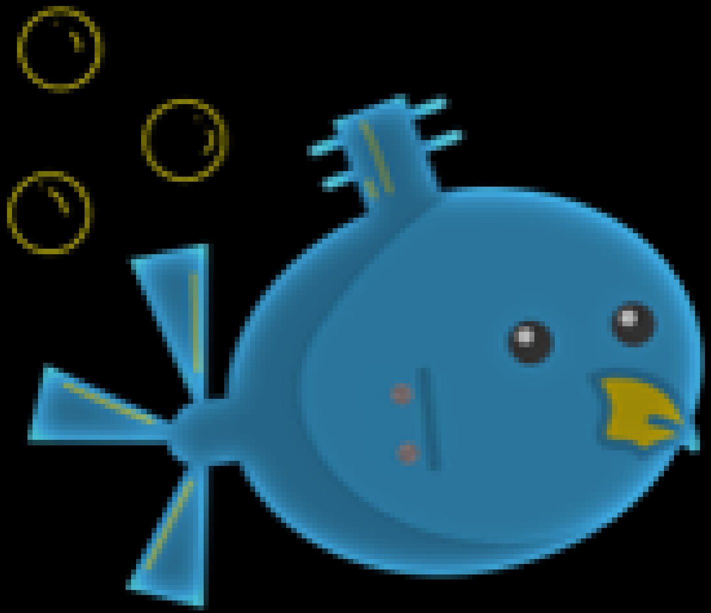
RoboSub



Advisors & Members

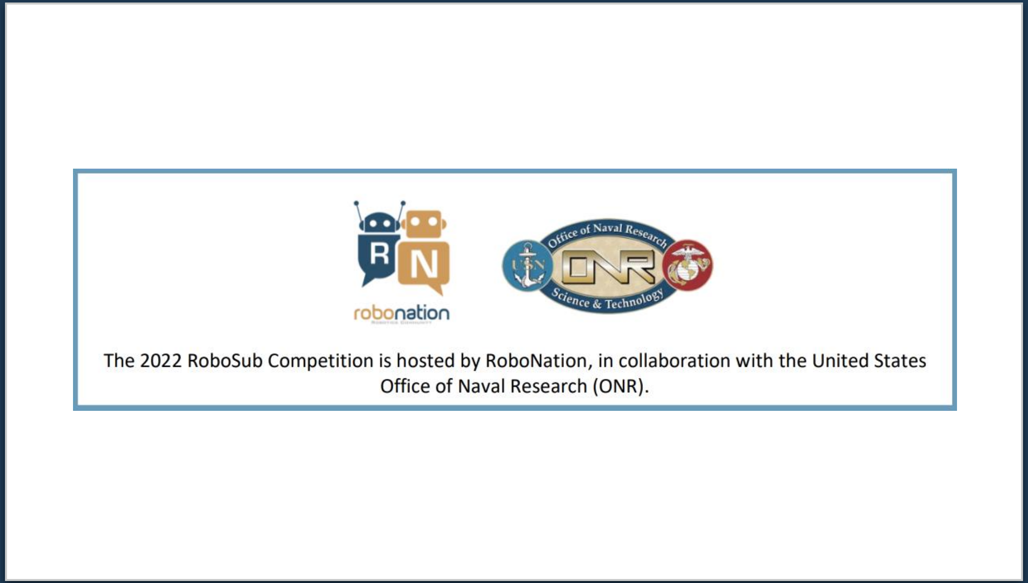
Members: Victor Solis, David Camacho, Hector Mora-Silva, Roberto Hernandez, Bart Rando, Andrew Huesser, Bailey Canham, Milca Ucelo Paiz, Thomas Benson, Brandon Cao

Advisor: Richard Cross

RoboSub Competition

- Hosted every year by RoboNation
  - International Competition
    - 39 in-person teams in 2022
    - More online
  - At University of Maryland
  - Different theme every year

Goal: Design, build, and test an Autonomous Underwater Vehicle (AUV) that will attempt to complete a series of tasks.



The 2022 RoboSub Competition is hosted by RoboNation, in collaboration with the United States Office of Naval Research (ONR).

Team Breakdown

Mechanical & Electrical Engineer Team

7 Mechanical Engineers

3 Electrical Engineers

Computer Science

10 Software Engineers

Lanturn

- Previous Year(s)
  - Not Manufactured
  - Untested Software
  - Picking off where they left off

Software Requirements

- Safety Requirements
- 100% Autonomous
- Artificial Intelligence
  - Decision Making
  - Navigation
  - Object Detection

- Sensors
  - Data Acquisition
- Actuator Control
  - Stabilization
  - Task Execution

Computer Science Team Breakdown

AUTONOMY

CONTROLS

NAVIGATION

COMPUTER VISION

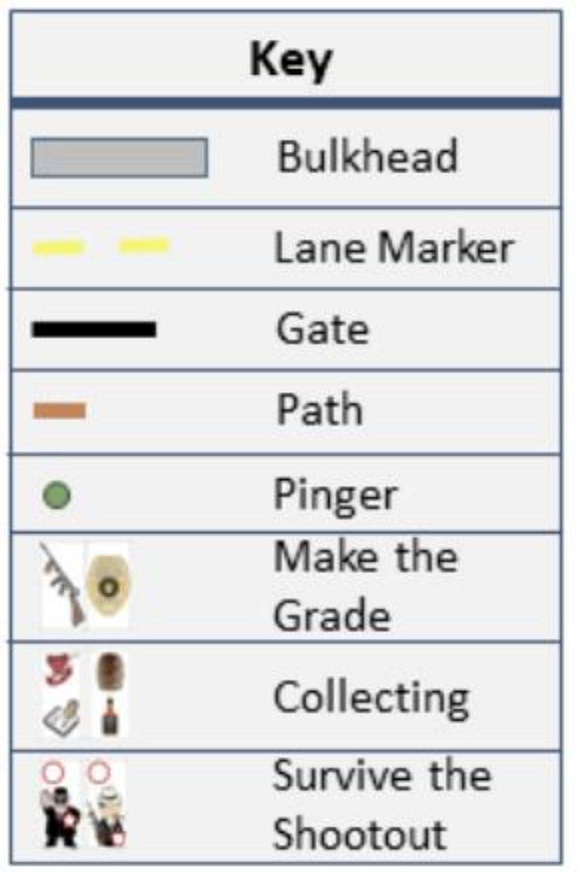
Computer Science Team

- Designing Based on Last year's competition Obstacles
- Mission Handbook released around March
- Once 2023 Mission Handbook is released, make adjustments to fit the requirements

RoboSub Club

- Two Submarines are allowed
- Senior Design RoboSub
- Club RoboSub
- Intervehicle Communication
- Points Based on Complexity of Communication

Competition Map



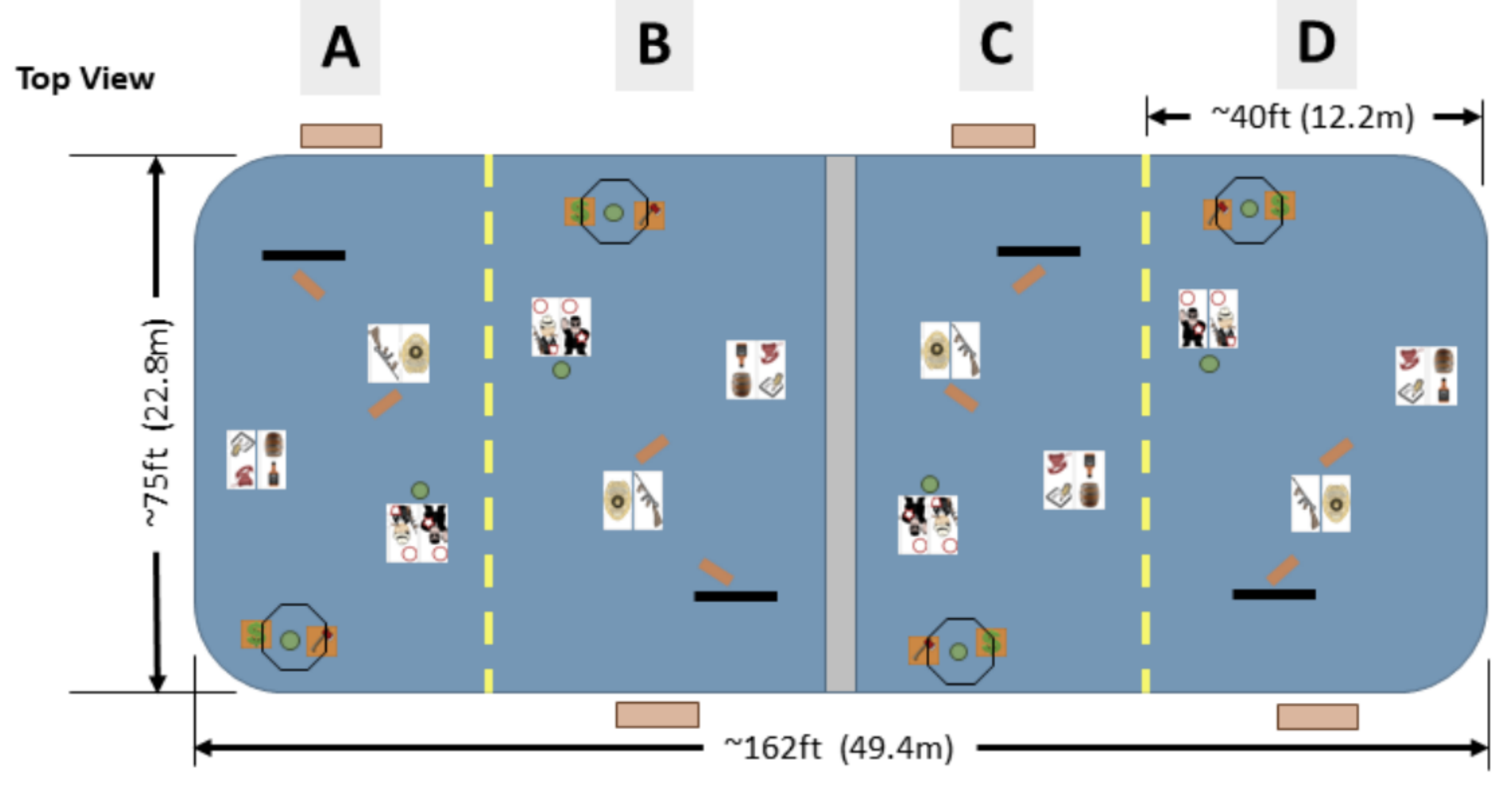
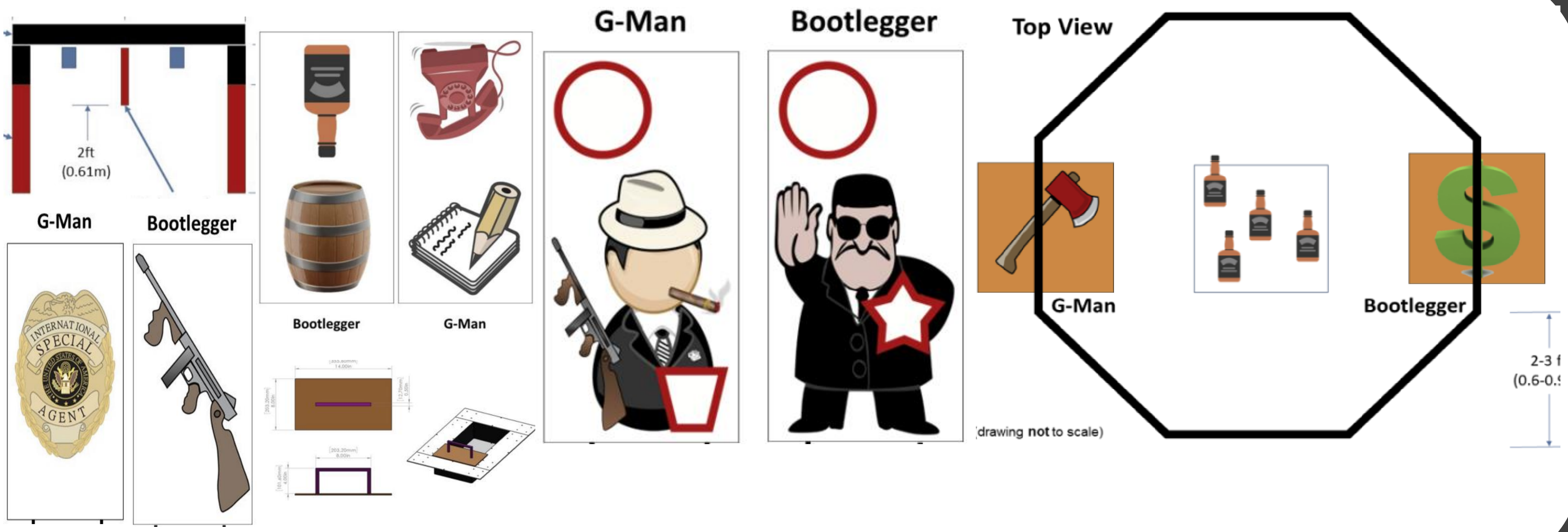
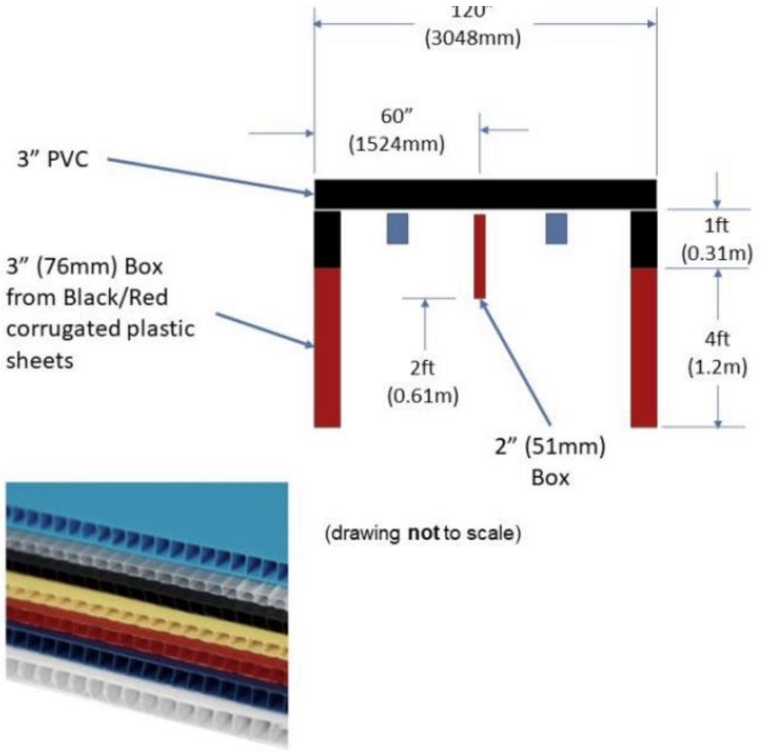


Figure 2: RoboSub Course Layout

Tasks


2ft
(0.61m)
G-Man
Bootlegger
INTERNATIONAL
SPECIAL
AGENT
Bootlegger
G-Man
G-Man
Bootlegger
Top View
G-Man
Bootlegger
(drawing not to scale)
2-3 f
(0.6-0.

Start:
Choose Gate


120"
(3048mm)
60"
(1524mm)
3" PVC
3" (76mm) Box from Black/Red corrugated plastic sheets
1ft
(0.31m)
4ft
(1.2m)
2ft
(0.61m)
2" (51mm) Box
(drawing **not** to scale)

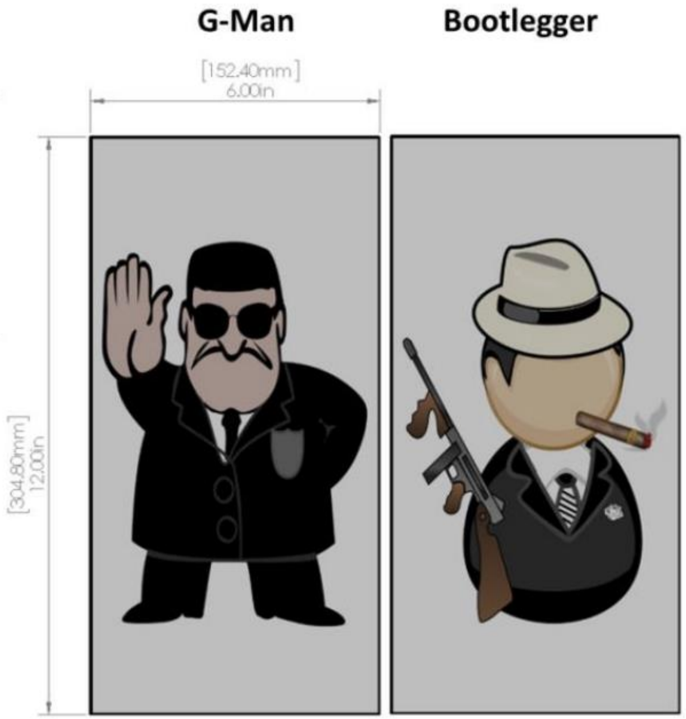


Figure 4: Choose Your Side

Follow Path

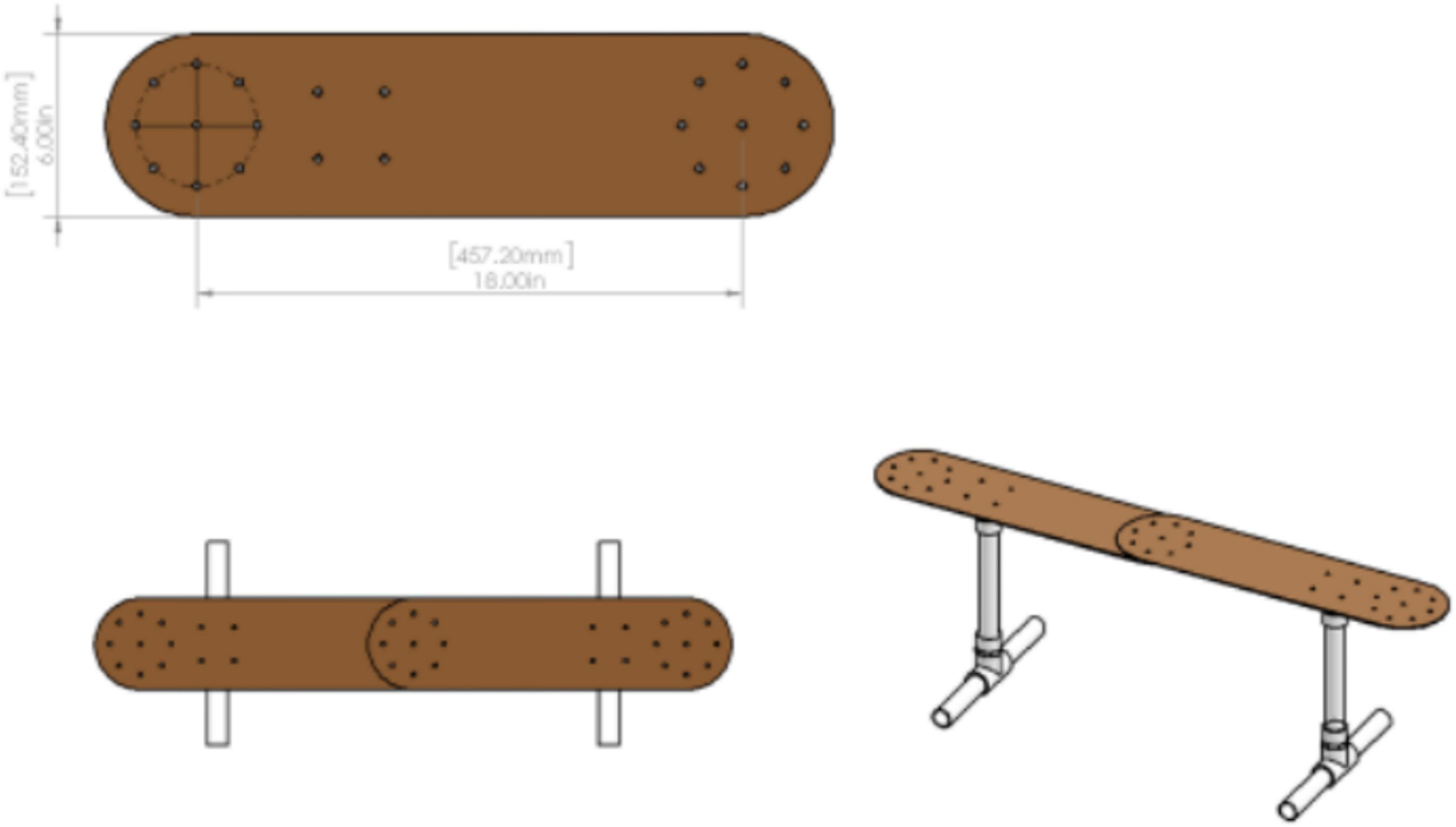


Figure 5: Path markers

Touch Buoy

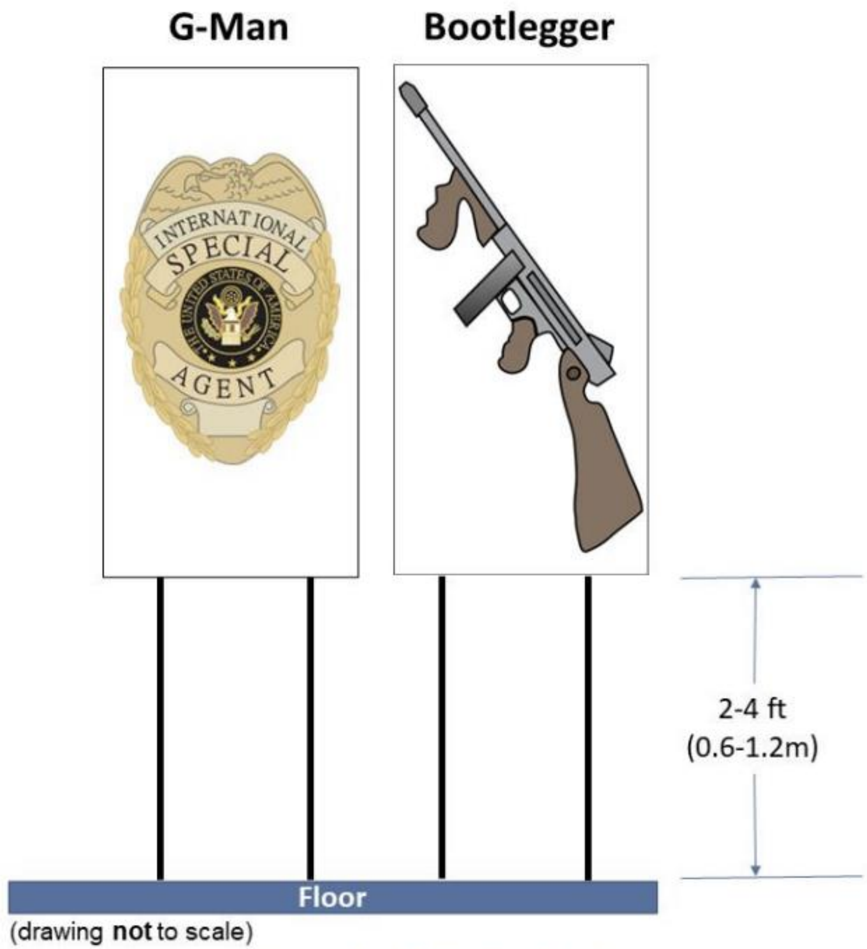
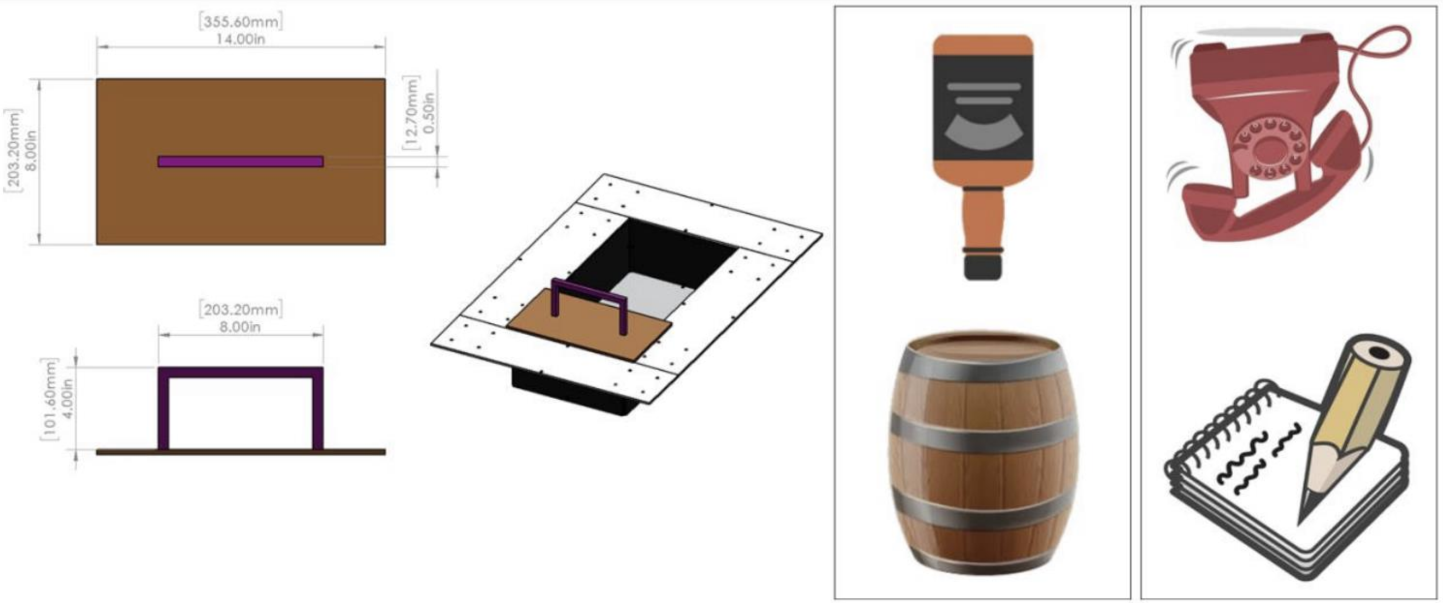


Figure 6 : Make the Grade

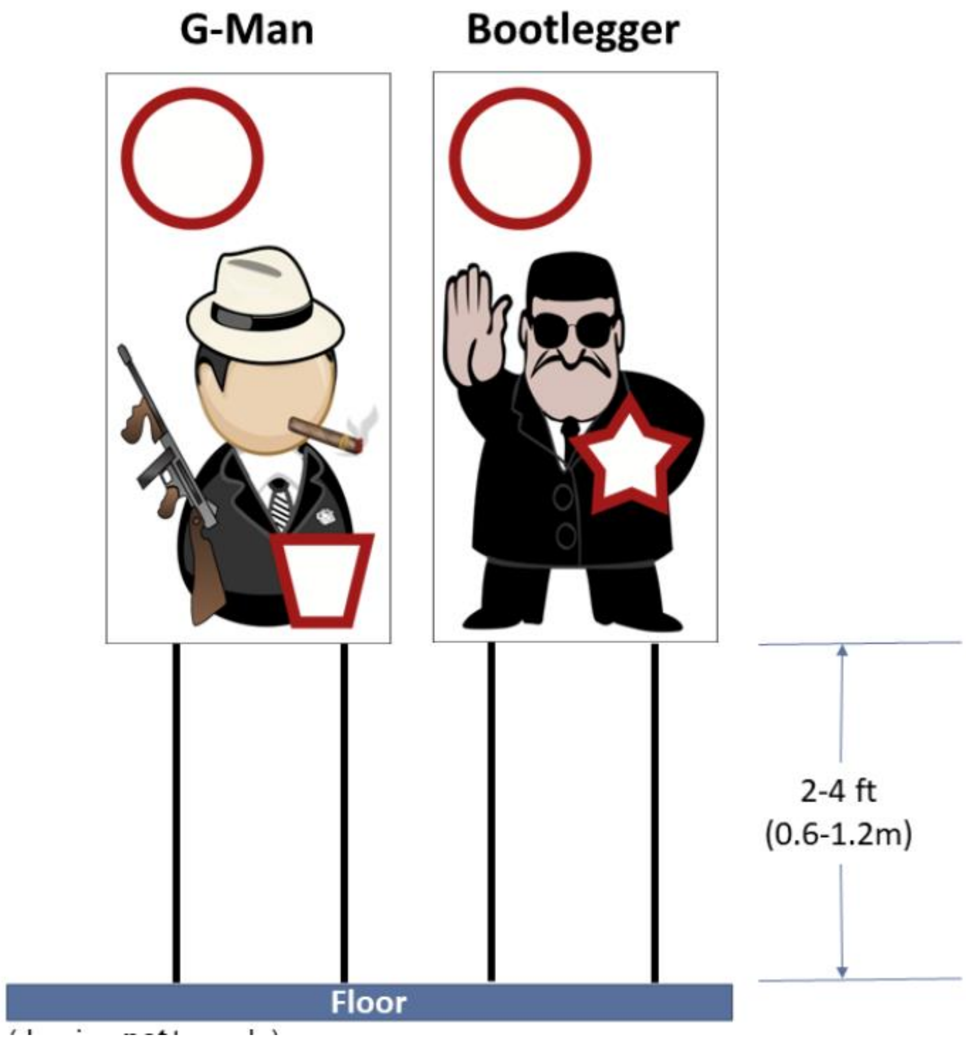
Collecting Bins



Bootlegger **G-Man**

Figure 7: Collecting

Torpedoes



Cash or Smash - Octagon

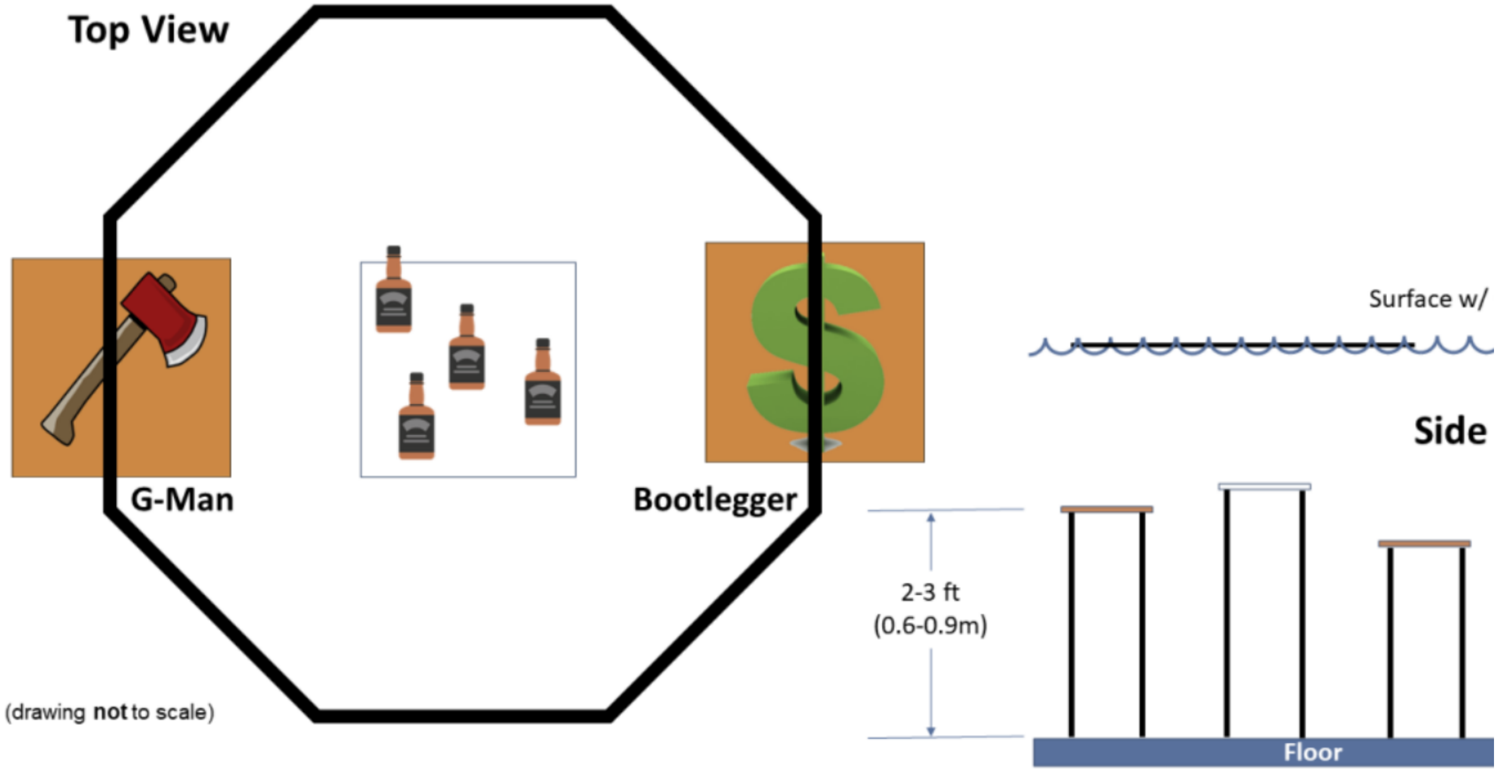


Figure 9: Cash or Smash

Autonomy

Hector Mora-Silva, Roberto Hernandez

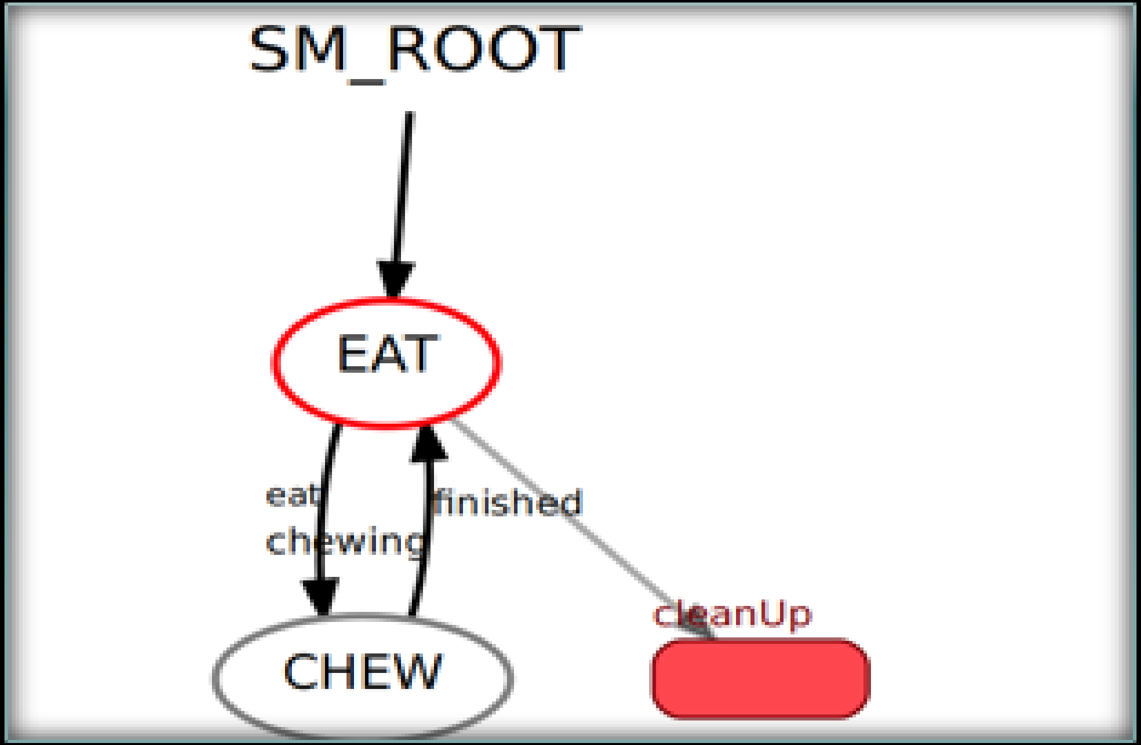
- Objective:
  - Implement and design the overall software structure for the AUV
  - Receive and provide data to all systems
  - Decisions and give instructions for the required tasks based on the current state

- States Overview
  - State Zero
  - 6 main tasks

ROS and SMACH

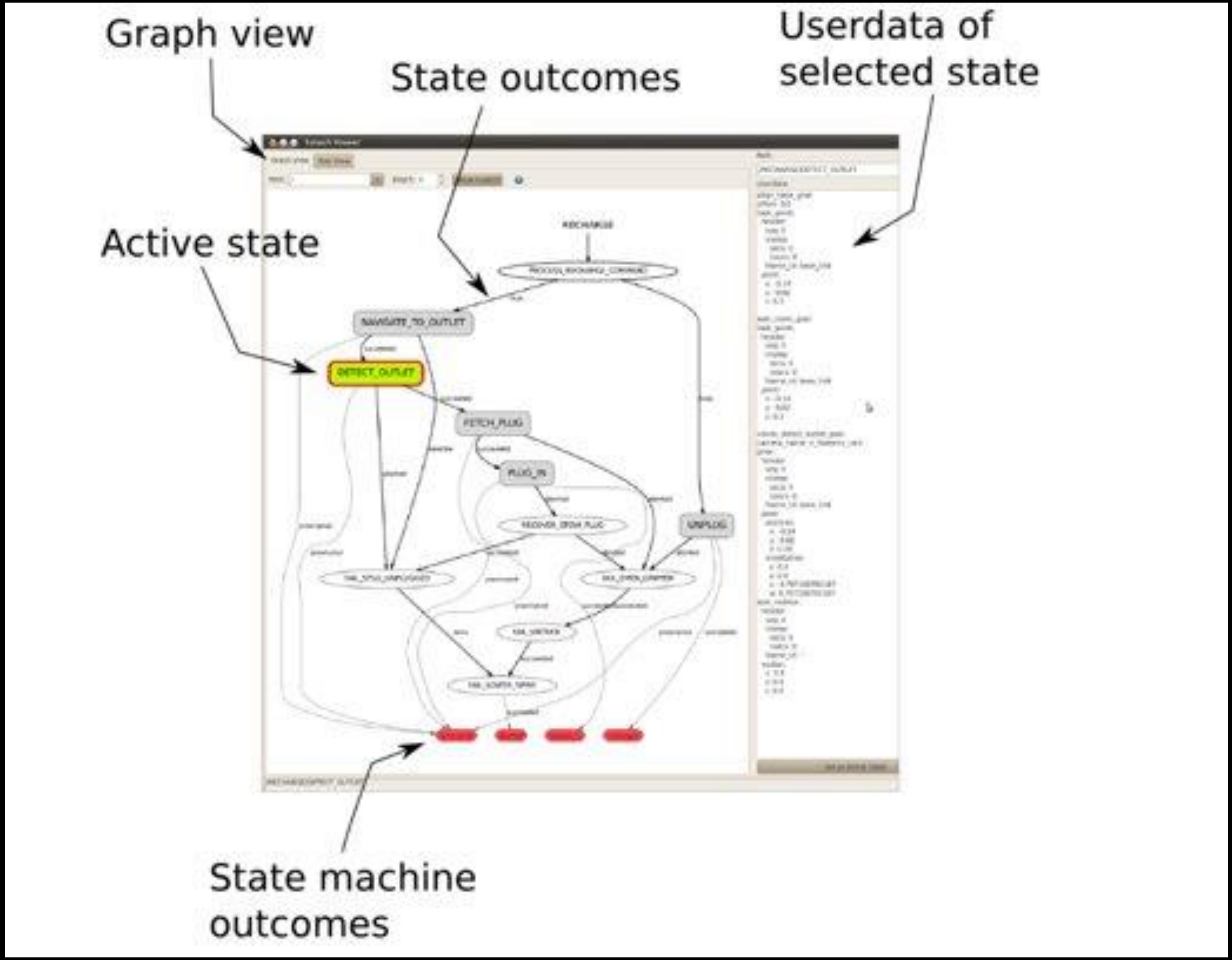
- Robot Operating System (ROS):
  - Libraries and tools to help with the development of robot applications
  - Publishers/Subscribers
    - A Publisher continually broadcasts a message
    - A Subscriber receives these messages

- State Machine (Smach): A package within ROS
  - ROS-independent Python library
  - Hierarchical state machines
  - Mainly to execute complex plans
  - The state machine on the right is a flow chart that shows different states with given outcomes that determine how the states transition



State Machine Viewer

- GUI that shows the state machines
  - Possible transitions between states
  - Currently active state
  - Values of user data that is passed around between states



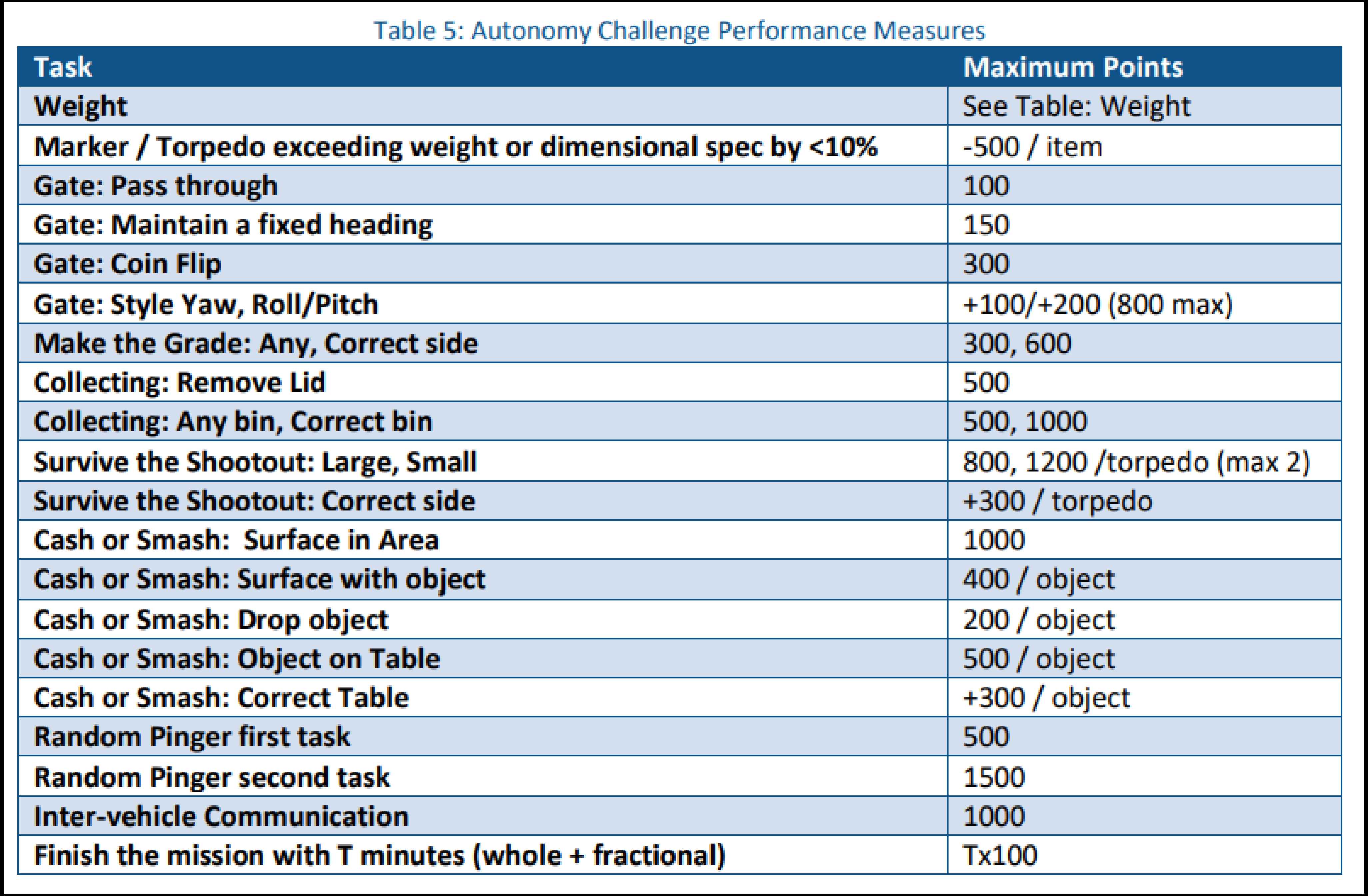
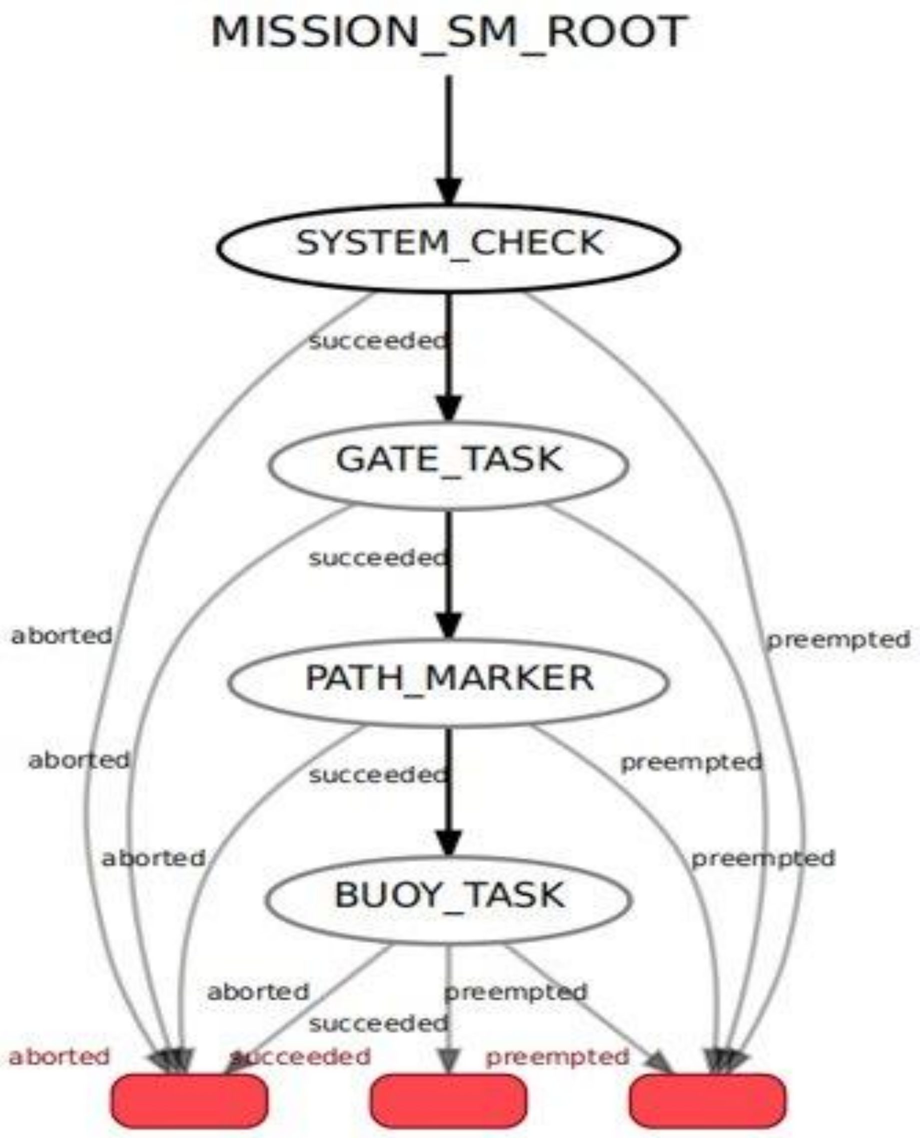


Table 5: Autonomy Challenge Performance Measures

Task	Maximum Points
Weight	See Table: Weight
Marker / Torpedo exceeding weight or dimensional spec by <10%	-500 / item
Gate: Pass through	100
Gate: Maintain a fixed heading	150
Gate: Coin Flip	300
Gate: Style Yaw, Roll/Pitch	+100/+200 (800 max)
Make the Grade: Any, Correct side	300, 600
Collecting: Remove Lid	500
Collecting: Any bin, Correct bin	500, 1000
Survive the Shootout: Large, Small	800, 1200 /torpedo (max 2)
Survive the Shootout: Correct side	+300 / torpedo
Cash or Smash: Surface in Area	1000
Cash or Smash: Surface with object	400 / object
Cash or Smash: Drop object	200 / object
Cash or Smash: Object on Table	500 / object
Cash or Smash: Correct Table	+300 / object
Random Pinger first task	500
Random Pinger second task	1500
Inter-vehicle Communication	1000
Finish the mission with T minutes (whole + fractional)	Tx100


MISSION_SM_ROOT
SYSTEM_CHECK
succeeded
GATE_TASK
succeeded
aborted
preempted
PATH_MARKER
aborted
preempted
succeeded
aborted
preempted
BUOY_TASK
aborted
preempted
succeeded
aborted
succeeded
preempted

Transition From Ros1 to Ros2

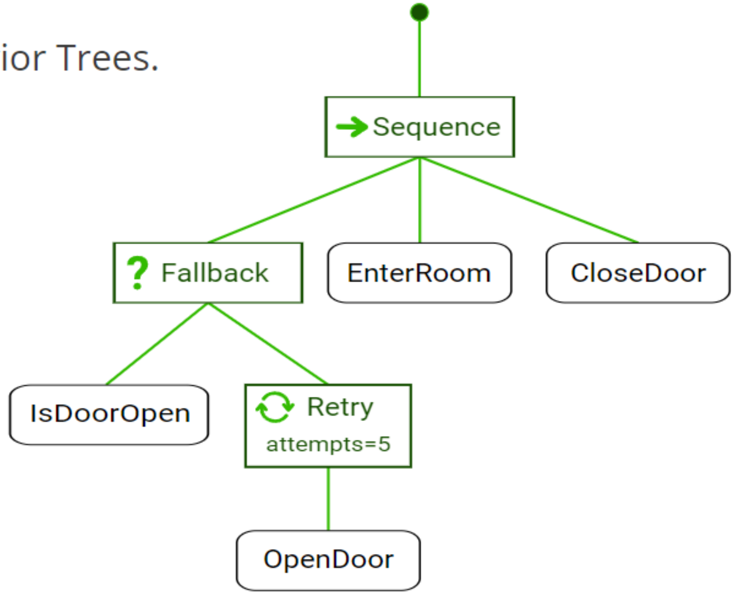
Started with ROS1 due to our current hardware. Unfortunately, the target operating system, newer software and adding other components started to be limited, so moving to a newer release from ROS2 was decided. Which brings advantages like using Behavior Trees instead of Finite State Machines.

Transition Finite State Machine to Behavior Trees

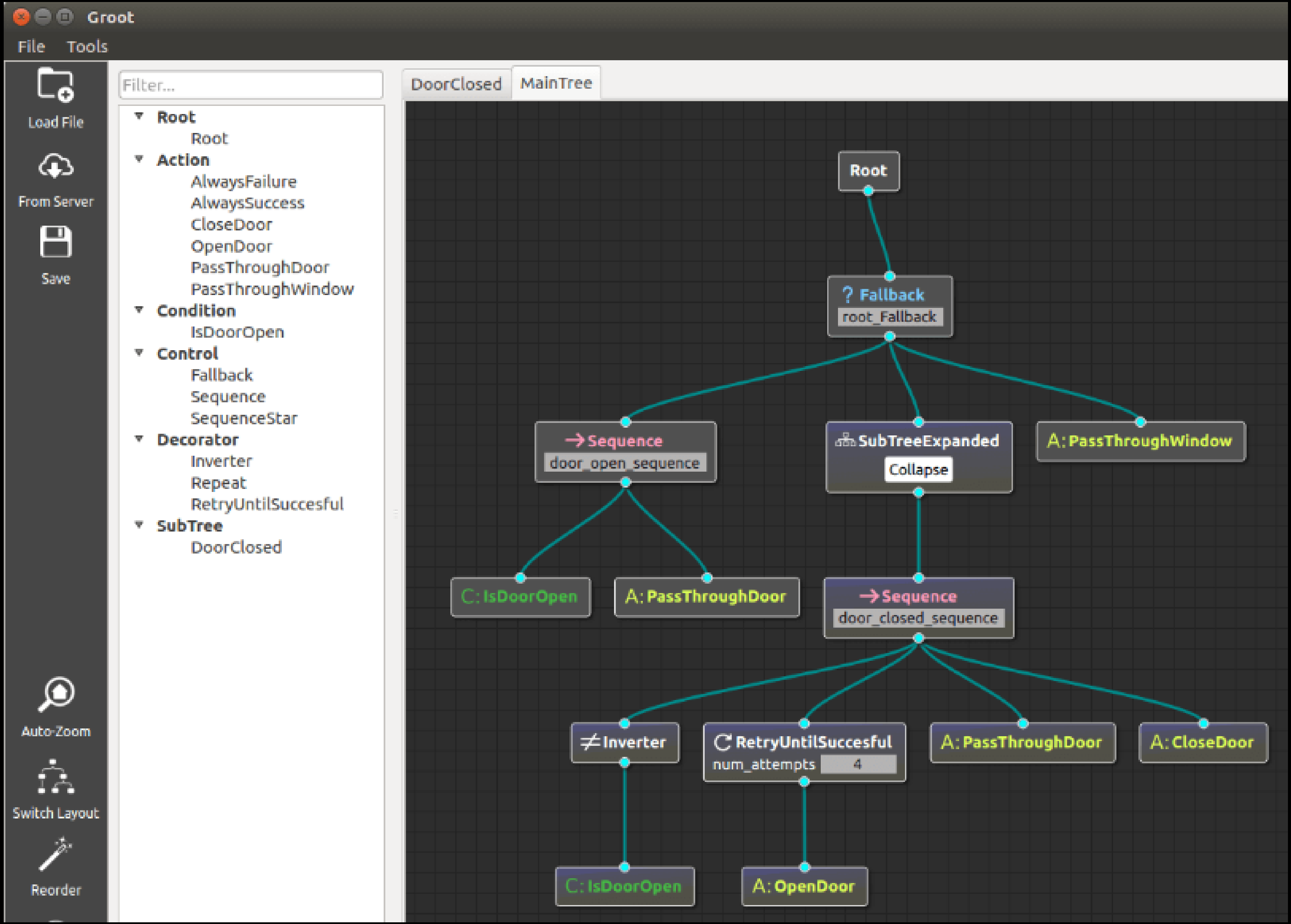
Focal Fossal

BehaviorTree.CPP

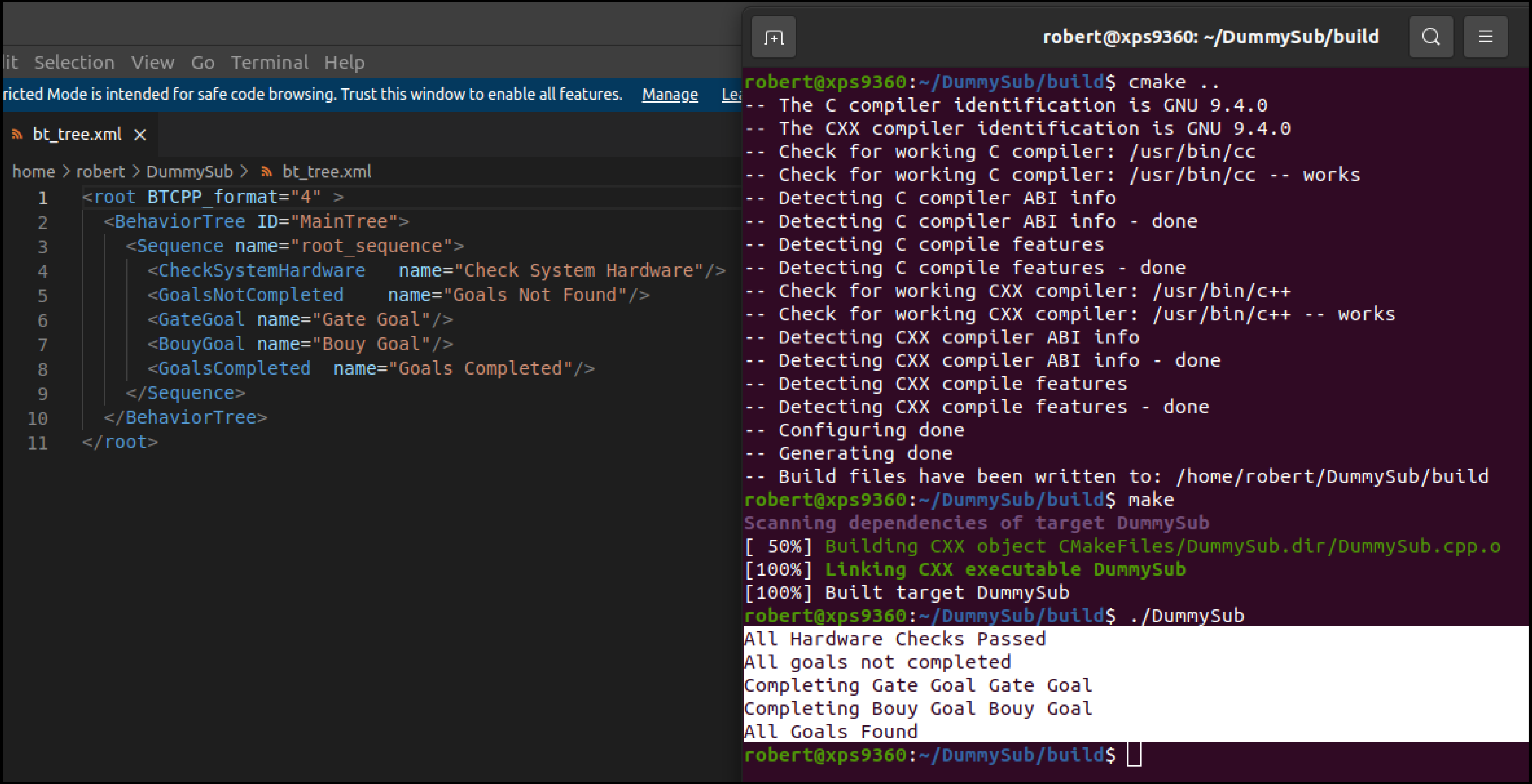
The C++ library to build Behavior Trees.
Batteries included.



- Visual Oriented
- Parrel task/function/states
- Goal Oriented vs Decision Oriented
- Higher Flexibility, more modular =easier to make changes
- Easier to reach functions/actions from another node directly


Groot
File
Tools
Load File
From Server
Save
Auto-Zoom
Switch Layout
Reorder
Filter...
Root
Root
Action
AlwaysFailure
AlwaysSuccess
CloseDoor
OpenDoor
PassThroughDoor
PassThroughWindow
Condition
IsDoorOpen
Control
Fallback
Sequence
SequenceStar
Decorator
Inverter
Repeat
RetryUntilSuccesful
SubTree
DoorClosed
DoorClosed
MainTree
Root
? Fallback
root_Fallback
→ Sequence
door_open_sequence
SubTreeExpanded
Collapse
A: PassThroughWindow
C: IsDoorOpen
A: PassThroughDoor
→ Sequence
door_closed_sequence
≠ Inverter
RetryUntilSuccesful
num_attempts 4
A: PassThroughDoor
A: CloseDoor
C: IsDoorOpen
A: OpenDoor

Basic sequence with Boolean 'success' for Nodes, next is implement logic via Groot and code hardware data .


it Selection View Go Terminal Help
bt_tree.xml
home > robert > DummySub > bt_tree.xml
robert@xps9360: ~/DummySub/build

ricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Lea

```xml
<root BTCPP_format="4" >
  <BehaviorTree ID="MainTree">
    <Sequence name="root_sequence">
      <CheckSystemHardware   name="Check System Hardware"/>
      <GoalsNotCompleted    name="Goals Not Found"/>
      <GateGoal name="Gate Goal"/>
      <BouyGoal name="Bouy Goal"/>
      <GoalsCompleted  name="Goals Completed"/>
    </Sequence>
  </BehaviorTree>
</root>
```

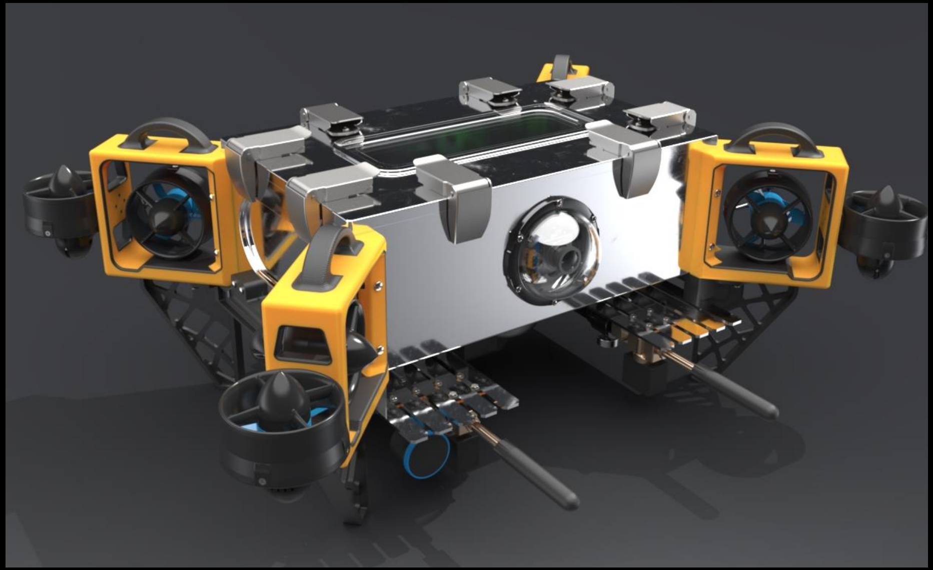
```
robert@xps9360:~/DummySub/build$ cmake ..
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/robert/DummySub/build
robert@xps9360:~/DummySub/build$ make
Scanning dependencies of target DummySub
[ 50%] Building CXX object CMakeFiles/DummySub.dir/DummySub.cpp.o
[100%] Linking CXX executable DummySub
[100%] Built target DummySub
robert@xps9360:~/DummySub/build$ ./DummySub
All Hardware Checks Passed
All goals not completed
Completing Gate Goal Gate Goal
Completing Bouy Goal Bouy Goal
All Goals Found
robert@xps9360:~/DummySub/build$ 
```

Computer Vision

Bailey Canham, Milca Ucelo

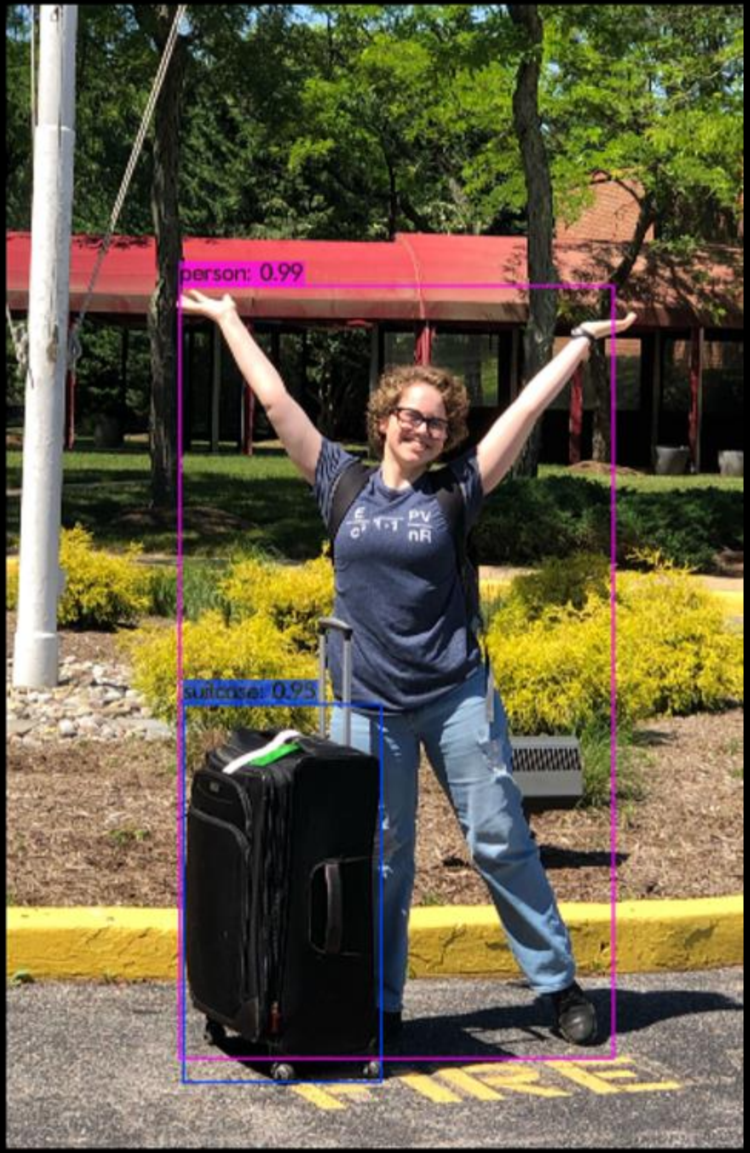
Overall Team Goal

- Our goal is to be able to identify important competition objects with a 70% accuracy
- We also would like to extrapolate as much information as possible from the identified objects, such as being able to tell how far the robot must rotate to be perpendicular to the object
- Additionally, we plan to utilize Google CoLab and evaluate how effective this tool is



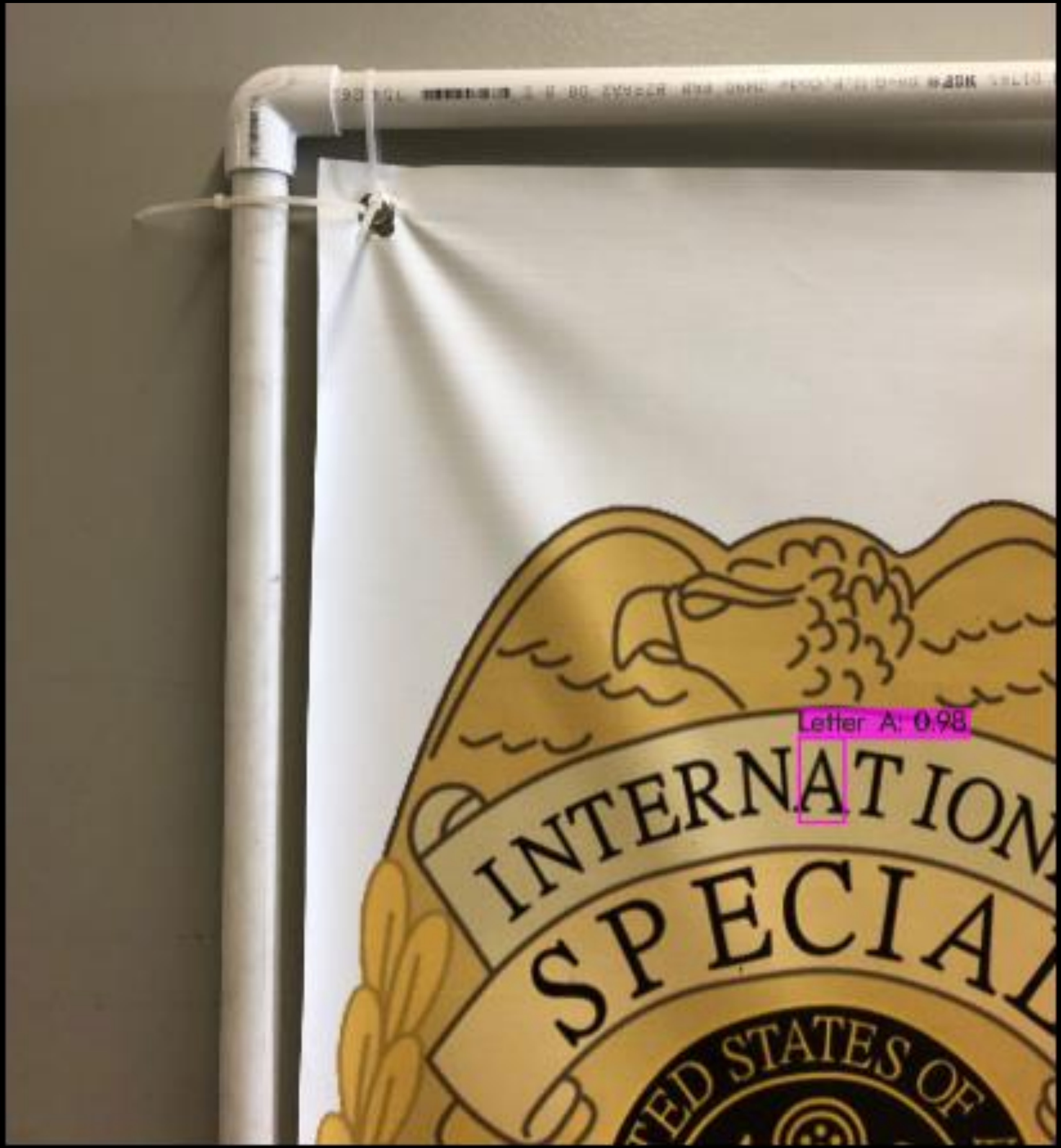
First Step: Exploration

- We wanted to get a better idea of how computer vision programs work and how to test them on Google CoLab
- We used a pre-trained computer vision program in order to get comfortable with how to use and evaluate these programs.



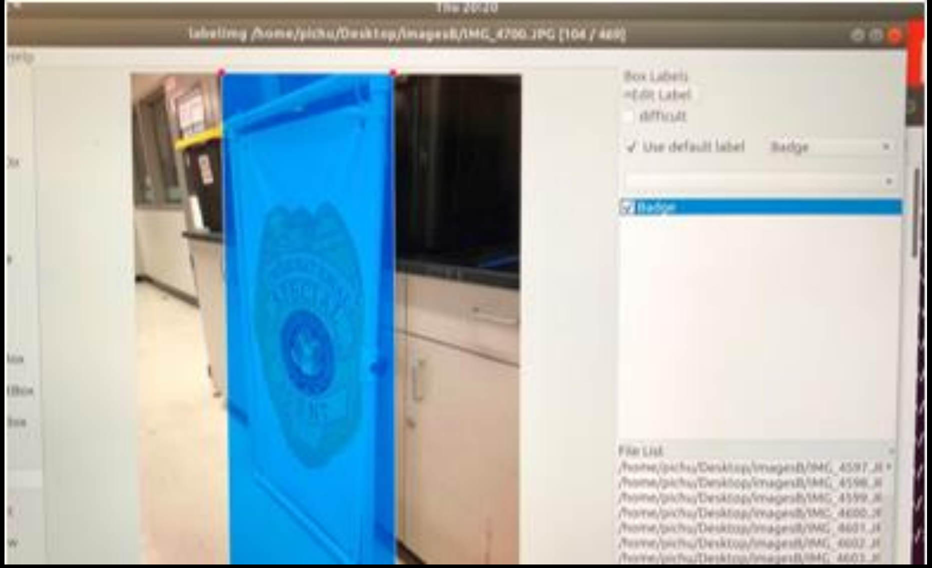
Second Step: Evaluation

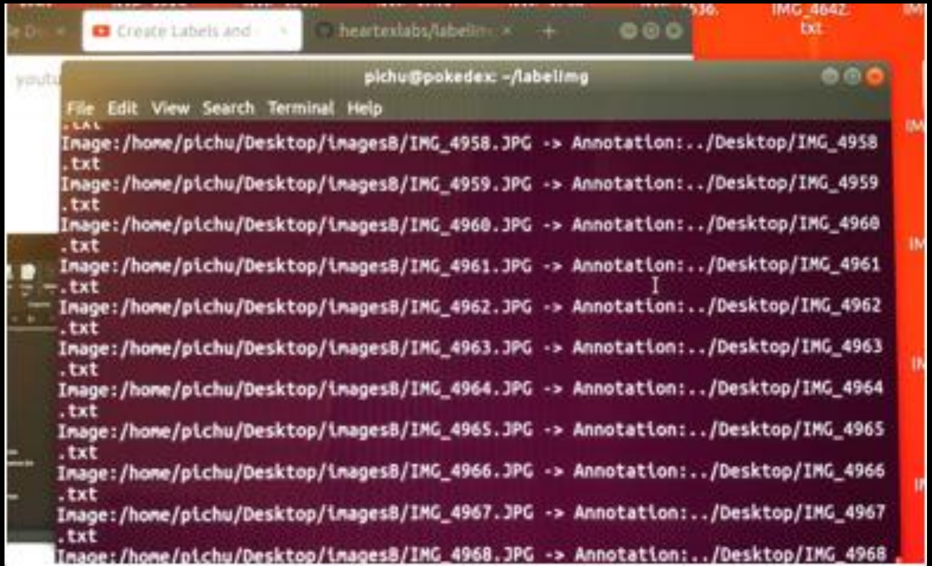
- We wanted to see how much progress the previous senior design team made in computer vision
- After testing the program, we learned that their computer vision program was only able to recognize the letter ‘A’
- Therefore, we needed to train our own model



Third Step: Prepare

- To train, we need to have an abundance of photos of the objects we wished to recognize, so we took over 500 photos of the objects we had available: The Badge Buoy and the Prequel Gate
- We then had to label each image using the program LabelIMG
- This produced a txt file that related to each photo

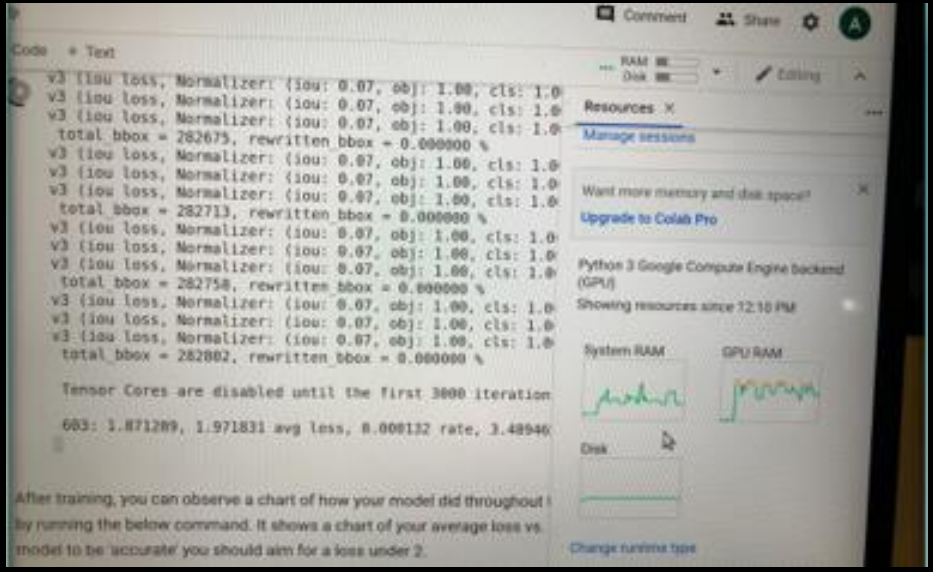

labelImg /home/pichu/Desktop/imagesB/IMG_4700.JPG [104 / 469]
Box Labels
Edit Label
difficult
✓ Use default label
Badge
Badge
File List
/home/pichu/Desktop/imagesB/IMG_4597.JP
/home/pichu/Desktop/imagesB/IMG_4598.JP
/home/pichu/Desktop/imagesB/IMG_4599.JP
/home/pichu/Desktop/imagesB/IMG_4600.JP
/home/pichu/Desktop/imagesB/IMG_4601.JP
/home/pichu/Desktop/imagesB/IMG_4602.JP
/home/pichu/Desktop/imagesB/IMG_4603.JP



```
Image:/home/pichu/Desktop/imagesB/IMG_4958.JPG -> Annotation:../Desktop/IMG_4958
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4959.JPG -> Annotation:../Desktop/IMG_4959
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4960.JPG -> Annotation:../Desktop/IMG_4960
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4961.JPG -> Annotation:../Desktop/IMG_4961
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4962.JPG -> Annotation:../Desktop/IMG_4962
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4963.JPG -> Annotation:../Desktop/IMG_4963
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4964.JPG -> Annotation:../Desktop/IMG_4964
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4965.JPG -> Annotation:../Desktop/IMG_4965
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4966.JPG -> Annotation:../Desktop/IMG_4966
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4967.JPG -> Annotation:../Desktop/IMG_4967
.txt
Image:/home/pichu/Desktop/imagesB/IMG_4968.JPG -> Annotation:../Desktop/IMG_4968
```

Fourth Step: Train

- We created the other needed files (obj.data, obj.names, yolov4.cfg, train.txt)
- Our first training attempt lasted 5 hours and got through around 500 iterations
- Google CoLab then kicked us out for using too many resources
- However, we were able to continue the training process once we were allowed access again and trained up to 1000 iterations



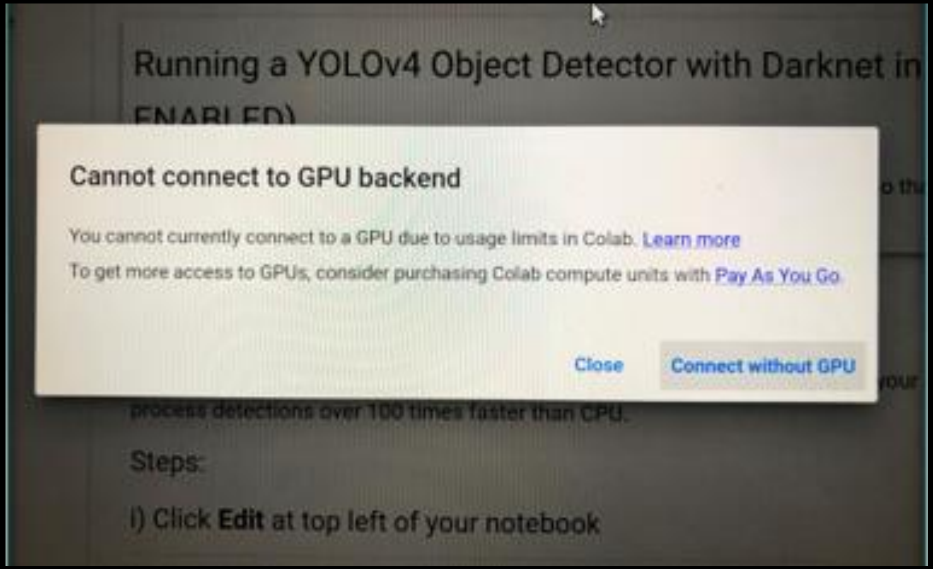
```
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
 total_bbox = 282675, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
 total_bbox = 282713, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
 total_bbox = 282758, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.0
 total_bbox = 282802, rewritten_bbox = 0.000000 %

 Tensor Cores are disabled until the first 3000 iteration

 603: 1.871289, 1.971831 avg loss, 0.000132 rate, 3.48946
```

After training, you can observe a chart of how your model did throughout
by running the below command. It shows a chart of your average loss vs
model to be 'accurate' you should aim for a loss under 2.

Want more memory and disk space?

Python 3 Google Compute Engine backend (GPU)

Showing resources since 12:10 PM



Running a YOLOv4 Object Detector with Darknet in

ENABLED)

Cannot connect to GPU backend

You cannot currently connect to a GPU due to usage limits in Colab. Learn more

To get more access to GPUs, consider purchasing Colab compute units with Pay As You Go

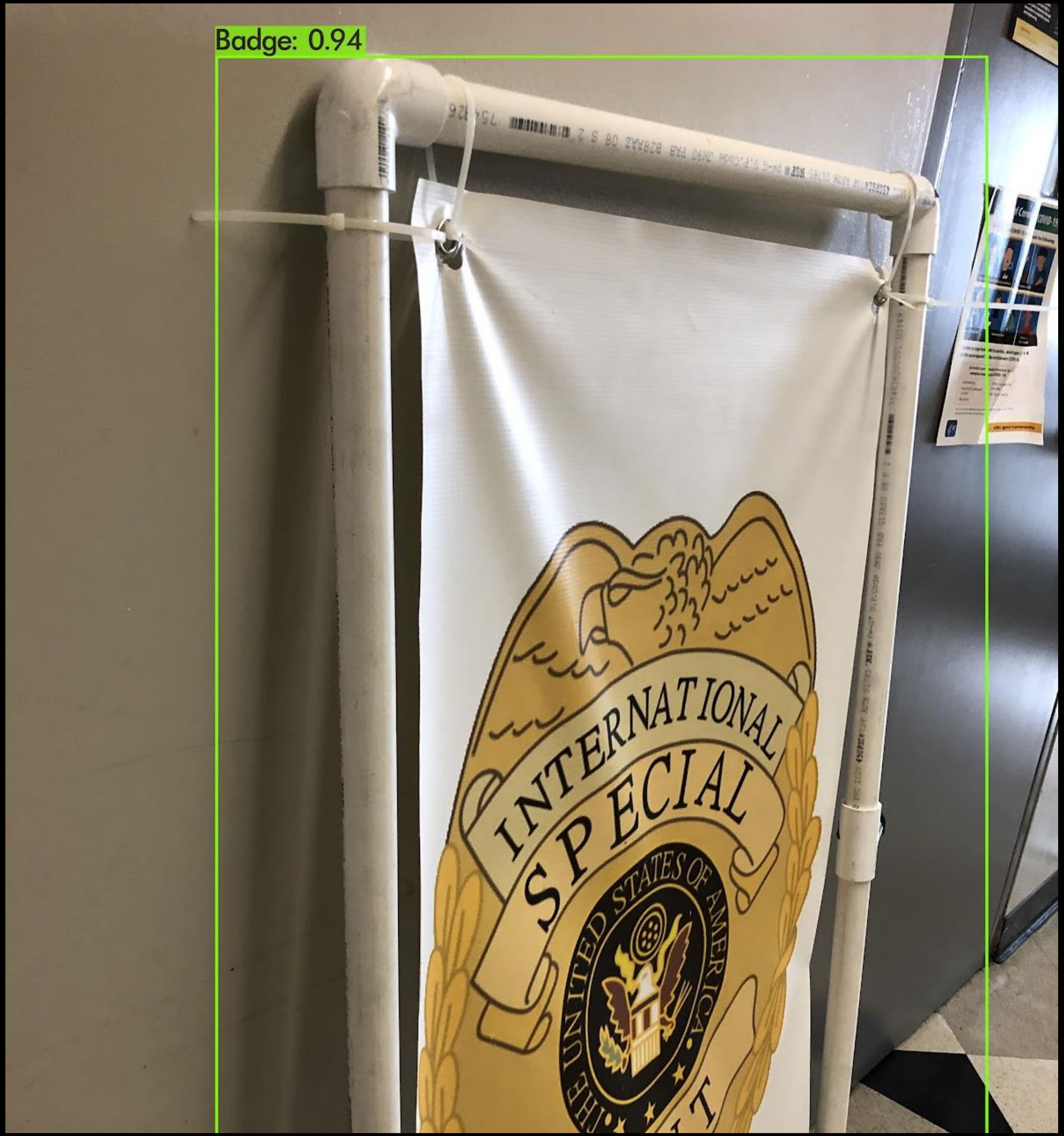
process detections over 100 times faster than CPU.

Steps:

i) Click **Edit** at top left of your notebook

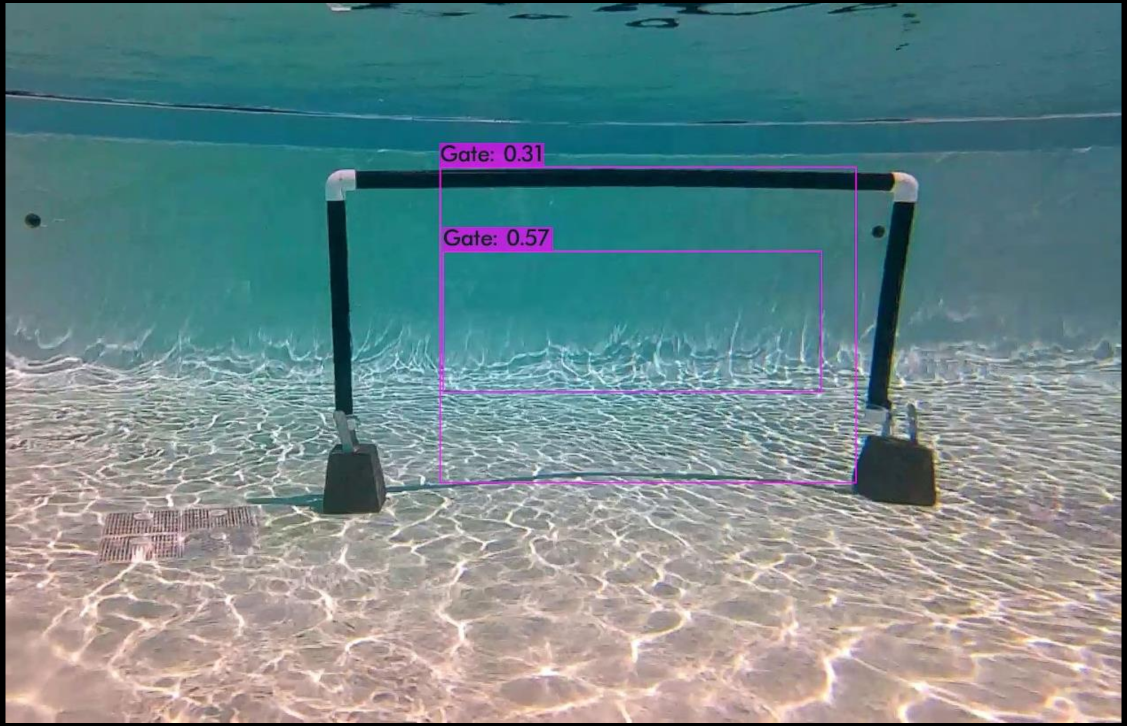
Fifth Step: Test

- Using our training, we tested other images of the Badge Buoy and the Prequel Gate
- The model was able to identify the Badge Buoy consistently, no matter orientation or cropping
- The model had a harder time identifying the Prequel Gate, but was still overall successful



Next Steps

- We will continue to improve our computer vision model to increase accuracy
- We will start to explore avenues of how to increase information flow between the computer vision system and the navigation system



Control Systems

BART RANDO / ANDREW HEUSSER

Initial Goals

1. Organize the existing code base into an Object Oriented design

2. Create modular classes for each distinct control system to simplify future expansions of the code base

3. Create new classes to support sensors which have not yet been implemented (VN-100, DVL, Hydrophones, Sonar)

4. Deploy, Test, and Refine the code to maximize accuracy and efficiency

Where we began & Initial Difficulties

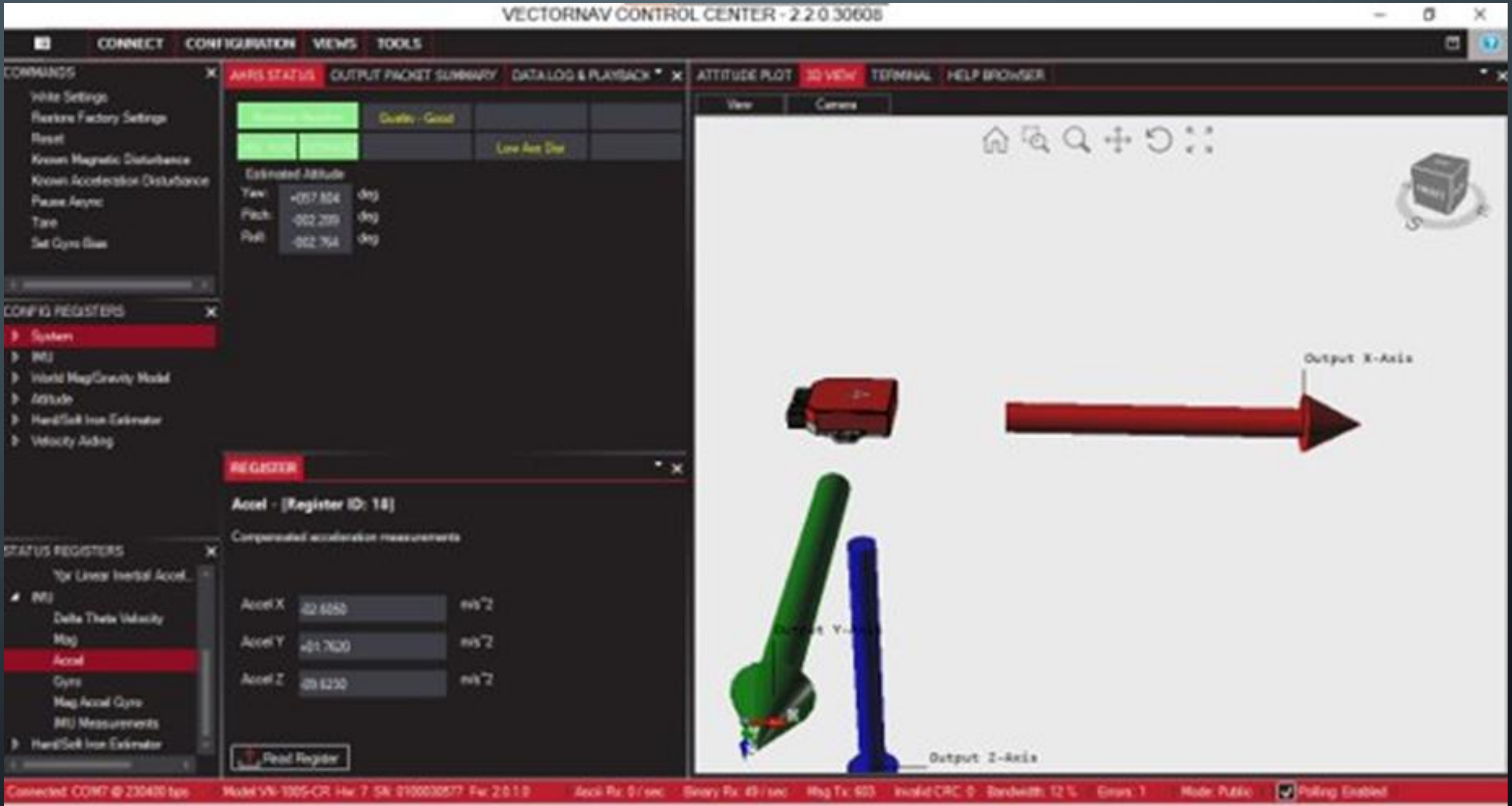
- The IMU (Inertial Measurement Unit) is the most vital sensor of all the control systems. This year we plan to upgrade the IMU from the Adafruit BNO055 to the Vector Nav VN-100.
- Our first goal was to write a working class that would allow our Teensy microcontroller to interface with the VN-100
- Since the VN-100 is extremely advanced and completely different from the BN0055, none of the previously written code is compatible with it.
- Vector Nav also closely controls the distribution of their proprietary libraries and instructions.
- The VN-100 is expensive and unique, so there isn't a lot of documentation freely available explaining the process as to how to implement it, and we need to be extremely careful in handling and wiring it, since any mistake could be very costly.
- Testing code for the VN-100 is not possible through simulation. The VN-100 must be physically connected to the Teensy for us to test if our code is working properly.
- The VN-100 was unused and has never been connected to a Teensy by anyone at CSULA previously. Nor was any printed documentation included with the sensor.

STEPS INVOLVED IN TRANSITIONING TO THE VN-100

- We needed to file an application get approved by Vector Nav to gain access to their proprietary libraries and control software
- We needed to modify the Vector Nav “Rugged External Connector” cable by adding a quick disconnect plug so we could attach it to the Teensy microcontroller, as well as to the control software as needed
- We needed to solder pins to the teensy board and assemble a breakout board to be able to connect and test various sensors during the development and testing phases of programming
- We needed to write a modular class specifically for the VN-100 that communicates through the UART communications format

Vector Nav Control Software

Sensor Position Visualization



Estimated Attitude

Yaw +057.804 deg
Pitch -002.209 deg
Roll -002.764 deg

Accel - [Register ID: 18]

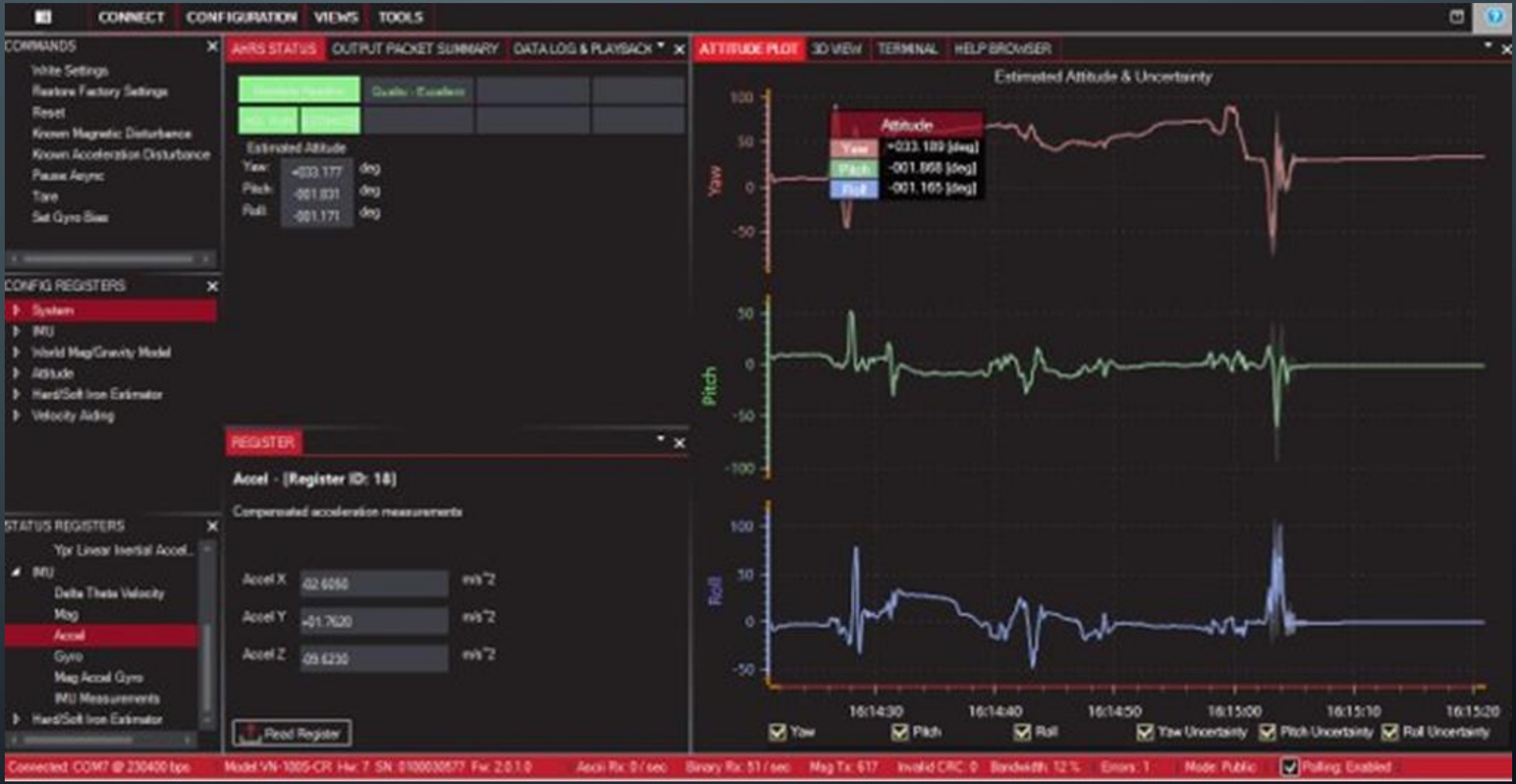
Compensated acceleration measurements

Accel X 02.6050 m/s^2
Accel Y +01.7620 m/s^2
Accel Z 09.6230 m/s^2

Connected: COM7 @ 230400 bps Model VN-100S-CR Hw 7 SN 0100030577 Fw 2.0.1.0 Ascii Rx: 0 / sec Binary Rx: 49 / sec Msg Tx: 603 Invalid CRC: 0 Bandwidth: 12 % Errors: 1 Mode: Public ☑ Polling Enabled

Vector Nav Control Software

Real Time Data and Configuration



Estimated Attitude

Yaw: +033.177 deg
Pitch: -001.031 deg
Roll: -001.171 deg

Attitude
Yaw +033.189 [deg]
Pitch -001.868 [deg]
Roll -001.165 [deg]

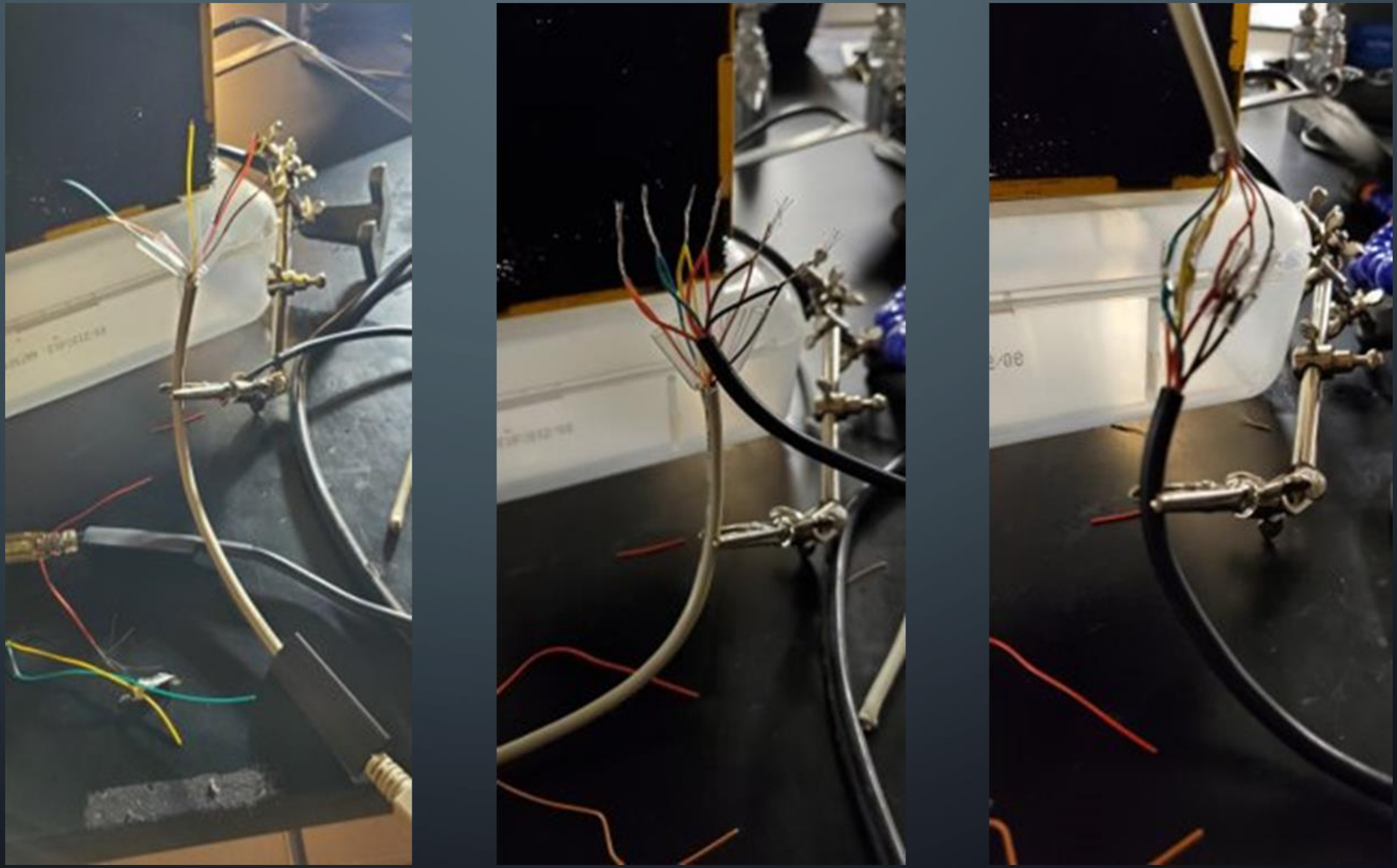
Accel - (Register ID: 18)

Compensated acceleration measurements

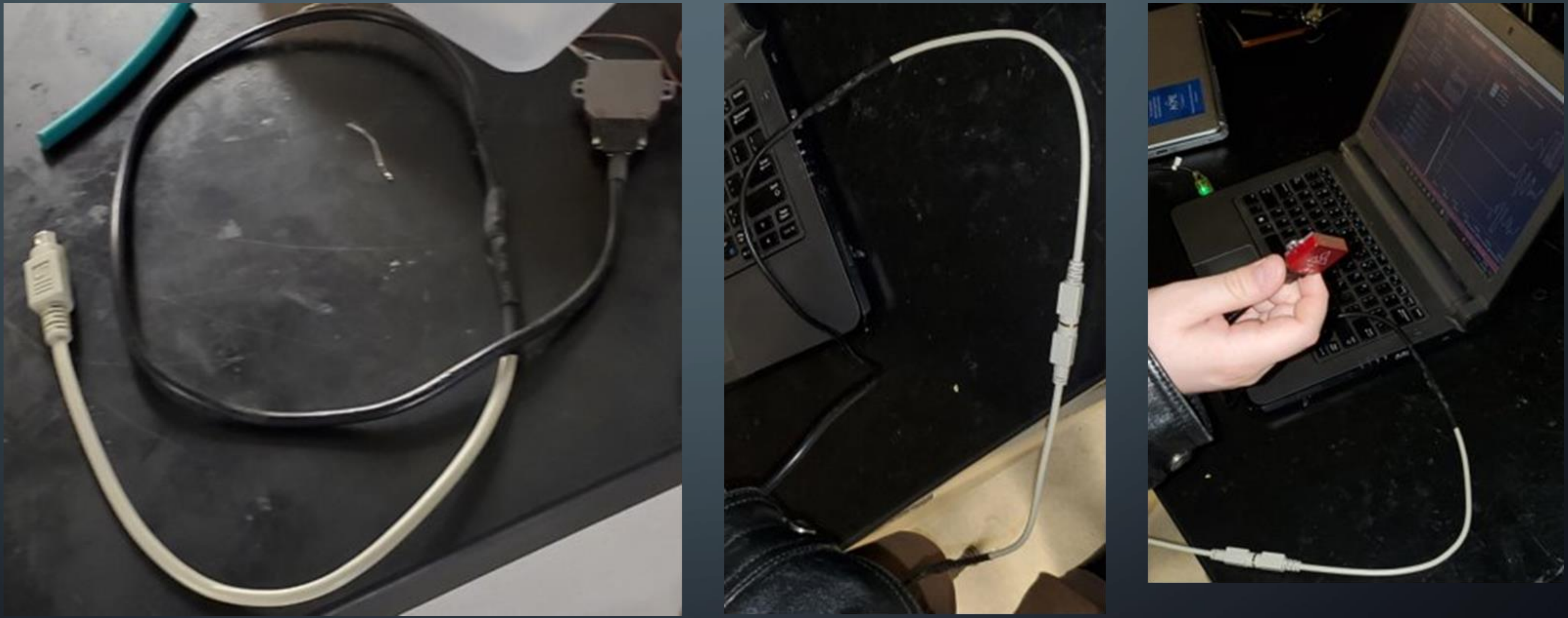
Accel X -02.6060 m/s^2
Accel Y +01.7620 m/s^2
Accel Z -09.6230 m/s^2

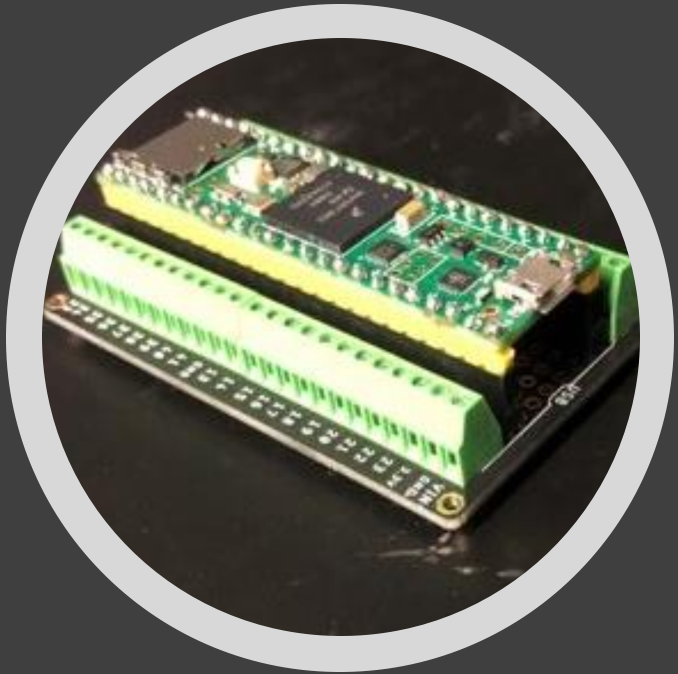
Connected: COM7 @ 230400 bps Model VN-100S-CR Hw: 7 SN: 0100030577 Fw: 2.0.1.0 Ascii Rx: 0 / sec Binary Rx: 51 / sec Msg Tx: 617 Invalid CRC: 0 Bandwidth: 12 % Errors: 1 Mode: Public ☑ Polling: Enabled

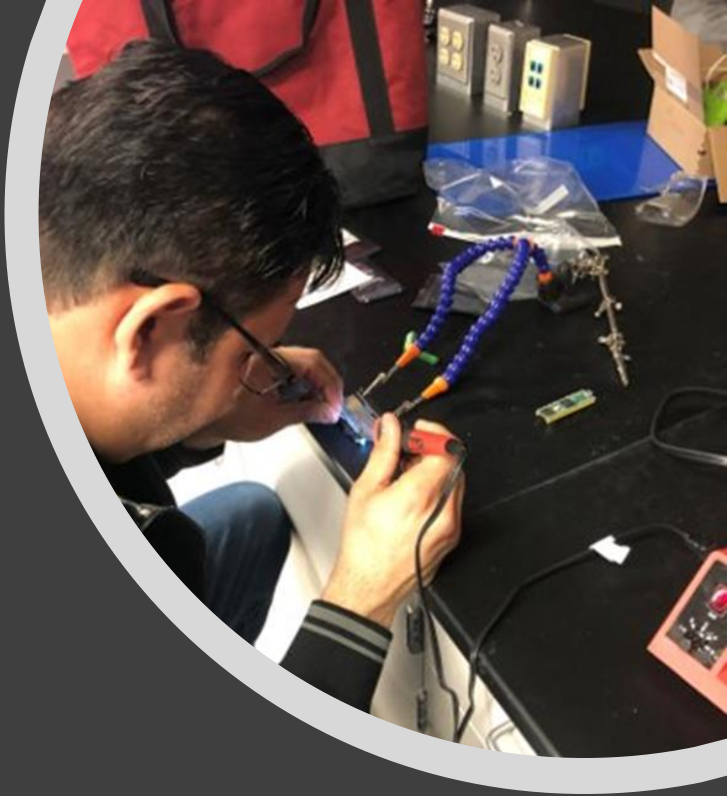
Modifying the VN-100 Connecter cable with an intermediate quick disconnect plug



Finished and Tested Connector Cable

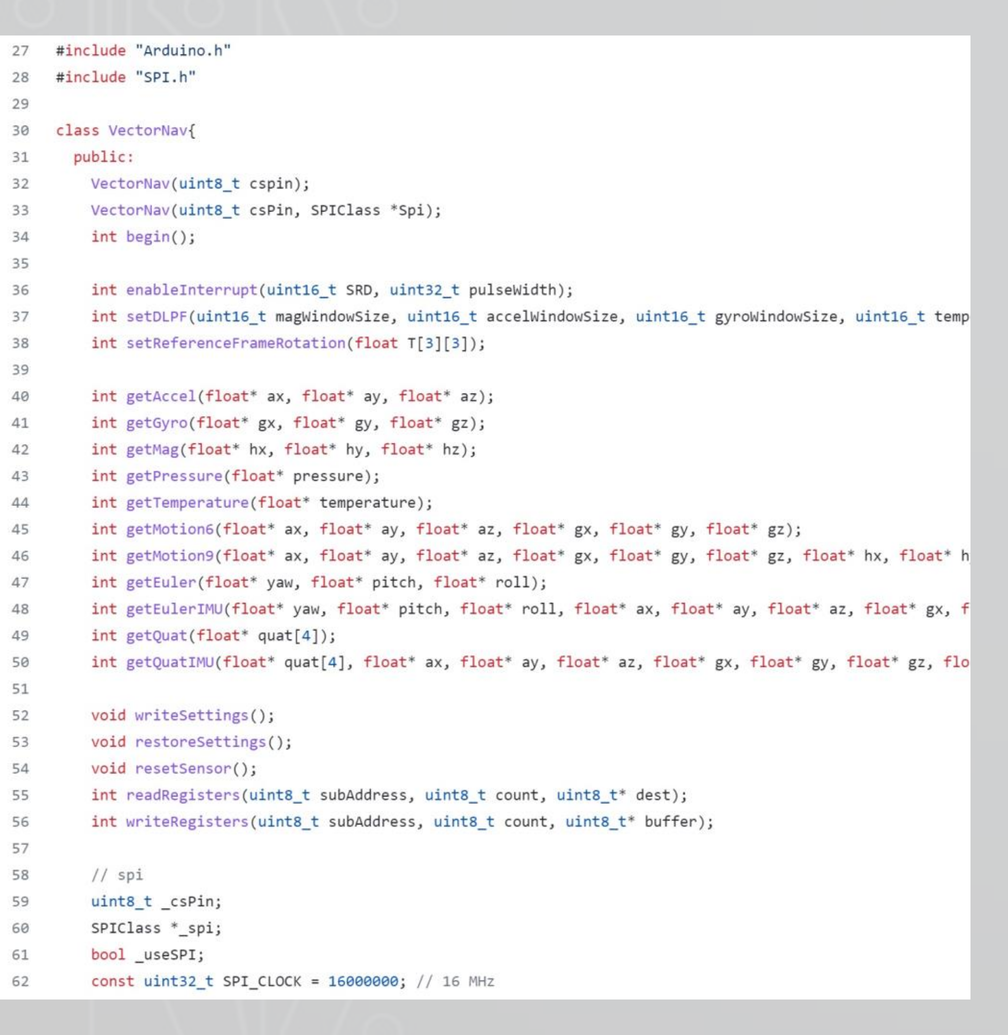






Soldering pins and assembling the breakout board for quick prototyping on the Teensy Microcontroller

CURRENT DIFFICULTIES



```cpp
#include "Arduino.h"
#include "SPI.h"

class VectorNav{
  public:
    VectorNav(uint8_t cspin);
    VectorNav(uint8_t csPin, SPIClass *Spi);
    int begin();

    int enableInterrupt(uint16_t SRD, uint32_t pulseWidth);
    int setDLPF(uint16_t magWindowSize, uint16_t accelWindowSize, uint16_t gyroWindowSize, uint16_t temp
    int setReferenceFrameRotation(float T[3][3]);

    int getAccel(float* ax, float* ay, float* az);
    int getGyro(float* gx, float* gy, float* gz);
    int getMag(float* hx, float* hy, float* hz);
    int getPressure(float* pressure);
    int getTemperature(float* temperature);
    int getMotion6(float* ax, float* ay, float* az, float* gx, float* gy, float* gz);
    int getMotion9(float* ax, float* ay, float* az, float* gx, float* gy, float* gz, float* hx, float* h
    int getEuler(float* yaw, float* pitch, float* roll);
    int getEulerIMU(float* yaw, float* pitch, float* roll, float* ax, float* ay, float* az, float* gx, f
    int getQuat(float* quat[4]);
    int getQuatIMU(float* quat[4], float* ax, float* ay, float* az, float* gx, float* gy, float* gz, flo

    void writeSettings();
    void restoreSettings();
    void resetSensor();
    int readRegisters(uint8_t subAddress, uint8_t count, uint8_t* dest);
    int writeRegisters(uint8_t subAddress, uint8_t count, uint8_t* buffer);

    // spi
    uint8_t _csPin;
    SPIClass *_spi;
    bool _useSPI;
    const uint32_t SPI_CLOCK = 16000000; // 16 MHz
```

- Although we are currently working on a modular class for the VN-100. Most instructions from Vector Nav are specifically written for SPI communication formats and not UART.
- This is because other versions of the VN-100 are capable of SPI, but the "rugged external connector" restricts access to the necessary pins required for communicating through SPI.
- As such we will need to write specific UART reading and parsing methods into the modular class we are currently working on, before we can even begin to communicate with the sensor.

Future Goals

Finish

- Finish writing and testing the UART based class for the VN-100

Create

- Create additional modular classes for existing sensors and control systems

Collaborate

- Collaborate with EE and ME students to begin developing classes for new systems that are being developed (DVL, hydrophones, Sonar)

Deploy and test

- Once a working prototype submarine is completed, deploy and test our code for proper functionality. Rewrite and correct code as necessary.

Sensors to be implemented

Name: VectorNav - 100 IMU

Details:

- Pitch, Yaw, Roll of sub
- C++ library
- UART Connection

Name: Teledyne Pathfinder DVL

Details:

- Tracking Translational Movement
- Depth
- Serial Connection

Name: AS-1Hydrophones

Details:

- Angle to underwater Pinger relative to front of sub
- Reading from Amplifier
- Array of Hydrophones connected to amplifier
- Analog Connection

Name: Blue Robotics Ping Sonar

Details:

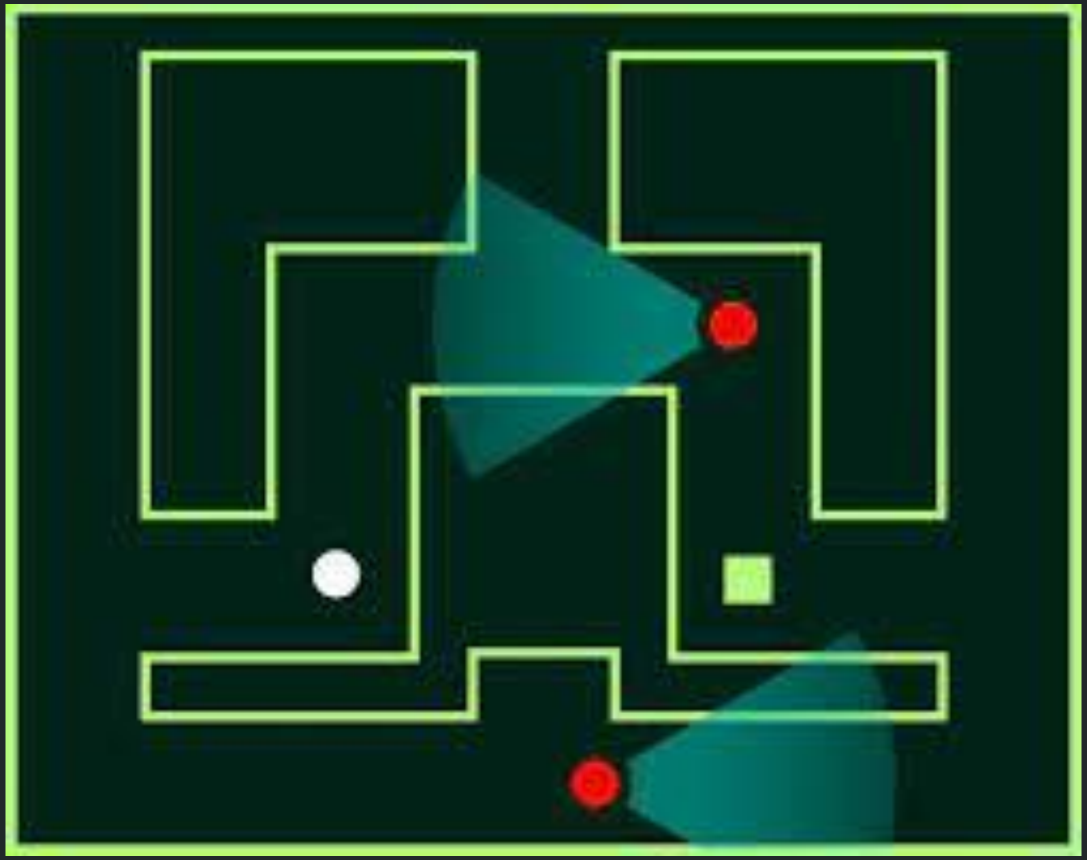
- Large object detection
- Arduino Library / Python
- Analog Connection

Navigation Team

Brandon Cao, Thomas Benson

Underwater S.L.A.M.

- Two Key Parts:
  - Localization
  - Mapping
- Produced strictly through sensor data
- Problem:
  - Noise corruption in data

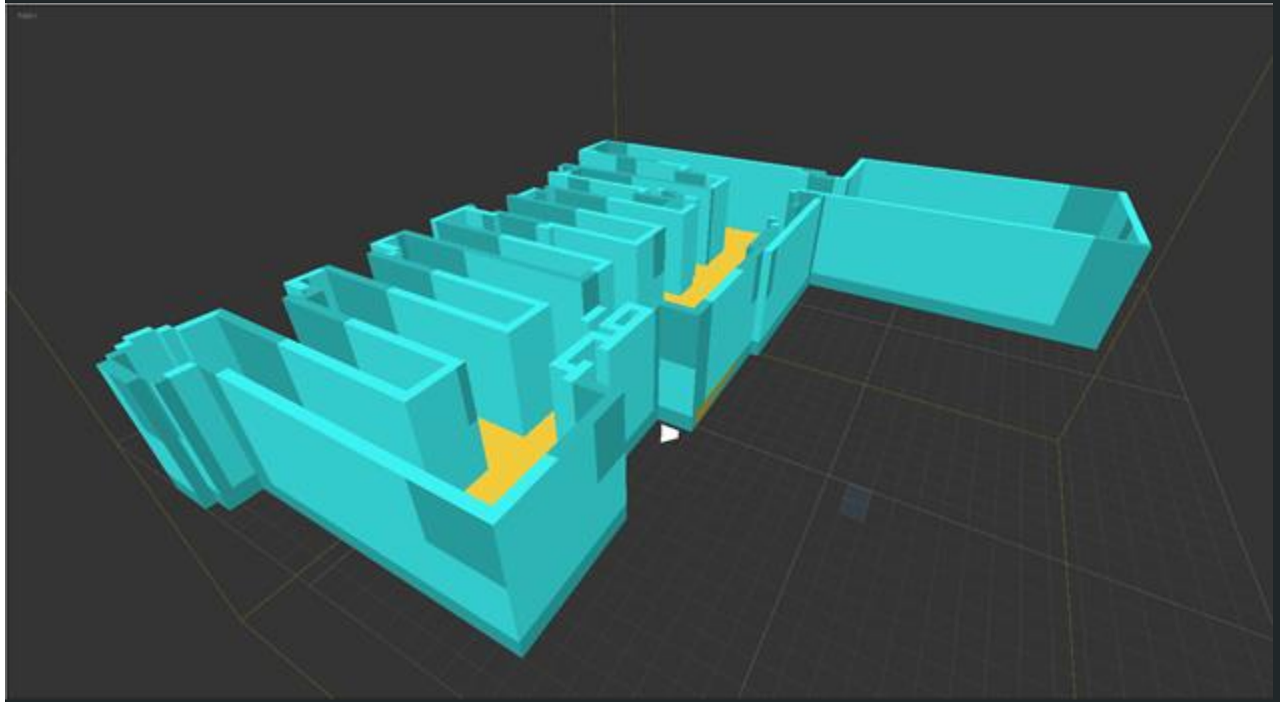


Localization

- Receive data from sensors through ROS nodes/topics.
- Process the data and publish it to the entire system.
- Use accelerometer data to calculate displacement.
- Minimize noise and error propagation.

Simulation Environment

- Need a test environment
- ROS Gazebo
  - Open Source 3D Robotics Simulator
- Arduino + IMU
  - Platform.IO



Remaining Tasks

- More Sensors
    - Barometer
    - Sonar
    - Hydrophones
- Mapping Software
    - Explore environment and build a map using sensors.
- Underwater Testing