Roboboat - building unmanned surfaced vessels from RC motorboats

A. Calce, P. Mojiri Forooshani, A. Speers, K. Watters, T. Young and M. Jenkin

Computer Science and Engineering
York University
Toronto, Ontario, Canada
(calce, pmojirif, speers, kylew, tom, jenkin)@cse.yorku.ca

***Abstract*— A large number of manufacturers provide wheeled mobile robots of various capabilities and at various price points. Fewer commercial suppliers exist for autonomous surface craft, epsecially in the lower "experimental research" segment of the market. Here we describe the process of repurposing an inexpensive radio-controlled (RC) electric motorboat as an autonomous surface craft. Standard electronics components are used to interface with the RC boat electronics, and the vessel is augmented with GPS, vision, and a tilt-compensated compass to provide the necessary onboard sensing capabilities to enable point-to-point and target-based control of the vehicle. A ROS-based control and sensing infrastructure is used to operate the vehicle on-board while 802.11n communication provides communication off-board. The vessel has been operated successfully in both the pool and ocean environment.**

***Keywords*-autonomous surface craft**

I. Introduction

Unmanned surface vessels (USVs) have undergone a rapid growth over the last five years and we are now beginning to see devices in a range of different applications[1]. Autonomous Surface Craft (ASC), also referred to as Unmanned Surface Vessels (USVs), have been developed for civil, military and research purposes. Proposed applications for autonomous surface vessles for civil applications include automated bathymetry, coastal surveillance, water sampling, and environmental monitoring[2]. Of particular importance is the ability of an ASC to operate as a mobile relay point between radio frequency transmissions in air and acoustic transmission undersea. In this role they can be deployed as part of a cooperative heterogeneous team including AUVs (Autonomous Underwater Vessels). This fact is a key point in many military applications[1]. Other Military applications include harbour security, mine sweeping and various military missions[1], [2], [3].

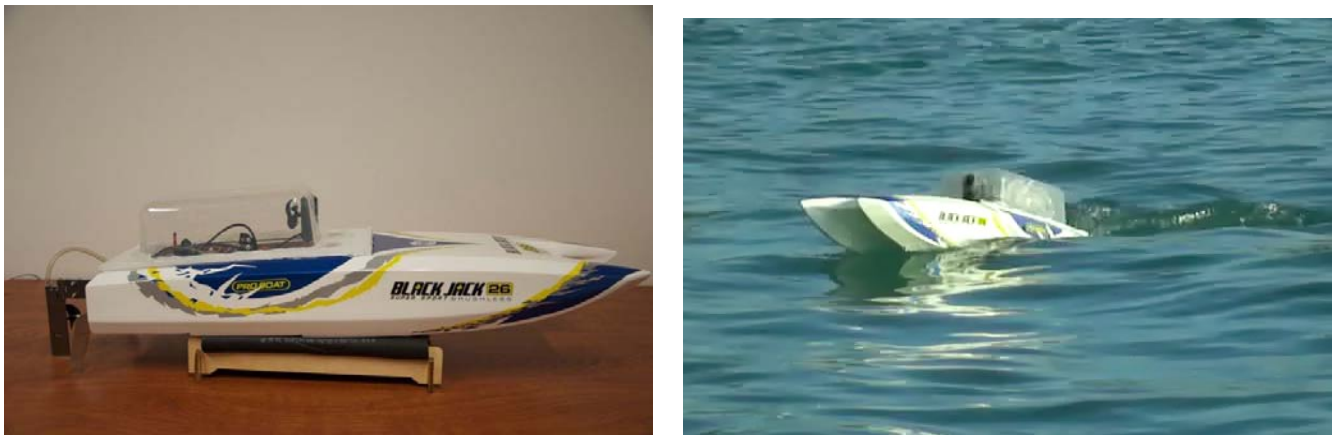
Over the last decade a number of different autonomous surface craft (ASC) prototypes have been developed. Allthough the set of potential applications for these devices is quite broad, three primary application domains have emerged: autonomous vehicle research, oceanography, and military. Sample platforms developed in these areas include:

Research platforms: The ARTEMIS platform was developed at MIT in 1993. This was a fish trawler-like vehicle[2]. The Caravela was developed by the DROS lab of Lisbon IST-ISR for the enhancement of acoustic communication with a companion autonomous underwater vehicle (AUV)[4]. The MIT autonomous kayak SCOUT (Surface Craft for Oceanographic and Undersea Testing) was built for developing and testing of distributed acoustic navigation algorithms for undersea vehicles[5]. There is also a large and active research community exploring the development of autonomous sailcraft (see [6], [7], [8] and [9] for examples). Many of these vessels have been developed around one of three major autonomous sailboat competitions.

Oceanography platforms: The Measuring Dolphin was developed by the University of Rostock for high accuracy positioning and water monitoring[2] while the autonomous catamaran Charlie was designed by Consiglio Nazionale delle Ricerche-Instituto di Studi sui Sistemi Intellienti per LAutomazione (CNR-ISSIA, Genova, Italy) for sampling the sea surface micro-layer in Antarctica [10]. The Delfim and the Springer were developed by the University of Plymouth for tracking of water pollution[2]. The vessels *ACES*[11] (Autonomous Coastal Exploration system) and *AutoCat* [12] were developed at MIT and used for the collection of hydrographic and bathymetric data.

Military platforms: A large number of USVs have been developed for military purposes including the QinetiQ Ltd SWIMS systems, and the Israeli Protector USV [2].

Although a number of different manufacturers of ASC's exist, very few are targeted at the lower end of the research/experimental market. Given the lack of such vehicles an attractive alternative approach, especially for research applications, is to repurpose watercraft as autonomous devices. Both vessels originally intended for human operation as well as radio-controlled (RC) vehicles have been repurposed. Although a number of such projects exist, much of the technical details associated with these projects is not well recorded and individual projects must often start again from scratch. Here we describe efforts to utilize standard tools from the RC and hobby electronics community to construct a fleet of autonomous surface vessels that expose a standard robot middleware (ROS[13]) to external users. The basic structures used in this project can be easily adapted to other applications and the basic electronic comonpents used are "standard" devices that are easily sourced.



(a) Side view (b) Operating in the open ocean

Figure 1. The "minnow". The autonomous service vessel is constructed from a modified RC motorboat, and has been augmented with onboard electronics and sensors. An 802.11g network connection provides offboard communication. To date, a fleet of two minnow's have been constructed with additional units underway.

II. Vessel design

A basic design question in the development of any autonomous system is the size of the device. Size, measured in volume or mass, provides a range of constraints related to power, locomotive strategies, onboard sensor capabilities, onboard computation, communication opportunities and the like. This constraint is especially true in the development of an autonomous surface vessel based on an RC platform. RC vessels are typically small, designed to be easily portable and with relatively short operational periods (typically under an hour, and often significantly less). On the plus side, RC vessels typically avoid regulatory issues associated with operating a "human sized" vessel autonomously. One critical question in terms of the repurposing of RC vessels is the nature of the vessel power plant. RC watercraft are developed with electrical systems for onboard control and communication, and are available with electric, gasoline and nitrox drive systems. Each of these drive alternatives come with their own capailities and constraints. Electrical powered systems are emission free and thus are easily deployed in indoor swimming pools, but they lack the extended range and maximum power output associated with gasoline and nitrox power plants. (A properly tuned nitrox boat can exceed 80 kilometers an hour in terms of speed.) Fortunately the basic control mechanisms remain unchanged over the various power plants, but the choice of plant type has a significant impact on maximum speed, available test environments and operational period.

For this project the "Blackjack 26" platform was chosen. The Blackjack 26 is a "ready to run" 26" fibreglass catamaran RC vessel powered by a single propeller and equipped with a single rudder (see Figure 1). The vessel is powered by a 1500 RPM brushless motor Brushless DC motor and utilizes a standard RC servo motor to provide positional control over the rudder. The boat comes equipped with a water-cooled 45A programmable Electronic Speed Controler (ESC) that provides control over the drive motor, power to the servomotor, and exposes itself to control circuitry as a standard servomotor. As shipped the vessel contins a pair of 7.2V sub-C battery packs that provide power for both the motors and the associated electronics. Normally the vessel is controlled by a remote radio controller, but this device was removed prior to repurposing the vehicle.

The physical structure of the vessels have been modified only in a very minor way through the addition of a (mostly) water-tight housing for the added electronics and sensors.

The robot operating system (ROS) has emerged as the defacto standard in terms of middleware for research robots. Rather than attempting to develop a custom software environment for the vehicle as much as possible of the software developed for the vehicle leverages existing ROS infrastructure. This choice enabled the project to leverage a significant open source code base but lead to a number of design decisions in terms of hardware and software infrastructure, as detailed later in this paper.

A. Electromechanical issues

A surface vessel like the Blackjack 26 is essentially a two degree-of-freedom (DOF) device. There is a rudder and throttle, both of which are controlled by 5V signal lines. The necessary control signal to these two inputs is provided by a radio receiver in the stock RC boat. Perhaps the first step in repurposing the vessel as an ASC is providing computer control of these two degrees of freedom. We chose to use an Arduino microcontroller[14] to provide this interface. The Arduino (μC) was chosen for a number of reasons related to ease of use, ability to provide the necessary control interface to the underlying hardware, and also the existence of an

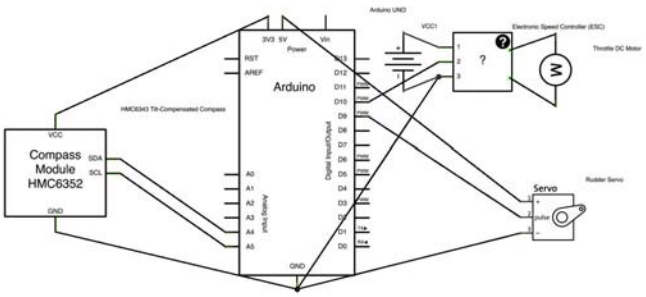


Figure 2. Low-level wiring

```
/*
* Set the rudder. This is a floating point value between -1 (hard a port) to
* +1 (hard a starbord). A value of 0 corresponds to straight ahead
*/
void rudderCb(const std_msgs::Float32 &new_rudder)
{
 int pos = 0;
 if((new_rudder.data < -1) || (new_rudder.data > 1)) {
   boat_msg.status = BOAT_STATUS_SHE_CANNA_TAKE_IT;
   pos = MID_RUDDER;
   boat_msg.rudder = 0.0;
 } else {
   if(new_rudder.data > 0)
     pos = (int)((MAX_RUDDER - MID_RUDDER) * new_rudder.data + MID_RUDDER);
   else
     pos = (int)((MID_RUDDER - MIN_RUDDER) * new_rudder.data + MID_RUDDER);
   boat_msg.rudder = new_rudder.data;
   boat_msg.status = BOAT_STATUS_AYE_AYE_CAPTAIN;
 }
 servoRud.write(pos);
}
ros::Subscriber <std_msgs::Float32> subRudder("setRudder", rudderCb);
```

Figure 3. Setting the rudder. `setRudder` is a ROS message that takes a floating point message in the range -1..+1 and sets the rudder from hard-a-port to hard-a-starboard.

Arduino ROS package[15] that allows ROS nodes to be executed on the Arduino directly and to be made visible to standard ROS nodes operating elsewhere.

The low-level electrical control subsystem is sketched in Figure 2. The Arduino microcontroller provides an interface to a tilt-compensated compass and provides reference voltages and PWM control signals to the rudder servo and the ESC controlling the throttle. The Arduino also provides power to the rudder servo. Normal power from the drive system provides power to the ESC. The Arduino is connected via USB to an onboard BeagleBoard SBC which provides higher-level control as well as providing power to the Arduino UWO.

Each of the motor controllers onboard the robot requires a reference ground and a 5V control line. These are easily mapped to the Arduino's digital out and reference lines as shown in Figure 2. The basic software infrastructure to provide a ROS message that sets the rudder orientation is shown in Figure 3 (the code for the throttle is similar). The `rosserial_arduino` package allows ROS nodes to operate on the Arduino with direct access to the Arduino control and data lines. This provides a very clean interface between low level vehicle control and the ROS environment.

In addition to subscribing to a `setRudder` and `setThrottle` message, the minnow also publishes a `vesselStatus` message which provides low-level information from the Arduino controller. Details of this are provided below.

Although the use of the Rosserial package greatly simplifies integrating low-level control of an actuator into a collection of ROS nodes, it is not without its limitations. The existing version of the library limits the number of publishers and subscribers on the Arduino. Nor are all primitive types supported in messages that can be passed. Furthermore, the limited bandwidth between the Arduino and its host computer is via a USB connection, and this also places limits on how much, and how quickly, data can be passed from code running on the Arduino and that running on the host device. (For example, sending long string messages is not to be encouraged given the limited 280 byte buffer size available.)

B. Computation

The limited volume and mass characteristics of the Blackjack 26 places extreme constraints in terms of potential onboard computation. Furthermore, the decision to build a ROS-compatable device requires that the onboard computer run some version of Linux. Although a number of small form factor devices meet all of these requirements, the decision was made to proceed with a BeagleBoard MX platform[16]. The benefits of using a BeagleBoard have been demonstrated against several other platforms small form factor boards (see [17]). The BeagleBoard runs a quasi-standard version of Linux, provides a reasonable number of input and output ports – including onboard video – and is reasonably inexpensive. Furthermore, it supports a number of different power inputs, including being able to be powered by a USB power source.

The BeagleBoard was configured to run Ubuntu Linux and Robot Operating System (ROS) [13]. When on, the BeagleBoard serves as a link between the base station computer (when instructions are sent to, or information is requested from the robot) and the low level systems of the robot through an 802.11g connection. The control of the robot's single servo, electronic speed controller, and 3D HMC6343 magnetometer compass [18] is delegated to the Arduino Uno [14] which is connected to the BeagleBoard via a USB connection. Connected directly to the BeagleBoard are the on-board camera and GPS units. Any commands to alter the motion of the robot are either generated on the Beagleboard or received by the BeagleBoard and directed to the Arduino Uno. The camera selected for use on the robot was a Logitech quick cam pro for notebooks. This camera provides images up to 2.0Mpixel at 5fps and is UVC compliant (allowing for simple interfacing within the Ubuntu environment). The GPS selected was the BU-353 USB GPS Navigation Receiver that connects to the BeagleBoard

through an RS232 connection which propagates the GPS data in NMEA format to the BeagleBoard.

The process of configuring a BeagleBoard to run ROS was not without its difficulties. The operating system most fully supported on the BeagleBoard family of single board computers is Angstrom, and initial development relied on this OS. ROS is not officially supported on the Angstrom OS, and efforts to compile ROS for the Angstrom-BeagleBoard combination did not lead to a robust ROS environment on the platform[1].

In order to provide a supported OS for ROS, a stripped-down version of Ubuntu ARM was deployed on the BeagleBoard. This provided a rather straight-forward installation of the majority of the ROS environment and its associated support libraries. ROS Electric Base and Ubuntu 11.04 (Natty Narwhal) were used as these were the most "cutting edge" compatible versions for the BeagelBoard platform at the time. The lack of prebuilt binaries for the ROS for this platform necessitated compiling ROS from scratch, a time-intensive task on the low-power BeagleBoard.

Initially the vehicle was powered using a Beaglejuice, a rechargeable battery pack with the same form factor as the BeagleBoard. Although this device was tested successfully under laboratory conditions, a Kensington 5V USB battery pack was used in recent ocean trials.

C. Sensing

Given the experimental nature of the vessel being constructed a wide range of different sensors have been built into the vehicle. The onboard sensor suite consists of

- **GPS** A ROS GPS node interfaces with the *BU-353 USB GPS Navigation Receiver*, parsing the data stream (in NMEA 0813 V2.2 format). Currently the node parses only the GPS system fix data (*$GPGGA*). These strings contain information on the latitude, longitude and altitude as well as associated data on fix quality and time of fix. The parsed latitude / longitude data is converted into a $+/-$ degree with positive values going North (Latitdue) and East (Longitude). The latitude and longitude are encapsulated in a custom gps message and are available to the rest of the system. If the GPS fix quality is zero a default error latitude / longitude value are returned.
- **Front facing video cameras** A USB QuickCam Logitech webcam with 640x480 resoltuion is mounted onboard the vessel (see Figure 1). This video source can be used both by a remote user to teleoperate the vehicle, as well as by ROS software to control the robot using visual landmarks. Given the Linux nature of the onboard computation and the use of ROS, the process of capturing video onboard the robot is accomplished using the standard Video4Linux library and images are transferred as a single ROS sensor_msgs::Image message. Other nodes, including nodes off-board the vehicle can subscribe to this published image stream. Figure 4 shows a frame from the image stream broadcast from the robot as it operates in the open ocean near Holetown, Barbados.
- **Tilt-compensated compass** Although considerably more expensive than their non-tilt-compensated counterpart, a tilt-compensated compass is essential on a surface vessel where wave action will ensure that the vehicle is rarely horizontal. Few tilt-compensated compasses exist with a USB interface, and thus it was necessary to decode the signal from the device in order to capture the vessel pitch, roll and yaw. Given the existence of the Arduino onboard the vessel to control the actuators, the Arduino was purposed to monitor the compass as well. The minnow publishes a `vesselStatus` message that contains the current commanded position of the rudder and throttle along with its pitch, roll and yaw. This custom message is defined on the Arduino and its definition is propogated to the rest of the ROS network through the Rosserial node definition.

In addition to these sensors, the onboard computer was equipped with an 802.11g wireless card to enable communication with an off-board controller.

Figure 4. Streamed digital camera footage from the minnow.

D. Robot control

Controlling software for the minnow robot is provided in ROS. The overall structure of the ROS environment is shown in Figure 5. Table I summarizes the various nodes that control the robot. Note that all of the nodes except `boatctl` are executed onboard in the BeagleBoard within the robot.

Most of the nodes provide simple encapsulation of one

[1]Compiling ROS on the Beagleboard itself is a timely process, and many of the Ubuntu packages upon which ROS is built are either not supported, or not supported as well, on the Angstrom OS.

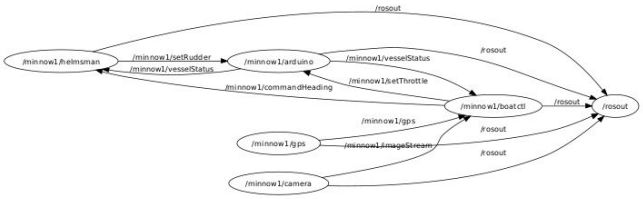


Figure 5. ROS control of the minnow. Here /minnow1 is driven by /boatctl.

of the basic sensor systems. For example, the `gps` node encapsulates the onboard GPS sensor and publishes the corresponding GPS signal. The `helmsman` node implements a standard PID controller to maintain a compass heading given a commanded heading (specified in in `commandHeading` message.

In order to ensure the widest possible interoperability of the ROS nodes across platforms, wherever possible the ROS nodes have been implemented in Python. This choice reduced development time substantively on the minnow robots as the time required to compile ROS code on the BeagleBoard is not insignificant.

III. Vehicle testing

The minnow USV's performed well in several pool trials as well as open water trials off the coast of Holetown, Barbados in January 2012. During these trials several things became apparent. First, the hull was able to take on the extra payload and run quite efficiently. At 20% throttle the boat was able to navigate fairly choppy waters and outpace human swimmers. A straightforward PID controller was constructed in ROS to provide bearing following. One issue that we expected to take considerable time during the field trials was tuning this PID controller for the open water conditions. Surprisingly, the tuning of the PID controller appears to be less critical in open water as opposed to a pool or pond setting as the rough surface acts to damp the system making changes in rudder position take time to accumulate into the overall vehicle trajectory. In all, the vehicle was successfully tele-operated visually and commanded to follow a compass heading during several trials both controlling the rudder position directly as well as controlling the compass heading through the PID "Helmsman" controller.

IV. Discussion and future work

During the trials the need for a heart-beat (still-alive) signal was made evident. The initial prototype would maintain it's last command until completion, received another command or until it ran out of power. This command structure would become problematic if the boat for any reason lost contact with the base station due to driving too far away from "home" or due to a wave blocking the signal. This latter problem was experienced during heavy swells where the minnow "lost sight" of the 802.11 base station. Modifying our ROS command structure to require a heart-beat in the command structure would allow for the boat to require a communication link to be present in order to maintain it's current operation. If it loses communications with the base station it would then return to some default action (i.e. set the throttle to zero).

Node	Description
arduino	encapsulates the rudder and throttle control as well as monitoring the compass
helmsman	provides bearing-following control
gps	encapculates onboard GPS sensor
camera	encapsualtes forward facing camera
boatctl	basic data logging and user interface

Table I
Basic ROS node functionality

On a more forward-looking theme the use of ROS as the core software architecture running the boat makes this project very expandable. We are looking forward to both leveraging the work of others on this platform (through the use of ROS nodes) as well as using multiple vehicles to perform a common task. In particular, ongoing work is exploring coordinated operation betwen a larger surface vehicle (a Clearpath King Fisher). This robot is equipped with a differential GPS system as well as a long range radio communications channel. This would allow the rest of the fleet (comprising of the cheaper minnows) to relay communications through the larger surface vehicle and perform longer range missions as well as benefit from more accurate positioning through the larger craft.

It is also anticipated that we will be performing joint actions with our AUV robot Kroy[19] and the Kingfisher platform. In these experiments we anticipate having one (or more) surface vehicles performing functions from the surface (perhaps finding places of interest for environmental modelling or surveying) with the large Kingfisher tracking visual targets on Kroy and commanding Kroy's actions.

Acknowledgment

The financial support of NSERC Canada is greatfully acknowledged.

References

[1] J. E. Manley, "Unmanned surface vehicles, 15 years of development," in *Oceans*, 2008, pp. 1–4.

[2] M. Caccia, "Autonomous surface craft: prototypes and basic research issues," in *Control and Automation*, 2006, pp. 1–6.

[3] V. Bertram, "Unmanned surface vehicles - a survey," Skibsteknisk Selskab, Copenhagen, Denmark, Tech. Rep., 2008.

[4] A. Pascoal, P. Oliveira, C. Silvestre, L. Sebastiao, M. Rufino, V. Barroso, J. Gomes, G. Ayela, P. Coince, M. Cardew, A. Ryan, H. Braithwaite, N. Cardew, J. Trepte, N. Seube, J. Champeau, P. Dhaussy, V. Sauce, R. Moitie, R. Santos, F. Cardigos, M. Brussieux, and P. Dando, "Robotic ocean vehicles for marine science applications: the european asimov project," in *OCEANS MTS/IEEE conference*, vol. 1, 2000, pp. 409–415.

[5] J. Curcio, J. Leonard, J. Vaganay, A. Patrikalakis, A. Bahr, D. Battle, H. Schmidt, and M. Grund, "Experiments in moving baseline navigation using autonomous surface craft," in *OCEANS NTS/IEEE conference*, vol. 1, 2005, pp. 730–735.

[6] N. A. Cruz and J. C. Alves, "Ocean sampling and surveillance using autonomous sailboats," in *The International Robotic Sailing Conference*, 2008, pp. 30–36.

[7] H. Erckens, G.-A. Busser, C. Pradalier, and R. Y. Siegwart, "Avalon: Navigation strategy and trajectory following controller for an autonomous sailing vessel," *IEEE Robotics and Automation Magazine*, p. 4554, 2010.

[8] J. Alves, T. Ramos, and N. Cruz, "A reconfigurable computing system for an autonomous sailboat," in *International Robotic Sailing Conference*, 2008, pp. 13–20.

[9] C. Sauze and M. Neal, "Design considerations for sailing robots performing long term autonomous oceanography," in *The International Robotic Sailing Conference*, Breitenbrunn, Austria, 2008, pp. 21–29.

[10] M. Caccia, "Laser-triangulation optical-correlation sensor for rov slow motion estimation," *IEEE Journal Oceanic Engineering*, vol. 31, pp. 711–727, 2006.

[11] J. E. Manley, "Development of the autonomous surface craft ACES," in *OCEANS MTS/IEEE conference*, vol. 2, 1997, pp. 827–832.

[12] J. E. Manley, A. Marsh, W. Cornforth, and C. Wiseman, "Evolution of the autonomous surface craft AutoCat," in *OCEANS MTS/IEEE conference*, vol. 1, 2000, pp. 403–408.

[13] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *Proc. of the Open-Source Software Workshop at the International Conference on Robotics and Automation (ICRA)*, 2009.

[14] http://www.arduino.cc.

[15] http://www.ros.org/news/2011/02/ros-meet-arduino.html.

[16] http://beagleboard.org/hardware-xM.

[17] K. Watters, C. Minwalla, M. Liscombe, H. Lio, P. Thomas, R. I. Hornsey, K. Ellis, and S. Jennings, "Characterization of an optical collision avoidance sensor," in *IEEE CCECE*, May 2011.

[18] http://www.honeywell.com.

[19] G. Dudek, P. Giguere, C. Prahacs, S. Saunderson, J. Sattar, L.-A. Torres-Mendez, M. Jenkin, A. German, A. Hogue, A. Ripsman, J. Zacher, E. Milios, H. Liu, and P. Zhang, "AQUA: An amphibious autonomous robot," *IEEE Comptuer*, pp. 48–53, 2007.