives such as ‘Orient’ which sets the submarines position

Future of Project
Our movement modules main purpose is to convert AI commands of movement into primitive thrusters values.
● Written in Python and utilizes FuzzyLite
● Converts high level commands such as forward, down, or Goto from the AI into the robosub’s thruster movements

Tools
● ZMQ: Asynchronous Messaging Library
● FuzzyLite: Fuzzy Logic Control Library
● RapidJSON: JSON parser for C++
● OpenCL: framework for writing code across heterogeneous platforms (CPU, GPU, DSP)
● GUI: graphical user interface
● AI - Artificial Intelligence

Acknowledgements
Mentor: Mike Kapus
Team Darkside: Alex Koszarek, Matthew Godmere, Feross Salameh, Jason Trost

Glossary
Tools

Team Darkside
NAVSEA
Advisors: Sakire Arslan Ay & Matt Taylor
University of Idaho CS team
Robosub club of the Palouse

Testing Framework
To conduct tests cases, a simple testing suite was written to eliminate the overhead of testing-facilities libraries while allowing us to modify the code at will.

Logger
In order to view into the complexity of our multithreaded software system, or failures, a custom logger was designed that each process can write to. The test cases are compiled into a single logfile in order to view the system’s execution exactly as it occurred.