# Autonomous Control for Remotely Operated Vehicles

Yingxue Li

Advisor Christopher Clark

Department of Computer Science

Spring Independent Work

May 8, 2012

# Acknowledgements

First and foremost, a big thank you to Professor Clark for his help and guidance throughout this year both in Malta and at Princeton, for being responsible for a group of college students and organizing the trip, and for always willing to help whenever we needed him. I would also like to thank the other students I worked with for this project, both at Princeton University and at California Polytechnic State University. Even though I didn't have the chance to work with them too much, I learned a lot from their experiences and through our conversations. Finally I would like to offer my gratitude to everyone who supported this trip, the National Science Foundation, the Princeton University School of Engineering, and also Dr. Timmy Gabin for guiding us in Malta. This project would not have been possible without everyone's help and support.

**Abstract**

The relatively new field of robotics has many practical applications, including mapping ancient cisterns in Malta. These field experiments, led by a team of researchers from the California Polytechnic State University help archaeologists better understand the structures the cisterns are built in and their surroundings. Because these underwater tunnels have such low visibility, it is difficult for a human pilot to effectively navigate them with only video and sonar. Thus, the goal of this paper is to propose and implement an autonomous control scheme for a underwater robot, such as a Remotely Operated Vehicle(ROV), so it can navigate through a cistern and take enough sonar scans so a complete map can be created. For this project, a stable controller was proposed and implemented for depth and horizontal control for an ROV. Although this method did not fully work for deployment due to an irregularity in one of the ROV thrusters, this could serve as a basis for further development on more complex paths.

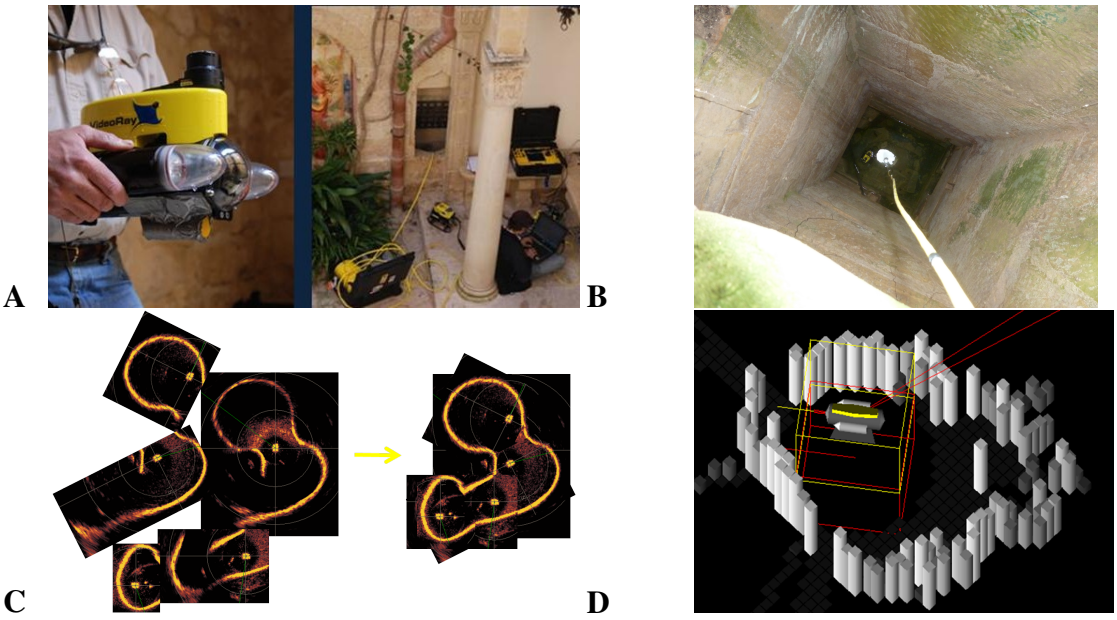



Figure 1: Images taken from [3]. (A) Image of the ROV (B) ROV being lowered into a cistern entrance. (C) Individual sonar scans on the left and complete mosaic on the right. (D) Example of probabilistic occupancy grid map for a different cistern generated from the SLAM algorithm.

# 1 Introduction

Cisterns are underground chambers built to store and provide water for a building and thus are important features to consider while studying such structures. A team of researchers from California Polytechnic State University, and this year Princeton University, in cooperation with Maltese archaeologists, used Remotely Operated Vehicles(ROV's) to study the interiors of these cisterns as well as attempt to map each one. The ROV used is a small underwater robot with a tether connection back to the operator (shown in Figure 1 A). Control commands are sent to the robot through the tether using the control box, while the sensor measurements are sent back to the control box or computer. The cisterns being studied are mainly located in private residences, fortresses, churches, and ancient temples. The Malta Cistern Exploration Project serves both to provide information about the cisterns to the private owners as well as the archaeologists studying them.

The current method for mapping these cisterns is to lower the ROV into the well-like entrance

to the cistern and manually drive around each chamber with the control box, taking sonar readings every half minute or so and documenting direction of robot, position in the cistern, and a hand drawn approximate map of the cistern in a log book. Distinct features such as corners or passageways were prioritized, while still ensuring complete coverage of the system, when deciding the location of the next sonar scan. After the ROV is evacuated from the cistern, an initial map is created by mosaicing together the sonar scans with respect to the additional information recorded in the log book.

To provide statistically sound maps, a SLAM algorithm was used to map in real time while the ROV is in the cistern (Figure 1 D). The SLAM algorithm uses a particle filter to localize the ROV and represents the cistern as an occupancy grid, where each square in the grid represents the possibility of the presence of a wall. As the robot navigates manually through the cistern, the value of each cell is calculated and represented in grayscale [8]. However, this method was not utilized in Malta during the Spring 2012 visit.

Although the resulting map created using the mosaicing method(shown above in Figure 1 C) appears clean and accurate, the individual sonar scans themselves are confusing, at times even with the added information from the log books. In addition, for both of the previously mentioned mapping techniques, visibility in the cisterns is often very poor, increasing the likelihood of a human error while piloting the ROV. Disorientation is a major issue while piloting, especially for an inexperienced operator. Even with the hand-drawn maps from the log books, the pilot needs to be extremely careful while navigating or attempting to free the ROV from obstacles on the floor of the cistern because many passageways and areas are roughly symmetrical, making it very difficult to re-localize once the pilot is lost.

Due to these main reasons, the focus of this project is to design and implement an autonomous

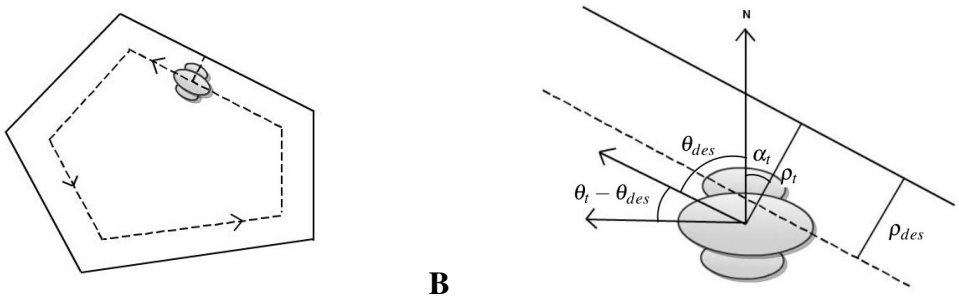
control system for the ROV to navigate without a pilot through a simple cistern. This is accomplished by implementing a wall-following method using proportional control to adjust the direction and depth of the ROV as it explores the cistern.

# 2 Background

Although there has been an abundance of research and implementation of wall following in land robots [1, 2], and even on the subject of sonar mapping and navigation [4], the number of instances of this method being used in underwater robots, such as an ROV is far fewer due to the limitations in underwater localization and sensor resolution. Most of the wall following methods mentioned utilize infrared sensors and ultrasonic sensors such as sonar to detect information about the surrounding area, but while underwater, there is a limited number of sensors that can be used, and the information gathered by such sensors are not always accurate due to disturbances in the water and interference from floating debris.

There has been previous work done with the VideoRay Pro 3, where researchers were successful in implementing a stable, autonomous controller for the ROV to track a desired trajectory underwater [7]. However, the original goal of this project was to implement an autonomous controller based on closest wall distance and angle received from a wall finding function, researched by a fellow student Anna Simpson, the wall following function in [7] could not be used because it required a more sophisticated knowledge of the surrounding when applied to autonomous navigation of a system of chambers.

There have also been instances of researchers implementing an autonomous trajectory tracking systems with collision detection for Autonomous Underwater Vehicles (AUV) such as the REMUS




**A** **B**

Figure 2: Top down view of ROV in mapping environment (A) Dotted line shows desired path of ROV while travelling with set distance away from wall. (B) Close up view of ROV tracking wall. $\alpha$ is compass angle to nearest wall, $\theta_{des}$ desired compass angle, and $\theta_t - \theta_{des}$ desired angle with respect to the ROV. The desired distance and current distance are represented by $\rho_{des}$ and $\rho_t$ respectively.

for Naval mission purposes [5]. These methods, too, are not possible with only wall-following and proportional control, but could prove helpful in improving and making adjustments to the simple method presented in this project.

# 3 Problem Definition

The purpose of this project is to design an autonomous tracking control system for an underwater robot(e.g. a Remotely Operated Vehicle, or ROV for short) so it is able to autonomously navigate a variety of cisterns in Malta with different shapes and structures. Since there is low visibility in some of the cisterns, it is very difficult for a human pilot to both navigate and localize the robot within each room. An autonomous method of exploring these cisterns will serve to both make the pilot's job easier and to decrease the human error within the whole process. This method will also enable 2D map construction using sonar scanning and ensure complete coverage of the cistern structure for mapping purposes.

To do this, the robot must follow the walls of the structure at some nominal distance, $\rho_{des}$ in Figure 2, to circumnavigate the entire cistern. In the diagram, $\rho_t$ represents the current distance

from the wall and $\theta_t$ represents the compass angle the robot is facing. The goal is to design two controllers, Equations 1 and 2, to calculate control values to send to the robot for depth control and horizontal movement control while minimizing the error functions, Equations 4, 5, and 6. Here, $\theta_{des}$ is $\alpha - \frac{\pi}{2}$ because we want to travel along a path parallel to the closest wall on the right. In the depth control equation, the desired depth thruster value is simply calculated from the current depth, while the desired values for the port and starboard thrusters are calculated from current distance to wall, current compass angle, and current compass angle to closest wall.

$$[U_d] = f_d(z_t) \tag{1}$$

$$[U_p, U_s] = f_{p,s}(\rho_t, \theta_t, \alpha_t) \tag{2}$$

$$\theta_{des} = \alpha - fracpi2 \tag{3}$$

$$e_{d,t} = z_{des} - z_t \tag{4}$$

$$e_{\rho,t} = \rho_{des} - \rho_t \tag{5}$$

$$e_{\theta,t} = \theta_{des} - \theta_t \tag{6}$$

In this case, we set $v$ equal to some $v_{nom}$ for simplicity. We also want to minimize $e_{d,t}$, $e_{\rho,t}$, and $e_{\theta,t}$ for $t = 0 : t_{max}$ and such that $v_{nom} \times t_{max}$ represents the length of the entire inner perimeter of the cistern, which is our desired path to follow.

The ROV is dependent upon a local control scheme because it is not equipped with a Global Positioning System or an underwater acoustic positioning system. Since the only positioning information the ROV receives is from the depth sensor and the sonar, the robot must localize itself

based on real-time features detected by the sonar. This method is not optimal in some situations, such as in a room with a very large diameter, in which case the sonar readings will not be able to reach the center of the room, leaving a hole in the final map. Issues also surface when the robot is presented with a narrow tunnel while following the wall. It is unclear if the robot should enter the tunnel, where it might run into obstacles or get dislodged inside the tunnel, or if the robot should ignore the tunnel altogether, possibly missing vital information or another section of the structure. Although these concerns would be crucial in the field while exploring ancient cisterns, for this project, we will mainly focus on testing in small, simple spaces to test the autonomous control method.

# 4 Controller Design

In this project, we will implement proportional controllers for both the depth and horizontal control, with proportional constant $K$ for each variable. For the depth controller, we simply want to send the depth thruster $\dot{z} = K_z(z_{des} - z_t)$ to drive $e_{d,t}$ to 0. This will keep the robot at a constant depth, $z_{des}$. However, for horizontal wall-following, the controls need to be much more complicated.

Since we want to design a controller that will follow a wall to the right, we want the ROV to move based on both the angle to the closest wall and the distance from it. To handle turns correctly, it will also need to recognize when to follow a new wall. Turning left is trivial because as the ROV approaches the new wall on the left, there will be a point where the closest distance to the new wall will be less than the closest distance to the old wall. Thus, it will turn to follow this new wall automatically. However, detecting a new wall to the right is not so simple. As the previous wall to

the right ends, the ROV might not register the next wall it needs to follow as the closest wall and will keep heading straight towards the inside of the chamber. To correct this, we propose that if the closest distance to a wall is greater than a certain threshold, $\rho_{max}$, the controller sends the signal to go straight at an angle of $\frac{\pi}{4}$. This will cause the robot to start turning in a circle with a small radius. The exact angle and speed should be adjusted based on $\rho_{des}$ and $\rho_{max}$ to avoid collision with the wall or over-turning. If there is indeed a new wall connecting to the right, the robot will encounter it and register it as the new closest wall and follow it. Since we are assuming the outerwalls of the cistern are continous, this method will always find a new wall to follow.

For the wall-following section, we want to calculate control values, $\dot{\theta}$ and $\dot{\rho}$, such that $e_{theta,t}$ and $e_{\rho,t}$ are driven to 0. In this report, we propose the following functions as a control scheme:

$$\dot{\theta} = -\omega = -K_\theta e_{\theta,t} - K_\rho e_{\rho,t} \tag{7}$$

$$\dot{\rho} = vsin(e_{\theta,t}) \approx ve_{\theta,t}, \text{ for small } \theta \tag{8}$$

From the first equation, we see that $\dot{\theta}$ takes into account both the angle difference to the desired angle and the distance to desired distance. Note that $\dot{\theta}$ is also equal to negative of the angular velocity, $\omega$, because while the compass angles range from 0 to $2\pi$ clockwise, the angles passed to the ROV thruster control are from $-\pi$ to $\pi$ counterclockwise.

To prove the stability of our controller, we use the method from [6]:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\rho} \end{bmatrix} = \begin{bmatrix} -K_\theta & -K_\rho \\ v & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \rho \end{bmatrix} \tag{9}$$

First, the control equations can be written as the matrix above. To show that the controller is stable, we need to find the $K_\theta$ and $K_\rho$'s such that the eigenvalues of the constant term matrix in the equation above are negative.

$$det(A - \lambda I) = (-K_\theta - \lambda)(-\lambda) + K_\rho v$$

$$\Rightarrow \lambda = \frac{-K_\theta \pm \sqrt{K_\theta^2 - 4K_\rho v}}{2}$$

Since $\lambda < 0$:

$$K_\theta^2 - 4K_\rho v < K_\theta^2$$

$$\Rightarrow K_\theta > 0 \text{ and } K_\rho > 0$$

Thus, we can see that the controller proposed above will be stable for all $K_\theta > 0$ and $K_\rho > 0$, meaning that it will drive the error terms $e_{d,t}$, $e_{\theta,t}$, and $e_{\rho,t}$ to 0, forcing the robot to drive along the outerwall with velocity $v_{nom}$.

# 5 Equipment

The ROV used in this project was a VideoRay Pro 3 ROV with dimensions 30.5 x 22.5 x 21 cm and weight 3.8 kgs (shown in Figure 3). The maximum speed is 2.6 knots, with a depth rating of

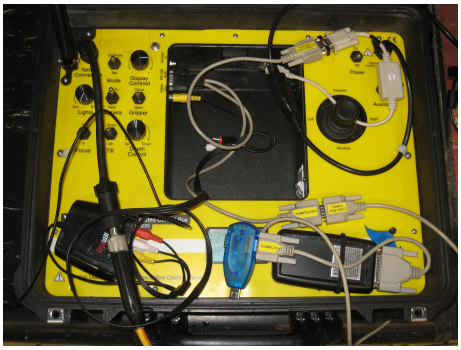
A  B 

Figure 3: (A) ROV with sonar mounted and tether at the back. (B) Control box for the ROV with video, sonar, and actuator control connections to computer.

152 m. The robot itself is equipped with both a front and back camera and two halogen lights in the front of each of its two horizontal thrusters. There is also a vertical thruster at the top to control depth. In the figure, there is a SeaSprite Scanning Sonar with range 2 - 75 meters attached in the front.

Although the ROV can be manually controlled from the control box with a built-in joystick and other various dials, we had to instead control the robot with a laptop in order to record sonar and video data as well as implement autonomous control with the C++ code. This involved many connections and wires. First, we used a Gigaware A/V to USB converter to import the video from the ROV front and back cameras. Several serial to USB were required to import the actuator control and sonar data to the laptop. There was no issue with the converter for thruster control, but for sonar data, we experienced buffering and freezing issues with the previous converters and had to contact VideoRay for ones with no freezing: an RS485 to RS232 converter and an EasySYNC ES-U-1002-A RS232 to USB converter. The exact model for the RS232 to USB converter is recommended because it is guaranteed to have no buffering issues.

All of the coding was done in Microsoft Visual Studios 2010. An $autonomousControl()$ function was added to the previous ROV code, of which most was provided by VideoRay; however,

Figure 4: Image of ROV deployment in DeNunzio Pool while running horizontal autonomous control.

due to the fact that most of the code was generated automatically, some of the structure and methods/variables were very confusing and took a lot of time to decipher. Also, due to the testing restrictions of an underwater robot and hardware issues like the serial adapter mentioned above, we were only able to fully test the autonomous control method in the Princeton University DeNunzio Pool, which meant that testing times were tightly restricted by pool availability and class times. For these reasons, there were not as many opportunities to test the code for this project.

# 6 Experiments and Data

For the purpose of verifying the proposed controllers, several experiments were designed to test the depth controller and the horizontal controller, both by varying $\alpha$ and $\rho$, the results of which are shown in the following graphs. For all three graphs, $K_d = 50.0$, $K_\theta = 20.0$, and $K_\rho = 40.0$. These values were determined experimentally to ensure fast convergence, with little overcorrection. All three graphs were graphed in Gnuplot with Bezier smoothing.

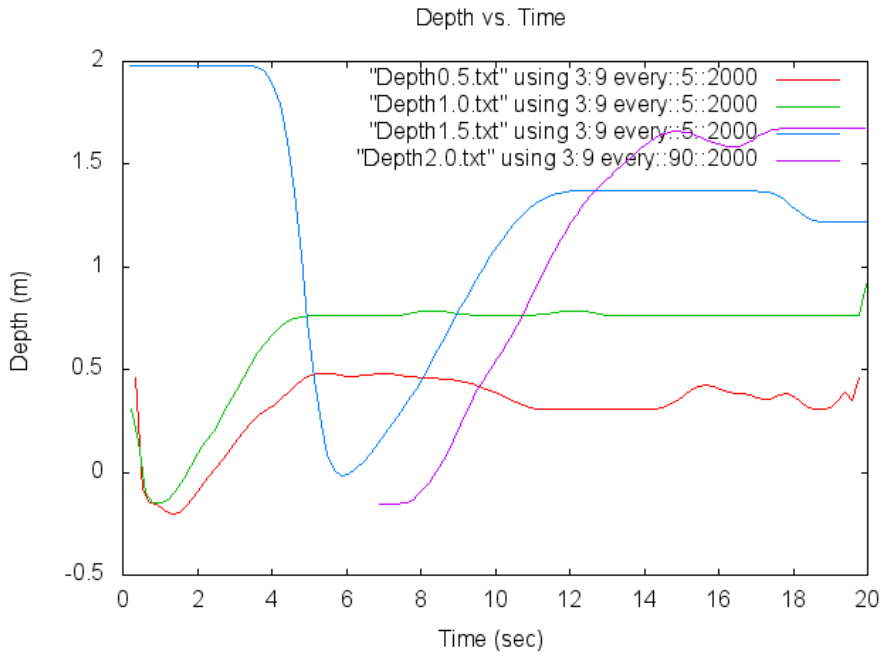



Figure 5: Graph of ROV vertical motion at several depths (0.5m, 1.0m, 1.5m, and 2.0m). In all cases, the ROV held at desired depth after several seconds although fluctuated slightly within the depth sensor resolution (0.5m).

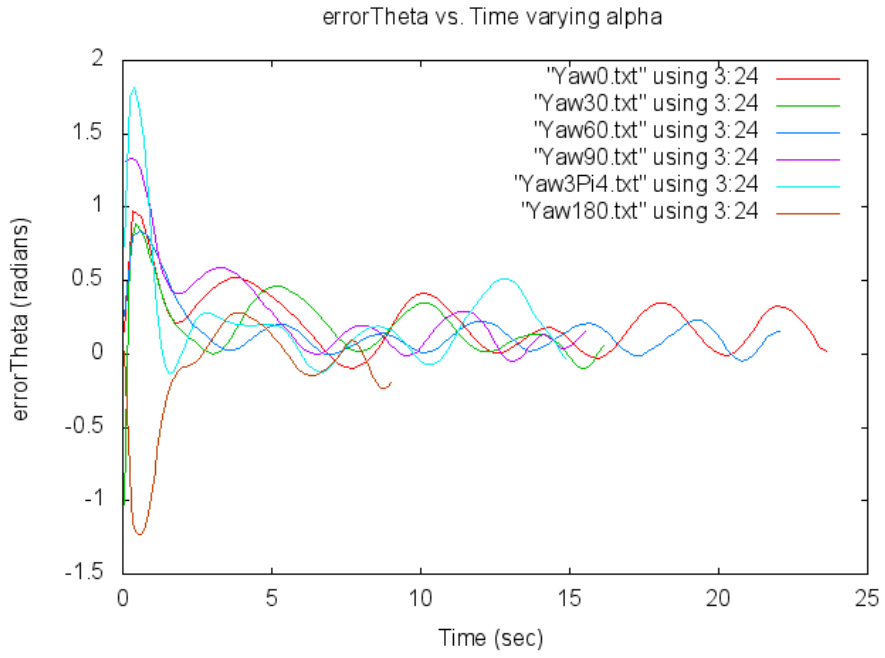



Figure 6: Graph of horizontal motion at many $\alpha$'s. For these experiments, $\rho_t = \rho_{des}$ In all cases, ROV tracked the correct $\theta_{des}$ and continued with a forward motion. Note the oscillations in the forward motion, to be discussed in the Discussion section.

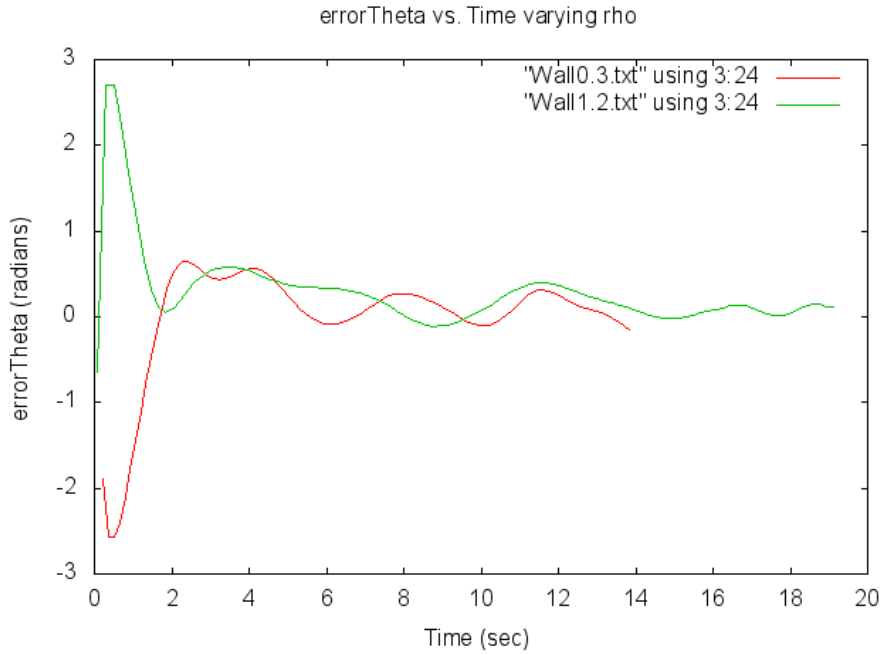



Figure 7: Graph of horizontal motion at many $\rho_t$'s. For these experiments, $\theta_t = \theta_{des}$ In all cases, ROV tracked the correct $\theta_{des}$ and continued with a forward motion. Note the oscillations in the forward motion, to be discussed in the Discussion section.

# 7 Discussion

For the depth controller, we can see from Figure 5 that the ROV depth sensor reading, $z_t$ converges to around 0.2m smaller than $z_{des}$ just a few seconds after starting. The reading then fluctuates around that value (fluctuations cannot be seen as clearly in the figure because of the Bezier smoothing) even though the control box displays the correct $z_{des}$ and the robot is motionless in the water. This can be mostly attributed to the low resolution of the depth sensor on the ROV. The resolution is around $0.5m$, and since all of the random fluctuations in $z_t$ are within $0.5m$ of $z_{des}$, we can attribute most of the error to the sensor and say that $e_{d,t}$ mostly went to 0 as $t$ increased.

As for the horizontal controller, we can also see from Figures 6 and 7 that the ROV converged to $\theta_{des}$ in only a few seconds and attempted to move in a straight line from there. However, in all datasets, there was an oscillation of around 0.3m once the robot started to move in a straight line.

As the ROV is also moving in a forward then backwards motion, this was not due to the error in the compass measurements. After this forward and backward motion was first noticed, additional tests were performed to test the thrusters. It was observed that when both thrusters were given the same speed and a forward direction, the robot travelled in a circle to the left. This could be due to the fact that the left thruster on the robot is spinning as fast as the right one.

The autonomous control method sends signals for the robot to move straight at a set speed, in this case at a thruster value of 50 (while max thruster value is 255), if $\dot{\theta}$ is below a certain threshold, $\frac{\pi}{6}$ in these experiments, in addition to the control scheme described in the Controller Design section. This was added to prevent over-correcting, when it was believed that the forward and backward movement of the robot was due to it always trying to correct the yaw even though $\theta_t$ was very close to $\theta_{des}$.

Therefore, the oscillations observed in Figures 6 and 7 always from 0 to around 0.3 radians was most likely due to the robot trying to go straight, but turning left instead due to the weak left thruster, increasing $\dot{\theta}$ until it is above $\frac{\pi}{6}$. The controller then attempts to correct itself by turning right until $\dot{\theta} \approx 0$, from which the robot attempts to go straight again, creating a cycle, which can be seen in both horizontal control graphs. One way to possibly solve this without tampering with the hardware of the thruster is to multiply a constant to the speed sent to the left thruster, forcing it to be proportionally the same as the speed of the right thruster. This was attempted after this issue was discovered; however, due to time constraints, the exact constant could not be determined. Fixing the thruster is one of the highest priorities before this project can move forward.

# 8 Conclusion

The goal of this project was to create and implement a stable autonomous controller for an ROV to navigate around a cistern by following its outer walls. Although it could not be coupled with real time information about the closest wall, the control scheme described in this paper is proven to be stable both theoretically and in the experiments for depth controller. For horizontal control, I believe if the left thruster issue could be resolved, the $e_{\theta,t}$ and $e_{\rho,t}$ could be driven down to 0 as well. The next step would be to couple it with the wall finding algorithm developed by a fellow student, Anna Simpson, and test this method in real time. The gains would also need to be fine tuned, but this system has proven to be functional in a real environment and could provide a foundation for a more complex autonomous controller for an ROV to navigate a cistern for mapping purposes.

**Honor Pledge:** This paper represents my own work in accordance with university regulations.

# References


[1] Borenstein, J., Koren, Y. “Real-time obstacle avoidance for fast mobile robots in cluttered environments," *IEEE Transactions on Systems, Man, and Cybernetics,* 1990. **19**: 1179-1187.

[2] Carelli, R., Freire, E.O. “Corridor navigation and wall-following stable control for sonar-based mobile robots," *Robotics and Autonomous Systems,* 2003. **45** (3-4): 235-247.

[3] Cistern Exploration Project. http://users.csc.calpoly.edu/∼cmclark/MaltaMapping/index.html, 2012.

[4] Elfes, A. “A sonar-based mapping and navigation system," *IEEE International Conference on Robotics and Automation,* 1986. 1151-1156.

[5] Healey, A. “Obstacle Avoidance While Bottom Following for the REMUS Autonomous Underwater Vehicle," *Proceedings of the IFAC Conference, Lisbon, Portugal*, 2004.

[6] Siegwart, R., Nourbakhsh, I. *Introduction to Autonomous Mobile Robots,* 2002. pp. 85-88.

[7] Wang, W. “Autonomous of a Differential Thrust Micro ROV," Master’s thesis, University of Waterloo, 2006.

[8] White, C., Hiranandani, D., Olstad, C., Buhagiar, K., Gabmin, T., Clark, C.M. “The Malta Cistern Mapping Project: Underwater Robot Mapping and Localization within Ancient Tunnel Systems," *Journal of Field Robotics,* 2010.

# A Appendix

```
/* code for autonomous thruster control. Called from RunControlLoop */
void Robot::autonomousControl()
{
       //Depth Control, keeps depth constant
       char VERTICAL_THRUSTER = 2;
       unsigned char vertSpeed;
       bool vertDir;
       connections[0]->z_desired = 0.5; //in deci?meters
       //calculates values for vertical thrusters
       double vertical_d = connections[0]->Kvert*((connections[0]->z_desired +
           0.2) - state_est.z);
       int vertical_i = convert((int)vertical_d, &vertSpeed, &vertDir);
       //sets vertical thrusters
       connections[0]->setThruster(VERTICAL_THRUSTER, vertSpeed, vertDir,
           vertical_i);


       //Yaw Control, keeps distance and angle to wall constant
       char STARBOARD_THRUSTER = 1;
       char PORT_THRUSTER = 0;
       double desiredDist = 0.75;
       double maxWallDist = 1.50;
       double desiredYaw;
       double K_theta = 20.0;
       double K_dist = 2.0;;
       unsigned char yawSpeed;
       bool yawDir;
       double closestAngle = closestWallBearing;

       //for debugging, sets angle and dist to wall as desired constant value
       closestAngle = -Pi/2.0;
   closestWallDist = desiredDist;

       double desiredAngle = angleDiff(closestAngle, Pi/2.0); //compass angle
           for desired angle
       double errorAngle = angleDiff(desiredAngle,z_compass) - (K_dist *
           (desiredDist - closestWallDist)); //desired yaw relative to robot
       //if lost, turn to the right
       if (closestWallDist > maxWallDist) {
              errorAngle = -Pi/4;
       }

       double u_theta = K_theta*errorAngle;

       double yaw_d = -u_theta;
```

```
        int yaw_i = convert(yaw_d, &yawSpeed, &yawDir);

        yawSpeed = min(yawSpeed, 255.0);

        //time since start
        double t = (clock() - start) / (double) CLOCKS_PER_SEC;


        if (errorAngle <= (Pi / 6.0) && errorAngle >= (-Pi / 6.0)) {
               yawSpeed = 50.0;
               connections[0]->setThruster(PORT_THRUSTER, yawSpeed, yawDir, yaw_i);
        }

        else {
               connections[0]->setThruster(PORT_THRUSTER, yawSpeed, !yawDir,
                   yaw_i);
        }

        connections[0]->setThruster(STARBOARD_THRUSTER, yawSpeed, yawDir, yaw_i);

        //logging
        char values[300];
        sprintf(values, "t = %g, z_des = %g, state.z = %g, Dist_des = %g; Dist =
            %g, closestWallBearing = %g, compass = %g, errorAngle = %g, u_theta =
            %g, speed = %g, yaw_d = %g\n",
               t, connections[0]->z_desired, state_est.z, desiredDist,
                   closestWallDist, closestAngle, z_compass, errorAngle, u_theta,
                   (double) yawSpeed, yaw_d);
        OutputDebugString(values);
} // end autonomousControl
```