Software Design Document

The SATIRE System

Robert Booth
Clayton Esposito
Taylor McRae
Sean Small

October 3, 2016
Revision 1

Contents

1 Introduction

1.1 Purpose

This software design document will be used to help provide documentation about how we plan to build, and develop our system controller for the underwater AUV. We will provide documentation regarding the class structure of our system, and how each of the components work together. Within this design documentation we will provide both a narrative explanation of our design, and a graphical explanation, shown by class diagrams and sequence diagrams.
The purpose of the SATIRE design document is to provide a description of the design for the underwater AUV, with enough information to allow our software development to continue with a solid understanding of what needs to be accomplished. This design document will provide necessary information about what each class will be doing within the system, and details about how the system will be built.

1.2 Scope

This design document is for a base level understanding of what our system will be trying to accomplish. We are tasked with creating a system that allows the underwater AUV to perform its basic functionality, and provide a template for further development on the system in future years. This system will be used in conjunction with a previously used underwater operating system (MOOS). There has been no previous code written for the system we are attempting to build, so our efforts will be placed under two areas: implementation of new code into the system, and making the system maintainable and open for future extension (considering that we are just making a basic system that will be used as a template for a more advanced future system).

1.3 Definitions, Acronyms, Abbreviations

SATIRE: Subsea Autonomous Traffic Identifying Recon Explorer

AUV: Autonomous Underwater Vehicle

MOOS: Mission Oriented Operating Suite

AO: Area of Operation

GPS: Global Positioning System

2 System Overview

2.1 Description of Problem

There is no current technology that is able to collect information in harbors across the world while being undetected by the human eye. The SATIRE System will build on the current MOOS platform to provide the client with an easy to use product that collects the desired data autonomously.

2.2 Technologies Used

The SATIRE System will communicate with input devices designed/compatible with the MOOS plattform. The devices include the Compass, the Sonar, the Humidity Sensor, the Temperature Sensor, the Hydrogen Sensor, the Leak Detector, the Battery Sensor, the Motor Controllers, and the GPS.
The target platform will be Linux, and the development environment is individual user preference. The client will be able to use a simplified GUI to prepare the AUV for launch.

2.3 System Architecture

ExternalSensorController():
The ExternalSensorController() will be in charge of monitoring all of the other external sensor classes. These external sensors are meant to collect all the data that is being processed outside of the tire, including the data to be collected by monitoring boats in the harbor, and the surroundings of the tire inside of the water. This controller will serve as an object that is created at the beginning of the task, and will not be destroyed until the very end of the mission, unlike some of it’s other subclasses.

- TireCompass(): The class is a subclass of the ExternalSensorController(). The class will be in charge of knowing which direction the tire is facing, and will be a very important feature in the navigation of the tire. Most of the tire’s navigation will be done out of reach from a satellite, so the tire will have to be able to navigate to a certain direction while not being able to communicate to any information outside of the water. This object will be created when the tire is told to either change it’s position, or when the tire must put information into it’s database in regards to the direction of the vehicles being monitored. To save memory, this object will be destroyed once it is no longer needed, and then re-created when it is given another task.

- TireSonar(): The class is a subclass of the ExternalSensorController(). The class will be in charge of analyzing the surroundings of the tire. The tire is going to need to navigate throughout random waters, with no previous information about its terrain, and be able to process that information in real time. This class will be very important while navigating through water because it will be our only way of preventing a collision between the tire and another object.

- TireVision(): The class is a subclass of the ExternalSensorController(). The class will be very similar to that of the TireSonar() class. Since the tire has no other means of sight besides sonar, then that is how the tire will navigate when unable to get any GPS data. This class is less concerned with avoiding objects, and is more concerned with using landmarks in the water to navigate to a given set of coordinates. This class will be the primary class used when the tire uses dead reckoning (navigating by use of a landmark).

- TireINS(): The class is a subclass of the ExternalSensorController(). The class will be responsible for interfacing with the Yaw, Pitch, Roll, and Ballast/Trim sensors. This data will be used along with GPS by the navigation and collision avoidance modules throughout operation.

InternalSensorController():
The ExternalSensorController() will be in charge of monitoring all of the internal sensor classes. These internal sensors are meant to collect all the data that is being processed

inside of the tire, including the data regarding the condition of the tire, and the ability to turn on the tire's motors. This controller will serve as an object that is created at the beginning of the task, and will not be destroyed until the very end of the mission, unlike some of it's other subclasses.

- TireHumidity(): The class is a subclass of the InternalSensorController(). The class will be in charge of monitoring the humidity levels of the tire. This class will make sure that the tire's humidity level does not get too high that it could compromise the integrity of the tire's sensors and other features. We will input a maximum humidity level into the system, and if the humidity level gets close to the max level, then this class will tell the tire to either turn off some of the sensors that could be affected, or move to a spot that would have lower humidity levels.

- TireTemp(): The class is a subclass of the InternalSensorController(). The class will be in charge of monitoring the temperature of the tire. This class will make sure that the tire's temperature does not get too high or too low that it could compromise the integrity of the tire's sensors and other features. We will input a maximum and minimum temperature into the system, and if the temperature gets close to the max or min level, then this class will tell the tire to either turn off some of the sensors that could be affected, or move to a spot that would have more serviceable temperatures.

- TireHydrogenDetector(): The class is a subclass of the InternalSensorController(). The class will be in charge of monitoring the Hydrogen levels of the tire. This class will make sure that the tire's Hydrogen levels does not get too high that it could compromise the integrity of the tire's sensors and other features. We will input a maximum Hydrogen level into the system, and if the hydrogen level gets close to the max level, then this class will tell the tire to either turn off some of the sensors that could be affected, or move to a spot that would have more serviceable hydrogen levels.

- TireAquaDetector(): The class is a subclass of the InternalSensorController(). The class will be in charge of monitoring if any water gets into the tire. Water coming into the tire can cause the sensors to all short circuit. So this class will shut down all sensors if any water comes in, and will send out a signal to an external source that the tire has been compromised.

- TirePowerMonitor(): The class is a subclass of the InternalSensorController(). The class will be in charge of monitoring the battery level of the tire. Once the battery of the tire starts to get to critical levels, then this class will send out a signal that the tire needs to be picked up because it cannot continue much longer.

- TireMotors(): The class is a subclass of the InternalSensorController(). The class will be in charge of turning on the motors to the tire so that that tire can move. Whenever the tire needs to move, or change its direction, then this object will be created, and then destroyed when the tire no longer needs to move.

CourseNavigation(): This class will be in charge of the navigation of the tire. It will use the objects from the ExternalSensorController(), along with other things, to help navigate the tire from its drop off location to its desired ending location. This class will have the tire use GPS location up until the tire is too far into the water the receive that data, then it will use its sensors from there to bring the tire to the end coordinates.

TireMovement(): This class is in charge of all the movements that the tire will be making throughout its mission. This class is pretty much an overseer of all the other objects that will be used because this class will have to use both the internal and external sensors. Whenever the TireMovement() class is called, it will create the TireMotor() object, and then follow the course navigation by moving the tire to that position.

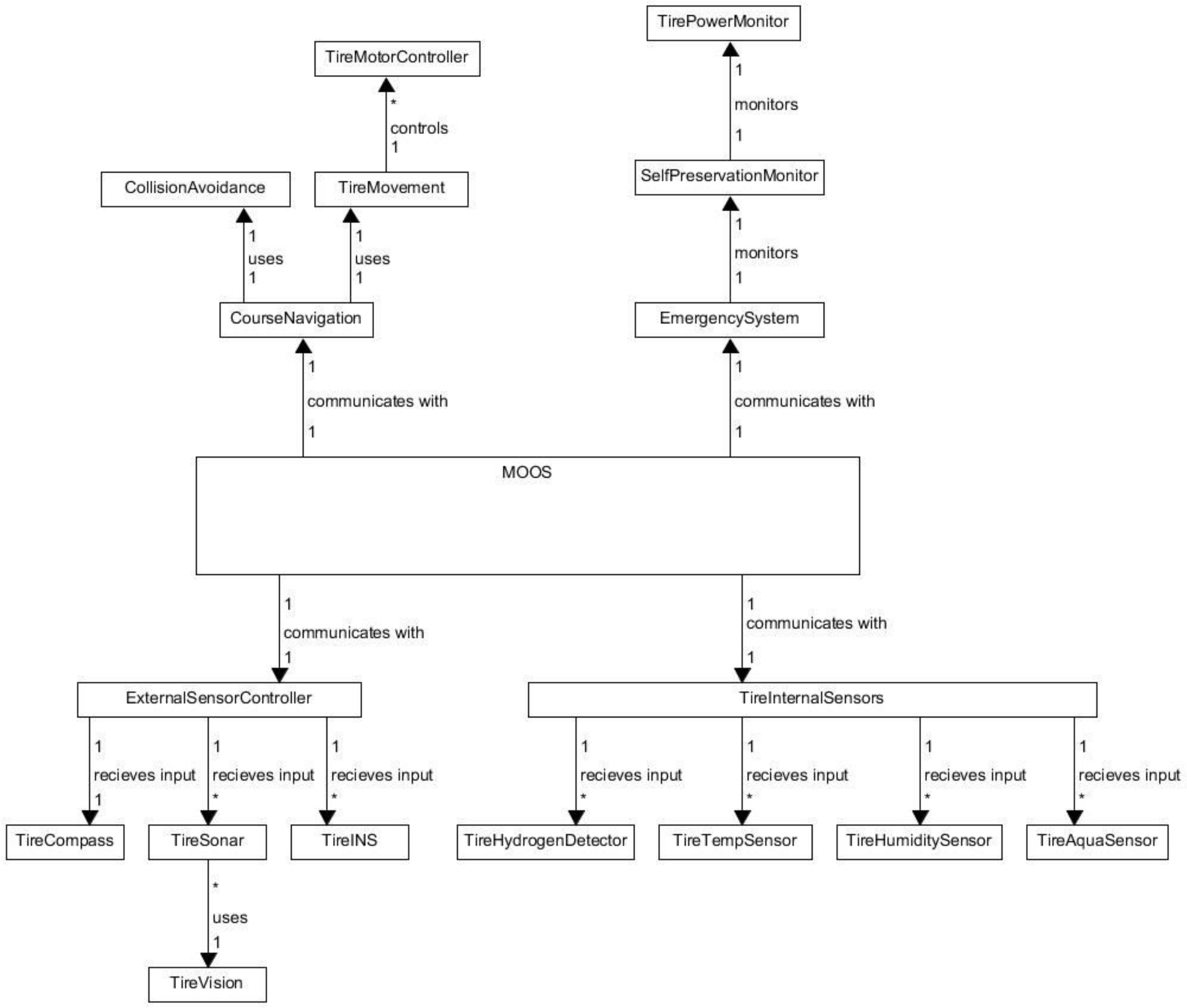
CollisionAvoidence(): This class is in charge of making the tire not crash into any outside objects when the tire is in the water. This class will use the tire's external sensors, primarily the TireSonar() class, to make sure the tire is within a certain distance of all other objects, and is not too close to the ground that it could crash, or not to high in the water that it could hit a potential boat.

EmergencySystem(): This class will be in charge of alerting the client that the tire is in critical condition, and needs to be picked up because it is compromised. This class will also cover the possibility of the tire being picked up by an unauthorized individual, and terminate all of the tire's data and system to make sure that nobody could know what the tire was doing in their body of water. This class is mostly meant to make sure the tire isn't destroyed, or compromised in any way.

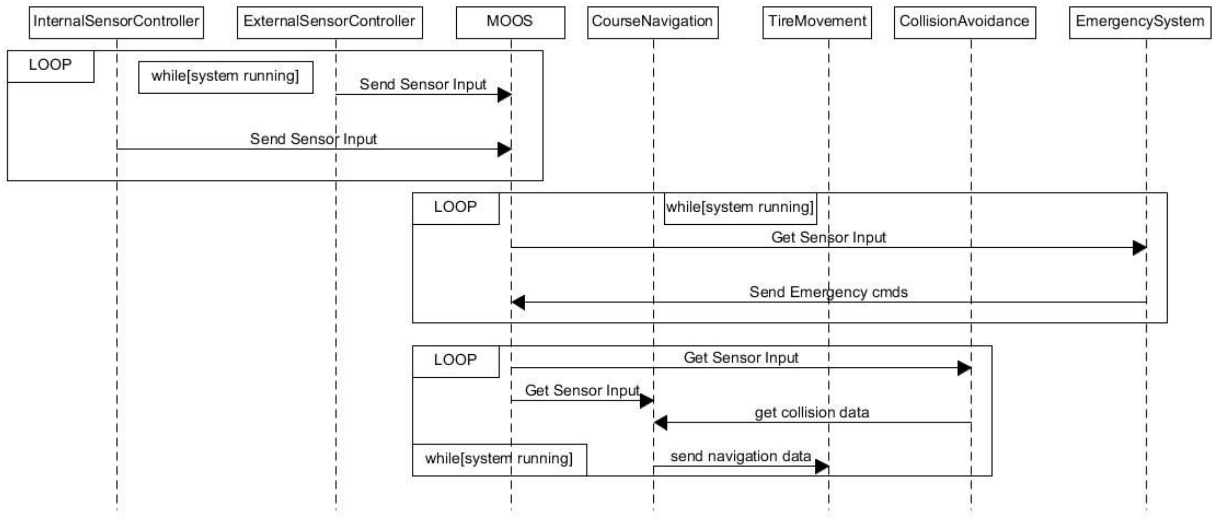
SelfPreservationMonitor(): This class will be in charge of making sure that the tire is not in certain conditions that could compromise the system. This class will use many of the internal sensor objects to make sure the tire doesn't break, and then use that information to make a decision on how to improve the state of the tire. This class will make sure the tire does not get destroyed, and that if anything does happen to compromise the system, then the system will be shut down, that way the data the tire accumulated could still potentially be accessed later.

3 Basic Flow

3.1 UML Diagram



3.2 System Sequence Diagram


InternalSensorController
ExternalSensorController
MOOS
CourseNavigation
TireMovement
CollisionAvoidance
EmergencySystem
LOOP
while[system running]
Send Sensor Input
Send Sensor Input
LOOP
while[system running]
Get Sensor Input
Send Emergency cmds
LOOP
Get Sensor Input
Get Sensor Input
get collision data
while[system running]
send navigation data