Fundamentals Of Robotics Final Project: Autonomous Robot Tugboat

Amy Phung

Project Team Members:
Everardo Gonzalez, Liz Leadley, Robert Wechsler

December 2018

1 Executive Summary

This project involved modifying a prebuilt tugboat and adding sensors, sensor mounts, Arduinos, and other hardware to upgrade the platform from being remote-controlled to a fully autonomous tugboat with radio communications to an off-board computer. To prove sufficient autonomous behavior, we programmed our boat to be able to do several low-level behaviors including wall following, obstacle circumnavigation, and autonomous docking and undocking. To further showcase the boat's autonomous capabilities, "missions" utilizing these low-level behaviors to create more complex behaviors were also added. Some missions we sought out to achieve included making figure 8s around two obstacles and target capture, which incorporates the boat's programmed low-level behaviors.

In order to accomplish these objectives, our team used a three Arduino setup with hardware serial communications between Arduinos, an XBee radio for communications between the Arduinos and the off-board computer, three sonars, six long-range sharp IRs, an IMU shield, and a Pixycam. In order to seamlessly integrate these additions to our boat, we also had to make some mechanical modifications including various sensor mounts. In the process of creating this tugboat, we also created various libraries that made it easier to interface with the boat. Each sensor had a library, each low-level behavior was included in a "Tugboat" object, and each high-level behavior was programmed in a separate "missions" library to distinguish low-level behaviors from high-level missions. We also added in a robust teleoperated control system to prevent the boat from being stranded in case of failures on any of the sensors.

Primary Contributions:

- System Design
- Software Design, Robot State Controller, Sensor Libraries

2 System Design

2.1 Mechanical Subsystem

Our mechanical subsystem design decisions primarily consisted of choosing the desired locations of the sensor mounts and in the design of mounts themselves.

Primary Mechanical Design Considerations

- Two IRs on side of boat for wall following and obstacle circumnavigation, put far apart to optimize wall following
- Pixycam facing forwards since we use it to dock and chase targets, both of which happen in front of the boat
- Two IRs facing forwards to determine how narrowly boat can turn
- Sensor mounts designed to be moved for flexibility in design changes

2.2 Electrical Subsystem

Our electrical subsystem originally consisted of two Arduinos communicating via the RX and TX pins, with one dedicated to processing sensor input and the other dedicated to driving the motors. However, we quickly found that this architecture added a tremendous amount of latency into our system, so we pivoted to a system that used a single Arduino Mega and a DFRobot Mega Sensor Shield V2.4 with an XBee to handle communications between the laptop and the tugboat. On top of the sensor shield, we also had an IMU (Arduino 9 Axis Motion Shield) and a Gravity I/O Expansion Shield v7.

2.3 Software Subsystem

Primary Software Design Considerations

- Libraries for modular code for each sensor
- Sensors object for storing all sensor data in a single object for ease of use and dot-indexing support. Clean list of pinouts is also available in header file
- Tugboat object for storing low-level behaviors and all data is updated at different rates, but tugboat itself is running at max clock speed
- Missions object for storing all mission code - interfaces with Tugboat for missions

3 Mechanical Modifications

3.1 Side IRs Placement and Mounting

Two IRs are mounted on the each side of the boat for effective wall following behavior. The IRs are mounted spaced out such that the difference between the two readings would be amplified in the case that the boat wasn't driving parallel and thus would make it easier to tell how much the boat needs to turn in order to achieve the desired wall following behavior.

When the mounts were first created, we were uncertain about what the final sensor layout for the boat would be, so we designed the mounts to be modular such that they could be mounted in almost any location along the boat.

3.2 Pixycam Placement and Mount Design

We chose to have the Pixycam facing forwards on the boat since its primary uses are in locating the markers for docking and chasing targets, both of which involve the boat driving head-on towards these markers.

Due to the fact that we were fairly confident in our initial Pixycam placement and that the Pixycam was an irreplaceable part of our boat (there were no backups in case of error), this mount was designed to be as robust as possible. The Pixycam is fastened to the boat via two L-brackets with screws that securely attach the Pixycam to the boat.

3.3 Front IRs Placement and Mount Design

We chose to have two IRs facing forwards to determine how narrowly the boat can turn to effectively navigate around obstacles. If either of front IRs detects an object, the boat cannot continue moving forwards without hitting the obstacle. If the boat turns until nothing is detected, then it will have a clear path forwards.

3.4 Sonars Placement and Mounting

We were fairly confident that we would be able to get the boat working with only the IRs and camera, but if we were to add sensors we would want to mount them such that the blind spots created by our layout are covered. Because of this, we decided to mount our sonars such that they covered these regions.

4 Electrical Design

Our electrical subsystem design changed significantly over the course of the project. At first, our setup involved three Arduino Unos with Hardware Serial communication to provide the necessary data flows, and our reasoning can be summarized as:

1. Three Arduinos because improved speed via a multi-threaded process, more memory, and e-stop and teleoperated capabilities in case one of the three processes fail
2. Pixycam was put on its own Arduino due to the slow vision processing clock cycle times and its known history for being unreliable
3. IRs and Sonars are on a separate Arduino. Since their measurements are based on voltage, we wanted to ensure that these sensors were not powered along the same power lines as the motors to minimize fluctuations in voltage. Since the sonars operate on sound pulses which travel at slow speeds when compared to light, there is a notable delay in data acquisition if obstacles are located at the limit of the sonar's detectable range. Due to this delay, acquiring sensor data operates on a time interval that is orders of magnitude larger than what is desired on our computation and decision-making Arduino.

4. XBee on "think" Arduino since we want our estop to be in the fastest timing loop possible

5. Originally, putting the IMU on the same Arduino as the IRs and Sonars made the most sense with our architecture, but we had to resort to putting the IMU on the think/act Arduino due to issues with data communication lines. The IMU used I2C communications which conflicted with the timing of our between-Arduino serial communications, and it used analog pins A4 and A5 which were already reserved for our IRs. Since the IMU data is known to be reliable with minimal impact on the loop timing, putting it on the think Arduino was our best choice given the conflicts.

However, after implementing this system, we ran into latency issues with data transfer time between Arduinos. The delay between sensing data and transferring it to the last Arduino added between 500-1000 ms of time, which outweighed any performance improvements we were aiming for. Our slowest sensors - the Pixycam and the sonars - operated on a clock speed of around 200-300 ms, which is a significant improvement over the delay from data transfer between Arduinos. Because of this, we switched to using only a single Arduino Mega with an expansion board to eliminate the transfer latency.

5 Software Design

Our team's control software involved various user-defined classes for optimal code readability and re-usability. For each of the sensors, we define a class that has the methods `init()`, `print()`, and `update()`. For sensors that require a user-defined pinout, this could be manually set by using the `pin` attribute, which is then used in `init()` to initialize the sensor. To extract the sensor data, most sensors update the `data` attribute in the `update()` function, though it differs a bit for sensors with multiple pieces of useful data.

Listing 1: IR class .h file with definitions for init(), print(), and update()

```
#ifndef IR_h
#define IR_h


#include "Arduino.h"

class IR
{
  public:
    IR();
    void init();
    void update();
    void print();

    int pin = 0; // Set default to 0,
    int x_offset = 0; // In cm
    int y_offset = 0;
    int heading = 0; // In degrees (0 is front, + is right, - is left)
    int data = 0;
```

```
  private:
    // define number of sensor readings to use for average filter
    int counts = 20;
    int readSensor();

    int raw_data = 0;

};

#endif
```

Listing 2: IR class .cpp file with definitions for init(), print(), and update()

```
#include "IR.h"

IR::IR()
{
}

void IR::init()
{
  pinMode(pin, INPUT);
}

void IR::print()
{
  Serial.println("IR Info:");
  Serial.print("IR data: "); Serial.println(data);

}

void IR::update()
{
  raw_data = readSensor();
  //raw_data = constrain(raw_data, 150, 400);
  data = 19807 * pow(raw_data, -1.0917); // In cm
}


// Private Functions:
int IR::readSensor()
{
  //average sensor reading
  int reading = 0;
  int cumReading = 0;
  int avgReading = 0;

  for (int i = 0; i < counts; i++) {
    reading = analogRead(pin);
    cumReading = reading + cumReading;
```

```
    delay(1);
  }

  avgReading = cumReading / counts;
  return avgReading;
};
```

To condense all the sensor data into one convenient object that could be passed between functions, we also defined a `Sensors` object. This also allowed us to put all of the sensor pinouts in a single place