# Software Design Document

GROUP 11

John, Durham, Alex, Ian, Ethan

Version 1.00

9th March 2017

**Abstract**

This document will provide a detailed overview of the software requirements and design specifications that have been used in our robot. Functions are covered with provisions to different views of the system and to design decisions made within them.

# Contents

# 1 EDIT HISTORY

- March 4th 2017 (Version 1.0.0):
    - **John Wu:** Initial set up of document sections (including table of contents)
- March 6th 2017 (Version 1.0.0):
    - **John Wu:** Created introduction, added case diagram, added class diagram

# 2 INTRODUCTION

## 2.1 Purpose

Our vehicle is a fully autonomous robot that is capable of localization, navigation, obstacle avoidance, odometry, and retrieving a ball that can be shot into a target. In addition, the robot is capable of defending all the listed capabilities. The purpose of this document is to provide a detailed overview about the software implementation used in the robot. The functionality will be covered with provisions from different views including class diagrams, sequence diagrams, and overall system architecture.

## 2.2 Context

See requirements document included in the reference package.

## 2.3 Audience

This technical document is intended for parties who are interested in maintaining, testing, or extending the software that is included with the project. It is assumed that all readers have a sufficient understanding of programming, software design, and the LeJOS API.

## 2.4 Scope

The scope of our project will be limited to only achieving the list of requirements specified by the client. Therefore, no additional features will be added.

# 3    FUNCTIONAL REQUIREMENTS

## 3.1    Overview

Shown below is a use case diagram of our system. In order to initialize, the robot must be connected to retrieve parameters from the competition server. Based on the parameters given, the robot will either play offense or defense. Regardless of its position, the robot will go through it standard routines which includes localization, odometer, navigation, and obstacle avoidance. On offense, the robot will have additional cases for retrieving the ball and shooting it. On defense, the robot will rely on its standard routines to properly defend.

Figure 1: Use Case Diagram
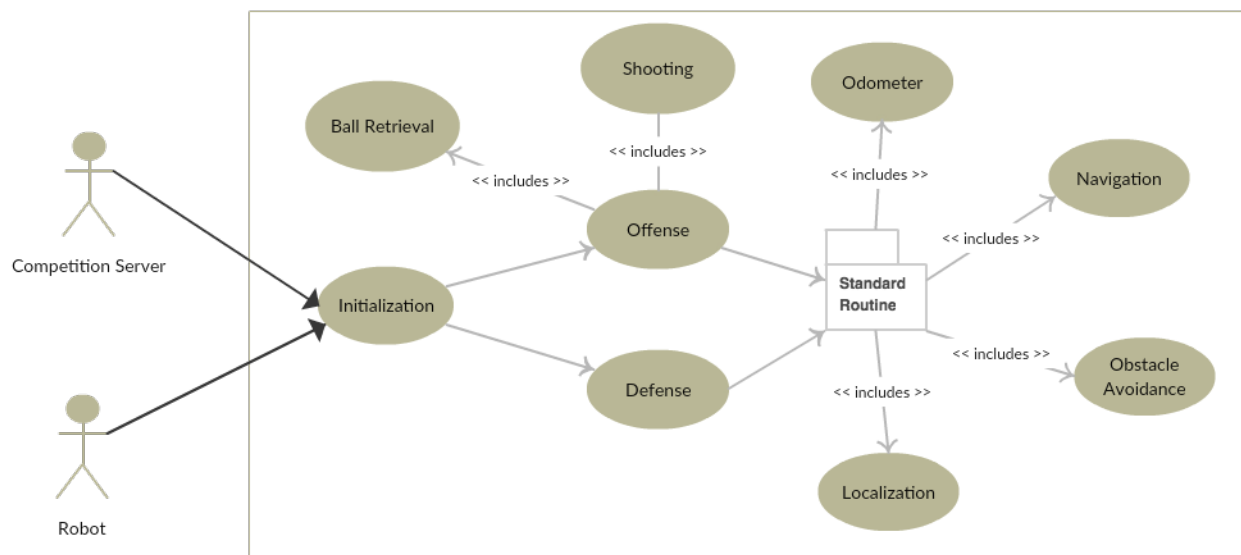
# 4   NON-FUNCTIONAL REQUIREMENTS

## 4.1   Battery Efficiency

## 4.2   Response Time

# 5   SYSTEM ARCHITECTURE

## 5.1   Overview

The following class diagram is a high level overview of the software architecture built into our system.

Figure 2: Class Diagram



**Parameters**
- forwardTeam : int
- defenseTeam : int
- forwardCorner : int
- defenseCorner : int
- forwardLine : int
- defenderZone : int[]
- ballDispenserPosition : int[]
- ballDispenserOrientation : String

+ getForwardTeam() : int
+ setForwardTeam(forwardTeam:int)
+ getDefenseTeam() : int
+ setDefenseTeam(defenseTeam:int)
+ getForwardCorner() : int
+ setForwardCorner(forwardCorner:int)
+ getDefenseCorner() : int
+ setDefenseCorner(defenseTeam:int)
+ getForwardLine() : int
+ setForwardLine(forwardCorner:int)
+ getDefenderZone() : int[]
+ setDefenderZone(defenderZone:int[])
+ getBallDispenserPosition : int[]
+ setBallDispenserPosition(setBallDispenserPosition:int[])
+ getBallDispenserOrientation : int[]
+ setBallDispenserOrientation(setBallDispenserOrientation:int[])

**Robot**
- leftMotor : EV3LargeRegulatedMotor
- rightMotor : EV3LargeRegulatedMotor
- launchMotor : EV3LargeRegulatedMotor
- forwardSensor : SampleProvider
- leftSensor : SampleProvider
- rightSensor : SampleProvider
- leftLightSensor : SampleProvider
- rightLightSensor : SampleProvider
- textScreen : TextLCD
- parameters : Parameters

+ main()
+ retrieveStartingParameters()

**WifiConnection**
- debugPrint : boolean
- teamNumber : int
- serverIP : String

+ getData()

**ExitButtonChecker**
- debugPrint : boolean
- connection : Socket
+ run()

**WifiProperties**
- SERVER_IP : String
- TEAM_NUMBER : int
- ENABLE_DEBUG_WIFI_PRINT : boolean
- PORT_NUMBER : int
- TIMEOUT : int

**Navigator**
- leftMotor : EV3LargeRegulatedMotor
- rightMotor : EV3LargeRegulatedMotor
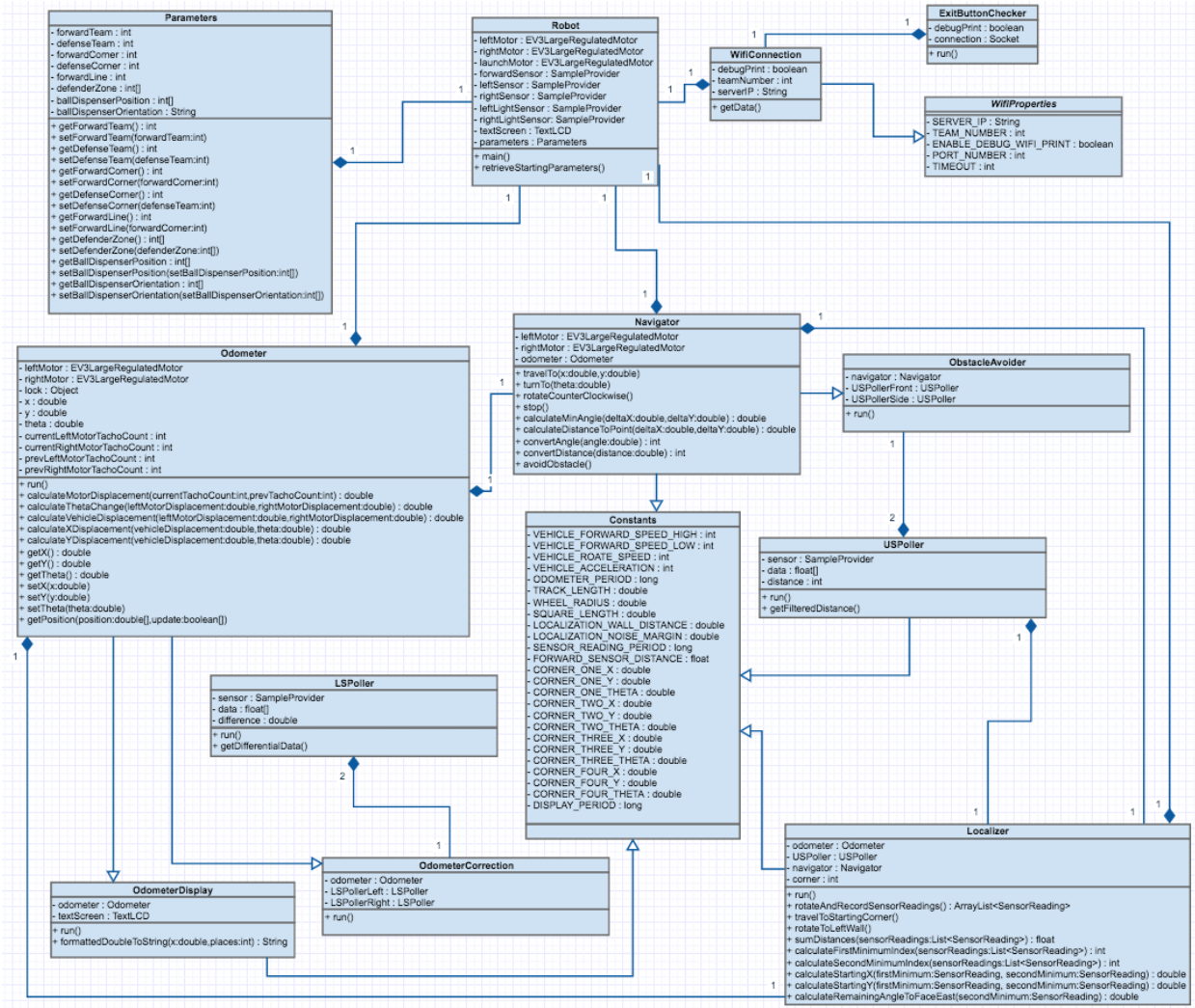- odometer : Odometer

+ travelTo(x:double,y:double)
+ turnTo(theta:double)
+ rotateCounterClockwise()
+ stop()
+ calculateMinAngle(deltaX:double,deltaY:double) : double
+ calculateDistanceToPoint(deltaX:double,deltaY:double) : double
+ convertAngle(angle:double) : int
+ convertDistance(distance:double) : int
+ avoidObstacle()

**ObstacleAvoider**
- navigator : Navigator
- USPollerFront : USPoller
- USPollerSide : USPoller
+ run()

**Odometer**
- leftMotor : EV3LargeRegulatedMotor
- rightMotor : EV3LargeRegulatedMotor
- lock : Object
- x : double
- y : double
- theta : double
- currentLeftMotorTachoCount : int
- currentRightMotorTachoCount : int
- prevLeftMotorTachoCount : int
- prevRightMotorTachoCount : int

+ run()
+ calculateMotorDisplacement(currentTachoCount:int,prevTachoCount:int) : double
+ calculateThetaChange(leftMotorDisplacement:double,rightMotorDisplacement:double) : double
+ calculateVehicleDisplacement(leftMotorDisplacement:double,rightMotorDisplacement:double) : double
+ calculateXDisplacement(vehicleDisplacement:double,theta:double) : double
+ calculateYDisplacement(vehicleDisplacement:double,theta:double) : double
+ getX() : double
+ getY() : double
+ getTheta() : double
+ setX(x:double)
+ setY(y:double)
+ setTheta(theta:double)
+ getPosition(position:double[],update:boolean[])

**Constants**
- VEHICLE_FORWARD_SPEED_HIGH : int
- VEHICLE_FORWARD_SPEED_LOW : int
- VEHICLE_ROATE_SPEED : int
- VEHICLE_ACCELERATION : int
- ODOMETER_PERIOD : long
- TRACK_LENGTH : double
- WHEEL_RADIUS : double
- SQUARE_LENGTH : double
- LOCALIZATION_WALL_DISTANCE : double
- LOCALIZATION_NOISE_MARGIN : double
- SENSOR_READING_PERIOD : long
- FORWARD_SENSOR_DISTANCE : float
- CORNER_ONE_X : double
- CORNER_ONE_Y : double
- CORNER_ONE_THETA : double
- CORNER_TWO_X : double
- CORNER_TWO_Y : double
- CORNER_TWO_THETA : double
- CORNER_THREE_X : double
- CORNER_THREE_Y : double
- CORNER_THREE_THETA : double
- CORNER_FOUR_X : double
- CORNER_FOUR_Y : double
- CORNER_FOUR_THETA : double
- DISPLAY_PERIOD : long

**USPoller**
- sensor : SampleProvider
- data : float[]
- distance : int
+ run()
+ getFilteredDistance()

**LSPoller**
- sensor : SampleProvider
- data : float[]
- difference : double
+ run()
+ getDifferentialData()

**OdometerCorrection**
- odometer : Odometer
- LSPollerLeft : LSPoller
- LSPollerRight : LSPoller
+ run()

**OdometerDisplay**
- odometer : Odometer
- textScreen : TextLCD
+ run()
+ formattedDoubleToString(x:double,places:int) : String

**Localizer**
- odometer : Odometer
- USPoller : USPoller
- navigator : Navigator
- corner : int
+ run()
+ rotateAndRecordSensorReadings() : ArrayList<SensorReading>
+ travelToStartingCorner()
+ rotateToLeftWall()
+ sumDistances(sensorReadings:List<SensorReading>) : float
+ calculateFirstMinimumIndex(sensorReadings:List<SensorReading>) : int
+ calculateSecondMinimumIndex(sensorReadings:List<SensorReading>) : int
+ calculateStartingX(firstMinimum:SensorReading, secondMinimum:SensorReading) : double
+ calculateStartingY(firstMinimum:SensorReading, secondMinimum:SensorReading) : double
+ calculateRemainingAngleToFaceEast(secondMinimum:SensorReading) : double

# 6 AUTOMATION

## 6.1 Unit Testing

## 6.2 Continuous Integration Build

# 7 REFERENCES

# 8 GLOSSARY OF TERMS