FINAL PRESENTATION

TEAM 4 – ROBOSUB

April 15, 2015

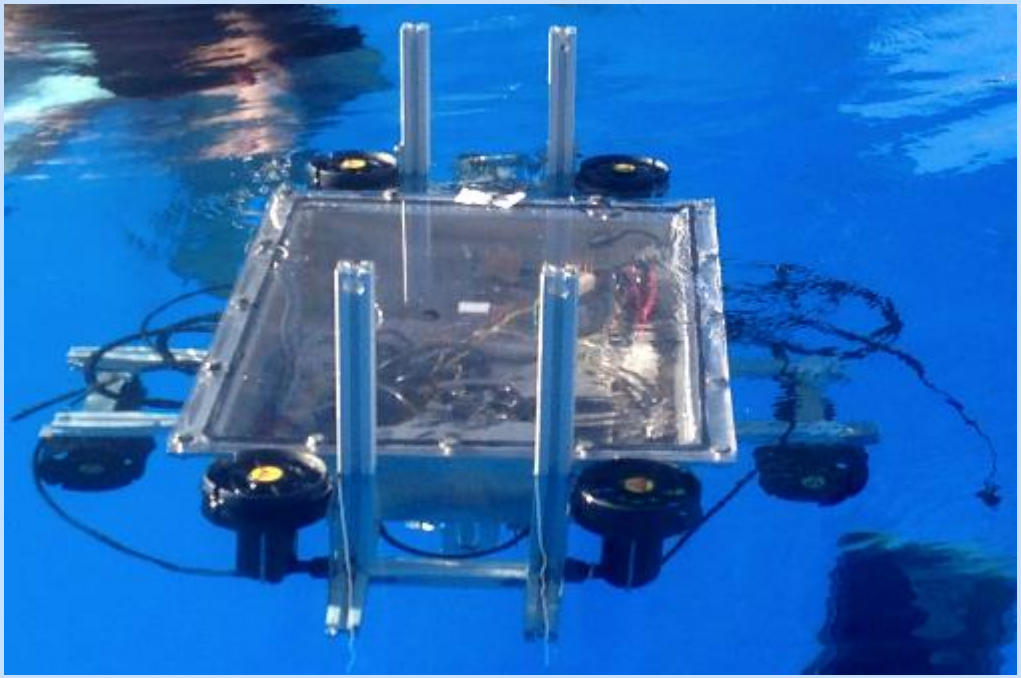
INTRODUCTION

Key Requirements

- Run autonomously without any attachments
- Change depth, direction, and speed
- Pass through and around PVC structures
- Recognize colors

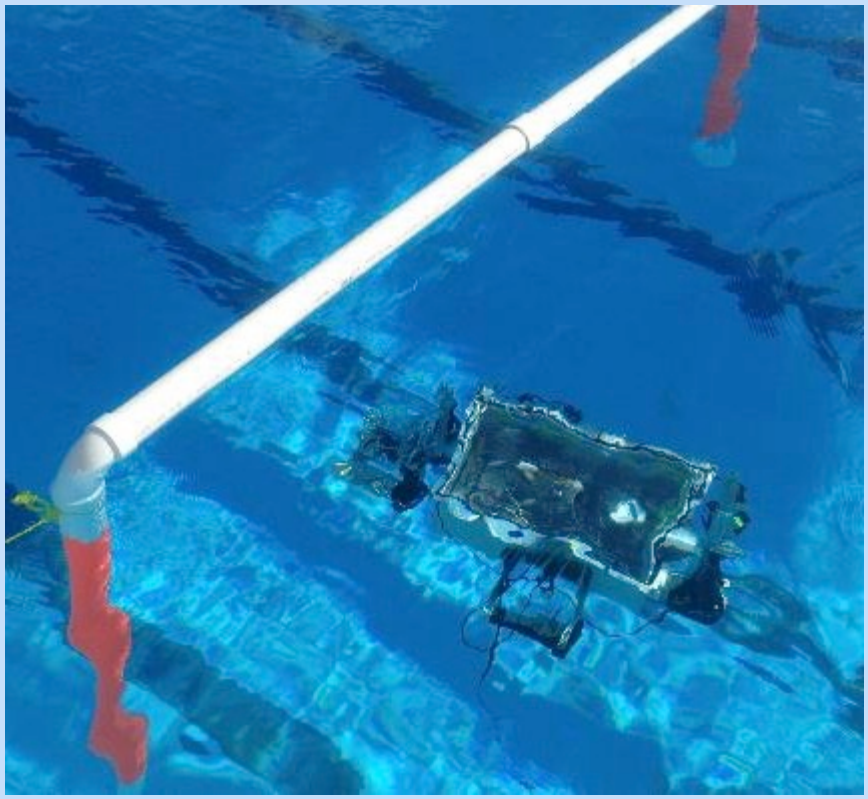
Key Limitations

- Must use last year's sub
- Sub must weigh under 125 lbs



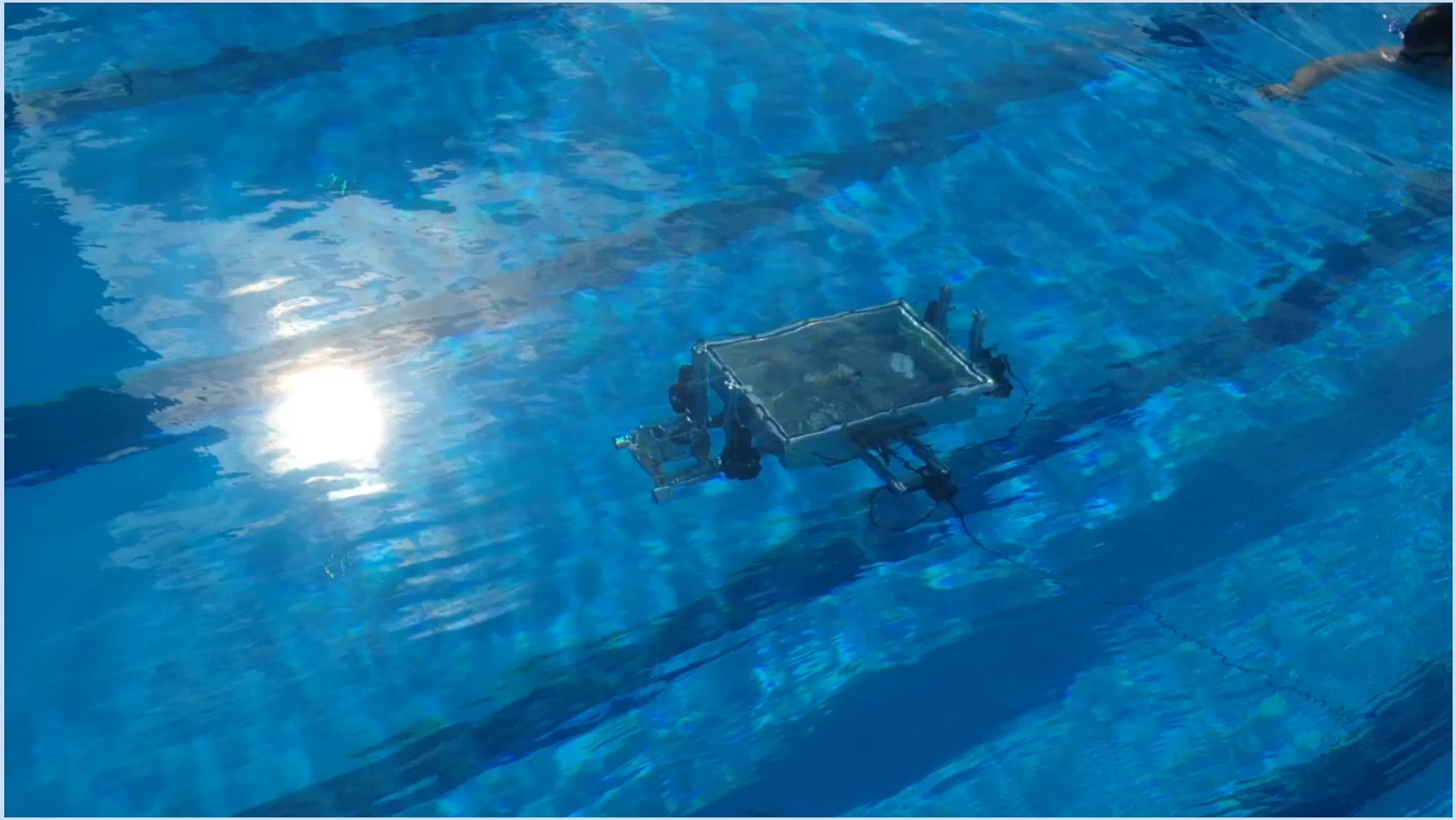
FINAL DESIGN OVERVIEW

- Added consistent basic functionality to the sub
- Implemented a kill switch
- Rotated top thrusters
- Implemented a depth sensor and accurate depth control



- Changed from magnetometer-based movement to gyroscope

MOVEMENT THROUGH GATE - VIDEO



ELLIOT MUDRICK - CONTRIBUTIONS

- Pool coordination
- Scheduling
- Sub maintenance
- Secondary diver
- Yaw control using gyroscope

YAW CONTROLLER

```
while(true)
        {
                pthread_mutex_lock(&imuMutex);
                measuredValue = (myIMU->getGyroZ()) * 0.02;
        pthread_mutex_unlock(&imuMutex);

                error = measuredValue;

                if(error < -0.5f){
                        pthread_mutex_lock(&surgeMutex);

                        /*
                         * If the thruster is off when the Yaw controller kicks in,
                         * we need to boost the initial value so the controller output
                         * has an effect
                         */
                        if(dataBuf[3] < 30 && dataBuf[3] >= 0)
                        {
                                initialThrustTmp = 30;
                                initialThrust = dataBuf[3];
                        }
                        else
                                initialThrust = initialThrustTmp = dataBuf[3];

                        prevError = error;
```

YAW CONTROLLER

```
do
{
                derivative = (measuredValue - prevError) / delayTime;
                prevError = measuredValue;
                output = kP * measuredValue + kD * derivative;
                tmp = convertYawCtrlOutput(output, false);
                dataBuf[3] = initialThrustTmp + tmp;
                std::cout << " mv: " << measuredValue << " A yaw error: " << error << " tmp: " << tmp
                        << "\n";

                usleep(500000);

                pthread_mutex_lock(&imuMutex);
                measuredValue = (myIMU->getGyroZ()) * 0.02;
                pthread_mutex_unlock(&imuMutex);

                 sumOfErrors += measuredValue;

}while(-1 * sumOfErrors < error);

                // return to initial thrust once yaw is corrected
                dataBuf[3] = initialThrust;
                pthread_mutex_unlock(&surgeMutex);
} //end if
```

SAMANTHA CHERBONNEAU - CONTRIBUTIONS

- Assistant Programmer
  - Pitch function - allows for smoother movement and stability of sub
- Web Developer
  - Create and maintain website
- Secretary
  - Minutes, Media, Final Formatting of Submissions

PITCH CONTROL

- Stability is key for proper movement, especially when moving forward
- Sub could tip upwards or downwards

- Needed to determine amount of error from center position
- Then correct only that amount

PITCH FUNCTION - pitchController

- Purpose: maintain the sub's pitch while stationary and in motion
- Run as a thread continuously
- Uses angular velocity value from IMU
- Uses modified PID section of code from previous year

WHILE LOOP

```
1  while(!myIsStopped)  //while true
2  {
3  pthread_mutex_lock(&imuMutex);
4  measuredValue = (myIMU->getGyroY()) * 0.02;  //use angular velocity from gyro
5  pthread_mutex_unlock(&imuMutex);
6
7    if(error < -0.4f) // pitch so front going down
8    {
9      pthread_mutex_lock(&heaveMutex);
10     initialThrust = dataBuf[4];
11     prevError = error;
```

Do ... While Loop

```
35       // return to initial thrust once pitch is corrected
36       dataBuf[4] = initialThrust;
37       dataBuf[5] = initialThrust;
38       pthread_mutex_unlock(&heaveMutex);
39   }
40
41   else if(error > 0.4f)
42   {
43     .
44     .
45     .
46   }
47   sumOfErrors = 0;
48   pthread_testcancel();
49   sleep(1);
50 }
51
```

DO…WHILE LOOP

```
13      do
14      {
15        derivative = (measuredValue - prevError) / delayTime;
16        prevError = measuredValue;
17        output = kP * measuredValue + kD * derivative;
18        tmp = convertPitchCtrlOutput(output, true);
19        std::cout << "A pitch entrance error: " << error << " Correction: "
20          << measuredValue << " out: " << output << " tmp: " << tmp << "\n";
21        dataBuf[4] = initialThrust + tmp;
22        dataBuf[5] = initialThrust + tmp;
23
24        usleep(500000);
25
26        if (measuredValue > 0)
27        sumOfErrors += measuredValue;
28
29        pthread_mutex_lock(&imuMutex);
30        measuredValue = (myIMU->getGyroY()) * 0.02;
31        pthread_mutex_unlock(&imuMutex);
32
33      }while(-1 * sumOfErrors > error);
```

OVERALL FUNCTION

```
while(!myIsStopped)  //while true
{
pthread_mutex_lock(&imuMutex);
measuredValue = (myIMU->getGyroY()) * 0.02;  //use angular velocity from gyro
pthread_mutex_unlock(&imuMutex);

  if(error < -0.4f) // pitch so front going down
  {
    pthread_mutex_lock(&heaveMutex);
    initialThrust = dataBuf[4];
    prevError = error;

    do
    {
      derivative = (measuredValue - prevError) / delayTime;
      prevError = measuredValue;
      output = kP * measuredValue + kD * derivative;
      tmp = convertPitchCtrlOutput(output, true);
      std::cout << "A pitch entrance error: " << error << " Correction: "
        << measuredValue << " out: " << output << " tmp: " << tmp << "\n";
      dataBuf[4] = initialThrust + tmp;
      dataBuf[5] = initialThrust + tmp;

      usleep(500000);

      if (measuredValue > 0)
      sumOfErrors += measuredValue;

      pthread_mutex_lock(&imuMutex);
      measuredValue = (myIMU->getGyroY()) * 0.02;
      pthread_mutex_unlock(&imuMutex);

    }while(-1 * sumOfErrors > error);

      // return to initial thrust once pitch is corrected
      dataBuf[4] = initialThrust;
      dataBuf[5] = initialThrust;
      pthread_mutex_unlock(&heaveMutex);
  }

  else if(error > 0.4f)
  {
    .
    .
    .
  }
  sumOfErrors = 0;
  pthread_testcancel();
  sleep(1);
}
```

WEBSITE

Samantha Cherbonneau

BLOG/MINUTES

FSU ROBOSUB PROJECT 2015

HOME ABOUT DELIVERABLES MEETING MINUTES OUR ROBOSUB

Meeting Minutes 4/6/15

4/6/2015

Attendance: All + Dr. Frank

Recent progress:

- Yaw is now properly adjusted and the sub moves straight and adjusted if shifted by water movement.
- The sub can successfully pass through the validation gate at a specific depth and stay straight while moving. In addition, it can recognize the gate and get a heading based on the center of the gate.
- The bottom camera functions and is able to recognize the orange path. It also can calculate a new heading based on the angle calculated by the direction of the path piece and use of a PID.

Next steps:

- Final demo is Tuesday, April 7 at 4:30 pm at the Morcom Aquatic Center.
- Need to make poster for design fair, write final report and create user manual (written as though talking to next year's team).

Meeting Minutes 3/23/15

3/23/2015

Attendance: All + Dr. Frank

Recent Developments:

Archives

April 2015
March 2015
February 2015
January 2015
December 2014
November 2014
October 2014
September 2014

Samantha Cherbonneau

DENNIS BOYD - CONTRIBUTIONS

- Contributions
  - Lead programmer
  - Testing/debugging
  - Depth control
  - IMU interfacing

DENNIS BOYD - CONTRIBUTIONS

- Lead programmer
  - Had a part in all coding
- Testing/Debugging
  - Primary debugger
  - As the only one with an Ethernet-capable Mac, would always run code
  - Make changes at the pool
  - Test and update those changes
  - Set tuning constants in PIDs
    - With input from group

DENNIS BOYD

- zControllerDepth
  - Primary writer
  - Based on last year's PID design
  - Modified to take an argument (the desired depth) as a pointer to allow dynamic changing of depth without needing to recreate thread
  - Found it would overshoot
    - Added limit to max thrust
  - Wasn't catching itself fast enough
    - Reduce sleep in loop and delay in Arduino (1/10)
  - Tuned constants
    - Realized needed more thrust for more error: increase kP
    - Realized need to correct more quickly: increase kD
  - Play with both until good results

DENNIS BOYD

- PressureSensorArduino.ino
  - Flashed on Arduino UNO
  - Spent a while getting garbage:
    - Baud rate too high
    - Reduced to recommended amount on Arduino tutorial
  - Applied to analog pin 0 (reads as integer 0 to 1023)
  - Convert to volts (max voltage 5 V)
  - Used Bjorn's depth readings to calculate slope:
    - feet/volt
  - Return feet to main routine

DENNIS BOYD

- Getting Values from IMU
  - Previously getting acceleration and magnetometer data only
  - Except this was getting to the main routine as garbage, or seg-faulting (discovered through much debugging).
    - Not given final code/changed over summer?
  - Changed to get just magnetometer readings (all we wanted at the time).
  - Calibrated
  - Learned of IMU Dance:
    - 1. Run ./example
    - 2. Open serial stream viewer
    - 3. Run ./example again
  - Changed to get all 9 values:

IMU CODE

```
/****************************************/
// Used to redirect values to main routine
else if (_mode == ROBOSUB)
{
     if(_input_pos == 36) // we received a full frame
     {
          if (_big_endian())
          {
            _swap_endianess(_input_buf.amg, 9);
          }
          setImuData(_input_buf.amg); // changed from ypra
          _input_pos = 0;
     }
}
```

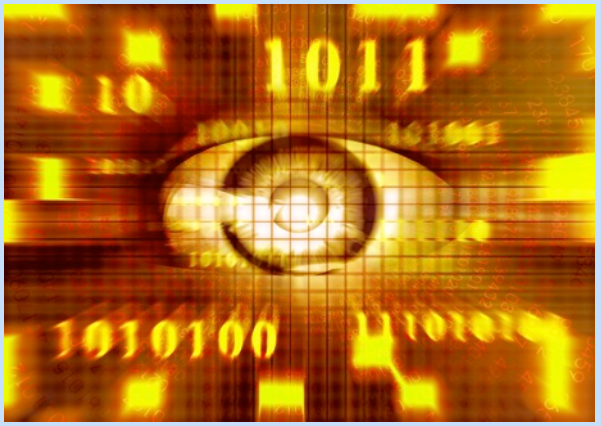
```
// These functions added for RoboSub use
void RazorAHRS::setImuData(const float data[])
{
    // format is {accel-x, accel-y, accel-z; yaw, roll, pitch; gyro-x, gyro-y, gyro-z}
    for(int i = 0; i < 9; i++)
    {
        //std::cout << "In loop at " << i << ": " << data[i] << "\n";
        this->myImuData[i] = data[i];
    }
}
float RazorAHRS::getGyroX()
{
    return this->myImuData[6];
}
float RazorAHRS::getGyroY()
{
    return this->myImuData[7];
}
float RazorAHRS::getGyroZ()
{
    return this->myImuData[8];
}
```

Example use in main routine:

```
pthread_mutex_lock(&imuMutex);
measuredValue = (myIMU->getGyroZ()) * 0.02;
pthread_mutex_unlock(&imuMutex);
```

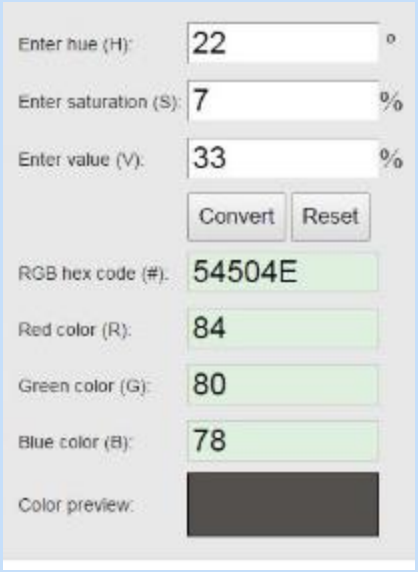
KEVIN MATUNGWA - CONTRIBUTIONS

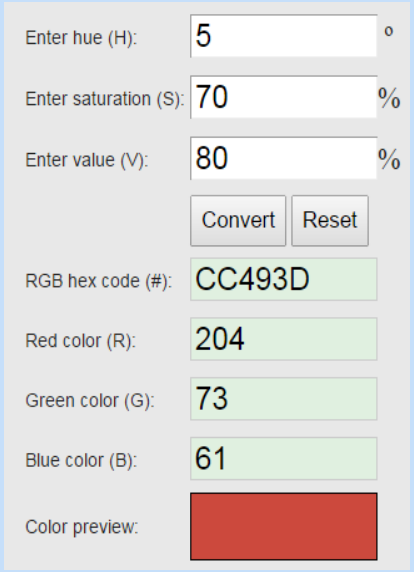
- Programmer and tester
  - Vision system (primary task)
    - Multiple object tracking (front facing cam)
      - Gate identification
      - Calculate object center point
      - Update the database
    - Object Orientation (bottom cam)
      - Detect orange path
      - Analyze the paths orientation
      - Update the database
  - Assisted in IMU firmware to software interfacing
      - Researched and assisted on IMU calibration
      - Decipher IMU firmware for key data used in software calculations
  - Assisted in yaw controller software module
      - Assisted Elliot on yaw PID controller
  - Vision testing

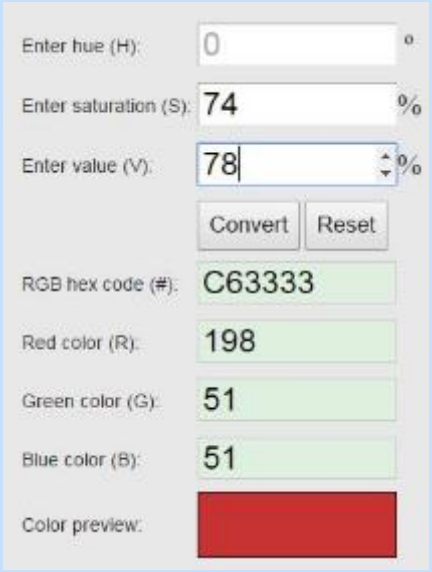


LAB VS POOL OBJECTS COLOR IDENTIFICATION

- Both cameras can now detect orange objects under water
- Due to change in lighting
  - Hue saturation value (HSV) corresponding to orange under water is defined with wider range of error







Min **Max** **Min** **Max**

Pool **Lab**

Kevin Matungwa

COLOR DETECT CODE

```
if(name=="orange"){

    //TODO: use "calibration mode" to find HSV min
    //and HSV max values

    setHSVmin(Scalar(0,191,200));
    setHSVmax(Scalar(22,256,256));

    //BGR value for orange:
    setColor(Scalar(0,165,255));

}
if(name=="orangeUW"){

    // use google to find HSV values from RGB values on the camera
    setHSVmin(Scalar(20,20,80));
    setHSVmax(Scalar(5,180,205));

    //BGR value for orange:
    setColor(Scalar(0,165,255));
}
```

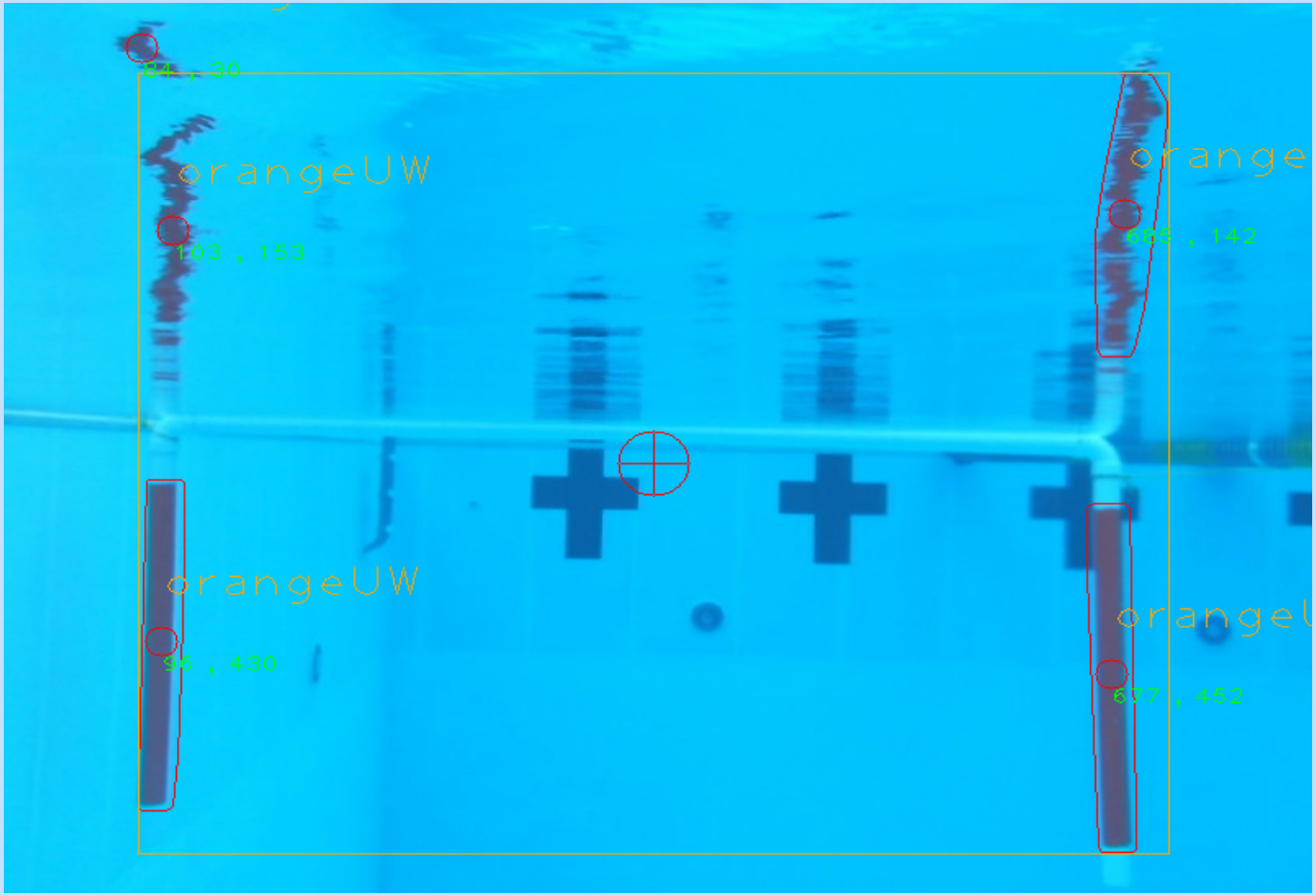
```
//create some temp RoboSubColors objects so that
//we can use their member functions/information
//use orange in lab or orangeUW at the pool
RoboSubColors  orange("orange"), orangeUW("orangeUW");

cvtColor(cameraFeed,HSV,COLOR_BGR2HSV); //convert form BGR to HSV
inRange(HSV,orange.getHSVmin(),orange.getHSVmax(),threshold);//check if within defined ranges of HSV
morphOps(threshold); //create structuring element that will be used to dilate and erode image.

//draw rectangle over determined contour
//determine the center of the rectangle
//update the database
trackFilteredObject(orange,threshold,HSV,feedClone,"1");
```

Kevin Matungwa

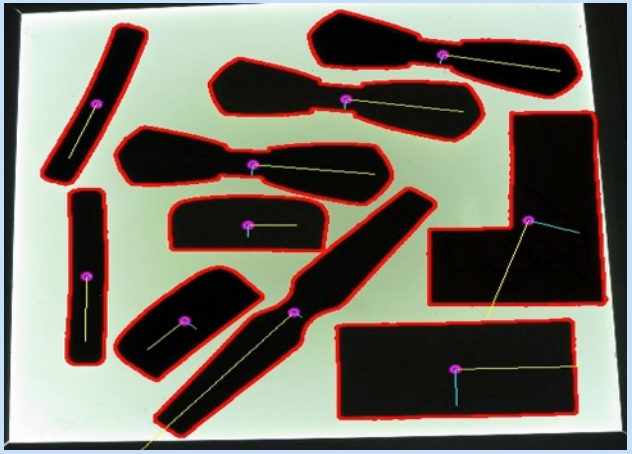
GATE IDENTIFICATION

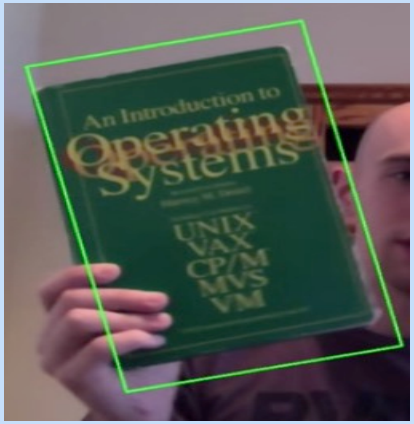


Kevin Matungwa

LINE FOLLOWING

Object Orientation Principal Component Analysis

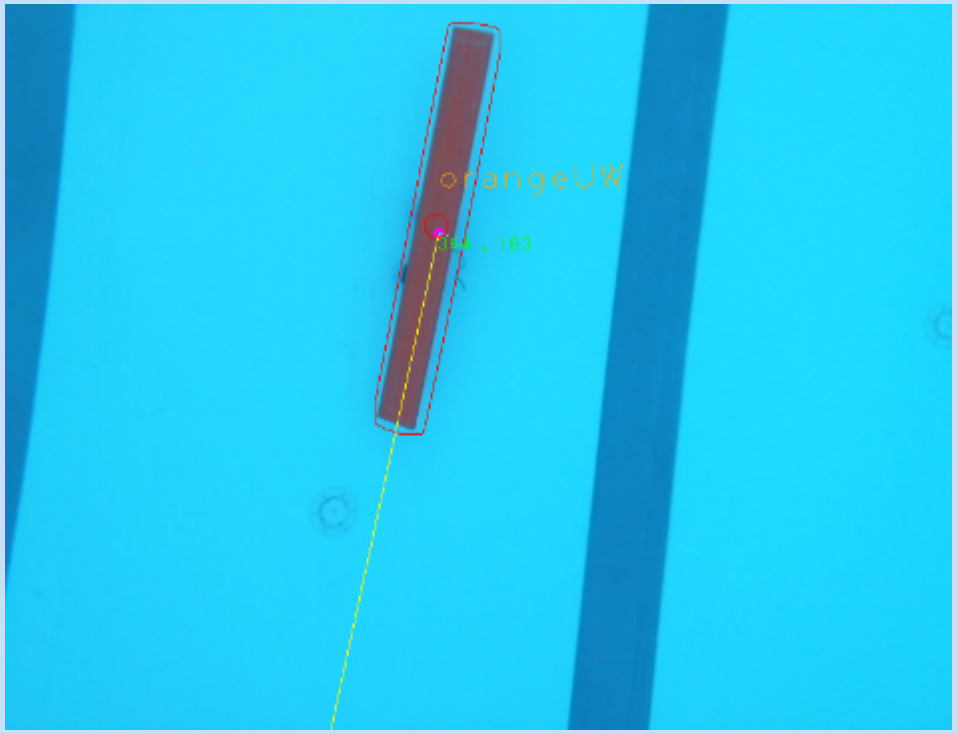




Camshift

Kevin Matungwa

Orientation of orange path detected relative to the Sub



Kevin Matungwa

OBJECT ORIENTATION CODE SNIPPET

- The function getOrientation() return the orientation of the analyzed object relative to the sub in radians

```
for( int i = 0; i< contours.size(); i++ )
  {
    drawContours(feedClone, hull, i, cv::Scalar(0,0,255), 1, 8, vector<Vec4i>(), 0, Point() );

    //getting the angle of the object
    angleY = getOrientation(hull[i], feedClone);

    //type cast angle to sting  to be stored in the database
    string angleYS = static_cast<ostringstream*>( &(ostringstream() << angleY) )->str();

    //update database
    db.open();
    if (angleYS != "0")
    db.updateQuery("UPDATE Tasks_List SET task_detected = 1, angle_y = "+angleYS+" WHERE task_id = "+task);
    db.close();
  }
```

Kevin Matungwa

BJORN CAMPBELL - CONTRIBUTIONS

- Electrical Components
  - Depth sensor
  - Kill switch
  - Motor controller reconfiguration
  - Ethernet cable
- Gate and Path
- Treasurer
- Sub Maintenance
- Pool Diver

DESIGN CONTRIBUTIONS

- Depth Sensor
- Kill Switch
  - Specifications
    - Budget
    - Performance
    - Attachment
  - Testing at all stages of attachment
  - Connection to AUV

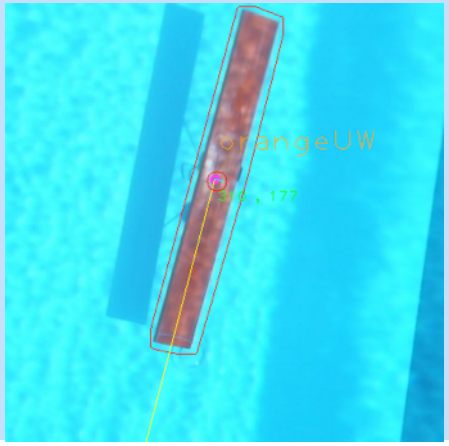
DESIGN CONTRIBUTIONS

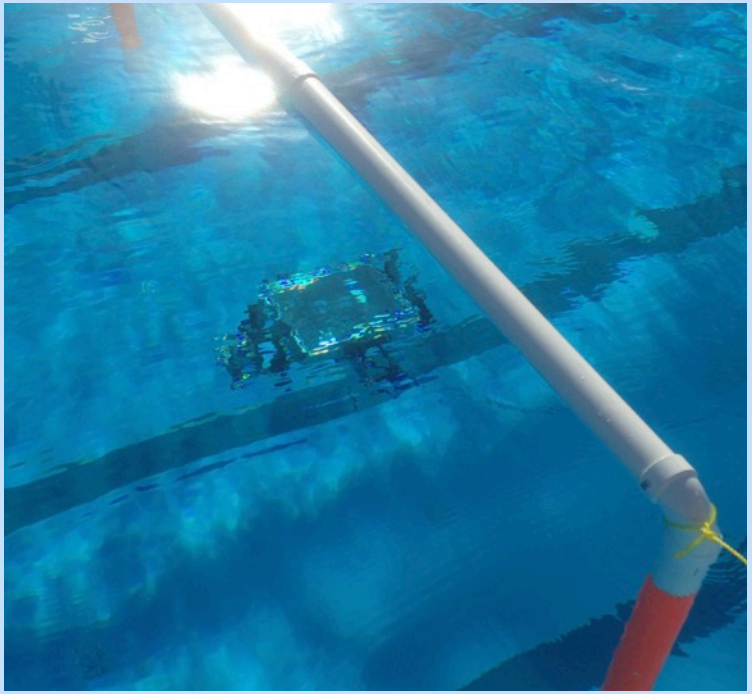
- Motor Controller Reconfiguration
  - Arduino Mega and 2 motor controller burned out
  - Decision
    - Order new parts VS redesign connections
  - Reconfigured Motor Controller Layout

- Ethernet Cable
  - Old 100ft cable not properly waterproofed
  - 100ft cable and Seacon connection damaged
  - Decision
    - Resuming testing main priority
    - New 100ft cable failed to work
    - Required reconfiguring of Seacon connections
    - Spliced two 25ft cables together

OTHER RESPONSIBILITIES

- Budget
- Diver
- Maintenance of Sub
  - Checking connections
  - Keeping batteries charged
  - Waterproofing
  - Transportation of Sub to pool
- Design of Gate and Path





FINAL BUDGET

D. Expense	Total
3” Diameter x 10’ Long PVC	$29.96
90 Degree Elbows 3” PVC	$5.14
3” Clean Out Tee PVC	$1.37
Blaze Orange Duck Tape	$6.74
Heat Gun	$14.99
Heat Shrink Tubing Assorted Sized	$4.99
Liquid Electrical Tape	$10.99
Heat Shrink Tubing Large Size	$16.99
Push Button Switch	$6.49
Hex Keys 10pc	$5.35
Cable Tie	$6.96
Zotac Wall Charger	$11.63
1”x6” – 8 FT Weather Shield Wood	$10.74
Hallow Braid Poly Rope (1/4” x50’)	$5.60
2” Diameter by 6’ Long PVC	$16.44
90 Degree Elbows 2” PVC	$1.66
2” Clean Out Tee PVC	$3.26
PVC Glue	$4.87
2” PVC Caps	$3.28
16/19 V Ah LI-Ion Universal External Battery	$74.76
Depth Sensor+Shipping	$371.24
Ethernet Cable	$25.80
Expenses Subtotal (including tax)	$639.25
Allotted Budget	$750.00

FINAL PROJECT STATUS

SUMMARY OF TESTS

Test	Status
Power Systems	Pass
Ascend and Submerge Sub Using Thrusters	Pass
Maintain Depth	Pass
Underwater Object Recognition	Pass
Gate Traversal	Pass
Line Following	N/A
Gate and Path Combined Traversal	N/A
Waterproofing	Pass
Initial Depth Sensor Test	Pass
Gate Construction	Pass
Kill switch	Pass

REQUIREMENTS MET

- ✓ Weight Under 125 lbs
- ✓ Can run autonomously
- ✓ Change depth, direction, and speed
- ✓ Can maintain depth
- ✓ Can pass through validation gate
- ✓ Can recognize orange objects using front and bottom cameras

DELIVERABLES

- The sub
- All associated files containing code
- User Guide

WHERE TO GO FROM HERE

NEXT STEPS

- Complete line following
- Attach functioning gripper and torpedoes
- Utilize color recognition for other tasks

QUESTIONS